UNIVERSITY OF CALIFORNIA
RIVERSIDE

Domain-Specific Analysis and Search on User-Generated Content

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Moloud Shahbazi

March 2017

Dissertation Committee:

Dr. Vagelis Hristidis, Chairperson
Dr. Vassilis Tsotras
Dr. Eamonn Keogh
Dr. Jiasi Chen

The Dissertation of Moloud Shahbazi is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

Firstly, I would like to express my gratitude to my adviser Prof. Vagelis Hristidis for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. Besides my adviser, I would like to thank the rest of my thesis committee: Prof. Vassilis Tsotros, Prof. Eamonn Keogh, and Dr. Jiasi Chen, for their insightful comments and encouragement. In addition, I want to thank Prof. Marek Chrobak, chair of CSE department, providing continuous support for graduate students.

My thanks also goes to Dr. Tolga Kunik, Dr. Joseph Bar and Nani Narayanan Srinivasan who provided me an opportunity to join their research teams as intern and have been my collaborators in conducting part of this research. I also would like to thank Dr. Sean Young, my collaborator in cyberbullying research project from University of California Institute for Prediction Technology and partially funding my research.

I thank my fellow lab mates in Databases and Data Mining lab for their feedback, cooperation and of course friendship. In addition I would like to express my gratitude to the staff of Systems group for their support and responsiveness.

Finally, I must express my profound gratitude to my parents, my siblings and my friends for providing me with unfailing support and continuous encouragement throughout my life.

Thanks for your presence!

To my father.

ABSTRACT OF THE DISSERTATION

Domain-Specific Analysis and Search on User-Generated Content

by

Moloud Shahbazi

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2017
Dr. Vagelis Hristidis, Chairperson

User-generated content on the Internet has been explosively growing in the current Web 2.0 era. This has been facilitated through widespread user access to the web through mobile devices, the rapid growth of social media applications, and review-based provider websites. The majority of this data is in the form of free text, as in social posts. Storing and querying this massive unstructured textual data is a challenging task that has been studied extensively recently.

Current search solutions, such as Google, Bing and Amazons internal search, are effective in allowing users to find relevant documents in large collections. Those solutions rely on several content and reputation-based factors including document relevance to the user query. However, capturing and exploiting user intent particularly, in a domain-specific setting, remains an open problem with a variety of research challenges. In this thesis, we study several such settings where existing search techniques are inadequate.

In particular, we studied the following subproblems where we are showcasing the benefit of leveraging domain-specific knowledge and user-generated content: 1) We argue for more effective item ranking for crowd-sourced review platforms and provide efficient algorithms to support it. 2) We provide a practical high-quality solution to build domain-specific ontologies from unstructured text documents. We describe our approach and provide fast and simple algorithms to use the generated ontology in extracting domain-specific features from the textual data. In particular, we describe our approach using a real-estate agency case study where domain agents are interested in evaluating the textual property descriptions. 3) We study how to search for similar documents, given a set of input documents, when the data source can only be accessed through a query interface (such as Google search). We propose a ranking model to extract effective query keywords from the input documents to retrieve similar documents through keyword-based search APIs. 4) We use data mining techniques to classify user-generated content on on-line forums in terms of its characteristics, such as bullying behavior. In particular, we crawl Yik Yak, an anonymous social media, to detect potentially harmful behaviors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the recent years of Web 2.0, user-generated content significantly contributed to the rapid growth of the Web content. Textual content mostly in form of web pages are a major part of this data. Users are easily able to spread their opinions through blogs, social media, crowd-source based websites, etc.

Web search tools such as Google.com and Bing.com are effective for general purpose document search. These search engines, provide web pages of interest to the users based on the query keywords that users input to specify their needs. They apply advanced scoring methods for the documents that match the query keywords in order to provide the users with a list of relevant documents that are ordered with a decreasing search quality order. For an effective web search tool, it is crucial to return the most relevant document first given the limited time that users can spend to find their documents of interest. The ranking score implemented by search engines is a complex combination of several factors including content relevance, quality and other factors. The challenge is to retrieve the high quality search results in a timely manner given the user search keywords. Even though this problem has been well studied for a long time, improving search quality is still an ongoing research problem as of today.

In this thesis, we argue that existing Web search engines are inadequate to solve several important specialized content search problems as discussed next. Specifically, this thesis is concerned with improving user experience in sifting through massive user-generated content on the web by taking into account the specific user demands. We provide effective and efficient solutions for the task.

In my research, I focused on four major sub-problems on improving user experience by modeling and leveraging user-generated content. In particular, we studied the following four problems:

1) Query-Specific Ranking of Reviewed Items One of the most popular types of user-generated content in Web 2.0 are the crowd-sourced reviews of online items (e.g. Amazon products). A review typically consists of a text description and a star rating score. A key challenge is how to leverage reviews to help users find the best items. Existing work has mainly focused on two directions. First, collaborative filtering studies how the reviews of similar users may be leveraged to recommend products to users [107]. Second, reviews – typically their average rating and number – are being used to rank products, along with other features like price, combined by learning to rank algorithms [24]. However, little work has studied how reviews can be used to perform query-specific ranking of items.

In this chapter we argue for a more effective use of reviews in item ranking, by introducing a ranking scheme which takes into account the user's interest in particular aspect of the search result. For example if one is looking for cameras with strong "battery", reviews related to "hours per charge" or "charging time" should receive higher weight than reviews on other camera features.

We define the problem as follows: *given* (a) a set of reviews for each item, where each review consists of a set of concepts and a rating score, (b) a query that consists of a set of concepts, and (c) a concept ontology on which we can define the semantic distances of the concepts, *compute* the top ranked items for the query. In our solution, each review's rating is weighed by the similarity (relevance) of the review to the query. We study two variants of the relevance measurement: the simpler one views terms as concepts and checks for exact match between query and review terms, while the second extracts concepts from the query and the reviews and measures their semantic similarity based on a concept graph.

A key challenge is that we cannot precompute a rating score for an item as it depends on the user query. Existing early termination algorithms that process list prefixes to compute the top-k results [43, 61] cannot be applied, because for an unseen review, we do not know its rating nor its similarity to the query. These two quantities have an intimate dependency to each other and the overall item score.

Further, due to the interplay between similarity (which may be partially known during the execution of the algorithm) and rating of unseen or partially seen reviews, we are faced with a combinatorial number of cases for computing the terminating condition threshold. Instead, we propose a linear cost method to compute this threshold.

Moreover, in contrast to the setting of top-k algorithms [43, 61] where multi-attribute objects are ranked, we rank items that are collections of objects (reviews). We show that this cannot be handled by simply adding another level of top-k lists aggregation. Neither can it be handled by creating a list of items for each concept c, where each item I has a score equal to the aggregate score of this item for query c. This would work for queries with a single concept, but if a query has multiple concepts this approach misses the overlap of concepts inside reviews. That is, the score of a review that contains two query concepts would be counted twice using this approach.

In this work, we define the novel problem of ranking of query results by taking into account the relevance of the items reviews to the query, which constitutes a fresh approach to leveraging reviews for item discovery. We continue by proposing efficient solutions for two popular concept-based similarity measures between a query and a review – the Jaccard similarity that measures the ratio of common concepts and the Path-Length similarity that exploits the ontological relationships between concepts. Finally, we present random and non-random access variants of our solution to compute top-k items ranked by the query-based score of the review. The result of our experimental study shows the effectiveness of our solutions as well as efficiency of algorithms comparing to baseline methods.

The details of this study are presented in Chapter 2.

2) Define and extract Domain-Specific Ontology: User-generated texual content is used for text relevance matching with user's query keywords. Full text indexing methods are the way to accommodate searching for the documents. However, in this work we focus on leveraging textual information to extract domain-specific conceptual features with goodness score to assign a score to the unstructured textual information. For this purpose, we provide an effective and efficient framework that includes solutions to build domain-specific ontologies from unstructured text documents.

In this work, we use a real estate agency as the running domain-specific case study. Real estate agents and other real estate professionals have to sift through hundreds or thousands of listings per day to locate the ones that they should focus on to satisfy their clients, for rent, sale or investment purposes. In order to select profitable property listings to invest, agents need to carefully asses the remarks made by listing agency because they often include crucial information. This includes things like remodeling information (new granite counter-top), financial conditions (short sale, foreclosure), etc. Conversely, for one reason or another, an agent will disclose that the home has "foundational issue", clearly a negative home attribute. To our best knowledge, current property valuation methods do not utilize this kind of information for ranking properties.

