National and Kapodistrian University of Athens

School of Sciences

Department of Informatics and Telecommunications

PhD THESIS

Mining and Managing User-Generated Content and Preferences

Georgios Valkanas

Athens

August 2014

European Union
European Social Fund

OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS, CULTURE & SPORTS
MANAGING AUTHORITY

NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

Co-financed by Greece and the European Union

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Σχολή Θετικών Επιστημών

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Διδακτορική Διατριβή

Εξόρυξη και Διαχείριση Περιεχομένου Παραγόμενο από Χρήστες και Προτιμήσεων

Γεώργιος Βαλκανάς

Αθήνα

Αύγουστος 2014

European Union
European Social Fund

OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS, CULTURE & SPORTS
MANAGING AUTHORITY
Co-financed by Greece and the European Union

NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

PhD Dissertation Thesis

Mining and Managing User-Generated Content and Preferences

Georgios Valkanas

Examination Date: August 4, 2014

Διδακτορική Διατριβή

Εξόρυξη και Διαχείριση Περιεχομένου Παραγόμενο από Χρήστες και Προτιμήσεων

Γεώργιος Βαλκανάς

Ημερομηνία Εξέτασης: 4 Αυγούστου 2014

Abstract

The World Wide Web is evolving, as new technologies constantly emerge. Online services and applications are more pervasive nowadays, making the boundaries between the physical and online world less transparent. Users are able to share online aspects of their everyday life; most importantly they feel *comfortable* with doing so. This sharing takes place in various forms, ranging from simple button clicks (e.g., "Like", "+1") to structured and semi-structured data (e.g., filling in forms, selecting from pre-defined options), to totally unstructured information (e.g., natural language, online videos). New technologies have also shifted the roles of end users: they are no longer simple information consumers, but they actively participate in the content creation process, providing feedback and voicing their opinions and interests.

To make the most of this unprecedented abundance of information, and make next-generation services more engaging for the user, we need techniques that are more expressive in terms of the returned results, and allow us to better understand the data at hand. In other words, we need techniques to better *manage* and *mine* the available information, which is, for the most part, *generated by users*. Clearly, these techniques must be highly efficient to cope with the volumes available in the "Big Data" era. For these reasons, in this thesis, we present techniques to manage the results of expressive queries, such as skyline, and mine online content that has been generated by users. Given the numerous scenarios and applications where content mining can be applied, we focus, in particular, to two cases: *review mining* and *social media analysis*.

More specifically, we focus on *preference queries*, where users can query a set of items, each associated with an attribute set. For each of the attributes, users can specify their preference on whether to minimize or maximize it, e.g., "minimize price", "maximize performance", etc. Such queries are also know as "pareto optimal", or "skyline queries". A drawback of this query type is that the result may become too large for the user to inspect manually. We propose an approach that addresses this issue, by selecting a set of *diverse* skyline results. We provide a formal definition of *skyline diversification* and present effi-

cient techniques to return such a set of points. The result can then be ranked according to established quality criteria. We also propose an alternative scheme for ranking skyline results, following an information retrieval approach.
Preference queries, such as the aforementioned, can be used to retrieve items in a pre-usage scenario. User preferences, however, can be explicitly expressed in post-usage scenarios, through the writing of reviews or similar feedback. In particular, several platforms allow users to submit their feedback and opinions on a service (e.g., staying at a hotel) or a product they bought (e.g., a camera). Mining reviews has gained considerable attention over the years due to its direct financial impact. In our work, we used reviews as a means to extract how well a product covered the needs of users, and build a rigid, formal framework for *mining competitors*. We proposed algorithms to efficiently retrieve the top-k most competitive items, given an item of interest, and evaluated our approach with a user study, demonstrating the efficacy of our formalism.
Finally, social media are currently some of the most prolific platforms in terms of content generated by users. Characteristic examples include Twitter, Facebook, Tumblr, etc. Unlike earlier online platforms where users were able to upload content, such as blogs, these services have two distinct characteristics: i) fast pace and ii) huge numbers of active, connected users. Therefore, the volume of generated content reaches exceptional levels, and results in a *stream* of data. With users talking constantly about their interests and surroundings, we are given the opportunity to identify in real time interesting information, such as events. *Event detection* is of paramount importance in several cases, ranging from crisis management to resource allocation. We present techniques that aim to identify events, as they occur, and operate in an online fashion to cope with the streaming nature of incoming data. We resort to affective theories of emotions, which give our analysis a dual perspective, in terms of the psychological impact of events on users. This is especially important as new disciplines gain rise, namely *computational journalism* and *computational psychology*. In addition to detecting events, our analysis reveals some interesting properties regarding the underlying medium and the discussions that take place therein.

Subject area: Web Mining, Text Mining

Keywords: web mining, review mining, event detection, user preferences

Περίληψη

Ο Παγκόσμιος Ιστός εξελίσσεται, καθώς νέες τεχνολογίες προκύπτουν συνεχώς. Οι σημερινές διαδικτυακές υπηρεσίες και εφαρμογές είναι πιο διαδεδομένες, κάνοντας τα όρια μεταξύ του φυσικού και του δικτυακού κόσμου πιο ασαφή. Οι χρήστες έχουν τη δυνατότητα να μοιράζονται διαδικτυακά πτυχές της καθημερινής τους ζωής. Σημαντικότερο, μάλιστα, είναι το γεγονός ότι νιώθουν άνετα να κάνουν κάτι τέτοιο. Αυτός ο διαμοιρασμός πληροφορίας λαμβάνει χώρα υπό ποικίλες μορφές, που κυμαίνονται από απλά πατήματα κουμπιών (π.χ. "Like", "+1"), μέχρι δομημένα και ημιδομημένα δεδομένα (π.χ. συμπλήρωση φόρμας, προεπιλεγμένες επιλογές), αλλά και εντελώς αδόμητη πληροφορία (π.χ., φυσική γλώσσα, διαδικτυακά βίντεο). Αυτές οι νέες τεχνολογίες έχουν, επίσης, μετατοπίσει τους ρόλους των τελικών χρηστών: δεν είναι πλέον απλοί καταναλωτές πληροφορίας, αλλά συμμετέχουν ενεργά στην διαδικασία δημιουργίας περιεχομένου, προσφέροντας ανατροφοδότηση και εκφράζοντας τις απόψεις τους και τα ενδιαφέροντά τους.

Για την αξιοποίηση στο έπακρο αυτής της πληθώρας πληροφορίας, και προκειμένου οι υπηρεσίες της νέας γενιάς να κάνουν τους χρήστες να συμμετέχουν πιο ενεργά, χρειαζόμαστε τεχνικές που είναι πιο εκφραστικές ως προς τα αποτελέσματα που επιστρέφουν, και μας επιτρέπουν να κατανοούμε καλύτερα τα υπάρχοντα δεδομένα. Με άλλα λόγια, χρειαζόμαστε τεχνικές για να μπορούμε, με καλύτερο τρόπο, να διαχειριζόμαστε και να εξορύσουμε την διαθέσιμη πληροφορία, η οποία, στην πλειονότητα των περιπτώσεων, παράγεται από χρήστες. Φυσικά, αυτές οι τεχνικές απαιτείται να είναι ιδιαίτερα αποδοτικές, για να μπορούμε να ανταπεξέλθουμε στον όγκο των δεδομένων που είναι διαθέσιμα στην εποχή των "Μεγάλων Δεδομένων". Για τους προηγούμενους λόγους, στην παρούσα διδακτορική διατριβή, παρουσιάζουμε τεχνικές που διαχειρίζονται τα αποτελέσματα εκφραστικών ερωτημάτων, όπως τα ερωτήματα κορυφογραμμής, και εξορύσουμε περιεχόμενο που υπάρχει διαδικτυακά και παράγεται από τους χρήστες. Δεδομένων των πολυάριθμων σεναρίων

και εφαρμογών όπου μπορεί να εφαρμοστεί η εξόρυξη περιεχόμενου, επικεντρωνόμαστε σε δύο περιπτώσεις: εξόρυξη πληροφορίας από κριτικές και ανάλυση κοινωνικών μέσων.
Πιο συγκεκριμένα, εστιάζουμε σε ερωτήματα προτιμήσεων, όπου οι χρήστες μπορούν να κάνουν ερωτήματα πάνω σε ένα σύνολο αντικείμενων, όπου κάθε ένα συσχετίζεται με ένα σύνολο χαρακτηριστικών. Για κάθε ένα από τα χαρακτηριστικά, οι χρήστες μπορούν να προσδιορίσουν την προτίμησή τους, αναφορικά με την ελαχιστοποίησή ή την μεγιστοποίησή του (του χαρακτηριστικού), λ.χ., "ελαχιστοποίηση τιμής", "μεγιστοποίηση απόδοσης", κ.λπ. Τέτοια ερωτήματα είναι γνωστά ως "Pareto βέλτιστα" ερωτήματα ή "Ερωτήματα Κορυφογραμμής". Ένα μειονέκτημα αυτού του τύπου ερωτημάτων είναι ότι το αποτέλεσμα μπορεί να γίνει υπερβολικά μεγάλο, προκειμένου ο χρήστης να το επιθεωρήσει με μη αυτόματο τρόπο. Προτείνουμε μια προσέγγιση αντιμετώπισης αυτού του προβλήματος, επιλέγοντας ένα σύνολο διαφορετικών αντικειμένων που ανήκουν στο αποτέλεσμα ενός ερωτήματος κορυφογραμμής. Δίνουμε έναν τυπικό ορισμό του προβλήματος της διαφοροποίησης της κορυφογραμμής και παρουσιάζουμε αποδοτικές τεχνικές που επιλύουν αυτό το πρόβλημα. Το αποτέλεσμα μπορεί εκ των υστέρων να βαθμονομηθεί σύμφωνα με καταξιωμένα ποιοτικά κριτήρια. Προτείνουμε επίσης ένα εναλλακτικό σχήμα για την βαθμονόμηση των αποτελεσμάτων που προκύπτουν από ερωτήματα κορυφογραμμής, ακολουθώντας μια προσέγγιση που βασίζεται στην Ανάκτηση Πληροφορίας.
Τα ερωτήματα προτιμήσεων, όπως αυτά που περιγράφηκαν προηγουμένως, μπορούν να χρησιμοποιηθούν για την ανάκτηση αντικειμένων σε ένα σενάριο προ-χρήσης. Οι προτιμήσεις χρηστών, ωστόσο, μπορούν να εκφραστούν ρητά και σε περιπτώσεις κατόπιν της χρήσης ενός αντικείμενου, μέσω της συγγραφής κριτικών ή παρεμφερούς ανατροφοδότησης. Συγκεκριμένα, ποικίλες πλατφόρμες επιτρέπουν στους χρήστες να υποβάλλουν την ανατροφοδότησή τους και τις απόψεις τους αναφορικά με μία υπηρεσία (π.χ., τη διαμονή σε ένα ξενοδοχείο) ή ένα προϊόν που αγόρασαν (π.χ., μία φωτογραφική μηχανή). Η εξόρυξη δεδομένων από κριτικές έχει αποκτήσει σημαντικό ενδιαφέρον τα τελευταία

χρόνια, εξ αιτίας του άμεσου οικονομικού αντίκτυπου. Στην εργασία μας, χρησιμοποιήσαμε διαδικτυακές κριτικές για να εξάγουμε πόσο καλά ένα προϊόν ικανοποίησε τις ανάγκες χρηστών, και να χτίσουμε ένα τυπικό πλαίσιο για την εξόρυξη ανταγωνιστών. Προτείναμε αλγορίθμους για την αποδοτική ανάκτηση των k πιο ανταγωνιστικών προϊόντων, δοθέντος ενός αντικείμενου που μας ενδιαφέρει, και αποτιμήσαμε την προσέγγισή μας με μία μελέτη χρηστών, καταδεικνύοντας την αποτελεσματικότητα του φορμαλισμού που προτείνουμε.
Τέλος, τα κοινωνικά μέσα είναι, στην παρούσα φάση, από τις πιο γόνιμες πλατφόρμες, αναφορικά με το περιεχόμενο που παράγεται από τους χρήστες. Χαρακτηριστικά παραδείγματα περιλαμβάνουν το Twitter, το Facebook, το Tumblr, κ.λπ. Σε αντίθεση με προηγούμενες διαδικτυακές πλατφόρμες όπου οι χρήστες μπορούσαν να ανεβάζουν περιεχόμενο, όπως τα ιστολόγια (blogs), αυτές οι υπηρεσίες έχουν 2 διαφορετικά χαρακτηριστικά: 1) γρήγορο ρυθμό ανανέωσης και 2) τεράστιους αριθμούς ενεργών, διασυνδεδεμένων χρηστών. Συνεπώς, ο όγκος του παραγόμενου περιεχομένου φθάνει σε εξαιρετικά επίπεδα, και οδηγεί σε μια ροή δεδομένων. Δεδομένου ότι οι χρήστες συζητάνε διαρκώς για τα ενδιαφέροντά τους και τον περιβάλλοντα χώρο τους, μας δίνεται η δυνατότητα να εξάγουμε σε πραγματικό χρόνο ενδιαφέρουσες πληροφορίες, όπως γεγονότα. Η ανίχνευση γεγονότων είναι υψίστης σημασίας σε αρκετές περιπτώσεις, από την σωστή κατανομή των πόρων μέχρι και την διαχείριση κρίσεων. Προτείνουμε τεχνικές που στοχεύουν στην ανίχνευση γεγονότων, καθώς αυτά προκύπτουν, και λειτουργούν κατά , για να ανταπεξέλθουν στην συνεχόμενη ροή των εισερχόμενων δεδομένων. Για την επίτευξη αυτού του στόχου, καταφεύγουμε σε θεωρίες συναισθημάτων, που προσδίδουν στην ανάλυσή μας μια διττή προοπτική, αναφορικά με τον ψυχολογικό αντίκτυπο που έχουν τα γεγονότα στους χρήστες. Επιπροσθέτως, η ανάλυσή μας αποκαλύπτει ορισμένα πολύ ενδιαφέροντα χαρακτηριστικά για το μέσο που εξετάζουμε (Twitter) και τις συζητήσεις που λαμβάνουν χώρα σε αυτό.

Συνοπτική Παρουσίαση της Διδακτορικής Διατριβής

Ο Παγκόσμιος Ιστός έχει αλλάξει δραματικά με το πέρασμα των χρόνων σε σχέση με την αρχική του ιδέα, και εξακολουθεί να εξελίσσεται καθώς νέες τεχνολογίες προκύπτουν διαρκώς. Οι διαδικτυακές υπηρεσίες και εφαρμογές είναι σήμερα πιο διαδεδομένες, επιτρέποντας στους χρήστες να μοιράζονται στο διαδίκτυο πτυχές της καθημερινότητάς τους. Πιο σημαντικό είναι το γεγονός ότι οι χρήστες νιώθουν άνετα να κάνουν κάτι τέτοιο, το οποίο αποτελεί σημαντική μετατόπιση αναφορικά με την αντιμετώπιση της ιδιωτικότητας στα ψηφιακά περιβάλλοντα. Αυτή η γενική αλλαγή στη συμπεριφορά των χρηστών έχει κάνει τη διαχωριστική γραμμή μεταξύ του ψηφιακού και του φυσικού κόσμου περισσότερο ασαφή.

Αυτός ο διαμοιρασμός της πληροφορίας πραγματοποιείται υπό ποικίλες μορφές, που κυμαίνονται από απλά πατήματα κουμπιών (π.χ. "Like", "+1"), μέχρι δομημένα και ημιδομημένα δεδομένα (π.χ. συμπλήρωση φόρμας, προεπιλεγμένες επιλογές), αλλά και εντελώς αδόμητη πληροφορία (π.χ., φυσική γλώσσα, διαδικτυακά βίντεο). Σημειώνεται ότι αυτή η πληροφορία δεν παράγεται πλέον από μεγάλες εταιρίες ή ακαδημαϊκά ινστιτούτα, όπως γινόταν στις απαρχές του Παγκόσμιου Ιστού. Αντιθέτως, δημιουργείται από πραγματικούς χρήστες, των οποίων ο ρόλος έχει αλλάξει από αυτόν του παθητικού καταναλωτή πληροφορίας σε αυτόν του ενεργού συμμετέχοντα στην διαδικασία παραγωγής περιεχομένου. Είναι επίσης ενδιαφέρον ότι οι εταιρίες πλέον ενθαρρύνουν την ανατροφοδότηση από χρήστες - αν και προσπαθούν κατά καιρούς να υποσκάψουν τους σχετικούς μηχανισμούς του συστήματος - και οι χρήστες αναζητούν γνώμες από άλλους, προκειμένου να σχηματίσουν εμπεριστατωμένες γνώμες και να λάβουν τεκμηριωμένες αποφάσεις.

Ο κοινός παρανομαστής αυτών των περιπτώσεων είναι ότι οι χρήστες εκφράζουν τις προτιμήσεις τους και τις προσωπικές τους απόψεις πάνω σε μία σειρά θεμάτων, όπως μουσική, προϊόντα, πολιτική, κ.λπ. Αν και οι νέες τεχνολογίες προσφέρουν τα απαραίτητα πλαίσια ώστε οι χρήστες να μπορούν να εκφραστούν, απαι-

τούνται καινοτόμες ιδέες για να μετατρέψουν τα διαθέσιμα δεδομένα σε χρήσιμη και αξιοποιήσιμη πληροφορία. Η ανάγκη αυτή μεταφράζεται, επί της ουσίας, σε ενδιαφέρονται και δύσκολα ερευνητικά ερωτήματα τα οποία οφείλουμε να αντιμετωπίσουμε, ώστε να προσφέρουμε τις υπηρεσίες της επόμενης γενιάς. Για παράδειγμα, χρειαζόμαστε πιο εκφραστικούς τύπους ερωτημάτων, όπου οι προτιμήσεις των χρηστών θα λαμβάνονται σοβαρά υπόψη. Συγχρόνως, οφείλουμε να αναπτύξουμε τεχνικές οι οποίες εξάγουν ουσιαστικές και διορατικές πληροφορίες από το ογκώδες περιεχόμενο που παράγεται από τους χρήστες.

Τα ερωτήματα κορυφογραμμής και τα τοπ-k ερωτήματα είναι δύο χαρακτηριστικά παραδείγματα τέτοιων ερωτημάτων. Τα τοπ-k ερωτήματα επιστρέφουν τα k καλύτερα αντικείμενα σύμφωνα με μια συνάρτηση βαθμολόγησης $f()$ των αντικειμένων. Από την άλλη πλευρά, τα ερωτήματα κορυφογραμμής υποστηρίζουν πολυ-κριτηριακή βελτιστοποίηση και είναι σχεδιασμένα ώστε να επιστρέφουν αντικείμενα με διαφορετικές ισορροπίες. Συγκεκριμένα, το ερώτημα αυτό ορίζεται πάνω σε ένα σύνολο χαρακτηριστικών, και το αποτέλεσμα περιλαμβάνει εκείνα τα αντικείμενα για τα οποία δεν μπορούμε να βρούμε καλύτερη εναλλακτική ως προς όλα τα χαρακτηριστικά. Και οι δύο τύποι ερωτημάτων έχουν τα δικά τους πλεονεκτήματα και μειονεκτήματα. Για παράδειγμα, τα ερωτήματα τοπ-k ελέγχουν το μέγεθος του αποτελέσματος μέσω της παραμέτρου k. Ωστόσο, το αποτέλεσμα εξαρτάται πάρα πολύ από την συνάρτηση βαθμολόγησης $f()$ και η επιλογή μιας καλής τέτοιας συνάρτησης $f()$ δεν είναι καθόλου εύκολη διαδικασία. Από την άλλη, τα ερωτήματα κορυφογραμμής είναι ευκολότερα ως προς την κατανόησή της, δεδομένου ότι οι προτιμήσεις ορίζονται πάνω σε κάθε χαρακτηριστικό ξεχωριστά, όμως, το μέγεθος του αποτελέσματος μπορεί να γίνει εύκολα υπερβολικά μεγάλο. Σε αυτές τις περιπτώσεις θα ήταν πολύ κουραστικό για τον χρήστη να κοιτάξει όλα τα αποτελέσματα χειροκίνητα. Συνεπώς, χρειαζόμαστε τεχνικές που συνδυάζουν τα πλεονεκτήματα και των δύο ερωτημάτων. Η ενσωμάτωση επιπλέον περιορισμών κατά την επιλογή των αποτελεσμάτων, όπως π.χ. η διαφορετικότητα των αντικειμένων, θα ήταν ιδιαίτερα

επιθυμητή, ιδιαίτερα αν θεωρήσουμε το γεγονός ότι τα ερωτήματα κορυφογραμμής στοχεύουν να επιστρέφουν σημεία με διαφορετικές ισορροπίες.
Και οι δύο αυτοί τύποι ερωτημάτων καταγράφουν τα ερωτήματα των χρηστών σε σενάρια *προ-χρήσης*, με χαρακτηριστικές τις περιπτώσεις όπου οι χρήστες αναζητούν νέα αντικείμενα ή πληροφορία. Οι χρήστες, ωστόσο, μπορούν να εκφράσουν τις προτιμήσεις τους και σε *μετά-χρήσεως* περιπτώσεις, υπό την μορφή της ανατροφοδότησης (feedback). Σε ορισμένους τομείς, η ανατροφοδότηση μπορεί να δίνεται ρητά μέσω των ενεργειών των χρηστών. Για παράδειγμα, η επιλογή ενός αποτελέσματος στα πλαίσια μιας αναζήτησης στο διαδίκτυο (web search) είναι μια μορφή θετικής ανατροφοδότησης, ενώ η παράλειψη / υπερπήδηση αποτελεσμάτων αποτελεί αρνητική ανατροφοδότηση για τα αντικείμενα που δεν επιλέχθηκαν. Ομοίως, η επιλογή ενός καταλόγου όπου θα αποθηκευτεί ένας διαδικτυακός πόρος είναι μια ρητή μορφή ανατροφοδότησης για τη συγκεκριμένη εφαρμογή. Η υποστήριξη των ενεργειών των χρηστών μέσω της κριτικής αξιοποίησης τέτοιας ανατροφοδότησης θα μπορούσε να βελτιώσει σημαντικά των εμπειρία του χρήστη (user experience).
Η ανατροφοδότηση από τους χρήστες μπορεί να λάβει και άλλες μορφές, όπως για παράδειγμα ημι-δομημένη πληροφορία και ελεύθερο κείμενο. Οι διαδικτυακές κριτικές (online reviews) αποτελούν ένα χαρακτηριστικό παράδειγμα της τελευταίας μορφής, που έχει προσελκύσει ιδιαίτερο ενδιαφέρον τα τελευταία χρόνια. Αυτό το αυξημένο ενδιαφέρον οφείλεται στον αντίκτυπο που έχουν οι κριτικές στην εμπορευσιμότητα των προϊόντων. Μάλιστα, έρευνες έχουν δείξει ότι οι χρήστες προτιμούν προϊόντα για τα οποία υπάρχουν κριτικές, ώστε να γνωρίζουν τα θετικά και τα αρνητικά τους, και να μπορούν, επομένως, να λάβουν τεκμηριωμένες αποφάσεις. Μέσω του συνδυασμού της ανατροφοδότησης από τους χρήστες και των τεχνικών προδιαγραφών των προϊόντων, μπορούμε να εξάγουμε ένα στιβαρό πλαίσιο για την ανάλυση και τη σύγκριση τέτοιων προϊόντων. Πιο συγκεκριμένα, βασιζόμενοι στις ανάγκες των χρηστών - όπως αυτές προκύπτουν μέσω της ανατροφοδότησης που παρέχουν - και το βαθμό στον οποίο ένα προϊόν μπορεί να καλύψει συγκεκριμένες ανά-

γκες - όπως προκύπτει από τα τεχνικά χαρακτηριστικά του -, είμαστε σε θέση να αναγνωρίζουμε πόσο *ανταγωνιστικά* είναι τα προϊόντα. Αυτό είναι ιδιαίτερα χρήσιμο, τόσο για τους παραγωγούς των προϊόντων (δλδ., τις εταιρίες), όπως και τους καταναλωτές (δλδ, τους τελικούς χρήστες). Παρ' όλη τη σημασία του, μέχρι πρότινως δεν υπήρχε ένα τέτοιο τυπικό πλαίσιο, που θα αναγνωρίζει *ανταγωνιστικά* προϊόντα. Η πρόσφατη διαθεσιμότητα των διαδικτυακών κριτικών μας επέτρεψε να ελέγξουμε τόσο την αποδοτικότητα όσο και την αποτελεσματικότητα τεχνικών που επιστρέφουν τα k πιο ανταγωνιστικά προϊόντα, δοθέντος ενός αντικείμενου που μας ενδιαφέρει.

Παρά την έκρηξη του πλήθους των διαδικτυακών κριτικών, ο όγκος αυτός δεν συγκρίνεται επουδενί με τον όγκο των δεδομένων που παράγεται από τα μέσα κοινωνικής δικτύωσης (social media). Τα πιο δημοφιλή κοινωνικά μέσα έχουν υιοθετηθεί από ένα πραγματικά μεγάλο πλήθος χρηστών, με την εταιρία Facebook να καυχιέται παραπάνω από 1.28 δισεκατομμύρια ενεργούς χρήστες κάθε μήνα (στοιχεία μέχρι και 31 Μαρτίου, 2014), και την εταιρία Twitter - που ιδρύθηκε αργότερα - να έχει περισσότερους από 255 εκατομμύρια ενεργούς χρήστες το μήνα (μέχρι και τον Ιούλιο 2014). Μια κινητήρια δύναμη των τεχνολογικών αυτών πλαισίων είναι το συστατικό της δικτύωσης, με τους συμμετέχοντες να συνδέονται μεταξύ τους, ως προαπαιτούμενο για το διαμοιρασμό πληροφορίας.

Αναμφίβολα, τα κοινωνικά μέσα αποτελούν μια από τις πιο γόνιμες ερευνητικές περιοχές σήμερα, όχι μόνο λόγω της υψηλής αποδοχής από χρήστες, αλλά και λόγοω της χρησιμότητας των δεδομένων τους για ποικίλες επιστήμες και κλάδους: πληροφορική, ψυχολογία, κοινωνιολογία και δημοσιογραφία είναι μόνο μερικές χαρακτηριστικές περιπτώσεις. Επιπροσθέτως, υπάρχουν πρακτικές εφαρμογές όπου τα δεδομένα μπορούν να αξιοποιηθούν. Διαφημιστικές εκστρατείες (advertising) και η ανίχνευση κοινοτήτων (community detection) είναι τυπικά παραδείγματα, ενώ η αναγνώριση συμβάντων (σε πραγματικό χρόνο), η ανάλυση αλληλεπιδράσεων μεταξύ χρηστών (interaction analysis), και η κατανόηση της συμπεριφοράς των χρηστών (user behavior analysis)

λαμβάνουν ολοένα και περισσότερη προσοχή. Η αποκωδικοποίηση του περιεχομένου που παράγεται από τους χρήστες σε αυτά τα μέσα είναι ιδιαίτερα δύσκολη, λόγω του όγκου των δεδομένων και της ποικιλομορφίας του περιεχομένου, που οφείλεται στον αντίστοιχο πληθυσμό και τα ενδιαφέροντά τους.
Η εξόρυξη δεδομένων σε τόσο μεγάλο όγκο πληροφορίας για την ανεύρεση γεγονότων, που είναι άξια για δημοσιοποίηση, είναι κάθε άλλο παρά εύκολη. Προηγούμενες τεχνικές επικεντρώθηκαν στην παρακολούθηση γεγονότων, υπονοώντας ότι το γεγονός έχει ήδη ανιχνευθεί ή είναι με κάποιο τρόπο γνωστό. Άλλοι ερευνητές απλοποιούν το πρόβλημα με το να ψάχνουν για συγκεκριμένες λέξεις-κλειδιά, τα οποία μπορούν να περιγράψουν επαρκώς ένα γεγονός. Η ανίχνευση γεγονότων, ανεξαρτήτως τύπου (π.χ. πολιτική, αθλητικά, κλπ), και χωρίς πρότερη γνώση αναφορικά με λέξεις που μπορούν να το περιγράψουν , απαιτεί μια διαφορετική προσέγγιση. Ως εναλλακτική, καταφεύγουμε σε θεωρίες που στηρίζονται στην ψυχολογία, σύμφωνα με τις οποίες τα γεγονότα έχουν επιπτώσεις στον ανθρώπινο ψυχισμό (και πιο συγκεκριμένα στην συναισθηματική του κατάσταση), εξωθώντας τους να εξωτερικεύσουν τις σκέψεις τους. Υποστηρίζουμε ότι γεγονότα που είναι άξια προς δημοσιοποίηση θα έχουν επίπτωση σε μεγάλες ομάδες χρηστών, και μέσω της παρακολούθησης της συναισθηματικής κατάστασης της ομάδας, θα μπορούμε να εντοπίσουμε απότομες μεταβολές, τις οποίες μπορούμε να συσχετίσουμε με την πηγή, δλδ, το συμβάν.
Στα πλαίσια αυτού του ερευνητικού προβλήματος, ωστόσο, υπάρχει μια πληθώρα άλλων ερωτημάτων που πρέπει να απαντηθούν. Για παράδειγμα, πρέπει να αναγνωρίζουμε την τοποθεσία όπου έλαβε χώρα ένα γεγονός, το οποίο θα μπορούσε να μας βοηθήσει να το περιγράψουμε, ή ακόμα καλύτερα, να το εξηγήσουμε. Η εξαγωγή της συναισθηματικής κατάστασης είναι ένα δύσκολο πρόβλημα, ακόμα και αν εστιάζουμε σε ένα μεμονωμένο χρήστη, πόσο μάλλον σε μία ομάδα χρηστών. Η αναγνώριση απότομων μεταβολών απαιτεί μια προσεκτική μοντελοποίηση του προβλήματος, καθώς επίσης και αποδοτικές τεχνικές, λόγω του υψηλού όγκου δεδομένων, τα οποία πρέπει να επεξεργαστούμε σε πραγματικό χρόνο. Η παρουσίαση / οπτικοποίηση της πληροφορίας

με κατάλληλο τρόπο, ώστε να γίνεται εύκολα κατανοητή, είναι ένα άλλο ζήτημα που πρέπει να λάβουμε υπόψη.

Contents

1 Introduction **35**

1.1 The Web 2.0 era and beyond 35

1.2 Contributions and roadmap of this thesis 38

2 Managing user preferences **41**

2.1 Introduction 41

2.2 Skyline diversification 43

2.2.1 Motivation 43

2.2.2 Related Work 46

2.2.3 Problem Definition 50

2.2.4 Straight-Forward Techniques 53

2.2.5 The SkyDiver Framework 55

2.2.5.1 Phase 1: Fingerprinting with MinHashing 55

2.2.5.2 Phase 2: Selection of Diverse Skyline Points 59

2.2.5.3 The LSH-Based Method 65

2.2.6 Experimental Evaluation 67

2.2.6.1 Algorithms and Data Sets 67

2.2.6.2 Experiments and Results 68

2.3 Skyline ranking with IR techniques 77

2.3.1 Introduction 77

2.3.2 DP-IDP weighting scheme 79

2.3.2.1 Inverse Dominance Power 79

2.3.2.2 Dominance Power 80

2.3.2.3 Putting it all together 81

2.3.3 Ranking the Skyline 82

2.3.3.1 Bounding the score 83

2.3.3.2 The SkyIR technique 88

2.3.4 Performance Evaluation 90

2.4 Summary 94

3 Mining user preferences 97
3.1 Introduction 97
3.2 Review Mining for Competitor Identification 98
3.2.1 Motivation 98
3.2.2 Related Work 101
3.2.3 Formalizing Competitiveness 102
3.2.3.1 Competitiveness via Coverage 102
3.2.3.2 Computing Coverage 105
3.2.4 Finding the Top-K Competitors 108
3.2.5 The CMiner Algorithm 109
3.2.5.1 The CMiner Algorithm 112
3.2.5.2 Algorithmic Complexity 115
3.2.6 Weight-Estimation for Feature-Subsets 115
3.2.7 Subset Ordering 116
3.2.8 Experimental Evaluation 118
3.2.9 Computational Time 121
3.2.10 A User Study 124
3.3 Summary 126

4 Mining user-generated content 129
4.1 Introduction 129
4.1.1 Research questions in Online social networks . 132
4.2 Towards principled solutions for mining online content . 133
4.2.1 Faceted crawling of the Twitter service 133
4.2.1.1 Related Work 134
4.2.1.2 Twitter API Background 135
4.2.1.3 Faceted Crawler Architecture 136
4.2.1.4 Use Cases 140
4.2.2 Harvesting content from Hidden-web databases 143
4.2.2.1 Related Work 145
4.2.2.2 Background 146
4.2.2.3 Rank-Aware Crawling 147
4.2.2.4 Ranking Distances 150
4.2.2.5 Experimental Evaluation 151
4.2.3 Mining data using execution engines and declarative languages 155

4.2.3.1 Related Work 158
4.2.3.2 In-Network Data Analysis with a Declarative Language 160
4.2.3.3 Query Refactoring 163
4.2.3.4 Experimental Evaluation 169
4.3 Comparing sampling policies 175
4.3.1 Related Work 177
4.3.1.1 Spatio-temporal analysis of Twitter feeds 177
4.3.1.2 Sampling Social Data Streams 178
4.3.2 The Data 178
4.3.3 Geo-location Coverage 179
4.3.4 Sentiment Analysis 182
4.3.5 Popular Topic Detection 183
4.3.5.1 Top-most retweeted posts 183
4.3.5.2 Retweet Burstiness 186
4.3.6 Graph Evolution 188
4.3.6.1 Temporal Retweet Graph 188
4.3.7 Linguistic Analysis 190
4.3.8 Efficiency 191
4.4 Detecting Events with User-Generated Content 192
4.4.1 Related Work 196
4.4.2 Modeling & Detecting Events 198
4.4.2.1 Approximating the Emotional State Distribution 201
4.4.2.2 Event Detection 203
4.4.3 Extracting Spatial Information 204
4.4.3.1 Related Work 206
4.4.3.2 Problem definition 207
4.4.3.3 Problem setting 208
4.4.3.4 Online geocoding 209
4.4.3.5 Data Cleaning 209
4.4.3.6 Algorithmic description 210
4.4.4 The TwInsight System 212
4.4.5 Event Extraction Workflow 215
4.4.6 Experiments 217
4.4.6.1 Experimental Setup 217
4.4.6.2 Geocoding Results 218
4.4.6.3 Event Detection Results 224
4.4.7 Visualizing Results 234

4.4.7.1 Visualizing Emotions 234
4.4.7.2 Visualizing Spatial Information 236
4.4.7.3 Visualizing Event Description 237
4.4.7.4 Putting It All Together 240
4.5 Summary 242

5 Conclusions and future directions 245
5.1 Summary of the thesis 245
5.2 Future Directions 249
5.2.1 Skyline diversification and novel query types . . 249
5.2.2 Competitor identification and review mining . . . 250
5.2.3 Mining user-generated content and social media analysis 250

6 References 253

List of Figures

2.1 Example of the skyline on a set of hotels with 2 attributes 42
2.2 Graph with dominance relations. 44
2.3 Solutions to dispersion problems. 52
2.4 Domination of MBRs. 59
2.5 Buckets and bit-vectors of skyline points. 66
2.6 Time for generating MinHash signatures vs signature size. 68
2.7 Time for generating MinHash signatures of size 100 for synthetic data sets. 69
2.8 Runtime for k = 10 diverse skyline points vs dimensionality. 70
2.9 Runtime vs number of diverse points (k). 71
2.10 Quality vs number of diverse points (k). 72
2.11 LSH vs MinHashing, for k = 10 diverse skyline points, with signature size fixed to 100 73
2.12 Example for the Dominance Power 80
2.13 Example of skyline and bipartite domination graph . . . 84
2.14 Collaborative upper bound 86
2.15 Total runtime versus cardinality for IND, k=5 91
2.16 Total runtime for various prioritization with IND, CB . . 92
2.17 Total runtime for ANT distribution 92
2.18 Forest cover 93
2.19 Maximum memory consumption for CB, k=5 94

3.1 Simplified example of our competitiveness paradigm . . 100
3.2 Geometric interpretation of pairwise coverage 105
3.3 An example of the skyline pyramid structure 111
3.4 Cumulative distribution of items across the first 6 layers of the skyline pyramid. 119
3.5 Distribution of feature subset weights 120

3.6 Average time (per item) to compute top-k competitors for the various datasets 122
3.7 Average number of computed coverages to find the top-k competitors for the various datasets 123
3.8 Average (per item) #evaluated feature subsets, before sequentially computing competitiveness scores 124
3.9 Results of the user study comparing our competitiveness paradigm with the Nearest-Neighbor approach. . . 126

4.1 Architectural designs of both classic web crawler and our Twitter faceted crawler 137
4.2 The IRanker interface 138
4.3 Schematic representation of Crawl Flow examples. . . 140
4.4 Comparing raw counts between crawled and geocoded locational information 140
4.5 Crawling comparison for basic user information 141
4.6 Average crawler performance for harvesting (a) user information and (b)-(c) user timelines. 142
4.7 Collective performance of the crawlers used for harvesting user timelines. 142
4.8 Sampling scenario statistics. 143
4.9 Effectiveness in harvesting information from Hidden Web sites 153
4.10 Syntax for Defining an Intensional Extent. 161
4.11 Example schema of two streams expressed in SNEEql. 162
4.12 Creating a Linear Regression Classifier. 162
4.13 Using the TropForestLRF intensional extent. 162
4.14 Creating a D3 outlier detection extent. 163
4.15 Using the d3od intensional extent. 164
4.16 SNEE optimization stack: (left) original stack, (right) query refactoring approach for data analysis techniques. 164
4.17 Creation of a templated Linear Regression classifier. . 165
4.18 Templated subquery for computing (a, b) values 165
4.19 General form of a query using the *LRF* extent 166
4.20 General form of a refactored query using Linear Regression 167
4.21 Refactored Query of Fig. 4.13. 167
4.22 Subquery of (ab_COMP) in Fig. 4.21 168

4.23 Refactored query of D3 outlier detection algorithm. . . . 169
4.24 Example of Linear Regression computation for Naïve, LC and SNEE-A approaches. 171
4.25 Average number of sent messages compared to the average network length for LR 172
4.26 Average number of sent bytes compared to the average network length for LR 172
4.27 Transmission energy consumption compared to the average network length for LR 173
4.28 Reception energy consumption compared to the average network length for LR 173
4.29 CPU energy consumption compared to the average network length for LR 174
4.30 Radio-CPU contribution(%) to power consumption. . . 174
4.31 Average total energy consumption compared to average network length for D3 174
4.32 Comparing default and Gardenhose samples for volume over time 179
4.33 Bounding boxes of Table 4.8, Crawl1: Green, Crawl2: Orange, Crawl3: Red, Crawl4: Blue 181
4.34 Different crawls from London 181
4.35 Comparing tweets with sentiment 183
4.36 Comparing the top-10000 most retweeted items 184
4.37 Burstiness of retweeting information 186
4.38 Statistical properties of the extracted retweet graph, over time 189
4.39 Tweets reflecting real-life events. 194
4.40 A timeseries on the daily emotions identified in the Twitter stream, between March 15 and May 24 2012 194
4.41 An example assignment of groups to virtual sensors. . 200
4.42 Approximating the data distribution in a sliding window. 202
4.43 Distribution of users based on how they discolse their location 205
4.44 Schematic interaction of our system's components . . . 216
4.45 Efficiency of geocoding using the Full and the Small gazetteers 220
4.46 Number of mapped, unmapped and overmapped locations by gazetteer usage 221

4.47 Effectiveness of our approach using the two gazetteers 223
4.48 #Times a trigger was raised, compared to the window size. a = 1min, r = 0.01, p=0.1 226
4.49 #Times a trigger was raised, compared to the window size. a = 5min, r = 0.01, p=0.1 227
4.50 Cardiograms of emotions, for three distinct timestamps in the United States 235
4.51 Summary of two events shown to the end-user, regarding the Champions League Final 238
4.52 List of identified events (and their summary), related to the eurovision contest 239
4.53 The overall GUI that the user sees. 240

List of Tables

2.1 k-max-coverage vs k-dispersion 49
2.2 Frequently used symbols 51
2.3 Evaluated algorithms 67
2.4 Basic data set characteristics 68
2.5 Comparing SkyDiver and Contour Representation techniques, Independent Dataset 74
2.6 Comparing SkyDiver and Contour Representation techniques, Forest Cover 76
2.7 Dataset Statistics 91

3.1 Feature-subsets and their respective value-spaces. . . 108
3.2 Dataset Statistics 119

4.1 Restrictions for some major query types. 136
4.2 Fields of the QueryLog relation 139
4.3 Jaccard Similarity between the GPS enabled crawls using the Streaming API 141
4.4 Crawling configurations 152
4.5 First 15 query terms by configuration 154
4.6 Template variables mapping for the query in Fig. 4.13. . 167
4.7 Structural properties of the topologies used in the experimental evaluation. 170
4.8 Description of the the GPS-driven crawls 180
4.9 Jaccard Similarity between the GPS-driven crawls . . . 181
4.10 Comparison of languages extracted from samples . . . 191
4.11 Efficiency of experiments (seconds) 191
4.12 Location dataset characteristics 218
4.13 Confusion matrix for the "oracle" experiment 222
4.14 #Times a trigger was raised, compared to the neighborhood range. a = 1min, w = 30, p = 0.1 225

4.15 Average Component Processing Time (ms) 228
4.16 Sample Summary of 15 Prominent Events Identified By *TwInsight* 231
4.17 Mapping of emotions to colors, and examples of corresponding tweets. 234

Preface

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

Chapter 1

Introduction

1.1 The Web 2.0 era and beyond

The World Wide Web has changed dramatically over the years since its initial inception, and is still evolving as new technologies emerge. Online services and applications are more pervasive nowadays, allowing users to share online aspects of their everyday lives. More importantly, users feel *comfortable* with doing so, which is a major shift in their attitude regarding privacy in digital environments. This general change in behavior has made the boundaries between the physical and online world less transparent.

Sharing of information takes place in various forms, ranging from simple button clicks, (e.g., "Like", "+1"), to structured and semi-structured data, (e.g., filling in forms, selecting from pre-defined options), to totally unstructured information, (e.g., natural language, pictures, online videos). Note that this information is no longer produced by large (academic) institutions and corporations, which was the case in the early days of the web. Rather, it is being generated by actual users, whose role has shifted from information consumers to active participants in the content creation process. It is also interesting that companies currently encourage user feedback -- although, they occasionaly try to game the system --, and users actively seek out opinions from others, in order to make informed decisions.

The common denominator is that users express their preferences and personal opinions on various topics, such as music, products, politics, etc. Although new technologies provide the necessary framework(s) for the users to express themselves, novel techniques are required to turn the available data into useful and actionable information. Such a need translates into interesting and challenging research questions, which we have to address, in order to provide the next generation services. For instance, more expressive query types are needed, whereby user preferences can be taken into account. At the same time, we should develop techniques that extract meaningful and insightful information from this high-volume, user-generated content.

Skyline and *top-*k queries are two such indicative examples. *Top-*k queries retrieve the k best items according to some scoring function $f()$. On the other hand, *skyline* queries support multi-objective optimization and are geared towards returning items with different trade-offs. In particular, the query is specified over a set of attributes, and the result contains those items for which we can not find an alternative that is better in the selected attributes. Both query types have their advantages and disadvantages. For example, top-k queries control the output size through the parameter k. However, the result is highly dependent on the ranking function $f()$, and selecting a good $f()$ is not a trivial task. On the other hand, skyline queries are easier for the user to grasp, given that preferences are defined on each attribute separately, but their query output size can become extremely large. It would, then, be very tedious for the user to inspect all the results manually. Consequently, we need techniques that combine the merits of both query types. Incorporating additional constraints in the result selection process, such as *diversification* of items, would also be highly desirable, especially if we consider the fact that *skyline* queries aim to return points with trade-offs.

Both of these query types capture *user preferences* in a *pre-usage* scenario, typically when users are searching for new items or information. Users may also express their preferences in a *post-usage* scenario, in the form of feedback. Within certain domains, feedback can be explicitly provided through a user's actions. For example, clicking on a result during a web search session is a form of positive feedback, whereas skipping results provides negative feedback for the skipped item. Similarly, selecting the directory where to save a web resource is an explicit form of feedback for this application scenario. Supporting user activities by making judicious use of such feedback would greatly improve the user experience.

User feedback can be provided in other formats as well, such as semi-structured and free text. Online reviews are a characteristic example of the latter form, that has received considerable attention in recent years. This increased attention is due to the impact that reviews have in the marketability of products. In fact, surveys have shown that users prefer products that have already been reviewed, so that they know the item's pros and cons, and can, therefore, make informed decisions. Through a combination of user feedback and product specifications, we can derive a rigid framework to analyze and compare such products. More specifically, based on the users' needs -- provided through their feedback -- and the extent to which a product can cover similar needs -- given by its characteristics --, we can identify how *competitive* products are. This is extremely useful for both item producers (e.g., the companies), as well as item consumers (e.g., the end users). Despite its importance, a formal framework to identify *competitive* items had been largely missing until now.The recent availability of online reviews has allowed us to test both the efficiency and efficacy of techniques that return the top-k *most competitive* products, with respect to a given item of interest.

Although online reviews have seen a sharp increase in numbers over the years, they are nowhere near the data volume produced in social media. Popular social media platforms have extremely high user adoption, with Facebook boasting more than 1.28 *billion* active users per month (as of March 31, 2014), and Twitter -- a later founded company -- having more than 255 million active users per month (as of July 2014). A driving force of these frameworks is their networking component, with people linking to one another, as a prerequisite to share information.

Undoubtedly, social media is among the most prolific areas for research nowadays, not only because of the user adoption, but also due to the usefulness of the data in various diverse disciplines: computer science, psychology, sociology and journalism to name a few. Moreover, there are practical applications where the data can be used. Advertising and community detection are typical use cases, whereas (real-time) event detection, interaction analysis, and user behavior understanding increasingly gain attention. Making sense of the user-genereated content in these mediums is also extremely challenging, because of the data volume and content diversity, which is as high as the underlying population and their interests.

Mining high volumes of data to identify (newsworthy) events is far from trivial. Previous techniques have focused on event monitoring, implying that the event is already identified or somehow known. Others simplify the problem by searching for specific keywords, which can accurately describe the event. Being able to identify events, regardless of their type, and without prior knowledge of any descriptive keywords, requires a different approach to tackle the problem. As an alternative, we can resort to psychological theories, according to which events impact the user psychologically (and most specifically their affective state), compelling them to externalize their thoughts. We argue that newsorthy events will impact large groups of users, and by monitoring a group's aggregate affective / emotional state, we will be able to capture abrupt changes and trace them back to the source, i.e., the event.
Within this research question, however, there are several other issues that need to be resolved first. For example, we must be able to identify an event's location, which could help us describe it, or better explain it. Extracting the affective state of a single user is challenging on its own, let alone for an entire group. Capturing abrupt changes requires a careful formulation of the problem, as well as efficient computation techniques, due to the high volumes of real-time data we are dealing with. Presenting the information in a suitable form, so that it is easily understandable, is another aspect we need to consider.

1.2 Contributions and roadmap of this thesis

The remainder of this thesis is organized as follows

- Chapter 2 discusses query types that incorporate user preferences. We discuss existing approaches and their shortcomings and suggest techniques to address them. In particular, we present a framework that can reduce the output size for skyline queries, by selecting a set of *divserse* skyline points. We formally define *diversification* in the context of skylines, and propose efficient algorithms to select the top-k most divserse skyline points. Moreover, we propose an approach to rank skyline points using an Information Retrieval-inspired scheme. Finally, we briefly discuss other techniques that combine notions from skyline and top-k queries, and attempt to control the output size.

- In Chapter 3 we talk about user preferences in two post-usage scenarios. More specifically, we propose a framework to identify competitors by leveraging information extracted from online reviews. We present efficient techniques to extract the top-k most competitive items, given an item of interest, and experimentally evaluate our algorithms using real datasets. We also evaluate experimentally the efficacy of our proposed formalism through a user study. In an alternative setting, we discuss how we can take feedback into account, to facilitate a user-initiated process. More specifically, we present a machine-learning framework to aid users in selecting the right directory during a download process. We discuss the features that we use for this particular problem, and conduct a user study to evaluate our approach.

- In Chapter 4 we propose techniques to mine social media data, with a focus on Twitter. Our goal is to *detect events*, as discussed by the users, while having Twitter as our only source of information. The constraint for using Twitter as a single source stems from a desire to report events as early as possible, with respect to their time of occurrence. After an initial review of the literature, we present our methodology for detecting events, using cognitive and affective theories of emotions developed in the psychology domain. We present efficient techniques that extract geographical information of users. We propose a classification-based framework to extract the affective state of users, and discuss how we can group users together, to monitor their aggregate emotional state. We formalize the problem of event detection in our setting, and experimentally evaluate the entire system, in terms of both efficiency and effectiveness. Detected events are displayed in a contextual user interface, conveying as much information as possible in a concise manner, to support decision making.

- Finally, Chapter 5 concludes this thesis, summarizing our key findings and contributions. We also discuss future steps to further our work in the various research axis presented herein.

Chapter 2

Managing user preferences

2.1 Introduction

Skyline queries, in the context of databases, were initially proposed in [42] and since then, they have attracted considerable attention by the database and data analysis community, as they perform multi-objective optimization without the need for user-defined scoring functions. The only input required by the user is the *preferences* regarding the minimization / maximization of attribute values. For example, if *price* and *quality* are two of the attributes, then users prefer to minimize price and maximize quality, selecting items which are (objectively) better than (i.e., *dominate*) others.
More formally, we assume a data set $\mathcal{D}$ composed of n points in a d-dimensional space. For each dimension, independently of the others, the user can define a specific preference, such as *min* or *max*, depending on whether they are interested in *minimizing* or *maximizing* the respective attribute. We say that a point p dominates another point q, where $p, q \in \mathcal{D}$, and denote $p \prec q$, if p is at least as good as q in all dimensions, and it is strictly better in at least one. The result $\mathcal{S}$ of a skyline query contains those points in $\mathcal{D}$ which are not dominated by any other.
To illustrate with an example, let us assume that our dataset consists of a set of hotels, for which we are interested in their price (y-axis) and distance to the beach (x-axis). We can present these items in a 2-dimensional plot, as shown in Fig. 2.1. In such a scenario, we want to *minimize* both attributes, i.e. select hotels as close to the beach as possible and as cheap as possible. The skyline consists of hotels

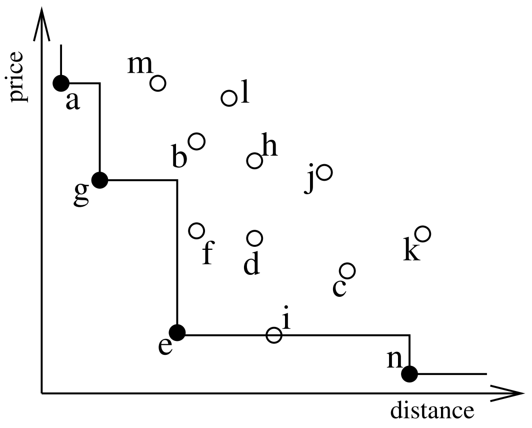


Figure 2.1: Example of the skyline on a set of hotels with 2 attributes

$\{a, g, e, n\}$, for which there is none cheaper and closer to the beach at the same time. On the other hand, we have no reason to select one of the other hotels -- assuming only these two dimensions --, as there is always a better alternative.

Past research on skyline queries in the context of databases has focused primarily on the efficiency aspect of the problem, i.e. how to compute the *skyline* set as quicklly as possible. In their seminal paper that introduced the *Skyline Operator*, Börzsönyi et al proposed two algorithms for this problem: a Block-Nested-Loop (BNL) and a divide-and-conquer (D&C) approach. BNL was later improved by presorting the points according to a monotone scoring function, resulting in *Sort-Filter-Skyline* (SFS) [59]. Kossman et al [121] proposed an algorithm based on nearest-neighbor search. Papadias et al. proposed *Branch-and-Bound Skyline* (BBS) [165], which uses a multidimensional index and it is proven to be I/O optimal. Processing of skyline queries received significant attention in other domains as well, including Peer-2-Peer networks, Mobile Ad-hoc Networks (MANETs), Parallel and Grid computing [30, 222, 111, 141, 132, 220, 66].

A major problem of the skyline, is that, depending on the data distribution and dimensionality, it is very likely that the result will contain a significantly high number of points. More formally, Bentley et al showed that for a randomly generated set of n points in d dimensions, the expected skyline cardinality is $|\mathcal{S}| = m = O((\ln n)^{d-1})$ [38]. In a data set containing 10^9 points, having about 10^3 skyline points may not be that much compared to the data set cardinality. However, in an era when "ten blue links" may sometimes seem too many [125], returning approximately 10^3 skyline points for manual inspection immediately negates the advantages of skyline queries, i.e., selecting high-quality points. Therefore, more sophisticated solutions are required.

2.2 Skyline diversification

2.2.1 Motivation

As already discussed, skyline queries suffer from the *skyline cardinality explosion* problem. Simply put, depending on the dataset characteristics, e.g., cardinality, dimensionality, distribution of points, the size of the skyline size may become too large. To overcome this issue, two main directions have been followed, both of which focus on the selection of a fixed-size subset of k skyline points.
The first alternative considers the entire dataset, selecting a set of k *skyline representatives*, which collectively dominate as many distinct points as possible [136]. The second alternative considers the skyline set alone, and selects k skyline points that best describe the skyline contour. Techniques that fall under this category include [238, 200] and make use of L_p norms as distance functions.
In this research, we also address the skyline cardinality explosion problem, by diversifying the skyline set, i.e. computing a subset of k skyline points with high diversity. The need for diversification arises in any context where there are users with varying tastes, e.g., web search [12, 18]. The skyline setting is no exception, considering that the query itself is designed to return results with varied trade-offs. For instance, some users may be looking for a "cheap" buy, whereas others for a "quality" one. Moreover, there may be some users who are interested in having an investigative look before proceeding with their purchase, therefore some (manual) post-processing is necessary to fulfill their needs [62, 166, 195]. Without knowing the user's true interests, our safest bet is to diversify the skyline result, to fulfill the needs of as many users as possible.
To this end, we propose a measure of diversity for skyline points which

is meaningful and intuitive given the setting, and, most importantly, builds upon the most fundamental skyline concept: the *dominance* relation. In particular, each skyline point is associated with its *dominated set* $\Gamma(p)$, i.e., the set of points that p dominates. The dominated set is an established measure of quality of skyline points [136, 230], thereby making it suitable as a building block for our diversification model.

More formally, the diversity between two skyline points p and q is defined as the *Jaccard distance* J_d of their corresponding dominated sets, i.e.,

$$J_d(p, q) = 1 - \frac{|\Gamma(p) \cap \Gamma(q)|}{|\Gamma(p) \cup \Gamma(q)|}$$

When $\Gamma(p)$ and $\Gamma(q)$ largely overlap, the diversity score will be small; conversely, sharing few dominated points results in high diversity. Finally, our diversification model inherently encourages large domination sets, because for a fixed number of commonly dominated points, the selected pair will be the one that collectively maximizes the domination score. The choice of J_d arises naturally, taking into consideration that it is the most widely used similarity measure for sets.

To motivate our approach further, consider Figure 2.2, where a set of points has been split into its skyline (upper) and the set of dominated points (lower). A directed edge exists to signify that the skyline point dominates the corresponding point at the bottom. This representation abstracts a multitude of domains:

- Nodes are product reviews and an edge exists when a product is at least as good as another.

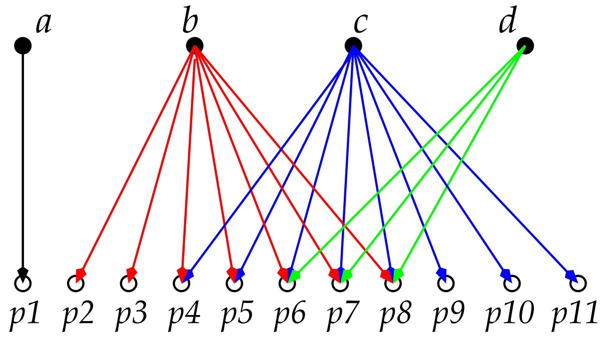


Figure 2.2: Graph with dominance relations.

- Nodes are web pages and a node dominates another if it contains at least as much information on a topic of interest.
- Nodes are web search results and an edge exists if a user selected one result over the others. The selected document becomes part of the skyline, whereas the rest (e.g., higher ranked documents) belong to the dominated set.

Note that the entire representation only relies on the *dominance* relation because this may be all we have. For instance, in our third example, we only know that a user preferred some documents over the rest, without explicitly knowing "*why*". Similarly, the data may belong to a 3rd party who has anonymized or obfuscated it and we are only presented with this dominance graph, but not the actual data values. This practically translates into an inability to build and use a multi-dimensional index.
We stress that diversification, as an objective, is *different* from coverage. Coverage focuses on the selection of skyline points that maximize the number of unique points that are being dominated. For example, in a 2 max-coverage approach, i.e., selecting the 2 skyline points that maximize the number of dominated points, the result would be the pair (b, c). However, their domination sets largely overlap, meaning that little *new* information will be provided. Similarly, d discusses topics already covered by both b and c. On the contrary, a may provide truly fresh information that none other does, despite the fact that it dominates a single point. An approach focusing on diversification, as the one that we propose, would return the pair (c, a): c dominates the most points and addresses a lot of the information found in b and d; a provides truly new information compared with c, and will attract users with a varied taste better than any other combination.
Overall, our contributions are briefly described as follows:

- We define a novel and intuitive measure of skyline diversity, which is solely based on the dominance property. In particular, the diversity between skyline points is computed as the Jaccard distance of their associated dominated sets. This makes our technique suitable for settings, such as partially-ordered domains or data with (multi-valued) categorical feature. Moreover, our formalism maintains the scale invariance property of skyline queries, contrary to other techniques, which also suffer from a need for additional distance functions or user input. Therefore, we advocate this measure as an intuitive approach for *(dis)similarity* computation between skyline points.

- Given our similarity measure, we proceed with presenting the problem of k-most diverse skyline points. The problem is mapped naturally to the k-dispersion problem. Since k-dispersion is NP-hard [123], we obtain a 2-approximation with respect to the optimal solution, by applying a greedy-based heuristic.
- To efficiently approximate the optimal solution, we propose the SkyDiver framework. Our framework may be applied regardless of the existence of an index; naturally, the use of an index may improve performance by reducing I/O operations. In particular, we employ *MinHash* signatures and provide theoretical guarantees for the effectiveness of our approach. Alternatively, *Locality Sensitive Hashing* (LSH) can be used, as a space-efficient approximation to MinHashing.
- We provide an extensive and comprehensive experimental evaluation of SkyDiver, using both real-life and synthetic datasets to verify our theoretical study, and experimentally portray the benefits of our approach against other alternatives that select a fixed subset of the skyline set.

The rest of this section is organized as follows. Related work is summarized in Section 2.2.2. Section 2.2.3 introduces some fundamental concepts with respect to the domain, and formally defines the problem. Some straightforward techniques to addressing the problem are discussed in 2.2.4, along with their inefficiencies. Our algorithms are studied in detail in Section 2.2.5 and evaluated experimentally in Section 2.2.6, using real-life and synthetic data sets.

2.2.2 Related Work

Diversity is a topic studied in several disciplines. Since the literature is very rich, we will briefly discuss in the following the most basic contributions that are closely related to ours.

Operations Research. Diversification in Operations Research has been used as a means of dispersion in optimization problems. In particular, this concept has applications in *facility location*, where the locations of k new stores or warehouses must be determined in order to be convenient to deliver products to clients. The intractability of the problem was first investigated in [123], where it was shown to be NP-Hard. Therefore, only approximation algorithms can be used to solve the problem in polynomial time, unless $P = NP$. Some heuristic-based algorithms are studied in [78], whereas [174] discusses problem variations in detail. They also showed that a 2-approximation is the best that we can get when the distance measure respects the triangular inequality. In [88] the authors provided a uniform treatment of the different dispersion problems and experimented with randomized heuristics. Finally, we note the work of [168] which studied upper bounds and exact algorithms for dispersion problems.

Information Retrieval. The need for some degree of result diversification became evident in the Information Retrieval (IR) community quite early [43]. It may serve as a means to cover various tastes or resolve query ambiguity [192], even after personalization factors have been taken into consideration. A diverse set of results may also serve as a good starting point for further query refinement or exploratory search [62]. In [12] a systematic approach to diversifying results is presented, aiming to minimize the risk of dissatisfaction of the average user. The problem is shown to be NP-hard, therefore, approximation algorithms are used to solve it efficiently. Finally, [18] investigates new scoring functions that take into account both the relevance and the diversity of the result set.

Database Management. The diversification problem has been also addressed by the database community. Diversification of XML results is studied in [140]. In [103], diversification is studied for points in Euclidean space and access methods are used. In [212], the DivDB system is developed, which provides result diversification by using an SQL interface, whereas [74] studied the dynamic case of the problem. Our research contributes to the data management discipline, by investigating diversification in the result of a skyline query[42], which is widely used to reveal the best items according to user preferences. Among the different algorithms proposed in the literature for skyline query processing, BBS [165] is preferred the most, because of two significant properties, namely *result progressiveness* and *I/O optimality*. However, in cases where indexing cannot be applied, one must resort to other alternatives. Two efficient algorithms for skyline computation without the use of an index are proposed in [69] and [191]. The first one is designed for the streaming case and performs multiple passes over the data returning approximate results. In contrast, the second one is designed for the I/O model and always provides correct results. In the following, we take a closer look at existing techniques that are mostly related to our work.

Skyline Diversity. Representing the skyline contour [200] has been suggested as an alternative for skyline diversification [238]. Both techniques use an L$_p$ norm (Euclidean distance in particular) as the measure of diversity between skyline points. This choice may be problematic in the following cases:

- the dimensions correspond to attributes that are difficult to combine (e.g., price and quality),
- the skyline is computed over a partially-ordered domain [234], and
- the attributes contain non-numerical values, e.g., when operating over a document collection where the attributes may be terms, q-grams or topics.

In situations such as the above, a multidimensional index can not be used, rendering previously proposed techniques infeasible or even inapplicable. Additionally, the Euclidean distance is sensitive to dimension scaling, meaning that a weighted distance measure might be more appropriate. Therefore, by selecting an off-the-shelf distance measure, the scale independence property of skylines is disregarded. More notable is the fact that only the skyline set $\mathcal{S}$ is used to determine a solution, disregarding the rest of the points. Note that this is in contrast with existing literature that accepts dominance power, i.e., the size of the dominated set $|\Gamma(p)|$, as a predominant quality characteristic of a skyline point [165, 230].

Coverage-based techniques. The techniques in [136, 84, 68] also consider the problem of reducing the skyline size and suggest to select a subset of k skyline points according to a maximum coverage criterion. In particular, the optimization goal is to maximize the number of distinct non-skyline points dominated by at least one of the k selected skyline points. Despite its set-oriented nature, this technique essentially solves a *different* problem, aiming to maximize the dominated set of the selected skyline points, and not to diversify them. Note that such a solution would have been highly attractive in conjunction with a greedy heuristic, as shown by the following lemma.

Lemma 1 *([47]) The greedy algorithm on a set-cover problem with finite VC-dimension v, yields an approximation ratio of O($v \log vc$), where c is the optimal solution.*

The set system of such a max-coverage instance has a finite VC-dimension [210] of d (d being the dimensionality of the problem) due to the axis-aligned hyper-rectangles of dominating regions, anchored to the upper right corner of the d-dimensional space [145]. From Lemma 1 and a reduction of max-coverage to set-cover, we can also expect a better approximation than the $1-1/e$ of the general case. To the best of our knowledge, such a remark has been largely overlooked in the skyline literature.
To better illustrate the difference between our objective and the one in coverage-based techniques, we have performed the following experiment: We computed the diversity and coverage scores, both by a k-dispersion and a k-max-coverage algorithm, for various data sets

Table 2.1: k-max-coverage vs k-dispersion

		k-max-coverage		**k-dispersion**	
Dataset	**k**	*coverage*	*diversity*	*coverage*	*diversity*
IND5M4D	2	98.4%	0.064	95.5%	1.000
	10	99.9%	0.064	95.8%	0.916
	50	100%	0.018	98.3%	0.553
FC5D	2	93.7%	0.304	88.6%	1.000
	10	98.9%	0.088	88.9%	0.941
	50	99.8%	0.032	93.2%	0.714
REC5D	2	70%	0.634	56.2%	1.000
	10	93.1%	0.328	56.7%	0.997
	50	98.6%	0.142	68.6%	0.864

(see Section 2.2.6 for details). Table 2.1 contains the results of this experiment.
We draw the following conclusions:

- Clearly, we can not solve the diversity problem through coverage. Coverage selects points with high overlap in their dominating regions, which sharply reduces diversity.
- When the objective is diversity, coverage is not as high as when aiming for coverage per se, but it is still high enough. This was expected, since the diversity measure tends to select points that cover a good portion of the dataset from their own viewpoint.

2.2.3 Problem Definition

Let $\mathcal{D}$ be a d-dimensional dataset, where without loss of generality smaller values are preferred, i.e., we are interested in *minimizing* each attribute.[1] We say that p = ($p.x_1$, ..., $p.x_d$) $\in \mathcal{D}$ *dominates* q = ($q.x_1$, ..., $q.x_d$) $\in \mathcal{D}$ (and write $p \prec q$), when: $\forall i \in \{1, ..., d\}, p.x_i \leq q.x_i \wedge \exists j \in \{1, ..., d\} : p.x_j < q.x_j$. The skyline $\mathcal{S} \subseteq \mathcal{D}$, is composed of all points in $\mathcal{D}$ that are not dominated by any other point.
Given a data set $\mathcal{D}$, the skyline set $\mathcal{S}$ and an integer k, $k \geq 2$, the goal of the diversification process is to return a subset $S_k \subseteq S$ containing k skyline points, aiming to maximize their diversity, i.e., the *dissimilarities* among the skyline points. To quantify the diversity between two skyline points we need a distance function $d : \mathcal{S} \times \mathcal{S} \rightarrow \Re$. Table 2.2 depicts the basic notations that are used frequently.
To overcome the limitations discussed in Section 2.2.2, we propose to use the *Jaccard distance* for diversity computation. Each skyline point p is associated with a subset of $\mathcal{D}$, containing all points dominated by p, denoted as $\Gamma(p)$, i.e.

$$\Gamma(p) = \{q \in \mathcal{D} | p \prec q\}$$

The *domination score* of p is the cardinality of $\Gamma(p)$. The similarity between p and q is defined as the Jaccard similarity between the sets

[1]We focus on numerical attributes for ease of presentation. Our approach applies to categorical ones equally well.

Table 2.2: Frequently used symbols

Symbol	**Description**
$\mathcal{D}$, $n = \lvert\mathcal{D}\rvert$	the data set and its cardinality
$\mathcal{S}$, $m = \lvert\mathcal{S}\rvert$	the skyline set and its cardinality
s_j	the j-th skyline point
k	number of diverse skyline points
t	size of each signature
M, $\widehat{M}$	domination and signature matrix
ξ	LSH similarity threshold
ζ	number of zones for LSH
$\Gamma(p)$	set of points dominated by p
$J_s(p,q)$	Jaccard similarity between p, q
$J_d(p,q)$	Jaccard distance between p, q
$\widehat{J_d}(p,q)$	Jaccard distance for signatures

$\Gamma(p)$ and $\Gamma(q)$, i.e.,

$$J_s(p,q) = \frac{|\Gamma(p) \cap \Gamma(q)|}{|\Gamma(p) \cup \Gamma(q)|}$$

and ranges between 0 and 1. The corresponding distance measure is thus $J_d(p,q) = 1 - J_s(p,q)$ and it is well known that it satisfies all metric properties. The selection of the Jaccard distance as a measure of diversity was based on the following rationale:

i) it relies solely on the dominance relations among points, therefore, no user-defined distance function or other input is required,

ii) the quality of the resulting set of points does not depend on the skyline $\mathcal{S}$ alone, but on the characteristics of $\mathcal{D}$ as well

iii) it leads to elegant ways of diversity computation by means of min-wise independent permutations, and

iv) it is the most widely accepted measure for set (dis)similarity.

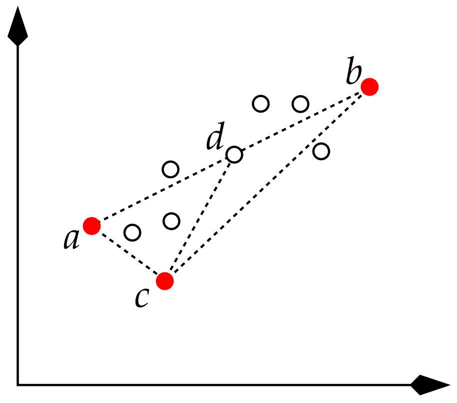
To facilitate diversification, we take the perspective of [74], viewing k-diversity as a *dispersion* problem. In k-dispersion, the goal is to find k

objects such that an objective function of their distance is optimized. The optimal solution of the k-dispersion problem is given by:

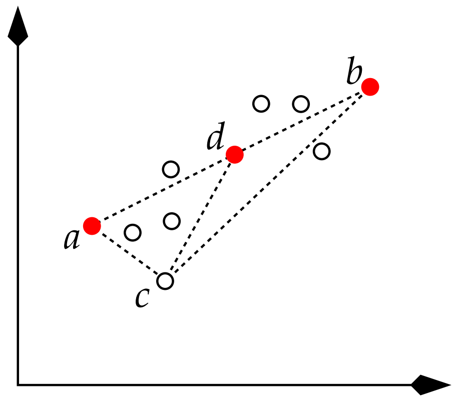
$$OPT = \arg\max_{\substack{\mathcal{A} \subseteq \mathcal{S} \\ |\mathcal{A}|=k}} f(\mathcal{A})$$

There are two basic alternatives for the objective function: *i*) in the k-MSDP (Max-Sum Dispersion Problem) the goal is to maximize the *sum of the pair-wise distances*, and *ii*) in the k-MMDP (Max-Min Dispersion Problem) the goal is to maximize *the minimum pair-wise distance*. Although either alternative can be employed, we choose to work with k-MMDP because it leads to 2-approximation algorithms, instead of the 4-approximation of k-MSDP [174].

Example 1 *Figure 2.3 illustrates the output of a k-MMDP and a k-MSDP for k=3. For simplicity, assume that objects are 2D points and the L$_2$ distance is used as the measure of diversity. By inspecting the two solutions, we observe that the solution for k-MMDP returns points that are more distant to each other than k-MSDP. Both solutions contain the objects a and b. However, k-MMDP returns d as the third point, whereas k-MSDP returns c. Observe that the distance between a and c is smaller than the distance between a and d. Thus, in k-MSDP, although the sum of distances between the returned points is maximized, small distances may still occur, because they are compensated by larger ones.*



(a) solution for 3-MSDP



(b) solution for 3-MMDP

Figure 2.3: Solutions to dispersion problems.

2.2.4 Straight-Forward Techniques

Before going into the particulars of our proposed framework, we describe briefly some straight-forward approaches and we report on their efficiency and effectiveness.

Brute-Force. This algorithm generates all pair-wise distances between skyline points, evaluates all $\binom{m}{k}$ alternatives and selects the optimal solution. Clearly, this method suffers from performance degradation by increasing the number of skyline points or the value k. In addition, there is a $O(m^2)$ cost to compute all pair-wise distances of the skyline points.

Simple Greedy. This method avoids the computation of all pair-wise distances among skyline points, by employing a heuristic-based algorithm, which guarantees a 2-approximation of the optimal solution. The main drawback of this approach is that in order to compute the Jaccard distance of two skyline points p and q, range queries must be executed to determine the cardinalities of the dominating sets $\Gamma(p)$, $\Gamma(q)$ and $\Gamma(p) \cap \Gamma(q)$. Evidently, the cost of such an approach is prohibitive, both with respect to I/O and CPU time, even when an aggregate multidimensional index is available.

Sampling-Based. One may be inclined to think that sampling $\mathcal{S}$ or $\mathcal{D}-\mathcal{S}$ will lead to a reduction of the cost to compute the k-most diverse skyline points. Taking a sample from $\mathcal{S}$ means that less than m skyline points will participate in the selection process, whereas sampling from $\mathcal{D}-\mathcal{S}$ results in fewer points that will contribute to the computation of the Jaccard distance between skyline points. In our case, sampling is not helpful as we discuss in the sequel.

Lemma 2 *Let S be a set of m items in a metric space and $\triangle$ the diameter (maximum pair-wise distance). Any one-pass deterministic or randomized algorithm, that uses less than or equal to $m/2$ items, will fail with probability at least 1/2 to compute $\triangle$ exactly or provide a 2-approximation.*

Proof 1 *We focus on data sets containing exactly m points. Each point is uniquely identified by its id between 1 and m. Define the data sets $D_1, D_2, ..., D_m$ as follows. Each data set D_i contains a set of $m-1$*

points that are clustered together in a minimum bounding sphere with diameter δ, whereas the point with id=i, $1 \leq i \leq m$ is located at a distance $2\delta + c$ from the center of the sphere, where c is a small constant. Let A be a deterministic algorithm that uses $s < m$ space. A randomly chosen D_i is selected and given as input to A. Each point of D_i is presented to A as a stream of points.
Algorithm A selects s points to maintain in a deterministic manner. Each pair of nodes has the same probability of being the one with the maximum distance. Since we have $\binom{m}{2}$ different distances from which exactly one is the maximum, with s points we can produce $\binom{s}{2}$ pair-wise distances, meaning that the probability of success is

$$P(success) = s(s-1)/m(m-1)$$

and the probability of failure is

$$P(failure) = 1 - s(s-1)/m(m-1)$$

Setting $s = m/2$ we get that

$$P(failure) = \frac{3m-2}{4m-4} \geq \frac{1}{2}, \forall m \geq 2$$

For the 2-approximation case, the difference is that the i-th element must be included in the s points selected by the algorithm. The distance between p_i and any of the other points lying inside the sphere is guaranteed to be at least $\Delta/2$. Thus, the success probability in this case equals s/m and consequently, the failure probability is $1 - s/m$. Therefore, even by maintaining $m/2$ elements from S any deterministic algorithm will err with probability 1/2.
Using Yao's minimax principle [228], the effectiveness of any one-pass randomized algorithm cannot be better.

Basically, the previous result states that if one wants to get a success probability larger than 1/2 in estimating the diameter Δ (i.e., the optimal solution to the 2-dispersion problem), at least $m/2$ points must be stored by any deterministic or randomized algorithm. The result may be extended for any k. On the other hand, sampling from $\mathcal{D} - \mathcal{S}$ is not an effective solution either, because of the *sparsity* issue.

To illustrate this effect, assume that the data set is viewed as a *domination matrix* [2] M with $n - m$ rows and m columns, $m = |\mathcal{S}|$ and $n = |\mathcal{D}|$. Therefore, each skyline point is represented by a single column, whereas a dominated point is represented by a row. In this matrix, the cell in the i-th row and the j-th column is 1 if the j-th skyline point dominates the i-th data point and 0 otherwise. The sparsity of the domination matrix depends heavily on the data distribution as well as on the dimensionality of the data space. As an example, for 10,000 uniformly distributed points, in 3 dimensions the percentage of zeros is 45%, in 5 dimensions it is 84% and in 7 dimensions the percentage of zeros reaches 97%. The percentage of zeros is higher in anticorrelated data sets. It is evident that with the existence of sparsity it is not possible to simply perform random sampling and then try to compute the diversity among skyline points. Such an approach could miss important parts of the columns (containing 1's), resulting in erroneous diversity computation.

2.2.5 The SkyDiver Framework

Given the shortcomings of the aforementioned solutions, more suitable techniques are required. In this section, we present the SkyDiver framework for the skyline diversification problem, which consists of two consecutive phases, *fingerprinting* and *selection*:

Phase 1: Fingerprinting. This phase generates a *signature* of reduced size for each skyline point, based on MinHashing. Alternatively, *Locality Sensitive Hashing* (LSH) can be employed as a memory efficient approximation of the MinHash signatures.

Phase 2: Selection. This phase is responsible for selecting the k most diverse skyline points. This step may be applied to either the MinHash or the LSH signatures.

2.2.5.1 Phase 1: Fingerprinting with MinHashing

The basic objective of the following method is threefold:

[2] This matrix is used only for illustration purposes and it is not constructed in practice.

1. to avoid the execution of range queries,
2. to avoid the computation of all $O(m^2)$ pairwise diversities, and
3. to be able to work either with or without an index.

In this respect, we propose the use of the *MinHashing* technique [46], because it fits nicely with our diversity measure and requires a single pass over the data. Each column of the domination matrix (i.e., skyline point) is represented by a *signature* of size t, such that $t << (n-m)$. Let $\mathcal{H}$ = {h_1, ..., h_t} be a set of t min-wise independent hash functions, where each h_i performs a random permutation of the rows. The cardinality of $\mathcal{H}$ (i.e., the number of hash functions used) determines the size of each signature. To generate random permutations of rows, each hash function $h_i \in \mathcal{H}$ is of the form

$$h_i(x) = a_i \cdot x + b_i \mod P$$

where P is a prime number larger than $n-m$ and a_i, b_i are randomly chosen constants taking integer values in $[1, P]$. Although such a family of hash functions does not satisfy the *min-wise independence property*, it is used as an approximation that works very well in practice. Moreover, it has been shown in [46] that the MinHash technique has a very nice property that is directly related to the Jaccard similarity. More specifically, if $J_s(p, q)$ is the Jaccard similarity between skyline points p and q, then for each hash function h_i it holds that

$$Prob[h_i(p) = h_i(q)] = J_s(p, q).$$

Recall that each row of the domination matrix M corresponds to a bit-array. If the j-th position of the i-th row is 1 then the j-th skyline point dominates the i-th point. Each row is hashed t times using the hash functions in $\mathcal{H}$ and the signature of each skyline point is updated accordingly. Therefore, each signature is composed of t integer values, capturing the first row identifier with a non-zero element for each permutation.

Index-Free Signature Generation. To speed up performance, most techniques that deal with skyline queries assume the presence of a multi-dimensional indexing scheme, e.g., an R-tree [100]. However, there are many reasons why the data set may not be supported such an index. Briefly, some of them are:

- high dimensionality, which deteriorates indexing performance,
- the data set may contain intermediate results and thus no index is available yet,
- operations performed on a projection of the data set in specific dimensions make the index inapplicable and
- the data set contains *categorical* attributes that prevent multi-dimensional indexing.

Algorithm 2.1 outlines the signature generation process, when an index is not available. The algorithm takes as input the set of skyline points $\mathcal{S}$, the family of hash functions $\mathcal{H}$ and the number t of signature slots. The output is a matrix $\widehat{M}$ with t rows and m columns, where m is the cardinality of the skyline set. Each column of $\widehat{M}$ stores the MinHash signature of the corresponding skyline point. Each data point is scanned once (Line 2) and it is checked against the skyline points to detect dominance relationships. If a skyline point s_j dominates the

Algorithm 2.1 SigGen-IF

```
    Input: $\mathcal{D}$ data set, $\mathcal{S}$ skyline set, $\mathcal{H}$ hash functions, $t$ number of slots per signature
    Output: $\widehat{M}$ signature matrix
 1: Initialize all cells of matrix $\widehat{M}$ with $\infty$;
 2: for ($rowcount \leftarrow 1$ to $|\mathcal{D}|$) do /* read data points */
 3:     $p \leftarrow$ next data point;
 4:     if ( $p$ is a skyline point ) then continue;
 5:     for $j \leftarrow 1$ to $|\mathcal{S}|$ do
 6:         if ($s_j \prec p$) then
 7:             UpdateMatrix( $rowcount$, $j$ );
 8:         return($\widehat{M}$);

    Procedure UpdateMatrix( $row$, $column$ )
 9: for ($i \leftarrow 1$ to $t$) do
10:     $v_i \leftarrow h_i(row)$; /* apply hash function */
11:     $\widehat{M}[i, column] \leftarrow min(\widehat{M}[i, column], v_i)$;
```

investigated point (Line 6), then the matrix containing the MinHash signatures is updated accordingly (Line 7). The procedure to update the signature matrix is given in Lines 9--11, where we iteratively apply the hash functions.

Note that the index-free technique requires a single pass over the multidimensional data set, provided that the skyline set is available. The advantage of this method is that no index is required, whereas the sequential scan of the data set is expected to be efficient taking into consideration that usually the data file is stored sequentially on the disk. Most notably, such an approach does not require that attributes are numeric, but can handle any domain (e.g., categorical, partially ordered) as long as dominance is well-defined. However, in cases where an index is already available, more efficient processing is possible, which we investigate in the next paragraphs.

Index-Based Signature Generation. Typically, data points that are close in the multidimensional space are expected to be dominated by the same subset of skyline points. This feature is unique in index-based techniques since the sequential scan of data points does not guarantee any locality of references, unless the data is presorted based on a spatial proximity criterion (e.g., space filling curves). Therefore, when an index is present, we can exploit this property and reduce processing costs by avoiding index probes. We discuss in detail the appropriateness of each approach in Section 2.2.6.

To better understand the index-based technique, we introduce two concepts regarding the dominance of Minimum Bounding Rectangles (MBRs), namely i) full and ii) partial dominance. MBRs are used by multi-dimensional indexing methods (e.g., R-Trees) to organize and group together data points that are close in the multidimensional space. Full dominance means that the lower left corner of the MBR is dominated. Partial dominance means that the MBR is *not* fully dominated, but its upper right corner is. We illustrate with an example to better explain the two notions.

Example 2 *Consider the set of points in Figure 2.4, enclosed by the minimum bounding rectangles R_1, R_2 and R_3. The skyline set is composed of a, b and c. Evidently, R_1 is fully dominated by b, whereas R_2 is fully dominated by a, b and c. Neither MBR is partially dominated.*

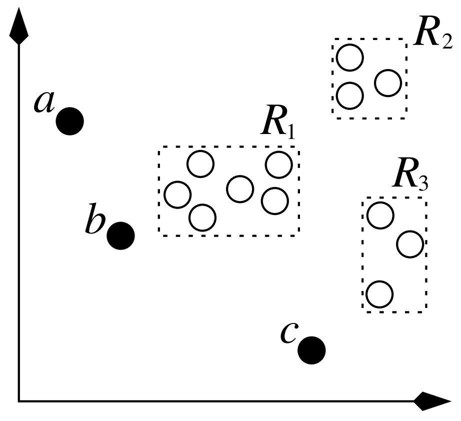


Figure 2.4: Domination of MBRs.

Therefore, for these two MBRs we avoid expanding the search toward the leaf and update the signatures immediately. In contrast, we have to increase the level of detail for R_3, because although it is fully dominated by c, it is partially dominated by b.

Algorithm 2.2 outlines the MinHash signature creation in the presence of an aggregate R-tree. A priority queue PQ is used to store R-tree entries that require further consideration. The algorithm removes the top entry e from PQ (Line 11) and checks whether it is partially or fully dominated by a skyline point (Line 14). Full dominance means that the lower left corner of e is dominated, whereas partial dominance means that e is *not* fully dominated, but its upper right corner is. Partial dominance prevails and if both relations exist, we need to visit e's subtree (Line 16) by queuing it in PQ. In case of exclusive full dominance, $UpdateFullDominance$ is called (Line 18), which updates the signatures without probing the index, by iterating over the number of enclosed objects in e (Lines 20--23).

2.2.5.2 Phase 2: Selection of Diverse Skyline Points

The next step involves the selection of k skyline points, aiming to maximize their diversity. To perform this step efficiently, we should avoid the computation of the exact diversity score between all pairs of skyline points, since that would be a costly operation. In particular, assuming that an R-tree index is available, this process would perform

Algorithm 2.2 SigGen-IB

```
    Input: $\mathcal{D}$ data set, $\mathcal{S}$ skyline set, $\mathcal{H}$ hash functions, $t$ number of slots per signature
    Output: $\widehat{M}$ signature matrix
 1: $rowcount \leftarrow 1$;
 2: Initialize all cells of matrix $\widehat{M}$ with $\infty$;
 3: Initialize priority queue $PQ$;
 4: for entry $e$ in $R$.root do
 5:     DominatedRel( $e$, $full$, $partial$ );
 6:     if ( !$partial.isEmpty$ ) then
 7:         $PQ$.insert( $e$ );
 8:         continue;
 9:     UpdateFullDominance( $e$, $full$ );
10: while ( !$PQ.empty$ ) do
11:     $e \leftarrow PQ.removeTop()$;
12:     $node \leftarrow R.read(e.id)$;
13:     for each entry $e'$ in $node$ do
14:         DominatedRel( $e'$, $full$, $partial$ );
15:         if ( !$partial.isEmpty$ and !$node.isLeaf$ ) then
16:             $PQ$.insert( $e'$ );
17:             continue;
18:         UpdateFullDominance( $e'$, $full$ );
19: return $\widehat{M}$ ;

    Procedure UpdateFullDominance( $e$, $fullDom$ )
20: for $k \leftarrow 1$ to $e.count$ do
21:     for $j \leftarrow 1$ to $|fullDom|$ do
22:         UpdateMatrix( $rowcount$, $S$.index($fullDom_j$) );
23:     $rowcount$++;
```

a quadratic number of range queries of large volume, to compute the Jaccard distances. Alternatively, if an index is not available, we would have to perform multiple passes over the raw data. Both of these cases lead to significant computation costs.

Thus, in our framework we exploit the signatures only, avoiding access to the raw data. We study two different techniques: the first one is based solely on the MinHash signatures whereas the second applies Locality-Sensitive Hashing aiming to reduce memory consumption and enable speed/accuracy trade-offs.

The Signature-Based Method. Let $\widehat{J}_s(p,q)$ be the *estimated* Jaccard similarity defined as the fraction of signature positions of p and q where their values agree. The corresponding *estimated* Jaccard distance $\widehat{J}_d(p,q)$ is simply $1-\widehat{J}_s(p,q)$. Since we have t different hash functions, formally we have:

$$\widehat{J}_s(p,q) = \frac{|j : 0 \leq j \leq t, h_j(p) = h_j(q)|}{t}$$

Lemma 3 *([172]) The distance function $\widehat{J}_d$ respects the triangular inequality.*

According to the previous discussion, the set of signatures along with the distance measure $\widehat{J}_d$ is a *metric space*. We need this result to apply a greedy heuristic for the k-dispersion problem that guarantees a 2-approximation with respect to the optimal solution [174].
The algorithm first determines the two points with the maximum pairwise distance, and then greedily adds more points to the result set, trying to maximize the minimum distance. The problem with this approach is that it requires quadratic complexity $O(m^2)$ to determine the two most distant points. Here, we use a different technique with $O(k^2m)$ complexity without sacrificing the 2-approximation guarantee. The basic difference with respect to the work in [174] is that instead of selecting the two most distant points, we start with the skyline point with the maximum domination score and then use the greedy approach to include more points, until k points are determined. We also resolve ties by selecting the points with highest domination score, thereby treating coverage as a secondary objective.
The algorithm outline for selecting k diverse skyline points is given in Algorithm 2.3. Next, we show that SelectDiverseSet achieves a 2-approximation with respect to the optimal solution. This can be proved by using the same rationale as the one used for the proof of Theorem 2 of [174]. Here, we give a simpler alternative.

Lemma 4 *Algorithm* SelectDiverseSet *reports a set of k skyline points in $O(k^2m)$ time, achieving a 2-approximation with respect to the optimal solution.*

Algorithm 2.3 SelectDiverseSet

```
   Input: $\mathcal{S}$ skyline set, $k$ number of required points, $F(.)$ distance measure used
   Output: $\mathcal{A}$ set of diverse skyline points
1: $m \leftarrow |\mathcal{S}|$; /* cardinality of skyline set */
2: $p \leftarrow$ skyline point in $\mathcal{S}$ with max dominance score;
3: $\mathcal{A} \leftarrow \{p\}$
4: while ($|\mathcal{A}| < k$) do
5:     $x \leftarrow \arg\max_{x \in \mathcal{S}-\mathcal{A}} \min_{y \in \mathcal{A}}\{F(x,y)\}$, i.e., find a point $x \in \mathcal{S}$-$\mathcal{A}$ such that the min distance $F(.)$ between $x$ and the points in $\mathcal{A}$ is maximized;
6:     Resolve ties by selecting the point with the max dominance score;
7:     $\mathcal{A} \leftarrow \mathcal{A} \cup \{x\}$;
8: return $\mathcal{A}$;
```

Proof 2 *The first point is selected in $O(m)$ time. To select the second one we compute the distance between each of the $m-1$ remaining points and the one selected. To select the third point we need $2(m-2)$ distance computations. Thus, to select all k points we need in total $m + (m-1) + 2(m-2) + \ldots + k(m-k) \in O(k^2 m)$ distance computations. Let, S_j denote the set containing the j skyline points selected so far, where $j \leq k$. Thus, when the first point is selected we have $|S_1| = 1$, whereas when all k points are selected $|S_k| = k$. Let p_1 be the first point selected. To select the second point p_2, the algorithm scans all skyline points and picks the one that maximizes its distance from p_1. Create a neighborhood $\mathcal{N}(s_i)$ for each skyline point $s_i \in S_j$, such that $\mathcal{N}(s_i) = \{q \in D : F(s_i, q) < OPT/2\}$. We argue that there must be a point in D not belonging to any of the j neighborhoods. If this is not the case, then the optimal solution to the k-MMDP problem could not be OPT, which is a contradiction. Thus, for any $j \leq k$ it is guaranteed that the minimum distance between points in S_j is at least $OPT/2$.*

According to [71], if $\Omega(\varepsilon^{-3}\beta^{-1}\log(1/\delta))$ is the signature size, where ε is the maximum allowed error ($0 < \varepsilon < 1$), then with probability at least $1-\delta$ it holds that

$$(1-\varepsilon)J_s(p,q) + \varepsilon\beta \leq \widehat{J}_s(p,q) \leq (1+\varepsilon)J_s(p,q) + \varepsilon\beta$$

where $0 < \beta < 1$ is the required precision. This essentially means that the 2-approximate greedy heuristic using $\widehat{J}_d$ as the distance measure, is applied on the (ε,δ)-approximation of the Jaccard distances, for which the inequalities are

$$(1+\varepsilon)J_d(p,q) - \varepsilon - \varepsilon\beta \leq \widehat{J}_d(p,q) \leq (1-\varepsilon)J_d(p,q) + \varepsilon - \varepsilon\beta$$

Consequently, by setting appropriate values for ε and δ, the signature distances can be made very close to the original Jaccard distances [63]. Due to distance distortions, as a result of embedding the distances in lower dimensionality, it is possible to obtain a sub-optimal solution. The following theorem relates the true optimal solution, to the one computed by working with MinHash signatures.

Theorem 1 *Let OPT be the value of the optimal solution to the k-diversity problem in the original space and let x, y denote the corresponding skyline points, i.e., $J_d(x,y) = OPT$. Similarly, let $\widehat{OPT}$ be the optimal value if the problem is solved using MinHash signatures and let a, b be the corresponding skyline points, i.e., $\widehat{J}_d(a,b) = \widehat{OPT}$. For a given ε and sufficiently small δ, it holds that: $J_d(a,b) \geq \frac{1+\varepsilon}{1-\varepsilon}OPT - \frac{2\varepsilon}{1-\varepsilon}$.*

Proof 3 *The optimal solution is given by a set of k points, $O_k = \{o_1, o_2, ..., o_k\}$, forming a k-clique, where nodes are skyline points and edges are weighted with the J_d of the adjacent points. Each k-clique is represented by a single value, i.e., the minimum of the $\binom{k}{2}$ distances among any two points in the clique, $R(O_k) = min_{\forall o_i, o_j}(J_d(o_i,o_j))$. This representation stems from our formalization of k-diversity as an instance of the k-MMDP, where we are interested in maximizing the minimum distance of the selected set. According to this abstraction, OPT is the representative distance of the optimal solution.*
For any set of k points, $P_k = \{p_1, p_2, ..., p_k\}$, $P_k \neq O_k$, it holds that

$$R(P_k) = \min_{\forall p_i, p_j} J_d(p_i,p_j) \leq R(O_k) = OPT$$

In other words, any other k-clique will either be at most as good as O_k but never better, e.g., containing an edge with equal minimum weight, or it will contain at least one distance strictly worse than OPT.

Let $\widehat{O}_k$ be the set of k points that we select as the optimal solution in the MinHash signature space and $\widehat{OPT}$ be their representative distance defined by the skyline points a and b, i.e. $\widehat{J}_d(a,b) = \widehat{OPT}$. It is not hard to verify that if $\widehat{O}_k = O_k$, or $\widehat{O}_k$ contains $R(O_k)$ but no worse edge, then $J_d(a,b) = \widehat{J}_d(a,b) = OPT$ and the inequality surely holds. The challenging case is when $\widehat{O}_k$ contains some points whose edge is worse in the original space than in the signature space. Specifically, the problem arises when in the signature space $\widehat{J}_d(a,b) \geq \widehat{J}_d(x,y)$, despite that $J_d(a,b) < OPT$ in the original space.
Let D_O be any distance from the optimal solution O_k in the original space, $OPT \leq D_O$. It holds,

$$OPT \leq D_O \overset{\varepsilon>0}{\Longrightarrow} (1+\varepsilon)OPT - \varepsilon \leq (1+\varepsilon)D_O - \varepsilon$$

Simply put, underestimating a higher value than OPT could also yield a sub-optimal result, but the worst case is obtained when we underestimate OPT itself. The same holds for overestimating $J_d(a,b)$ and lower values. On the other hand, overestimating a value $J_d(w,z)$, $J_d(w,z) < J_d(a,b) < OPT$ so that $\widehat{J}_d(w,z) \geq \widehat{J}_d(a,b) \geq \widehat{J}_d(x,y)$ contradicts our assumption that $\widehat{J}_d(a,b)$ is the optimal solution in the signature space; otherwise, $\widehat{J}_d(w,z)$ would have been selected. For the worst case scenario to occur, i.e., $\widehat{J}_d(a,b) \geq \widehat{J}_d(x,y)$, where $J_d(a,b)$ has been overestimated and OPT has been underestimated, it should hold:

$$\widehat{J}_d(a,b) \geq \widehat{J}_d(x,y) \Leftrightarrow (1-\varepsilon)J_d(a,b) + \varepsilon \geq (1+\varepsilon)OPT - \varepsilon \Leftrightarrow$$

$$J_d(a,b) \geq \frac{(1+\varepsilon)}{(1-\varepsilon)}OPT - \frac{2\varepsilon}{1-\varepsilon}$$

Corollary 1 *Let a and b be the two skyline points defining the solution of the 2-approximation heuristic for the k-diversity problem, when run over the MinHash signatures, where $J_d(a,b)$ is their corresponding distance. Also, let OPT be the optimal distance. Then, the following holds: $J_d(a,b) \geq \frac{1}{2}\frac{(1+\varepsilon)}{(1-\varepsilon)}OPT - \frac{\varepsilon}{1-\varepsilon}$.*

For the previous bounds to work, we have assumed that the parameter δ is very small. This is because if the distances are not well-preserved in the signature space, then we cannot have a guarantee about the solution in the original space.

2.2.5.3 The LSH-Based Method

A limitation of the MinHash signature-based approach is that the size of the signatures may have to be increased significantly in order to reduce the error probability, resulting in: *i*) increased processing costs during distance computation and *ii*) increased memory requirements. Keeping the signature size as small as possible has a direct negative impact on accuracy, due to Theorem 1, a result we also experimentally validate. To address this problem we propose to apply *Locality Sensitive Hashing* (LSH) [113].

Recall the signature matrix that we introduced earlier, where columns represernt skyline points and rows correspond to hash function. We split the signature matrix to ζ zones, each containing r rows such that $\zeta \cdot r = m$. For each zone, a hash function is applied and each signature part is hashed to a bucket.

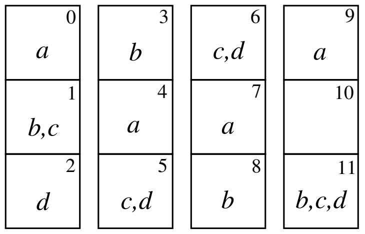
Based on this scheme, the probability that two skyline points s_1, s_2, will hash to different buckets in all zones equals [3] $(1 - J_s(s_1, s_2)^r)^\zeta$, whereas the probability that they will hash to the same bucket in at least one zone equals $1 - (1 - J_s(s_1, s_2)^r)^\zeta$. Our target is *to select skyline points that are hashed in different buckets in all zones* or if this is not possible, *to minimize the number of skyline points that fall in the same bucket in some of the zones*. The values of r and ζ are controlled by the value of the threshold ξ, which is selected such that $\zeta \cdot r = m$ and $\xi \approx (1/\zeta)^{(1/r)}$. Basically, the threshold ξ controls the shape of the sigmoid function $1 - (1 - J_s(s_1, s_2)^r)^\zeta$. For example, we may assume that we consider two points similar if their similarity is more than 20%, 50% or 80%.

Let B denote the number of buckets used per zone. Each skyline point is seen as a bit-vector containing $\zeta \cdot B$ dimensions, where a

[3]To be precise, we should use $\widehat{J}_s(s_1, s_2)$, but since the distortion of the similarities can be made arbitrarily small, we can safely assume that $\widehat{J}_s(s_1, s_2) \approx J_s(s_1, s_2)$.

value of 1 (0) denotes that the skyline point is hashed (not hashed) to the corresponding bucket. Consequently, two skyline points s_1, s_2 are dissimilar if they both have a value of 1 in as few bit-vector positions as possible. Therefore, the diversity is quantified by the *Hamming distance* between their corresponding bit-vector representations. In particular, we observe that the number of buckets where two skyline points disagree equals half the Hamming distance between their corresponding bit-vectors. Note also that since each skyline point is necessarily hashed in exactly one bucket in each hashtable, the L_1 norm of its bit-vector is equal to ζ, i.e., $||bv(s_i)||_1 = \zeta, \forall i \in [1, m]$.

Example 3 *Figure 2.5 depicts a possible distribution of signatures into buckets, when ζ=4 and B=3. The number shown in the upper-right corner of each bucket corresponds to the position in the bit-vectors, which are presented on the right. Each bit-vector contains $\zeta \cdot B$=12 bits, where exactly four of them are set. By observing points a and b we see that they are never hashed to the same bucket. Therefore, their distance should be equal to 4, whereas one can easily verify that the Hamming distance between $bv(a)$ and $bv(b)$ is 8.*



(a) hashtables for the zones

$bv(a)$= 100 010 010 100

$bv(b)$= 010 100 001 001

$bv(c)$= 010 001 100 001

$bv(d)$= 001 001 100 001

(b) skyline bit-vectors

Figure 2.5: Buckets and bit-vectors of skyline points.

Since the Hamming distance satisfies the triangular inequality, the 2-approximation heuristic is immediately applicable. Thus, to determine the k most diverse skyline points, algorithm SelectDiverseSet of Algorithm 2.3 is applied by using the Hamming distance on the bit-vectors instead of the signature-based distance that has been used previously. We denote this algorithm as SkyDiver-LSH.

2.2.6 Experimental Evaluation

In this section, we report on the results of a comprehensive set of experiments, towards comparing the various techniques covered in the previous sections. First, we present the implemented algorithms as well as the data sets used.

2.2.6.1 Algorithms and Data Sets

We have implemented four different algorithms, presented in Table 2.3. We remind the reader that our proposed approaches, SkyDiver-MH and SkyDiver-LSH, can be applied regardless of having an index in place.

Table 2.3: Evaluated algorithms

Algorithm	Reference
Brute-Force (BF)	Brute-force algo. (Sec. 2.2.4)
Simple-Greedy (SG)	Simple greedy algo. (Sec. 2.2.4)
SkyDiver-MH (MH)	MinHash-based algo. (Sec. 2.2.5.1)
SkyDiver-LSH (LSH)	LSH-based algo. (Sec. 2.2.5.3)

We have generated synthetic data sets following the *independent* (IND) and *anticorrelated* (ANT) distributions, using the methodology presented in [42]. In addition, we have used two real-life data sets: *Forest Cover* (FC) downloaded from UCI Machine Learning Repository (http://kdd.ics.uci.edu) and *Recipes* (REC) [127] (Sparkrecipes.com), where each data point is a recipe and its attributes are the nutritional values for several common compounds, e.g., carbohydrates, protein, calcium etc. Table 2.4 summarizes the basic dataset characteristics, with default values underlined. For the real datasets, we select the first d dimensions to compare the techniques against dimensionality. The code was written in C++ and all experiments were run on a Quad-Core @3.5GHz machine, with 4Gb RAM, running Linux. Each data set was indexed by an aggregate R*-tree, with a 4Kb page size. An associated cache with 20% of the corresponding R*-tree's blocks was used with every experiment. Timings reported in the graphs are in seconds, measured as CPU processing time and assuming a default

Table 2.4: Basic data set characteristics

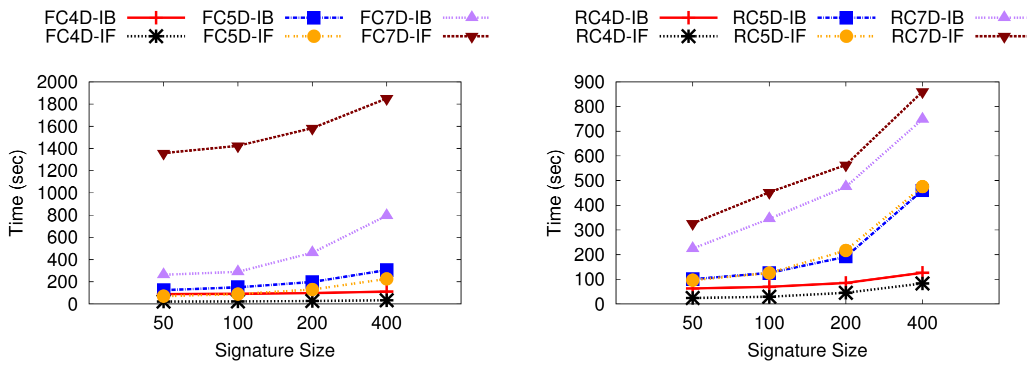
Data set	Cardinality	Dimensionality
Independent (IND)	1M, 2M, 5M, 7M	2, 3, 4, 6
Anticorrelated (ANT)	1M, 2M, 5M, 7M	2, 3, 4, 6
Forest Cover (FC)	$\sim$ 581K	4, 5, 7
Recipes (REC)	$\sim$ 365K	4, 5, 7

value of 8ms per page fault. Unless stated otherwise, all values reported below refer to the 2-step process of finding the k-most diverse skyline points, without the cost of finding the skyline itself as it does not affect the relative performance of the algorithms. Regarding effectiveness, we report the minimum (Jaccard) distance among the pair of points that has been selected by each approach.

2.2.6.2 Experiments and Results

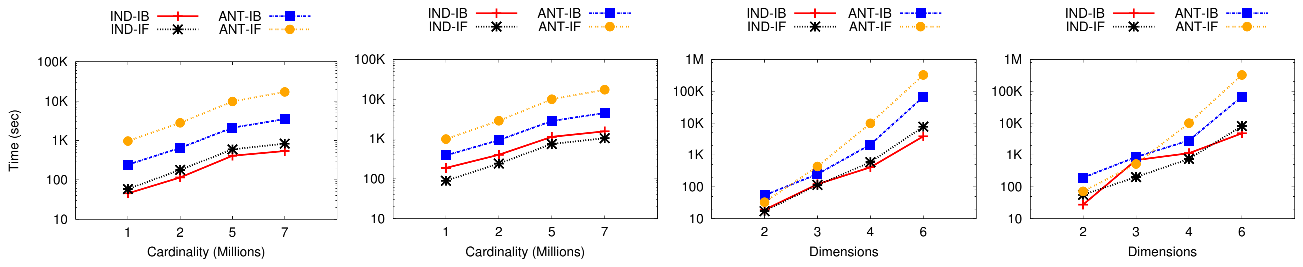
We begin by evaluating when signature creation should use an index (IB) or not (IF), in case we have such an option. We then evaluate the efficiency of all techniques compared to various parameters, using the IB approach, since BF and SG use the index as well. We finally report on result quality and memory consumption.

Signature Generation. Our first experiment focuses on the cost for signature generation and how the index affects this step. Figures 2.6(a)-



(a) FC (b) REC

Figure 2.6: Time for generating MinHash signatures vs signature size.



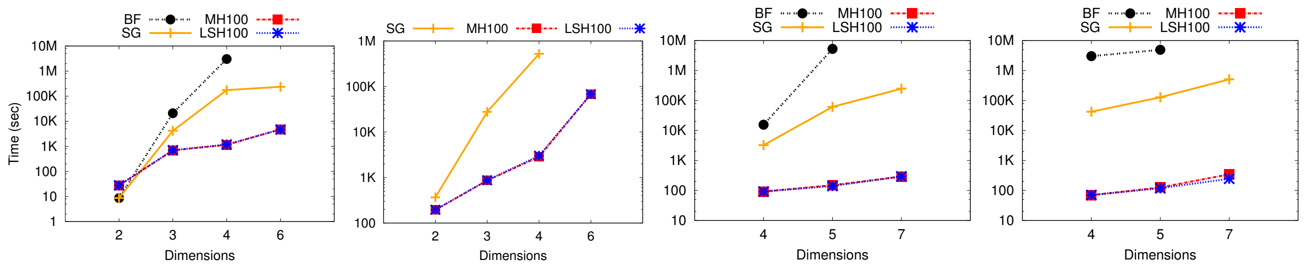
(a) CPU time (b) Total time (c) CPU time (d) Total time

Figure 2.7: Time for generating MinHash signatures of size 100 for synthetic data sets.

(b) show the signature generation time as a function of the signature size, for all dimensions of FC and REC data sets, respectively. Clearly, by increasing the signature size, the signature generation phase requires more time. Nevertheless, selecting IB or IF seems to be unrelated to signature size. We have obtained similar results for the IND and ANT distributions.

Figures 2.7(a)-(d) report the time taken to generate the signatures, depending on whether an index was used (IB - index based) or not (IF - index free), for varied cardinalities / dimensionalities of IND and ANT data sets. More specifically, figures 2.7(a) and 2.7(b) report CPU and total time -- I/O's included -- respectively, for varying cardinalities and default dimensionality $d = 4$. ANT data consistently favor the IB approach. However, for IND data, the IF method is more efficient when total time is concerned. On the contrary, when taking only CPU time into account, IB is better. This is due to a lot of I/Os on the R-tree, more than what a linear scan on the actual data set requires.

Even more interesting is the case when we vary the dimensionality, as shown in Figures 2.7(c) and 2.7(d). In particular, for ANT data sets, low dimensionality favors the IF approach. In this case, the costs are mostly due to I/Os. However, as dimensionality increases, more dominance checks are executed, which IF performs naïvely. On the other hand, IB saves on CPU costs, by utilizing the index. For IND data, IB and IF differ marginally for few dimensions and as d increases, IB is favored. For 2D, the R-tree saves several I/O operations, and the overall cost of IB is much lower. However, for average dimensionality, the I/O cost sharply increases for IB, making it less suitable. Basically, partial dominations of MBRs have dramatically increased, which necessitate that we decompose them further, resulting in additional I/Os.



(a) IND (b) ANT (c) FC (d) REC

Figure 2.8: Runtime for k = 10 diverse skyline points vs dimensionality.

Specifically, we observe an $\sim 70\times$ increase in the number of I/Os from 2D to 3D, but the I/O increase is niche as d increases from that point onwards. A big part of the R-tree has to be traversed when $d \geq 3$, yet several dominance checks are saved, explaining why CPU-costs do not follow this trend. Given the efficiency when the signature size is set to 100 and the fact that we achieve good quality (Figure 2.10), as we discuss in the next paragraphs, we adopt it as the default signature size for the rest of our evaluation.

User Guide. We propose the following scheme which is experimentally validated: The IB method should be considered: i) when the R-tree can be memory resident, assuming enough resources, whereas for a disk-resident index ii) for average and high-dimensional data ($d \geq 4$) and iii) when $d = 2$, provided we are dealing with IND data. In the few remaining cases, IF should be favored.

Runtime VS Dimensionality. We now turn our attention to the efficiency of the techniques for selecting the k-most diverse skyline points. Figures 2.8(a)-(d) demonstrate the performance of the employed algorithms on all data sets, for varying dimensionalities. In particular, we have plotted the overall time taken to compute the 10-most diverse skyline points, including the time for generating the signatures (for the cases of MH and LSH).

As expected, BF shows the worst performance, given that it searches exhaustively for the optimal solution. By increasing the dimensionality, the number of skyline points increases too and, consequently, so does the number of $\binom{m}{k}$ enumerations. Moreover, unlike the other techniques, BF's reported times are for $k = 2$; $k = 10$ yields even more enumerations, since the skyline contains a few hundred points

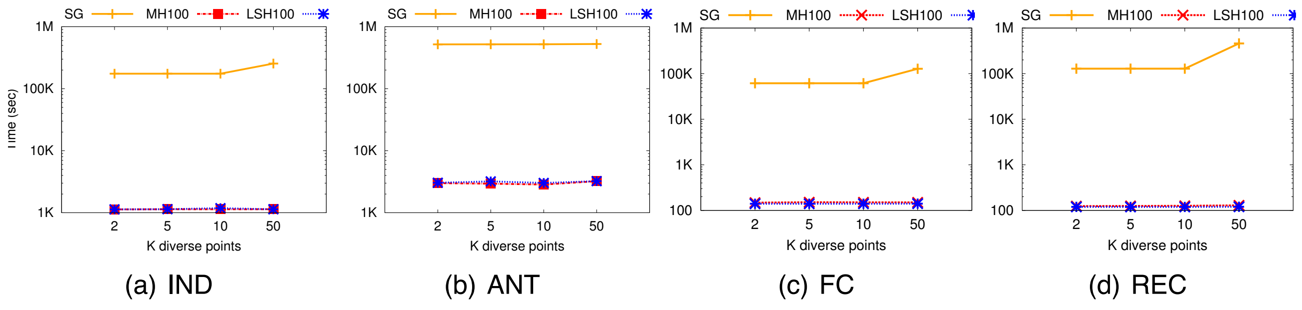
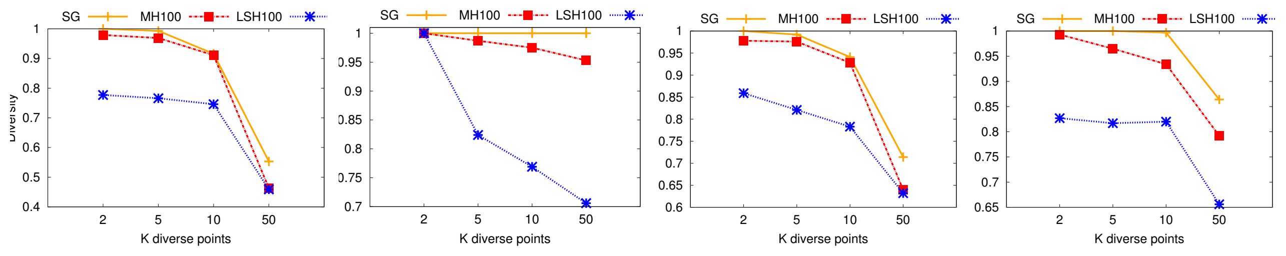


Figure 2.9: Runtime vs number of diverse points (k).

at best. We even ran BF for $k = 5$, but none of the experiments had concluded within a 10 day period of execution (in wall clock time). Not surprisingly, BF is inappropriate for practical applications, even when dealing with small skyline sets and reasonable values of k. For this reason, we omit it from subsequent experiments.

The greedy algorithm SG, which computes the actual diversity scores by performing range queries, is inferior to MH and LSH by, approximately 2-3 orders of magnitude. Note that we have boosted SG, by maintaining in-memory the minimum distance of each non-selected skyline point. Even so, most of the time is spent on I/Os due to range queries for Jaccard distance computations, whereas CPU cost is only a fraction. This validates our goal to keep range queries to a minimum. SG performs better only for the IND data set and $d = 2$, where there are very few skyline points ($\sim 5 - 10$) and signature creation phase places enough overhead to make the signature-based techniques slightly worse. In all other occasions, SG's performance is worse; in fact it did not complete for the anticorrelated dataset with 6 dimensions (ANT 6D setting)8. Finally, though MH and LSH differ slightly, at this granularity their difference is not discernible.

Runtime VS Number of Points (k). Figures 2.9(a)-(d) portray the efficiency of the techniques with respect to the number of requested points. The graphs clearly support our earlier findings that MH and LSH are superior to SG by orders of magnitude, for reasonable values of k. All three algorithms exhibit a consistent behavior in all data sets and k values: MH and LSH perform almost identically for all k values, with LSH being slightly better as shown in Figure 2.9(c)-(d), which is one of the main reasons to consider it over MH. CPU costs are



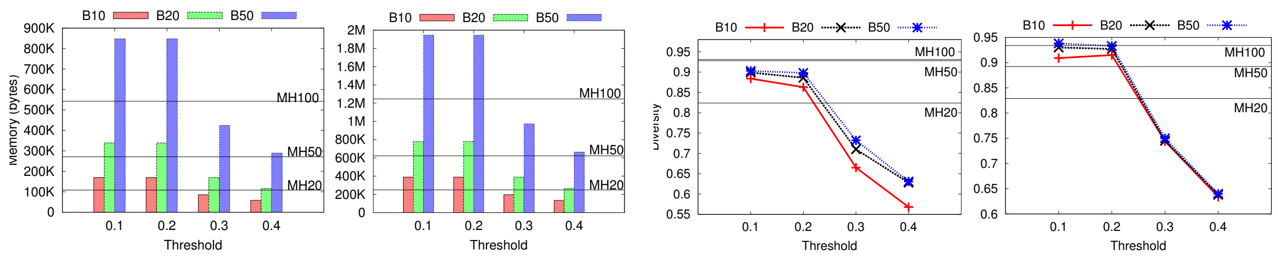
(a) IND (b) ANT (c) FC (d) REC

Figure 2.10: Quality vs number of diverse points (k).

minimal for these techniques, accounting for no more than 45 sec for ANT, and at most 2 seconds for the other data sets, with k=50 and default values for the other parameters. On the other hand, for all k values, SG is burdened with an excessive number of I/Os, due to range queries, despite being boosted. The technique also shows a noticeable increase in runtime for k=50, across all data sets, as a result of increased CPU costs. This is because when increasing k, the pair-wise Jaccard distance computations add-up to a more noticeable amount, given that range queries require O(d) checks, and recursively descend the R-tree if needed, to compute the intersection.

Quality of Results. We now turn our attention to the effectiveness aspect of our approach. Figures 2.10(a)-(d) demonstrate the diversity score, i.e., the minimum Jaccard distance in the original space, of the selected set of skyline points, for different values of k (number of selected points). As expected, by increasing k, the minimum Jaccard distance is reduced. SG performs better than MH and LSH in general, however, the latter two achieve very good performance, given their efficiency savings. With the exception of REC data set, MH is only slightly worse than SG for k values up to 10. In constrast, LSH has a steeper decline in the diversity score, but requires less memory, as shown in Figure 2.11(a)-(b) and explained in the sequel.

MinHashing VS LSH. Figure 2.11 depicts a comparison between MH and LSH, demonstrating the memory vs accuracy trade-off. In particular, we have performed a series of experiments with a fixed k value (k=10), while varying the parameters ξ (threshold) and B (number of buckets per zone) for LSH, and (varying) the signature size for MinHash. By increasing ξ, the number of zones ζ is reduced, which increases memory savings. In addition, maintaining fewer buckets per



(a) FC - Memory (b) REC - Memory (c) FC - Quality (d) REC - Quality

Figure 2.11: LSH vs MinHashing, for k = 10 diverse skyline points, with signature size fixed to 100

zone reduces memory consumption further. The price we pay in this case is a drop in accuracy. As expected, the accuracy of LSH is lower than that of MH as shown in Figure 2.11(c)-(d), whereas the savings in storage are more sensitive to the value of ξ, due to the high correlation between ξ and ζ as shown in Figure 2.11(a)-(b). For example, by using LSH with ξ = 0.2 and B=20, we need around 300Kb for the FC data set, whereas MH requires almost 600Kb. Moreover, the corresponding diversity score obtained by LSH is 0.88 when MH performs marginally better obtaining a diversity score of 0.93. Overall, the significant reduction in memory requirements make LSH a very attractive alternative, in cases where we are willing to sacrifice accuracy, up to an acceptable level.

Another key observation is that by simply reducing the signature size in MinHashing does not give promising results. For example, using a threshold of 0.2, with 10 buckets per zone, LSH obtains results of similar or better quality, while requiring less memory than MH50. In general, the accuracy of MinHashing drops rapidly by decreasing the signature size, whereas by carefully controlling the threshold and the number of buckets per zone, LSH can be effectively tuned.

Comparing against contour representation. As we have discussed in earlier paragraphs, contour representation has been suggested as an alternative for skyline diversification, where the measure of diversity is given by the Euclidean distance (L_2) of the skyline points. Under the Euclidean distance, the problem is still formally defined as an instance of k-MMDP.

Despite the disadvantages of these techniques, which we have outlined in the previous paragraphs (e.g., inapplicable in certain domains,

Table 2.5: Comparing SkyDiver and Contour Representation techniques, Independent Dataset

		SkyDiver			**Contour Representation**		
Dataset	**k**	*Coverage*	*Jaccard*	L_p *distance*	*Coverage*	*Jaccard*	L_p *distance*
Indep n=5M d=2	2	99.93%	0.0352	0.0353	99.93%	0.0352	0.0353
	5	100%	8.2x10^{-4}	8.3x10^{-4}	100%	8.2x10^{-4}	8.3x10^{-4}
	10	100%	8.2x10^{-4}	8.3x10^{-4}	100%	8.2x10^{-4}	8.3x10^{-4}
	25	100%	8.2x10^{-4}	8.3x10^{-4}	100%	8.2x10^{-4}	8.3x10^{-4}
	50	100%	8.2x10^{-4}	8.3x10^{-4}	100%	8.2x10^{-4}	8.3x10^{-4}
Indep n=5M d=3	2	99.12%	0.9927	1.0022	99.12%	0.9927	1.0022
	5	99.20%	**0.8426**	0.1861	**99.34**%	0.5362	**0.4855**
	10	99.55%	**0.5634**	0.0470	**99.72**%	0.2863	**0.2626**
	25	99.86%	**0.2262**	0.0470	**99.93**%	0.1484	**0.1223**
	50	99.96%	**0.1004**	0.0312	**99.97**%	0.0810	**0.0603**
Indep n=5M d=4	2	95.14%	**0.9998**	1.2568	95.14%	0.9982	**1.3300**
	5	95.17%	**0.9931**	0.9887	**95.18**%	0.9692	**1.1080**
	10	95.30%	**0.9392**	0.1088	**96.13**%	0.7555	**0.6934**
	25	96.51%	**0.7269**	0.1088	**97.70**%	0.5431	**0.4106**
	50	97.83%	**0.5716**	0.0309	**99.01**%	0.3464	**0.2617**
Indep n=5M d=5	2	85.20%	0.9999	1.6167	85.20%	0.9999	1.6167
	5	85.21%	**0.9990**	0.9039	**85.24**%	0.9914	**1.3525**
	10	85.26%	**0.9857**	0.6747	**85.54**%	0.9083	**0.9855**
	25	85.70%	**0.9355**	0.2138	**88.35**%	0.7950	**0.6511**
	50	87.89%	**0.8644**	0.1170	**93.26**%	0.5647	**0.4957**

mandatory requirement of an index), it would be interesting to see how they compare against our proposed approach. Note that the comparison is only meaningful when both approaches are applicable, i.e., numerical attributes indexed by an R-tree.

Given that contour representation only considers the skyline, thereby using considerably less information, a comparison in terms of efficiency will not be particularly helpful. Therefore, our focus will be on the quality of the produced solutions, namely *Coverage* of the dataset, minimum *Jaccard* distance of dominated sets and minimum *Euclidean distance* of the selected skyline points. For this comparison, SkyDiver employs the Simple Greedy approach, using the exact computation of Jaccard distances. Recall that this is the upper bound of quality we can achieve, and we can tune SkyDiver-MH or SkyDiver-LSH appropriately to reach these values.

Tables 2.5 and 2.6 summarize the result of two such experiments, for an independent dataset with 5 million points and for the forest cover dataset, respectively. In both cases, we have varied the dimensionality of the dataset. Higher values are always preferred, and we have marked in bold the best value in each case. In case of a tie for a quality measure, none of the values are emphasized.

For the independent case, we observe that in 2-dimensional data, the two techniques are tied in all used measures. In fact, for $k \geq 5$, we observe that none of the techniques improve, which is expected, given that the skyline only contains 5 points in this case.

Through careful inspection of Table 2.5 we observe that SkyDiver is always better in terms of the Jaccard distance, whereas Contour Representation is better in L_p distance. This means that either technique performs the best for the objective function that it optimizes, which is hardly surprising. What is interesting, however, for the independent dataset is that contour representation is also better in terms of coverage. Therefore, although the selected skyline points are far from each other in the Euclidean space, they clearly correlate with coverage as an objective. Recall that we have already shown that coverage is different from diversification, a result validated again with this experiment, as diversity drops considerably with Contour Representation.

Since the independent dataset is a synthetic one, we would like to draw some conclusions for a real dataset as well. For this reason, we have performed the same experiment with the Forest Cover dataset, and the results are given in Table 2.6. As with the independent dataset, Contour Representation is better at L_p distance, whereas SkyDiver is better at Jaccard.

Of particular interest is the fact that SkyDiver is sometimes also better at coverage by a large margin (much larger than in the independent case). For example, for $d = 4$ and $k = 2$, SkyDiver is better at coverage over 5%, while for $k = 5$ by slightly less than 4%. For $d = 5$ and similar k values, SkyDiver achieves a better performance by 4%, which drops to 3% for higher k values (same d). Meanwhile, for the independent dataset, Contour was better by 5.5% in a single case ($d = 5$, $k = 50$), whereas it was never better by more than 3% in all other occasions (and less than 1% in many of them).

It is also interesting that Contour achieves better coverage for higher

Table 2.6: Comparing SkyDiver and Contour Representation techniques, Forest Cover

		SkyDiver			**Contour Representation**		
Dataset	**k**	*Coverage*	*Jaccard*	L_p *distance*	*Coverage*	*Jaccard*	L_p *distance*
FC d=2	2	**99.99**%	**0.0729**	**0.0601**	98.53%	0.0589	0.0446
	5	99.99%	**0.0127**	**0.0115**	99.99%	0.0062	0.0081
	10	100%	0.0029	0.0028	100%	0.0029	0.0028
	25	100%	0.0029	0.0028	100%	0.0029	0.0028
	50	100%	0.0029	0.0028	100%	0.0029	0.0028
FC d=3	2	**98.65**%	**0.9592**	0.4515	93.52%	0.8803	**0.8811**
	5	**99.60**%	**0.7110**	0.1558	95.30%	0.4787	**0.2917**
	10	99.80%	**0.3640**	0.0682	**99.90**%	0.1012	**0.1260**
	25	99.93%	**0.0461**	0.0253	**99.99**%	0.0164	**0.0378**
	50	100%	0.0029	0.0028	100%	0.0029	0.0028
FC d=4	2	**98.64**%	**0.9595**	0.4520	93.20%	0.9518	**0.9249**
	5	**99.38**%	**0.7430**	0.1748	95.65%	0.4175	**0.3181**
	10	99.58%	**0.4787**	0.0727	**99.88**%	0.1133	**0.1279**
	25	99.91%	**0.1904**	0.0365	**99.99**%	0.0554	**0.0581**
	50	100%	0.0031	0.0028	100%	0.0031	0.0028
FC d=5	2	**88.61**%	**0.9995**	0.9308	84.49%	0.9991	**1.2534**
	5	**88.67**%	**0.9918**	0.5255	84.78%	0.9511	**0.7740**
	10	**88.90**%	**0.9411**	0.3245	87.12%	0.7433	**0.4470**
	25	**90.76**%	**0.8320**	0.1353	87.82%	0.6535	**0.3075**
	50	93.19%	**0.7135**	0.0894	**95.62**%	0.2447	**0.2249**
FC d=6	2	84.67%	0.9999	1.1226	84.67%	0.9999	**1.3434**
	5	84.67%	**0.9999**	0.7711	**85.16**%	0.9595	**0.8954**
	10	84.70%	**0.9977**	0.3088	**86.55**%	0.7857	**0.6327**
	25	85.47%	**0.9061**	0.2572	**88.72**%	0.6734	**0.4041**
	50	86.99%	**0.8231**	0.0827	**91.44**%	0.4386	**0.3007**

values of k (25, 50), and medium dimensionality (3 and 4). The reason is that in these cases, the skyline contains less than 50 points. Therefore, the 25 selected points are more than half the skyline. Note that SkyDiver manages to be much better in terms of the Jaccard distance (though decreasing), even for half the skyline. On the other hand, Contour increases its coverage by at least 7% (compared with less than 5% for SkyDiver) and decreases its Jaccard score substantially. This fact validates the claim that Contour selects points which generally correlate with coverage, despite being far in the Euclidean space.