Manual exploration of numerous textual descriptions in order to select This is clearly time consuming, translating into significant labor costs. Needless to say that an automated ordering of the items by their investability inferred from agency's remarks is a key factor to lower the costs in finding most profitable properties.

In this work, we propose a novel methodology to assign a "goodness" score based on the textual description of a listing. This score can then be combined with other structured attributes if available to generate an overall goodness score for an item whereby enabling the users to find more pertinent item characteristics.

Specifically, we first build a collection of real-estate-specific concepts that are phrases that describes a real-world entity, such as "granite counter top," to use for annotation. Each concept in this collection is manually labeled with a numeric goodness value by experts. For example, “sell as-is” is a negative concept for ’turn-key’ investors not wishing to further invest to enhancing the condition of a property (like replacing old carpets.) To that end we’ve created a continuous vector representations for vocabulary words by analyzing a corpus of real estate descriptions in order to annotate the property’s textual information with scored concepts.

The details of this study are presented in Chapter 3.

3) Finding Similar Documents using Keyword-Based Search Interfaces A common problem in Information Retrieval is that a user wants to retrieve documents similar to a given set of relevant documents. For example, a patent attorney may have a few documents provided by a client describing an invention, and would like to search for patents similar to these input documents. Similarly, a scientist may search for related work in an area, and may have access to a few documents related to this area. A Web user may also have found a set of documents related to a topic, e.g., related to the topic of academic scandals, and may be looking for more similar documents on the Web.

Although there are powerful keyword search interfaces on top of various collections – LexixNexis for patent search [3], Google Search API for the Web [1], etc. – a common limitation they have is that they do not allow the user to input a set of documents (one or thousands), but expect a relatively small number of terms as a query. Further, these interfaces typically charge a fee for every page of query results. For example, the LexisNexis Statistical Gateway charges \$0.30-\$0.40 per query, and Google Enterprise charges \$100 for 20,000 queries, where each query returns one page of results. Note that a common property in all these collections is that the user may not have access to the underlying collection through any way other than through the provided search APIs. Hence, to use these interfaces, one has to extract sets of important terms from the input documents, to formulate queries. These queries should ideally return many similar documents in high rank positions of the results.

Given a collection is accessible only through a search interface, we propose effective techniques to generate queries from a set of input documents, which return similar documents to the input documents in high positions. We refer to this problem as *Docs2Queries*.

The Docs2Queries problem has received limited attention by the community [106, 124]. The state-of-art works focus on extracting good terms from the input documents, given a basic understanding of the ranking formula, which is generally tf-idf (term frequency and inverse document frequency) based. Specifically, they select terms with high tf-idf score. This is a reasonable heuristic, but its drawback is that it ignores the language model of the collection and limits the heuristics to use only information from the input documents.

We propose a more principled approach to select the best queries, which also considers the language model of the collection (specifically, an estimation of it, as the full collection is not available to us).

The details of this study are presented in Chapter 4.

4) Content Analysis on Anonymous Social Networks

College attendance marks an important period of psychosocial development with significant implications for a healthy and productive adulthood. The academic and social demands of college life are often strenuous and pose a risk to students health and well-being.

One problem among college freshman, for example, is poor sleep [113] which has been linked to a number of adverse consequences in this population, including higher rates of depressive symptoms and stress [120, 50], weight gain [98], and poor academic performance [35]. Another relatively recent problem among college students, due to technologies like social media, is cyberbullying, which can lead to depression and suicide.

In this study, we use data mining techniques to classify user-generated content on anonymous online forums in terms of its characteristics, such as bullying. In particular, we crawl Yik Yak website to collect users discussions. We use the discussions to identify topics discussed and user behaviors to study the behavior trends in different locations and develop a tool to detect potentially harmful behaviors.

Yik Yak is a relatively recent addition to the social media world that quickly grew in popularity among U.S. college students after its inception in 2013. It functions as an online bulletin board on which users within the same geographic area (e.g., a college campus) can post and read messages anonymously. Critics of the social network argueaided by anecdotal evidence relayed through media reportsthat anonymous posting encourages harassment and bullying [63, 7, 8, 117]. In a recent content analysis of Yik Yak conversations, Black, Mezzina, and Thompson did not find evidence for a pervasive culture of harassment and abuse, but they did observe derogatory and incendiary comments, arguably racist and sexist messages, and several likely instance of bullying [19].

Based on these findings, we carried out an exploratory study of posts on the Yik Yak social network. Our goal is to help provide insights for school administrators and public health researchers on the prevalence and popularity of messaging behaviors such as bullying and social support, and of topics discussed on the network. Knowledge of these activities on Yik Yak can then be used to improve student well-being, for example, by guiding interventions that promote healthy and prosocial behaviors.

The details of this study are presented in Chapter 5.

Chapter 2

Query-Specific Ranking of Reviewed Items

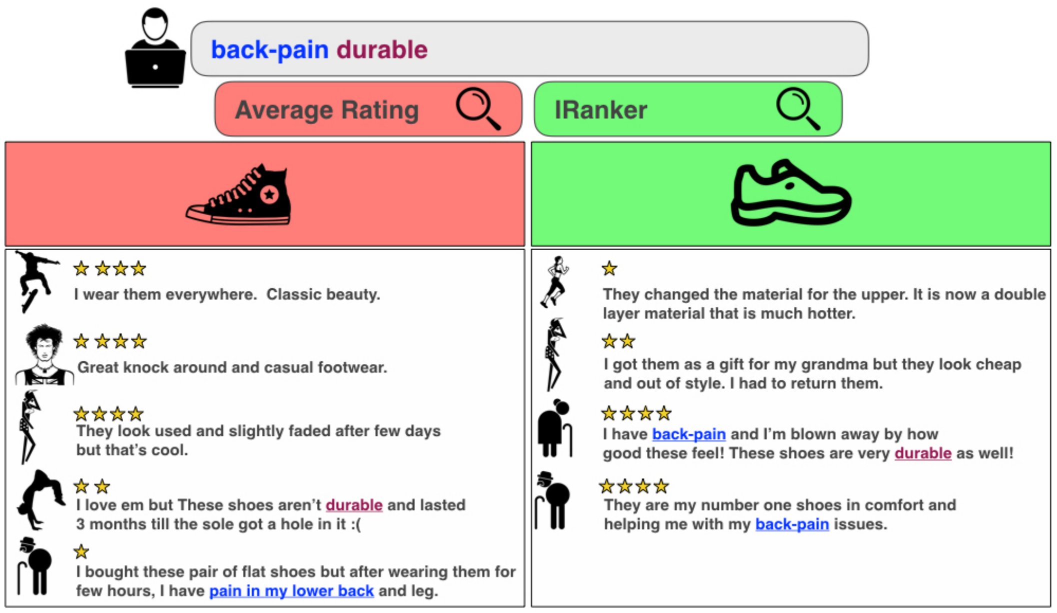
Item (e.g., product) reviews are one of the most popular types of user-generated content in Web 2.0. Reviews have been effectively used in collaborative filtering to recommend products to users based on similar users, and also to compute a product's star rating. However, little work has studied how reviews can be used to perform query-specific ranking of items. In this chapter, we present efficient top-k algorithms to rank items, by weighing each review's rating by its relevance to the user query. Further, we show how our algorithms can efficiently handle ontological relationships between query and review concepts to improve the search quality. We consider various query-review similarity distance semantics, and propose both random and non-random access algorithm variants. We perform a comprehensive evaluation of our methods on multiple datasets and show that they significantly outperform baseline approaches in terms of query response time. In addition, a user study shows that the accuracy of the results is superior compared to the baseline methods.

2.1 Introduction

In the current era of Web 2.0, users generate huge numbers of reviews for products and services in variant domains such as movies and physicians. A review typically consists of a text description and a star rating score. There has been much work on several aspects of Internet reviews such as extracting features from reviews [93], summarizing them [59], and detecting fake reviews [88].

A key challenge is how to leverage reviews to help users find the best items. Existing work has mainly focused on two directions. First, collaborative filtering studies how the reviews of similar users may be leveraged to recommend products to users [107]. Second, reviews – typically their average rating and number – are being used to rank products, along with other features like price, combined by learning to rank algorithms [24].