Finally, when $d \geq 5$, a lot more points belong to the skyline (≥ 1300). Much like in the previous cases, for small k values (compared with the skyline size) SkyDiver is better at coverage as well, but Contour performs better as k grows larger. For $k = 6$, where more points belong to the skyline ($|\mathcal{S}| = 2728$), Contour is better at coverage for as low as $k = 3$, since there are a lot more points to select from. Even so, the technique is still worse regarding the Jaccard distance, which drops to less than half for $k = 50$.
Consequently, in addition to the disadvantages of the Contour Representation approach we have talked about, we have experimentally shown that this technique favors coverage as a goal, and not diversity. In other words, this technique solves a different problem from the one that we proposed in this thesis.

2.3 Skyline ranking with IR techniques

2.3.1 Introduction

In Section 2.2 we presented an approach to address the skyline cardinality explosion problem by diversifying the skyline set. Meanwhile, we briefly discussed some alternatives for tackling the same problem. The common denominator of these techniques is to return a subset of k skyline points, where k is a user- or application-defined parameter. The subset has some specific properties, e.g., collectively maximizes coverage [136], captures the contour of the skyline [200], diversifies the skyline [209, 200], etc. Nevertheless, these techniques generally fail to differentiate between the returned points in terms of some qualitative aspect. Moreover, they are mapped to NP-Hard problems, so we can only efficiently approximate the solutions, unless P=NP.
As also discussed in the beginning of this chapter, top-k queries are another technique to control the result size. Top-k queries require a function $f() : \mathcal{D} \rightarrow \mathbb{R}$, mapping data points to a real value, which can be used to fully rank the dataset, and researchers have also investigated ranking of the skyline set.
We identify two categories, depending on the amount of information used to rank the points: In the first case, the entire dataset is used,

and the importance of a skyline point is given by the number of points it dominates [165, 231]. The major shortcoming of this category is that dominated points are equally important. For example, a point p_1 dominated by 10 skyline points and another one p_2 dominated by 100 contribute the same weight to their dominators. Considering that skyline queries have an inherent relation to sorting [38], and that distance measures for sorted lists heavily rely on the relative positions of items, it feels counter-intuitive to use the same weight for all dominated points.

The second category relies on the skyline $\mathcal{S}$ alone, and typically uses dominance relations in all possible subspaces [215, 51]. As a result, such techniques ignore the dataset characteristics, except for the skyline. They are also generally inefficient, as they need to consider O(2^d) non-empty subspaces. Moreover, they favor skyline points with extreme values in a single dimension and have been shown to produce correlated results, whereas some of them [215] have not been sufficiently evaluated.

To address these shortcomings, we present a novel ranking scheme for skyline points. The importance of a skyline point sp is given by aggregating the importance scores of its dominated set, $\Gamma(sp)$. We argue that the importance of a dominated point should be affected by the number of skyline points that dominate it, as well as the relative position of that point in the dominated set.

To that end, we propose that the importance of a dominated point is inversely proportional to the number of skyline points that dominate it. Regarding the relative positions of dominated points, we propose that points in the same (different) layer of minima with respect to the sp should contribute equally (differently).

For instance, if $sp_1 \prec a$, $sp_1 \prec b$, and a and b do not dominate each other, they contribute equally to sp_1. Otherwise, if $a \prec b$, then $score(a) > score(b)$. Note that the factor regarding the relative position of a dominated point p has a more local taste (i.e., per skyline point), whereas the number of skyline points dominating p is a more global aspect. Therefore, our technique promotes skyline points that dominate *genuine* points, i.e., points which are not dominated by many others, and is a hybrid approach.

To capture both aspects in a single scoring function, we apply a modified version of the renowned *Term Frequency-Inverse Document Frequency* (TF-IDF) weighting scheme and present efficient algorithms that derive the top-k skyline points according to it. Our contributions are briefly described as follows:

- We define a novel, generic and intuitive measure of importance for skyline points. Inspired by the renowned *tf-idf* weighting scheme from information retrieval, our method promotes skyline points that dominate *genuine* points.
- We present efficient algorithms that compute the top-k most important skyline points, given our measure.
- We provide an extensive experimental evaluation, in terms of efficiency using both real-life and synthetic datasets.

The rest of this Section is organized as follows. Section 2.3.2 gives the details of our scoring model, followed by Section 2.3.3 where we present the algorithms to derive the top-k result. Section 2.3.4 contains the experimental evaluation of our proposed techniques.

2.3.2 DP-IDP weighting scheme

Our proposed measure, *dp-idp*, which stands for *Dominance Power - Inverse Dominance Power*, is inspired by the renowned *tf-idf* weighting scheme from Information Retrieval. The general rationale is that dominated points are not equally important, and that they impact skyline points differently. Therefore, their contribution depends on some local (per skyline point) and some global characteristics (the entire skyline), much like *tf-idf* uses local and global information to find important keywords in a document corpus. In the following paragraphs we present our ranking scheme.

2.3.2.1 Inverse Dominance Power

We will start with *inverse dominance power* (idp), which is easier to define, due to its more global view. The *inverse dominance power* of a point $p \in (\mathcal{D} \setminus \mathcal{S})$ is the number of skyline points which dominate

p. This factor is similar to *idf* in the sense that the more frequently p appears in a skyline point's dominated set, the lower the importance of p. More formally:

$$idp(p) = \log \frac{|\mathcal{S}|}{|\{sp \in \mathcal{S} : sp \prec p\}|}$$

An interesting property of idp is the following: Assume a set of points $q_1, q_2, ..., q_m$ dominated by all skyline points, i.e., $\forall sp \in \mathcal{S}$, $sp \prec q_j$, $j = 1, .., m$. The contributing score of the q_i's will be 0, due to the log in the idp factor. Such points do not alter the ranking of $\mathcal{S}$, either with ours or simpler models (e.g., $|\Gamma(sp)|$), because they affect all skyline points the same.

2.3.2.2 Dominance Power

There are several ways we could define the *dominance power* of a dominated point. Given that we want to measure this factor with respect to a skyline point sp, we argue that its relative position to sp should matter. As a result, the same dominated point may contribute differently to different skyline points.

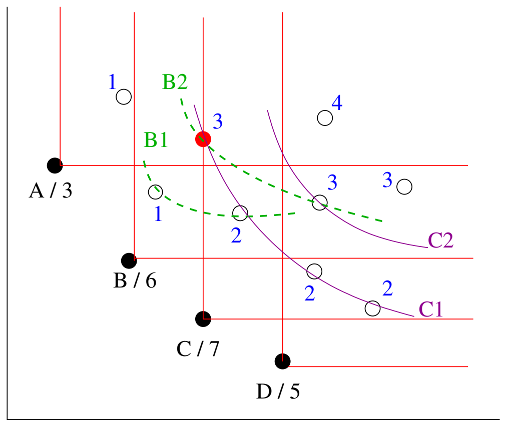


Figure 2.12: Example for the Dominance Power

To avoid introducing more artifacts in our model, we choose the dominance relation as our building block. More specifically, we find the *layer of minima* [4] $lm(p, sp)$ where the dominated point p falls in, with respect to sp. The dominance power of that point is then given by the inverse of the layer where it lies, i.e.,

$$dp(p, sp) = \frac{1}{lm(p, sp)}$$

Figure 2.12 portrays the skyline of a dataset as black filled points, and the dominance region for each skyline point. Moreover, it shows the layers of minima for skyline points B and C, in dotted green and purple respectively. We observe that the red-filled point, dominated by both B and C, lies at different layers of minima. Being easier to reach it from C, should render it more important for C. On the contrary, there is an additional layer for skyline point B prior to reaching that point, that decreases its importance. This is similar to *term frequency*, where the same term is weighted differently, depending on its occurrence in each document.

2.3.2.3 Putting it all together

Given our previous discussion, we can now formally introduce how we compute the importance of a skyline point sp. We use an additive model, because i) it is monotonous (dominating more points increases the overall importance) ii) it is comprehensive and iii) it leads to efficient computations, as we followingly discuss. Therefore, the importance of a skyline point sp is given by:

$$score(sp) = \sum_{p: sp \prec p} \frac{1}{lm(p, sp)} \times \log \frac{|\mathcal{S}|}{|\{sp' \in \mathcal{S} : sp' \prec p\}|}$$

The additive model also favors skyline points that dominate more *genuine* points, i.e., points dominated by few others in general (not just skyline points). This is important, because those skyline points are the reason why the dominated ones cannot be part of the skyline. For example, if we remove B from the skyline in Figure 2.12 (e.g., a hotel

[4] In the literature, the term *layer of maxima* is more common. Here, we use the term *layer of minima* because we assume that smaller values are preferred.

Algorithm 2.4 Baseline

```
   Input: Skyline $\mathcal{S}$, Dataset $\mathcal{D}$, Integer $k$
   Output: Top-$k$ skyline points with highest score
1: for every $sp \in \mathcal{S}$ do
2:     score( $sp$ ) $\leftarrow$ 0; layer $\leftarrow$ 1; $lm \leftarrow$ NextLayer( $sp$, $\emptyset$ );
3:     while ($lm \neq \emptyset$) do
4:         for every $p \in lm$ do
5:             score( $sp$ ) += $\frac{1}{layer} \times log\frac{|\mathcal{S}|}{|\{sp' : sp' \prec p\}|}$;
6:         $lm \leftarrow$ NextLayer( $sp_i$, $lm$ ); layer++;
7: Order by descending score($sp$);
8: Return $k$ highest skyline points;
```

being fully booked), the point next to it will enter the skyline at once (similarly for the closest point dominated by A). Additionally, points dominated by the entire skyline still have no effect. Finally, note that the sum of $tf - idf$ values is also used in IR systems, to score the entire document against a query.

2.3.3 Ranking the Skyline

Algorithm 2.4 gives the baseline approach to rank the skyline $\mathcal{S}$ with our proposed scheme. For each skyline point sp (line 1), we extract one-by-one its layers of minima(lines 2--6). $NextLayer$ uses BBS [165] internally. For every point in each layer (line 4), we find how many skyline points in $\mathcal{S}$ dominate it, and together with the layer's index, we update the score of sp (line 5). After ordering the skyline in decreasing score order (line 7), we return the top-k ranked items.

Unfortunately, this approach is computationally expensive, due to repeated evaluations. It does not perform any pruning, but computes the exact score of all skyline points, despite our interest in the top-k results alone. Finally, it lacks any notion of *progressiveness*, as we need to rank the entire skyline first. For all these reasons, we present an alternative approach, that relies on bounding the score of a skyline point.

2.3.3.1 Bounding the score

Bounding the score of a skyline point sp will help us reduce computations, by pruning away those that will not make it to the top-k positions. To achieve this, we use the number of points that sp dominates, $|\Gamma(sp)|$. We can then derive lower and upper bounds of the score of a skyline point, as shown in the next paragraphs.

Loose Bounds. In the simplest case, we can consider each skyline point independently of the others. In that case, bounds are derived as follows. A skyline point obtains its maximum score when *all* the points it dominates are in the same (first) layer, and they are *not* dominated by any other skyline point. In that case, the upper bound is:

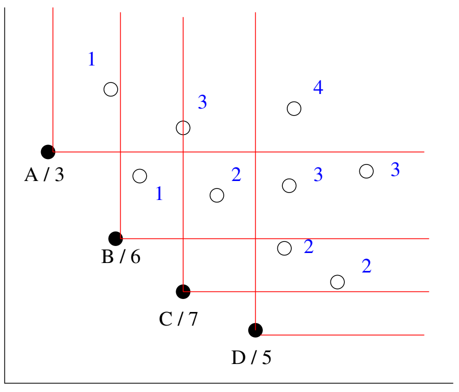
$$\overline{score}(sp) = |\Gamma(sp)| \times \log |\mathcal{S}|$$

On the other hand, the lower bound is obtained when every point is dominated by the entire skyline $\mathcal{S}$. In that case, the score is 0, due to the $idp(sp)$ factor. However, this bound only holds for the skyline point sp_{min} with the minimum $|\Gamma(sp_{min})|$, unless, of course, every point in the skyline dominates every dominated point. The rest of the skyline dominates some points, which can not be dominated by sp_{min}. Consider, for instance, that $|\Gamma(sp_{min})| = 3$ and that $|\Gamma(sp')| = 5$. By definition, the 2 additional points dominated by sp' can not be dominated by sp_{min}, otherwise $|\Gamma(sp_{min})| = 5$. Therefore, the surplus will be dominated by $|\mathcal{S}| - 1$ skyline points, and a correlated distribution will give the lowest score value. In a correlated distribution, each point is a layer of minima on its own, which reduces the dp factor. This improves slightly the lower bound:

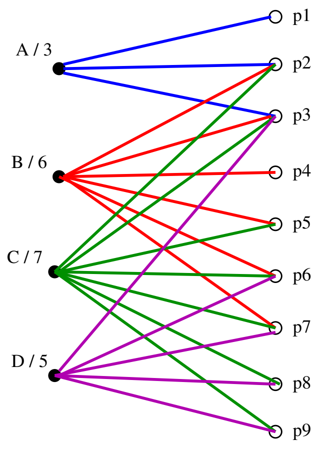
$$\underline{score}(sp) = \log \frac{|\mathcal{S}|}{|\mathcal{S}| - 1} \times \sum_{1}^{n-min_{\Gamma}} \frac{1}{i}$$

Collaborative Bounds. Despite their simplicity, the above bounds have limited pruning capability. Assume, for instance, a dataset $\mathcal{D}$, with $|\mathcal{D}| = 1M$ and $|\mathcal{S}| = 800$. If $|\Gamma(sp)| = 300K$, then $\overline{score}(sp) \simeq 871K$, and $\underline{score}(sp) \simeq 3 \times 10^{-3}$. Note that a skyline point sp' with

$|\Gamma(sp')| = 1$, has an upper bound of ~2.9, making it eligible for consideration in the second round! Apparrently, the computational gains of such bounds are easily swept away.
To address this issue, we derive stricter bounds, through additional information from other skyline points. To better illustrate this approach, we visualize the problem as a bipartite graph. Figure 2.13 shows a dataset, with its skyline and dominance regions on the left, and the resulting bipartite graph on the right. The bipartite graph has the same semantics as the one in Fig. 2.2. The left hand side of the graph contains the skyline, whereas the right hand side has the dominated points. There is an edge between a skyline point $sp \in \mathcal{S}$ and a dominated point $p \in \mathcal{D} \setminus \mathcal{S}$, iff $sp \prec p$.
We start with the upper bound. Due to the additive model, the score of a skyline point is maximized when the contribution of each dominated point is maximized. It is easy to see that dp is maximized when the dominated point is at the earliest possible layer. To maximize idp, we rely on the Pigeonhole Principle. For any two skyline points sp_1, sp_2, if $\Gamma(sp_1) + \Gamma(sp_2) > |\mathcal{D}|$, then at least $|\mathcal{D}| - (\Gamma(sp_1) + \Gamma(sp_2))$ dominated points are shared by sp_1 and sp_2. Dominating more common points reduces $idp()$, so this factor is maximized when the overlap is minimized. Given this information, the question now becomes "*How should we assign the common edges to maximize the score of a skyline point*"? Lemma 5 answers this question.



(a) Motivating Example



(b) Bipartite Form

Figure 2.13: Example of skyline and bipartite domination graph

Lemma 5 *Let sp be a skyline point, p_x and p_y two dominated points, where $sp \prec p_x$ and $sp \prec p_y$ and $lm(p_x) = lm(p_y) = l$. Assigning an edge to the point dominated by more skyline points gives a higher $score(sp)$.*

Proof 4 *Let $\mathcal{S}_x$, $\mathcal{S}_y$ be the current sets of skyline points dominating p_x and p_y, respectively. Assigning an edge to either p_x or p_y gives two different bipartite graphs, with $\mathcal{S}'_x$ and $\mathcal{S}'_y$ being the new dominating sets of these points. It holds that $|\mathcal{S}'_x| = |\mathcal{S}_x| + 1$ (same for $\mathcal{S}'_y$), due to the new edge, i.e., one more dominating skyline point. The resulting bipartite graphs differ only in the assignment of this edge, which impacts the weights of p_x and p_y. The weights of all other dominated points remain unchanged. Assume that adding the edge to p_x yields a higher score. It so holds:*

$$\frac{1}{l} \times \log\frac{|\mathcal{S}|}{|\mathcal{S}'_x|} + \frac{1}{l} \times \log\frac{|\mathcal{S}|}{|\mathcal{S}_y|} > \frac{1}{l} \times \log\frac{|\mathcal{S}|}{|\mathcal{S}_x|} + \frac{1}{l} \times \log\frac{|\mathcal{S}|}{|\mathcal{S}'_y|} \Rightarrow$$

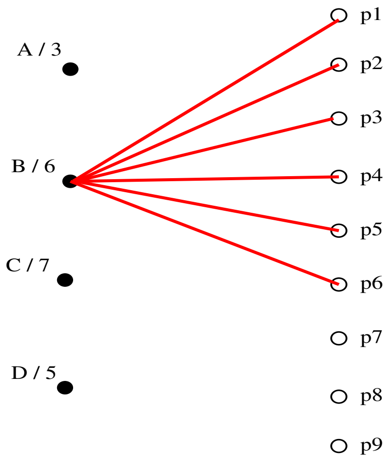
$$\log(\frac{|\mathcal{S}|}{|\mathcal{S}'_x|} \times \frac{|\mathcal{S}|}{|\mathcal{S}_y|}) > \log(\frac{|\mathcal{S}|}{|\mathcal{S}_x|} \times \frac{|\mathcal{S}|}{|\mathcal{S}'_y|}) \Rightarrow |\mathcal{S}_x| \cdot |\mathcal{S}'_y| > |\mathcal{S}'_x| \cdot |\mathcal{S}_y| \Rightarrow$$

$$|\mathcal{S}_x| \cdot (|\mathcal{S}_y| + 1) > (|\mathcal{S}_x| + 1) \cdot |\mathcal{S}_y| \Rightarrow |\mathcal{S}_x| > |\mathcal{S}_y|$$

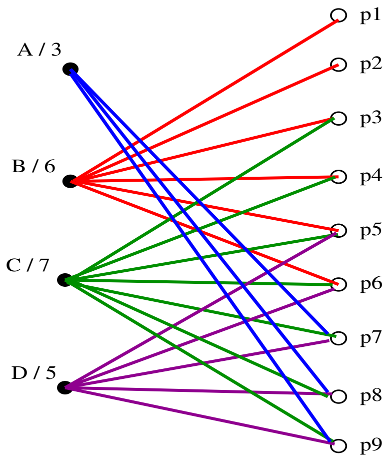
The above result tells us that a higher score is achieved by adding the extra edge to the dominated point that currently has the highest indegree. Such a result can also be efficiently integrated in an algorithm to compute the upper bound of a skyline point's score. Figure 2.14(b) shows the edge assignment for the upper bound of skyline points B, using the above result.

A naive implementation of the upper bound can be very inefficient [5], because it requires too many counter updates for the commonly dominated points. Since we must compute the bound of each skyline point sp independently and repeatedly, as new layers are extracted, we need a more efficient approach. Algorithm 2.5 presents this improved technique.

[5] Our experiments showed that this step alone can make up for up to 10 seconds of CPU processing time, for the datasets that we consider in our experiments.



(a) Dominance set



(b) Upper Bound for B

Figure 2.14: Collaborative upper bound

The improved algorithm takes as input the dataset $\mathcal{D}$, the skyline $\mathcal{S}$, the skyline point of interest poi, and two values $minIDP$ and $layer$. As we extract more layers for poi, we must compute the number of skyline points dominating each of the extracted points, which we need for the idp value. The minimum value that we have seen this far is stored in $minIDP$, and is different for each skyline point. This value practically tells us that any point in subsequent layers will be domi-

Algorithm 2.5 Score Upper Bounding

```
    Input: $\mathcal{D}$, $\mathcal{S}$, Skyline Point $poi$, $minIDP$, $layer$
    Output: Upper bound of $poi$
 1: $v_{idp}$.push( $minIDP$ ); $v_{pnd}$.push( pending( $poi$ ) );
 2: Sort $\mathcal{S}$, in decreasing $|\Gamma(sp)|$;
 3: for every $sp \in \mathcal{S}$, $sp \neq poi$ do
 4:     surplus = $|\Gamma(poi)|$ - seen( $poi$ ) + $|\Gamma(sp)|$ > $|\mathcal{D}|$
 5:     if ( surplus > 0 ) then
 6:         $v_{pnd}$[last] = $v_{pnd}$[last] - surplus;
 7:         $v_{pnd}$.push( surplus );
 8:         $v_{idp}$.push( $v_{idp}$[last] + 1 );
 9: $ub \leftarrow 0$;
10: for ( $i$ = 0; $i$ < $v_{idp}$.size(); $i$++ ) do
11:     $ub \leftarrow \frac{1}{layer+1} * v_{pnd}[i] * \log \frac{|\mathcal{S}|}{v_{idp}[i]}$
12: Return $ub$;
```

nated by *at least* $minIDP$ skyline points, due to dominance being a transitive relation. The $layer$ tells us which was the index of the last layer of minima extracted for poi.

The algorithm uses two vectors, storing the $minIDP$ value and the number of *unseen* points, that have not yet been extracted for poi (line 1). For example, if $|\Gamma(poi)|$ = 100, and we have extracted 30 points, then $unseen(poi)$ = 70. In simple terms, the vectors store, in aggregate, how many points (v_{PND}) can be dominated by that many skyline points (v_{IDP}). We sort the skyline points in decreasing order of their dominance power (line 2). We iterate over them (line 3), and select those points that will share common edges with poi, using the Pigeonhole Principle (lines 4--5). The surplus of points is removed from the last position (line 6) and is appended, incremented by 1 (lines 7--8). With these values, we can compute the upper bound according to the DP-IDP scheme (lines 10-11).

To better explain lines 5--7, assume $v_{pnd}[last]$ = 60, $v_{idp}[last]$ = 4, and $surplus$ = 25. This means that 60 points will be dominated by 4 skyline points and the current sp will share at least 25 dominated points with poi. As a result, we must add an edge (i.e., increment the idp) for an equal number of unseen points from poi. These must be selected from the points with maximum current idp, due to Lemma 5. Processing the skyline in decreasing order of dominance power ensures that we are properly assigning edges, and that the maximum idp is in the last posititION. Given these values, 25 points will be computed with an idp of 5, which we append, whereas 60-25=35 will remain with an idp of 4, which we update.

For the lower bounds we could follow a similar reasoning. Unfortunately, the edge assignment problem in this case is not as straightforward. Although certain properties are self-evident, e.g., dp decreases with a correlated distribution, they do not necessarily result in the lowest possible score for a point. The reason is that a correlated distribution enforces certain constraints with respect to the dominance relations among points in the skyline. For example, in a correlated distribution, as soon as a point is dominated, any other point in a subsequent layer will also be dominated, due to the *transitivity* of dominance, i.e., if $a \prec b$ and $b \prec c$, then $a \prec c$.

Consequently, a correlated distribution does not ensure a minimum score for the skyline points, unless we violate the constraint of the number of points that they dominate. Therefore, we may have to reassign edges, and, possibly, reconsider the layer where some points are, i.e., break the correlated distribution, to minimize the score. Therefore, in our current work, we proceed with the collaborative upper bounds.

2.3.3.2 The SkyIR technique

Now that we have shown how we can efficiently bound the score of a skyline point, using easily extracted information, we turn our focus to finding the top-k most important skyline information, according to our DP-IDP weighting scheme. Algorithm 2.6 shows the general idea of execution of our technique, to efficiently compute the top-k skyline points. Our algorithm processes the points according to a prioritization scheme, and employs pruning of skyline points that will certainly not be in the final top-k result.
The algorithm starts by initializing appropriate information of the skyline points (lines 1-4), such as their dominance count, known score, and priority value, according to the prioritization scheme that we use (see below). This information is essential to compute the bounds of skyline points, using the techniques of the previous section.
We add each skyline point to a priority queue, using its priority value (line 4). We also initialize the k-th value, i.e., the value of the k-th ranked skyline point, to 0. We then enter a loop, each time extracting the top-most item from the priority queue, poi (line 7). If the upper bound of that point's score is below the k-th value, there is no need for further examination (lines 8 -- 10). In this case we discard it and proceed with the next one from the priority queue. Otherwise, we extract the next layer of poi, provided there is one (lines 11--12). We update the point's score using this layer (line 13) and try to add poi in the top-k result. If the point was not added, i.e., its value was below the k-th known score, and it can not be further updated, we discard it and proceed with the next point from the priority queue (lines 14--17). If the point was added, we keep track of the k-th value in the top-k result. If we can further update the point's score, we compute its new priority and add it back in the priority queue (lines 20--21). The

Algorithm 2.6 SkyIR

```
    Input: Skyline $\mathcal{S}$, Dataset $\mathcal{D}$, Integer $k$
    Output: Top-$k$ skyline points with highest score

 1: for every $sp \in \mathcal{S}$ do
 2:     $sp_\Gamma \leftarrow |\Gamma(sp)|$
 3:     $sp_{score} \leftarrow 0$;
 4:     $priorityQueue$.enqueue( $sp_{prior}, sp$ );
 5: $kScore \leftarrow 0$;
 6: while (!$priorityQueue$.empty()) do
 7:     $poi \leftarrow priorityQueue$.dequeue();
 8:     if ( UpperBound( $poi$ ) ) < $kScore$ ) then
 9:         Discard $poi$;
10:         continue;
11:     if ( pending( $poi$ ) > 0 ) then
12:         $lm \leftarrow$ NextLayer( $poi, lm$ );
13:         $poi_{score} \leftarrow$ updateScore( $poi, lm$ );
14:         $added \leftarrow$ topk.insert( $poi, poi_{score}$ );
15:         if (!$added$ AND pending( $poi$ ) == 0) then
16:             Discard $poi$;
17:             continue;
18:         if ( topk[$k$] > $kScore$ ) then
19:             $kScore \leftarrow$ topk[$k$]
20:     if ( pending( $poi$ ) > 0 ) then
21:         $priorityQueue$.enqueue( $sp_{prior}, sp$ );
22: Return topk;
```

loop ends when the priority queue becomes empty, meaning no other points can update their score. The top-k list contains the final result.

Priority Schemes. Our SkyIR algorithm relies on a prioritization scheme to process the skyline points. In this thesis we experiment with the following prioritization schemes.

- **Round Robin** (RRB): Items are processed in a round robin fashion. According to this scheme, we can not process the same skyline point twice, unless we have processed every other skyline point first. This scheme also allows for an implementation that relies on arrays rather than the general priority queue, leading to faster (main memory) acceses.

- **Pending** (PND): The priority of an item is the number of points that it has not yet processed. For example, if a skyline point dominates 100 points, and it has already "seen" 30, its priority will be 70. Therefore, skyline points with more dominated points through which they can update their score will have a higher priority.
- **Upper Bound** (UBS): The priority of a skyline point is given by the upper bound of its score. In other words, its priority is its potential to achieve a high final score. Similarly to the previous scheme, a higher upper bound results in a higher priority for the skyline point. Given that the upper bound can be used as a point's priority, it is even more important to have an efficient technique to compute it, like the one we presented in Algorithm 2.5.

2.3.4 Performance Evaluation

In this section, we report on the results of our experimental evaluation. The experiments were run on a Quad-Core @2.5GHz machine, with 8Gb RAM, running Linux. The code was written in C++ and compiled with g++ 4.7.2, with -O3 optimization. The datasets we consider were indexed by an aggregate R*-tree, with a 4Kb page size. An associated cache with 20% of the corresponding R*-tree's blocks was used with every experiment. Unless stated otherwise, the reported timings are in seconds, measured as CPU processing time and assuming a default value of 8ms per page fault.

Datasets and Algorithms. We generated datasets with *independent* (IND) and *anticorrelated* (ANT) distributions, as in [42], and also use Forest Cover [6]. Table 2.7 shows their basic properties. Although the datasets that we consider seem rather small in size (up to 500K), one should keep in mind that our weighting scheme extracts *all* of the minimal layers for each skyline point. This problem is known to be difficult for high dimensionality even in the RAM model [48]. We also discuss approaches in the following sections to address this issue.
The algorithms that we evaluate are Baseline and SkyIR. For SkyIR we want to compare the performance of the **Loose** (LS) and **Collaborative** (CB) bounds, and how the three prioritization schemes affect

[6] http://kdd.ics.uci.edu

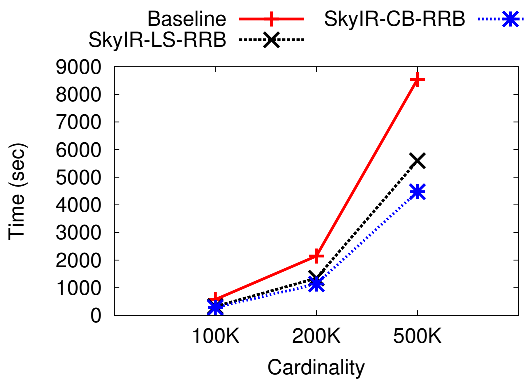
Table 2.7: Dataset Statistics

Data set	Cardinality	Dimensionality
Independent (IND)	100K, <u>200K</u>, 500K	2,<u>3</u>,4
Anticorrelated (ANT)	100K, <u>200K</u>, 500K	2,<u>3</u>,4
Forest Cover (FC)	580K	2,3,4

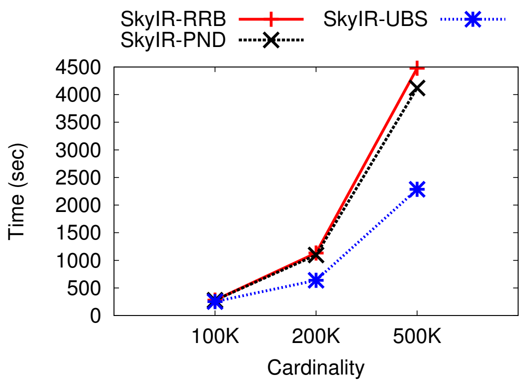
the results. We use the abbreviations as suffixes to indicate what we compare each time.

Runtime. Figure 2.15(a) shows the total runtime for the independent distribution, when varying the dataset cardinality, with k=5. The naive approach is the worst, whereas SkyIR with collaborative bounds performs the best of the techniques, and we have obtained similar results when varying dimensionality and k.

As seen in Figure 2.15(b), the UBS prioritization scheme outperforms all others, resulting in up to 3$\times$ improvement compared to the baseline. Similar results are obtained for different priorities with the loose bounds, but the differences are less pronounced. An important observation from these plots is that the problem we are solving is not linear with the cardinality of points. The reason is that as the cardinality increases, there are more layers of minima to extract, and the computational costs are increased substantially, as a result of both more CPU processing and page faults.



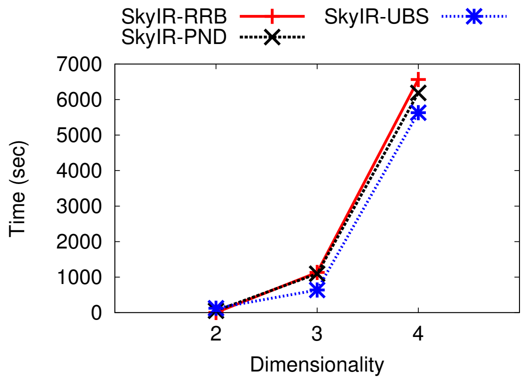
(a) Techniques



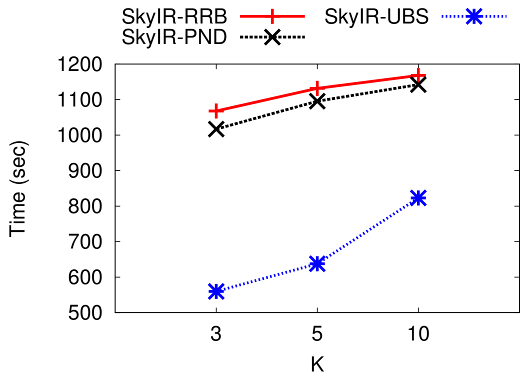
(b) Prioritization for CB

Figure 2.15: Total runtime versus cardinality for IND, k=5

Figures 2.16(a)-(b) demonstrate how each prioritization scheme performs with the collaborative bounds. Figure 2.16(a) shows the perfor-

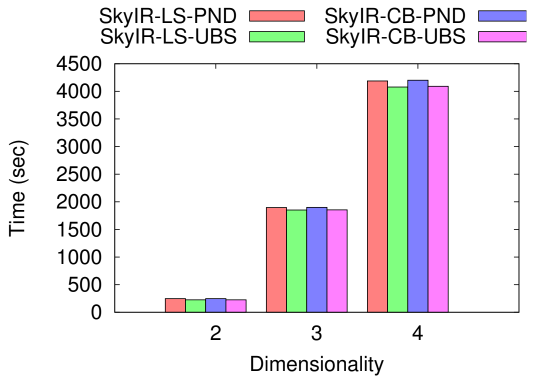


(a) VS Dimensionality (k=5)

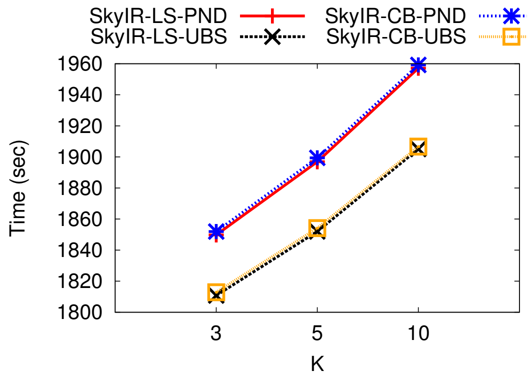


(b) VS k (d=3)

Figure 2.16: Total runtime for various prioritization with IND, CB



(a) VS Dimensionality



(b) VS k

Figure 2.17: Total runtime for ANT distribution

mance when varying the data dimensionality. We observe that UBS performs the best for d = 3, 4, while being slightly worse for d=2. The reason for that is our array-based implementation, which is faster than the reordering of the priority queue maintained by PND and UBS. However, as seen in Figure 2.16(b) there is a huge improvement with UBS for d>2. The improvement increases with lower values of k, going up to 40%, because the collaborative bounds can prune away more points, reducing the computational costs.

Figures 2.17(a) and (b) demonstrate how the prioritization schemes perform for the anticorrelated distribution (ANT), versus dimensionality and k, respectively. Once again, UBS is better than PND. The loose bounds appear to be slightly better than the collaborative, but

not considerably. The difference comes from the fact that the loose bounds are less computationally intensive. The more interesting fact, however, is that ANT appears to be *easier* when compared to IND. In particular, for d=4, it takes $\sim$6000 seconds for CB to compute the top-5 for IND, whereas it takes $\sim$4000 seconds for ANT. The reason is again that IND has more layers to extract, and is more CPU hungry. Even though ANT has a lot of page faults, its CPU time is minimal.
Finally, Figure 2.18 compares the loose and collaborative bounds, when applying the UBS technique on the real dataset FC. We observe that the CB technique performs better than LB for all tested dimensions.

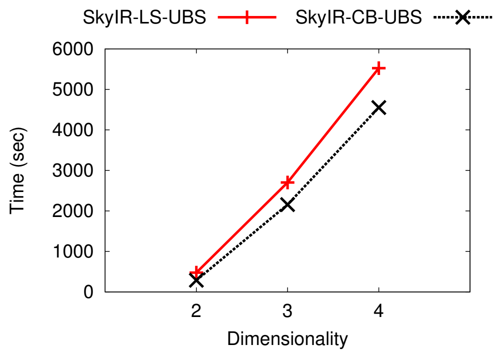
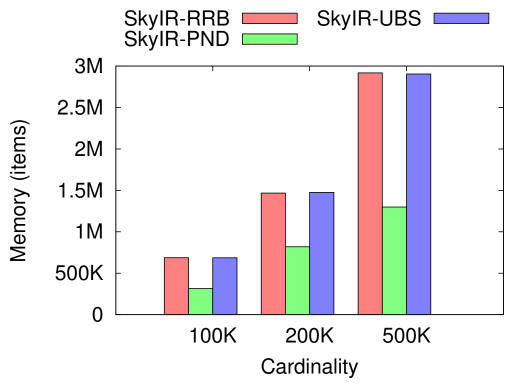
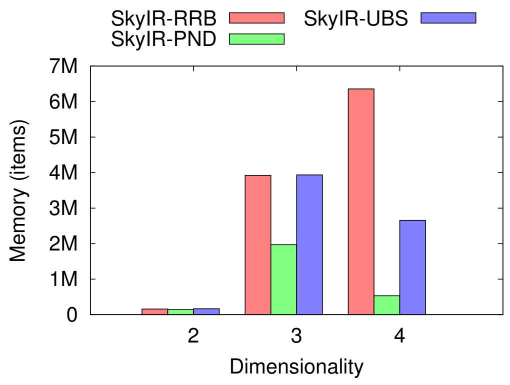


Figure 2.18: Forest cover

Memory Consumption. Finally, we compute the maximum number of items that we must maintain in memory while computing the top-k result. Figures 2.19(a) and (b) show this for IND and ANT, respectively, using the CB technique. We observe that the number of maintained items increases as the cardinality of IND also increases. On the other hand, increasing the dimensionality of ANT, does not have a similar effect: the number of memory points increases as we go from 2D to 3D, but drops again as we proceed to 4D. This may be explained again by the fact that ANT has less layers of minima to retrieve. For a fixed cardinality, more dimensions spread the points more, increasing the points retrieved with each layer. This decreases the information we must store to proceed to the next layer, giving as the plot of Figure 2.19(b).
Generally speaking, the schemes RRB and UBS behave almost the same (with the exception of ANT). We should stress the fact, however,



(a) IND



(b) ANT

Figure 2.19: Maximum memory consumption for CB, k=5

that the pending scheme (PND) *always* results in less memory utilization. This is because the scheme will stick to a single point and try to reduce its number of pending points as much as possible, whereas the other schemes will rotate more over different points. This is an interesting outcome, because PND would be a good alternative in systems with limited resources.

2.4 Summary

Skyline queries are a very expressive and intuitive query type, allowing the user to define their preferences on each attribute individually. The result contains a set of items from the dataset, for which there are no other points which are better in all dimensions.
A major issue with skyline queries is that the size of the skyline result may become too large, depending on the dataset characteristics (cardinality, dimensionality, distribution). In this thesis, we proposed techniques to tackle this issue, by selecting a subset of k skyline points, which have maximum diversity. We presented how diversification can be applied to the skyline context, by employing a distance function that is based entirely on the most fundamental concept of skyline queries: the *dominance* relationship. Consequently, no artificial distance functions are required.
In particular, we quantified the diversity between two skyline points as

the Jaccard distance of their corresponding domination sets, capturing dataset characteristics in the process. To confront the NP-hardness of the problem, we resort to a 2-approximation greedy heuristic. To achieve better performance, at the (slight) expense of quality, we employ MinHash signatures and Locality Sensitive Hashing.
Our framework is applicable even when an index is not available, but we presented techniques which can take advantage of an R-tree like multi-dimensional indexing scheme, to boost performance. We experimentally validated the performance of our approach using real-life and synthetic data sets, achieving orders of magnitude better runtime performance in comparison to straight-forward techniques, without sacrificing quality too much. Our experiments also demonstrated a clear difference between our proposed formalism and other techniques which select a subset of k skyline points.
In addition to selecting a set of k high-quality skyline points, we also presented techniques that rank such points in a way that reflects their individual quality. Inspired by the renowned *tf-idf* weighting scheme from Information Retrieval domain, we presented a novel model that incorporates both local and global characteristics of the skyline points to be ranked. We proposed efficient techniques for finding the top-k result, by bounding the maximum score of a skyline point and employing pruning, combined with different ordering schemes to process the points. We also experimentally evaluated the proposed bounding and prioritization schemes in terms of runtime efficiency and memroy consumption.

Chapter 3

Mining user preferences

3.1 Introduction

In the previous Chapter, we discussed how preferences can be taken into account, in order to identify meaningful and interesting objects for the user. In these cases, preferences are provided in advance as part of the input, where the user specifies, for instance, those attributes she cares about, as well as whether they should be minimized or maximized. In that sense, preferences are defined in a pre-usage scenario of the potential items or services, i.e., when the user is searching and exploring the available information, but before they have actually used it.

However, user preferences over items can be provided in other forms as well, such as user feedback. Online platforms allow feedback to be given in various forms, such as totally structured with strict semantics, e.g., clicking on a "Like", "+1" or other endorsement-like button, or (semi-)structured, such as controlled vocabularies and free text. In the latter cases, information extraction techniques come into play, to mine and better understand the available information.

A characteristic example of such unstructured information is online reviews, where users express their opinions regarding products or services and discuss which aspects they liked (or did not like), how much they enjoyed the product as a whole and each feature individually, etc. Typically, this type of user feedback is provided in semi-structured or free text form. Review mining, i.e., the domain of applying text and data mining techniques to online reviews with the purpose of converting textual information into structured data, has seen a considerable surge over the past years, due to the monetary power of that information. Building upon existing techniques that extract information from reviews, in this thesis, we present techniques that utilize the extracted information and develop a robust, formal framework to identify competitors. Such a framework has been missing until now, and, to the best of our knowledge, our work is the first one to fill this blank.

3.2 Review Mining for Competitor Identification

3.2.1 Motivation

Competitiveness is a challenge that every product or service provider has to face, regardless of the application domain. A significant amount of relevant work has demonstrated the strategic importance of identifying and monitoring an entity's competitors [170]. In fact, a long line of research from the marketing and management community has been devoted to empirical managerial methods for competitor identification [72, 61, 82, 39, 169], as well as to methods for analyzing competitors [56], defending against competitive incursions, and devising appropriate response strategies [86, 57]. Our own work focuses on competitor identification, a key step for any competitiveness-driven study or application. Contrary to the significant amount of available work by the marketing community, the problem has been largely overlooked by computer scientists. For the latter, the challenge is to propose formalizations and competitor-identification algorithms that can utilize the vast amounts of rich data that is nowadays available on the web and other digital sources. Some progress toward this direction has been made by the information systems community [134, 144, 133, 31, 164, 233]. While the proposed approaches help motivate the problem, they present significant shortcomings. These include the lack of a formal definition of competitiveness, as well as the existence of assumptions that limit the applicability of these approaches. Specifically, these techniques are based on mining comparative expressions (e.g. "Item A is better than Item B") from the Web or other textual sources. Even though such expressions can be indicators of competitiveness, they are absent in many domains. For example, consider the domain of vacation packages (e.g flight-hotel-car combinations). In this case, the items have no assigned name by which they can be queried or compared with each other. Further, the frequency of textual comparative evidence can vary greatly across domains. For example, when comparing brand names from the domain of technology (e.g. "Google Vs Yahoo" or "Sony Vs Panasonic"), it is indeed likely that comparative patterns can be found by simply querying the web. However, it is trivial to consider other mainstream domains where such findings are extremely scarce, if not non-existent(e.g. shoes, jewelery, hotels, restaurants, furniture). Finally, even in domains where such approaches are applicable, they cannot actually evaluate the competitiveness relationship between *any* two items. Instead, they can only identify a *subset* of the competitors, based on the available evidence. Our own work overcomes these drawbacks, by providing a formal definition of competitiveness that is applicable across domains. On a high-level, the fundamental problem we address in our work is the following:

Problem 1 *We are given a set of items $\mathcal{I}$, defined within the feature space $\mathcal{F}$ of a particular domain. Then, given any pair of items I, I' from $\mathcal{I}$ we want to define a function $C_{\mathcal{F}}(I, I')$ that computes the competitiveness between the two in the context of the domain.*

As mentioned in the statement of the problem, our notion of competitiveness is based on the feature-space $\mathcal{F}$ of the corresponding domain. Our competitiveness paradigm is based on the following observation: *the competitiveness between two items is based on whether they compete for the attention and business of the same group of users (i.e. the same market share), and to what extent*. For example, two restaurants that exist in different countries are obviously not competitive to each other, since there is no overlap between their target groups. In the ideal scenario, we would have access to the complete set of users that could be interested in a given item. Then, given any two items, we could trivially compete their competitiveness based on the overlap of their respective sets. In practice, however, this is clearly not an option. Taking this into consideration, we formalize the competitiveness between two items based on their respective *features*. For example, if the considered items are MP3 Players, $\mathcal{F}$ could consist of the features *price, sound quality, battery life, connectivity, capacity* and *design*. Our motivation is that, regardless of the domain, users compare and evaluate items based on their features. Therefore, by attaching our formalization to the feature space, we ensure the availability of a consistent and informative resource for competitiveness evaluation. We provide a (simplified) overview of our approach in Figure 3.1.

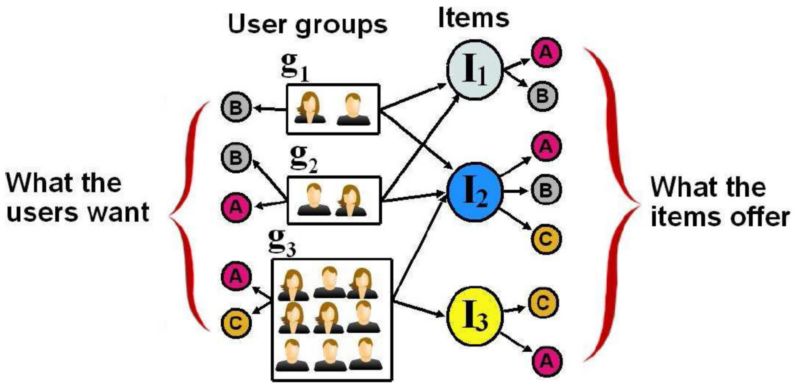


Figure 3.1: Simplified example of our competitiveness paradigm

The figure illustrates the competitiveness between three different items I_1, I_2 and I_3. Each item is mapped to the set of features that it can offer to the users. Three distinct features are considered in this example: A, B and C. Note that, for this simple example, we only consider binary features (i.e. available/not available). Our actual formalization accounts for a much richer space of binary, categorical and numerical features. The left side of the figure shows three groups of users (g_1, g_2, g_3). The example assumes that these are the only groups in existence. Users are grouped based on their preferences with respect to the features. For example, the users in group g_2 are only interested in features A and B. As can be seen by the figure, items I_1 and I_3 are not competitive to each other, since they simply do not appeal to the same groups of users. On the other hand, I_2 is in competition with both I_1 (for groups g_1 and g_2) and I_3 (for g_3). Finally, another interesting observation is that I_2 competes with I_1 for a total of 4 users, and with I_3 for a total of 9 users. In other words, I_3 is a stronger competitor for I_2, since it claims a much larger portion of I_2's market-share than I_1. In our work, we propose ways to deduce these user-groups from sources such as query logs and customer reviews, and describe methods to estimate the size of the market share that they represent. Our work is the first to utilize the opinions expressed in customer reviews as a resource for mining competitiveness.

The formal definition of the competitiveness $C_{\mathcal{F}}(I_i, I_j)$ between two items I_i and I_j, in the context of their domain's feature-space $\mathcal{F}$, is the first contribution of our work. As we demonstrate in our experiments, the evaluation of competitiveness can be a major computational challenge when dealing with real datasets of hundreds or even thousands of items. Motivated by this, we propose an algorithm for the natural problem of finding the top-k competitors of a given item.The proposed framework is geared toward scalability and efficiency, which makes it applicable to domains with large populations of items.

In Section 3.2.3 we introduce our formalization of competitiveness. In Section 3.2.4 we show how we use this formalization toward an efficient framework for finding the top-k competitors of a given item. In Section 3.2.2 we discuss previous related work. The experimental evaluation of our work is presented in Section 3.2.8.

3.2.2 Related Work

To the best of our knowledge, our work is the first to consider domain-invariant competitor mining. Nonetheless, our work has ties to previous work on relevant fields.

Competitor Mining. A previous line of work [134, 133, 31, 225] focuses on mining competitors based on comparative expressions found in web results and other textual corpora. The intuition is that the frequency (i.e. statistically significant) occurrence of expressions like "Item A is better than Item B" or "item A Vs. Item B" are indicative of the competitiveness relationship between the two items. However, as we have already discussed in the introduction, such comparative evidence are typically scarce, or even non-existent in many mainstream domains. As a result, the applicability of such approaches is greatly limited.

Finding Competitive Products. Recent work [217, 218, 237] has explored competitiveness in the context of *product design*. The first step in these approaches is the definition of a dominance function that represents the value of a product. This can measure domination of other items or potential customers. The goal is then to use this function to create items that are not dominated by other, or maximize items with the maximum possible dominance value. A similar line of work [224, 223] represents items as points in a multidimensional space and looks for subspaces where the appeal of the item is maximized. While relevant, the above projects have a completely different focus from our own, and hence the proposed approaches are not applicable in our setting (and vice versa).

Skyline computation. Our work leverages concepts and techniques from the extensive literature on skyline computation [42, 121, 165]. These include the dominance concept among items, as well as the construction of the skyline pyramid used by our CMiner algorithm. Our work also has ties to the recent publications in *reverse skyline* queries [213, 214]. Even though the focus of our work is different, we intend to utilize the advances in this field to extend the functionality and improve the efficiency of our framework in future work.

3.2.3 Formalizing Competitiveness

In this section, we describe how we can formalize and measure the competitiveness between any two items within a given domain. This formalization serves as the building block of our framework. Note that our definition can be easily extended to handle more than two items at a time.

3.2.3.1 Competitiveness via Coverage

In order to synthesize a competitor-mining method that works across domains, we need a formalization of competitiveness that is both accurate and flexible. Motivated by this, we build upon a crucial factor that remains consistent across domains: *user preferences*. In every market, the ultimate goal is to convert users into customers by meeting their individual requirements. Consider a single user u, interested in a specific domain (e.g. restaurants). While the domain may contain numerous items, the user will ultimately choose only one to spend his money on. In a typical scenario, the user follows the following steps:

1. Encode requirements and preferences in a query.
2. Submit the query to a search engine and retrieve the matching items.
3. Process matching items and make the final choice.

We observe that the items that do not match the user's criteria are never considered. In other words, they never get the chance to *compete* for his attention. As far this single user is concerned, the set of competitors consists of the matching items retrieved by the search engine. Consider the following motivating example:

Example: *A user is trying to pick a restaurant for dinner. He has a very limited budget and is only interested in Italian restaurants in the Boston Area. Only the restaurants that satisfy these criteria will compete for the user's attention. On the other hand, Chinese restaurants, restaurants from New York, and expensive establishments are not truly competitors with respect to this particular user, since they are outside the boundaries of his personal requirements and thus never had a chance to be chosen.*

In this example, the user is interested in the features *{price range, location, food type}*. The respective values that encode the user's requirements are *{Cheap, Boston, Italian}*. Clearly, any other assignment of values could be specified for the same query (e.g. *{Cheap, Chicago, Chinese}*). In fact, each possible value-assignment represents the preferences of a different user. Formally, given a subset of features $\mathcal{F}'$, let $\mathcal{V}_{\mathcal{F}'}$ be the complete space of all possible value assignments over the features in $\mathcal{F}'$. We observe that every item covers a portion of the entire space $\mathcal{V}_{\mathcal{F}'}$, and, hence, covers the corresponding population of users. For example, a cheap restaurant in Boston that serves both Italian and American food *covers* a user who is interesting in cheap Italian food in Boston, as well as a user who is interested in cheap American food in the same city.

In order to evaluate the competitiveness of two given items I_i, I_j in the context of a subset of features $\mathcal{F}'$, we need to compute the number of possible value assignments over $\mathcal{F}'$ that are satisfied by *both* items. Formally, we define pairwise coverage as follows:

Definition 1 [Pairwise Coverage] *Given the complete set of features $\mathcal{F}$ in a given domain of interest, let $\mathcal{V}_{\mathcal{F}'}$ be the complete space of all possible value-assignments over the features in a subset $\mathcal{F}' \subseteq \mathcal{F}$. Then, the* coverage $cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_k)$ *of a pair of items I_i and I_j with respect to $\mathcal{V}_{\mathcal{F}'}$ is defined as the portion of $\mathcal{V}_{\mathcal{F}'}$ that is covered by both items.*

Considering the above definition, we observe that the coverage of each dimension (i.e. each feature $F \in \mathcal{F}'$) is independent of the others. Therefore, we first compute the percentage of each dimension that is covered by the pair. We can then optimally compute the coverage of the entire space $\mathcal{V}_{\mathcal{F}'}$ as the product of the respective coverage values $\mathcal{V}_{\{F\}}$ for every $F \in \mathcal{F}'$. Formally:

$$cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) = \prod_{F \in \mathcal{F}'} cov(\mathcal{V}_{\{F\}}, I_i, I_j) \tag{3.1}$$

This computation has a clear geometric interpretation: The portion of the space $\mathcal{V}_{\mathcal{F}'}$ that is covered by a pair of items can be represented as a hyper-rectangle in $|\mathcal{F}'|$-dimensional space. For each dimension F, $cov(\mathcal{V}_{\{F\}}, I_i, I_j)$ gives us the portion of the dimension that is covered by the two items. Finally, by multiplying the individual coverage values, we are essentially computing the volume of the hyper-rectangle that represents the entire space $\mathcal{V}_{\mathcal{F}'}$.
Figure 3.2 illustrates the pairwise coverage provided by two items I_1, I_2 in the context of the two dimensional space defined by two features F_1 and F_2. As shown in the figure, the two pairwise coverage of the the two items is defined by the highlighted rectangle. The three points q_1, q_2, q_3 represent different value assignments for these two features. Every assignment that falls within the rectangle is *covered* by both items.
Definition 1 allows us to evaluate the coverage provided by a pair of items to (the value space of) any subset of features $\mathcal{F}'$. Conceptually, $\mathcal{F}'$ captures the fraction of the population that is interested in the features included in $\mathcal{F}'$. In practice, the size of the corresponding population varies across subsets. For example, in the domain of restaurants, the subset *{food quality, price range}* is arguably of interest to more users than the subset *{Wi-Fi availability, delivery options}*.

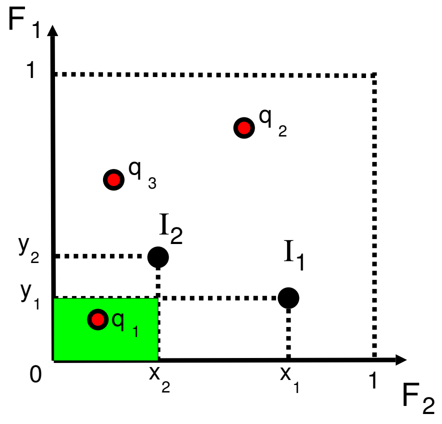


Figure 3.2: Geometric interpretation of pairwise coverage

To account for this in our definition of competitiveness, we attach a popularity weight $w(\mathcal{F}')$ to each feature subset. We revisit the computation of these weights in Section 3.2.6, where we discuss practical methods for learning the weights from sources such as *query logs* and *customer reviews*. For the remaining of our analysis, we assume that the weights are provided as part of the input. Further, we define $\mathcal{Q}$ to be the collection of subsets with a non-zero weight. Formally: $\mathcal{Q} = \{\mathcal{F}' \in 2^{\mathcal{F}} : w(\mathcal{F}') > 0\}$. Taking the above into consideration, we formally define the competitiveness of two items I_i, I_i as follows:

Definition 2 [Competitiveness] *Given the complete set of features $\mathcal{F}$ of a domain of interest, let $\mathcal{Q}$ be the set of all subsets of $\mathcal{F}$ that have a non-zero popularity weight. Then, the competitiveness of two given items I_i and I_i is defined as:*

$$C_{\mathcal{F}}(I_i, I_j) = \sum_{\mathcal{F}' \in \mathcal{Q}} w(\mathcal{F}') \times cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) \qquad (3.2)$$

where $cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j)$ is the portion of $\mathcal{V}_{\mathcal{F}'}$ that is covered by both I_i and I_j.

3.2.3.2 Computing Coverage

Our definition of competitiveness between two given items is based on the pairwise coverage that they provide to the value space of the different subsets of features. We observe that the value space is complex, since it can contain different types of features. By supporting virtually every reasonable feature-type (numeric, ordinal, boolean, categorical), our framework guarantees the flexibility required to encode the requirements of virtually *any* potential customer. Next, we discuss the different types of features that we consider in our work, and show how coverage is defined for each of them.

Categorical Features. In this thesis, we identify two sub-types of categorical features: single-value and multi-value. For a single-value feature F, each item assumes exactly one value from the respective value-space $\mathcal{V}_{\{F\}}$, e.g. the brand of a digital camera. Clearly, boolean features are simply a special case of this group, assuming values from *{YES, NO}*. The pairwise coverage of two items I_i, I_j, given a single-value feature F, is defined as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \begin{cases} 1 & \text{if } I_i[F] = I_j[F] \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

For a multi-value feature F, each item can be mapped to any *subset* of values from the respective value-space $\mathcal{V}_{\{F\}}$. Assume, for example, the feature *parking* from Table 3.1, referring to the parking facilities of a restaurant. Since a restaurant can in fact provide any number of these options (even all of them), *parking* is a multi-value categorical feature. We define the pairwise coverage of two items I_i, I_j for a multi-value feature F, as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \frac{|I_i[F] \cap I_j[F]|}{|\mathcal{V}_{\{F\}}|} \tag{3.4}$$

Conceptually, the covered portion is defined as the overlap between the sets of values that are mapped to each item, divided by the total number of possible values for F. Clearly, the dividend is always a value in $[0, 1]$.

Ordinal Features. The value space $\mathcal{V}_{\{F\}}$ of an ordinal feature F assumes values from a finite *ordered* list: $\mathcal{V}_{\{F\}} = \{v_1, v_2, v_3, ...\}$. Depending on the nature of the feature, higher or lower states may be preferable. For example, for the feature *price range*, lower values are

preferable. On the other hand, for the feature *stars rating*, higher values are better.
First, let us introduce two functions that will aid us in our definition of coverage in the context of ordinal features. Given an ordinal feature F and two values $v_1, v_2 \in \mathcal{V}_{\{F\}}$, let $loser(F, v_1, v_2)$ return the least preferable between the two values. In addition, let $weq(F, v_1)$ return the set of values that are worse or equal to v_1. For example, for the *price range* feature discussed above, weq(F, $$$)={$$$, $$$$}. Then, the pairwise coverage of two given items to the value space is defined as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \frac{|weq(loser(F, I_i[F], I_j[F]))|}{|\mathcal{V}_{\{F\}}|} \tag{3.5}$$

As in the case of categorical features, $cov(\mathcal{V}_{\{F\}}, I_i, I_j)$ takes values in $[0, 1]$.

Numeric Features. A numeric feature F takes values from a continuous pre-defined range. Without loss of generality, we assume that all numeric features are normalized to take values in $[0, 1]$. Higher or lower values may be preferable, depending on the nature of the feature. As in the case of ordinal features, we define $loser(F, v_1, v_2)$ to return the least preferable of two given values for a feature F. Then, given two items I_i, I_j, the pairwise coverage of the value space $\mathcal{V}_{\{F\}}$ is defined as:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = loser(F, I_i, I_j) \tag{3.6}$$

Conceptually, the value space that is commonly covered by the two items is bounded by the one with the least preferable value. Now that we have provided a definition of coverage for every supported feature-type, we present the following example.

Example: Consider the subset of features $\mathcal{F}'$ shown in Table 3.1. The respective representations for two items I_i, I_j are { $$$, *Boston*, *{Street, Valet}*, 0.8 } and { $$, *Boston*, *{Street, Priv. Lot}*, 0.6 }. Then, following Eq. 3.1, the pairwise coverage of the two items of is com-

Table 3.1: Feature-subsets and their respective value-spaces.

Feature	Type	Value-Set $\mathcal{V}_{\{\mathcal{F}\}}$
price range	ordinal	{$, $$, $$$, $$$$}
location	categorical (single)	{*Boston, New York*}
parking	categorical (multi)	{*Street, Priv. Lot, Valet*}
food quality	numeric	$[0, 1]$

puted as follows:

$$cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) = \frac{2}{4} \times 1 \times \frac{1}{3} \times 0.6 = 0.1$$

3.2.4 Finding the Top-K Competitors

In the previous section we presented a formal definition of the competitiveness between any two items. Given this definition, we study the natural problem of finding the top-k competitors of a given item. Formally, the problem is defined as follows:

Problem 2 *We are given a set of items $\mathcal{I}$, defined within the feature space $\mathcal{F}$ of a domain. Then, given a single item $I \in \mathcal{I}$, we want to identify the k items from $\mathcal{I} \setminus \{I\}$, that maximize the pairwise competitiveness with I:*

$$I^* = \underset{I' \in \mathcal{I} \setminus \{I\}}{\text{argmax}} \, C_{\mathcal{F}}(I, I') \tag{3.7}$$

A naive algorithm for this problem would iterate over all items in $I' \in \mathcal{I} \setminus \{I\}$. For each such item I', it would compute $w(\mathcal{F}') \times cov(\mathcal{V}_{\mathcal{F}'}, I, I')$ for every subset $\mathcal{F}' \in \mathcal{Q}$, where $\mathcal{Q}$ is the collection subsets with a non-zero weight. It would then be trivial to obtain the top-k competitors for the given item. However, considering that $\mathcal{I}$ can contain thousands of items, the computational cost can be overwhelming. We demonstrate this in our experiments, where we compare our own CMiner technique with the naive approach.

3.2.5 The CMiner Algorithm

Motivated by the inefficiency of the naive approach, we present CMiner, a new algorithm for Problem 2. Our approach combines scalability with the ability to handle the online arrival of new items. The latter is crucial in many mainstream domains. As an example, consider the case when items are vacation packages. In such a domain, an arbitrary number of new packages can be introduced at any point in time. Hence, we would like to preprocess the data in a way that allows us to compute the competitors of a new package without having to repeat the entire computation effort.
First, we define the concept of *item dominance*, which will aid us in our further analysis:

Definition 3 [Item Dominance]: *Given two items I_i, I_j from a set $\mathcal{I}$ defined within a feature-space $\mathcal{F}$, we say that an item I_i dominates an item I_j if both of the following conditions are true:*

1. *$I_j[F] \subseteq I_i[F]$, for every multi-value categorical feature $F \in \mathcal{F}$*
2. *$I_i[F] \geq I_j[F]$, for every ordinal, numerical, or single-categorical feature $F \in \mathcal{F}$.*

Conceptually, an item dominates another if it has better values for all features of the considered space $\mathcal{F}$. Clearly, if I_i dominates I_j, then it is also more competitive with respect to any other item from $\mathcal{I}$ (since it covers at least as much coverage to any possible sub-space as I_j). This observation motivates us to utilize the *skyline* of the entire set of items $\mathcal{I}$. The skyline is a well-studied concept that represents the subset of points in a set that are not dominated by any other point in the set [42]. We refer to the skyline of a set of items $\mathcal{I}$ as $Sky(\mathcal{I})$. The concept of the skyline leads to the following lemma:

Lemma 6 *Given the skyline $Sky(\mathcal{I})$ of a set of items $\mathcal{I}$ and an item $I_i \in \mathcal{I}$, let $\mathcal{Y}$ contain the k items with the highest $\mathcal{C}_{\mathcal{F}}(\cdot, I_i)$ values from $Sky(\mathcal{I})$. Then, an item $I_j \in \mathcal{I}$ can only be in the top-k competitors of I_i, if $I_j \in \mathcal{Y}$ or I_j is dominated by one of the items in $\mathcal{Y}$.*

Proof 5 *We will prove this by contradiction. Let I_j be an item that is not included in $\mathcal{Y}$, and has a competitiveness value that is higher or equal to that of some item $I \in \mathcal{Y}$. The assumption is that I_j is not dominated by any of the items in $\mathcal{Y}$. Observe that I_j cannot be in the skyline, since its competitiveness would have included it in $\mathcal{Y}$. Hence, it is guaranteed to be dominated by at least one of the items in the skyline. This means that there is an item $I' \in Sky(\mathcal{I})$ with a competitiveness higher or equal to that of I_j. However, since I_j has a greater (or equal) value than one of the items in $\mathcal{Y}$, the same applies for I' which is guaranteed to be included in $\mathcal{Y}$. This leads to a contradiction, since we assumed that none of the items in $\mathcal{Y}$ dominate I'.*

By applying Lemma 6, we do not need to consider the entire set of items in order to find the top-k competitors of a given item I^*. Instead, it is sufficient to recursively check for the items that are dominated by the current top-k items from the upper levels. In order to fully utilize this observation, we construct a structure that greatly reduces the number of items that need to be considered for the computation of the top-k competitors set. We refer to this structure as the *skyline pyramid*. The pyramid can be constructed by recursively computing the skyline and removing the skyline points from the current set, until the entire collection of items has been exhausted. Standard techniques can be used for computing the skyline on each iteration [165], as well as for updating the pyramid in case new items are introduced [121]. Each item from the i_{th} layer of the skyline is assigned an inlink from the item from i_{th} level that dominates it. If multiple such dominators exist, we simply choose one randomly. This is simply done to avoid re-checking the dominated item during the competitor-finding process, and does not affect the optimality of the result. An example of the skyline pyramid is shown in Figure 3.3. The left side of the figure shows the complete dominance graph for a given set of items. An edge $I_i \rightarrow I_j$ means that I_i dominates I_j. The right side of the figure shows the skyline domination pyramid.

Lemma 7 *Assume the skyline pyramid structure on a set of items $\mathcal{I}$. We can retrieve the optimal top-k competitors set with respect to an*

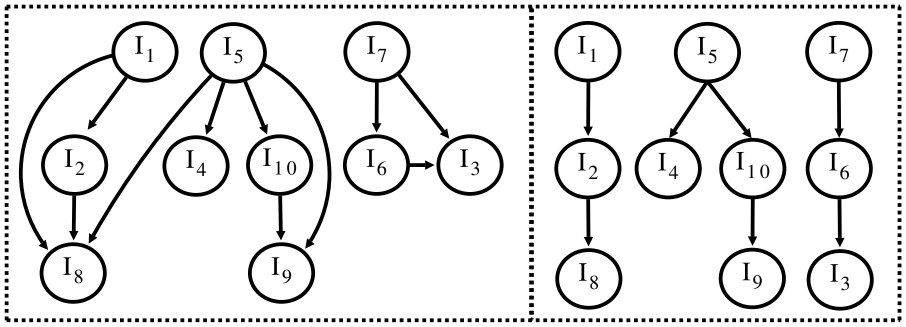


Figure 3.3: An example of the skyline pyramid structure

item I^*, iff we maintain exactly one edge for each item I_{i+1} in layer $i+1$ from any item I_i in layer i, such that I_i dominates I_{i+1}.

Proof 6 *First, we will show that we need to maintain at least one such edge, between an item from layer $i+1$ and an item from layer i. The proof is immediately derived from Lemma 6, since a top-k competitor could be an item from layer $i+1$, which is dominated by some point in layer i. If we do not maintain such an edge, then item I_{i+1} will never be checked and the top-k result will not be optimal. Therefore, we need to maintain at least one such edge, for any item that is dominated by some other.*
We will now show that it is sufficient to maintain at most one such edge, i.e., if item I_{i+1} is dominated by two or more points from I_i, say I_1 and I_2, keeping only one of these edges will yield the optimal result. The proof is given by contradiction. Assume 3 points, I_1, I_2 and I_3, with I_1 and I_2 being in the i-th skyline layer and I_3 in the $(i+1)$-th layer, and that both I_1 and I_2 dominate I_3. Additionally, assume that I_1 and I_3 should be in the top-k result, but I_2 should not, and that from the two edges $I_1 \rightarrow I_3$, $I_2 \rightarrow I_3$, we keep the latter ($I_2 \rightarrow I_3$). If I_2 is not in the top-k result, then $\exists I_k$ s.t. $\mathcal{C}_{\mathcal{F}}(I^*\ I_2) < \mathcal{C}_{\mathcal{F}}(I^*, I_k)$. However, $\mathcal{C}_{\mathcal{F}}(I^*, I_3) < \mathcal{C}_{\mathcal{F}}(I^*, I_2)$, because that is how the skyline pyramid is constructed. Therefore, $\mathcal{C}_{\mathcal{F}}(I^*, I_3) < \mathcal{C}_{\mathcal{F}}(I^*, I_k)$ and I_3 should not be in the top-k result, hence the contradiction.

A corollary from the above proof is that for an item I to be taken into consideration as a candidate for the top-k set, all of its masters should have been eligible for consideration in the previous round. That is all of I's masters should be among the top-k competitors. Since all of

I's masters need to be in the top-k result, we could maintain the edge $J \to I$ for the point J that is farthest from I, with respect to some predefined criteria (e.g. $L1$ distance, or $\mathcal{C}_{\mathcal{F}}(J, J)$).
The pseudocode for the pyramid-extraction process is given in Algorithm 3.1. We refer to this process as `PyramidFinder`. The input to `PyramidFinder` is the set of items $\mathcal{I}$. The output is the *skyline pyramid* $\mathcal{D}_{\mathcal{I}}$. In our experiments, we explore the construction of the skyline pyramid for large datasets, discuss its characteristics, and demonstrate its usefuleness in the context of top-k competitor search.

Algorithm 3.1 PyramidFinder

```
    Input: Set of items $\mathcal{I}$
    Output: Dominance Pyramid $\mathcal{D}_{\mathcal{I}}$
 1: $\mathcal{D}_{\mathcal{I}}[0] \leftarrow$ Sky($\mathcal{I}$)
 2: $\mathcal{Z} \leftarrow \mathcal{I} \setminus$ Skyline($\mathcal{I}$)
 3: $level \leftarrow 1.$
 4: while $\mathcal{Z}$ is not empty do
 5:     $\mathcal{D}_{\mathcal{I}}[level] \leftarrow$ Sky($\mathcal{Z}$)
 6:     for every item $I_j \in \mathcal{D}_{\mathcal{I}}[level]$ do
 7:         for every item $I_i \in \mathcal{D}_{\mathcal{I}}[level-1]$ do
 8:             if $I_i$ dominates $I_j$ then
 9:                 Add a link $I_i \rightarrow I_j$
10:                 break
11:     $\mathcal{Z} \leftarrow \mathcal{Z} \setminus$ skyline($\mathcal{Z}$)
12:     $level \leftarrow level + 1$
```

3.2.5.1 The CMiner Algorithm

Next, we present CMiner, an optimal algorithm for finding the top-k competitors of any given item. Our algorithm makes use of the skyline pyramid described earlier in this section, in order to reduce the number of items that need to be considered and minimize the number of required coverage computation. The intuition is that, since we only care about the top-k competitors, we can incrementally compute the score of each candidate and stop when it is guaranteed that the top-k have emerged. The pseudocode is given in Algorithm 3.2.

The input to the algorithm includes the set of items $\mathcal{I}$, the set of features $\mathcal{F}$, the item of interest I^*, the number k of top competitors to

Algorithm 3.2 CMiner

```
Input: Set of items $\mathcal{I}$, Item of interest $I^* \in \mathcal{I}$, feature space $\mathcal{F}$, Collection $\mathcal{Q}$ of feature-subsets with non-zero weights, skyline pyramid $\mathcal{D}_{\mathcal{I}}$, int k
Output: Set of top-k competitors for $I^*$ from $\mathcal{I} \setminus \{I^*\}$

 1: $TopK \leftarrow masters(I^*)$
 2: if ( $k \leq |TopK|$ ) then
 3:     return $TopK$
 4: $k \leftarrow k - |TopK|$
 5: $LB \leftarrow -1$
 6: $low(I) \leftarrow 0, \forall I \in \mathcal{X}$.
 7: $up(I) \leftarrow \sum_{\mathcal{F}' \in \mathcal{Q}} w(\mathcal{F}') \times (cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)), \forall I \in \mathcal{X}$.
 8: $\mathcal{X} \leftarrow$ getSlaves($TopK, \mathcal{D}_{\mathcal{I}}$) $\cup \mathcal{D}_{\mathcal{I}}[0]$
 9: while ( $|\mathcal{X}|$ != 0 ) do
10:     $\mathcal{X} \leftarrow$ updateTopK($k, LB, \mathcal{X}$)
11:     if ( $|\mathcal{X}|$ != 0 ) then
12:         $TopK \leftarrow merge(TopK, \mathcal{X})$
13:         if ( $|TopK| = k$ ) then
14:             $LB \leftarrow TopK[k]$
15:         $\mathcal{X} \leftarrow$ getSlaves($\mathcal{X}, \mathcal{D}_{\mathcal{I}}$)
16: return $TopK$

17: Procedure updateTopK(k, LB, $\mathcal{X}$)
18: $localTopK \leftarrow \emptyset$
19: for every $\mathcal{F}' \in \mathcal{Q}$, in sorted order do
20:     $sc \leftarrow w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)$
21:     $localTopK \leftarrow \emptyset$
22:     for every item $I \in \mathcal{X}$ do
23:         $up(I) \leftarrow up(I) - sc + w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I)$
24:         if ( $up(I) < LB$ ) then
25:             $\mathcal{X} \leftarrow \mathcal{X} \setminus \{I\}$
26:         else
27:             $low(I) \leftarrow low(I) + w(\mathcal{F}') \times (cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I))$
28:             $localTopK.add(I, low(I))$
29:             if ( $|localTopK| = k$ ) then
30:                 $LB \leftarrow localTopK[k]$
31:     if ( $|\mathcal{X}| \leq k$ ) then
32:         break
33: for every item $I \in \mathcal{X}$ do
34:     for every remaining $\mathcal{F}' \in \mathcal{Q}$ do
35:         $low(I) \leftarrow low(I) + w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I)$
36:     $localTopK.add(I, low(I))$
37: return localTopK
```

retrieve, the collection $\mathcal{Q}$ of feature-subsets with non-zero weights, and the skyline pyramid $\mathcal{D}_{\mathcal{I}}$.

In lines 1-4, the algorithm uses *masters*(I^*) to retrieve the set of items that *dominate* I^*. Note that this set can be easily pre-computed for all the items during the pyramid-construction phase. These items are guaranteed to have the maximum possible competitiveness with I^*. If at least k such items exist, we can just report them and conclude. Otherwise, we append them to the final result and decrement our budget of k accordingly. The LB variable maintains the lowest lower bound from the current top-k set. This is used as pruning threshold for the candidates. In lines 6-7 we initialize the upper and lower bounds for each candidate. In line 8 we initialize the set of candidates $\mathcal{X}$ as the union of the items in the first layer of the pyramid and the items dominated by those in the *TopK*. The latter is returned via calling `getSlaves`$(TopK, \mathcal{D}_{\mathcal{I}})$.

In every iteration of lines 9-15, the algorithm does the following: (i) it feeds the set of candidates $\mathcal{X}$ routine, which prunes items based on the LB threshold, (ii) updates the *TopK* set via the *(*$merge$*)*$(\cdot)$ function, (iii) updates the pruning threshold LB, (iv) expands the set of items by including the items that they dominate.

Discussion of `UpdateTopK()`. This routine processes the items in $\mathcal{X}$ and finds at most k with the highest competitiveness scores among $\mathcal{X}$, subject to the condition that this score is higher than the global pruning threshold LB. The approach uses two bounds low and up, for every $I \in \mathcal{X}$. $low(I)$ maintains the competitiveness score of item I, as new feature subsets are considered. $up(I)$ is an optimistic upper bound on I's competitiveness score. Therefore, up begins with the maximum possible competitiveness score, $\mathcal{C}_{\mathcal{F}}(I^*, I^*)$.

For every feature subset, we examine all items in $\mathcal{X}$ and update their up value. If at any point $up(I) < LB$ (line 24), item I can be safely removed from the $\mathcal{X}$. If, at any point, $|\mathcal{X}|$ becomes less or equal to k, the loop over the subsets comes to a halt. In lines 33-36 we update the lower bounds of the remaining items in $\mathcal{X}$. We do this outside the loop, in order to avoid unnecessary bound checking and improve performance. Observe that the routine processes subsets in *sorted order*. In Section 3.2.6, we elaborate on the impact of the ordering on the performance of CMiner.

3.2.5.2 Algorithmic Complexity

The complexity of CMiner depends on the number of points in each layer of $\mathcal{D}_{\mathcal{I}}$. According to Bentley et al. [38], for n uniformly-distributed d-dimensional data points (items), the expected size of the skyline is $\Theta(\frac{ln^{d-1}n}{(d-1)!})$. Since we need to examine at most k skyline layers to find the top-k result, this value is upper-bounded by $\Theta(k * \frac{ln^{d-1}n}{(d-1)!})$. This bound naively assumes that each layer should be considered entirely. In practice, however, we only need to check a small fraction of items that are dominated by the items considered in the previous layer. For instance, for uniform distribution, with consecutive skyline layers of similar sizes, the number of points to be considered will be in the order of k, since links will be evenly distributed among the skyline points. As we only expand the top-k items in each step, at most k new items will be introduced. Therefore, for small values of k, the complexity of CMiner is written as O ($|\mathcal{I}|$ * $|\mathcal{Q}|$ * k^2), where $\mathcal{Q}$ is the set of feature subsets with non-zero weights.

3.2.6 Weight-Estimation for Feature-Subsets

Our analysis has assumed that the weight $w(\mathcal{F}')$ of each subset of features $\mathcal{F}'$ is provided as input. In this section, we discuss methods for computing these weights.
The motivation of assigning a different weight to each feature-subset stems from the real-life observation that not all features are equally important to users. Based on this, a straightforward approach is to consider the weight of each individual feature separately, and then aggregate to the subset-level. This aggregation could be achieved by selecting the sum, average, median, maximum or minimum over all the individual features in a set. This approach assumes independence among the features of an item. This assumption, however, is not always valid. For example, it may be the case that people who are interested in the screen resolution of a laptop computer are also more likely to be interested in the included graphics card. This motivates an approach that considers the popularity of *feature-subsets* instead of individual features. We identify two sources from which we can

learn the popularity of a subset of features: *query logs* and *customer reviews*.

Query logs. The first source is the query logs of the search engine on the website where the items are hosted. Regardless of the interface through which the user encodes his preferences in a query, the set of selected feature is always recorded in a dedicated log. Assuming the existence of a large enough user-base, we can simply estimate the popularity of a feature-subset based on the number of times that it was queried upon by the users.

Customer reviews. In cases when query logs are unavailable or inadequate, the weights of the subsets can be estimated by considering the *reviews* that are available for the items in the domain. As an example of such a dataset, consider the union of the review sets that are available for all the digital cameras offered on amazon.com. Each of these reviews comments on a particular subset of attributes from the digital-camera domain. Hence, the review corpus serves as an intuitive way to access user preferences. For example, a user who is greatly interested in the wheelchair-accessibility of a restaurant is more likely to discuss this feature in his review. We implement and employ review mining as means for estimating the weights of feature-subsets in our experiments. In practice, one can choose to ignore subsets that appear less frequently than a set threshold. In our own experiments, we consider all subsets that appear at least once.

3.2.7 Subset Ordering

Given an item of interest I^*, CMiner iterates over the given set of subsets and computes the coverage provided by I^* and each candidate item to the value-space that corresponds to each subset. Given our definition of competitiveness, we next consider **IC**, an ordering scheme that, given an item of interest I^*, processes subsets in descending order of $w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)$ values. As stated in the following lemma, **IC** achieves the optimal *convergence rate* (i.e. there exists no ordering that can result to a faster convergence).

Lemma 8 [IC Convergence Rate]: Given two items I_i, I_j, the ordering imposed by the **IC** scheme results in the fastest possible convergence to the target-value $C_{\mathcal{F}}(I_i, I_j)$ (i.e. the true competitiveness between the two items)

Proof 7 *Assume that we want to compute the competitiveness between the target item I^* and a candidate I'. Let l_i and u_i be the lower and upper competitiveness bounds, after checking $\mathcal{F}'_i$, the i-th feature subset imposed by the **IC** scheme. For ease of notation, we use $C_{\mathcal{F}'_i}(I_1, I_2) = w(\mathcal{F}'_i) \times cov(\mathcal{V}_{\mathcal{F}'_i}, I_1, I_2)$, for any two items. Every time a new subset is considered, l_i and u_i are updated, until finally all the subsets have been evaluated and both variables converge to the actual competitiveness score $C_{\mathcal{F}}(I^*, I')$. We now define $T_i = u_i - l_i$. Since both l_i and u_i ultimately converge to $C_{\mathcal{F}}(I^*, I')$, T_i converges to 0. Also, $T_i \geq 0, \forall i$. The convergence rate of T_i is:*

$$\frac{T_i}{T_{i-1}} = 1 - \frac{C_{\mathcal{F}'_i}(I^*, I^*)}{u_{i-1} - l_{i-1}} \tag{3.8}$$

Also, we know that:

$$u_i = C_{\mathcal{F}}(I^*, I^*) - \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I^*) + \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I^*)$$

and

$$l_i = \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I')$$

By immediate replacement in Eq. 3.8, the convergence rate becomes:

$$\frac{T_i}{T_{i-1}} = 1 - \frac{C_{\mathcal{F}'_i}(I^*, I^*)}{C_{\mathcal{F}}(I^*, I^*) - \sum_{j=1}^{i-1} C_{\mathcal{F}'_j}(I^*, I^*)} \tag{3.9}$$

*As it can be seen by Eq. 3.9, the convergence rate depends only on the score of the target item I^*. The **IC** ordering scheme processes* *subsets in decreasing order of $C_{\mathcal{F}'_j}(I^*, I^*)$, which is the maximum possible coverage that any item can jointly achieve with I^*. Thus, the numerator is equal to the i_{th} maximum possible value among all feature-subsets. Similarly, the difference in the denominator is minimized, since the subtracted sum maintains the highest possible value (and $C_{\mathcal{F}}(I^*, I^*)$ is constant).*

Given **IC**'s optimality with respect to the convergence rate, we adopt it as the standard ordering scheme for CMiner. In Section 3.2.8, we present experiments that demonstrate the superiority of **IC** when compared with other sub-optimal ordering schemes. In fact, it will become obvious that the order in which the subsets are processed can have a tremendous impact on the overall efficiency of the algorithm.

3.2.8 Experimental Evaluation

For our experimental evaluation, we compiled the following datasets:

Digital `Cameras` **from** `Amazon.com`: The features of this domain includes the objective attributes of each camera (e.g. *price, number of megapixels*), as well as numeric attributes representing the opinions of the users on the item's different characteristics (e.g. *photo quality, video quality*). These were extracted via the opinion method by Ding et al. [73], which assigns a numeric opinion-value to each feature of an item, given the corpus of reviews. All scores were normalized to be in $[0, 1]$, with higher scores being preferable. The same method was also used for the datasets from `Booking.com` and `TripAdvisor.com`.

`Hotels` **from** `Booking.com`**:** The feature-set for this domain consists of objective features (e.g. *price, location*) and the opinion values extracted from the relevant reviews on different attributes (e.g. *cleanliness, service quality*).

`Restaurants` **in New York from** `TripAdvisor.com`**:** The feature-set for this domain consists of objective features (e.g. *type of food served*) and the opinion values extracted from the relevant reviews on different attributes (e.g. *food quality,service quality* etc.)

`Recipes` **from** `Sparkrecipes.com`**:** The feature-set for each recipe consisted of the different nutritional values (e.g. grams of protein and carbohydrates), which are available on the website.

The datasets were intentionally selected from different domains to portray the cross-domain applicability of our approach. Table 3.2 summarizes some basic statistics for each dataset.

Table 3.2: Dataset Statistics

Dataset	**#Items**	**#Feats.**	**#Subsets**	**Skyline Layers**
Cameras	579	21	14779	5
Hotels	1283	8	127	5
Restaurants	4622	8	64	12
Recipes	100000	22	133	22

For each dataset, the second, third, fourth and fifth columns include the number of items, the number of features, the number of distinct feature-subsets, and the number of layers in the respective skyline pyramid, respectively. The feature subsets were extracted from the set of reviews that is available for each dataset; the frequency of each subset of features is equal to the number of times they were included together in a review. To conclude the description of our datasets, we present some statistics on the skyline-pyramid structure constructed for each corpus. Figure 3.4 shows the distribution of items in the first

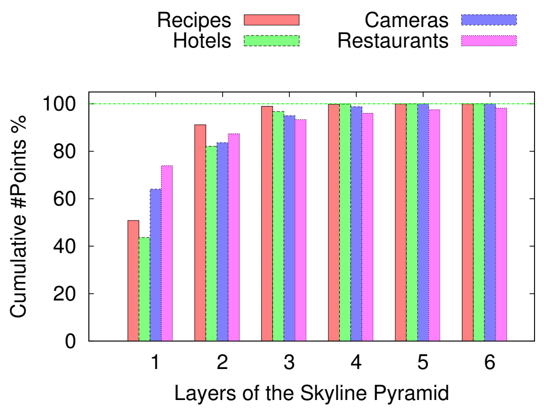
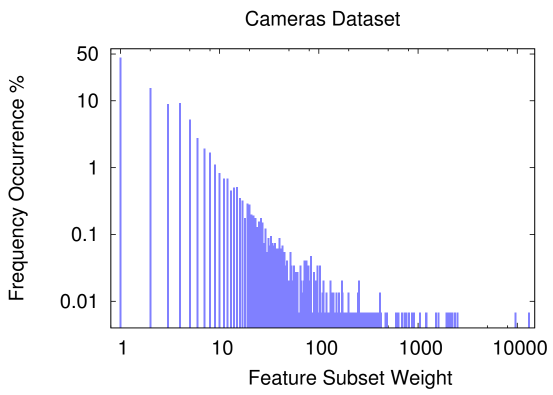
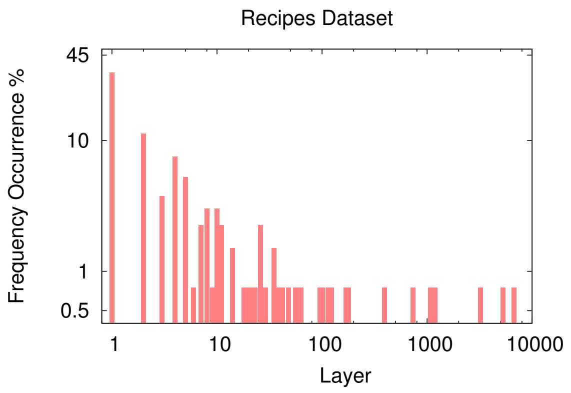


Figure 3.4: Cumulative distribution of items across the first 6 layers of the skyline pyramid.



(a) Restaurants



(b) Recipes

Figure 3.5: Distribution of feature subset weights

6 skyline layers of each dataset.

We observe that, for all datasets, nearly 99% of the items can be found within the first 4 layers, with the majority of those falling within the first 2 layers. This is due to the large dimensionality of the feature space, which makes it difficult for items to dominate one another. As we show in the following experiment, the skyline pyramid helps our CMiner algorithm to clearly outperform the baselines with resepect to computational cost. This is despite the high concentration of items within the first layers, since CMiner can effectively traverse the pyramid and consider only a small fraction of the included items.

Recall that in Section 3.2.6, we argued that the feature subsets are not all equally important and for that reason we applied a different weight, learned from query logs or user reviews. Figures 3.5a-b), show the distribution that the feature subset weights follow for the cameras and recipes datasets. They portray, for each dataset, the portion of subsets that have a specific weight. For example, in the cameras dataset in Fig. 3.5(a), nearly 30% of all feature subsets have been queried only once, whereas subsets that have been queried approximately 10 times make up for 1% of the entire feature subset set. Clearly, for the cameras dataset, the feature subset weights follow a power-law distribution. A similar tendency can be observed for the recipes dataset, in Fig. 3.5(b). This observation could be taken into account to devise even more efficient algorithms, for the identification of the top-k competitors. Though we have not pursued this direction in this work, we plan to do so in our future work.

Baselines. We compare our CMiner algorithm with two baselines. The first is the Naive approach described in Section 3.2.4. The second is a clustering-based approach that works as follows. First, it iterates over the considered feature-subsets. For each subset $\mathcal{F}'$, it identifies the set of items that have the same value assignment for the features in $\mathcal{F}'$, and places them in the same group. Thus, $\mathcal{F}'$ is mapped to different groups of items with the save value-assignments over its features. The algorithm then iterates over the reported groups. For each group, it updates the pairwise coverage provided to $\mathcal{V}'_{\mathcal{F}}$ by the target item I^* and an arbitrary item from the group (it can be any item, since they all have the same values with respect to $\mathcal{F}'$). The computed coverage is then used to update the competitiveness of all the items in the group. The process continues until the optimal competitiveness scores for all items have been computed. Assuming there are at most M groups per feature-subset, the runtime complexity is O($|\mathcal{I}|$ * M * $|\mathcal{Q}|$). Obviously, when each group is a singleton, the algorithm degrades to the Naive case. We refer to this technique as *GMiner*.

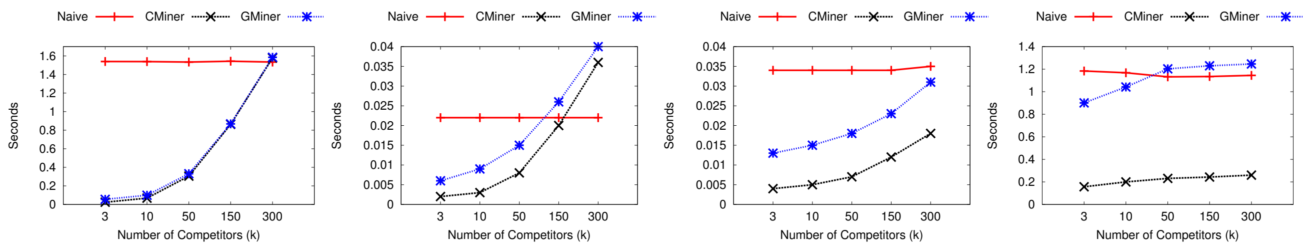
To demonstrate the effect of ordering feature subsets on efficiency, and evaluate the performance of our **IC** ordering scheme presented in Section 3.2.6, we have performed a set of related experiments. Overall, we compare the following ordering schemes:

- **W-ASC**: Ascending order by weight (i.e. $w(\mathcal{F}'))$
- **W-DSC**: Descending order by weight
- **IC**: Descending order by $w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)$

All experiments were run on an desktop with a Quad-Core 3.5GHz Processor and 2GB RAM.

3.2.9 Computational Time

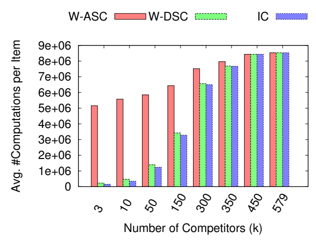
In this experiment we compare CMiner with the two baselines (Naive and GMiner), in terms of computational time. First, we use each of three algorithms to compute the set of top-k competitors for each item in the four datasets. We repeat the process for $k \in \{3, 10, 50, 150, 300\}$,



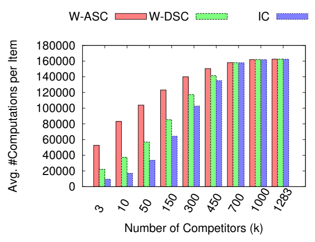
(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Figure 3.6: Average time (per item) to compute top-k competitors for the various datasets

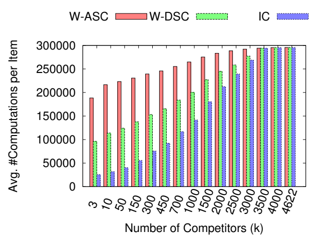
i.e., reasonable values in practical application scenarios. The results for the four datasets are shown in Figures. 3.6a-d). The x-axis holds the different values of k. The y-axis holds the respective computational times (in seconds). We report the average time for each item. The figures motivate some interesting observations. First, the Naive algorithm consistently reports the same computational time regardless of k, since it naively computes the competitiveness of every single item in the corpus with respect to the target item. Thus, any trivial variations in the required time are due to the process of maintaining the top-k set. In general, Naive is outperformed by the two other algorithms, and is only competitive for very large values of k for the `Hotels Dataset`. For the `Cameras` dataset, CMiner and GMiner, exhibit almost identical running times. The similarity between the last two algorithms is due to the very large number of distinct feature-subsets for this dataset, in comparison with the other 3. In particular, this dataset has 14779 different subsets and GMiner identifies, on average, 443.63 groups per subset. This means that the algorithm saves roughly a total of $(579 - 443) \times 14779 =$ 2009944 coverage computations per item, allowing it to be competitive to the otherwise superior CMiner. In fact, for the other datasets, CMiner displays a clear advantage. This advantage is maximized for the `Recipes` dataset, which is the most populous of the four, in terms of included items. The experiment on this dataset also illustrates the scalability of the approach with respect to k. For the `Hotels` and `Restaurants` datasets, even though the computational time of CMiner appears to rise as k increases for the other three datasets, it never goes above 0.035 seconds. For the `Cameras` dataset, the large number of considered subsets has an adverse of the scalability of CMiner, since it results in larger number of required



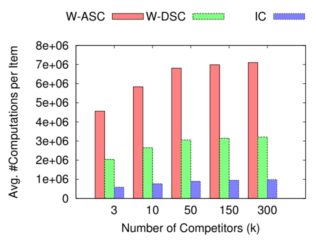
(a) Cameras



(b) Hotels



(c) Restaurants



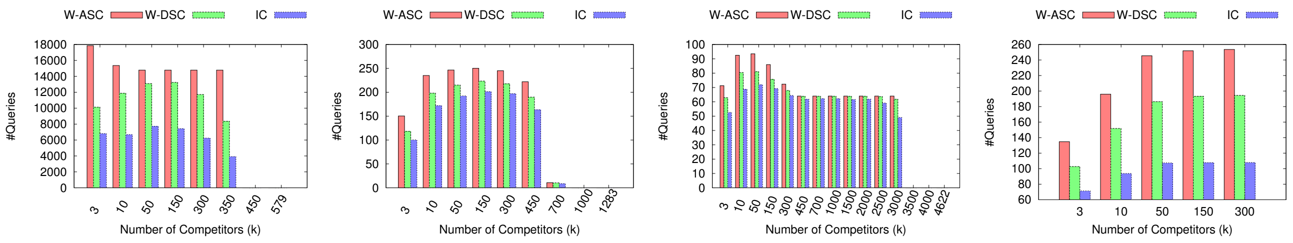
(d) Recipes

Figure 3.7: Average number of computed coverages to find the top-k competitors for the various datasets

computations for larger values of k. Recall that, for our experiments, we considered all subsets that appear at least once. However, our findings motivate us to consider pruning the set of subsets by setting a lower bound on their observed frequency.

Ordering Efficiency. In Fig. 3.7, we evaluate the impact of the ordering scheme on efficiency, for all four datasets, for different values of k. The figures show the average number of computed coverages for every item in the dataset, when finding its top-k competitors. That is, for every feature subset, we store the number of items for which we must update their bounds. Evidently, IC needs to compute a lot less coverage scores than the other two methods -- less than half than the second best in most occasions --, validating in practice Lemma 8. As k grows larger (x-axis), more scores need to be computed for all techniques, reaching the maximum value when k is equal to the dataset size (last set of bars). This is expected, since setting k equal to the dataset size asks that we fully rank all items in the dataset. Note that fewer coverage computations imply that more points are discarded in each iteration, inside the updateTopK procedure of our CMiner algorithm.

One could argue that IC prunes away some points initially, but performs as many iterations as the other two ordering schemes, which would also result in fewer coverage evaluations. For this reason, we have also plotted the graphs in Fig. 3.8, depicting the average number of feature subsets that need to be processed within the updateTopK procedure, before each scheme falls back to a sequential scan on the (at most) k remaining items (lines 33-36 in Alg. 3.2). The fact that IC examines fewer queries (y-axis) is a clear indication of its faster con-



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Figure 3.8: Average (per item) #evaluated feature subsets, before sequentially computing competitiveness scores

vergence and, consequently, more aggressive -- but correct nonetheless -- pruning. Therefore, it reaches the number k of requested competitors, and breaks out of the loop, sooner than the other two alternatives. Given that within the updateTopK we only alter the ordering scheme, the procedure's input and output are the same across the three alternatives. Therefore, the difference in performance can only be attributed to the ordering scheme, demonstrating the superiority of IC.

A final remark we should make from Fig. 3.8 refers to the fluctuations in the number of queries, as we vary the number of competitors k. The number of queries that CMiner will check depends largely on the number of given points, during each iteration of the updateTopK method. More specifically, it depends on whether the provided points exceed the requested number k. Take the `cameras` dataset for instance. Up to the point that k = 350, IC needs to process, approximately, 4000 queries. However, after that point, the number of queries it uses to prune away points drops dramatically (almost 0). The reason is that the first skyline layer of the cameras dataset (Fig. 3.4) contains about 380 points. Therefore, when we request k = 400 points, the full skyline layer will be scanned, reducing significantly the number of bound checks CMiner performs. The same holds for the other datasets.

3.2.10 A User Study

In order to validate our competitiveness paradigm, we conduct a user study as follows. First, we select 10 random items from the Digital Cameras corpus. We refer to these 10 items as the *seed*. For each item I^* in the seed, we compute its competitiveness with every other item in the corpus, according to our definition. We refer to our approach as `CMiner`. We also rank all the items in the corpus based on their distance to I^* in the feature-space. The L_1 distance was used for numeric and ordinal features, and the Jaccard distance was used for categorical attributes. We refer to this as the `NN` approach (i.e. Nearest Neighbor). We then chose the two items with the highest score, the two items with the lowest score, and two items from the middle of the ranked list. This was repeated for both approaches, for a total of 12 candidates per item in the seed (6 per approach). We then created a user study on the online survey-site `kwiksurveys.com/`. In total, the survey was taken by 20 different human annotators. Each of the 10 seed-items was paired with each of its 12 corresponding candidates, for a total of 120 different pairs. The pairs were shown to the annotators in a randomized order. The users were also given access to a table including the values of each item in the pair for every feature. For each pair, the annotator was asked whether he would consider buying the candidate instead of the seed item. The possible answers were "YES", "NO" and "NOT SURE". The results are shown in Figure 3.9.

The y-axis holds the percentage covered by each approach. The figure shows 3 pairs of bars. The left bar of each pair corresponds to our approach (CMiner), while the right bar to the NN approach. The first two bars from the left represent the responses of the users to the top-ranked candidates for each approach. The two bars in the middle represent the responses to the candidates ranked in the middle, and, finally, the two bars on the right represent the responses to the bottom-ranked candidates. Each bar captures the fraction of each of the three possible responses. The lower, middle, and upper part of the bar represent the "YES", "NO" and "NOT SURE" responses, respectively. For example, the first bar on the left, reveals that about 90% of the annotators would consider our top-ranked candidates as a replacement for the seed item. The remaining 10% was evenly divided between the "NO" and "NOT SURE" responses.

The figure motivates some very interesting observations. First, we observe that the vast majority of our top-ranked were identified by the annotators as possible replacements for the seed item. These are thus verified as strong competitors that could deprive the seed item from potential customers and decrease its market share. On the

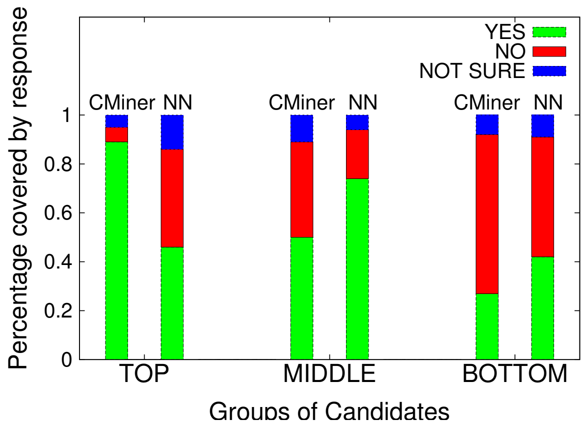


Figure 3.9: Results of the user study comparing our competitiveness paradigm with the Nearest-Neighbor approach.

other hand, the top-ranked candidates of the NN approach were often rejected by the users, who did not consider these items to be competitive. The middle-ranked candidates of our approach attracted mixed responses from the annotators, indicating that it was not straightforward to determine whether the item is indeed competitive or not. An interesting observation is that the middle-ranked candidates of the NN approach were more popular than its top-ranked ones. The interpretation is that this approach fails to emulate the way the users perceive the competitiveness between two items. Finally, the bottom-ranked candidates of our approach were consistently rejected by the annotators, verifying their lack of competitiveness to the seed item. The bottom-ranked items by the NN approach were also frequently rejected, indicating that it is easier to identify items that are *not* competitive to the target.
In conclusion, the survey demonstrated the ability of our paradigm to capture the competitiveness between two items. Further, our approach consistently outperformed an intuitive baseline, indicating that the task is non-trivial.

3.3 Summary

In this section of the thesis, we presented a formal definition of the competitiveness between two items. Our formalization is applicable across domains, overcoming the shortcomings of previous approaches. We consider a number of factors that have been overlooked by previous approaches, such as the position of the items in the multi-dimensional feature space, and the preferences and opinions of the users. A user study was conducted to verify the validity of our notion of competitiveness. Based on our competitiveness paradigm, we addressed the problem of finding the top-k competitors of a given item. The proposed framework is designed to be efficient and scalable, in order to be applicable to domains with large populations of items. Our methodology was evaluated via an experimental evaluation on real data from different domains.

Chapter 4

Mining user-generated content

4.1 Introduction

Despite the prevalence of online reviews and their economic power, the most defining online service of the contemporary web is that of social media. Users actively participate in online networking sites, and there is a plethora of these services, covering a wide spectrum of domains. MySpace [1] is one of the first networking sites targeting musicians, while Flickr [2] aims at photographers and LinkedIn [3] focuses on professional networking. There are also general purpose social networks, mostly used for leisure, with Facebook [4] and Twitter [5] being the most characteristic examples.
A major difference between these sites and other online services is that the networking component is an integral part of the service itself. For example, web forums -- the closest example of online user communities prior to social media -- organized information hierarchically by topic, subtopics, discussions (threads) within a subtopic and posts made by the users. In other words, the primary focus was placed on the written content itself and in the organization and retrieval of information, and only secondarily, if at all, to the users posting it. This is far from surprising, considering that web forums were primarily a tool to help other users with similar problems (e.g., technological forums).

[1] https://myspace.com/
[2] http://www.flickr.com
[3] http://www.linkedin.com
[4] http://www.facebook.com
[5] http://twitter.com

For the same reason, there was no notion of *following* a user in order to receive updates regarding what they post.

Contrary to these platforms, online social networks make the user a first class citizen, and connections are their driving force. Users may interact with each other, engage in discussions and talk about their interests. In the end, the information that they will receive depends greatly on who they have connected with. It is particularly interesting that there is a strong interplay between network structure (i.e., user connetions) and topical similarity between users [177].

However, the social component is not the only difference between social networks and earlier online platforms. For example, there has been a major shift in the users' attitude towards privacy in online settings. In particular, users tended to identify themselves through nicknames and aliases in earlier years. On the contrary, they now tend to use their real names, and, in fact, feel comfortable with doing so. The number of active users are also largely apart. More specifically, online social networks have hundreds of millions of active users (some up to a billion), as opposed with the hundreds of thousands or few millions users of earlier online platforms (e.g., web forums). Another difference is that the user base is very diverse in terms of demographics, i.e., location, age, language, as well as discussed interests, which has also been pointed in past research [139, 153]. Finally, as a result of the aforementioned factors, and on occasions due to certain platform restrictions, user discussions tend to be more conversational, rather than individuallistic, as is the case with blogs. Given the large user base, this leads to a continuous, real-time flow of information.

With the advent of these new platforms, several research opportunities arise. The networking component alone may be the subject of research questions, such as community detection, outlier detection, spammer identification, detection of common / frequent structural patterns, graph similarity and so on. Most of these problems stem from the modeling of the social network as a graph. Moreover, by allowing users to express themselves more freely, without tying their participation to the service to a specific reason (e.g., seek for help), new research problems can be formulated and solved. User profiling, personalization and recommendation are some well-studied problems in the literature, even outside the social media domain. Finally, the nature of the medium (real-time information flow) requires the use of fast techniques, that can cope with the user-generated data streams. Particularly interesting, however, is the fact that online social networks are a prolific area for interdisciplinary reserach. The fact that people are more open towards these services, interact with one another on a regular basis, and share content online regarding their everyday lives allows sciences very distinct from computer science, such as sociology, psychology and medical science, to propose new hypotheses, develop new theories or refine existing ones and, most significantly, test these ideas at a large scale. The latter is very important for these sciences, given that, until now, collecting the necessary data to test ideas was a tedious process. As a result, algorithmic approaches have been proposed to test these theories with real world data, that are now readily available. This has led to the generation of new disciplines such as:

- *computational psychology*: The use of algorithmic techniques to address questions from psychology. Characteristic examples include the identification of personality traits [171], understanding a user's emotional state from what they write, as well as the effect of external stimuli on their emotional state (known as *emotional contagion*).
- *computational sociology*: The use of algorithmic techniques to understand how people interact with each other within communities, who they connect with, who they talk to, how communities are formed over certain topics (e.g., politics) and their structural properties, role identification of certain users in a community, etc.
- *health care informatics*: The use of algorithmic techniques to support health care, identify health problems by location and their spreading [67, 21] or food-related outbreaks [22]. Such application scenarios are characteristic of the Big Data era.

It is easy to see that these research opportunities would not have been made possible without the proliferation of social media. For example, computational sociology relies heavily on the underlying network structure, which is an integral part of these services, unlike past settings. Similarly, computational psychology creates user profiles by utilizing all possible generated content (and social ties), which is primarily driven by the user's need to express themselves, and not by a need to search for information. The last discipline covers a much broader spectrum, such as organization and retrieval of medical records, privacy-preserving techniques, etc. But there is a rising trend within it, to use more and more social media information, because of its real-time nature and cost effectiveness [104].

4.1.1 Research questions in Online social networks

From the previous discussion, it is easy to understand that there are endless possibilities for research questions in social media. As a result, it would be practically impossible to consider all of them in a dissertation, much less in a single chapter. Therefore, we will only consider a small subset. Moreover, we will focus on Twitter, a prevalent social media service. The service is generic in the sense that users may talk about anything, is very diverse in both topics and users, and imposes certain restrictions that make it unique.

In particular, the chapter is organized in a way that each section considers and addresses a different problem that may arise in such a setting. Section 4.2 considers principled techniques to collect, process and analyze the data from social media. More specifically, we present the architectural design and implementation details of a crawler for the Twitter service, one of the most prevalent social media sites. We also present techniques to harvest data from query-based interfaces, that take into account the ranking of the returned results. Finally, query execution engines have already been proposed for streaming data, accepting queries in declarative languages (SQL variants), and executing them efficiently, as they take into account the domain's particularities. When such engines are in place, it is time consuming, inefficient and error-prone to build data mining techniques in an ad-hoc fashion. As an alternative, we propose to accommodate them within the declarative language, and present an approach that allows them to be optimized by the execution engine as well. These ideas are applicable in any domain where a query execution engine is used, such as social media, sensor networks and relational settings.

One of the reasons why Twitter has been so well received by the research community is its data policy towards researchers. In particular,

Twitter provides Application Programming Interfaces (APIs) to retrieve information of interest, including both content and the social component. Moreover, it provides access to the stream of user generated content with various sampling rates. Therefore, researchers have access to a wide range of the platform's content. Regarding the provided samples, the service offers two different schemes: a 1% sample and a 10% sample, which is only provided upon request. Access to the full stream (100%) is also provided, but only with a fee. Therefore, the focus of Section 4.3 is to compare the two sampling ratios, and identify their differences, other than the data volume.
Finally, Twitter has been characterized as a news reporting site [124], due to the real-time nature of the information that its users generate. Given the high diversity of the service's users, discussed events cover a wide spectrum as well. For these reasons, in Section 4.4 we present algorithmic solutions to identify (newsworthy) events, using Twitter data as our only source of information. We discuss particular subproblems that arise in such a setting, and propose efficient solutions, to cope with the real time nature of the data. Moreover, we present the architectural design of our system, while also describing the data flow between components. Section 4.5 concludes this chapter, and summarizes our basic findings.

4.2 Towards principled solutions for mining online content

4.2.1 Faceted crawling of the Twitter service

Numerous prototypes have been built on top of social networks, recognizing their importance in a variety of occasions [124, 96, 221, 205, 116, 91, 89, 29, 32]. Depending on their objective, these applications rely on different key properties of the underlying data. For example, in the *Big Data* era *voluminous* data is the norm. *Emergency management* and *location based services* require access to *locational* information [14, 204, 76]. *News reporting* must have acces to *newsorthy* and *high-quality* information from *high-quality* posters. *Social graphs* are another major asset, integral to computational sociology, but also important for community detection, node importance identification and

information diffusion, to name a few.
Building the infrastructure that retrieves the desired information is both time consuming and technically challenging. For example, the Twitter service offers two distinct *Application Programming Interfaces* (APIs) to retrieve data, with distinct limitations: Access is restricted to authenticated (i.e., registered) users, and to public tweets, i.e., tweets visible to anyone. The default *streaming* API returns only 1% of public tweets [6], while the *REST* API limits the number of requests issued within a specific timeframe, imitating the politeness principle [58].
To address these problems, in this thesis, we also consider the problem of an efficient crawler for the Twitter service, to fetch content with desired properties. Those properties refer to different *facets* of the data (tweets, users, graph, location, etc.), giving rise to the *faceted crawling problem*. We present our system's design and implementation details, and provide experimental evidence of our crawler's performance. We also discuss lessons learned from our interaction with the service's APIs.

4.2.1.1 Related Work

Harvesting web documents is as old a task as the web itself. Search engines rely on web crawlers [45, 49, 106] to fetch online documents, which they subsequently index and make available. These are built for the *surface* web, where webpages are reachable through hyperlinks. However, Twitter's multiple information *facets* do not allow a straightforward modification of existing crawlers, which would also violate the *politeness policy*, given the *real-time* nature of the medium. Similar problems exist for crawlers of the *Hidden Web* [40, 85], where information can be accessed through query forms. Given that we discuss Hidden Web techniques in a subsequent section, we do not elaborate more on this subject here.
Several libraries exist [6], to access Twitter's APIs programmatically, one resource at a time. Therefore, these are not complete solutions, whereas we facilitate *faceted* crawling through the *crawl flow*.
The work in [9] also discusses *facets* on Twitter, but in a conceptually different way. More importantly, our research goals are different: [9]

[6]Elevated access can be granted for a fee.

is interested in enriching tweets with "context", whereas we aim at the implementation of an efficient and robust crawler. Therefore, [9] can be thought of as an application, that one can build on top of our proposed infrastructure.

4.2.1.2 Twitter API Background

Twitter provides two main Application Programming Interfaces (APIs) to access publicly available data, i.e., data that anyone can see: There is i) a REST-ful one, which probes the service with HTTP requests, and ii) a streaming API, which resembles a publish-subscribe mechanism. In both cases, the user can apply filters, to restrict the information they are looking for. In both cases the user needs to be authenticated through one of the available options. In the next paragraphs, we give a more detailed overview of these two APIs.

REST API. The REST API uses HTTP requests (i.e., *GET*, *POST*) to perform the communication between the end user and the Twitter service. This API supports multiple query types, each of which can be employed by contacting a carefully constructed URL, with all the necessary information.

From the REST API specification [4], we identify four types of restrictions, which we must take into account in our crawler. Table 4.1 gives additional details.

- **Rate restrictions**: The number of queries of a specific type that the developer can issue within the 15 minute window.
- **Maximum Result Size**: The upper bound on the results of a particular query. For instance, even if a user has posted 5000 tweets, we are only able to access the most recent 3200.
- **Probing Result Size**: The number of results that we can retrieve each time we probe the service with that particular query. For instance, a query for a user's timeline will return *at most* 200 tweets.
- **Maximum Query Size**: The number of objects that we can query simultaneously with a single probe to the service. Typically this is 1, (e.g., 1 tweet each time, using its id), but there are some exceptions (e.g., *lookup* at most 100 users).

Table 4.1: Restrictions for some major query types.

Query	Rate	Max Result	Probe Result	API limit
User Lookup	180	∞	100	100
Tweet Show	180	1	1	1
Friends	15	∞	5000	1
Followers	15	∞	5000	1
Timeline	180	3200	200	1
Retweets	15	100	100	1

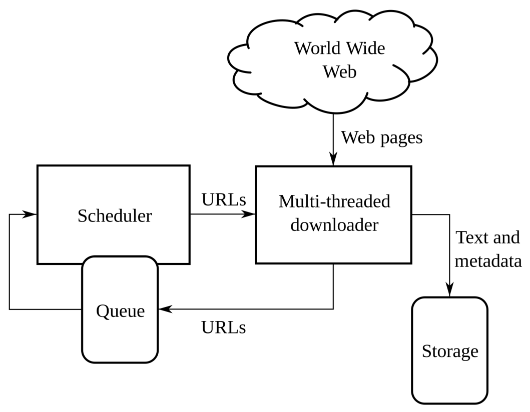
Streaming API. Through this API, one can receive data as a *flow* of tweets. The API returns a 1% sample of all public posts, though not uniformly [155]. Consequently, data received through this API may reflect fluctuations of the actual stream, e.g., increase / decrease of posts, temporal patterns of user interactions, etc. A drawback of this API is that it can not be used for all information facets, e.g., it does not return the social graph.

4.2.1.3 Faceted Crawler Architecture

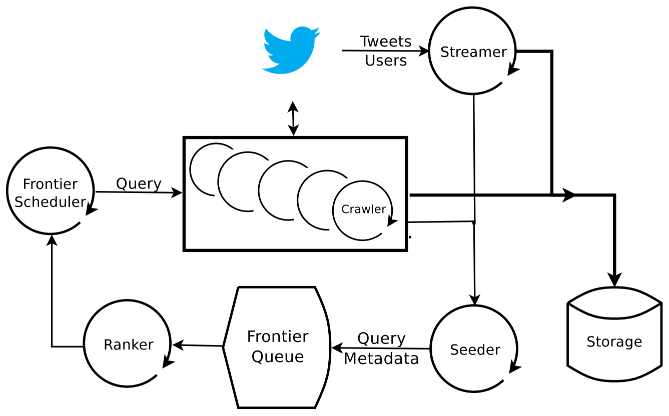
Figure 4.1 shows the architecture of a classic web crawler [49] on the left, compared against our Faceted crawler architecture, on the right. The two designs appear to be similar for the most part. The contents of the frontier queue, however, in the two cases are different, because surface crawlers need only handle URLs of the next pages to fetch. On the contrary, our crawler needs to handle different query types, each of which takes different parameters.
The components Seeder, Ranker and Streamer are also different. The latter exists to harvest data using the *streaming* API. The Seeder allows support for various applications in a unified way. The Ranker is separate from the scheduler to simplifyc application development, and to combine multiple query types.

Streaming API. The Streamer component exists to obtain information from the `Streaming` API, as shown in Figure 4.1. The component opens a connection to the service, receives the stream of tweets, and may forward it for processing, storage and seeding.

REST-based crawling. Aside the Streamer, the components in Figure 4.1 largely resemble a classic crawler architecture. However, the



(a) Classic Web Crawler



(b) Twitter Faceted Crawler

Figure 4.1: Architectural designs of both classic web crawler and our Twitter faceted crawler

actual design is quite different. We have already pointed out the difference in the frontier queue. Moreover, each application may have a different crawling process, therefore we need to efficiently multiplex queries. For this reason, we have decoupled crawling (i.e., accessing the service) from seeding.

Scheduler. A major component of our system is the Scheduler, responsible for queueing crawl tasks. A crawl task contains information about the query type and all of the parameters that accompany it. The

Algorithm 4.1 Scheduler Algorithm

```
    Input: Database $db$, Ranker $ranker$
    Output: $outQueue$
    Shared Queue $queue$, $timedQueue$

    //Main Thread
 1: while !$stopped$ do
 2:     $qry \leftarrow queue$.dequeue();
 3:     $data \leftarrow ranker$.getNext( $qry$ );
 4:     $outQueue$.enqueue( $qry$, $data$ );
 5:     $db$.store( $qry.qryMeta$ );
 6:     $timedQueue$.enqueue( $qry$, NOW + $qry$.ival );

    EventTrigger()
 7: $nextQuery \leftarrow timedQueue$.dequeue();
 8: $top \leftarrow timedQueue$.top();
 9: $queue$.enqueue( $nextQuery$ );
10: resetTimer( $top$.TIME - NOW );
```

Scheduler is also responsible for enforcing the rate limits. To achieve this, the component operates in an event-driven manner, shown in Algorithm 4.1.

The component starts with the query types of the crawling process. It will enqueue these queries to a $timedQueue$, and will trigger the event, leading to the execution of the "EventTrigger()" method. The queries that triggered the alarm are dequeued and passed to the $queue$ for crawling. We then reset the timer to trigger for the next query item. Items in $queue$ are processed one by one, by the main scheduler thread. For the current query type, we probe th "Frontier Queue" (Line 3), implemented as a database relation, and pass the result for crawling. The scheduler stores metadata in the database (e.g., statistics) and requeues the query for timely execution, computed through its rate limit.

Ranker. As seen in Algorithm 4.1, the scheduler relies on a `Ranker` object. A `Ranker` implements our `IRanker` interface, shown in Figure 4.2. The `init()` method is used to properly initialize resources (e.g., database relations). The *getNext()* method returns the next item to submit to a crawler as our next query. The *id* is decided by the scheduler to simplify the architecture. The *query* also contains the `RateLimit` information. This allows for a common interface across queries.

```
public interface IRanker{
  public void init();
  public List getNext( long qid, Query query );
}
```

Figure 4.2: The IRanker interface

Seeder. As shown in Algorithm 4.2, the Seeder operates in an endless loop, much like the Scheduler. It receives information from the result queue $\mathcal{RQ}$ (Line 2), where crawlers write the result of probing Twitter. Results are forwarded for storage (line 3). We then update the frontier in two steps. First, update the current query (line 4). The result is a boolean value that determines whether we should move to the second step, i.e., update subsequent frontiers (lines 5-8).

The QueryLog Relation. To restart after a (forceful) shutdown, and monitor our system's performance, we store appropriate information,

Algorithm 4.2 Seeder Algorithm

```
Input: Database $db$, ResultQueue $\mathcal{RQ}$, CrawlFlow $\mathcal{CF}$

while !$stopped$ do
    ($result$, $qry$) $\leftarrow$ $\mathcal{RQ}$.dequeue();
    storeResult( $result$ );
    update = $\mathcal{CF}$.stepSeeding( $qry$, $result$ );
    if (update) then
        $nxtQrs \leftarrow \mathcal{CF}$.nextQueries( $qryMeta$ );
        for ( $i$ = 0; $i$ < $nxtQrs$.size; $i$++ ) do
            $nxtQrs$( $i$ ).stepSeeding( $qry$, $result$ );
    $db$.store( $qry.qryMeta$ );
```

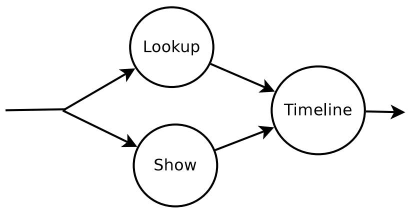
in a relational table, called QuerLog. A partial view is shown in Table 4.2. Statistics of the system include rank time, seed time, etc (not shown here). Fields **qid** and **result** are straightforward. Values of the **result** field can be found in [5]. The next three fields ensure that the rate limits are enforced in cases of failure or restarts.

Table 4.2: Fields of the QueryLog relation

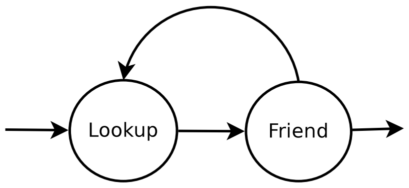
Field	Description
qid	Unique Identifier for the Query
result	Code Signifying how the query
toq	The type of query associated with this tuple
crawler	An identifier for a crawler
tssched	Timestamp when this query was scheduled

Crawl Flow. To further simplify the crawling process, we introduce the concept of a Crawl Flow. The idea is that on Twitter, a crawl is driven by the underlying application, which can be generally expressed as a sequence of faceted probes (with cycles). The Crawl Flow can be thought of as a state automaton, and defines the sequence of the queries to the service. Figure 4.3 shows a schematic representation of two Crawl Flows that we provide, a user's timeline, and sampling the social graph. Through the Crawl Flow, the user specifies:

- The general execution sequence of queries. The sequence may contain loops (including self-loops), depending on the goal.
- An object implementing the IRank interface.



(a) Timeline



(b) Sampling

Figure 4.3: Schematic representation of Crawl Flow examples.

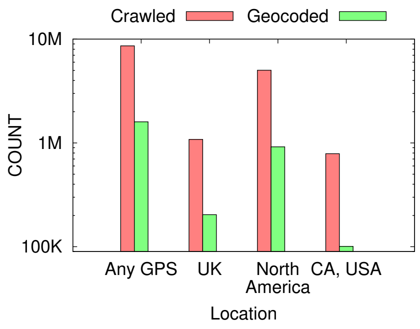
- An object implementing the ISeed interface.

4.2.1.4 Use Cases

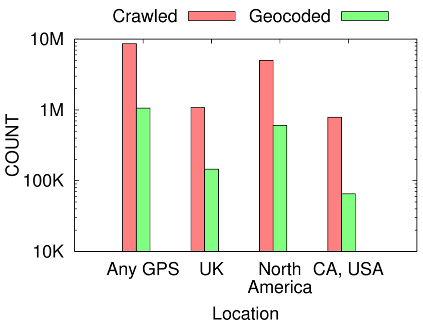
We have fully implemented our crawler in Java 1.6 and used the Twitter4j library [7] for method probes. We used PostgreSQL 8.4 with its default configuration, but any SQL-compliant database will work. Each component runs on a separate thread. Our experiments were run on an Ubuntu Linux 64bit machine with a quad core @3.4GHz and 16Gb of RAM, setting only half of it as Java's heap space.