In this chapter we argue for a more effective use of reviews in item ranking, by introducing a ranking scheme which takes into account the user's interest in particular aspect of the search result. Figure 2.1 shows an example of a user of an on-line footwear store who searches for "durable" shoes that are also suitable for "back-pain". If the search results are sorted by the commonly used average rating score, the left shoe is ranked higher, even though the right shoe has better ratings for the user's properties of interest. *IRanker improves the results' quality by considering the relevance of each review to the user query*, where a query can be expressed by a list of keywords or concepts. Similarly, if one is looking



☆☆☆☆
I wear them everywhere. Classic beauty.

☆☆☆☆
Great knock around and casual footwear.

☆☆☆☆
They look used and slightly faded after few days but that's cool.

☆☆
I love em but These shoes aren't durable and lasted 3 months till the sole got a hole in it :(

☆
I bought these pair of flat shoes but after wearing them for few hours, I have pain in my lower back and leg.

☆
They changed the material for the upper. It is now a double layer material that is much hotter.

☆☆
I got them as a gift for my grandma but they look cheap and out of style. I had to return them.

☆☆☆☆
I have back-pain and I'm blown away by how good these feel! These shoes are very durable as well!

☆☆☆☆
They are my number one shoes in comfort and helping me with my back-pain issues.

Figure 2.1: Two possible top search result for a query.

for cameras with strong "battery", reviews related to "hours per charge" or "charging time" should receive higher weight than reviews on other camera features.

As another application, consider a physician who has neutral reviews on managing "heart disease" but positive reviews on "anemia" treatment. This physician should be ranked relatively high if one is looking for a physician for treating "erythrocytosis", given that "erythrocytosis" is semantically very related to "anemia" but not to "heart disease", as shown in the concept graph of Figure 2.2. This figure shows a subset of the SNOMED-CT ontology, which contains more than 310,000 health-related concepts including symptoms and disorders [118]. Clearly, the semantic relationships between the concepts (referred as features in other review papers) of a review and the query must be taken into account when computing the relevance of a review.

Problem Statement: Our core problem is defined as follows: *given* (a) a set of reviews for each item, where each review consists of a set of concepts and a rating score, (b) a query that consists of a set of concepts, and (c) a concept ontology on which we can define the semantic distances of the concepts, *compute* the top ranked items for the query. In our solution, overviewed in Figure 2.3, each review's rating is weighed by the similarity (relevance) of the review to the query. We study two variants: the simpler one views terms as concepts and checks for exact match between query and review terms, while the second extracts concepts from the query and the reviews (see Sections 3.2 and 2.3 for details) and measures their semantic similarity based on a concept graph.

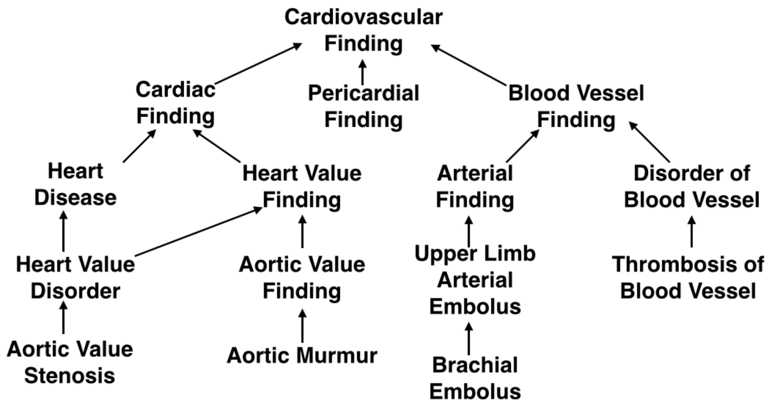


Figure 2.2: A subset of SNOMED-CT medical ontology. In this directed acyclic graph, the medical concepts are connected via directed "is-a" links.

Challenges and Algorithms Overview: A key challenge is that we cannot precompute a rating score for an item as it depends on the user query. Existing early termination algorithms that process list prefixes to compute the top-k results [43, 61] cannot be applied, because for an unseen review, we do not know its rating nor its similarity to the query. These two quantities have an intimate dependency to each other and the overall item score.

In particular, to compute the minimum or maximum score of an item, which are quantities necessary for early termination, we must decide what the extreme rating and similarity values of the unseen reviews may be. For the maximum score estimation, one may think that this can be achieved by considering the maximum possible rating and the maximum possible similarity for the unseen reviews. However, we prove that this is not correct. In particular, we show that for some of the partially seen reviews the maximum possible similarity must be considered, whereas for other reviews the minimum similarity must be considered.

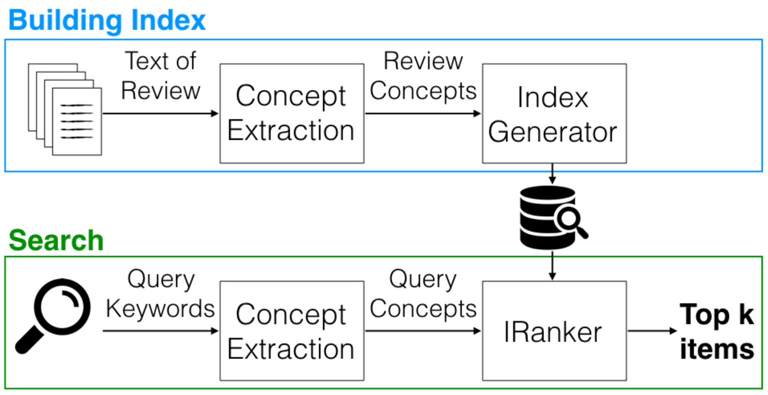


Figure 2.3: Overview of the steps involved in computing the top-k search results. First, necessary data indexes are built. Then, IRanker computes the top-k items by accessing the data stored in these indexes.

Further, due to the interplay between similarity (which may be partially known during the execution of the algorithm) and rating of unseen or partially seen reviews, we are faced with a combinatorial number of cases for computing the terminating condition threshold. Instead, we propose a linear cost method to compute this threshold (see Theorems 2.4.1 and 2.4.2 for unseen reviews, and Theorems 2.4.3 and 2.4.4 for partially seen reviews).

Moreover, in contrast to the setting of top-k algorithms [43, 61] where multi-attribute objects are ranked, we rank items that are collections of objects (reviews). We show that this cannot be handled by simply adding another level of top-k lists aggregation. Neither can it be handled by creating a list of items for each concept c, where each item I has a score equal to the aggregate score of this item for query c. This would work for queries with a single concept, but if a query has multiple concepts this approach misses the overlap of concepts inside reviews. That is, the score of a review that contains two query concepts would be counted twice using this approach.

Another challenge in that due to the relationships of the concepts on the ontology graph (Figure 2.2), we must traverse several concept lists for a single query concept c, starting from the list of c and then later move to the lists of the neighbors of c in the concept graph G. The fact that several lists are traversed complicates the estimation of the upper and lower bounds of the similarity of unseen reviews, which are quantities necessary for a top-k algorithm.

In this chapter, we make the following contributions:

- We define the novel problem of ranking of query results by taking into account the relevance of the items reviews to the query, which constitutes a fresh approach to leveraging reviews for item discovery (Section 2.3).

- We propose efficient solutions for two popular concept-based similarity measures between a query and a review – the Jaccard similarity that measures the ratio of common concepts and the Path-Length similarity that exploits the ontological relationships between concepts. We present random and non-random access variants (Section 2.4).

- We perform a user study and show that the ranking semantics used in this chapter have better quality than baseline ranking methods (Section 2.5.2).
- We show that our algorithms outperform baseline methods on both real and synthetic datasets. In particular, we experiment on Amazon products [82] using Jaccard similarity and physician reviews using both Jaccard and the SNOMED-CT biomedical ontology [118] (Section 2.5.1).

Related work is presented in Section 3.2 and we conclude in Section 3.5.

2.2 Related Work

Reviews summarization: There is much work on extracting concepts (features) from reviews and using these concepts to summarize the reviews for a user. Typically, the features and their corresponding sentiment are extracted from each review [93], and then the sentiment for each feature is aggregated across all the reviews of a product [59]. However, this precomputed summarization does not work in our setting because it ignores the ontological relationships between features (concepts). For instance, suppose that for an item we have aggregated the rating for each concept and it has score 0.6 for "Arterial Finding" and 0.2 for "Disorder of Blood Vessel" (see Figure 2.2). Then, if a query specified their parent concept "Blood Vessel Finding", we cannot simply consider the average score. Because the score of a review that contains both "Arterial Finding" and "Disorder of Blood Vessel" would be double counted. Even if similarity measures that are based on exact concept matching is used, given that a query contains an arbitrary number of concepts, precomputing the item score for every possible query is not a practical solution.