Crawling by Location. Location is a very important aspect of tweets. Tweets with location can be retrieved through a geographical filter, specified as 2D bounding box with GPS. Despite its accuracy, GPS is not the sole approach to geocode data. External geocoders [204] can be applied directly to the streaming API. Figure 4.4 shows the number

[7] http://twitter4j.org



(a) Tweet Counts



(b) User Counts

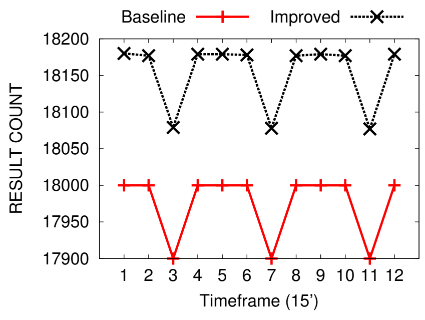
Figure 4.4: Comparing raw counts between crawled and geocoded locational information

of tweets (on the left) and users (on the right). Custom geocoding (Geocoded) can extract an additional 10% to GPS-filtered information (Crawled).
Changing the bounding box of a crawl returns different results, even when the boxes overlap. Table 4.9 depicts the similarities in terms of received users (upper right, in red) and tweets (lower left, in blue), where each crawler is configured to monitor the corresponding location. Despite some commonly shared users and tweets, a lot of new content is being delivered by each stream.

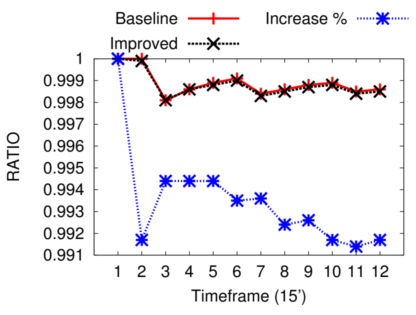
Table 4.3: Jaccard Similarity between the GPS enabled crawls using the Streaming API

	Any GPS	UK	N Amer	CA, USA
Any GPS	1.0	0.069	0.249	0.038
UK	0.057	1.0	6×10^{-4}	0.001
N Amer	0.218	0.0	1.0	0.138
CA, USA	0.042	0.0	0.145	1.0

Crawling Basic User Information. User timelines are useful in several cases, e.g., behavior analysis. To efficiently crawl a user's timeline, it is best to know the number of expected results. We improve user information extraction by 1%, as seen in Figure 4.5(a), through the combination of two query types. Figure 4.5(b) depicts this improvement (blue line) as a percentage. The 1 less query out of every 4, shown in Figure 4.5(a), is due to our best-effort approach for crawling the service, which has tight time constraints.
Crawling the Timeline. Figure 4.6(a) shows how much time is spent

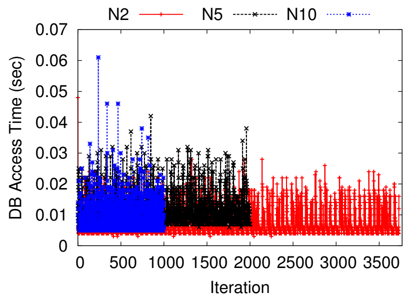


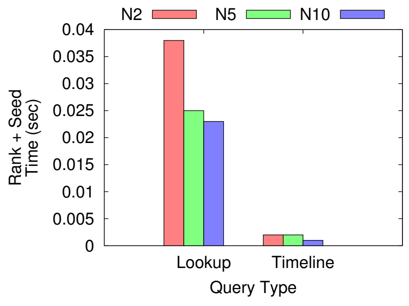
(a) Results Received



(b) Improvement %

Figure 4.5: Crawling comparison for basic user information


(a) Avg. getNext() for Lookup


(b) Avg. Rank+Seed time

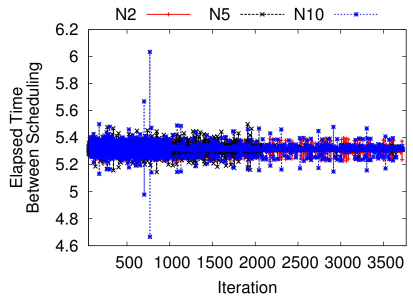
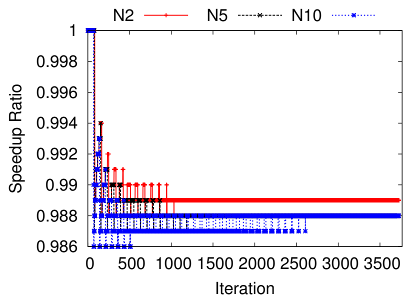

(c) Avg. Elapsed Time for Scheduling Timeline

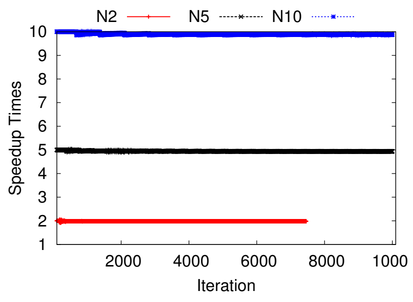
Figure 4.6: Average crawler performance for harvesting (a) user information and (b)-(c) user timelines.

on the *getNext* method by the Ranker component, which is below 10ms. Both the *getNext* and *stepSeeding* methods do not take, on average, more than 4ms (Figure 4.6(b)).

The average elapsed time between consecutive schedulings of Timeline queries is shown in Figure 4.6(c). The interval does not increase, but stabilizes over time. Even with multiple crawlers, our system maintains its performance. With 10 crawlers, we are at more than 98.5% of the optimal case for Timeline queries (Figure 4.7(a)) perform similarly for Lookup queries (Figure 4.7(b)). Evidently, more crawlers yield more results faster (Figure 4.7(c)).

Sampling. For large networks, sampling is important. We have implemented the Metropolis-Hastings algorithm [196], through the IRank and ISeed interfaces. Instead of *thinning*, we use reservoir sampling


(a) Avg. Timeline Query Optimal Ratio


(b) Lookup Speedup

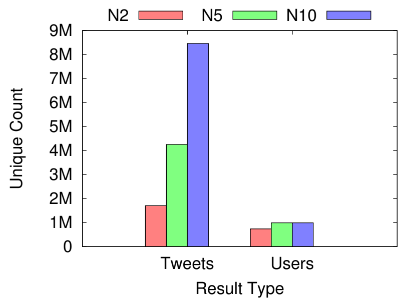
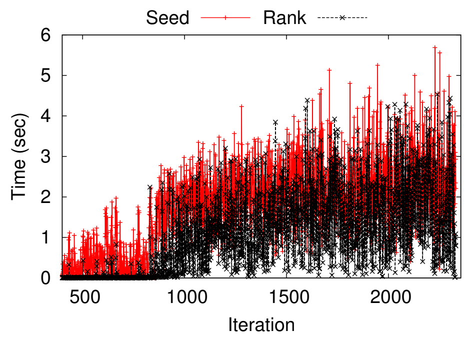

(c) Volume of crawled Information

Figure 4.7: Collective performance of the crawlers used for harvesting user timelines.

on the collected nodes, which has the same effect. Figure 4.8a) shows the time required for ranking and seeding. This is the only case where the timings are high, compared to other use cases, and eventually reach an average of ∼2.5 seconds. The reason is that sampling, synchronizes on shared resources. Regardless, our implementation is well within the timeframe between subsequent queries of this type (60 seconds).



(a) Runtimes

Figure 4.8: Sampling scenario statistics.

Retweet Graph. Retweets are a key concept in Twitter, allowing users to re-post / endorse tweets of others. They can be used to identify cascades of information, leaving the actual reason as a latent feature to be explored. Crawling retweets has been implemented through appropriate seeders / rankers in our framework.

Retrieve tweets by ID. Per Twitter policy, one may only disclose the tweet IDs of their dataset. Therefore, this use case becomes very important for reproducibility of results and fair comparison of techniques.

Crawl Social Graph. We have also implemented a BFS traversal of the social graph of Twitter, through Seeders and Rankers.

4.2.2 Harvesting content from Hidden-web databases

Although users have become accustomed to surfing the web by following hyperlinks, during the early years of this millenium, it became apparrent that not all of the web information is accessible this way. More specifically, early reports [40] suggested that there is an ever increasing amount of information *hidden* behind Web search interfaces

and query forms. This information was not only estimated to be much larger than the *surface* web, i.e., the portion of the web accessible through hyperlinks, but also very high in quality. Consequently, researchers focused on techniques to retrieve this content, which is collectively known as the "Hidden Web".
Harvesting content from the Hidden Web originally focused on structured information stored in relational databases, but soon enough extended to accommodate textual information [159, 211, 114]. Moreover, despite the techniques being developped to collect as much of the information as possible and to subsequently make it available through a search engine's index, they can be used in any occasion where a search field exists. For example, they can be applied in a social media context to collect almost all information regarding a topic of interest.
One thing that we should consider when crawling a Hidden Web site is the ranking of the returned results. Most techniques assume that when probing a Hidden Web site for information, the full information will be retrieved. In practice, however, only a subset of the results can be obtained, and which subset depends on how results will be ranked by the search engine. Previous techniques did not account for this fact. Given that the actual ranking function of the Hidden Web site is unknown to us, as well as the features used, we propose techniques that take into account the ranking of the returned information in a generic way. In short, our contributions in this thesis are the following:

- We extend previous algorithmic techniques that cover the content of a Hidden Web site while taking ranking into account. Our proposed approach is generic and uses domain-independent features, so that it may be applied to numerous Hidden-Web sites without major modifications.
- Since our technique determines which queries to use based on the retrieved ranked results, we show that it manages to achieve better coverage performance than previously existing techniques.
- We provide an experimental evaluation of our proposed technique, by crawling YouTube. An interesting outcome of the evaluation is that our techniques can be very easily parallelized in order to crawl Hidden-Web sites.

The rest of this section is organized as follows: We start with related work on the topic in Section Section 4.2.2.1. Section 4.2.2.2 formally defines the problem, followed by Section 4.2.2.3 which introduces our proposed approach. Our experimental findings are presented and discussed in Section 4.2.2.5.

4.2.2.1 Related Work

Crawling Hidden Web sites is related to several fields pertaining the web, such as Web Mining and Web IR. The existence of the Hidden Web (HW) was brought to light in the early years of this millennium [40]. Research on the subject has focused on various aspects, such as understanding query forms [85, 186, 235], classifying the sites based on their content [115], accessing it [105, 70] and searching the surface web to discover HW entry points [211]. Our work focuses on surfacing HW content, i.e. retrieving content from those sites, so that it can be indexed, thereby it relates to the works in [159, 137, 147]. The general technique in these works, as in ours, is to probe queries to the HW site, retrieve the actual content and then select the next query with which to probe. These techniques focus on coverage, i.e. retrieve as big a portion of the site's content as possible. The key difference is that they do not take the ranking aspect into account.
Ranking has been extensively studied both within and outside the scope of web-related disciplines. Its importance is significantly higher in the web domain, due to the size of the web and the fact that users rarely view more than the top ranked documents. It is, therefore, an ever-going research topic, with various settings and applications. To rank documents on the web, several approaches have been proposed, such as machine learning [175], link structure [45], web content analysis and anchor text. Our current work differs in that we do not aim at building a new ranking function for HW sites. Rather we are interested in taking ranking performed by HW sites into account, while retrieving their content. Ranking in the context of HW sites has been studied in [20], where the authors want to identify the order in which to probe sources, in order to provide users with appropriate information. Their work differs from ours in that we are interested in the ranking performed by HW sites *per se*, not to globally rank the HW sites themselves with respect to a user need. Moreover, the authors

in [20] perform sampling of HW sites, as an external step of their approach, whereas our primary goal is to crawl HW sites and crawling is an integral part of our strategy.
Finally, our employed approach relates in notion to the active learning paradigm [189]. Active learning is based on the idea that a classifier "*may achieve higher accuracy if it is allowed to choose the next training label from which to learn*". To do this, a measure of the error that is introduced by a potential label is required. Then, the label that is expected to maximize the gain is selected and used to train the classifier. Although existing crawling algorithms for HW sites mostly rely on mathematical models to choose their next query, simplifying our technique's intuition, our work relates in the following manner: we choose the term for which we can not accurately predict its global ranking, had we used it to probe the database. Nevertheless, our work differs from active learning approaches, as we do not use a machine learning classifier, nor is it our primary concern to learn the ranking function of each HW site. Moreover, we are interested in coverage as well. Finally, to the best of our knowledge, active learning approaches have not been used in the context of Hidden Web sites.

4.2.2.2 Background

We start by formally defining the problem of ranking-aware Hidden-Web crawling and our goals. To make our discussion more concrete, we assume that a Hidden-Web site is associated with a database D, containing documents $D = \{d_1, d_2, ..., d_{|D|}\}$. We represent each document d_i within D as a bag of words, i.e. $d_i = \{w_i^1, w_i^2, ..., w_i^{|w_i|}\}$. Similarly, we represent the set of unique terms contained by all documents in D by $T = \{t_1, t_2, ..., t_{|T|}\}$.
We define $D_q \subseteq D$ to be the set of documents retrieved by query q. These documents are the *relevant* ones w.r.t. q as they were returned by the Hidden-Web site. Let R be a *ranking* (permutation) of D_q, i.e. $R = R(q, D_q) \rightarrow \{1, 2, .., |D_q|\}$. Note that we are only interested in the *rank* (position) of each document. Also each document appears only once in a ranking. Based on this notation, we can now formally define our goal in rank-aware crawling of a Hidden-Web site:

Definition 4 *Given a Hidden-Web site, identify the minimum set of queries that covers the content from the site that allows us to approximate the ranking of all potential queries.*

Intuitively, we want to retrieve as much content from the Hidden Web site as possible, while maintaining the queries probed to the site at a minimum. At the same time, we are interested in the ranking that each query is associated with. Since the set of unique terms in D is T, there are $|T|$ rankings for single-keyword queries, and $2^{|T|}$ combinations of queries. However, since keyword search commonly employs *AND* semantics which restrict the results, we can achieve better coverage by issuing only single keyword queries. Therefore, since coverage is a basic goal, we want to issue fewer queries than $|T|$ while being able to approximate the ranking of any $t_i \in T$ that has been retrieved so far.

4.2.2.3 Rank-Aware Crawling

In this section we develop and describe our proposed metric, to show how we may include the ranking performed by a Hidden-Web site as a paremeter while crawling it.

Efficiency. As we already discussed in the previous section, a fundamental goal of our techniques is to minimize the download cost when crawling a Hidden-Web site while achieving high coverage. Similar to [159], in order to compare among crawling techniques, we use the notion of *efficiency* for a query term t, defined as:

$$Efficiency(t) = \frac{P_{new}(t)}{Cost(t)}$$

where P_{new} is the fraction of new items (over all current items) that term t is expected to retrieve and $Cost$ is the overall cost associated with issuing term t, measured for example in money, bandwidth, or communications between the crawler and the Hidden-Web site. In [159] the authors use the $Efficiency$ metric described above to determine which queries the crawler should issue to the Hidden-Web site. Their goal is to maximize the coverage of the site using the minimum number of queries. Since we are also interested in coverage we will be using the $Efficiency$ metric as well. However, as we also aim at approximating the ranked results of the queries coming from a Hidden Web site, we will extend this metric by considering the "ranking gain" of the query term as discussed next.

Ranking Gain. Apart from the coverage aspect, we are also interested in the ranking associated with each term. First, we describe the intuition behind the ranking gain of a term: it is a measure of the degree to which we can approximate the term's ranking, i.e. the ordering of results we would obtain by querying this term. In other words, we evaluate how similar a *derived* ranking would be to the actual one. If we are able to accurately reconstruct the ranking of a term, then that term's gain (*w.r.t.* ranking) would be close to 0. On the contrary, if the derived ranking introduces a big error in our attempt to approximate the actual one, then, we would need to query this term. The need to correctly derive the ranking of a term is important for several reasons, both for single and multi-keyword queries. Finally, we note that already queried terms have a ranking gain of 0, as their actual ranking is known.

Therefore, to evaluate the ranking gain of a term we would need to know its actual ranking, so that we can measure the distance between the two. However, in that case, we would not need to derive it in the first place, hence, we take a different approach on computing the ranking gain. The idea is that documents in which the term already exists in may provide clues about the overall ranking for that term. Even if they do not provide evidence for the actual ranking itself, they can still be useful.

Once the ranked result of querying the site with term t has been retrieved, we parse the documents and extract the terms they contain. For each term, we maintain a set of inverted indexes of the documents it is contained in with respect to each probed query, keeping the ordering in which the result was returned. This results in a set of orderings for each term, which we can aggregate to obtain a single one and compute the ranking gain as their level of disagreement. The more these rankings disagree among them, the more likely we consider it to be that the inferred ranking will be misleading. That is because their aggregate list, which seems as a natural choice for the derived list, would try to average all distances, which would result in a lot of information being lost. We can then define the measure used to compute the ranking gain of a term t as:

$$RankingGain(t) = \sum_{i=1}^{N} d(r_i, agg(t))$$

where $agg(t)$ is the aggregate list of the N rankings r_i, $\forall i = 1, .., N$, of term t and d is a distance function between two ranked inputs. An aggregate list is one that minimizes the distance between itself and all other input lists.
What we state in the above equation is that by querying t, what we gain for its ranking aspect is equal to the overall disagreement of the rankings that t belongs to already. Terms with identical rankings among queries have a ranking gain of 0, as we do not benefit with respect to this parameter. Terms with rankings that exhibit a strong correlation will have a lower benefit for that factor compared to ones with higher discordance.
A major drawback of this approach is that it needs to compute the aggregate list of each term t. However, ranking aggregation is known to be NP-Hard [75].Moreover, it can not be efficiently maintained incrementally. This entails recomputation of the aggregate lists from scratch. Hence, it is in our best interest to avoid consuming the crawler's resources on computing aggregate lists. Instead, we use the following observation:

Observation 1 *The bigger the distance is between two ranked inputs of a term t, the bigger the ranking gain of this term will be.*

We can then reformulate the ranking gain as

$$RankingGain(t) = \frac{2}{N * (N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} d(r_i, r_j)$$

hence computing the pair-wise distances between all ranked inputs of a term. To avoid boosting inputs with bigger lengths, we take the average of pair-wise distances, by dividing with the number of all possible pairs.
Putting it all together. Since we are interested in both maximizing coverage and achieving good results in approximating the ranking

of the Hidden-Web site, we combine the two cost models presented above into a single one. More specifically, we use their weighted average as:

$$Gain(t) = \frac{(1-w) * P_{new}(t) + w * RankingGain(t)}{Cost(t)}$$

The cost is paid only once, as it is related to the query term and to the number of pages that it will retrieve, and is the same for both coverage and ranking gain. Retrieved documents are parsed and terms are extracted, and we select as the subsequent query the one with the highest (overall) $Gain$.

4.2.2.4 Ranking Distances

There are several measures that are particularly suited to compute the distance between two or more ordered lists, most of which aim at finding the degree of correlation. Such rankings include Kendall τ, Spearman footrule and Spearman ρ, top-K variants, nDCG etc. We have used the ones that are most commonly used in the bibliography, due to several properties that they exhibit (i.e. extended Condorcet criterion). In our work we experimented with different metrics for computing ranking distances as they directly affect our crawling strategies. More specifically, we experimented with Kendall *tau*, *Spearman Footrule*, *Top-K* and variations of these, to account for potential ties or different ranking lengths.

Kendall τ. Kendall τ measures the correlation of two given lists, as the number of pairwise swappings, given by the equation

$$\tau = \frac{2 * (n_c - n_d)}{n * (n-1)}$$

where n_c is the number of concordant pairs, n_d the number of discordant pairs and n the number of distinct elements in both lists. We normalize the result in the $[0, 1]$ range.

We have also used Kendall $\tau - b$ to account for ties, following the methodology in [79]

Spearman Footrule. Another commonly used ranking distance is the *Spearman footrule*, where the distance of two rankings $r1$ and $r2$ is given by:

$$D(r_1, r_2) = \sum_{i=1}^{n} |r_{1_i} - r_{2_i}|$$

and r_{i_j} is the index of $j-th$ element in ranking r_i. To address the problem of ranking length variation, we have also used *scaled Spearman footrule (SSF)*, given by

$$D(r_1, r_2) = \sum_{i=1}^{n} |\frac{r_{1_i}}{|r_1|} - \frac{r_{2_i}}{|r_2|}|$$

Top-K. In essence, users are not interested in the entire ranking that a web site performs but rather in the top ranked ones, known as the top-*k* documents. Hence, the top-*k* distance captures the difference between the first *k* ranked documents, disregarding all others. Given a ranking r, we denote by $r(k)$ its first *k* entries. Using the formula from [80], the distance between r_1 and r_2 up to position i is given by

$$\delta_i(r_1, r_2) = \frac{|(r_1(i) \cup r_2(i)) - (r_1(i) \cap r_2(i))|}{2 * i}$$

This distance captures the fraction of non commonly shared items in the first i positions. Then, to get the overall distance, we sum δ's, for $i = 1, ..., k$. More formally

$$D(r_1, r_2) = \frac{1}{k} \sum_{i=1}^{k} \delta_i(r_1, r_2).$$

4.2.2.5 Experimental Evaluation

Experimental setup. We have conducted a set of experiments to measure our approach in terms of effectiveness, on YouTube [232]. YouTube is a video sharing web site, where users may upload videos and search for them through a simple keyword interface.

The YouTube service limits its results to 1000 items per query. However, there may be duplicates, i.e. the same video (identified by url) appears in different ranking positions. In such cases, we maintain only the first occurrence, which is when a user will see it for the first time and possibly select it. After removing duplicates, each query returns $\sim$800 results on average. This poses an upper bound on the number of videos we expect to see. Our results are from crawling the site between March 1st and March 28th, 2011, and each crawler may probe the service up to 300 times. Each configuration runs independently of the others and computes its statistics based on the set of documents that it has retrieved by itself.
We consider as documents the text-based information of the videos, i.e. title, description and tags. We did not include the user comments as they are not always directly related to the video itself.

Harvest rate. We start our presentation of the experimental results by first evaluating the number of documents that each approach manages to retrieve from the Hidden Web site. The keywords used to probe the YouTube service are selected according to our discussion in Section 4.2.2.3, and we employ various ranking distance functions, as shown in Table 4.4. As a baseline comparison, we use the techniques presented in [159].

Table 4.4: Crawling configurations

ID	**Distance**
CVR	*None*
KTA	*Kendall τ-a*
KTB	*Kendall τ-b*
SF	*Spearman Footrule*
SSF	*Scaled Spearman Footrule*
TOPK	*Top-X*
cKTB	*KTB with Jaccard weighting*
cSF	*SF with Jaccard weighting*
cSSF	*SSF with Jaccard weighting*
cTOPK	*TopX with Jaccard weighting*

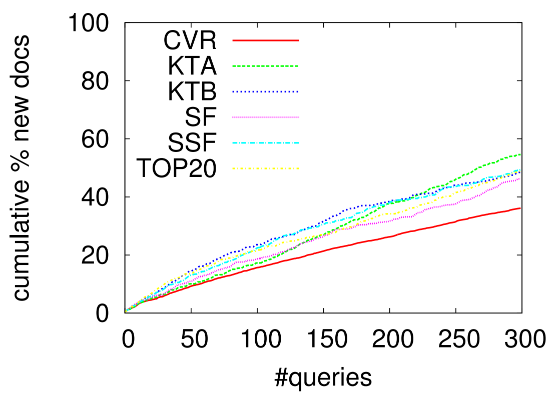
Figures 4.9(a) and 4.9(b) show the number of documents that each configuration retrieved from the Hidden Web site. Knowing the upper limit of YouTube query results (i.e. 800), we display our findings as

a percentage of the *optimal* retrieval case, where each new query retrieves the maximum number of new distinct documents. We note that the approach that does not take ranking into account at all, performs the lowest among all of the techniques.

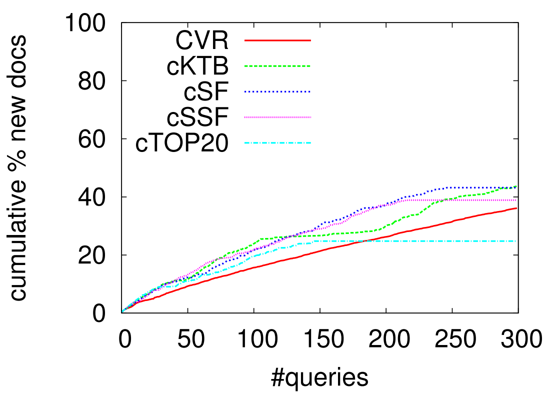
The graphs in Fig. 4.9(a) are a direct application of the discussion in Section 4.2.2.3, using different distances. We have also experimented with a variation of these techniques, the results of which are shown in Fig. 4.9(b). In this case, we have weighted the outcome of the distance function by the *Jaccard coefficient* of the two rankings, i.e. the fraction of their common documents.

As we observe from the graphs, different configurations (i.e., different ranking functions) perform differently. Overall, all policies achieve better coverage than [159], which does not account for result ranking. The highest harvest rate is given by Kendall τ-a, followed by SSF. The reason that the coverage-only approach performs the lowest, is that it relies on static features alone, whereas YouTube ranks results in a query dependent manner. Though we do not know the exact ranking function of the Hidden-Web site, loosing the (explicit) coverage aspect and increasing other, more dynamic, features seems to be beneficial. Interestingly, we also observe that the policies start performing better after issuing about 100 queries. This is due to the fact that the policies need to acquire some knowledge about the document collection before they start selecting good queries to probe.

Empirical comparison of the approaches. Table 4.5 shows the 15



(a) Harvest rate (%) of approaches



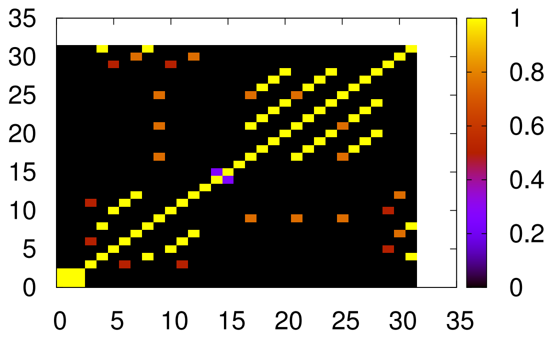
(b) Harvest rate of approaches

Figure 4.9: Effectiveness in harvesting information from Hidden Web sites

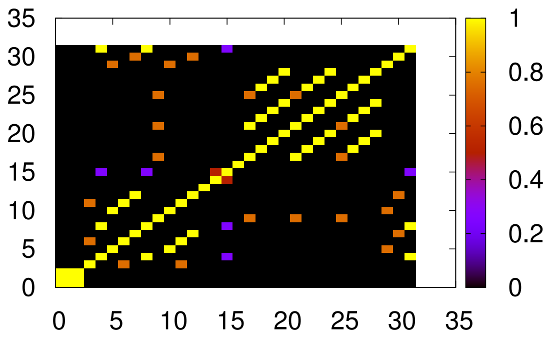
Table 4.5: First 15 query terms by configuration

No Ranking	KTa	KTb	SF	SSF	Top-20
soup	soup	soup	soup	soup	soup
http	http	http	http	http	http
twitter	homemade	parents	annual	prepares	listening
www	doneness	retro	bold	coulis	sausage
youtube	panlasangpinoy	residents	casserole	fresco	nutrition
watch	margarine	traumatic	acidity	penne	steamed
follow	toweling	deborah	aparta	unbelievable	songifications
add	fashioned	ixzz	beet	mixer	wonton
video	foodwishes	casserole	antioxidants	shawtayee	siu
user	foodies	litre	absorbable	stvplayer	intestines
center	dmark	hing	acquire	sinatras	powders
subscription	swirls	hamburgers	antibacterial	larder	enhances
machinima	dmarkii	mattar	antiparasitic	broadcaster	intricate
tags	secretlifeofabionerd	sooji	antiinflammatory	gilbrook	craftsmanship
high	hungrynation	drizzle	acnes	tyres	luggage

first terms for each of the approaches. Apart from the 1st query, which was the seed term and the 2nd one, where the ranking has not yet taken effect, it is clear that all configurations differentiate from each other as soon as the 3rd query. An interesting observation is that, apart from the coverage approach, the first 10 query terms in all configurations appear to be semantically related to the first one, i.e. *food*.
Figure 4.10(a) shows how the configurations correlate with each other in terms of the selected queries. This grid map shows the number of commonly queried terms between any two of our about 30 configurations, measured as the percentage of their intersection. The closer the value is to 1.0, the higher the correlation. It is clear from Fig. 4.10(a), that certain configuration, not on the diagonal, exhibit a high correlation. These in fact use the same distance measure (e.g. *KTb*), with different weight w. Nevertheless, apart from these configurations, the rest have very low correlation (below 25% in most cases). This means that the query terms selected to probe the Hidden-Web site are entirely different, despite the fact that all configurations started with the same one.
Moreover, the fact that configurations probe with different queries is not sufficient on its own, as they could be retrieving similar documents. For this reason, Fig. 4.10(b) shows a similar map, measuring the correlation of retrieved documents by each configuration. Again, this is



(a) Probed queries correlation among configurations



(b) Retrieved documents correlation among configurations

computed as the pairwise intersection of documents for all configurations. We can clearly see that unless the terms are the same, the configurations retrieve entirely different portions of the Hidden Web site. This is the main hint that using the ranking aspect results in a more *breadth*-wise search.

4.2.3 Mining data using execution engines and declarative languages

The use of declarative languages to retrieve information has its merits, and has been extended from the relational to other settings as well, such as streaming environments [24, 44] and the web [25, 193]. Meanwhile, bridging classic query processing with data analysis and mining techniques has long been of interest in the database community, as exemplified by the numerous prototypes [102, 112, 157, 183, 143, 201, 219]. These prototypes generaly propose a new declarative query language, or enhance the existing one, to support data mining constructs. One of the advantages of declarative languages is that they are typically processed by query execution engines, which optimize for certain objectives, tied to the domain of application. As a result, data mining techniques can greatly benefit from this type of merging, given that they will be also optimized for the domain.
Although several alternatives have been proposed to merge data mining with query processing, they typically suffer from the following shortcomings:

- they also propose their unique query language

- focus on specific algorithms, and
- are developed having a single domain in mind.

Most of them also rely on User Defined Functions (UDFs), which have been generally criticized for being "black-boxes" and not optimization-friendly [55, 154]. Sub-optimal plans, however, impact differently one domain from the other, making these techniques unfit to port between domains. For example, sensor networks are constrained on resources, face a distributed setting and dynamic environment, and are substantially different from a relational database setting. A poor execution plan that runs in a relational database simply increases running time of the query, leaving the user to wait. On the contrary, a poor plan destined to run in a sensor network could completely drain nodes from their energy and render the network useless. Therefore, it is important to optimize the code for each setting of application, but doing so manually is error prone and requires significant technical effort.
To successfully integrate query processing and data analysis, we identify the following desiderata:

i) The primary objective is to support data analysis and mining tasks, such as clustering, classification, outlier detection, etc., in a consistent way across domains, e.g. relational databases, distributed databases, or streaming sensor networks.

ii) *Efficiency* is still a major concern, although its definition is highly dependent on the domain. For instance, relational databases are interested in reducing response time, whereas for a sensor network energy consumption is a first-class citizen.

iii) Ease of programmability and development of data analysis techniques is an additional goal, as it increases a programmer's productivity. Note that the integration of new techniques needs to satisfy the *efficiency* constraint we mentioned above.

iv) Maximize adoption and ease of use by the end-users.

To address these concerns, we propose an approach that allows users to express data mining tasks through a high-level declarative language, e.g. SQL. Our contributions in that respect can be summarized as follows:

i) We give an approach to define and execute data analysis techniques using declarative queries. The main advantages are speed in development and deployment, a simple syntax and a system that takes care of correctness implications.

ii) We conceptualize data analysis techniques as *intensional* extents, i.e. sources whose data do not have to be acquired or stored, as opposed to *extensional* ones, and derive a flexible framework where we can combine these two types within the same query.

iii) We propose a *query refactoring* approach, motivated by the idea that several data mining algorithms can be expressed as algebraic operators. Therefore, unlike UDFs, we can leverage the optimizers that query processing engines already have.

We showcase our approach through two entirely different data analysis techniques. We focus on the sensor network domain, and in particular aim for in-network execution of data mining techniques. The selection of the domain is driven by two main reasons. First, sensor networks need to take the temporal aspect into account, due to the constraint on the resources of the sensing nodes and the fact that the environmental conditions change over time. Therefore, the sensing nodes can not maintain the entire history of sensed data, so better models are required. The temporal aspect is also present in social media data, as discussions and posted information typically addresses current issues. Consequently, we would like to use an infrastructure that inherently supports temporal information. Secondly, the limited capabilities of the nodes make the sensor network domain a very demanding one. Therefore, if our proposed approach performs well in such a demanding setting, it would be really worth considering for other domains as well. Finally, there have been open sourced solutions regarding query execution engines for sensor networks [], which allow building upon both the execution engine and the language itself. For the reasons mentioned above, we built a prototype system, SNEE-A, that enhances a query execution engine for the sensor network domain, so that it accommodates data mining techniques. We demonstrate our approach through two different data mining techniques:
Online correlation: If we know there is a correlation between the values of two measured variables, e.g. temperature and humidity, is it

possible to predict humidity knowing only temperature readings and can this be done efficiently?
Outlier detection: Given a sensor network deployment that measures humidity and temperature, we want to be informed of anomalies in readings.
The rest of the Section is organized as follows: Section 4.2.3.1 discusses related work. Section 4.2.3.2 presents the necessary language extensions to integrate data analysis with standard query processing, whereas Section 4.2.3.3 the needed modifications to the query engine. Our proposed approach is experimentally evaluated in Section 4.2.3.4.

4.2.3.1 Related Work

We first focus on general approaches that bring together data mining tasks and classic query processing, and then focus on more specific approaches for the in-network setting, given the application domain. Recall that sensor networks are similar to social media, assuming that we are interested in temporal data types, so that we may consider and analyze posted information as a stream.
Unifying query execution engines with data mining techniques under a common query language has been a research topic since the mid 1990s. Support for association rule mining [102, 112, 182] and classification [157, 183] at the language level has been examined. These approaches, however, were developed for the relational setting and were too narrow on their supported algorithms. For example, both classification techniques dealt with decision trees. More importantly, though, they all employ UDFs to achieve their goals. UDFs have long been critized as effectively being "black boxes" and not optimization-friendly [55, 154]. However, the impact of non-optimizable query operators may largely vary from setting to setting. For instance, a poor execution plan in the relational setting results in longer execution times and lower user satisfaction. On the other hand, poor optimization for in-network processing is detrimental, because it can drain node energy very quickly, rendering the sensor network practically useless.
The work in [160] employs SQL queries to perform K-Means clustering through the use of triggers and vendor-specific SQL scripting exten-

sions. However, the objective of that work is not to integrate data mining with a query language, but rather to use (relational) database technologies to perform K-Means clustering. Also note that the utilities used therein (i.e., triggers) operate differently in relational and streaming environments.

Moreover, when moving from the classic relational domain to more complex ones, e.g. streaming environments, the declarative language itself is constrained in expressiveness. To overcome this, most proposed systems and techniques introduced their own declarative query language, which is usually an SQL variation for that setting, such as CQL [24] (Continuous Query Language) and variants [118], ACQP (Acquisitional Query Processing) [146] and SNEEql [44] (SNEE query language). ESL, proposed in [143], employs User Defined Aggregates (UDAs), a subset of UDFs, thereby inheriting their drawbacks. These languages, however, for the most part, do not focus on data analysis and mining support. MMDL [201], an ESL extension, is a step forward in this direction for the streaming setting. However, as pointed out in [219], memory requirements of UDAs (therefore ESL and MMDL) cannot be clearly estimated from their syntactic structure, which does not fit well the resource-limited sensor network setting.

We now briefly discuss query execution engines for sensor network, as our application domain of choice is such. Typically, there are two lines of work in this area: i) Gather all sensed data to a central node, the *sink* and perform operations in a centralized environment or ii) view the network as a distributed processing query engine, and (partially or entirely) evaluate queries in-network.

Systems that fall under this category include STREAM (Stanford Stream Data Manager) [23], Aurora [8], Borealis [7], TelegraphCQ [122] and the more recent SMM (Stream Mill Miner) [201]. SMM is the only one among them to target specifically at data mining support. It uses UDAs, thus inheriting their drawbacks which we already discussed.

Works under the second category include the Cougar project [229], which introduced database concepts in sensor networks, as well as some in-network aggregation. Madden *et al.* developed one of the most well-known in-network query processing frameworks, TinyDB [146]. Sensor readings are represented by a relational table, optimization is limited to operator reordering and the same load is distributed among

the nodes in the participating set, disregarding their position in the network topology. Despite their novelty in in-network query processing, none of them considers data mining tasks.
SNEE (Sensor NEtwork Engine) [83], is a sensor network query execution engine, optimizing queries submitted in a declarative language, *SNEEql*. SNEE considers multiple parameters that affect network efficiency, e.g. network topology, node availability, energy consumption of operators. By default, SNEE optimizes node power consumption, and maximizes network longevity. Quality of service requirements may also be imposed (e.g. delivery constraints), which effectively alter the optimization goal.
A hybrid approach is adopted by the recently presented *AnduIN* [118]. *AnduIN* uses a declarative, streaming language variant and supports data analysis techniques through UDFs at the query level. Another difference is that we model data analysis as algebraic operators and leverage the execution engine's optimizer, whereas *AnduIN* uses UDFs and evaluates code performance offline, through simulations.
Regarding in- and out of network custom data analysis and mining techniques, there is a large body of literature [11], not to mention for classic settings. However, these are stand alone solutions and not integrated with a query processing engine, which we aim for. It has also been discussed that they sometimes contradict established notions of relational databases [53], let alone streaming environments. Furthermore, in these cases, optimization issues are a responsibility of the algorithm's designer, despite the existence of optimizers in the processing engines, which we would like to take advantage of.

4.2.3.2 In-Network Data Analysis with a Declarative Language

Towards fulfilling our goals, we follow a holistic methodology that involves:

a) extending the declarative language appropriately, so that data analysis techniques are supported at the query language level.

b) implementing them as extensions to the query optimization stack, building on the contribution that they can be denoted by intensional extents.

For ease of discussion, we will use SNEE and SNEEql [44] as the query execution engine and language respectively. We chose SNEE due to its well-defined and modular query optimization stack, that extends the classical two-phase optimization approach from distributed query processing [120], as well as for the various optimization goals it supports. SNEEql is a declarative query language for sensor networks inspired by expressive classical stream query languages such as CQL [24]. Nevertheless, we stress that our findings apply in similar approaches where declarative languages are applicable.

Extending SNEEql. To support data analysis tasks at the declarative level, we manipulate them as *extensional* extent (i.e. relation, stream), leaving the query language syntax intact. As a distinction, We refer to them as *intensional* extents, i.e. sources of information for which it is not necessary that their tuples are acquired or stored.
Users create data analysis and mining tasks through `CREATE` statements, much like creating a view in relational databases, which alters SNEE's metadata to accommodate the new extent. To support this functionality, we extend SNEEql's data definition language (DDL), utilizing a hierarchical decomposition of data analysis categories and their techniques. Figure 4.10 shows the updated DDL syntax. Tokens in bold are reserved terms, while the rest are replaced by the corresponding rule. Unmatched tokens refer to specific algorithms and their respective parameters, e.g. the value k for k-Means.

Online correlation. Assume, for instance, the two extensional stream extents of Fig. 4.11, one for the amazon forest that reports tempera-

```
DDLIntExtent ::= createClause fromClause;
createClause ::=  CREATE dattype [ datsubtype, datparams ]  identifier
fromClause   ::=  FROM ( fromItem  )
dattype      ::=  CLASSIFIER | CLUSTER | SAMPLE |
                  ASSOCIATION_RULE | OUTLIER_DETECTION |
                  PROBFN | VIEW
datsubtype   ::=  linearRegression | knn | d3 | kmeans | ...
datparams    ::= paramListItem, dataparams | paramListItem
identifier   ::=  Any valid identifier
fromItem     ::=  Either an extent in the schema, or a sub-query
```

Figure 4.10: Syntax for Defining an Intensional Extent.

```
Schema:
  AmazonForest:stream (id:int, time:ts, temperature:float)
  TropicalForestData:stream (id:int, time:ts, temperature:float, humidity:float)
```

Figure 4.11: Example schema of two streams expressed in SNEEql.

ture values, and a more general tropical forest stream that reports temperatures and humidity values.

Figure 4.12 shows the creation of a linear regression classifier over *TropicalForestData*, using tuples within a 20 minute window to construct it. We can then use that classifier in subsequent queries with *TropForestLRF* as the extent's name, as shown in Fig. 4.13. Here we wish to predict humidity values from the *AmazonForest* extent given its current temperature (this is what `NOW' refers to). Incorporating intensional extents in such a way also has a natural interpretation in terms of query semantics: "Give me the humidity value of a tuple from (virtual) relation *TropForestLRF*, for which the temperature is equal to the current sensed temperature from *AmazonForest*". This makes our approach easy to understand for users who are familiar with SQL but not data analysis techniques.

Conceptually, when an intensional extent variable appears in an equality condition in the `WHERE` clause, what happens is akin to variable binding in logic languages, e.g. Datalog [10], after all necessary semantic checks have successfully completed. Our *query refactoring*

```
CREATE CLASSIFIER [linearRegression, humidity] TropForestLRF
FROM (
    SELECT RSTREAM temperature, humidity
    FROM TropicalForestData[FROM NOW-20 MIN TO NOW]
);
```

Figure 4.12: Creating a Linear Regression Classifier.

```
SELECT RSTREAM AF.temperature, LRF.humidity
FROM   TropForestLRF LRF, AmazonForest[NOW] AF
WHERE  AF.temperature = LRF.temperature;
```

Figure 4.13: Using the TropForestLRF intensional extent.

approach makes extensive use of these value bindings.
Note that *TropForestLRF* is constantly updated, as it is an *intensional* extent, built over the *TropicalForestsData* stream. As data is acquired from that extent, the classifier is updated as well. More generally, intensional extents inherit the acquisitional properties of extensional ones, upon which they are built.

Outlier detection. Our proposed approach can also handle more complex constructs, such as the D3 outlier detection algorithm [197]. Detecting outliers is crucial as it may serve as a data cleaning step, and outliers may be the result of an external event (e.g., faulty hardware).
D3 uses sampling and the Epanechnikov kernel density estimator, to approximate the distribution of sensed data. It reports data as outliers if they have low probability to have been drawn from the same underlying distribution that created their (multi- dimensional) neighboring data. D3 requires two parameters: a neighborhood *range* and a *probability* threshold. Figure 4.14 shows how to create a D3 intensional extent over the temperature values of *AmazonForest* from the last 20 minutes, where $range = 5$ and $probability = 15$%. Figure 4.15 shows a query using that extent to check whether the most recent tuple is an outlier. As we can observe from it, intensional extents can also be part of a self-join query.

4.2.3.3 Query Refactoring

We now present how an existing query execution infrastructure, i.e. SNEE, can be modified, so that it supports data analysis techniques. When a query is submitted, we check with SNEE's metadata whether it contains intensional extents or not. The occurrence of an inten-

```
CREATE OUTLIER_DETECTION [D3, 5, 0.15] d3od
FROM (
    SELECT RSTREAM temperature
    FROM AmazonForest[FROM NOW-20 MIN TO NOW]
);
```

Figure 4.14: Creating a D3 outlier detection extent.

```
SELECT RSTREAM AF.temperature
FROM   AmazonForest[NOW] AF, d3od od
WHERE  AF.temperature = od.temperature;
```

Figure 4.15: Using the d3od intensional extent.

sional extent triggers its substitution by a templated subplan, which performs its algorithmic computations. We collectively refer to this process as *query refactoring* [208]. In essense, we reformulate an initially posed query into an equivalent one, that is also expressed in the same declarative language (SNEEql).

This approach has the added advantage that data analysis techniques are no longer black boxes but can leverage the engine's optimizer, without altering query semantics. Query refactoring only affects the parts of the query related to the intensional extent, leaving the rest intact. To better illustrate the needed modifications, Fig. 4.16 shows how query refactoring changes SNEE's optimization stack.

The output of query refactoring depends on the intensional extent(s) used. This process is transparent to the user, who will simply write the initial query. It follows that we need not perform any changes to the

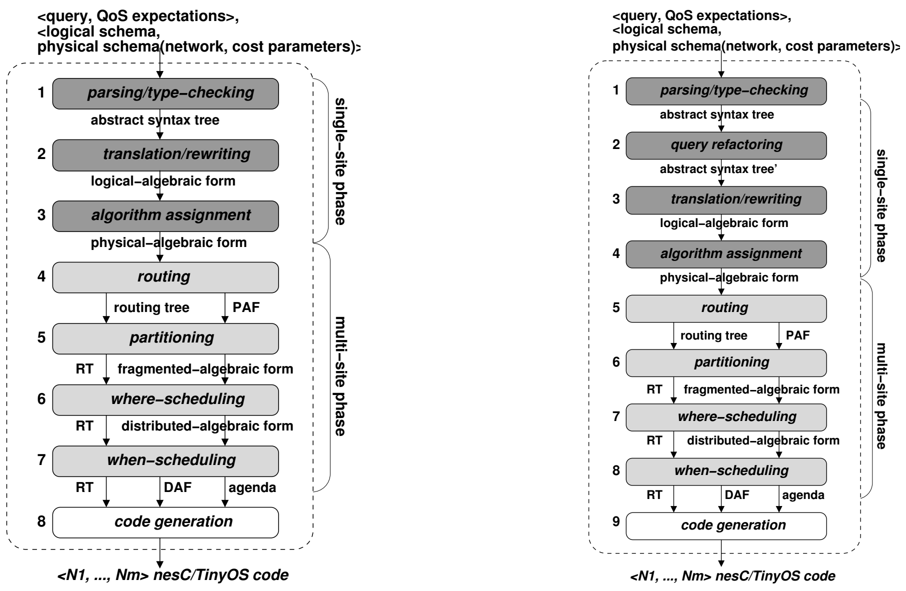


Figure 4.16: SNEE optimization stack: (left) original stack, (right) query refactoring approach for data analysis techniques.

query language level to support data analysis in such a way. On the downside, our approach is limited by the expressive power of SNEEql. Provided that the algorithmic description of a data analysis technique can be expressed in the query language, it can then be incorporated in our approach easily. We now demonstrate how query refactoring is applied to the two examples from Section 4.2.3.2.

Linear Regression. Assume an extent *LRFSource* with attributes *X* and *Y*, as shown in Fig. 4.17. Attribute *X* is the independent variable and *Y* is the dependent one. Values in bold are placeholders for actual extents and attribute names. Training this classifier is equivalent to computing coefficients (a, b) based on *LRFSource*, i.e., the source over which the extent was created. These values need to be appropriately updated as new readings are acquired, as discussed in Section 4.2.3.2. These goals can be achieved through the templated SNEEql subplan of Fig. 4.18.

Figure 4.19 shows a SNEEql query using the LRF extent from Figure 4.17. The last argument in the `SELECT` clause of Figure 4.19 is

```
CREATE CLASSIFIER [linearRegression, Y] LRF
FROM (
  SELECT RSTREAM X, Y
  FROM LRFSource
);
```

Figure 4.17: Creation of a templated Linear Regression classifier.

```
SELECT RSTREAM (r.n*r.sxy - r.sx*r.sy) / (r.n*r.sxx - r.sx*r.sx) as a,
       (r.sy*r.sxx - r.sx*r.sxy) / (r.n*r.sxx - r.sx*r.sx) as b
FROM (
       SELECT RSTREAM COUNT(t.X) as n,
           SUM(t.X) as sx, SUM(t.Y) as sy,
           SUM(t.X * t.Y) as sxy, SUM(t.X * t.X) as sxx
       FROM (
           SELECT RSTREAM X, Y
           FROM LRFSource
       ) t
) r;
```

Figure 4.18: Templated subquery for computing (a, b) values

the dependent variable of the classifier. The position of the dependent variable is not important; it has only been placed at the end to ease readability. Other extents can also be included in queries with intensional extents and projected attributes may also appear in the `SELECT` clause. The OP_is in the `WHERE` clause are standard boolean operators, e.g., *AND*, *OR*, combining the w_is, which are either boolean expressions or express join conditions between attributes of extents. When the user submits a query such as the one shown in Figure 4.19, we check whether it contains an intensional extent, using the stored metadata of the execution engine. For the partciular example that we consider here, we find that *LRF* is an intensional extent, at which point query refactoring comes into effect. Briefly explained, the steps that we perform are the following:

- Locate in the `WHERE` clause which expression of the w_is joins the **lri.X** variable. In particular, we are looking for expressions of the form ***lri.X***=*Z* or *Z*=***lri.X***, where Z is the attribute name of another extent. Let us assume that this is w_l in Fig. 4.19.
- Replace all occurrences of **lri.Y** with **lri**.$a * Z +$ **lri**.b.
- Remove w_l from the `WHERE` clause, as it will not be used anymore.
- *LRF* becomes a placeholder for the subplan of Fig. 4.18, and is substituted accordingly. We briefly refer to it as **LRF_ab**.

Figure 4.20 shows the templated form of the refactored query, once we have applied these steps. In the case of the *TropForestLRF* classifier from Fig. 4.13, the mappings that we obtain from these steps are shown in Table 4.6. Through careful substitution in the templated code, we also get the subplan for computing values (a, b) (Fig. 4.21), which substitutes **LRF_ab**. The overall refactored SNEEql query for our *TropForestLRF* classifier is given in Fig. 4.22. Note that all of

```
SELECT RSTREAM $s_1$, $s_2$, ..., $s_n$, lri.Y
FROM $e_1$, $e_2$, ..., $e_m$, LRF lri
WHERE $w_1$ $OP_1$ $w_2$ $OP_2$ ... $OP_{l-1}$ $w_l$
```

Figure 4.19: General form of a query using the *LRF* extent

```
SELECT RSTREAM s1, s2, ..., sn, lri.a * Z + lri.b
FROM e1, e2, ..., em, (LRF_ab) lri
WHERE w1 OP1 w2 OP2 ... OPl-2 wl-1
```

Figure 4.20: General form of a refactored query using Linear Regression

Table 4.6: Template variables mapping for the query in Fig. 4.13.

Template Variable	Mapping

LRFSource	SELECT RSTREAM temperature, humidity
	FROM TropicalForestsData[FROM NOW-20 MIN TO NOW]

LRF	TropForestLRF
X	temperature
Y	humidity
Z	AF.temperature

these operators are directly optimizable through the existing infrastructure.

D3 Outlier Detection. We have already discussed that our query refactoring approach is able to handle more complex constructs. Figure 4.23 shows the refactored query of the one in Fig. 4.15. The **STDEV** operator computes the standard deviation of the temperature tuples in the window specified in the FROM clause, which was provided when creating the extent. Note that this is just a convenient way of writing the computation of standard deviation, which can be also expressed through additional subqueries. Furthermore, this type of notation allows us to use more efficient, approximate algorithms [27], if we see fit. Recall that, during creation, the range was set to 5 and the probability threshold to 15%. The range is used to compute the closed form of the Epanechnikov integral, whereas the probability filters points which are outliers, in the `WHERE` clause. We have marked the parameters with bold to distinguish them from the same values used as part of other expressions. It is easy to see, nevertheless, that

```
SELECT RSTREAM AF.temperature, LRF.a * AF.temperature + LRF.b
FROM   AmazonForest[NOW] AF, (ab_COMP) LRF;
```

Figure 4.21: Refactored Query of Fig. 4.13.

```
SELECT RSTREAM (r.n*r.sxy - r.sx*r.sy) / (r.n*r.sxx - r.sx*r.sx) as a,
       (r.sy*r.sxx - r.sx*r.sxy) / (r.n*r.sxx - r.sx*r.sx) as b
FROM (
   SELECT RSTREAM COUNT(t.temperature) as n,
          SUM(t.temperature) as sx, SUM(t.humidity) as sy,
          SUM(t.temperature*t.humidity) as sxy,
          SUM(t.temperature*t.temperature) as sxx
   FROM (
          SELECT RSTREAM temperature, humidity
          FROM   TropicalForestsData[FROM NOW-20 MIN TO NOW]
   ) t
) r;
```

Figure 4.22: Subquery of (ab_COMP) in Fig. 4.21

all of the operators used are optimizable as they rely on operators defined in [44].

Extensions. Query refactoring is a general technique, that can be applied to all settings with declarative query languages, e.g., relational, streaming, which is another advantage of our methodology. Nevertheless, expressing data analysis techniques as SNEEql queries is non-trivial in its own right. The fact that merging classical query processing with data mining has been an active reasearch topic for many years is indicative of its complexity. Additionally, given that intensional and extensional extents can now be interleaved, several query refactorings are possible, leaving room for additional optimizations.

Additional extensions include how to efficiently materialize such data mining models and reuse them. Clearly, this is not always possible. For example, materialization is meaningful in a static environment like a relation database, but in a streaming environment, where the classifier is constantly updated as new data points arrive, such an alternative may be indifferent. What *is* interesting in both settings, however, is how to precompile and save the query operator tree of a data mining task, to subsequently integrate it in a new query, thus performing incremental optimizations. Although we do not consider such issues in this thesis, they are interesting research directions for the future.

```
SELECT RSTREAM od.temperature
FROM (
  SELECT x.temperature,
         ( 1/COUNT(y.temperature) ) * ( (1/4)^1 ) *
         SUM( (3 * 2 * 5 / q3.b1) -
          ( ( (x.temperature - y.temperature + 5) / q3.b1 )^3 -
            ( (x.temperature - y.temperature - 5) / q3.b1 )^3 ) ) as probability
  FROM (
    SELECT SQRT(5)*q1.sigma*(q2.rsize^(-1/5)) as b1
    FROM (
      SELECT STDEV(temperature) as sigma
      FROM AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN]
    ) q1,
    (
      SELECT COUNT(temperature) as rsize
      FROM AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN]
    ) q2
  ) q3,
  AmazonForest[now] x, AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN] y
  WHERE abs( (x.temperature - y.temperature) / q3.b1 ) < 1
  GROUP BY x.temperature
) od
WHERE od.probability < 0.15;
```

Figure 4.23: Refactored query of D3 outlier detection algorithm.

4.2.3.4 Experimental Evaluation

Efficiency in sensor networks is almost synomymous with energy consumption, which includes both CPU and radio energy consumption. To gain better insights, we also measured the number of transmitted messages and bytes. These two aspects are crucial in determining radio energy consumption. We evaluated all approaches using Avrora [202], a sensor network simulator, that provides accurate per-node statistics. All sources were written in nesC 2.0/TinyOS 2.x [87, 107] for MicaZ motes. The discussion will focus on linear regression, due to its widespread adoption as a data analysis tool, and the fact that we obtain similar results for outlier detection (Fig. 4.31).

We experimented with various topologies and Table 4.7 summarizes some of their structural properties. The topologies are not directly comparable as their structural properties differ. For instance, an 8-

Table 4.7: Structural properties of the topologies used in the experimental evaluation.

Size	Avg. Length	Max. Length	Leaf Nodes	Description
4	1.3	2	2	Tree
5	1	1	4	Star
8	2.14	3	3	Tree
9	1.75	3	4	Tree
11	1.7	2	7	Tree
12	2.18	4	6	Tree
20	1.79	4	9	Tree

node star-like network will behave differently from an 8-node chain. Because of this, extrapolating the results to other topologies should be performed with caution. Given a static topology during initialization, we construct a minimum-hop routing tree rooted at the sink, using one of several existing algorithms [16, 52, 227]. SNEE also uses the topology to find the best query routing tree, and does so during the routing stage of query optimization. As such, we have excluded the cost of building the routing tree from the graphs displayed below.

Given that we apply our method in a streaming setting, we experimented with various window, slide and acquisition intervals, and we will be using the following caption notation in the experimental figures for convenience: W:*w*, S:*s*, A:*a*, to signify them respectively. Varying the window and slide parameters gave similar results, so we omit these figures. Experiments were run for 300 seconds of simulated execution time. The periodic nature of the queries allows us to scale up the results for longer periods.

Handcrafted Algorithms. We implemented two baselines, both of which perform a depth-first traversal of the reverse routing tree. The sink is responsible for initiating a new tree traversal, close to the end of each acquisition interval, so that the result is timely reported.

In our first approach, Naïve, the sink probes nodes separately for data one after the other, receiving and aggregating the tuples. The second one, LC for "Local-Computation", traverses the tree and aggregates values in a postordered fashion, where each node is contacted by its parent only once within an epoch. On the contrary, SNEE optimizes queries based on a time-strict agenda. This allows nodes to contact each other at predefined times, using a push-based scheme. SNEE

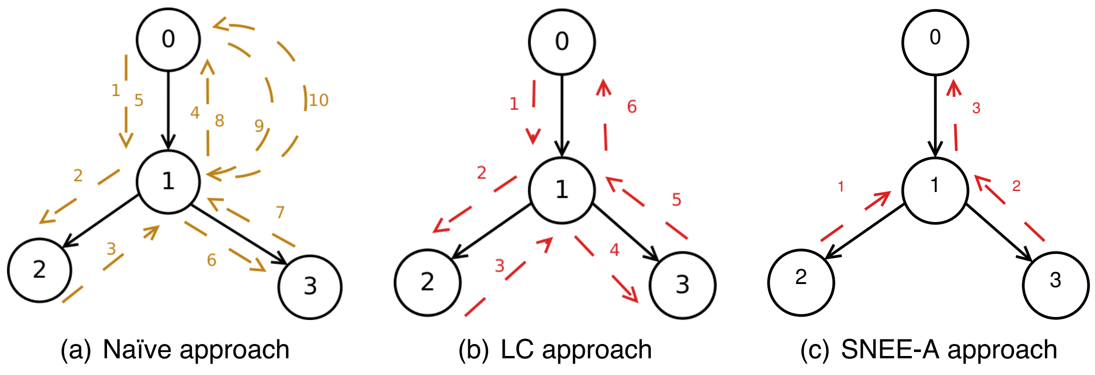


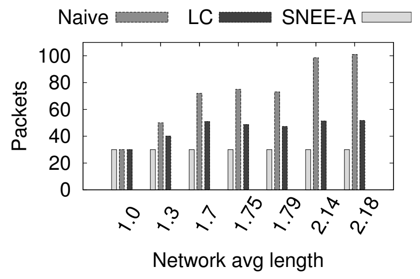
Figure 4.24: Example of Linear Regression computation for Naïve, LC and SNEE-A approaches.

calculates these times by utilizing the routing tree and its optimization cost models. All approaches are graphically portrayed in Fig. 4.24 for a simple 4-node example. Labels denote the sequence in which node communication occurs, until we obtain the full result.

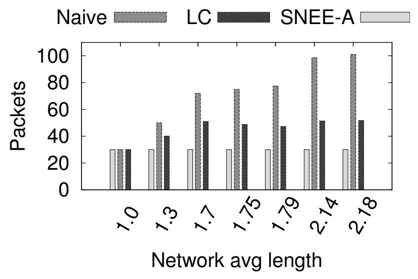
Radio Communication. Figure 4.25 shows the average, per-node, number of sent packets (y-axis) for the various acquisition intervals. On the x-axis, we plot the average network length, which is a better indicator of how the network is affected, than the network size. We observe that in practically all cases SNEE-A's performance is superior to the handcrafted alternatives, due to its push-based scheme. Differences are negligible when the average length is 1, due to the star-shaped topology, where each node sends directly to the sink. However, as the average network length grows higher, the difference among the techniques increases. The reason is that SNEE-A exhibits a steady performance, which is expected as each node will send data only once during a single epoch. On the other hand, the custom techniques require additional (control) messages to probe nodes for data.

The graphs in Fig. 4.26 show the average number of bytes sent in total by each node. SNEE-A and LC have a steady behavior across all acquisition intervals, unlike Naïve, as a result of local aggregations, performed by the two former approaches. Even so, SNEE-A is still superior to the other two, due to the control messages that the handcrafted implementations rely on.

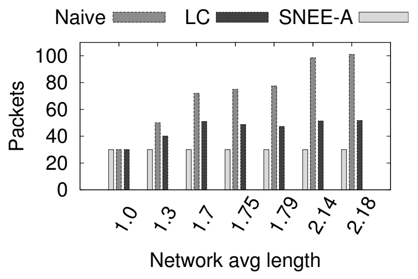
We can also see that the type of information to send depends on a combination of the sensing rate, the window size and the structural



(a) W:10, S:10, A:2

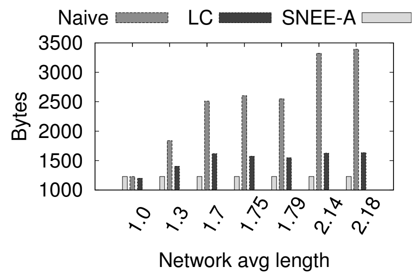


(b) W:10, S:10, A:5

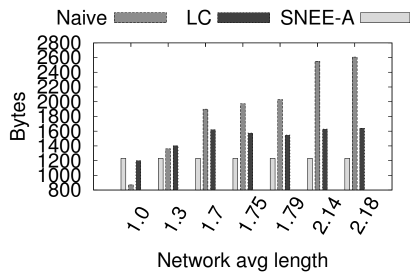


(c) W:10, S:10, A:10

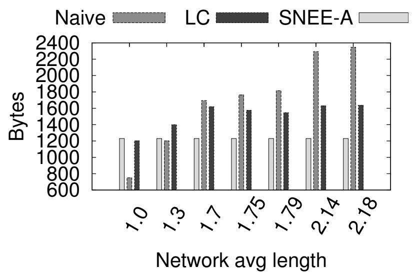
Figure 4.25: Average number of sent messages compared to the average network length for LR



(a) W:10, S:10, A:2



(b) W:10, S:10, A:5



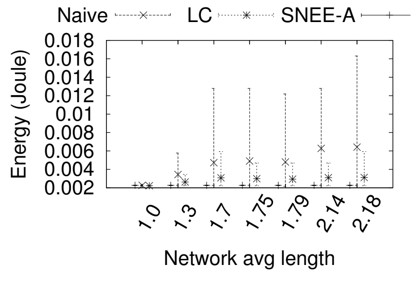
(c) W:10, S:10, A:10

Figure 4.26: Average number of sent bytes compared to the average network length for LR

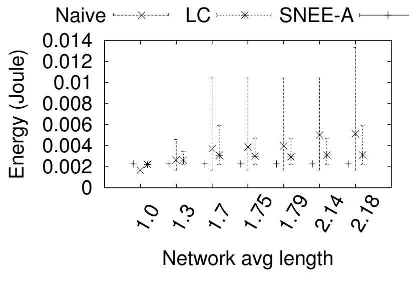
properties. For instance, for high sensing rates (Fig. 4.26(a)), it is preferrable to send aggregate information. However, as the sensing interval increases (Fig. 4.26(c)), fewer readings are taken during an epoch and sending the raw data becomes more efficient. Such optimizations are beneficial to a lot more queries when implemented within a query execution engine.

Radio Energy. Fig. 4.27 shows the average, minimum and maximum transmission energy consumption per node, for all techniques, clearly favoring SNEE-A. Given the push-based model and partial aggregations of SNEE-A, all three values are identical. This is not true for the custom implementations which use control messages, the number of which is affected by the network's structural properties. Figure 4.27 also shows the load distribution among the nodes, with SNEE-A, distributing it almost evenly.

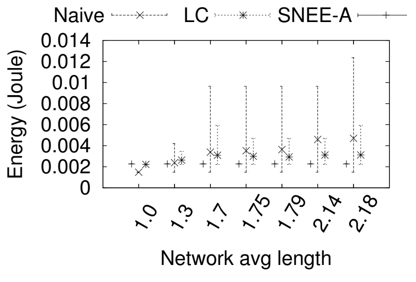
However, the factor that mostly affects radio energy consumption is the energy consumed while the radio is *on* waiting for messages, de-



(a) W:10, S:10, A:2

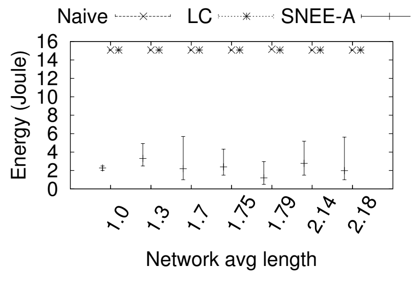


(b) W:10, S:10, A:5

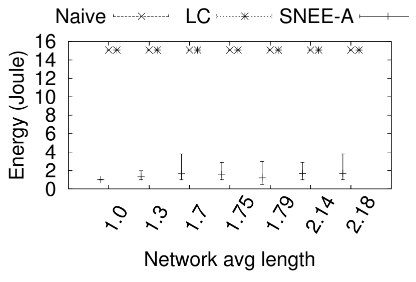


(c) W:10, S:10, A:10

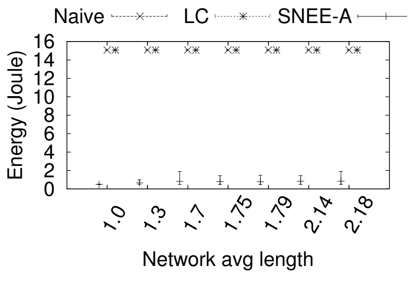
Figure 4.27: Transmission energy consumption compared to the average network length for LR



(a) W:10, S:10, A:2



(b) W:10, S:10, A:5



(c) W:10, S:10, A:10

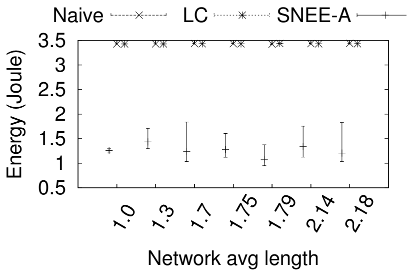
Figure 4.28: Reception energy consumption compared to the average network length for LR

picted in Fig. 4.28. The bespoke techniques have the radio switched *on* constantly, because it is impractical to manually compute when nodes will communicate. On the other hand, as a result of its agenda-driven task execution, SNEE-A performs radio management, switching it *on* and *off* for each node independently. This level of control over the radio allows SNEE-A to achieve better performance.

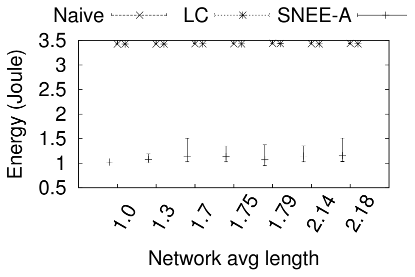
Therefore, although we aimed for communication optimization as well, i.e., minimize transmission costs, that was not sufficient on its own. This validates our objective to utilize existing infrastructures and query engine optimizers.

CPU Usage. We finally turn our attention to the CPU consumption of the nodes. Figure 4.29 shows how the minimum, average and maximum CPU energy consumption is affected by the network's properties. Clearly, CPU follows a trend similar with radio reception, once again advantaging SNEE-A, for reasons we previously described.

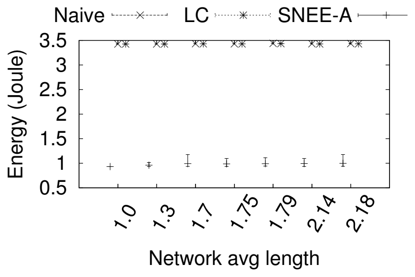
These remarks are backed up by the graph in Fig. 4.30, showing the



(a) W:10, S:10, A:2



(b) W:10, S:10, A:5



(c) W:10, S:10, A:10

Figure 4.29: CPU energy consumption compared to the average network length for LR

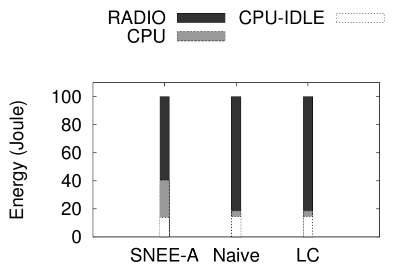
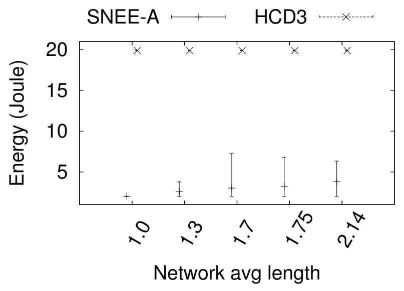
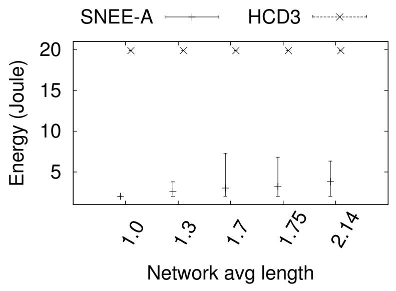


Figure 4.30: Radio-CPU contribution(%) to power consumption.



(a) W:10, S:10, A:2



(b) W:10, S:10, A:10

Figure 4.31: Average total energy consumption compared to average network length for D3

percentage with which the CPU and Radio components contribute to the total energy consumption, with an emphasis on the energy spent while the mote is idle. Even with SNEE-A's power management, the radio remains the dominant factor of consumption. However, the CPU is idle for proportionately less time compared to Naïve and LC. This implies that with SNEE-A, we make use of the resources of the node, when they are indeed required.

Ease of Programmability. One could argue that the handcrafted alternatives are inefficient because of their communication scheme. Although this is partly true, as demonstrated, most of the energy consumption is due to the radio being *idle*, which is a matter of *when* nodes communicate, rather than how they do so. Secondly, building manually a push-based approach, like the one used in SNEE-A, or even computing when nodes should communicate is impractical, as it involves accurate computation of processing and communication times for each node.

On the other hand, building a system or module that provides these accurate timings basically duplicates what the query engine already does. It is even less practical to rework this component when moving between settings (e.g., relational, streaming, distributed etc.). This brings us to another benefit of query refactoring: the time taken to write -- and debug -- the handcrafted code. For instance, SNEE-A has a clear advantage against the handcrafted alternatives, as i) the developer uses high-level (declarative) languages instead of low-level, and ii) the system autogenerates and deploys code for all nodes in the network, optimized for the requested goal. Finally, note that if we change the optimization goal, the bespoke techniques must be reimplemented, whereas for SNEE-A (and similar execution engines) it is a single parameter.

4.3 Comparing sampling policies

As we pointed out in the introduction of this Chapter, a contributing factor of Twitter's success and adoption by researchers is its data policy towards the academic community and application developers. In particular, Twitter provides access to its data through two distinct Application Programming Interfaces (APIs), and Section 4.2 discussed the architecture of an efficient crawler for the service. As also pointed out in that section, each API poses certain restrictions, with the Streaming API limiting access to 1% of all public tweets. Despite the 1% sample being the default behavior, one may obtain access to higher data ratios for research purposes (up to 10%) or for a fee.
It is easy to see that access to higher ratios will increase the computational costs, given the 10-fold increase in data. This may, in turn, increase the financial cost to be able to cope with the data volume of these rates. A natural question then is what are the merits of higher data ratios? In other words, for a given application, is the 1% sample sufficient or are the advantages of higher ratios such that make the financial and computational costs negligible? For example, is the 1% sample reliable enough for the purposes of an application? Recent work [155] demonstrated that the default sample returned by the Twitter service is not a truely uniform one.

Similar in spirit, we, too, are interested in evaluating the extent to which analytical processes are affected by the aforementioned limitation, i.e., having access to a limited proportion of the entire information. In particular, we apply a plethora of analyses on two subsets of Twitter public data, obtained through the service's sampling API's. The first one is the default 1% sample, whereas the second is the Gardenhose sample that our research group has access to, returning 10% of all public data. We extensively evaluate their relative performance in numerous scenarios. The difference between our current work and the one in [155] is that we focus on specific aspects of the data, namely spatial and temporal, which are inherent due to the nature of the medium itself, as we explain in the following.
Given that it would be impossible to apply all data analytical techniques in order to evaluate extensively the obtained samples, we select a representative subset, motivated by different application scenarios, and report on those findings. More specifically, we answer the following research questions:

- *Sentiment Analysis*: Sentiment analysis has been used to evaluate the performance, and predict the outcome of potical debates and elections [64, 150], to perform brand monitoring and event detection [205, 149], to name a few. Consequently, we want to study how the Twitter API sampling affects the widely used spatio-temporal analysis task of Sentiment Analysis.

- *Geo-located information*: How many geo-located tweets are provided by the Streaming API and the Gardenhose? How is the relation between these two values varying through space and time? In all of these tasks we study how the difference between the two streams varies in different locations. Stefanidis et al. [194] reported that approximately 16% of the Twitter feeds in their experiments had detailed location information with it in the form of coordinates, while another 45% of the tweets they collected had some geo-location information at coarser granularity (e.g. the city level).

- *Popular tweets*: We extract trending topics of various locations using the Twitter API and the Gardenhose and study their differences.

- *Social Graph Evolution*: We focus on the retweet graph, and want to see how the sampling process affects certain of its measures.
- *Linguistic Analysis*: We apply language detection, to compare the linguistic statistical properties in the received sample against ground truth data. More specifically, we answer the question whether the written languages found in Twitter are a representative sample from languages in the physical world.

4.3.1 Related Work

We will focus on the techniques that may be applied to Twitter data, given the analytical purposes that we are considering in this Section.

4.3.1.1 Spatio-temporal analysis of Twitter feeds

As we have already discussed in previous paragraphs of this Chapter, the Twitter service is characterized by a high degree of diversity regarding its users, in terms of location, spoken languages, backrgound, interests, etc. The spatial aspect is of paramount importance for a large number of applications, such as event detection and response, targeted advertising and community detection to name a few. Towards that end, users are able to geo-tag their tweets, i.e., attach GPS information.
Unfortunately, despite providing high precision information, it has been shown in numerous studies [194, 204] that the percentage of GPS-tagged tweets is too low. Moreover, such information is typically mediated through other location-based services, e.g. Foursquare[8], which reduces the textual content provided by the users themselves. To address this shortcoming, researchers have proposed techniques which aim to extract spatial information from the text, either of the tweets or the users' profiles [14, 17, 108, 204].

[8]https://foursquare.com/

4.3.1.2 Sampling Social Data Streams

The number of users who are actively using the service, and the amount of information posted daily are enormous. To cope with these sizes, sampling is usually employed to downsize the data, while maintaining the properties of interest. For example, the work in [91] proposed techniques to perform online sampling on the social graph of Facebook. This approach is equivalent to a uniform sampling of the nodes, without knowing the entire graph in advance. The authors in [90] apply sampling on users, in order to identify topical experts.
The work most closely related to ours is [155]. Having access to the Firehose, the authors compare the default sampling policy of Twitter against the entire Twitter stream. One of the main outcomes of their work was that the sample provided through Twitter's default streaming API is not a random sample. Compared with that work, we want to evaluate the performance of the 10% sample (Gardenhose) and contrast it with the 1% default sample. We also take a more temporal standpoint of evaluation, and focus on more analytical processes, such as sentiment analysis and linguistic analysis. We are also interested in evaluating properties of retweeted posts, that go beyond the retweet graph itself. Neverthe data, while maintaining the properties of interest. For example, the work in [91] heless, the reported values in [155] can be used as ground truth information, considering that they had access to the full Twitter stream.

4.3.2 The Data

Our experimental setup relies on data received from the Twitter service, and in particular the service's streaming API[9], which follows the publish-subscribe paradigm: users subscribe to the service, with a request to receive data. Twitter sends the data to the subscribed users, according to a sampling policy. This results in the service being less stressed by contiuous probes for new data.
The default sampling policy returns 1% of all publicly available tweets, i.e., tweets from users who have allowed everyone to see their posts. Our group has also been granted access to the Gardenhose, which

[9]https://dev.twitter.com/docs/api/1.1/get/statuses/sample

returns a 10% sample of the publicly available tweets. In both cases, the sampling policy is controlled entirely by Twitter.

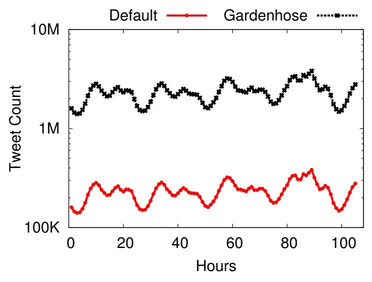
Our main set of experiments is conducted on two datasets, obtained by crawling the 1% and 10% of the service over the same period of time. We monitor the Twitter stream for slightly over a 4-day period in November 2013, and store the tweets as they are provided. We subsequently perform our analyses in an offline fashion, using a custom workflow infrastructure [207].

Figure 4.32a shows the amount of information we collected within each hour from the onset of our experiment, for both sampling policies. An immediate observation is that the two samples differ by an order of magnitude, which is expected given the sampling percentages offered by the service. Secondly, we see that both samples exhibit the same temporal pattern, with the same increases / decreases appearing in both streams. Finally, it is interesting to note that in both cases, there is a certain periodicity in the data, which coincides with the 24-hour cycle of a day.

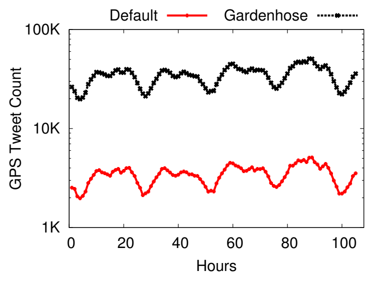
4.3.3 Geo-location Coverage

An important aspect of Twitter data is that several of them are geotagged, meaning that the posting user has attached a GPS-quality signal to the tweet when uploading the information. Such information can be particularly important to understand where the user is and what they are refering to.

Figure 4.32b shows the number of geotagged tweets that were received from the two Twitter sampled streams, the default one (red)



(a) All tweets



(b) GPS-tagged tweets

Figure 4.32: Comparing default and Gardenhose samples for volume over time

and Gardenhose (black). It is interesting that we observe the same temporally periodic pattern as the one in Figure 4.32a. Moreover, the geotagged tweets are between 1-2% of their respective raw sampled data, and the two streams (of geotagged tweets) differ by an order of magnitude, which is a result of the Gardenhose returning 10$\times$ more tweets that the default sample. Finally, several of the fluctuations that we observed in Figure 4.32a have been flatened out when we consider the geotagged tweets alone.

Twitter also allows its users to ask for geotagged information. In this case, the user connects to the streaming API, but indicates that they are interested in geotagged tweets. To do this, the user provides four geodetic coordinates: $[(lat_{min}, lon_{min})(lat_{max}, lon_{max})]$, which constitute a bounding box, and Twitter returns tweets that fall within this region. In this particular case, the volume of the returned results is the same for the two samples. The reason is that a different mechanism is used by the service. Given that there is no difference between the two sampling ratios in this case, we omit these figures.

However, we would like to check whether there are any other differences that may arise from the use of this mechanism. To this end, we focused on a particular region in London, and applied different bounding boxes, which slightly overlap. Table 4.8 shows the coordinates for the bounding boxes that were used, along with the number of tweets that were received, whereas Figure 4.33 visualizes these on a map.

Table 4.9 shows the similarity between the collected tweets. In particular, we have computed the Jaccard similarity of the received data and report these values. It is interesting that the measured similarity is generaly quite high, even when the overlap is low (e.g., Crawl 1 and 2). As the overlap increases between the bounding boxes that we applied, so does the similarity between two different crawls.

Table 4.8: Description of the the GPS-driven crawls

ID	Bounding Box	#Tweets
Crawl1	[(-0.1754, 51.4830), (-0.0704, 51.5327)]	35275
Crawl2	[(-0.2654, 51.4830), (-0.1604, 51.5327)]	27811
Crawl3	[(-0.2254, 51.4830), (-0.1204, 51.5327)]	27811
Crawl4	[(-0.1854, 51.4830), (-0.0804, 51.5327)]	27811

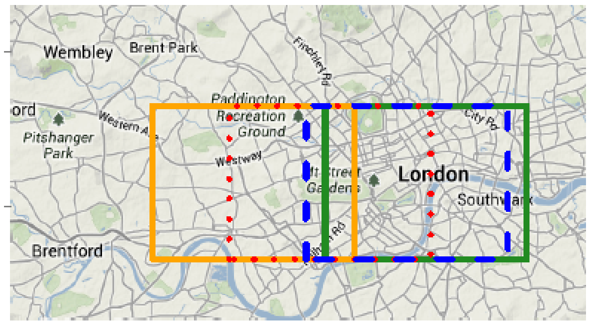


Figure 4.33: Bounding boxes of Table 4.8, Crawl1: Green, Crawl2: Orange, Crawl3: Red, Crawl4: Blue

Table 4.9: Jaccard Similarity between the GPS-driven crawls

	Crawl1	**Crawl2**	**Crawl3**	**Crawl4**
Crawl1	1.0	X	X	X
Crawl2	0.527	1.0	X	X
Crawl3	0.671	0.788	1.0	X
Crawl4	0.866	0.612	0.777	1.0

Figure 4.34 shows how the 4 distinct bounded-driven crawls performed over time for a single day. With some minor fluctuations, we observe that all of them follow the exact same pattern. Note that the first half-hours, where there is a steep decline in volume, are in the early hours of the day because the crawl was started around 10:30pm. Therefore, the x-values between 3 and 15 depict the volume between midnight and 8 o'clock in the morning.

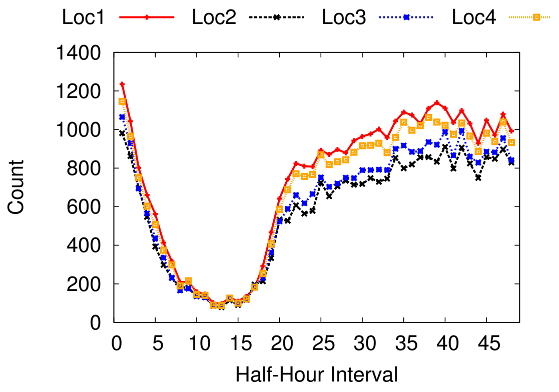


Figure 4.34: Different crawls from London

4.3.4 Sentiment Analysis

Sentiment analysis is probably one of the most frequent tasks applied on Twitter [110, 150, 138, 162]. The vast availability of opinions expressed in Twitter raised the interest of the research community as well as the industry. Hence we consider this problem as one of most critical tasks where the sufficiency of the Streaming API (1%) should be evaluated.

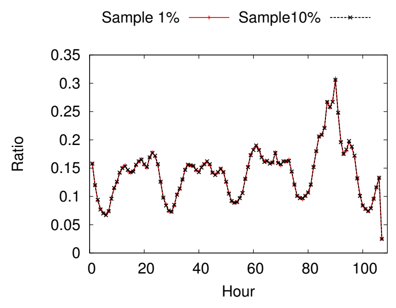
In general, the problem of sentiment analysis is that given a text segment t_i, it is requested to assign it into one of the polarity classes (`negative, neutral, positive`) according to the sentiment that it expresses. In fact, most of the times, the output is a *sentiment rate* in $[0, 1]$. In the setting that we consider, the task is to assign such a label to each tweet individually.

For our analysis, we employed a lexicon-based approach, whereby positive opinion words contribute towards the positive classification of the text whereas negative opinion words contribute towards negative classification. The obvious advance of a lexicon-based approach is that no training data are required and that there are low computational requirements. Naturally, to apply such a technique, two sets of words are required: a positive and a negative one. We utilize the lists provided in [109]. Given the text of a *tweet*, we count how many words appearing in it express positive (w_+) or negative (w_-) opinion. We then assign the tweet to one of the classes as follows:

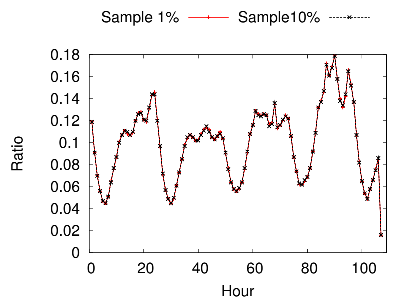
$$sentiment(tweet) = \begin{cases} positive & : |w_+| > |w_-| \\ negative & : |w_+| < |w_-| \\ neutral & : otherwise \end{cases}$$

Figure 4.35 reports the ratio of positive and negative tweets, per hour, over all tweets received during the same time period. Interestingly, we observe that the ratio of tweets is the *same* in both occasions, although the absolute values differ by an order of magnitude. The ratio is higher for positive tweets, with certain cases having twice as many positive tweets. There is also periodicity in the data, similar to the one that we observed in previous sections.

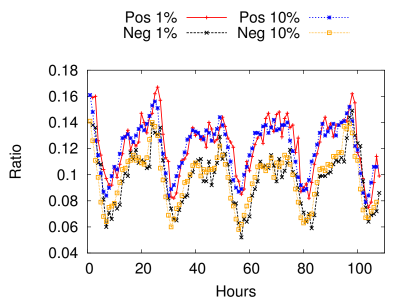
Inherently, Sentiment Analysis has spatio-temporal characteristics. In the last USA presidential elections, many organizations kept track of the sentiment *during* the electoral period (trend) for *each one of the*



(a) Positive Sentiment



(b) Negative sentiment



(c) Sentiment in geotagged tweets

Figure 4.35: Comparing tweets with sentiment

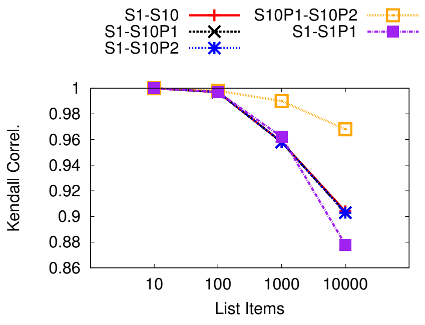
states. For this reason, we also provide an experimental comparison by applying sentiment analysis to the subset of geotagged tweets that were received with the two sampling policies.
As we observe from Figure 4.35c, the ratios of positive and negative geotagged tweets exhibit similar patterns to the general stream, shown before. Even in geotagged tweets, there are more positive ones than negative, regardless of the streaming policy used. The ratios, however, are in principle lower than in the general stream, meaning that geotagged tweets offer less sentiment-oriented information.

4.3.5 Popular Topic Detection

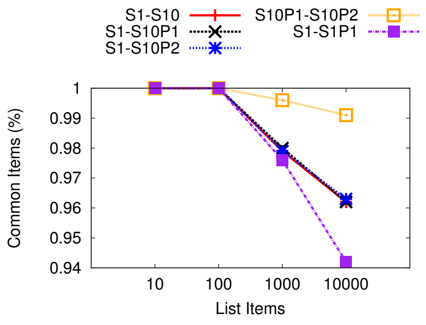
One of Twitter's most characteristic features is the ability of its users to *retweet* other posts. Such an action allows for fast dissemination of information, leading to *viral* posts, which are a means to identify trending topics and trendsetters [178]. Retweeting also implies that the user is interested in the content of the original post, and that they are endorsing it, which can be a genuine resource for community detection [32].

4.3.5.1 Top-most retweeted posts

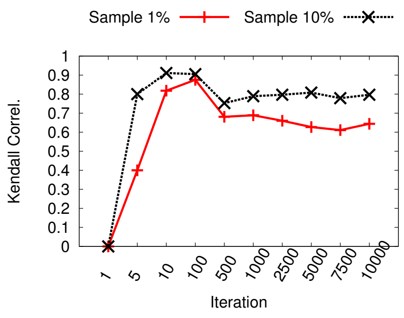
A first kind of analysis we are interested in, is to see whether the two sampling policies differ in terms of the information that they return, with respect to retweets. Towards that end, we conducted the following experiment: We extract the top-k most retweeted posts, that appear



(a) By iteration



(b) By interval size



(c) Times Received VS Times Retweeted

Figure 4.36: Comparing the top-10000 most retweeted items

in our data. Among other information, Twitter provides the number of times that a post has been retweeted, which serves as the ground truth for ranking. For each of the top-k tweets, we also maintain the number of times that they appear in our dataset.
At the end of this analysis, we obtain a top-k list for each sample, ranked in descending order of their retweet count (ground truth). We want to compare the degree of agreement between the two lists, one obtained from each sample. Given that these are ranked lists, we compare them using Kendall's $\tau - b$, which is given by the following equation:

$$\tau = \frac{(n_c - n_d)}{\sqrt{N_1 * N_2}}$$

Kendall's method performs a pairwise comparison of the first n items in the lists, and finds how many pairs appear in the same order (n_c), and how many do not (n_d). In the denominator N_1 and N_2 are the number of items not-tied in the lists. Items which appear in one list but not the other are appended at the end [79]. The result is in the range $[-1, 1]$, where -1 means that the two lists are completely reversed, and 1 is that the two lists are identical. We repeat the comparison for various (sub)list sizes.
We extracted the top-10000 most retweeted items, as they appeared in the 1% and 10% samples. We then compare the two sublists (one for each sample), starting with the top-10 and increasing each time its size by an order of magnitude (top-100, top-1000, etc.). We also

compare how many of the most retweeted posts are shared in the two lists. To check for any bias from Twitter's sampling policy, we also randomly split each sample in half, and rerun our experiment.

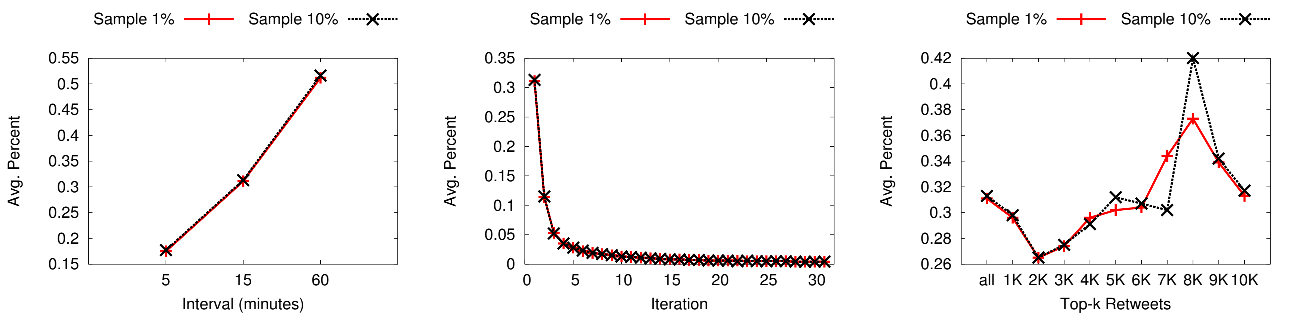
Figures 4.36a-4.36b show the results of this experiment. A label with S1 and S10 refers to the default or Gardenhose sample, respectively. Labels with P1 or P2 refer to either half of that stream, e.g., S1P1 means the first half of the 1% sample. The omission of a P{1,2} part refer to the entire stream.

Firstly, we observe that up to the top-100 items, the two lists are identical: they contain exactly the same tweets, uniquely identified by their id, (Figure 4.36b), and they have the exact same ranking (Figure 4.36a). In other words, if one is only interested in extremely popular tweets, which are but a small fraction of retweeted posts, the 1% sample is adequate. However, if one wants to see the bigger picture, and go beyond the first top-100, the 1% sample starts being problematic.

More specifically, correlation drops to 0.9 when we consider the 10K most retweeted posts. Note that 10K tweets are a very small subset, compared with the entire dataset. As a measure of comparison, the 1% sample returns more than 100K tweets per hour. Despite the high correlation at top-1K and top-10K, it is clear that the 1% sample results in reduced quality, as more items are considered. It is important to note that we obtained similar results when using Kendall $\tau - a$, which only considers common items. Therefore, the drop in correlation is not only due to dissimilar sets. The ranking is affected because the 1% sampling policy does not obtain medium-sized retweets as frequently as the 10% sample.

Regarding the halved streams, we observe that the two Gardenhose subsets (S10P1-S10P2) exhibit high correlation, even at the top-10K items. Moreover, the 1% sample shows the same correlation with these two subsets (S1-S10P1, S1-S10P2). This means that the top-most retweeted posts are retrieved multiple times with the 10% sample. On the contrary, this is not the case for the 1% sample (S1-S1P1), validating our claim regarding stale rankings. We expect the correlation to be even lower as we increase the most retweeted items.

As we already described, for each of the tweets appearing in our top-10K most retweeted posts, we maintained the number of times it appears in our dataset. We are then able to rank these items (in de-



(a) By interval size (b) By iteration, 15' interval (c) By popularity, 1st interval, 15min

Figure 4.37: Burstiness of retweeting information

scending order), according to the number of times that we encountered them, and compare them against the lists ranked by the actual retweet count, given by Twitter. Figure 4.36(c) shows the result of this experiment.

Interestingly, the top-1 most retweeted post is not the one that we obtain most times, irrespectively of the sample used. On the other hand, we obtain high correlation starting from the top-5. In the long run, the 10% results in a 0.8 correlation between the two lists, whereas, the 1% sample is significantly lower at 0.7. This practically tells us that the 10% returns items at a much higher rate than the 1%. In combination with the plots in Figures 4.36a-4.36b, we conclude that the 1% easily results in stale information.

4.3.5.2 Retweet Burstiness

Viral posts become popular, i.e., they receive a lot of retweets, over a short period of time. The rate at which users retweet information plays an important role in capturing this as an ongoing trending topic. Moreover, a post that rapidly gains attention could be the result of an ongoing event. For this reason, we want to evaluate whether there is a difference between the rates of receiving retweets.

To answer this question, we performed the following experiment: For each of the top-10K most retweeted posts which we extracted from our dataset, we computed how many times we received it within a time period after the tweet was first posted. For instance, with a 5 minute interval, we want to know how many times we received a tweet 5', 10', 15', 20' etc., after it was originally posted. Figures 4.37(a)-4.37(c) show these results.

Figure 4.37(a) shows the following: For each of the top-10K most retweeted posts, we count the percentage of retweets that we received during the first M' minutes after it was posted. Each point in our plot is the average over all of the top-10K most retweeted posts. As expected, when we increase the interval size, more tweets fall within the first interval. It is interesting that more than half of the retweets are received at most within the first hour of the original tweet, while one third of the retweets are received during the first 15 minutes. There is no significant difference between the two sampling policies in that respect.

Figure 4.37(b) shows how the average of received retweets behaves as a function of the i-th interval, after the original post, with a 15' interval size. As we have seen, during the first 15 minutes, we receive approximately one third of all retweets. This value drops to $\sim$12% in the second quarter and to 5% within 3 quarters of an hour. After this point, we receive very few tweets in every interval. Once more, we do not observe any notable differences between the 1% and 10% samples.

Until now, we averaged over all of the top-most retweeted items. As we saw in our earlier experiment, the behavior was different, if we consider lower ranked items. To check whether this holds for burstiness as well, we did the following: We fix the interval size to 15 minutes and zoom in on the first interval. We split the top most retweeted posts to 1K batches, and rerun our previous experiment (computing the average percentage). For instance, "3K" on the x-axis means that we compute the average of the tweets ranked in positions 2001-3000.

A striking result is that the low-ranked retweeted posts receive (in percentage) more retweets during the first interval. The two samples also differ in these lower ranked retweeted posts, with the most notable differences appearing between the items ranked in positions $[6001, 8000]$. Moreover, posts ranked between $[3001, 7000]$ are closer to the total average. A similar result has been observed with the 2nd interval after the post.

4.3.6 Graph Evolution

One of the major assets of any social networking site, such as Facebook, Twitter, Google+, etc, is its social component. Although, the term "social component" is typically perceived as synonymous to the explicit social graph, there is more information which can be used in that direction.

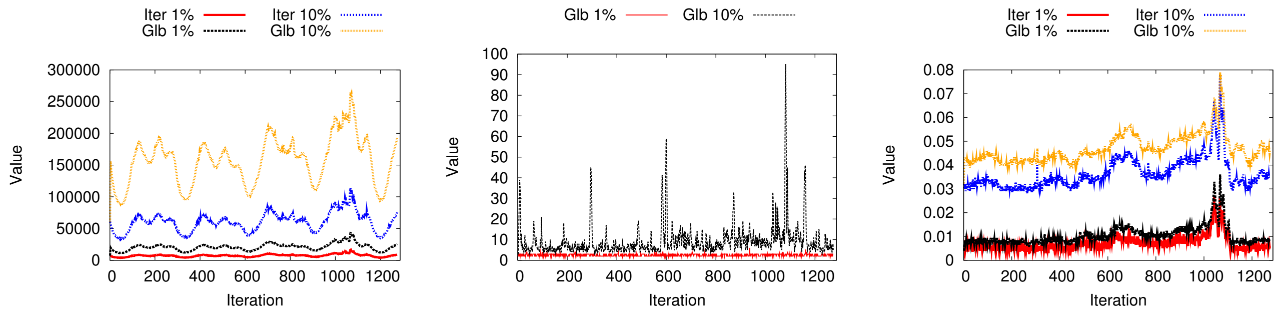
More specifically, users engage in discussions, reply to each other either to form an arguement or respond to questions, endorse views by favoriting, "liking", and retweeting, or simply mention other entities / users in the content they upload. All such actions are explicit forms of interaction between the users. In that sense, the social graph is a subset of what constitutes the social component of the social medium.

We are interested in identifying key properties of the retweet graph, extracted over time from the incoming stream of tweets. We would like to know how well these properties correlate with the ground truth data, as presented in [155] where the entire Twitter stream was used, when we consider the 10% sample.

4.3.6.1 Temporal Retweet Graph

Retweets are a very particular characteristic of the Twitter service. As already mentioned, it allows users to repost tweets, thereby endorsing and acknowledging the original poster at the same time. If user $\mathcal{A}$ retweeted a post, originally posted by user $\mathcal{B}$, then we add an edge from user $\mathcal{A}$ to user $\mathcal{B}$. This is a directed graph, much like Twitter's explicit social graph. Note that we do not focus on a particular tweet in this case, as this would form a star-shaped graph. Therefore, the graph can be the result of multiple individual tweets, posted at different timestamps.

Compared with [155], we want to see how the retweet graph changes over time. Rather than taking daily snapshots of the graph and average them, we would like our graph to incorporate a more continuous notion of time. To achieve this, the edges of our graph are weighted and we decay them over time. The edges are removed when their weight drops below a certain threshold. More specifically, we construct our retweet graph in the following manner:



(a) Size (b) Maximal Connected Component Size (c) Clustering Coefficient (overall)

Figure 4.38: Statistical properties of the extracted retweet graph, over time

- **Step 1:** We use an interval size, similar to the one used for the Retweet analysis. We extract the retweet graph using the tweets that were posted during the first interval. This is our starting graph $\mathcal{G}_0$.
- **Step 2:** Proceed to the next (i-th) interval. Extract the graph of that interval, which we denote by $\mathcal{G}_i$
- **Step 3:** The initial edge weight from a node $\mathcal{X}$ to a node $\mathcal{Y}$ is equal to the number of times that user $\mathcal{X}$ retweeted any post from node $\mathcal{Y}$. We normalize the weights, so that, for each node, the total outgoing edge weight is 1.
- **Step 4:** Decay the graph $\mathcal{G}_{i-1}$, that we have aggregated until this point, using an exponential function. This means that the weight of each edge becomes $w = w * \text{exp}^{-x}$. If that edge drops below a certain threshold t, remove the edge. This implies that the edge is too old, and has not been updated recently.
- **Step 5:** Add the decayed graph $\mathcal{G}_{i-1}$ to the one extracted at the current iteration, $\mathcal{G}_i$. The graph contains the union of the two node sets. We add an edge between two nodes, iff there is such an edge in $\mathcal{G}_i$ or $\mathcal{G}_{i-1}$. If such an edge exists in both graphs, the edge weight is the sum of the two individual weights in either graph. Proceed with Step 2, until all intervals have been processed.

Figure 4.38 shows the results of this experiment. In particular, it depicts the number of nodes that the entire graph contains. We have plotted both the statistics for the aggregated graph until the i-th iteration, as well as the statistics for graph $\mathcal{G}_i$ of each iteration. We note that the graph exhibits a periodicity akin to the one shown in the data volume and sentiment analysis. As we can see, the global graph contains the most nodes of all cases. However, its size does not necessarily increase, as old nodes are discarded, because they did not appear in a more recent interval.

Figure 4.38(b) shows the size of the Largest Connected Component (LCC), as a function of the interval. We observe that the size of the LCC does not share the same periodicity we saw in other cases. Rather, in various occasions, the graph size will increase significantly and then return to normal values. We also computed the clustering coefficient of the 4 graphs we extracted: 2 for the global case and 2 for each iteration (1 per sample). It is interesting that, over time (Figure 4.38(c)), the clustering coefficient of the Gardenhose retweet graph is very close to the one reported by [155]. This, in fact, means that the retweet graph we extract from the Gardenhose, yields similar results to the ground truth data.

4.3.7 Linguistic Analysis

As a final experiment, we would like to see whether there is a correlation between the spoken languages in Twitter, and the ground truth obtained from studies in the physical world. Moreover, we want to check whether there are any differences regarding the two sampling policies. To perform this experiment, we used language detection software [10] and obtained ground truth information from Wikipedia[11,12]. We map each tweet to a language and count the number of tweets with that language. We then derive a ranked list for the languages, based on the absolute counts, and we compare the derived list for each sample with the ground truth using Kendall τ.

Table 4.10 depicts the results of this comparison. Correlation is lower when we consider native speakers, as opposed to lists ranked by the number of people in the world who speak that language. Regardless,

[10] https://code.google.com/p/language-detection/

[11] http://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers

[12] http://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

Table 4.10: Comparison of languages extracted from samples

	Ethnologue	Spoken Popularity
Sample 1%	0.158	0.342
Sample 10%	0.155	0.342

even if we account for the fact that the language detection software is not perfect (i.e., is not 100% accurate), the correlation between the extracted list from Twitter data and the ground truth is extremely low. This holds for both sample sizes. Therefore, there is an inherent bias in the data, which is not due to the sampling policy, but mostly because of the user base of the service itself. This means that researchers on the field of linguistic analysis, who rely on Twitter data, should be weary of this inherent bias.

4.3.8 Efficiency

Table 4.11 shows the efficiency of each experiment for the two sample sizes. In particular, we measure the wall clock time from the point that we started processing the input, up to the point that the full output was written (either to the standard output, or to a file). Our experiments were run on a single Quad-core machine @3.4GHz, with 16Gb RAM, running Ubuntu Linux.

Despite the fact that the actual data differ by an order of magnitude, the running times do not differ by the same amount. There are, of course, various reasons for that, including caching, other processes (e.g., daemons) running simultaneously, process context switching, etc. It is clear, however, that processing takes substantially more time, on a single machine.

Table 4.11: Efficiency of experiments (seconds)

	Sample 1%	Sample 10%
Sentiment Analysis	147.276	2058.264
RT Graph Evolution	175.061	2531.362

4.4 Detecting Events with User-Generated Content

One of the major characteristics of social media in general, and microblogging services in particular, such as Twitter, Tumblr and Plurk, is the rapid updating of their user-generated content, which also occurs in huge numbers. For instance, Twitter now counts more than 200 million active users, with an approximate 400 million "tweets" on a daily basis [13].

This *constant* updating of information has earned them the name "*Live web*" or "*Now web*". Due to the nature of the posted content, its topical diversity and how it spreads through user interactions, these platforms may serve as real-time news reporting and / or crisis management tools, as exemplified with the recent political termoil in the Middle East, with Japanese earthquakes [180], or the 2007 Southern California wildfires [198]. The role of these platforms as real-time news reporting systems became evident very early [124].

As a result, several approaches have been proposed to automatically identify newsworthy real-life information in social media, especially from Twitter given its open data policy that we have talked about in previous paragraphs. Unfortunately, automating *event identification* is not that easy. By *event*, we mean important phenomena with a spatial and temporal dimension in the physical world. Some of the challenges of this task are:

- The large adoption means that we must process in *real time* voluminous amounts of data,
- The content (text) is typically *short*, very noisy, with a lot of slang and personal style, and diverse in numerous ways, regarding location, languages and themes
- the precise location of a user is generally scarce, which means that additional techniques are required in order to address the original problem

Taking into account these impediments, it is no surprise that most existing works that deal with event detection in Twitter simplify the problem by focusing on a specific event type [37, 180, 190]. They

[13]https://business.twitter.com/audiences-twitter, access Aug 2013

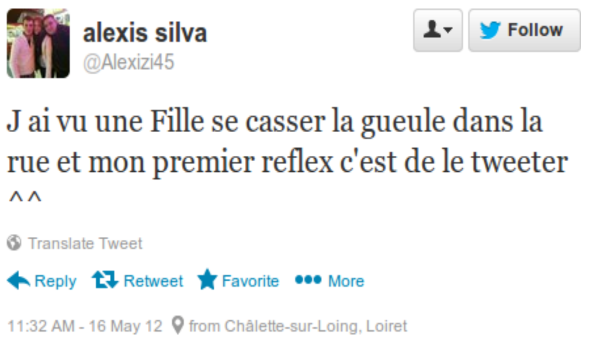
then monitor the stream for specific terms, or *#hashtags* (i.e., user generated topic labels). However, this can only work when the event can be described by a handful of terms, e.g., "[..] *now shaking* [..]" for earthquakes. Clearly, it is impossible to detect *genuine* events by such means, as the descriptive terms are unknown a priori.

Another alternative is to use online clustering [15, 36, 149] or term burstiness [119, 128], so that *trends* emerge as a set of frequently co-occurring terms. However, these techniques suffer from scalability issues, as they are known to be quite inefficient even for a small fraction of the Twitter stream [15, 128]. They are also sensitive to popular terms or large groups of users with similar interests. Spammers are also known to use such terms in their tweets, in order to "blend in" and obtain a higher visibility for their posts [96], only making matters worse. Combined with highly personal [167], and poor writing style the terms describing the event may take a while to surface with these approaches. Furthermore, a trend is *not* necessarily indicative of an event. Rather the contrary, since they are *always* present, as users constantly discuss their interests and popular terms emerge. They can also be the result of recurring phenomena, such as a prominent hashtag, e.g. "Follow Friday" (#FF), or misleading at times: "Dear santa" and "Merry Christmas" were trending at some point in May and June 2012, respectively, despite the fact that they are really out of season. In fact, as we experimentally show, clustering approaches are also ineffective, when trying to return events in a timelier manner, in such a noisy setting.

Therefore, in this thesis, we too undertake the task of detecting events in a stream of microblogs. The main challenge is to devise techniques that work regardless of the category such events belong to, e.g., sports, politics, natural phenomena, etc. To achieve this, we employ techniques grounded on influential theories of emotions, such as *Cognitive* and *Affective* [151]. According to these theories, users are urged to express themselves due to an event from the real world. Figure 4.39 shows indicative tweets, based on real life events of varied gravity. In both cases the users externalized their thoughts as a result of a real life event, but, as the user in Figure 4.39a) puts it, tweeting about that event was his *very first* reaction.

[14]http://twitter.com/137650823/status/202677925663866880

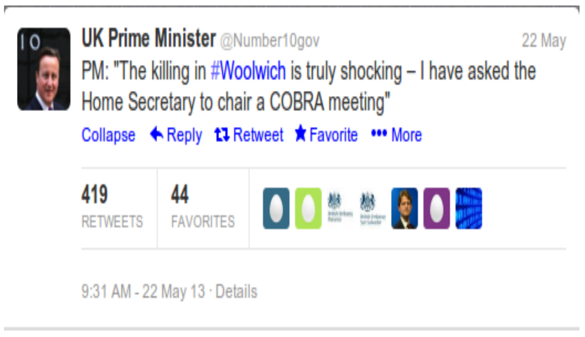
[15]http://twitter.com/Number10Gov/status/337244366181634050


alexis silva
@Alexizi45

J ai vu une Fille se casser la gueule dans la rue et mon premier reflex c'est de le tweeter ^^

11:32 AM - 16 May 12 from Châlette-sur-Loing, Loiret

(a) Everyday incident[14]


UK Prime Minister @Number10gov 22 May
PM: "The killing in #Woolwich is truly shocking – I have asked the Home Secretary to chair a COBRA meeting"

419 RETWEETS 44 FAVORITES

9:31 AM - 22 May 13 · Details

(b) Woolwich incident[15]

Figure 4.39: Tweets reflecting real-life events.

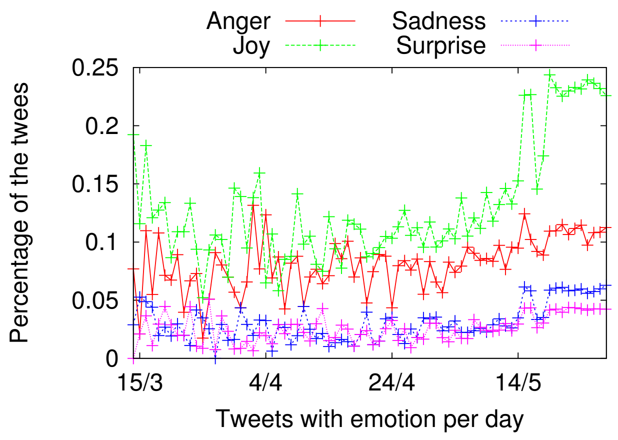


Figure 4.40: A timeseries on the daily emotions identified in the Twitter stream, between March 15 and May 24 2012

By monitoring the Twitter stream we can access these reactions. Moreover, we argue that such tweets will not be a flat description of the event, but will also convey the user's *emotional state*, partially disclosing how it affected them. An event can then be modeled as a *time*- and *place*- related phenomenon, which triggered a significant change in the emotional state of a (potentially large) group of people and our goal is to automatically capture such sudden changes.
Figure 4.40 validates our claim: We plot the relative occurrence of the 4 most prominent emotions, from a sample of the Twitter stream, between May and March 2012. We ommit neutral tweets, which we assert to be non-informative. Surges in *anger* in early April are related with the Syrian uprising, whereas the high values of *joy* towards the end are due to the Champions League final, the Eurovision song

contest etc.
To address the sheer volume of data, we employ online event detection techniques applied on user input, thereby reinforcing the *social sensors* naming convention. More specifically, we use *aggregate* information, making our approach scalable and efficient. The general idea is to group users together and assign them to a virtual sensor, which will monitor their emotional state over time and space, in an online fashion. Information aggregation also ensures that the identified events are of interest to a large group of people. Our technique can be run in parallel, achieving high throughput and scalability, managing to efficiently process large volumes of data.
Overall, our contributions can be briefly described as follows:

- We set up a rigorous framework to aggregate emotions as a means of identifying real-life events.
- We evaluate the importance of emotions, coupled with temporal and spatial information in event detection, and demonstrate their vital role in this data mining task.
- We propose a technique to associate a user with spatial information, a task that must be taken into account in order to address event detection. The proposed technique is efficient and can be applied online, so that event detection may be performed in real-time.
- We present a detailed end-to-end architecture of our approach, which we compare against the state-of-the-art in event detection, using a large crawl of Twitter data, received in a streaming fashion.
- As part of the infrastructure, we provide a contextual user interface to display the identified events in real time. The interface shows all information that we use to identify events, to provide a better understanding of the event and its characteristics.

We also note that, contrary to alternatives, our approach inherently serves a dual purpose: detect new events **and** monitor a group's emotional reaction to it. This could prove extremely useful in decision making or social sciences. Both the way that we identify events and its contextual visualization help in this direction.

The rest of this Section is organized as follows: Section 4.4.1 presents existing work on event detection techniques. Section 4.4.2 discusses our event detection model and algorithmic approach, followed by Section 4.4.3 which discusses our approach for extracting spatial information from user-generated content and evaluates the proposed technique. Section 4.4.4 gives an engineering perspective of our system, followed by Section 4.4.6 which experimentally evaluates our event detection approach.

4.4.1 Related Work

Despite our work's ties with psychology and sentiment analysis, it is impractical to provide a detailed overview of these fields. Therefore, we focus on key aspects that relate to our problem and discuss research regarding event detection from microblogs, which is our main objective.

Psychology. Emotions are a major discipline in psychology, and several theories have been proposed to understand them [130] from various aspects, including evolutionary, social and cultural, as well as procedural. The basic intuition of our work can be seen to have grounds on procedural as well as cognitive and affective theories [151]. The latter, in particular, argue that emotions are the result of an external stimulus (i.e., an event), which will influence a person's attitude or behavior. Under such an affective state of mind, users are more likely to externalize their thoughts.

Sentiment analysis. Sentiment analysis and opinion mining [163] have been broadly studied in various domains and settings. Textual content is classified to a positive or negative class, and works have been proposed for Twitter in particular [33, 93]. Extensions deal with strength of polarity or by considering more target classes [41, 126, 129, 152]. Our current research differs from this body of work in that sentiment analysis is for us a tool to achieve our goal, i.e., event detection of real world phenomena. Therefore any such proposed technique can fit our algorithmic framework.

Event detection in Social Media. Event identification from Twitter appears to receive an increasing interest lately. Early works focus on events of specific types, e.g. earthquakes [180] or news [181]. They whitelist specific keywords and phrases, e.g. "[..]*now shaking*[..]", or sources of information, respectively. Evidently, these approaches are inapplicable for type-independent event detection, in a medium as dynamic and diverse as Twitter.

A closely related concept is *trending topics* or *trends*, i.e., terms which gain in popularity over a period of time. Trends are practically bursty phenomena [119] of term or hashtag cooccurrences. However, trends are not necessarily indicative of events; rather the contrary, since they are *always* present. For instance, a big fan base discussing their popular music idol, easily results in a trending topic, regardless of an event actually happening. They are also related with recurring phenomena, such as TV shows, or *memes*, e.g., the "Follow Friday" (#FF) hashtag. Given that our definition of an event ties it to a specific location in the physical world, techniques for spatiotemporal burstiness [128] might be considered. However, these share the same inefficiencies with the classical bursty approaches, and are also computationally expensive.

A type-independent approach was proposed in [36], where the authors applied online clustering through appropriate similarity measures. However, their methodology was meant for Flickr, an online photo-sharing service, with characteristics very different from Twitter: i) Shooting a photo requires the physical presence of user *u* at location *l* at time *t*. This information is not always available. ii) Users select the tags of their photos carefully, in order to maximize their visibility. iii) Microblogs are voluminous and updated at a very fast pace.

The most closely related work to ours is [221], which is the current state-of-the-art for event detection in Twitter. The authors employ wavelet-based techniques to capture important differences in the "*energy*" of individual terms in sliding windows. Tokens are then used to represent nodes in a graph, where edge weights encode strength of cooccurrence between terms. Subsequently, they apply a modularity-based graph partitioning algorithm to obtain groups of terms that share similar burstiness patterns. Evidently, this method has huge memory requirements, as it must maintain a sliding window with the occurrences of every token, even if it was encountered only once, because it may become bursty in the future. To address the quadratic complexity of the clustering step, the authors filter out tokens with the *median absolute deviation*. However, when applied online, this measure filters out important tokens as well. Moreover, at the conceptual level,

the authors' claim that "emotional expressions are not useful in defining events". We argue the exact opposite, motivated by emotional theories. We return to this approach in our experimental section.
The works in [35, 176] aim to better understand the temporal information in tweets (e.g., "[..]*tomorrow I*[..]"), and build full calendars of events. Finally, we note the works in [99, 161] that deal with the quality of extracted events from Twitter. These techniques address an orthogonal problem, and they could be applied to our output.
Most of the systems discussed above also contained a visual component, to present to the user the extracted information, depending on the output type. For example, [180] provided a web-based earthquake alerting service, simulating an online seismograph for Japan. The information presented to the user was the time and place where the earthquake occurred, as identified by the proposed system, as well as the feed of tweets talking about earthquakes.
In an extended version of [221] [16], the authors presented a User Interface to their approach, but for this particular purpose they limited themselves to a very specific event type: the SGE 2011 elections. A distinctive difference with this work, is that we do not constrain ourselves to events of a particular type, when it comes to visualization. Most importantly, we are interested in providing as much information regarding the event as possible, to help users make informed decisions. We also map users to location, using custom location extraction techniques which is assumed to be known *a priori* in [221].
A visual perspective for event detection was also undertaken in the TEDAS [135] and TwitterStand [181] systems. In both cases, the main issue is visualization of information (i.e., events), rather than extracting them, as events are taken from whitelisted sources, which are practically news reporting agencies / channels. Neither of these approaches consider the users' reactions to identified events, which we do, and the system in [135] lacks a spatial information component.

4.4.2 Modeling & Detecting Events

In this section we formalize our definition of an event, and proceed with our problem statement. Similarly to [34], we define an event as

[16] www.hpl.hp.com/techreports/2011/HPL-2011-98.pdf

follows:

An event **e** *is a real-world phenomenon, that occurred at some specific time* **t** *and is usually tied to a location* **l**.

However, we are only able to monitor the aftermath of the event, i.e., its effects on actual people who provide their input. According to influential theories of emotions [151], the event will have a significant impact on the emotional state of the users that experienced it. Because of this, they will be urged to externalize their emotional state, i.e., the way they feel, and will be inclined to post a message about it. Therefore, we can model the emotional state of a user as the number of tweets they post conveying one of several emotions or moods [129]: *excited, sad, angry, confused*, etc. Since an event from the real world is by definition tied to a location, we expect the *first responders* to be geographically linked as well.
Taking all that into account, our problem can be stated as:

Problem 3 [Event Detection] *Given a time ordered stream of tweets as input, we want to identify those messages which i) alter significantly and abruptly the emotional state of a (potentially) large group of users, and ii) can be traced back to event e.*

This definition fits nicely with an outlier detection formalisation, where we observe a sudden and significant change in the emotions of users, with respect to the recent history. Monitoring individual users is very inefficient resource-wise, and will not provide significant clues regarding the event anyhow. It also raises ethical questions at best, as being very intrusive on a user's privacy.
To overcome these limitations, we use *aggregate* information, extracted from larger user groups, $\mathcal{G}_i$. Users are clustered together according to their geographical location, extracted from available information. We then monitor the emotional state of each geographically distributed group independently of the others and report an event when the group's *cumulative* emotional state changes suddenly. Note that this approach covers inherently the part of the definition that wants the event to affect large groups of users.
Instead of putting all users to a single group, which has no local coherency, we decompose $\mathcal{G}$ into smaller groups $\mathcal{G}_i$ and organize them

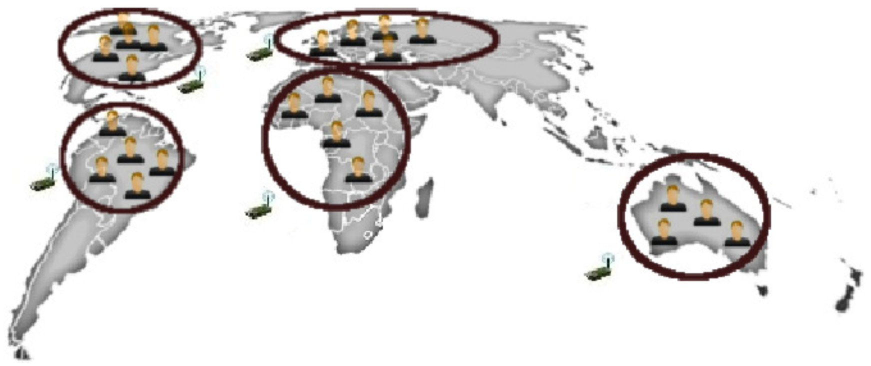


Figure 4.41: An example assignment of groups to virtual sensors.

hierarchically. We denote $\mathcal{G}_i^j$ as group i at level j, assuming that leaf nodes are at j = 0. The hierarchy can be administrative (e.g., country, state, etc.), or constructed algorithmically, e.g., via hierarhical clustering. For a fixed level j in the hierarchy, it holds that $\cup\mathcal{G}_i^j$ = $\mathcal{G}$ and $\cap\mathcal{G}_i^j = \emptyset$, and $\mathcal{G}_i^j = \cup\mathcal{G}_k^{j-1}$. Evidently, this decomposition is a trade-off, providing high-level granularity versus a higher need in resources. We then assign each group $\mathcal{G}_i^0$ to a virtual sensor s_i, which *senses* (i.e., reads) all of the tweets from that group. Sensors at higher levels gather information from their children. Figure 4.41 shows an example of user grouping, with their assigned virtual sensor nearby. [17]. Upon arrival, each tweet is classified to one of the emotions that we monitor. Using an *aggregation interval* a (e.g., a=1min), the sensor produces a single value for each emotion, which is the respective *count* of tweets conveying that emotion during that period. The aggregation interval acts as a discretization unit, to cope with the streaming nature of the medium. The sensor operates over the w most recent points with a sliding window. This results in a much simpler model than the more intricate, 2-stage, multi-level wavelet coefficients of [221]. The combination of a and w specify the history length, based on which the sensor will identify events.

Assume, for instance, a sensor s_i, with a = 5 minutes and w = 12. The sensor maintains a history of the past 5 $\times$ 12 = 60 minutes. Every 5 minutes, s_i will process a single value for each emotion, extracted from the tweets received during that interval from the group of users

[17]World Map image from http://www.wpclipart.com/

that it monitors. The oldest point will be discarded and the new one will take its place.

4.4.2.1 Approximating the Emotional State Distribution

Given that a user's emotional state is a result of several factors, it would be unfounded to assume that it will follow a predefined distribution, much less a static one. In fact, we need to approximate it and maintain it efficiently in an online fashion. To achieve this, we can estimate the Probability Density Function (*PDF*) of the distribution, by observing the reactions of each group $\mathcal{G}_i$. Non-parametric models are a great fit for this purpose and kernel estimators have been shown to achieve good performance for this task [98], while being efficient.
Kernel estimation is based on the idea that each point distributes its weight in its surrounding area, and the *kernel function* describes how this is done. The function $f(x)$ which describes the distribution to approximate is given by the following equation

$$f(x) = \frac{1}{|\mathcal{T}|} \sum_{r \in \mathcal{R}} k(\overline{r} - \overline{x})$$

Here, $\mathcal{T}$ is the actual set of values that we want to approximate, $\mathcal{R}$ is a sample of the data, that each sensor s_i maintains, and $k(x)$ is the kernel function that describes how each data point distributes its weight. Given that the choice of the kernel function has little significance over the estimation output [185], we use the Epanechnikov kernel, which has a closed form integral, and can thus be computed very efficiently. The Epanechnikov kernel is given by the following equation

$$k(x) = \begin{cases} (\frac{3}{4})^d \frac{1}{B_1 B_2 .. B_d} \prod_{1 \leq i \leq d} (1 - (\frac{x_i}{B_i})^2) \\ \qquad\qquad\qquad \text{if } \forall i, 1 \leq i \leq d, |\frac{x_i}{B_i}| < 1 \\ \\ 0, otherwise \end{cases}$$

where B_i is the kernel's bandwidth, computed with Scott's rule [185], $B_i = \sqrt{5}\sigma_i |\mathcal{R}|^{-\frac{1}{d+4}}$. The kernel is suitable for multi-dimensional data and σ_i is the standard deviation for the i-th dimension (i.e., emotion),

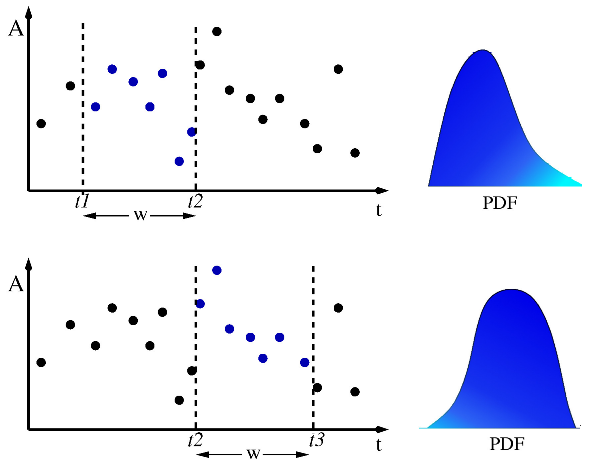


Figure 4.42: Approximating the data distribution in a sliding window.

which can be efficiently and accurately maintained in an online fashion. For simplicity, we ignore the interplay of emotions, and set d = 1. Finally, we note that values are normalized in the $[0, 1]^d$ space. However, we do not find this really restrictive: As a straightforward approach, we can normalize with the maximum value allowed by the system's architecture (e.g., $2^{32} - 1$ for int). Alternatively, we could rely on system specification requirements regarding the load it must sustain, which will also be an upper bound (within constant factor) on the values it can process.

Since we operate under a sliding window model, we need to efficiently approximate the distribution of the data which currently fall within the window. Figure 4.42 graphically portrays this requirement, demonstrating for two consecutive time instances the contents of a sliding window (points in blue) and their respective PDF. As time advances (from top to bottom), new points arrive and expired ones are evicted. Therefore, we must update our kernel estimation at each timepoint; using data aggregation during time intervals, instead of monitoring a stream, makes this computation tractable.