Concepts extraction, ontologies and concept similarity: In order to compute semantic similarity of concepts, we rely on domain specific ontologies. In our experiments, we use doctors and their online reviews collected from www.vitals.com, a doctor reviews website. For the doctor reviews data, we use existing SNOMED-CT ontology [118], which is one of the largest and more widely used ontologies for general health text. There are multiple NLP-based tools such as cTAKES [101] and MetaMap [9] to extract general health concepts from free text. Note that our solutions are not limited to the health domain, as domain specific information extraction tools may be applied to extract concepts [25], along with domain specific ontologies, or to define distances between terms [86]. Alternatively, existing ontologies may be used, such as ConceptNet, which is a general purpose ontology [77]. Further, previous work has studied how a feature ontology can be built for product reviews [110].

There are several works that study the semantic distances of concepts in ontologies. XOntoRank [44] proposes several alternative measures that generally include the number of edges between two concepts and possibly the fan-out of a concept. They exploits ontological relationships to answer keyword queries on XML documents. There are several studies that review semantic similarity measures applied to semantic ontologies and classify them according to their strategies [91, 84, 45]. According to these studies, complex strategies create performance limitations while they do not significantly improve the quality of semantic similarity.

Ranking database query results: There has been work on ranking database tuples [26] that satisfy a conjunctive query. In that work, all results satisfy all query conditions and are ranked based on their unspecified attributes. A key difference is that in our problem we rank "complex" items that consist of reviews and also the reviews contain interrelated concepts.

Top-k algorithms: Document-at-a-time and term-at-a-time (DAAT) strategies [12, 112] have been commonly used for computing the top-k query results in IR systems. These strategies compute complete document scores by accessing the whole lists of documents indexed by query terms. Thus, they are inefficient when applied to large corpus with large lists of documents. There are several optimization techniques for both approaches that reduce the time to generate the document scores. One optimization for DAAT is proposed by Broder et al. [23], where the term lists are sorted by document id, and chunks of the lists are skipped if they have no chance of making it to the top-k result. They use a branch-and-bound approach to compute efficient moves for the cursors associated to the postings lists. However, this optimization still requires to process the lists until the end, and also cannot handle ontological distances. Another line of work is the Threshold Algorithm (TA), with the most well-studied example being [43]. Fagin's TA algorithm computes the optimal top-k items with different features and score per feature based on a monotonic aggregation function, where there is a ranked list of items per feature.

Our work with respect to prior work: Our work follows the general paradigm of parallel accessing ordered lists of elements and terminating when a threshold condition is satisfied [61]. As described in Section 4.1 *there is no top-k algorithm that handles the unique requirements of our problem* which include: (a) ranked items are complex, i.e., they consist of reviews, and (b) each review not only has a score but a similarity to the query concepts.

Table 2.1: Examples of doctor reviews.

Doctor	Original Text of the Review	Review Concepts	Rating
D1	“He remembered that I have trouble with my heart valve and prescribed medicine accordingly.”	{heart valve, medicine}	1
D1	“An excellent doctor who has guided me through my diabetic struggles.”	{Diabetic}	0.5

Specifically, each document (review) has a rating and the item’s aggregate score is the weighted average of the ratings of the item’s reviews, where reviews are weighted by their relevance to the query. Hence, if we rank reviews in the index lists by the score, it may be the case that the highest ranked reviews have very low similarity to the query Q and hence low effect on the overall score of an item. Requirement (a) requires new list traversal techniques whereas requirement (b) requires novel threshold estimation techniques.

2.3 Definitions

We begin by defining the key terms that are necessary to state the problem. A *concept* is a semantic entity, which is related to other concepts through a *concept graph.* A concept graph is a graph consisting of concepts as nodes and the semantic relation between the concepts as the links. In our experiments, we use the SNOMED-CT ontology [84] as our concept graph, where we consider directed links of type “is-a” as the semantic relations

(Figure 2.2). Note that the concept graph is utilized in only one of our two similarity variants discussed below.

An *item* I, which may be a doctor or an Amazon product, contains a set $\{r_1, r_2, ..., r_n\}$ of user reviews. A *review* $r = \{c_1, c_2, ..., c_n; rating\}$ consists of a set of concepts c_i and a numeric rating score between 0 and 1. For instance, a review may be $r = \{\text{“}cancer\text{”}, \text{“}cardiac\text{”}, \text{“}EKG\text{”}; 0.6\}$, which expresses that the concepts “cancer”, “cardiac” and “EKG” are mentioned in the review, and the user rated the item with a score of 0.6.

It is possible that a reviewer may have a more positive opinion about some of the concepts in a review, and hence using the review’s overall score for all concepts may be misleading. Unfortunately, we found that assigning individual sentiments to each concept is challenging and inaccurate using existing methods, so we have left it as future work. If such fine-grained scoring were available, our algorithms could be easily adapted such that each (concept, rating) pair is viewed as a separate single-concept review.

The concepts associated with a review are extracted from the text of the review using a domain-specific information extraction method. In our experiments we use MetaMap [9], an NLP tool developed by the NIH to extract UMLS (SNOMED-CT is a subset of UMLS) medical concepts. Several other such tools are available for the medical and other domains. Table 2.1 shows doctor reviews from the Web and the concepts extracted by MetaMap. If the user expresses the query in free text, the concepts are extracted in a similar way.

Item Ranking Problem: Given a collection $\mathcal{I}$ of items, a *query* $Q = \{q_1, q_2, ..., q_m\}$, which is a set of concepts, and k requested top results, return the top-k items in $\mathcal{I}$ with highest score $score(Q, I)$.

Table 2.2: Key notation.

Symbol	**Definition**	**Section**
$score(Q, I)$	Item I's ranking score given query Q	2.3
$QRSim^J(Q, r)$	Jaccard-based query-review similarity	2.3
$QRSim^P(Q, r)$	Path-Length based query-review similarity	2.3
y_{max}	Maximum possible score of an unseen item	2.4.1
Y	Vector of last accessed reviews from concept list	2.4.1
$XPScore(Q, I)$	Maximum possible score of the partially seen item I given query Q	2.4.2
$MPScore(Q, I)$	Minimum possible score of the partially seen item I given query Q	2.4.2
U	Set of possible [Rating, Similarity] pairs of unseen reviews	2.4.2
$MPQRSim(Q, r)$	Minimum possible query-review similarity	2.4.2
$XPQRSim(Q, r)$	Maximum possible query-review similarity	2.4.2

The scoring function in Equation 2.1 is partly inspired by Zhang et al. [128], where they weigh product ratings based on the usefulness of each review. The main distinction of our method is that we average reviews' rating weighted by their relevance to the query. If review usefulness is available, it can also be multiplied. Note that usefulness is fixed for a review, in contrast to relevance which is query-dependent.

$$score(Q, I) = \frac{\sum_{r \in I} QRSim(Q, r.concepts) \times r.rating}{\sum_{r \in I} QRSim(Q, r.concepts)}, \tag{2.1}$$

where $QRSim(Q, r)$ is the similarity of Q to the concepts *r.conce-pts* in r. Table 2.1 demonstrates two example reviews of a doctor. If a user searches for doctors by submitting query "aortic valve", the first review is relevant because "aortic valve" is a child of "heart valve" in Figure 2.2, but the second review is not.

We can extend this ranking function to include other item and review features such as number of item reviews, price or popularity or review helpfulness and freshness. Our main focus is on how to compute the top results efficiently.

The problem of computing the semantic similarity between two sets of concepts (of the item and the query) has been extensively studied in the past [96, 13, 84]. In this work, we consider two popular and representative similarity measures.The first and simpler is the Jaccard similarity, which does not consider the possible relationships between the concepts. Such a measure would be appropriate for applications where no concept graph is available, or only exact concept matches are appropriate.

$$QRSim^J(Q,r) = \frac{|Q \cap r.concepts|}{|Q \cup r.concepts|} \tag{2.2}$$

The second similarity measure considers the semantic distances between concepts in the concept graph G. For each concept in the query, we find the closest concept in the review.

$$QRSim^P(Q,r) = \sum_{c_i \in Q} \max_{c_j \in r.concepts} \{1 - distance(c_i, c_j)\} \tag{2.3}$$

Similarly to previous work [96] and [13], we define the semantic distance $distance(c_i,$ $c_j)$ between two concepts c_i and c_j as the shortest path distance between them in the concept graph. As in previous work [10], in practice we set a maximum threshold T on the shortest-path distance between two concepts for them to be meaningfully similar (e.g., $T = 3$ edges). Then, to normalize $distance(c_i, c_j)$ in $[0, 1]$, we divide by T. If the shortest path distance is greater than T then we set $distance(c_i, c_j)$ to 1. Table 2.2 summarizes the key notation of this chapter.

2.4 Ranking Algorithms

In this section, we present our solutions to compute the top-k items for the item ranking problem.

Background on TA and NRA algorithms: Early termination algorithms form the basis for our algorithms. Fagin et al. [43] proposed two algorithms with different access modes – TA has random access and NRA does not – for finding the top-k multi-attribute objects, based on a monotonic aggregation function, when there is ranked list for each attribute. These algorithms perform sorted access to the attribute lists in parallel, and terminate when k objects are found whose scores are greater than the maximum possible score of other objects.

Proposed Algorithms: As discussed in Section 4.1, existing algorithms like TA and NRA cannot solve our problem. Here we propose the IRanker algorithms that address these challenges. We first present RA-IRanker, an algorithm which solves our problem by computing exact top-k results of the search query using random access to data indexes. Because random accesses are costly, we then propose NRA-IRanker, a non-random access top-k items ranking algorithm that also computes the exact results. NRA-IRanker also reduces the required index disk space. We show how each algorithm can be adapted for Jaccard and Path-Length similarity measures. For the proposed algorithms to work, the aggregate function in Equation 2.1 must be monotonic on the review ratings. That is, if the rating of any review of an item is increased, the score of that item is also increases, or stays the same. It is fairly easy to see that Equation 2.1 satisfies this condition.

Indexes: In order to store the data for the algorithms, we use two indexes for RA-IRanker and one index for NRA-IRanker. Table 2.3 shows different indexes with the information stored in each one.

Concepts index is an inverted index that has a list for each concept. This list contains reviews that include the key concept sorted by decreasing review rating. Figure 2.5 shows an example of a *Concepts* index.

Items index is a map used by RA-IRanker. It has a list for each item, which stores the reviews of that item along with their concepts.

In our random access method, RA-IRanker, the exact score of an item I is computed when it is seen for the first time while processing a review in a query concept list. For every new item, a random access to *Items* index with item-id retrieves all the reviews in order to compute the exact score of an item. Note that for the Path-Length similarity, we also access the neighboring concepts lists as needed. In NRA-IRanker, since we do not perform random access at all, we must estimate the minimum and the maximum possible similarities of every seen review, in addition to the minimum and the maximum possible scores of every item. Next we explain each algorithm in detail.

2.4.1 RA-IRanker

Note that as we mentioned previously, every concept in *Concepts* index, points to a sorted list of reviews which contains the key concept. Each review entry in the list

Table 2.3: Indexes used by each IRanker variant. content of index entries for each algorithm are listed.

IRanker Algorithms	**Concepts Index** *(Sequential Access)*	**Items Index** *(Random Access)*
RA-IRanker	concept→ List of (review-id,item-id,rating) tuples sorted by rating	item-id→ List of (review-id, review concepts, rating) tuples
NRA-IRanker	concept→List of (review-id, item-id, rating, #item-reviews, #review-concepts) tuples sorted by rating	---

is a (review-id, item-id, review rating) tuple. The lists of review entries are sorted in a descending order by the review rating. Figure 2.5 shows a simple example of a *Concepts* index.

Jaccard Similarity

When a query Q is issued, RA-IRanker works as following:

While top-k items are not found, repeat:

1. Do a sequential sorted access to the query concepts lists in parallel. Every time select next review from the list with the maximum current review score. Then, update the maximum possible score of the unseen review which we denote by y_{max}.

2. For each item that has been seen in the concept list, do a random access to *Items* index. Then, compute the item's exact score using all its reviews that are fetched from *Items* index.

3. Check termination condition by examining if there are k items with score greater than or equal to the y_{max}. Note that both maximum possible rating of the unseen review

and the maximum possible score of the unseen items are equal to y_{max} (see text below for details).

To compute y_{max}, we need to estimate the reviews of the unseen item that maximizes the score; note that we do not know which item that is yet. According to Equation 2.1, if an unseen item has one or more reviews, and all of them contain only the query concept (a condition which maximizes the Jaccard similarity shown in Equation 2.2) with maximum rating (which we assume to be equal to the last seen review's rating), then y_{max} is equal to that rating. As a result, y_{max} is equal to the maximum rating of the last seen review across all query concepts lists. Formally, if $Y = \{y_1, \cdots, y_m\}$, where y_i is the current rating of the ith concept list, then $y_{max} = max\ Y$.

Path-Length Similarity

If Path-Length similarity is used, in addition to the query concepts, we also need to consider semantically close concepts to each query concept which are obtained from the concept graph. For each query concept, we find concepts in the graph that are no farther than a threshold T steps from them in a breadth-first search way.

Similarly to Jaccard-based approach, we iterate over the lists of query concepts and their neighbors, where the similarity of a review is adjusted based on the concept's distance from the query concept as described in Equation 2.3. Figure 2.4 shows an example graph with two query concepts and their neighbor concepts with distance threshold $T = 2$. For each concept node, a chunk of reviews from concept index, as well as the rating of the last seen review from this list, is stored in the memory.

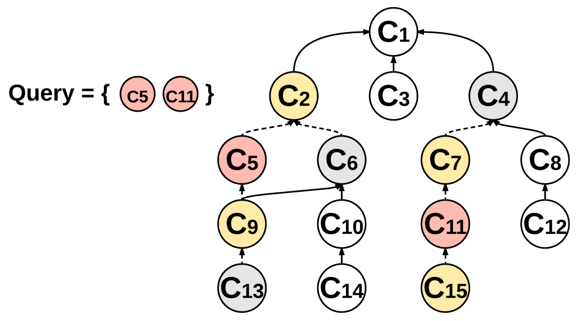


Figure 2.4: A labeled DAG representing sub-graph of SNOMED-CT ontology in Figure 2.2. Node 'C_5' and node 'C_{11}' are two query concepts and their neighborhood graph is extended with $T = 2$.

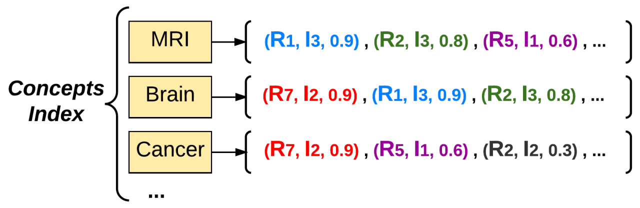

Concepts Index
MRI
(R1, I3, 0.9) , (R2, I3, 0.8) , (R5, I1, 0.6) , ...
Brain
(R7, I2, 0.9) , (R1, I3, 0.9) , (R2, I3, 0.8) , ...
Cancer
(R7, I2, 0.9) , (R5, I1, 0.6) , (R2, I2, 0.3) , ...
...

Figure 2.5: An example of *Concepts* index in RA-IRanker. For each concept, a list of the reviews that contain that concept is stored in the index. Each review entry in the concepts' lists is a (Review-Id, Item-Id, Rating) tuple.

2.4.2 NRA-IRanker

Similarly to the previous method, NRA-IRanker processes the reviews by accessing the lists of query concepts in parallel but unlike that, it does not make any random access to *Items* index. *Concepts* index for NRA-IRanker is slightly different than RA-IRanker. In NRA-IRanker, every review in a concept list includes two extra values for the total number of item's reviews and the total number of the concepts of the review. Since there is no random access made for reviews, NRA-IRanker needs to keep track of partially seen items as well as the partially seen reviews.

As a result of NRA-IRanker access pattern, there are three types of reviews: *(1) Seen reviews, (2) Partially seen reviews and (3) Unseen reviews.* A *seen* review is a review that has been seen in all possible query concept lists, except the lists that have been completely read or have a current rating smaller than the review's rating. Thus, the exact similarity of the seen reviews to the query can be computed. A *partially seen* review is the one currently seen in at least one query concept list and could be seen in more lists in the future. For every partially seen review, we maintain a minimum and a maximum possible similarity to the query.