In order to approximate the data distribution, we need to i) maintain online a random sample over the data that fall within the window w, and ii) keep track of the standard deviation σ of the values within w. Both of these values are very easy to maintain in a streaming environment. We use "chain sampling" [26] to produce the random sample, which will randomly select a point s from the sample to evict, regardless of s being expired or not, and replace it with the new point p. Although we could maintain the entire stream, given enough resources, this is not necessarily a good idea, as shown in Section 4.4.6. Moreover, sampling serves as an indirect way for filtering spurious bursts.

4.4.2.2 Event Detection

Having our online kernel density estimator in place, we can now use it to identify changes in the data distribution. The rationale is to identify events on the basis that the most recent aggregate emotional state of users was not "as expected", according to what we have seen so far. Therefore, if a sudden change was observed, this could be caused by an external phenomenon.
Due to our "chain-sampling" approach, we always maintain a sample which reflects the latest distribution from the data, and consequently, the most recent emotional state among the users. Similar problems have been examined in sensor networks [197], but we have the advantage that i) we can maintain the full window, if we want to, as we are not as heavily constrained on our resources, and ii) an event of significance will have a more lasting effect on the users, so we expect at least one point to be inserted in the sample.
To characterize the new point as a significant deviation, we first compute its probability mass over the sample $\mathcal{R}$, according to our kernel $k(x)$. More specifically, for each new point p we evaluate the quantity

$$P(p,r) = \frac{1}{|\mathcal{R}|} \int_{[p-r,p+r]} \sum_{t_i \in \mathcal{R}} k(x - t_i) dx$$

The value r is the neighborhood range, within which to search for points from $\mathcal{R}$. From the definition of the Epanechnikov kernel, the values need to be in the $(p_i - r - B_i, p_i + r + B_i)$ range, to contribute to the integral. If that probability $P(p,r)$ is below a certain threshold, we say that this tuple is an outlier. In our setting this means that a significant change was detected in the emotional state of the observed population. Since this could be the result of an occurring event, we should trigger additional mechanisms to describe it. Therefore, event detection is decoupled from event description.

4.4.3 Extracting Spatial Information

As we have already discussed, we group users based on their spatial proximity. Spatial information has become ubiquitous over the years in applications. In fact, we can see a clear shift from the "spatial is special" argument made in the '90s and early 2000 [19] to a general adoption of a "spatial is everywhere" statement. One of the reasons of this transition is that locational information can be used to map information from the online back to the physical world, as we do in our case. Moreover, we can use this type of information to contextualize data and provide localized recommendations [199] through Location-Based Services (e.g., Gowalla, Foursquares, etc.), and multiple techniques have been proposed that rely exclusively on this aspect [76, 108, 180, 81, 216]. Finally, such is its importance that concealing the location of a user is actively researched in privacy preserving data mining [13, 92, 226, 95].

However, spatial information regarding the users is generally scarce on social media, despite its central role in today's applications. The reason is that most of the existing techniques expect geodetic coordinates to provide their service effectively, and rely on GPS-enabled devices, e.g., smartphones, that provide highly accurate coordinates. Unfortunately, users that share their location through a GPS-device are currently far less than those who do not. Therefore, a significant amount of information is lost, unless we can link them to a physical place as well. Figure 4.43 demonstrates the number of Twitter users received through the Gardenhose (10% sample) over a 2-month period, who share their location through a GPS mechanism as opposed to those who did not. Almost half of the users who provide GPS location (green bar) have done so through a status update, whereas the other half (grey bar) give GPS location in their profile. Regardless, these users make up only 1% of the Twitter users, whereas approximately 60% provide textual information. Even if we explicitly asked for GPS-enabled tweets, the figure practically states that most information on Twitter would remain idle, which is clearly underutilization of information.

On the bright side, most social network users share their location publically on their profile in text form, including, for instance, the city, state, and country (or equivalents) they live in. Therefore, they pro-

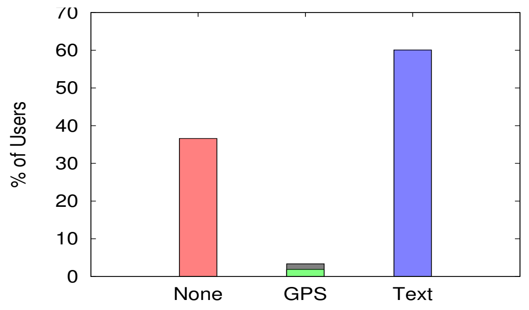


Figure 4.43: Distribution of users based on how they discolse their location

vide a coarse view of their whereabouts in general, usually at city level granularity, or they attach such information to each of their status updates. Being able to pinpoint users at the city level is still important and sufficient for a variety of applications, including our case (newsworthy event detection). Other applications are spatiotemporal burstiness [128], sentiment analysis or even demographics, which has been historically handled at a higher-than-GPS level.

The problem of extracting the location of a user in terms of (*latitude, longitude*) given a textual query is known in the literature as *geocoding*. Building a geocoding service is not easy, mostly because of the need for a complete reference database, as well as the heavy development and fine-tuning it entails [187, 54]. Moreover, despite the abundance of online geocoders, relying on these services as external resources is impractical and sometimes impossible as a result of their terms of use. Given the hard limits that the services pose on their daily query quota, it would take several months to geocode, e.g., all Twitter users, which are approximately 140 million [18] at the moment.

Motivated by the aforementioned problems, and a need for high volume of localized data, in this thesis we tackle the problem of *geocoding* locations, provided by the users as textual information. Unlike existing sophisticated and complex algorithms, which are common in online map services [1, 2, 3] we rely on software and data which are publically available online, making our approach practical and easy to implement. We present a simple, lightweight, yet efficient algorithm to geocode user locations and place them on the map, at the best possible granularity. Finally, our approach is effective and is able to

[18]`https://business.twitter.com/basics/what-is-twitter/`

augment the pool of location-mapped users significantly, as we experimentally demonstrate on a large corpus of Twitter users.

4.4.3.1 Related Work

Our main objective is to return geodetic coordinates ("latitude, longitude"), which are the nearest to a location, represented in text form. This process is commonly known as *geocoding* and its common use refers to mapping street addresses to coordinates. Note that effective geocoding in that sense requires complete reference datasets of street names, which are unavailable to the public for all countries. Moreover, existing methods assume that the query location will be well-formed, or will adhere to some structure to guide the search, e.g. comma separation of administrative hierarchies: *Street No, Street, City, State*. Early works on geocoding investigated address tokenization techniques, employing rule-based or Hidden Markov Model [60, 156], because non-standard address formats were used. Note that according to a recent survey [94], the problem still persists in most countries.
These methods, however, used public health records, hence, sufficient information on the location of the patient (city, county, state) was present, and were constrained within a single country. On the contrary, our approach is challenged by high diversity, as social networking sites are used by people around the globe. Moreover, users hardly ever provide a street level location and write in their personal style. Location nicknames (e.g. "Fog City" for San Francisco), abbreviations ("LA" for Los Angeles, or "KCMO" for "Kansas City, Missouri") and (intentional) mispellings (e.g., "Laweezyana") are also customary. Finally, note that city-level granularity is sufficient for our purposes, yet we still need to overcome the other shortcomings.
The most relevant work to ours is [187], used in commercial online map service [3]. There are several technical differences in our approach and theirs, with the most basic being that we do not have access to a complete reference dataset, but rely on online resources as our primary data, making our approach easy to implement. Moreover, we employ lightweight, yet efficient algorithms, unlike the sophisticated algorithms presented in [187], which rely on mature commercial technology [54] and parameter fine-tuning. We stress that, although

our goals overlap, our incentive is not to compete with these technologies, but provide simple, yet effective, algorithmic mechanisms to geocode information. This objective is driven by our need to perform online event detection using Twitter data, that we receive as a stream. Exploting online resources for geocoding has been researched in the past [28], but with a different goal in mind: to discriminate and accurately identify the location of a street address. The idea stems from the lack of fine-grained information in common online datasets (e.g. TIGER), and the authors counter this problem by using tax and real-estate sites, which provide additional information on the census blocks. Since address level location is scarce in online social networks, this methodology is inapplicable in our case.
Other approaches propose to use online resources to automatically construct and maintain gazetteers, which are essentially the datasets of reference locations used in geocoding and other applications (e.g., named entity recognition) [236, 203]. Their advantage is that document level linking -- which is common in Wikipedia -- corresponds to spatial proximity or relation (e.g., administrative hierarchy). Nevertheless, the goal of building gazetteers is orthogonal to ours. The better the accuracy of gazetteers, the better our approach will perform. Given the public availability of datasets with good quality and easy-to-crawl resources, we do not currently consider these methods.
Similar in spirit are the works in [65, 188] in that they also try to derive a location from user tags. However, their approach differs in that users carefully select tags to describe a photogoraph because that will increase their photos visibility, unlike the social network setting. Moreover, photographs are usually about landmarks, which can be accurately identified and placed on a map. Finally, the Flickr service itself allows users to geocode their photos by selecting a place on a map, which can provide an accurate reference dataset of location-related tags.

4.4.3.2 Problem definition

The problem of geocoding basically means to transform a location of type A, to another --usually equivalent-- location of type B, through a meaningful mechanism. In our setting, the goal is to map textual locations, provided as user generated input, to a (*latitude, longitude*) pair or an equivalent identifier (that can be mapped back to these coordinates). The problem can then be defined as follows (tailored for our setting):

Problem 4 *Given a user location $\mathcal{L}$, provided as a set of tokens (terms), find a set of geodetic coordinates (i.e., latitude, longitude), that can accurately describe $\mathcal{L}$.*

Given this definition, our research focuses on the following desiderata:

1 Using available text descriptions from online resources

2 Investigate the accuracy and efficiency of simple techniques

4.4.3.3 Problem setting

Note that the geocoding definition is general enough and several transformation mechanisms could be employed. Moreover, the level of required accuracy is subjective, and usually is application-dependent. For event detection, achieving city level accuracy is sufficient. In that respect, returning a set of geodetic coordinates that describe the city (e.g., the center of the city) is considered enough. We are also highly interested in efficiency, due to the voluminous amount of data that is generated in social networks.

Unfortunately, existing online resources are notorious for being incomplete or inaccurate [173], even for the US. To address this problem, we utilize the wealth of information available on the Web and fill in missing gaps. As we are not interested in street level accuracy, the GeoNames dataset [19] is sufficient for our purposes. Despite its richness, the dataset lacks a hierarchy of administrative units (e.g. county, state, etc) for most countries. To overcome this, we can crawl online resources, such as the Flickr places dataset, or even mine Wikipedia, which will provide better information. For instance, we can use wikipedia to construct reference lists for abbreviations, synonyms or nicknames of locations (e.g. "Fog City" or "Frisco" for "San Francisco", 2-lettered US and Canada states abbreviations, etc.)

[19] http://www.geonames.org/

4.4.3.4 Online geocoding

A straightforward solution one might consider would be to query online geocoders, such as GeoNames server, OpenStreetMaps server, or even online mappings services. However, online mapping services pose a hard limit on the number of the allowed daily query quota. Even if such quotas were not present, crawling "politeness" should still be honored, which would eventually pose an upper bound on the number of submitted queries.
Although such an approach would work well for a few locations (say, a few thousand), it is infeasible for online social networks. Twitter currently claims 140 million users, whereas Facebook has nearly 1 billion active users per month [20]. Even if we take into account that 60% of the users provide their location in text form and assume a high query quota is allowed (Yahoo! allows 50K queries/day), it would take several months to geocode all of them. Finally, online geocoders are accessed through HTTP requests, which is known to be a slow communication protocol, and do not allow batch processing requests.

4.4.3.5 Data Cleaning

Another issue is that users tend to write funny quotes in their profile such as *behind you*, *why do you ask*, etc., so as not to disclose their location. Therefore, it is mandatory to clean such data to come up with location information that will be of use.
For instance, users multiplex different encodings, charsets and fonts to make up their location. Though these locations are readable and well-understood by humans, they can not be mathced to an actual location, unless they are transformed to another encoding. Users also tend to surround their text with emoticons or various shapes (e.g. hearts, stars, bars, etc.) to make them fancy. We have manually compiled such lists and intend to make them publically available.
We also have lists to remove japanese, korean and arabic text. A major issue with japanese and korean is that text tokenizers and analyzers --including the one used by Lucene-- are not well suited for these languages, which do not use whitespace delimiters. Assuming

[20] `http://newsroom.fb.com/content/default.aspx?NewsAreaId=22`

that an appropriate tokenizer is present, our algorithm is still applicable. Due to this fact, such locations resulted in a high rate of false positives. Given that we are unable to understand these languages, we removed all such locations, using custom built lists from online dictionaries, alphabets and the actual text from the locations.
Finally, given that (custom) data cleaning is a time consuming process, we could make use of machine learning approaches for this purpose. Sentences that do not refer to actual locations may emerge as topics and we could employ topic extraction algorithms (e.g. Mallet [21]). Alternatively, we could use co-occurrence relations or even maximal sets to identify these. We plan to thoroughly investigate these approaches as part of future work.

4.4.3.6 Algorithmic description

Algorithm 4.3 gives a detailed description of our lightweight geocoder. The algorithm takes as input the reference database (our *gazetteer*) $\mathcal{G}$ and the associated hierarchy of locations $\mathcal{H}$. In our approach a 4 level hierarchy was used: i) suburbs and towns, ii) counties, iii) states and iv) countries. Coarser or more fine-grained hierarchies could be employed, but we see that these levels work well in practice and align well with human intuition. The algorithm also takes the location $\mathcal{L}$ that we wish to geocode.
The entire process consists of 3 phases: *cleaning*, *tokenization* and *searching*. The cleaning process lower cases the characters to simplify subsequent steps. Due to the nature of the social networks datasets, cleaning also means that we need to correct fonts and encodings, and convert them to the one used in $\mathcal{G}$. Once this has been performed, we remove decoratory symbols (e.g., hearts, stars etc) that users insert in their location. Unfortunately, a drawback of using commodity software is that ordinary tokenization will not recognize these characters as text delimiters. The problem that then occurs is that tokens will not match valid locations in $\mathcal{G}$. As a final step, we remove characters which we can not process, e.g., japanese in our case, although this step should be optional in a complete system.

[21] `http://mallet.cs.umass.edu/`

Algorithm 4.3 LightGeocoder

```
    Input: Reference database (gazetteer) $\mathcal{G}$, Locations Hierarchy $\mathcal{H}$, Location to geocode $\mathcal{L}$
    Output: Set of possible geodetic coordinates $\mathcal{C}$

 1: //Location preprocessing
 2: $cleanLocation \leftarrow lowerCase(\mathcal{L})$
 3: $cleanLocation \leftarrow CorrectFonts(cleanLocation)$
 4: $cleanLocation \leftarrow RemoveDecoration(cleanLocation)$
 5: $cleanLocation \leftarrow RemoveCharacters(cleanLocation)$
 6:
 7: //Search term extraction
 8: $tokens \leftarrow tokenize(cleanLocation)$
 9: $searchTokens \leftarrow \emptyset$
10: $newToken \leftarrow tokens[0]$
11: for ( $i \leftarrow 1; i! = tokens.size(); i$++ ) do
12:     $search \leftarrow newToken +$ " " $+ tokens[i]$
13:     if ( $\mathcal{G}.getResult(search).size()$ != 0 ) then
14:         $newToken \leftarrow search$
15:     else
16:         $searchTokens \leftarrow searchTokens \cup newToken$
17:         $newToken \leftarrow tokens[i]$
18:
19: //Location Matching
20: $matches \leftarrow \emptyset$
21: for ( $i \leftarrow 0; i! = searchTokens.size(); i$++ ) do
22:     $search \leftarrow searchTokens[i]$
23:     $interpr \leftarrow \mathcal{G}.getResult(search)$
24:     if ( $matches.isEmpty()$ != 0 ) then
25:         $matches \leftarrow interpr$
26:         continue
27:     $newMatches \leftarrow matches$
28:     for ( $j \leftarrow 0; j! = interpr.size(); j$++ ) do
29:         $ancestor \leftarrow \mathcal{H}.search(interpr[j], matches)$
30:         if ($ancestor! = null$) then
31:             $newMatches \leftarrow newMatches \setminus ancestor$
32:             $newMatches \leftarrow newMatches \cup interpr[j]$
33:     $matches \leftarrow newMatches$
34:
35: $\mathcal{C} \leftarrow \emptyset$
36: for ( $i \leftarrow 0; i! = matches.size(); j$++ ) do
37:     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{G}.coordiantes(matches[i])$
38: return $\mathcal{C}$
```

Once the cleaning process is done, we tokenize the input location. We apply simple tokenization on whitespaces. We then check whether we can combine tokens with one another, into larger token sequences. Intuitively, the idea is to combine tokens which make up valid locations, e.g. "los angeles" instead of separately having "los" and "angeles". During this step, we do not care about the actual locations that can be retrieved, only whether additonal locations can be retrieved. This step creates a set of search tokens to use in the final step.
Finally, we perform the actual location mapping step. For each location from the search tokens, we get a list of id's, which we call *interpretations*. An *interpretation* is simply a possible location against which a token can be matched, without using other contextual information. For each new interpretation (lines 28-32), we check in the hierarchy $\mathcal{H}$ whether there is a child-parent relation with another interpretation that we obtained for previous search tokens. If we identify such a child-parent relation, we remove the parent id and insert the child's. This way, we increase the level of granularity for the location we have been given. Note that we maintain locations for which we can eventually establish a connection in the hierarchy. Once this process is complete, we query our database $\mathcal{G}$ for the actual coordinates of the locations that we have extracted, which we then return.
As a final note, the (worst case) algorithmic complexity of our technique is $O(\prod inter(token))$, where $inter(token)$ is the number of interpretations that each token may have. However, tokens are usually few (at most 7), and each one is matched to few locations. As we demonstrate in our experimental evaluation, the actual running time of the entire process is minimal, keeping the approach efficient for our purposes.

4.4.4 The TwInsight System

In this section we discuss the components that make up our system, as well as the data and its workflow. There are several components, in order to provide the desired functionality, each of which works independently of the others. Consequently, we can substitute any one of them with better alternatives.

Twitter Data. Our only source of information is the Twitter stream, receiving tweets through the service's streaming API [22]. Having access to the Gardenhose we receive a 10% sample of all public tweets that are posted to the service. Unlike previous techniques, we do not rely on external sources [161, 131] or whitelists [181]. To retrieve the information, we use our crawler, described in Section 4.2.1.

Emotion Classification. Emotions are a key ingredient in our approach, and constitute the basic input to our algorithm (following the tweet). We use a set of 6 emotions proposed by American psychologist Paul Ekman [77]: *anger, fear, disgust, happiness, sadness, surprise*. We also use a neutral (*None*) state, to describe the absence of an emotion. We do not consider neutral tweets, as we think that they are uninformative.

In order to extract emotions from tweets, we have trained a classifier using both structural and semantic features. We consider general punctuation, such as exclamation marks, question marks, quotation marks, etc. We also consider letter capitalization / duplication (e.g. "*yesssss!!!*" instead of "*yes*"), number of retweets, whether it is a retweet (RT) itself, number of mentioned entities, hashtags and urls. The basic rationale behind using structural features of the tweet is that we expect truly spontaneous reactions to contain these characteristics. Moreover, these features are very easy to extract, because they are either directly provided by Twitter (e.g., retweets), or can be extracted through shallow parsing.

We also compiled lists of emoticons from online resources, such as Wikipedia, and by manual inspection of the tweets, which are good indicators of sentiment [93, 33]. Finally, we use lists of words and phrases which are known to be related to emotions. For this purpose, we use the Affective WordNet [142] vocabulary, and the moods dataset [129] which contains several sentiment-tagged words.

It is important to note that classifying tweets to emotions is harder than classic sentiment analysis. The reason is not only the additonal target classes, but rather the inherent ambiguity of certain words and the way we use them. For instance, according to Affective WordNet, "amazing" is related to *surprise*. However, in the sentence "That book was amazing!", the intent is to express *joy*. Sentiment analysis would correctly identify the sentence as "positive", but an emotions classifier

[22] https://dev.twitter.com/

may easily mislabel it. Similarly, *disgust* can easily be misinterpreted as *anger*, as in "I hate eggplants!!!!", but in either case the sentiment will be "negative".

Location Extraction. To extract a user's location, we use the approach presented in Section 4.4.3. More elaborate techniques could also be considered [108, 14], but these require a significant amount of data for each Twitter user, to build an accurate language profile. We remind the reader that we use the GeoNames database [23] and Flickr data [24] to create an administrative hierarchy of places. The administrative hierarchy is necessary to discriminate between places with the same name, such as "Athens" in Georgia, US from "Athens" in Greece. Flickr provides data up to a town or suburb level, which we believe is more than sufficient for our purposes.
Along with every tweet, the Twitter service also sends information about the poster, such as the "Location" field of her profile. Tweets are mapped to locations as follows: First check whether it contains a GPS signal, which occurs when the tweet is automatically posted from a location based service (e.g. Foursquares). Otherwise, use the "Location" field and map the user with our custom gazetteer. Therefore, for that particular tweet, the user is assigned to the sensor that monitors the respective area. We do not consider tweets from users who can not be mapped to a location.

Event Detection. We assume a set of virtual sensors, each of which is in charge of monitoring a specific location. Each sensor runs the event detection mechanism presented in Section 4.4.2, applied on the *counts* of each emotion separately, measured in every aggregation interval. Event detection runs in an *online* fashion, to address the "*as events occur*" requirement of our problem. When an outlier is detected in the emotional state of the users, a signal is raised, notifying the event extraction mechanism to be put into effect. The general idea is to run a lightweight, yet effective, mechanism for event detection, which will trigger the more heavyweight event extraction module when it should be executed, instead of constantly running the latter.

Event extraction. The event extraction mechanism is responsible for finding the most informative terms which describe the event that oc-

[23]http://geonames.org/
[24]http://www.flickr.com/places/info/

curred. The input of this component is a set of tweets (e.g., their ids) expressing the same emotion and were observed during the aggregation interval that triggered the event. The output is a set of keywords which can effectively describe the event. Various techniques can be employed at this stage, e.g., term frequency, TF-IDF, or similarity-based [50, 158, 36]. We employ term frequency in our experiments.
It is possible for certain events (e.g., elections) to invoke different or contradicting emotions. Such cases can be handled as follows: tweets for the same event will (most likely) share a fair amount of common terms and, therefore, can be grouped together. Note that clustering based approaches will also fail, unless the tweets share some terms. The user can then be notified of the event and the emotions in an aggregate way (e.g., 60% happy, 30% angry, 10% sad). In fact, informing the end-user of the different emotions that an event may elicit adds a semantic dimension, which can be very helpful for social scientists and decision makers alike. This is a unique trait of our technique, not shared by other approaches.

4.4.5 Event Extraction Workflow

From the description of an event e and our event detection mechanism, it should be clear by now that we need the following information: a location l, the time of occurrence t, a set of keywords to describe it, and the emotions that were elicited as a result of the event. The standalone subsystems used to extract this information have already been described. Therefore, in the next paragraphs, we describe how we put the pieces of the puzzle together to serve our ultimate goal of event detection.
Figure 4.44 shows a schematic view of the components and their interaction[25]. The Twitter stream is our system's input, feeding two components, namely the *emotions classifier* and the *location extraction* subsystem. Through the location extraction process, each incoming tweet is mapped to a location, which will also be the location of the event (assuming one occurs). As discussed in Section 4.4.2, we use a set of virtual sensors, each of which is solely responsible for a specific

[25]Storage image by Barry Mieny, under CC BY-NC-SA license.

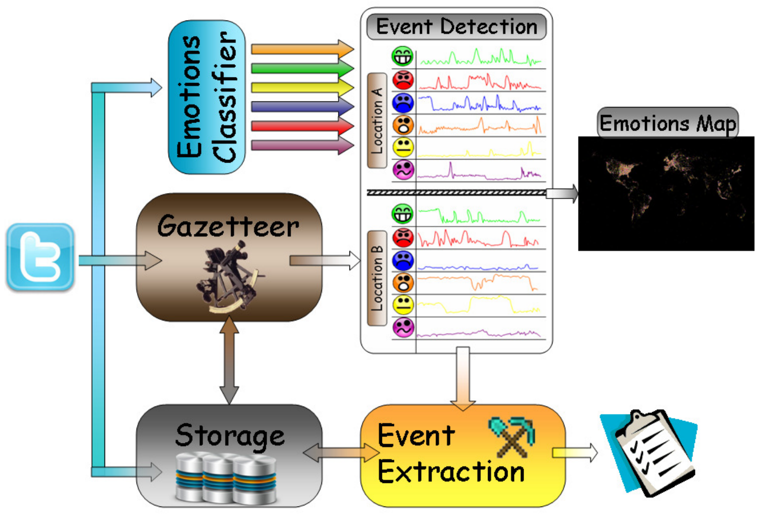


Figure 4.44: Schematic interaction of our system's components

location. Therefore, the tweet will be forwarded to the virtual sensor responsible for the location it was mapped to.
Meanwhile, the tweet has been classified to one of the 7 emotions that we use. If the classifier indicated that the tweet conveys a non-neutral emotion, the virtual sensor will further process it. Otherwise, no additional actions are performed. In any case, we store all tweets in our database. It is worth noting that this approach allows for an elegant integration with spam detection mechanisms: spam tweets can be cast to the *neutral* class, thereby preventing them from any subsequent processing.
All in all, when a sensor receives a tweet for further processing, we already know its location and which of the 6 non-neutral emotions it has been cast to. For each emotion independently, the sensor counts how many tweets it has received during the last aggregation interval, and produces a single value (the respective count) at the end of each aggregation interval. Each such value serves as input to a separate instance of the event detection mechanism we have previously described, running on the same sensor. The event detection module updates its values and identifies whether a surge, i.e., an event, in any emotional state has occurred. In such a case, we report the end time of the aggregation interval as the event's time of occurrence t. Additionally, the tweet ids that caused the peak for that particular emotion are passed on to the event extraction mechanism, which

will go over them and summarize the event appropriately. This final step will provide the descriptive keywords of the event, which can then be presented to the user with all the necessary information: *location, timestamp, emotion* and *description*. Notice how the event extraction step is put into motion only if an event was detected. A sample output can be found in [206].

4.4.6 Experiments

4.4.6.1 Experimental Setup

In this section we present experimental evidence of our proposed techniques, used to extract events from Twitter data. In particular, we evaluate our geocoding mechanism, described in Section 4.4.3, as well as our event detection mechanism, described in Section 4.4.2.
We have crawled Twitter through the Gardenhose between the start of April and end of May 2012, and have obtained a dataset of a little less than 300 million tweets. Considering that the sampling policy is entirely at Twitter's discretion, we have no control over the type of data we receive in terms of content, location, or language, nor the rates in which they arrive. The tweets have been posted by 29 million unique users (identified by their id). Taking into account the analysis of Section 4.3, using the Gardenhose sample is better for our purposes, as we want to track general events in real-time, whereas the default sample is appropriate (in the temporal dimension) only for the most popular discussed topics.
Given our final objective -- event detection --, we filter the received tweets by language language (english, french, german, greek, spanish) and locations (USA and european timezones), so that we can understand what is being talked about, and evaluate our system in terms of effectiveness. After this filtering step, we obtain approximately 11 million unique users. Note that several of them are in fact non-english speakers, however, the default language option is english. This is not a major issue as we can filter out such users further down the processing chain, based on the content of their tweets, using either lexicon-based or machine learning approaches, e.g. [179].

4.4.6.2 Geocoding Results

Out of the 11 million users, we have extracted approximately 2.5 million locations, which means that in a uniform distribution there is an average of 4 users per location. Over the 2-month period that we crawled Twitter, about 150K unique users changed their profile location, which we use for geocoding. Therefore, in the worst case, we are still left with 2.35 million unique locations to geocode. This leads to the following conclusions:

- Users will update their profile location if they are inclined to do so.
- For the most part, for short periods of time (e.g., spanning a few months), the general location of a user remains unchanged. By "general" we mean a broad area where they have mostly been during that period.
- Since users do not change their profile location that often, geocoding it and storing it for future retrieval is meaningful and can be easily performed. Note that in case of users who have enabled a geo-tagging service, their location is available. Nevertheless, these services use a well-formed naming convention, making the geocoding problem much easier.

After applying the cleaning process that we have described in Section 4.4.3.1, we derive a dataset of 1.8 million locations. Within this dataset, the GPS-enabled locations are less than 100K. Table 4.12 summarizes the basic properties of the dataset used for geocoding.

Table 4.12: Location dataset characteristics

Variable	**Value**
#Initial users	29 million
#Users after filtering	11 million
#Uncleaned (unique) locations	2.5 million
#Cleaned (unique) locations	1.8 million
Average token count / location	3.147
Full gazetteer size	3,490,337 entries
Small gazetteer size	1,042,742 entries

Our geocoding approach requires the use of a reference database (i.e., gazetteer). For this purpose, we created two gazetteers, both of which include all of the data that we crawled from the Flickr Places website. Their difference lies in the entries used from the GeoNames dataset, for the countries we want. The first one contains all of the GeoName entries, and we denote this one as *Full*. The second one uses only entries that have been marked as "administrative" (e.g., a county) or "populated area" (e.g., village). We denote the latter dataset as *Small*. Each GeoName entry has been assigned to the closest location of similar administrative type from the Flickr dataset, as we have described in the previous sections, to be able to efficiently exploit the hierarchy. Finally, we have indexed each reference database using Lucene [26], to allow for IR-style retrieval of locations.

Geocoding Efficiency. We start by evaluating our system in terms of efficiency. The geocoding process is written in Java 1.6 and ran on a single node, with Quad Core @3GHz. We measure the time taken to run the LightGeocoder algorithm for each user and report the cumulative time required for all 1.8 million unique, cleaned locations. Therefore, the reported values show how much time is required for the geocoding process in a real system implementation.
Figure 4.45 shows the efficiency of our geocoding scheme, with each of the two gazetteers. Clearly, when the *Full* gazeetteer is used the runtime increases, because each token has more possible interpretations and the index contains more entries to search against. On the other hand, the *Small* gazeetteer is more lightweight and requires less time to come up with the set of possible locations. Although there is a 3:1 ratio in the gazetteer sizes, the processing times differ by a factor of 10; this is can be explained as the hierarchy consists of 4 levels (including countries).
Not surprisingly, both approaches are linear to the number of locations mapped. Therefore, we can cut down the total elapsed time by using multi-threaded solutions or by distributing the workload among more computers. Regardless, the average (per location) geocoding runtime is 3.4 ms when using the *Small* database, whereas it is 14.4ms with the *Full* database, which still allows us to do the geocoding in real time. Finally, the cleaning process takes a negligible amount of time

[26]`http://lucene.apache.org/core/`

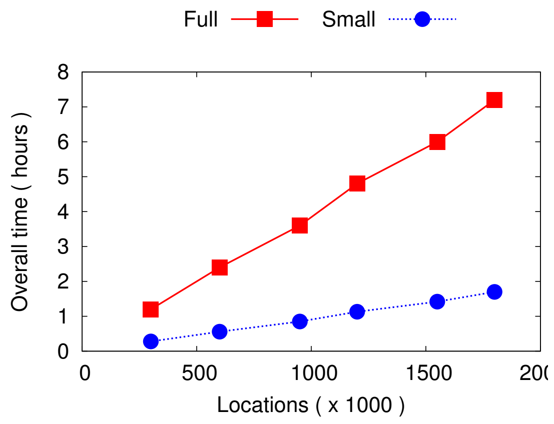


Figure 4.45: Efficiency of geocoding using the Full and the Small gazetteers

(in the order of 10^{-5} sec), and does not influence the overall efficiency, as it is linear in the text's length and performed through lookups.

Effectiveness Evaluation. A basic goal of our research was also to investigate the effectiveness using simple techniques and publically available data. To measure our technique's effectiveness, we performed a series of experiments. First off, and in order to simplify the evaluation process, we assume that if our geocoder returns more than 5 possible matches, then we can not map the corresponding location at the granularity we desire. However, we discriminate between "*overmapping*" a location and not mapping it at all. Effectively, we can see the number of locations that have been mapped, the ones that have been "*overmapped*" and the remaining ones that have not been mapped. Figure 4.46 shows the number of locations (in log scale) that fall inside each of these categories when either of the gazetteers is considered.

The interesting outcome of this experiment is that, contrary to what one might expect, the *Small* gazetter maps more locations than the *Full* gazetteer, especially when it leaves more locations unmatched! Apparently, the large number of indexed entities present in the *Full* gazetteer introduce more disambiguation and make it harder for the algorithm to select a small set of possible matches. This leads us to the conclusion that it is not only important to clean the incoming locations, but we should also pay special care to the reference dataset.

Mapping (or not) more or fewer locations may be statistically useful, but does not realy hint us on how well our system performs. For this

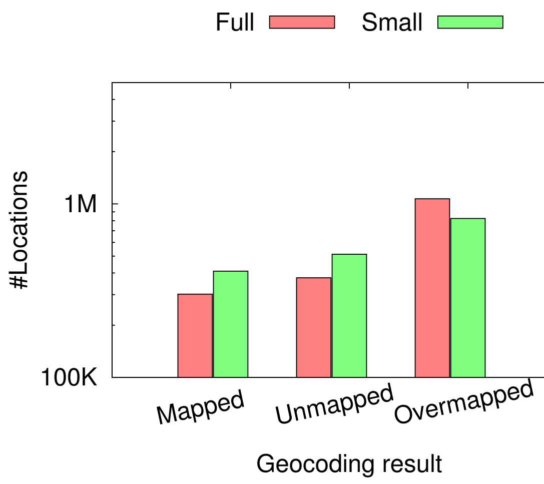


Figure 4.46: Number of mapped, unmapped and overmapped locations by gazetteer usage

reason, we have conducted an *oracle* experiment, querying a service which is particularly suited to the geocoding task. Specifically, we queried the Yahoo! maps geocoding service with 10K locations, randomly sampled from the 1,8M locations that we experimented with. The service returns its response in XML format, with a ranked list of potential matches. If the location could not be matched, an empty ranked list is returned.
We used the default parameters which return at most the top-10 possible locations, along with a GPS location and a small textual description. Each result is also accompanied by a quality element of integer type, which is essentially the granularity (or precision) of the resulting location. Higher values indicate greater precision. For instance, quality values greater than 80 correspond to points, whereas values between 31 and 40 are town level areas.
We initialy present a confusion matrix, which measures 4 values:

- *True positives*: the number of textual description identified as a location by both approaches
- *True negatives*: the number of textual information that have been correctly identified as **not** being locations
- *False positives*: the number of textual information which (our approach) identifed as being possible locations, whereas Yahoo! maps did not consider them as such.

- *False negatives*: the number of locations which we failed to identify as being a valid location, whereas Yahoo! maps returned a ranked result set.

Table 4.13 contains the values for these four measures for both of the reference databases. The matrix is read as a combinatiom of its row and column, for the corresponding gazetteer. For instance, the upper right corner reveals the *true* (vertical) *positive* (row) for the *Simple* gazetteer. Two important things to mention are: i) *overmapped* locations are handled as *unmapped*, on the basis that we can not discriminate between them easily with our current approach, and ii) the *true positives* metric only measures whether both techniques identified a description as being a location or not. It does not consider, whether the same location has been indeed selected, which we discuss followingly.

Table 4.13: Confusion matrix for the "oracle" experiment

	Full		**Small**	
	True	*False*	*True*	*False*
Positive	1699	114	2363	75
Negative	656	7551	695	6887

For the columns labeled with **True** (**False**), higher (lower) values are preferred. Once again, we draw the conclusion that "simpler is better". The small gazetteer performs better than the full gazetteer in all four occasions. In fact, if we carefully observe the matrix, it appears that when we move from the small to the full gazetteer, there is a transfer of true positives to false negatives (a result of "overmapping" locations), and a similar transfer of true negatives to false positives. Clearly, this artifact has been introduced by the GeoNames dataset, since everything else has remained unchanged.
We now drill down on the true positives which we identified with each approach. We want to see if the locations extracted by our geocoder fall within the top most locations returned by the Yahoo! maps. To this end, we measure the percentage of documents retrieved by our approach that can be matched to one of the top-n results by Yahoo!. For instance, if the top-1 result is present in our returned list, we assume that we have a match at position 1. Because we currently avoid

ranking, this essentially means that we have the prospect of returning one of our results as a top-1 value (and stop at that point).

We select up to the top-5 results, returned by Yahoo! maps, which have already been provided as a ranked list. For each result, provided that its quality is up to a city level ($quality > 30$), we extract the closest city location from our crawled Flickr dataset, which is the basis for the hierarchy. Followingly, we check if at least one of our geocoder's proposed locations is within Yahoo! map's top results.

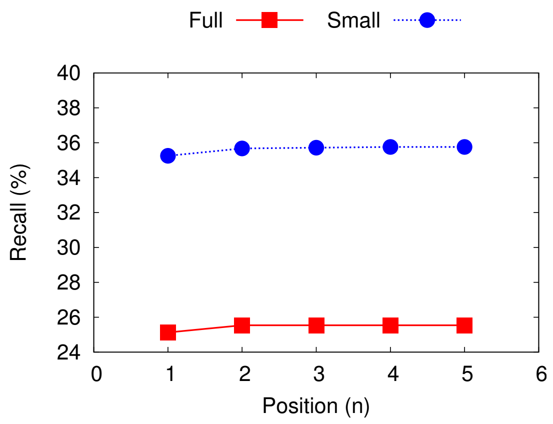


Figure 4.47: Effectiveness of our approach using the two gazetteers

Figure 4.47 demonstrates the cumulative measure we discussed in the previous paragraph. We observe that the simple gazetteer did not only achieve a high true positive ratio (see Table 4.13), but it has been able to correctly identify the correct location at a percentage of 36% with simple techniques. It is also better than the *full* gazetteer by at least 10%. In actual numbers, the simple gazetteer matched correctly around 845 locations, as opposed to a mere 434 that the full one did.

As a final note, it is important to mention that, despite the misclassifications (false positives, false negatives), even with the full gazetteer it appears that we can increase our geo-tagged users data pool by roughly 5% of present **locations**. This translates to several users, especially if we consider the average 4:1 user-location ratio. Clearly, our gold-standard data is quite small, and generalizations should be cautioned, but it still provides good evidence that profile location is useful and can be extracted with lightweight approaches, such as the one presented in this thesis.

4.4.6.3 Event Detection Results

Since spatial information is essential for our event detection mechanism, we only consider those users who have been mapped to a location, using the technique we have already described. In particular, we only consider tweets written in *english, spanish, german, greek* by users from *Canada, France, Greece, Germany, Ireland, Spain, UK, US*. This gives us approximately 33.5M tweets, from a little less than 400K unique users. We order the tweets by their timestamp and replay the stream, feeding them to the event detection mechanism. We apply no other data cleaning, since we would like to perform our evaluation as it should occur in a real-life setup.

Techniques. We have implemented our system, *TwInsight*, and the state-of-the-art, EDCoW [221], in Java 1.6, both of which rely on Twitter alone as input. We run our experiments on a Quad Core machine, @3.5 GHz, with 16GB RAM and report the average of three runs.

Emotions Classification. To train our classifier, we asked from 30 individuals, of varied expertise on Twitter, to annotate a set of tweets with one of Ekman's emotions, or a "None" option, according to their belief that the tweet conveyed (or not) an emotion. They were also allowed to skip tweets they felt unsure about. The tweets were randomly selected from the initial dataset of 300M, to avoid biasing the evaluation, and no consesus phase took place.

This process created a gold standard of nearly 6700 tweets [27]. From these, we removed about 100 tweets that were outside the set of accepted languages, despite the Twitter users having indicated otherwise in their profiles. Taking our objective into account, we performed the following cleaning: For every tweet with an emotion and more than 100 retweets, we created an identical entry in the dataset with a random retweet count of up to 100. We then replaced the annotator's choice with "None" in the initial tweet and kept both versions. Most of these tweets were funny quotes and memes -- a commonplace in Twitter --, and had the same id, thus they could be efficiently identified through simpler mechanisms (counting the id of a tweet), even if they refer to an ongoing event. Finally, we feel that a user *reporting* an event, will *tweet* about it in their own words, than search for an

[27]The dataset will be available upon request

Table 4.14: #Times a trigger was raised, compared to the neighborhood range. a = 1min, w = 30, p = 0.1

	50% Sample		100% Sample	
Range	**Neutral**	**Joy**	**Neutral**	**Joy**
0.001	4977	1637	5315	1942
0.01	4266	1168	4138	1280
0.1	4198	1252	4088	1274

existing one and retweet it.
We experimented with various classifiers in Weka [101], including SVMs and decision trees. Selecting the most frequent class ("None") would give a ~34% accuracy. In the end, a C4.5 decision tree returned the highest accuracy (64.39%), in 10-fold cross validation, with other classifiers yielding similar results. Clearly, these values can be improved, but one should also consider the subjective nature of the experiment. Most importantly, though, this did not prevent us from identifying meaningful events. Finally, techniques such as [117] are applicable, given the high discrepancy in sizes between our gold standar and the experimental dataset.

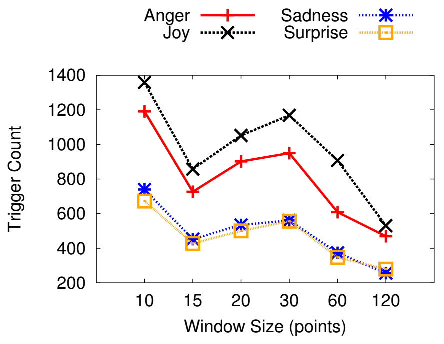
Effect of parameters. In this section we evaluate the effect of parameters of our kernel-based mechanism in the event identification process. In particular, we want to see how they affect the times a trigger is raised, putting the event extraction phase into motion. Although a high number of triggering might fulfill more users' needs, it also means that *TwInsight* is stressed more, especially if these are false positives.
Table 4.14 shows the number of times that the event extraction phase was triggered, for various neighborhood ranges and sample sizes. We use 1 minute aggregation and maintain a window of 30 points, resulting in a history length of 30 minutes. We compare against an approach that relies on simple counting of received tuples ("Neutral"), instead of distinguishing between emotions.
Increasing the neighborhood range results in less triggers, because each point distributes its weight to a broader area. Therefore, new points receive more weight from previous ones, and are less likely to raise an event. Although the raw counts may seem quite high, they account for less than 2.5% of all minutes in the 2 month period.

In addition to the semantic implications of distinguishing among tweets using emotions, this technique also appears to be a more lightweight approach when looking for events, as there is a 3$\times$ cut-down of triggering times. Also note that far fewer tuples are considered, because tweets classified as "None" are dropped. In other words, subsequent modules will be triggered fewer times and operate on fewer tweets, resulting in a reduced load overall.

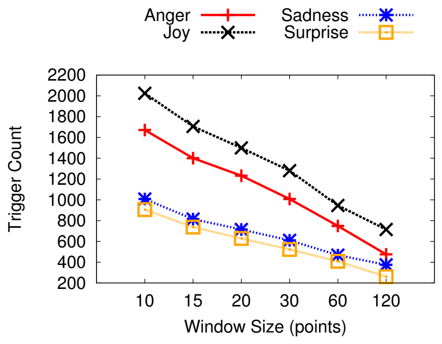
Spatio-Temporal Locality of Emotions. In this section we discuss the effect of spatio-temporal locality of emotions, and its importance in event detection. Figure 4.48 shows the number of times our approach raised an event as a function of the history it maintains, when aggregating emotions over the past 1 minute and monitoring the entire stream at once (we use only one sensor).

Interestingly, a bigger sample size results in more triggers. This is due to maintaining outdated information compared to the fast pace of the medium. As we increase the history length, triggering drops slowly (Figure 4.48(b)), because new points can be matched against more sampled data, and are less likely to be flagged as outliers. Recall that we monitor the entire stream here, implying that there is a continuous flow among all the regions we monitor.

On the other hand, Figure 4.48(a) leads to a very interesting observation. Using a 50% sample, there is a dramatic drop in the number of triggers, when we increase the window size from 10 to 15; from that point, until a window size of 30, triggering events increases slightly, and begins decreasing from that point on. This means that for

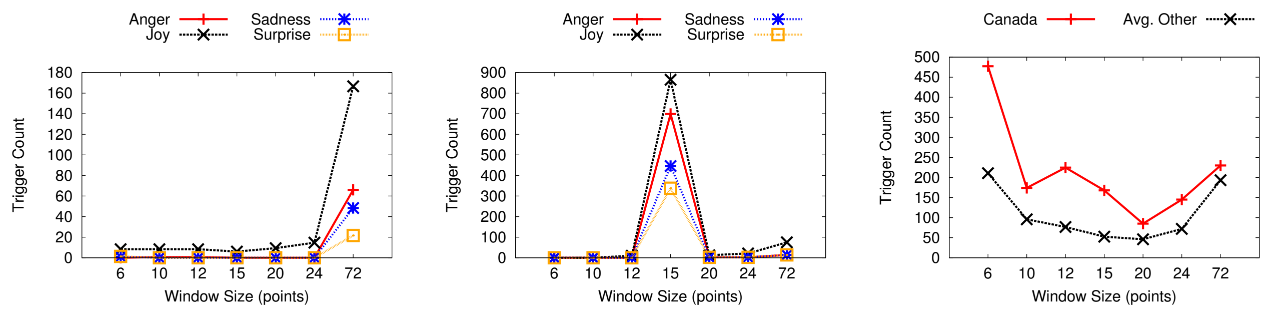


(a) Sample size = 50%



(b) Sample size = 100%

Figure 4.48: #Times a trigger was raised, compared to the window size. a = 1min, r = 0.01, p=0.1



(a) Sample Size = 50% (b) Sample Size = 100% (c) joy in Canada

Figure 4.49: #Times a trigger was raised, compared to the window size. a = 5min, r = 0.01, p=0.1

1 minute aggregations, there are rapid changes in the observed emotions; therefore a window of 10 points may be too narrow, to maintain a representative "history". On the other hand, a window between 15 -- 30 minutes seems like a better choice. This result correlates very well with the real time nature of the medium, where people tend to speak and respond very quickly to their tweets. It also means that events that are present in our data create some momentum over a mid-size period ($\sim$30minutes), and then dissipate.

Given that 1-minute aggregations may be too aggressive, we also experimented with 5-minute aggregations. As shown in Figure 4.49, 5-minutes aggregations are smoother with 50% sampling. Moreover, triggering is increased towards the end, where the window size is at least 24 points, i.e. 2hours. However, most triggers did not correspond to particularly meaningful events using simple term frequency for event description. This implies that there are not such sudden changes in 5-minute aggregations, when monitoring the entire stream. This has also been validated with longer aggregation periods.

What is more interesting, however, is the spike we observe in Figure 4.49(b), for a window size of 15 points (75 minutes), which does not exhibit a similar behavior with the one of Figure 4.48(a). The explanation is given in Figure 4.49(c), where we plot the emotion of *joy* for Canada compared to the average of the rest of the countries. The discrepancy between the two lines for w=12 and w=15 is the cause of the spike shown in Figure 4.49(b). In practice, this means that emotions are also exhibit spatial coherency. Therefore, it is better to perform localized monitoring, rather than bluntly looking into the entire stream at once.

Efficiency Performance. In the following paragraphs, we compare *TwInsight* in terms of efficiency and effectiveness against *EDCoW* [221], the current state-of-the-art for event detection in Twitter.
Table 4.15 summarizes for both systems the average time taken by each component to apply its functionality on a newly received tuple. Location extraction is a common component. Therefore, any differences lie in the classification and event detection steps.
We achieve a total 6$\times$ speed-up compared to *EDCoW*, with the event detection step being orders of magnitude faster. In other words, our approach requires considerably less time to identify events, and is expected to scale better as the number of tweets increases. The reason is that *TwInsight* operates in an online fashion, and new tuples are processed only once upon arrival. On the contrary, *EDCoW* requires between 20 and 200ms, depending on the value of Δ, which specifies the number of Stage-2 signals. This difference in efficiency is the result of several reasons: i) *EDCoW* revisits tuples to produce the Stage-2 signal values, ii) computes the wavelet coefficients in all levels, which is a function of Δ, iii) computes signals' autocorrelations -- an unavoidable cost, iv) applies the *median absolute deviation* filter twice and iii) performs the clustering step each time anew.
We would also like to note that we have run *TwInsight* on a simple laptop with 2GB RAM. On the contrary, *EDCoW* can not operate with so little memory, as it must maintain (at least) the stage-1 signals for every word it has encountered.

Effectiveness Performance. To evaluate the effectiveness of the techniques, we compare their ability to identify significant events from our dataset. Obviously, neither technique can identify events which

Table 4.15: Average Component Processing Time (ms)

Module	**TwInsight**	**EDCoW**
Location Extraction	3.36	
Classification	0.35	-
Event Detection	0.001	20 -- 200
Total	3.72	23.36 -- 203.36

are not mentioned at all. For this reason, we present significant events that each technique extracted, cross validating them with online resources.

EDCoW Results. We initially run *EDCoW* with the parameters suggested by the authors, i.e., 10 minute intervals for stage-1 signals, Δ = 6, γ = 40, over the duration of one day. As described in [221], an event *must contain at least two terms, but not too many*. Note that, for these parameters, the average description length of an event in their dataset was 2.23 terms, with at most 3 tokens.

In our dataset, *EDCoW* finds 709 events with at least 2 keywords. However, the average event description is 39.7 terms (σ = 26.6). Therefore, we set γ = 200, to increase the number of filtered tokens. Although this is 5 times higher than what the authors originally used, the average description remains high: 21 terms (σ = 23.2). In other words, most of the identified events would have been filtered out due to low significance.

Regardless, we searched for terms that we knew to have occurred during our crawl. The terms "eurovision" (the contest), "bayern" and "drogba" (for the Champions League final), did not return any results. The term "chelsea", -- the other competing team --, appears in 4 events, when using γ = 40; the term does not appear when γ = 200. Most importantly, though, the term appears on May 24 2012, 5 days *after* the match. One of those 4 cases is shown below. Clearly, most of the terms are unrelated:

bobcat, liar, desir, doc, chainz, push, selena, quedo, pasado, understood, gustan, howard, older, rare, technolog, **chelsea***, stadium, phoenix, fit, concern, psalm, thug, duda, pacer, irish, hah, hacerlo, provid, debat, swag, mum, pregunto, lama, vou, lux, strike, swallow, cuerpo, grow, goal, theori, singer, yung, lookin, 500, slapen, lea, suspens, 2016, ignor, marri*

Given our goal to identify events in a timely fashion, we also experimented with 1 minute aggregations, Δ = 5, Γ = 40, over the duration of half an hour, similarly to our parameters. This gives 588 events, with an average description length of 6 tokens, but is filled with cusses. The only exception of a real-life event is the case of "don,celtic" identified on May 27, 00:27 GMT. The event most likely refers to the Boston Celtics game, which took place on May 26, 8:00 ET (i.e., May 27, 00:00 GMT).

All in all, *EDCoW* failed to identify meaningful events that we knew about during the 2-month period of our experiment. We identify two key reasons for this:

- With the exception of our filtering step, so that we only consider specific languages and locations, we received tweets as a stream, and did not bias the data collection process. On the contrary, the data collection process of [221] could have easily biased the received tweets towards common terms, because of the Snowball harvesting technique, i.e., following social ties of a seed set to extract additional tweets. Past research has shown that connected users in online communities do not only share topical interests, but also use similar wordings [184, 177]. Moreover, the pre-processing of the dataset performed in [221] ensured an adequate volume of the tokens that would indicate events, while discarding unrelated words.
- The clustering step is prone to the data cleaning process and groups tokens together aggressively. This can be validated by the need for EDCoW's *event significance* step, to prune clusters with too many terms. However, as our experimental evaluation showed, in spite of this step, the clusters may still contain too many terms, if the original data is not carefully selected. This finding has also been verified by subsequent research [97].

TwInsight. On the contrary, *TwInsight* identified several events during this period, and Table 4.16 presents a brief summary of the most prominent, along with the associated emotion, where and when it was discovered, given our objective to identify events *as they occur*.
Event 1 concerns the speech that Erskine Bowles, a renowned figure in the US, gave to the graduates of an american university on that day. Given that posted excerpts contained terms such as "debt", and "crazy", they were flagged with the emotion of "Fear".
Event 2 is about an australian company that failed to raise a required amount of \$150m, to finish the construction of the Ararat prison. The company went under voluntary administration [28].

[28] http://www.theaustralian.com.au/business/property/st-hilliers-arafat-arm-fails-to-secure-150m-goes-into-administration/story-fn9656lz-1226357382452

Table 4.16: Sample Summary of 15 Prominent Events Identified By *TwInsight*

ID	Emotion	Where	When (GMT)	Description
1	Fear	US	13/05, 13:53	bowles crazy crisis debt national single spent year ẽrskine #2012AUGrad
2	Surprise	Canada	15/05, 23:42	$150m administration ararat couldn due extra funding hilliers
3	Sadness	Canada	16/05, 2:20	spade heat okc ship sigh spurs win @luggageboii @monalove810 calling
4	Anger	Germany	16/05, 07:52	aus frankfurt live radio @eThn0 http://t.co/ij9MNILL occupy
5	Joy	UK	19/05, 18:37	bayern win based champions chelsea excited fair fan final germans
6	Joy	UK	19/05, 19:27	15th 3rd 5th chris game games goal kreider nyr wel
7	Joy	Germany	19/05, 20:23	thomas bayern championsleague cfc mueller muller müller
8	Joy	UK	19/05, 20:29	didier drogba fucking beauty enjoying fair gal gaz goal great
9	Sadness	Canada	20/05, 22:42	died bich breaking mio robin singer @rodneyedwards gib gibb opa
10	Anger	Canada	20/05, 23:04	nose didnt live nato plz police protestors riot tryna tweet
11	Anger	Canada	20/05, 15:19	@ctvcalgary aime ambition chacun earthquake femme frais http://t.co/0hJEez9Q italy kills
12	Anger	US	20/05, 11:23	@Mou2amara alive assad onus prove regime shawkat showusshaukat syria
13	Anger	Ireland	26/05, 19:11	fahey ireland jesus keith paul squad suck tlist caled
14	Anger	Greece	26/05, 19:52	eurovision rain too much #Eurovision2012 kuulaaaaaaa
15	Joy	US	27/05, 00:58	celtics comin days game left #IRELAND #SOEXCITED @NICKIMINAJ looking

Event 3 can be traced back to various NBA games occurring on *May 15*, but are observed on *May 16* due to the time difference. Both "Spurs" and "Heat" played on May 15 ("Spurs" when the event was detected), and "OKC" had a game the following day. Note that these games appear together because we report events at the country level, rather than the city level.

Event 4 refers to the "Blockupy Frnakfurt" movement in Germany on May 16/05.

Event 5 is about a major event in our dataset, the Champions League (CL) 2012 finals, between Bayern and Chelsea, that took place on May 19. The game began at 20:45 CEST (18:45 GMT), and there are supportive tweets for the teams just as the match was about to begin.
Event 6 is related to Chris Kreider, a NY Rangers (NYR) Hockey player, who scored a goal in 5' 16" of the 3rd period. A regular hockey game has three 20' periods, with 2 intermissions of 17' each. The game began at 18:00 GMT (13:00 ET [29]), placing the goal no sooner than 19:19 GMT, not accounting for any delays, and certainly no later than 19:27, when we identify it.
Event 7 is related to the goal by Bayern's football player, Thomas Müller, in the CL final. The goal was scored in the 83rd minute of the match, i.e. on 22:23 CEST (20:23 GMT). This places our finding the event the moment that it actually occurred and was posted. We identify similar tweets in Canada and Spain, at the *exact* same timestamp. Clearly, the event is related with Joy.
Event 8 is about the equilizer, scored by Didier Drogba in the CL finals. The goal was scored in minute 88' of the game, i.e. on 22:28 CEST (20:28 GMT), and we identify several joyous tweets on 20:29, right after the goal.
Event 9 is about the death of Bee Gee's singer Robin Gibb. He was pronounced dead at 23:30 BST (22:30 GMT) on May 20th [30], and a surge in sad tweets is seen at 22:42, only 10' after his death.
Event 10 concerns the riots in Chicago, where protestors were opposing Nato's Summit [31]. Due to words "protestors", "opposing", "riot", the conveyed feeling is anger.
Event 11 is about the earthquake in Italy, on May 20, that resulted in the death of six people, among them a woman.
Event 12 refers to Assef Shawkat, deputy Minister of Defense of Syria. On May 20, 2012, there was a claim he had been murdered [32], and tweets requesting proof were posted. We have also found tweets on 26th and 27th of May regarding the Houla Massacre of the Syrian civic war which occurred on May 25. We ommit such tweets, as they

[29] http://www.nhl.com/ice/recap.htm?id=2011030313
[30] http://www.bbc.co.uk/news/entertainment-arts-18140862
[31] http://www.huffingtonpost.com/2012/05/20/nato-summit-chicago-protesters_n_1530789.html
[32] http://newsfromsyria.com/2012/05/20/asef-shawkat-assassinated/

contain URLs to pictures of immense brutality.
Event 13 is about Keith Fahey, an Irish football player, who was injured and pulled out of the national team.
Event 14 is one of many regarding the Eurovision contest, which took place on May 26. Most of them are related with Joy, however, the one we show here is related with anger. The original tweets are in greek and we show the transliteration. The day of the contest was a rainy and the phrase "too much" indicates the posters' dislike of both the contest and the fact that it was raining. Moreover, *Kuula* was the name of the Estonian song, but also happens to be a Greek name, popularized by an 80's greek comedy, to convey anguish. Moreover, the tweets we have identified follow the sequence in which the songs were performed. For instance, the tweets of this event appear after tweets containing "eurovision" and "Italy", which is expected, as Estonia performed right after Italy. Finally, the contest started at 21:00 CET [33] (19:00 GMT), and the Estonian song was performed 52 minutes within the contest [34]. Therefore, we identify the event as timely as possible.
Event 15 is about the NBA game between Boston Celtics and the 76ers. The game started at 20:00 ET, *26 May* 2012, which is 1:00 am GMT, **May 27th**. Discussions on Twitter prior to the game, especially as it was about to start, led to the event being detected. There were also some feelings of excitement concerning Nicki Minaj's upcoming performance in Ireland.
To sum up, *TwInsight* identified several events of varying types, emotions, and intensity, even though we use simpler techniques for event description. Note that we also identified different events happening on the same day, because we can operate effectively in smaller time intervals. Moreover, depending on its type, an event's transition from a latent state to one where it has gained enough visibility may vary, nevertheless, we are still able to capture such changes. Therefore, we have been able to identify considerably more meaningful events, with more efficient techniques, compared with *EDCoW*. Finally, it is worth noting that, even though we presented the locations where the events were observed, we are able to identify most of them even when

[33]http://www.eurovision.tv/page/baku-2012/about/shows
[34]http://www.youtube.com/watch?v=fjue0I4Hyko#t=51m40s

we monitor the entire stream.

4.4.7 Visualizing Results

Taking into account what type of information constitutes an event (i.e., time, location, keywords), and how we have proposed to address the event identification problem (i.e., affective theories of emotions), there are three major components that our User Interface should have:

1. Location Extraction / Geocoding of users
2. Emotion extraction from tweets
3. Event description / textual summarization

The columns of Table 4.16 clearly indicate the type of information we want to visualize. We discuss how we do that in a contextualized way, in the following paragraphs.

4.4.7.1 Visualizing Emotions

Regarding emotions, we consider the model of 6 basic emotions proposed by american psychologist Paul Ekman [77]. We also use a "Neutral" (or "None") emotion, to indicate the absence of one, leading to a total of 7 target classes. Color psychology and respective emotional theories have associated certain emotions with specific colors. We build upon these theories, and use these mappings when visualizing this type of information. These mappings are shown in Table 4.17.

Table 4.17: Mapping of emotions to colors, and examples of corresponding tweets.

Emotion	Color	Example
Neutral	White	I am Dept. of Informatics & Telecommunications (Athens, Greece)
Anger	Red	I hate it when I do something and everybody finds out! :@
Disgust	Purple	RT Retweet this if you too are offended by #HoulaMassacre #Syria
Fear	Yellow	I'm afraid this won't work out well
Joy	Green	Goaaaaaaaaaaaaaaal!!!!! Let's go @chelseafc!!! #cfc
Sadness	Blue	I miss my baby :(
Surprise	Orange	@gvalk are you serious!?

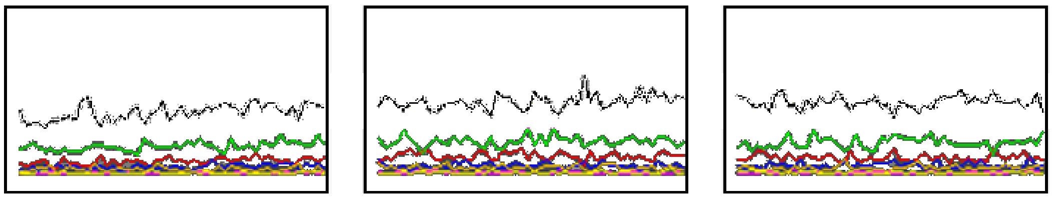
Information regarding the extracted emotions from tweets is visualized in two ways: First, we use a world map and place that information there, as described in the following section. Second, we visualize how the aggregate emotional state changes over time for each monitored location, thereby providing a spatio-temporal hedonometer.

We use two techniques to visualize the hedonometer: The first one is to use cardiograms, whereas the second one is to use histograms. In the first case, all emotions of a specific location are shown in the same area. Fig. 4.50 shows an example of this visualization, in three distinct timestamps, for the United States.This enables the end-user to understand the interplay -- or lack thereof -- of emotions experienced in that area.