As we mentioned earlier, the total number of reviews of an item is stored in the review entries. Therefore, the number of unseen reviews of a *partially seen* item is available for computing the minimum and the maximum possible score of a *partially seen* item. We define three types of items using number of unseen reviews: *(1) Seen items, (2) Partially seen items and (3) Unseen items.* A *seen* item is an item for which all reviews have been seen in the concepts lists and their similarities are computed. Thus, the item's exact score is already computed. *Partially seen* items are the items that have at least one *seen* or *partially seen* review but not all of their reviews are seen. The minimum and the maximum possible score needs to be maintained for these items in order to be used when checking the termination condition. Figure 2.6 illustrates various types of reviews and items according to the access patterns defined above.

Jaccard Similarity

Computing Review Similarity Bounds: Before we describe the process of computing the bounds, it is necessary to define the concept of an "unsure" list. If a partially seen review is not yet seen in a concept list and the current rating of this list is greater than

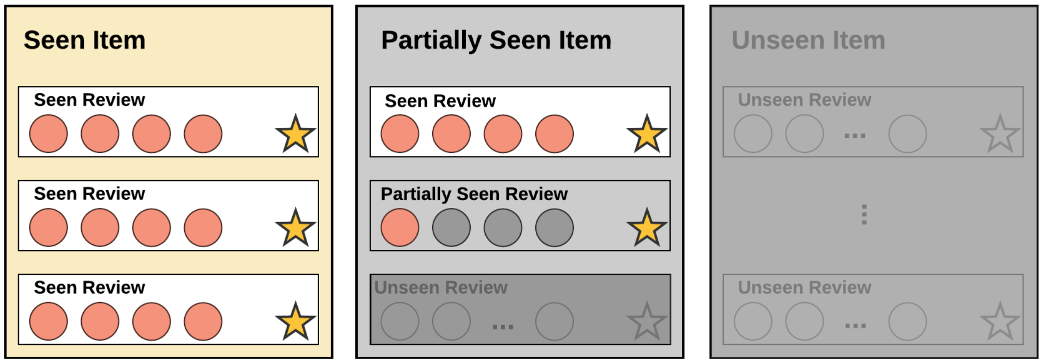


Figure 2.6: Example of various types of reviews and items based on the access pattern taxonomy we define in Section 2.4.2. The unseen entities are shown in dark gray. Star signs denote review ratings and circles represent the concepts.

or equal to the rating of the review, the presence of the review in this list is "unsure" (since concept lists are sorted by review rating). If a review r is "unsure" in z lists out of the $|Q|$ lists of the query concepts, the maximum possible similarity of this review to the query is achieved when this review is in all "unsure" lists. On the other hand, the minimum possible similarity is achieved when the review does not exist in any "unsure" list. Equations 2.4 and 2.5 define the maximum and the minimum possible similarities of the partially seen reviews to the query Q respectively.

$$XPQRsim^J(Q,r) = \frac{z+f}{|r|+|Q|-z-f} \quad , z + f \leq |Q| \tag{2.4}$$

$$MPQRsim^J(Q,r) = \frac{f}{|r|+|Q|-f} \quad , z + f \leq |Q| \tag{2.5}$$

Where f is the number of concepts lists that review is "seen" in them and z, the number of concepts lists that review r has a chance to be found there is defined as follows:

$$z = |\{y_i | y_i \geq r.rating, y_i \in Y, y_i \neq null\}|$$

where Y is the set of current ratings of query concept lists as defined in section 2.4.1. A *null* value as a current rating means that the list has been read completely. Based on Equations 2.4 and 2.5, since $z = 0$ for the seen reviews, the minimum and the maximum possible similarities are equal to the exact similarity of the review. For unseen reviews, where $f = 0$, the minimum possible similarity of u, an unseen review, and Q is 0.

The maximum possible similarity of an unseen review depends on its rating, as explained by the example in Figure 2.7. The leftmost point $(1/4, 7/9)$ denotes that if the rating of an unseen review is equal to 1/4, then its maximum similarity would be 7/9, because at best it may contain seven of the nine query concepts, as the first two lists have been completely read (null) and the last six lists have current ratings greater than 1/4. Similarly, the other points denote possible rating and maximum possible similarity combinations, $[rating, similarity]$.

Formally, we define U, the set of possible $[rating, similarity]$ pairs for $QRSim^J$ as follows:

$$U = \{[y_l, \frac{|l|y_o \geq y_l, y_o \in Y|}{|Q|}]|y_l \in Y\}$$

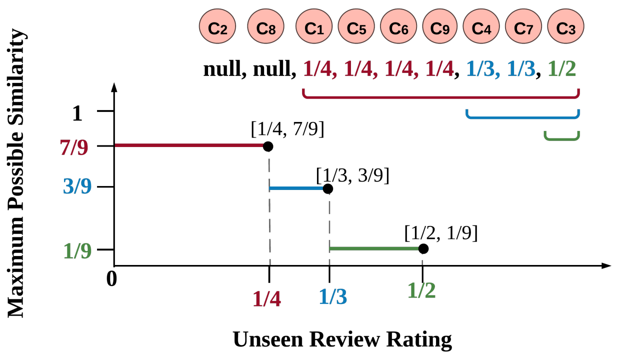


Figure 2.7: An example of possible $[rating, similarity]$ pairs of an unseen review shown by black points on the diagram for a query consisting of nine query concepts. The lists that are completely read have "null" value for their current rating (i.e rating of last seen review in the list).

Note that if an unseen review does not include any query concept then its similarity is equal to 0 and it does not affect the item's score.

Computing Item Score Bounds: Now that we have defined the space of possible $[rating, simila\text{-}rity]$ values for unseen reviews, we need to decide which combination of them would maximize the score of a partially seen item. For that, we use Theorem 2.4.1, while Theorem 2.4.2 is used to compute the minimum score of a partially seen item.

Theorem 2.4.1 *Let I be an item with h unseen reviews and a set R of seen reviews. If there is a set U of candidate $[s, w]$ (s stands for the rating and w stands for similarity) pairs, in order to maximize the score of I, we can reuse a single pair in U for all h unseen reviews.*

Proof. We define:

$$V = \sum_{i \in R} w_i \times s_i \qquad \textit{and} \qquad W = \sum_{i \in R} w_i$$

Let's define $XPScore(Q, I)$ as the maximum possible score of item I with h unseen reviews:

$$\begin{aligned} XPScore(Q, I) &= \frac{V + \sum\limits_{i=1:h} w_i \times s_i}{W + \sum\limits_{i=1:h} w_i} \\ &= \frac{\sum\limits_{i=1:h} (w_i s_i + V/h)}{\sum\limits_{i=1:h} (w_i + W/h)} \end{aligned}$$

Let's define:

$$\begin{aligned} A_i &= h w_i s_i + V \\ B_i &= h w_i + W \\ XPScore(Q, I) &= \frac{\sum\limits_{i=1:h} A_i}{\sum\limits_{i=1:h} B_i} \end{aligned}$$

There is only a single pair for all h reviews that maximizes $\frac{\sum_{i=1:h} A_i}{\sum_{i=1:h} B_i}$ and is the pair with the maximum $\frac{A_i}{B_i}$. As a result, the maximum possible score of the item is equal to:

$$XPScore(Q, I) = \max_u \{\frac{V + h \cdot w_u \cdot s_u}{W + h \cdot w_u}\}$$

■

Hence, only one pair in Figure 2.7 must be considered when computing the maximum score of a partially seen item. Similarly, we have a theorem for the minimum score below.

Theorem 2.4.2 *Let I be an item with h unseen reviews and a set of R seen reviews. To minimize the score of I, we use pair* $[0, \max w_j]$ *for all h unseen reviews.*

Proof. The fact that all unseen reviews should have the same $[rating, similarity]$ assignment to minimize an item's score is proved similar to Theorem 2.4.1. Therefore, the pair that minimizes $\frac{A_u}{B_u}$ also minimize the item's score. The pair that minimizes the item's score has minimum possible rating equal to 0 and maximum possible similarity equal to the maximum of the maximum possible similarities of possible $[rating, similarity]$ pairs in U.

■

Note that if an unseen review does not include any query concept then its similarity is equal to 0 and it does not affect the item's score.

Next, we describe how we compute the minimum and the maximum possible scores of partially seen items based on the weighted mean of review ratings for items as defined in Equation 2.1 using the minimum and the maximum possible similarities of the partially seen and unseen reviews.

It is not practical to check all combinations of reviews similarities in order to compute the maximum score. Instead, we invoke Theorem 2.4.3 to compute the maximum possible score of a partially seen item with an optimal greedy solution, which has complexity linear on the number of item reviews.

Theorem 2.4.3 *Given an item I with a set R of n partially seen reviews, where each review r has a known $r.rating$, minimum similarity $MPQRSim(Q, r)$, and maximum similarity $XPQRSim(Q, r)$, there is a review $r_i \in R$ such that the score of I is maximized if we assign*

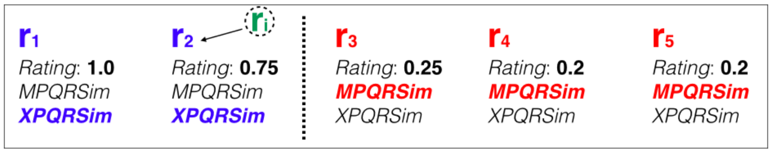


Figure 2.8: An example case of Theorem 2.4.3. Here we show a list of an item's reviews sorted by review rating in descending order. The maximum possible score of the item is achieved when the first two reviews are weighted by the maximum possible similarity, and the last three reviews are weighted by the minimum possible similarity.

similarity $XPQRSim(Q, r)$ to all reviews with rating greater than or equal to $r_i.rating$ and $MPQRSim(Q, r)$ to the rest.

Proof. For presentation simplicity, we view seen reviews as partially seen reviews for which the minimum and the maximum possible similarities to the query are equal to the exact similarity.

We show an example scenario for Theorem 2.4.3 in Figure 2.8. This theorem states that in computing the maximum possible score of the item: (1) Only the minimum and the maximum possible similarities should be considered and not intermediate similarity values. (2) The choice of maximum vs. minimum similarity for each review is not independent of the other reviews, but if a review r_1 has a higher rating than review r_2, then if the minimum similarity is used for r_2, the minimum similarity should be used for r_1 as well. This is proved by contradiction. ■

Figure 2.8 shows a list of five reviews sorted by rating, and r_2 is the divider review (review r_i as stated in Theorem 2.4.3). As stated before, Theorem 2.4.3 helps in finding the maximum possible score of a partially seen item with complexity O(n) instead of considering all combinations of minimum and maximum possible similarities of the reviews. Note that Theorem 2.4.3 assumes that all the reviews of the item are partially seen and their rating is known. In the presence of unseen reviews with unknown ratings, we compute a locally maximum item score for every possible pair of $[rating, similarity]$ of unseen reviews. Therefore, the maximum possible score of the item is the maximum of all local maximum scores (for a $[rating, similarity]$ pair).

Algorithm 1 shows the computation of maximum possible score of a partially seen item with R, a set of partially seen reviews, and u, a candidate pair of $[rating, similarity]$ for h unseen reviews. Every review in R has a rating, a minimum, and a maximum possible similarity to Q. In this algorithm, $PR(rating, minSim, maxSim)$ generates a new candidate unseen review representing h unseen reviews. Since all unseen reviews have identical $[rating, similarity]$ pairs, the unseen reviews are trivially equivalent to a single review with candidate similarity multiplied by h. Algorithm 2 computes the maximum possible score of the item for different possible pairs of rating and similarity in set U, and return the maximum of these locally maximum possible scores.

In order to calculate the minimum possible score of a partially seen item, we use Theorem 2.4.4. This theorem is proved similar to Theorem 2.4.3. As a result we avoid to include it in the chapter.

Theorem 2.4.4 *Given an item I with R, a set of n partially seen reviews and each review r, with a known $r.rating$, $MPQRSim(Q, r)$, and $XPQRSim(Q, r)$, there is a review $r_i \in R$ such that the score of item is minimized if we assign $MPQRSim(Q, r)$ to all reviews with rating greater than or equal to $r_i.rating$, and $XPQRSim(Q, r)$ to the rest.*

Algorithm 1 NRA-IRanker, Maximum possible score of partially seen item without unseen reviews.

```
1: procedure XPScore($R$,$u$,$h$)
   R: partially seen reviews of the item,
   u: candidate $[rating, similarity]$ pair,
   h: number of unseen reviews
2:     $R.add(PR(u.rating, 0, u.similarity \times h))$
3:     $SumWeights = \sum_{r \in R} r.minSim$
4:     $WeightedSum = \sum_{r \in R} r.rating \times r.minSim$
5:     $Score = WeightedSum/SumWeights$
6:     $R = sort\_by\_rating(R)$
7:     FOR $(i = 1; i <= R.size(); i++)$ DO
8:         $SumWeights+ = (r.maxSim - r.minSim)$
9:         $WeightedSum+ = r.rating \times (r.maxSim - r.minSim)$
10:        IF $Score < (WeightedSum/SumWeights)$ THEN
11:            $Score = (WeightedSum/SumWeights)$
12:        else
13:            return $Score$
14:        end IF
15:    end FOR
16: end procedure
```

While processing the reviews in query concept lists, the minimum and the maximum possible similarities of the partially seen reviews as well as the minimum and the maximum possible scores of the partially seen items keeps getting updated. The algorithm terminates when there are k seen or partially seen items with a minimum possible score that is greater than or equal to 1) the maximum possible score of the rest of seen or partially seen items, and 2) y_{max}, the maximum possible score of the unseen items.

Path-Length Similarity

If similarity of the query and review is based on the Path-Length measure, as with the Path-Length based RA-IRanker, we need to process the lists of query concepts and their neighbors in the concept graph. Here, we compute the set of possible $[ratings, similarity]$

Algorithm 2 NRA-IRanker, Maximum possible score of partially seen item with unseen reviews.

```
1: procedure XPScoreAll($R$,$U$,$h$)
   R: partially seen reviews of the item,
   U: Set of possible $[rating, similarity]$ pairs,
   h: number of unseen reviews
2:     $MaxScore = 0$
3:     FOR $u \in U$ DO
4:         $score = XPScore(R, u, h)$
5:         IF $MaxScore < score$ THEN
6:             $MaxScore = score$
7:         end IF
8:     end FOR
9:     return $MaxScore$
10: end procedure
```

pairs for unseen reviews in a different way than Jaccard similarity. Since every query concept is extended to a list of neighbor concepts, we have to process all concepts and their neighbors that are reached via a shortest path from the query concepts that is less than the distance threshold T, as seen in Figure 2.4.

To compute set U, possible $[rating, similarity]$ pairs for unseen reviews, we first select a possible rating from the set of current ratings y_i of both the query concepts and their neighboring concepts (with distance less than threshold T). Then, in order to compute the maximum possible similarity for the selected rating, we find the concepts that have shortest distance from every query concept whose current rating is greater than or equal to the selected rating. By summing up the similarities of these concepts, we compute the maximum possible similarity for the selected rating.

In addition, the computation of minimum and the maximum possible similarities of partially seen reviews is different than the Jaccard-based version of the algorithm. The rating of a review becomes known when it is accessed for the first time in one of the concept lists. Therefore, to calculate the maximum possible similarity of a partially seen review, for each query concept, we find the concept with the shortest distance (maximum concept similarity) to the query concept; the concept we find either has the review already processed in its list, or its current rating is greater than or equal to the rating of the review. The maximum possible similarity is the sum of the concept similarities as defined in Equation 2.3. For the minimum similarity of a partially seen review, we only consider query concepts for which the review has been seen in their lists, or in one of their neighboring concept lists. If the review is seen in multiple lists of neighboring concepts of a query concept, then the concept similarity of the nearest concept is used. On the other hand, for the query concepts for which the review is not seen in neither their list nor the list of their neighboring concepts, we assume the concept similarity is 0. As defined in Equation 2.3, the minimum possible similarity of the partially seen review is the sum of minimum possible similarities per query concept.

2.4.3 Incorporating Additional Item and Review Features

As we briefly mentioned in Section 2.3, one may want to also incorporate other features into the ranking function of Equation 2.1. In this section we show how the top-k algorithms are modified to accommodate such features; specifically, we consider two types of features: *item features*, such as number of reviews or price, and *review features*, such as review helpfulness or date.