For instance, neutral emotions (black line) make up most of the number of tweets that are received, with emotions of joy being second in line. An important observation is the difference between the trends of tweets conveying an emotion and the neutral ones. As a specific example, consider Fig. 4.50(b), where we can clearly see a distinctive surge in *neutral* tweets, right after the middle. However, this surge is not shared by tweets conveying an emotion, implying that we can avoid spurious bursts by using emotional theories.

The figures also validate our intuition that we should rely on emotions to detect events, rather than use simpler aggregations: If the lines were identical (even if simply translated on the y-axis), there would be no merit in using emotional signals; monitoring the entire stream at once (i.e., tweet counting) should be sufficient. This is not the case, as the lines are different from one another.

The second approach, shown in Fig. 4.53 at the bottom of the screen uses histograms, and displays each emotion separately. Once again, the emotions are updated along the temporal dimension. This visual-



(a) $t1$ (b) $t2$ (c) $t3$

Figure 4.50: Cardiograms of emotions, for three distinct timestamps in the United States

ization allows for better understanding of how each emotion is varied in a specific area over time. It also gives a clearer view of the magnitude of each emotion at a specific point in time.

4.4.7.2 Visualizing Spatial Information

Using our custom geocoding system, described in Section 4.4.3.1, we are able to map users to specific geodetic coordinates, i.e. (lat, lon). Geodetic coordinates can be visualized on a 2D World Map, through appropriate projections. We employ Mercatorian projections, which convert geodetic coordinates to cartesian, and are the most widely used. The projected world map is visualized using KML (Keyhole Markup Language) files, thereby conforming to OGC-compatible Open Standards.
The map makes up for the central part of our Graphical User Interface (GUI), as shown in Figure 4.53 [35]. The user is allowed to zoom in and out of areas, by selecting from a set of target countries, for which we currently perform emotion detection. Once a tweet has passed through emotional extraction and geocoding, it is displayed on the map in the following way: Given the location where the tweet was mapped to, we descend the hierarchy, in a random way, until we reach the lowest levels, i.e. a town or suburb in our case. If the tweet was mapped to a town / suburb in the first place, there is nothing more to do. Given the emotion of that tweet, we then color that region with the respective color of that emotion.
Currently, "coloring a town" means that we set the pixel corresponding to its location on the map to the color of the emotion. Although we only set a few pixels to that color, we expect that, in the aggregate, surges of emotions will become evident. Coloring pixels instead of broader areas has the advantage that we can update the UI easily and most importantly in real time. Newer emotions take precedence over older ones, and a town is always colored based on the most recent information. Finally, as time goes by, old town colorings are removed from the map, to accomodate for newer information, or simply returning it to the original color.

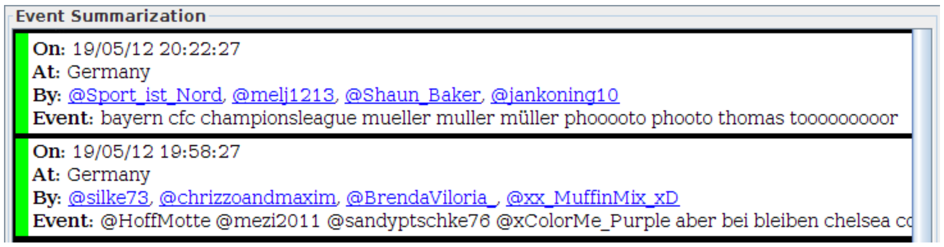
[35] The colors have been reversed, for printing efficiency

4.4.7.3 Visualizing Event Description

Once an event is identified, we need to present it to the user, so that they are notified about it, and we should provide as much information as possible. Consequently, a separate area of the UI is devoted to this purpose. As new events arrive, older ones are evicted from the list, for which we have already notified the end-user. The general information we currently show is:

- **Date**: The date and time (shown in UTC/GMT) when the event was identified. This is practically based on timestamps of incoming tweets.
- **At**: A description of the location, where the event was detected. These descriptions are based on what the user is currently monitoring. For instance, if they are monitoring at a country level, the "At" field would be "United Kingdom", even if the event was identified in Manchester.
- **By**: A list of microbloggers who talked about the event. These are practically links to the original tweets, based on which the event was identified.
- **Event**: A list of terms describing the identified event. The description is used as a fast way for the end-user to know what is going on.

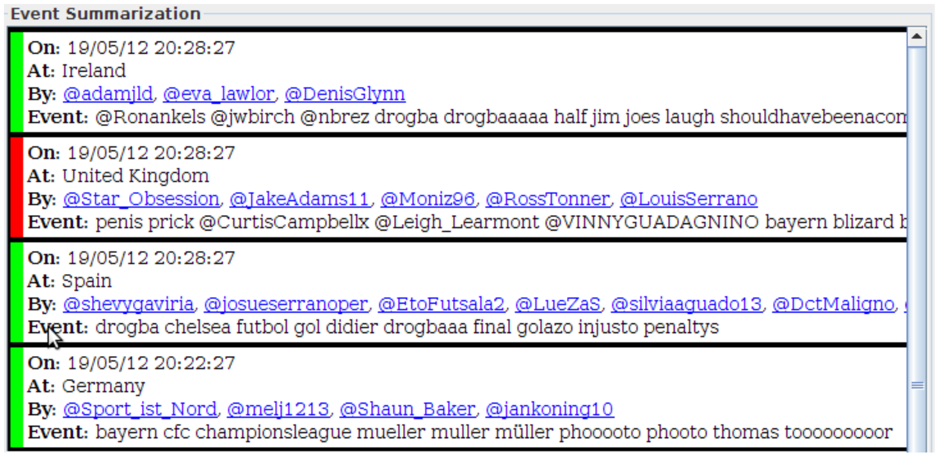
Fig. 4.51(a),(b) demonstrate a subset of the events shown to the user with respect to the Champions Leagufe final, an easily identifiable event in our dataset. The figures are used to illustrate the fact that new events are added in the list.
Fig. 4.51(a) shows a distinctive (sub)event of the Champions League finals, which is the goal scored by Bayern's football player Thomas Müller. The summary of the event clearly indicates that the event has been identified in Germany, which is only natural given that Bayern Munich is based in Germany, not to mention that the Allianz Arena stadium, where the final took place, is also in Germany.
Similarly, Fig. 4.51(b) shows the (sub)event of the goal scored by Didier Drogba, Chelsea's football player. Notice that the previous event (Bayern's goal) is pushed down the list, to make room for the newly


Event Summarization

On: 19/05/12 20:22:27
At: Germany
By: @Sport_ist_Nord, @melj1213, @Shaun_Baker, @jankoning10
Event: bayern cfc championsleague mueller muller müller phooooto phooto thomas tooooooooor

On: 19/05/12 19:58:27
At: Germany
By: @silke73, @chrizzoandmaxim, @BrendaViloria_, @xx_MuffinMix_xD
Event: @HoffMotte @mezi2011 @sandyptschke76 @xColorMe_Purple aber bei bleiben chelsea co

(a) Goal by Müller


Event Summarization

On: 19/05/12 20:28:27
At: Ireland
By: @adamjld, @eva_lawlor, @DenisGlynn
Event: @Ronankels @jwbirch @nbrez drogba drogbaaaaa half jim joes laugh shouldhavebeenacon

On: 19/05/12 20:28:27
At: United Kingdom
By: @Star_Obsession, @JakeAdams11, @Moniz96, @RossTonner, @LouisSerrano
Event: penis prick @CurtisCampbellx @Leigh_Learmont @VINNYGUADAGNINO bayern blizard b

On: 19/05/12 20:28:27
At: Spain
By: @shevygaviria, @josueserranoper, @EtoFutsala2, @LueZaS, @silviaaguado13, @DctMaligno,
Event: drogba chelsea futbol gol didier drogbaaa final golazo injusto penaltys

On: 19/05/12 20:22:27
At: Germany
By: @Sport_ist_Nord, @melj1213, @Shaun_Baker, @jankoning10
Event: bayern cfc championsleague mueller muller müller phooooto phooto thomas tooooooooor

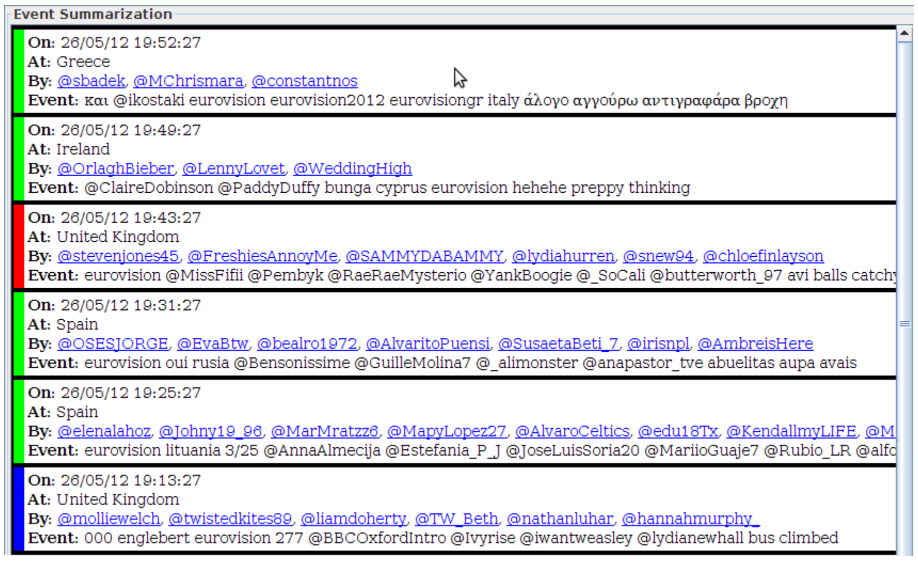
(b) Goal by Drogba

Figure 4.51: Summary of two events shown to the end-user, regarding the Champions League Final

identified event(s) describing Chelsea's goal. The same event [36] is identified in *three* locations, because each one of them is being monitored separately by the user: Spain, United Kingdom and Ireland. Were we monitoring these locations collectively, the event would have been identified only once, but the "**At**" field would be different. Also notice that the user is promptly notified about both events [37], with respect to when the goals were scored.

Finally, an additional piece of information that we present to the user is the emotion associated with the event. This is again displayed as a color, to the far left of the event description. For instance, most of the events that we identified are associated with the color "Green", signifying "Joy". This is expected, as most of the tweets are cheerful about the goals scored, by the team they are supporting. The only

[36] We know its the same event due to the descriptions.
[37] http://en.wikipedia.org/wiki/2012_UEFA_Champions_League_Final


Event Summarization

🟩 **On:** 26/05/12 19:52:27
At: Greece
By: @sbadek, @MChrismara, @constantnos
Event: και @ikostaki eurovision eurovision2012 eurovisiongr italy άλογο αγγούρω αντιγραφάρα βροχη

🟩 **On:** 26/05/12 19:49:27
At: Ireland
By: @OrlaghBieber, @LennyLovet, @WeddingHigh
Event: @ClaireDobinson @PaddyDuffy bunga cyprus eurovision hehehe preppy thinking

🟥 **On:** 26/05/12 19:43:27
At: United Kingdom
By: @stevenjones45, @FreshiesAnnoyMe, @SAMMYDABAMMY, @lydiahurren, @snew94, @chloefinlayson
Event: eurovision @MissFifii @Pembyk @RaeRaeMysterio @YankBoogie @_SoCali @butterworth_97 avi balls catchy

🟩 **On:** 26/05/12 19:31:27
At: Spain
By: @OSESJORGE, @EvaBtw, @bealro1972, @AlvaritoPuensi, @SusaetaBeti_7, @irisnpl, @AmbreisHere
Event: eurovision oui rusia @Bensonissime @GuilleMolina7 @_alimonster @anapastor_tve abuelitas aupa avais

🟩 **On:** 26/05/12 19:25:27
At: Spain
By: @elenalahoz, @Johny19_96, @MarMratzz6, @MapyLopez27, @AlvaroCeltics, @edu18Tx, @KendallmyLIFE, @M
Event: eurovision lituania 3/25 @AnnaAlmecija @Estefania_P_J @JoseLuisSoria20 @MariioGuaje7 @Rubio_LR @alfo

🟦 **On:** 26/05/12 19:13:27
At: United Kingdom
By: @molliewelch, @twistedkites89, @liamdoherty, @TW_Beth, @nathanluhar, @hannahmurphy_
Event: 000 englebert eurovision 277 @BBCOxfordIntro @Ivyrise @iwantweasley @lydianewhall bus climbed

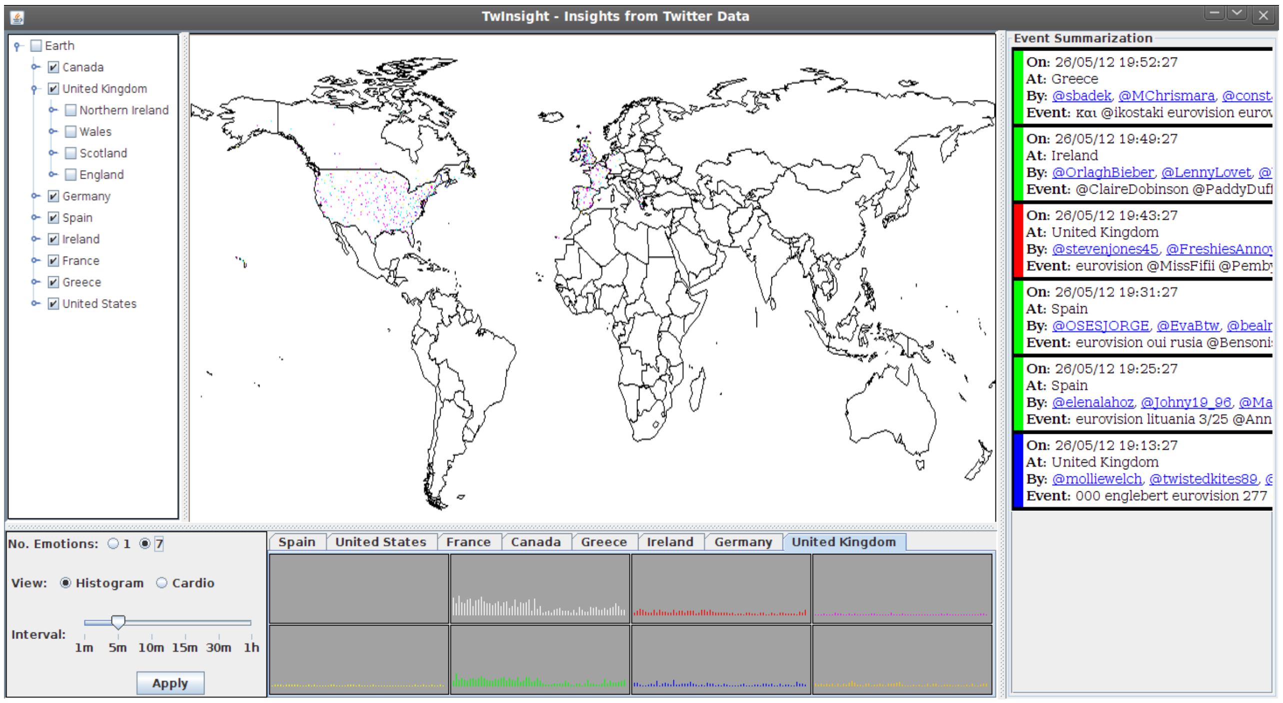
Figure 4.52: List of identified events (and their summary), related to the eurovision contest

exception is the goal scored by Drogba, that is associated with "Red", i.e., "Anger" -- clearly not what one might expect. Most surprisingly, the event is identified in the United Kingdom, Chelsea's homeplace. However, if we look closely, we will see that the term "Bayern" is in the description of the event and not "Chelsea" (or "goal", or "Drogba") , with some less than flattering words. It is useful to note, nevertheless, that these terms have come up during other runs of our approach, due to our sampling-based approach in event detection.

Fig. 4.52 shows a second list of events, with their respective summarization. The events all correspond to the Eurovision 2012 song contest final, which stirred up considerable discussions. Starting from the bottom and moving upwards the events list, we can easily verify [38] that the identified events describe the sequence in which the participating teams competed in the contest.

For instance, Engelbert opened the contest, which started at 19:00 GMT [39]. The singer was representing the United Kingdom, singing a ballad, thereby creating some moody feelings, as exemplified by the color "blue" (sadness) next to the event description. About 12 minutes later, we see an event containing the term "lituania", which was com-

[38]http://www.eurovision.tv/page/baku-2012/about/shows/final
[39]http://www.eurovision.tv/page/baku-2012/about/shows



- ☐ Earth
  - ☑ Canada
  - ☑ United Kingdom
    - ☐ Northern Ireland
    - ☐ Wales
    - ☐ Scotland
    - ☐ England
  - ☑ Germany
  - ☑ Spain
  - ☑ Ireland
  - ☑ France
  - ☑ Greece
  - ☑ United States

No. Emotions: 1 [✓ 7]

View: [✓ Histogram] Cardio

Interval: 1m [✓ 5m] 10m 15m 30m 1h

Event Summarization

On: 26/05/12 19:52:27
At: Greece
By: @sbadek, @MChrismara, @const
Event: και @ikostaki eurovision eurov

On: 26/05/12 19:49:27
At: Ireland
By: @OrlaghBieber, @LennyLovet, @
Event: @ClaireDobinson @PaddyDufl

On: 26/05/12 19:43:27
At: United Kingdom
By: @stevenjones45, @FreshiesAnnoy
Event: eurovision @MissFifii @Pemby

On: 26/05/12 19:31:27
At: Spain
By: @OSESJORGE, @EvaBtw, @bealr
Event: eurovision oui rusia @Bensoni

On: 26/05/12 19:25:27
At: Spain
By: @elenalahoz, @Johny19_96, @Ma
Event: eurovision lituania 3/25 @Ann

On: 26/05/12 19:13:27
At: United Kingdom
By: @molliewelch, @twistedkites89, @
Event: 000 englebert eurovision 277

Figure 4.53: The overall GUI that the user sees.

peting 4th in line. Hungary and Albania do not show up in this run, although Hungary received a very low ranking overall, and we did not see that many discussions concerning its participation. Lithuania was next, and with a maximum of 3 minutes per song, the user sees the event -- as it is identified according to the discussions -- right when it occurred. as shown in Fig. 4.52, we also identified discussions regarding Russia (which is written with a single "s" in Spanish), Cyprus and Italy.

4.4.7.4 Putting It All Together

In addition to the separate components that make up our system, the user is able to control the event detection process through a set of available options. These options are available at all times, and can be altered while the system is running. Overall, our UI provides the following options that affect our system's functionality:

- Number of emotions against which tweets are classified.

- Monitored locations, including parents and children nodes in the hierarchy.
- Size of aggregation interval, i.e., number of minutes between two consecutive runs of our event detection mechanism.

Fig. 4.53 shows the complete version of our UI, as this is shown to the end-user. Observe the histogram type of monitoring the emotions at the lower section of the screen, that we have already discussed. Also note that emotions are shown on the world map, coloring specific pixels appropriately. The coloring result is more prominent in France, Ireland, Spain, the UK and the US. Regarding the United States, note that very few tweets are mapped in the state of Alaska. The reason is that, despite the arbitrary assignment to town locations, we descend the hierarchy from the initial geocoded location. Therefore, if a tweet was mapped to New York City, it will be mapped to a suburb of Manhattan or Brooklyn, but never to a city in Alaska. As the population in Alaska is far lower than other states, this is an indirect validation of our geocoding service.
The options available to the user can be seen in the left hand side of the UI. In the upper part of the options area, we display the areas that the user is allowed to select from. These are shown in a tree structure that reflects the hierarchy we currently rely on to identify events and display information. Note that event monitoring is currently performed at the country level and that it is also possible to select parent nodes separately from children node (e.g., United Kingdom is selected, but none of its children are). Therefore, only the countries appear in the bottom area of the UI, which shows the aggregate emotional state per region.
In the lower part of that area, we can clearly see options that affect the emotions that we monitor. Currently, the user is able to select between monitoring all 7 emotions, or just 1. The latter case is identical to monitoring the rate at which tweets arrive, regardless of any emotion that they may convey. We also allow various aggregation intervals: 1 minute, 5 minutes, 10 minutes, etc. By reducing the aggregation interval, users will be notified about events more timely, but more events will be generated. By contrast, increasing the aggregation interval will generate fewer events, but the user will be notified about them less promptly. They can also switch to monitoring a sin-

gle "emotion", which is practically equivalent to monitoring the rate at which tuples are received. In the same part of the UI, we also clearly see the option to switch between histogram and cardiogram view of emotions.
An option not shown in this UI is that the user may select between real-time identification of events, by monitoring the Twitter stream, or replaying stored streams. The latter functionality can be used to improve all aspects of our sub-system, including our UI, as well as give access and insights to historical data. This is simply done by passing additional arguments when the system starts.

4.5 Summary

Social media are a highly prolific area for research, especially due to their wide adoption by the users. A distinctive characteristic of social media platforms, compared to prior online services, is their social component, where users may connect with each other, forming directed or undirected social graphs. This component lays the ground for new research questions in various disciplines, including social sciences, economics and, of course, computer science.
In this chapter, we considered various research topics that arise in such a novel area of research, ranging from efficient ways to harvest information, to techniques that automatically identify meaningful information (i.e., events), that could prove useful for a number of applications, such as computational journalism, crisis management or resource allocation to name a few.
In particular, we presented the architecture and implementation details of a crawler for the Twitter service, a popular social media platform in the contemporary webosphere. The service offers two distinct Application Programming Interfaces to provide access to its data: a Streaming interface, whereby user generated content is received as a data stream, and a Probe-based one, where users query the service for specific information. Information may be collected according to several characteristics, such as the social component, spatial information, topical information (keywords), and so on. Our crawler efficiently harvests information based on these characteristics.

We also presented how data analysis can be incorporated in declarative languages, so that it may be optimized and executed efficiently in domain-specific query execution engines. The main advantage of this approach is that the underlying optimizer knows the particularities of the domain, and ensures the efficient execution of the query. We proposed a technique whereby data analysis techniques are formalized as intensional extents, can be expressed at the language level as well, and are subsequently refactored to be optimized by the execution engine. Although our methodology is independent of a domain of application, we demonstrated its performance in a sensor network domain, which is a resource-constrained setting and the temporal dimension plays a vital role.

Finally, we used social media data to identify newsworthy events in real-time, given that past research had shown that the discussed topics reflect what is happening in the physical world. We started with an initial analysis of the statistical properties of the received content provided in a streaming fashion, because multiple sample percentages may be used. The purpose of this analysis was to understand the differences between the sampling policies and the merits of using one over the other. Based on this analysis, we used the appropriate stream, and went on to present our event detection mechanism for newsworthy information, using social media data. Our event detection mechanism relies on and incorporates cognitive and affective theories of emotions, allowing our solution to be viewed as an analytical tool for the social sciences as well: not only an approach to identify events, but also a way of understanding how different events impact different people. In order to be able to identify events, we presented a geocoding service, which was used to assign spatial information to the users and later groupd them into large, geographical groups, which are monitored independently. Our approach for detecting events also revealed certain properties regarding the discussions of events, namely that events generate a medium-term momentum (i.e., a lot of people start talking about it for some time), but then dissipate. Our event detection mechanism operates in an online fashion, making it suitable for a fast paced medium like Twitter, and social media in general.

Chapter 5

Conclusions and future directions

5.1 Summary of the thesis

The landscape of the World Wide Web is changing, following the technological advancements that occur at the hardware and the software level. Online services and applications are more pervasive nowadays, especially as smartphones and other devices (e.g., wearables) allow users to monitor and share their daily activities, and connectivity to the internet is practically constant with wireless technologies. Users also interact with each other directly and in real time, through social networking platforms, openly expressing their opinions over politics, sports, news, and general interests.

All these interactions generate a tremendous amount of information. Managing and mining such information is a key step to providing the user with engaging services that fulfill their needs. The information may be used by an existing service either to improve its quality or to add new functionality, or it may serve as the data to empower new services.

In this thesis, we considered three specific cases of managing and mining such information, while addressing both objectives. In short, the contributions of this thesis can be summarized as follows:

1. A technique to reduce the output size of skyline queries, that captures diversification of trade-off points at its core, and an alternative scheme for ranking them.
2. A formal framework to quantify the competitiveness of items, and efficient algorithms to return the top-k competitors.

3. A complete framework to efficiently gather, process and mine information from a popular social networking platform, Twitter.

4. A comparative analysis of the sampling policies of the Twitter social networking platform.

5. A technique to detect events from social media data in real-time, using custom geocoding and relying on cognitive and affective theories of emotions, which also provides a dual perspective, informing us of the way that users react to events.

In all of these cases, we experimentally evaluated and validated our proposed techniques, comparing them with existing equivalents or meaningful baselines.
Regarding our first objective, we provided a formal framework to diversify skyline points in an intuitive way and return k skyline points with maximum diversity. Our proposed approach builds upon established measures of importance for skyline points and only requires that the dominance property be defined. Consequently, it is applicable in any setting where skyline queries may be applied. We model the problem as a Max-Min instance of a k-Dispersion Problem [], where k controls the output size of a skyline query. To efficiently compute the k skyline points with maximum diversity, we presented a two-phase framework, named SkyDiver, that first transforms the original space into a more compact, yet approximate one, and secondly selects the k points, operating in the transformed space. For the transformation, we use MinHash signatures [46] which fits nicely with our definition of diversification. In particular, we compute the diversity of two skyline points as the Jaccard distance of their domination sets, i.e., the sets of points that they dominate. For a more compact representation of the original space, we may also employ Locality Sensitive Hashing [113]. Experimental evaluations demonstrated that the SkyDiver framework is much faster than a naive implementation of the solution, while being very effective at the same time. Comparison against other techniques that control the output of skyline queries [200, 136] showed that i) diversification, as an objective, is different from coverage, and that ii) distance-based representation of the skyline correlates more with the coverage objective.

For our second objective, we presented a formal framework that quantifies competitiveness between items defined in the same feature space. Our proposed framework is generic enough to accommodate all feature types. Our formalism of competitiveness maps naturally the notion that "items that address the same target group are more competitive than items with different target groups". A target group can be identified as a group of people interested in a specific feature subset of the original feature space, and by considering all possible feature subsets, we take into account all possible target groups of an item. Then, for each target group, we quantify the degree to which two items may fulfill the same needs. Our formalism also allows for different groups to contribute with different weights. Given our formalism, we also presented efficient techniques to identify the top-k competitors of an item of interest. We then experimentally evaluated our techniques, both in terms of efficiency and effectiveness. Our experiments revealed that the presented technique is highly efficient, especially compared against a straightforward implementation. Unlike a nearest neighbor baseline, our formalism is also very effective in identifying competitive items, as demonstrated by the user study we conducted.

For our third objective, we implemented a crawler for the Twitter service, which is a prevalent social networking platform, that has received much attention by the industry and academia alike. Our proposed architecture ressembles that of a crawler designed for the surface web [49], with certain changes to account for the differences of the Twitter service, namely i) a streaming component to receive data, ii) a different mechanism to enqueue queries, given the custom rate limits of the service. We also presented an approach that harvests information from web sites that provide a query interface. Such techniques have been popularized for the Hidden Web [40], and we augment existing models [159] to account for the ranking of the returned results. Our experiments demonstrated that by accounting for this aspect, we are able to retrieve more content (i.e., cover the content better) with the same number of queries, when compared with the default scheme. The final part of this contribution refers to building an intermediate layer for query execution engines, so that so that data mining algorithms may be expressed in structured languages (e.g., SQL, CQL, etc), and executed efficiently by the underlying execution engine. We demonstrated how to achieve this with an existing execution engine destined for sensor networks [83]. The selection of a sensor network query execution engine is because queries in this setting also consider the temporal aspect, which is also an integral part of social media data. Our approach is to model data mining algorithms as extensional extents, allowing us to include them in declarative queries like any other extent (e.g., relation, view, etc.) [208]. For the same reason, query execution engines are able to optimize these constructs, as long as they can be expressed with the declarative language.

Regarding our fourth objective, we were interested in comparing the sampling policies that Twitter uses to provide data in a streaming fashion. In particular, Twitter provides either a 1% or a 10% sample of all public posts, and we conducted a set of experiments to identify key differences in the two policies, with respect to the data acquired. Our analysis covered a broad spectrum, including spatial and temporal information, sentiment extraction, popular topic detection, the retweet graph and linguistic information. Our analysis revealed that the 1% sample is good enough when someone is only interested in very popular topics, whereas higher sampling ratios are needed in order to uncover less popular topics. It also showed that by monitoring the retweet graph over extended periods of time, we may achieve similar results, in terms of graph characteristics, as when having access to the entire stream (100%). The received information may also change by slightly adjusting the bounding box of the monitored location. Finally, our linguistic analysis showed that the languages in Twitter, ranked by volume of written text, are very different -- in order and in percentages -- from ground truth data collected through world-wide surveys.

For our final goal, we presented an approach that is able to identify events in real time using social media data. Unlike previous techniques, our approach is not tied to events of a specific type, thereby making it generic. At the core of our model lies an outlier detection mechanism which monitors the aggregate emotional state of users, grouped by their spatial proximity. To extract the location of users, we built a custom geocoding mechanism that relies on open-source software and data available on the web. For the emotional state of users, we employ a classifier with 7 target classes, augmenting the 6 emotions proposed by psychologist P. Ekman [77] with a "None" class. Twitter posts are then classified to one of the 7 classes, allowing us to collectively monitor the emotional state of the group. Comparing against the state of the art, which uses online clustering [221], our approach is more efficient and is able to identify a lot more events. On the contrary, [221] is unable to identify meaningful events, because the technique requires careful data collection to work properly, which is impossible for practical scenarios. Our proposed technique also serves a dual purpose, as it can be used to observe how users react to certain event types, which is not possible with previous techniques designed for event detection.

5.2 Future Directions

There are various research directions to follow within the scope of mining and managing user-generated content and their preferences, and push forward the research that has been presented in this thesis. We discuss some of these possibilities in the next few paragraphs.

5.2.1 Skyline diversification and novel query types

Interesting variations of the skyline diversification problem include settings where the dominance property is not clearly defined, as is the case with the skyline over groups of points [148]. The dominance property, which is the most integral part of skyline computation, does not extend to groups in a straightforward manner. Based on past literature, we could define diversification using the γ-dominance between groups of points. We note that our definition of skyline diversification is easier to extend to such a setting, compared with diversification techniques that rely on L_p norms. Another direction regarding the efficiency perspective of the problem would be to parallelize computations, to scale our proposed technique for massive datasets.
Efficiency was also one aspect to improve when ranking skyline points with our IR ranking scheme. A reason for that is that we only considered the upper bound of a skyline point's score. Therefore, by considering the lower bound as well, we may be able to improve the efficiency of the presented technique. An alternative approach is to use approximate techniques, with quality guarantees, that result in the same final

ranking as the one obtained through exact computations. There are two options to achieve this: i) by sampling the original space, so that we perform the computations using fewer points, and ii) by using the first top-l layers of minima, as earlier layers contribute with a higher weight. Moreover, earlier layers contain a significantly larger number of points, therefore one would expect that the final ranking would not be affected too much by the layers of minima towards the end.

5.2.2 Competitor identification and review mining

The notion of competitiveness builds on the premise that items target the same group of people and fulfill their needs. To identify the degree to which these needs are fulfilled, we used review mining techniques, and assigned a score to each discussed feature (e.g., quality of product, usability, etc). One aspect that our current methodology did not take into account was that users expressed their opinions of the product based on some specific usage type. In other words, the review was biased by the user's expectations of the item. As an example, a normal computer processor may be adequate for office work or browsing the internet, but will not be sufficient for experiments or gaming. If a user bought such a processor for a gaming machine, they would definitely be unhappy, and this would be reflected in their review. By carefully mining the reviews to identify the cases where an item is best suited, we may not only improve competitor identification, but also recommendations for other products. Such a goal may also change the searching paradigm for products to specifying the intended use.

5.2.3 Mining user-generated content and social media analysis

Despite the attention that the field has attracted, social media remains an unexplored area to a large extent. Regarding event detection, better techniques to extract the location of users are required. The reason is that spatial information may be part of the post, which we do not currently consider. Another issue is that certain events may not necessarily be tied to a location. A way to address this is to map certain entities or keywords of a post with a specific location, combining information from external resources, such as Wikipedia or FreeBase.

Another limitation of our existing approach is that the same event may be discovered in multiple locations, eliciting different emotions. For this reason, we need an aggregation phase that groups together the separate descriptions into a single one. This aggregation phase may take place after the event description of each separately identified event, grouping together events that share common terms. The opposite may also occur, where two different events may have occurred at the same time, in the same place. In such a case, we need to split the output event into two (or more). This may be achieved by finding whether the descriptive keywords co-occur in any of the posts. If they do not, we may consider them as separate events.

The above requirements have two additional byproducts. First, more elaborate techniques should be preferred in this case, to extract more descriptive keywords of the event. Second, we should be able to filter out false positives, i.e., cases that our technique identified as an event, but do not really correspond to one. Ideally, this should be done in real time, and without the use of external resources for cross-checking. One way to achieve this would be a preliminary analysis that would provide a prior probability of a term appearing in an event. However, one should keep in mind the special characteristics of the Twitter platform (personal writing style and slang, short text) when applying such prior probabilities.

As far as event detection is concerned, alternative user groupings could be considered. For example, instead of grouping users by their location, we could apply a clustering algorithm to group users together according to their topical proximity. By topical proximity we mean the users' similarity of what they post. Of course, a user may be part of several communities, in which case a fuzzy clustering algorithm may be more appropriate, so that each user may belong to -- and therefore update -- more than one groups.

Moreover, there are several other research directions to consider regarding social media. The social network is a major asset of these platforms and uncovering the underlying mechanism with which links are formed is extremely interesting. Knowing this mechanism allows us to better study and understand user behavior as well as graph structure, its properties and evolution. The problem becomes particularly challenging given that the collection of the graph is a very slow process, and techniques that could infer the graph would be preferred.

Finally, event and timeline summarization is another direction to consider. In this case, given a set of time-ordered posts, we are interested in a (much smaller) subset that captures the major (sub)events or adequately describes the original set. For example, in a football match, cards, goals and fouls are the subevents of interest. In a political campaign, the turning points that affected the outcome should be the output of such a technique. We note that the selected subset changes at different temporal granularities, even for the same event. With the diversity, volume and quality of the user-generated content in social media, the need for techniques that effectively summarize information and capture the major milestones, can only be expected to increase.

Chapter 6

References

[1] http://maps.yahoo.com/.

[2] http://maps.google.com/.

[3] http://www.bing.com/maps/.

[4] https://dev.twitter.com/docs/api/1.1.

[5] https://dev.twitter.com/docs/error-codes-responses.

[6] https://dev.twitter.com/docs/twitter-libraries.

[7] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277--289, January 2005.

[8] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120--139, August 2003.

[9] F. Abel, I. Celik, G.-J. Houben, and P. Siehndel. Leveraging the semantics of tweets for adaptive faceted search on twitter. In *ISCW*, pages 1--17, 2011.

[10] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[11] C. Aggarwal, editor. *Data Streams -- Models and Algorithms*. Springer, 2007.

[12] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5--14, 2009.

[13] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439--450, 2000.

[14] A. Ahmed, L. Hong, and A. J. Smola. Hierarchical geographical modeling of user locations from social media posts. In *WWW*, 2013.

[15] F. Alvanaki, S. Michel, K. Ramamritham, and G. Weikum. See what's enblogue: real-time emergent topic identification in social media. In *EDBT*, pages 336--347, 2012.

[16] P. Andreou, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras. Workload-aware query routing trees in wireless sensor networks. In *MDM*, pages 189--196, 2008.

[17] G. L. Andrienko, N. V. Andrienko, H. Bosch, T. Ertl, G. Fuchs, P. Jankowski, and D. Thom. Thematic patterns in georeferenced tweets through space-time visual analytics. *Computing in Science and Engineering*, 15(3):72--82, 2013.

[18] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD*, pages 781--792, 2011.

[19] L. Anselin. What is special about spatial data? alternative perspectives on spatial data analysis. In *Symposium on Spatial Statistics, Past, Present and Future*, pages 63--77, 1989.

[20] B. Arai, G. Das, D. Gunopulos, V. Hristidis, and N. Koudas. An access cost-aware approach for object retrieval over multiple sources. *PVLDB*, 3(1):1125--1136, 2010.

[21] E. Aramaki, S. Maskawa, and M. Morita. Twitter catches the flu: Detecting influenza epidemics using twitter. In *EMNLP*, pages 1568--1576, 2011.

[22] E. Aramaki, S. Maskawa, and M. Morita. Information extraction from social media for public health. In *SIGKDD The Data Frameworks Track*, 2014.

[23] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: the stanford stream data manager (demonstration description). In *SIGMOD*, 2003.

[24] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121--142, 2006.

[25] G. O. Arocena and A. O. Mendelzon. Weboql: Restructuring documents, databases, and webs. In *ICDE*, pages 24--33, 1998.

[26] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633--634, 2002.

[27] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS*, pages 234--243, 2003.

[28] R. Bakshi, C. A. Knoblock, and S. Thakkar. Exploiting online sources to accurately geocode addresses. In *ACM-GIS Workshop*, pages 194--203, 2004.

[29] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an influencer: quantifying influence on twitter. In *WSDM*, pages 65--74, 2011.

[30] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256--273, 2004.

[31] S. Bao, R. Li, Y. Yu, and Y. Cao. Competitor mining with the web. *IEEE Transactions on Knowledge Data Engineering*, pages 1297--1310, 2008.

[32] N. Barbieri, F. Bonchi, and G. Manco. Influence-based network-oblivious community detection. In *ICDM*, 2013.

[33] L. Barbosa and J. Feng. Robust sentiment detection on twitter from biased and noisy data. In *COLING '10: Posters*, pages 36--44, 2010.

[34] H. Becker, F. Chen, D. Iter, M. Naaman, and L. Gravano. Automatic identification and presentation of twitter content for planned events. In *ICWSM*, 2011.

[35] H. Becker, D. Iter, M. Naaman, and L. Gravano. Identifying content for planned events across social media sites. In *WSDM*, pages 533--542, 2012.

[36] H. Becker, M. Naaman, and L. Gravano. Learning similarity metrics for event identification in social media. In *WSDM*, pages 291--300, 2010.

[37] E. Benson, A. Haghighi, and R. Barzilay. Event discovery in social media feeds. In *ACL-HLT*, 2011.

[38] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *Journal of the ACM*, 25(4):536--543, 1978.

[39] M. Bergen and M. A. Peteraf. Competitor identification and competitor analysis: a broad-based managerial approach. *Managerial and Decision Economics*, 2002.

[40] M. Bergman. The deep web: Surfacing hidden value. Technical report, 2001. http://www.brightplanet.com/images/uploads/DeepWebWhitePaper_20091015.pdf.

[41] J. Bollen, H. Mao, and A. Pepe. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. In *ICWSM*, 2011.

[42] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421--430, 2001.

[43] B. Boyce. Beyond topicality: A two stage view of relevance and the retrieval process. *Information Processing & Management*, 18(3):105--109, 1982.

[44] C. Y. Brenninkmeijer, I. Galpin, A. A. Fernandes, and N. W. Paton. A semantics for a query language over sensors, streams and relations. In *BNCOD*, pages 87--99, 2008.

[45] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107--117, 1998.

[46] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630--659, 2000.

[47] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension: (preliminary version). In *SCG*, pages 293--302, 1994.

[48] A. L. Buchsbaum and M. T. Goodrich. Three-dimensional layers of maxima. *Algorithmica*, 39:275--286, July 2004.

[49] C. Castillo. Effective web crawling. *SIGIR Forum*, 39(1):55--56, June 2005.

[50] D. Chakrabarti and K. Punera. Event summarization using tweets. In *ICWSM*, 2011.

[51] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478--495, 2006.

[52] G. Chatzimilioudis, A. Cuzzocrea, and D. Gunopulos. Optimizing query routing trees in wireless sensor networks. In *ICTAI (2)*, pages 315--322, 2010.

[53] S. Chaudhuri. Data mining and database systems: Where is the intersection? *Data Engineering Bulletin*, 21, 1998.

[54] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313--324, 2003.

[55] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Transactions on Database Systems*, 1999.

[56] M.-J. Chen. Competitor analysis and interfirm rivalry: Toward a theoretical integration. *Academy of Management Review*, 1996.

[57] M.-J. Chen and I. C. MacMillan. Nonresponse and delayed response to competitive moves: The roles of competitor dependence and action irreversibility. *The Academy of Management Journal*, 35(3):539--570, August 1992.

[58] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4):390--426, dec 2003.

[59] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717--719, 2002.

[60] T. Churches, P. Christen, K. Lim, and J. Zhu. Preparation of name and address data for record linkage using hidden markov models. *BMC Medical Informatics and Decision Making*, 2(1), 2002.

[61] B. H. Clark and D. B. Montgomery. Managerial identification of competitors. *Journal of Marketing*, 1999.

[62] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659--666, 2008.

[63] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64--78, 2001.

[64] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, A. Flammini, and F. Menczer. Political polarization on twitter. In *ICWSM*, 2011.

[65] D. J. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. Mapping the world's photos. In *WWW*, pages 761--770, 2009.

[66] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel distributed processing of constrained skyline queries by filtering. In *ICDE*, 2008.

[67] A. Culotta. Towards detecting influenza epidemics by analyzing twitter messages. In *SOMA*, pages 115--122, 2010.

[68] A. Das Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative skylines using threshold-based preference distributions. In *ICDE*, pages 387--398, 2011.

[69] A. Das Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized multi-pass streaming skyline algorithms. *PVLDB*, 2(1):85--96, 2009.

[70] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD*, pages 629--640, 2007.

[71] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *ESA*, pages 323--334, 2002.

[72] R. Deshpandé and H. Gatingon. Competitive analysis. *Marketing Letters*, 1994.

[73] X. Ding, B. Liu, and P. S. Yu. A holistic lexicon-based approach to opinion mining. In *WSDM*, 2008.

[74] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, pages 216--227, 2012.

[75] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613--622, 2001.

[76] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing. A latent variable model for geographic lexical variation. In *EMNLP*, pages 1277--1287, 2010.

[77] P. Ekman, W. Friesen, and P. Ellsworth. *Emotion in the human face: guide-lines for research and an integration of findings*. Pergamon Press, 1972.

[78] E. Erkut, Y. Ülküsal, and O. Yenicerioğlu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103--1113, 1994.

[79] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628--648, 2006.

[80] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA*, pages 28--36, 2003.

[81] L. Ferrari, A. Rosi, M. Mamei, and F. Zambonelli. Extracting urban patterns from location-based social networks. In *LBSN Workshop*, pages 9--16, 2011.

[82] W. T. Few. Managerial competitor identification: Integrating the categorization, economic and organizational identity perspectives. *Doctoral Dissertaion*, 2007.

[83] I. Galpin, C. Y. A. Brenninkmeijer, A. J. G. Gray, F. Jabeen, A. A. A. Fernandes, and N. W. Paton. Snee: a query processor for wireless sensor networks. *Distributed and Parallel Databases*, 29(1--2):31--85, 2011.

[84] Y. Gao, J. Hu, G. Chen, and C. Chen. Finding the most desirable skyline objects. In *DASFAA*, pages 116--122, 2010.

[85] H. Garcia-Molina. Challenges in crawling the web. In *BNCOD*, page 3, 2003.

[86] H. Gatignon, E. Anderson, and K. Helsen. Competitive reactions to market entry: Explaining interfirm differences. *Journal of Marketing Research*, 1989.

[87] D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI*, pages 1--11, 2003.

[88] J. B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175--181, 1996.

[89] S. Ghosh, G. Korlam, and N. Ganguly. Spammers' networks within online social networks: a case-study on twitter. In *WWW*, pages 41--42, 2011.

[90] S. Ghosh, M. B. Zafar, P. Bhattacharya, N. Sharma, N. Ganguly, and K. Gummadi. On sampling the wisdom of crowds: Random vs expert sampling of the twitter stream. In *CIKM*, 2013.

[91] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, pages 2498--2506, 2010.

[92] A. Gkoulalas-Divanis, P. Kalnis, and V. S. Verykios. Providing k-anonymity in location based services. *SIGKDD Explorations*, 12(1):3--10, Nov. 2010.

[93] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1--6, 2009.

[94] D. Goldberg, J. Wilson, and C. Knoblock. From text to geographic coordinates: the current state of geocoding. *URISA Journal*, 19(1):33--47, 2007.

[95] P. Golle and K. Partridge. On the anonymity of home/work location pairs. In *Pervasive*, pages 390--397, 2009.

[96] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *CCS*, 2010.

[97] A. Guille and C. Favre. Mention-anomaly-based event detection and tracking in twitter. In *ASONAM*, 2014.

[98] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD*, pages 463--474, 2000.

[99] M. Gupta, P. Zhao, and J. Han. Evaluating event credibility on twitter. In *SDM*, pages 153--164, 2012.

[100] A. Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Record*, 14(2), June 1984.

[101] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10--18, nov 2009.

[102] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. Dmql: A data mining query language for relational databases. In *SIGMOD workshop on Research issues on Data Mining and knowledge discovery*, pages 27--33, 1996.

[103] J. R. Haritsa. The kndn problem: A quest for unity in diversity. *IEEE Data Engineering Bulletin*, 32(4):15--22, 2009.

[104] C. Hawn. Take two aspirin and tweet me in the morning: How twitter, facebook, and other social media are reshaping health care. *Health Affairs*, 28(2):361--368, 2009.

[105] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50(5):94--101, 2007.

[106] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219--229, Apr. 1999.

[107] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *ASPLOS*, pages 93--104, 2000.

[108] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *WWW*, 2012.

[109] M. Hu and B. Liu. Mining and summarizing customer reviews. In *SIGKDD*, pages 168--177, 2004.

[110] Y. Hu, F. Wang, and S. Kambhampati. Listening to the crowd: Automated analysis of events via aggregated twitter sentiment. In *IJCAI*, 2013.

[111] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in manets. In *ICDE*, 2006.

[112] T. Imieliński and A. Virmani. Msql: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373--408, 1999.

[113] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604--613, 1998.

[114] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems*, 32(4):21, 2007.

[115] P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: categorizing hidden web databases. In *SIGMOD*, pages 67--78, 2001.

[116] K. Y. Kamath and J. Caverlee. Content-based crowd retrieval on the real-time web. In *CIKM*, pages 195--204, 2012.

[117] N. Katariya, A. Iyer, and S. Sarawagi. Active evaluation of classifiers on large datasets. In *ICDM*, pages 329--338, 2012.

[118] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K.-U. Sattler. Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in anduin. *Distributed and Parallel Databases*, 29(1--2):151--183, 2011.

[119] J. Kleinberg. Bursty and hierarchical structure in streams. In *SIGKDD*, pages 91--101, 2002.

[120] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422--469, 2000.

[121] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. *PVDLB*, pages 275--286, 2002.

[122] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: An architectural status report. *IEEE Data Engineering Bulletin*, 26(1):11--18, March 2003.

[123] C.-C. Kuo, F. Glover, and K. S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171--1185, 1993.

[124] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591--600, 2010.

[125] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Transactions on Information Systems*, 19(3):242--262, 2001.

[126] T. Lansdall-Welfare, V. Lampos, and N. Cristianini. Effects of the recession on public mood in the uk. In *WWW Companion*, 2012.

[127] T. Lappas, G. Valkanas, and D. Gunopulos. Efficient and domain-invariant competitor mining. In *SIGKDD*, pages 408--416, 2012.

[128] T. Lappas, M. R. Vieira, D. Gunopulos, and V. J. Tsotras. On the spatiotemporal burstiness of terms. *PVLDB*, 5(9):836--847, 2012.

[129] G. Leshed and J. J. Kaye. Understanding how bloggers feel: recognizing affect in blog posts. In *CHI*, 2006.

[130] M. Lewis, J. Haviland-Jones, and L. Barrett. *Handbook of Emotions, Third Edition*. Guilford Publications, 2010.

[131] C. Li, A. Sun, and A. Datta. Twevent: segment-based event detection from tweets. In *CIKM*, 2012.

[132] H. Li, Q. Tan, and W.-C. Lee. Efficient progressive processing of skyline queries in peer-to-peer systems. In *Infoscale*, 2006.

[133] R. Li, S. Bao, J. Wang, Y. Liu, and Y. Yu. Web scale competitor discovery using mutual information. In *ADMA*, 2006.

[134] R. Li, S. Bao, J. Wang, Y. Yu, and Y. Cao. Cominer: An effective algorithm for mining competitors from the web. In *ICDM*, pages 948--952, 2006.

[135] R. Li, K. H. Lei, R. Khadiwala, and K.-C. Chang. Tedas: A twitter-based event detection and analysis system. In *ICDE*. IEEE, 2012.

[136] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86--95, 2007.

[137] J. Liu, Z. Wu, L. Jiang, Q. Zheng, and X. Liu. Crawling deep web content through query forms. In *WEBIST*, pages 634--642, 2009.

[138] K.-L. Liu, W.-J. Li, and M. Guo. Emoticon smoothed language models for twitter sentiment analysis. In *AAAI*, 2012.

[139] Y. Liu, C. Kliman-Silver, and A. Mislove. The tweets they are a-changin': Evolution of twitter users and behavior. In *ICWSM*, 2014.

[140] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1):313--324, 2009.

[141] E. Lo, K. Y. Yip, K.-I. Lin, and D. W. Cheung. Progressive skylining over web-accessible databases. *Data & Knowledge Engineering*, 57(2):122--147, 2006.

[142] B. M. Luisa Bentivogli, Pamela Forner and E. Pianta. Revising wordnet domains hierarchy: Semantics, coverage, and balancing. In *COLING, Workshop on Multilingual Linguistic Resources*, pages 101--108, 2004.

[143] C. Luo, H. Thakkar, H. Wang, and C. Zaniolo. A native extension of sql for mining data streams. In *SIGMOD*, pages 873--875, 2005.

[144] Z. Ma, G. Pant, and O. R. L. Sheng. Mining competitor relationships from online news: A network-based approach. *Electronic Commerce Research and Applications*, 2011.

[145] W. Maass. Efficient agnostic pac-learning with simple hypothesis. In *COLT*, pages 67--75, 1994.

[146] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122--173, 2005.

[147] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Googleś deep web crawl. *The VLDB Journal*, 1(2):1241--1252, 2008.

[148] M. Magnani and I. Assent. From stars to galaxies: skyline queries on aggregate data. In *EDBT*, pages 477--488, 2013.

[149] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, 2010.

[150] Y. Mejova, P. Srinivasan, and B. Boynton. Gop primary season on twitter: "popular" political sentiment in social media. In *WSDM*, pages 517--526, 2013.

[151] M. Mikolajczak, V. Tran, C. Brotheridge, and J. J. Gross. *Using an emotion regulation framework to predict the outcomes of emotional labour*, chapter 11. Emerald, 2009.

[152] G. Mishne and M. de Rijke. Capturing global mood levels using blog posts. In *AAAI-CAAW Symposium*, pages 145--152, 2006.

[153] A. Mislove, S. Lehmann, Y.-Y. Ahn, J.-P. Onnela, and J. N. Rosenquist. Understanding the demographics of twitter users. In *ICWSM*, pages 554--557, 2011.

[154] G. Mitchell, S. B. Zdonik, and U. Dayal. Object-oriented query optimization: What's the problem? Technical report, Providence, RI, USA, 1991.

[155] F. Morstatter, J. ürgen Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. In *ICWSM*, 2013.

[156] J. Murphy and D. R. Armitage. Merging the modelled and working address database: A question of dynamics and data quality. http://limerickcity.ie/IT/GIS-GeographicalInformationSystems/AssociatedDocuments/NewsletterFile,3692,en.pdf, Accessed on Sept. 2, 2012.

[157] A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad. Integration of data mining with database technology. In *VLDB*, pages 719--722, 2000.

[158] J. Nichols, J. Mahmud, and C. Drews. Summarizing sporting events using twitter. In *IUI*, pages 189--198, 2012.

[159] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *JCDL*, pages 100--109, 2005.

[160] C. Ordonez. Integrating k-means clustering with a relational dbms using sql. *IEEE Transactions on Knowledge and Data Engineering*, 18:188--201, February 2006.

[161] M. Osborne, S. Petrovic, R. McCreadie, C. Macdonald, and I. Ounis. Bieber no more: First story detection using twitter and wikipedia. In *SIGIR 2012 Workshop on Time-aware Information Access (#TAIA2012)*, 2012.

[162] G. Paltoglou and M. Thelwall. Twitter, myspace, digg: Unsupervised sentiment analysis in social media. *ACM Transactions on Intelligent Systems and Technology*, 3(4):66:1--66:19, Sept. 2012.

[163] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1--135, Jan. 2008.

[164] G. Pant and O. R. L. Sheng. Avoiding the blind spots: Competitor identification using web text and linkage structure. In *ICIS*, 2009.

[165] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41--82, 2005.

[166] A. N. Papadopoulos, A. Lyritsis, and Y. Manolopoulos. Skygraph: an algorithm for important subgraph discovery in relational graphs. *Data Mining and Knowledge Discovery*, 17(1):57--76, 2008.

[167] J. W. Pennebaker and L. A. King. Linguistic styles: language use as an individual difference. *Journal of personality and social psychology*, 77(6):1296, 1999.

[168] D. Pisinger. Upper bounds and exact algorithms for p-dispersion problems. *Computers & Operations Research*, 33(5):1380--1398, 2006.

[169] J. F. Porac and H. Thomas. Taxonomic mental models in competitor definition. *The Academy of Management Review*, 2008.

[170] M. E. Porter. *Competitive Strategy: Techniques for Analyzing Industries and Competitors*. Free Press, 1980.

[171] D. Quercia, M. Kosinski, D. Stillwell, and J. Crowcroft. Our twitter profiles, our selves: Predicting personality with twitter. In *SocialCom*, October 2011.

[172] A. Rajaraman and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[173] J. H. Ratcliffe. On the accuracy of tiger-type geocoded address data in relation to cadastral and census areal units. *International Journal of Geographical Information Science*, 15(5):473--485, 2001.

[174] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tavyi. Heuristic and special case algorithms for dispersion problema. *Operations Research*, 42(2):299--310, 1994.

[175] M. Richardson. Beyond pagerank: Machine learning for static ranking. In *WWW*, pages 707--715. ACM Press, 2006.

[176] A. Ritter, Mausam, O. Etzioni, and S. Clark. Open domain event extraction from twitter. In *SIGKDD*, pages 1104--1112, 2012.

[177] D. M. Romero, C. Tan, and J. Ugander. On the interplay between social and topical structure. In *ICWSM*, 2013.

[178] D. Saez-Trumper, G. Comarela, V. Almeida, R. Baeza-Yates, and F. Benevenuto. Finding trendsetters in information networks. In *SIGKDD*, pages 1014--1022, 2012.

[179] S. Sagiroglu, U. Yavanoglu, and E. N. Guven. Web based machine learning for language identification and translation. In *ICMLA*, pages 280--285, 2007.

[180] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, pages 851--860, 2010.

[181] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *SIGGIS*, pages 42--51, 2009.

[182] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. *Data Mining and Knowledge Discovery*, 4(2):89--125, 2000.

[183] K.-U. Sattler and O. Dunemann. Sql database primitives for decision tree classifiers. In *CIKM*, pages 379--386, 2001.

[184] R. Schifanella, A. Barrat, C. Cattuto, B. Markines, and F. Menczer. Folks in folksonomies: social link prediction from shared metadata. In *WSDM*, 2010.

[185] D. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley, 1992.

[186] P. Senellart, A. Mittal, D. Muschick, R. Gilleron, and M. Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *WIDM*, pages 9--16, 2008.

[187] V. Sengar, T. Joshi, J. Joy, S. Prakash, and K. Toyama. Robust location search from text queries. In *GIS*, pages 1--8, 2007.

[188] P. Serdyukov, V. Murdock, and R. van Zwol. Placing flickr photos on a map. In *SIGIR*, pages 484--491, 2009.

[189] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin--Madison, 2009.

[190] D. A. Shamma, L. Kennedy, and E. F. Churchill. Tweet the debates: understanding community annotation of uncollected sources. In *SIGMM Workshop on Social Media*, pages 3--10, 2009.

[191] C. Sheng and Y. Tao. On finding skylines in external memory. In *PODS*, pages 107--116, 2011.

[192] K. Spärck-Jones, S. E. Robertson, and M. Sanderson. Ambiguous requests: implications for retrieval tests, systems and theories. *SIGIR Forum*, 41(2):8--17, 2007.

[193] E. Spertus and L. A. Stein. Squeal: A structured query language for the web. In *WWW*, pages 95--103, 2000.

[194] A. Stefanidis, A. Crooks, and J. Radzikowski. Harvesting ambient geospatial information from social media feeds. *GeoJournal*, 78(2):319--338, 2013.

[195] J. Stoyanovich, W. Mee, and K. A. Ross. Semantic ranking and result visualization for life sciences publications. In *ICDE*, pages 860--871, 2010.

[196] D. Stutzbach, R. Rejaie, N. G. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. In *Internet Measurement Conference*, pages 27--40, 2006.

[197] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187--198, 2006.

[198] J. Sutton, L. Palen, and I. Shlovski. Back-channels on the front lines: Emerging use of social media in the 2007 southern california wildfires. In *ISCRAM*, 2008.

[199] P. Symeonidis, A. Papadimitriou, Y. Manolopoulos, P. Senkul, and I. H. Toroslu. Geo-social recommendations based on incremental tensor reduction and local path traversal. In *GIS-LBSN*, pages 89--96, 2011.

[200] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892--903, 2009.

[201] H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, and C. Zaniolo. Smm: A data stream management system for knowledge discovery. In *ICDE*, pages 757--768, April 2011.

[202] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN*, 2005.

[203] A. Toral and R. Munoz. A proposal to automatically build and maintain gazetteers for named entity recognition by using wikipedia. *EACL*, 2006.

[204] G. Valkanas and D. Gunopulos. Location extraction from social networks with commodity software and online data. In *ICDM Workshops (SSTDM)*, 2012.

[205] G. Valkanas and D. Gunopulos. How the live web feels about events. In *CIKM*, 2013.

[206] G. Valkanas and D. Gunopulos. A ui prototype for emotion-based event detection in the live web. In *CHI-KDD*, pages 89--100, 2013.

[207] G. Valkanas, D. Gunopulos, I. Boutsis, and V. Kalogeraki. An architecture for detecting events in real-time using massive heterogeneous data sources. In *BigMine*, pages 103--109, 2013.

[208] G. Valkanas, D. Gunopulos, I. Galpin, A. J. G. Gray, and A. A. A. Fernandes. Extending query languages for in-network query processing. In *MobiDE*, pages 34--41, 2011.

[209] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos. Skydiver: A framework for efficient skyline diversification. In *EDBT*, pages 406--417, 2013.

[210] V. Vapnik. *Statistical learning theory*. Wiley, 1998.

[211] K. Vieira, L. Barbosa, J. Freire, and A. S. da Silva. Siphon++: a hidden-webcrawler for keyword-based interfaces. In *CIKM*, pages 1361--1362, 2008.

[212] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. Divdb: A system for diversifying query results. *PVLDB*, 4(12):1395--1398, 2011.

[213] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365--376, 2010.

[214] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, pages 364--372, 2010.

[215] A. Vlachou and M. Vazirgiannis. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data & Knowledge Engineering*, 69(9):943--964, 2010.

[216] S. Wakamiya, R. Lee, and K. Sumiya. Crowd-based urban characterization: extracting crowd behavioral patterns in urban areas from twitter. In *LBSN Workshop*, pages 77--84, 2011.

[217] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *PVLDB*, pages 898--909, 2009.

[218] Q. Wan, R. C.-W. Wong, and Y. Peng. Finding top-k profitable products. In *ICDE*, pages 1055--1066, 2011.

[219] H. Wang and C. Zaniolo. Atlas: A native extension of sql for data mining. In *SDM*, pages 130--141, 2003.

[220] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126--1135, 2007.

[221] J. Weng and B.-S. Lee. Event detection in twitter. In *ICWSM*, 2011.

[222] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112--130, 2006.

[223] T. Wu, Y. Sun, C. Li, and J. Han. Region-based online promotion analysis. In *EDBT*, pages 63--74, 2010.

[224] T. Wu, D. Xin, Q. Mei, and J. Han. Promotion analysis in multi-dimensional space. *PVLDB*, pages 109--120, 2009.

[225] K. Xu, S. S. Liao, J. Li, and Y. Song. Mining comparative opinions from customer reviews for competitive intelligence. *Decision Support Systems*, 2011.

[226] T. Xu and Y. Cai. Location anonymity in continuous location-based services. In *GIS*, pages 1--8, 2007.

[227] Y. Yang, H.-H. Wu, and H.-H. Chen. SHORT: shortest hop routing tree for wireless sensor networks. *International Journal of Sensor Networks*, 2:368--374, July 2007.

[228] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS*, pages 222--227, 1977.

[229] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9--18, 2002.

[230] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *VLDB*, pages 483--494, 2007.

[231] M. L. Yiu and N. Mamoulis. Multi-dimensional top-*k* dominating queries. In *VLDB*, pages 695--718, 2009.

[232] YouTube Service. http://www.youtube.com/.

[233] D. Zelenko and O. Semin. Automatic competitor identification from public information sources. *International Journal of Computational Intelligence and Applications*, 2002.

[234] S. Zhang, N. Mamoulis, D. W. Cheung, and B. Kao. Efficient skyline evaluation over partially ordered domains. *PVLDB*, 3(1-2):1255--1266, 2010.

[235] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD*, pages 107--118, 2004.

[236] Z. Zhang and J. Iria. A novel approach to automatic gazetteer generation using wikipedia. In *People's Web*, pages 1--9, 2009.

[237] Z. Zhang, L. V. S. Lakshmanan, and A. K. H. Tung. On domination game analysis for microeconomic data mining. *ACM Transactions on Knowledge Discovery from Data*, 2009.

[238] H. Zhenhua, X. Yang, and L. Ziyu. l-skydiv query: Effectively improve the usefulness of skylines. *SCIENCE CHINA Information Sciences*, 53(9):1785--1799, 2010.