Equation 2.6 is a generalization of Equation 2.1:

$$score(Q, I) = g(scoreF(Q, I), G_1, G_2, \cdots) \tag{2.6}$$

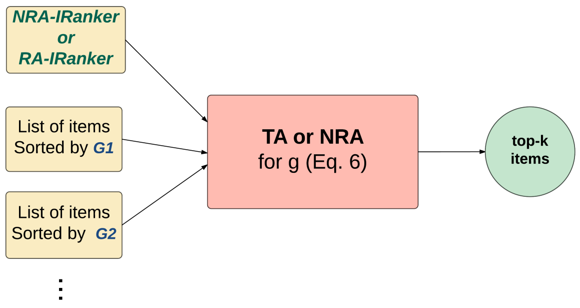


Figure 2.9: The scheme to combine multiple ranking features of the items.

where scoreF(Q,I) incorporates all review weighing features such as review helpfulness, and G_i are the item features such as price and popularity. Specifically,

$$scoreF(Q, I) = \frac{\sum_{r \in I} QRSimF(Q, r) \times r.rating}{\sum_{r \in I} QRSimF(Q, r)} \tag{2.7}$$

Note that $QRSimF(Q, r)$ is a generalization of $QRSim(Q, r)$, which includes all review weighing features, in addition to the review similarity $QRSim(Q, r)$, as follows:

$$QRSimF(Q, r) = f(QRSim(Q, r.concepts), F_1, F_2, \cdots), \tag{2.8}$$

where F_is are review weighing features. For our top-k algorithms to work, function f should be monotonic on $QRSim$ and F_i's, in order to be able to compute lower and upper bounds for $QRSimF$, given the bounds on $QRSim$ and F_i's. Note that $QRSim$ is simply replaced by $QRSimF$ in the top-k algorithms.

Further, to be able to execute a top-k algorithm to compute the final item scores from Equation 2.6, aggregation function g must be monotonic on its arguments. Then, we can apply an algorithm like NRA on top of the ranked item features lists, as shown in

Figure 2.9. Note that this is a two-level top-k algorithm, as the arguments of g may be computed in a pipelined manner as well.

In particular, the first argument of g, $scoreF(Q, I)$ is executed by the RA-IRanker or NRA-IRanker algorithms described above. A GetNext() method is implemented for each argument of g so that the two levels are connected in a pipelined manner.

2.5 Evaluation Results

In this section we evaluate the run-time performance and the quality of the search result of our proposed method.

2.5.1 Quantitative Analysis

We start by describing experimental settings followed by the experimental results. Then, we compare the run-time performance of our proposed algorithms against two baselines and discuss the results in detail.

Experimental Settings

Baseline algorithms: We use two algorithms as baselines for the two different similarity functions. The baselines for Jaccard similarity and Path-Length similarity are called Base-J and Base-P respectively.

Base-J, uses the same *Concepts* index as used in NRA-IRanker algorithms to get all reviews that contain at least one of the query concepts. We need the number of concepts for each review in concept index, in order to avoid making an extra access to the review index for each review. Then, we group the reviews of each item and compute the weighted average based on Equation 2.2.

In Base-P, for each query concept, we get all the query concepts and their semantically related concepts within a distance up to a threshold T. For each query concept, we read all the reviews in the lists of its related concepts and add them to a new list which is kept in memory. For each review in this list, we maintain its distance to the query concept. If a review is added to a query concept list several times with different distances, we only consider the one with the minimum distance. Then, we group query concepts for each review and reviews for each item. Next, for each review, we compute the review similarity based on equation 2.3. Finally, for each item, we compute the IRanker item score using review rating and similarities. Base-P requires the same concept index as in RA-IRanker where each review in the review list of a concept contains review-id, item-id and a rating.

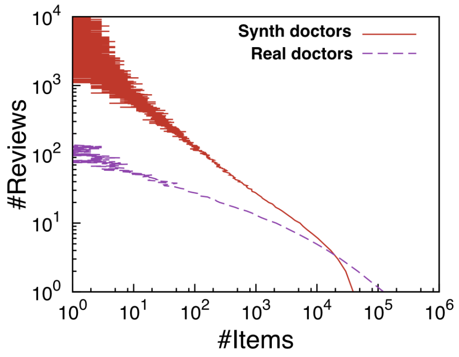


Figure 2.10: Distribution of reviews for real and synthetic doctor reviews datasets.

Datasets: In order to evaluate our algorithms, we use healthcare provider reviews that are collected by crawling vitals.com and ucomparehealthcare.com websites. We merge items and reviews of these two websites into a single dataset and in the following we will

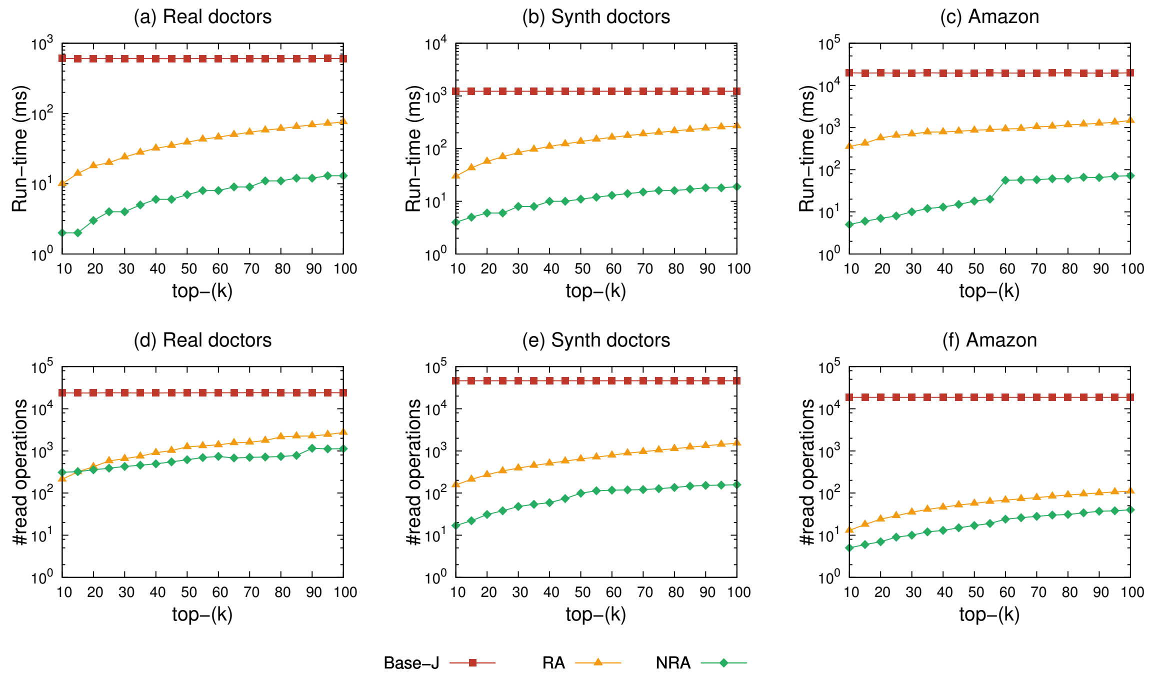


Figure 2.11: (a),(b),(c). Run-time performance using Jaccard similarity function with queries of size $|Q| = 3$ for real doctors, synthetic doctors and Amazon datasets respectively. (d),(e),(f). Number of random read accesses to indexes used by each algorithm for the experiments in (a), (b) and (c)

refer to the combined dataset as the “Real doctors” dataset. We extract medical concepts from the textual content of the reviews using the MetaMap tool [9]. Furthermore, we built a synthetic dataset to increase the number of reviews of each item (medical provider) from “Real doctors” dataset in a principled way. The number of additional reviews is generated by a Zipf random number generator between 0 and 10^3. Figure 2.10 shows the distribution of the number of reviews across real and synthetic doctors datasets. For each additional synthetic review of an item, we calculate the number of concepts based on a mixture distribution of the number of concepts across the real dataset and the distribution of the number of concepts for that item. The mixture distribution of global and local distributions is as follows.

$$P(x) = \alpha \times P_G(x) + (1 - \alpha) \times P_L(x)$$

Where P_G is the probability according to the global distribution and P_L is the probability according to the local distribution of an item. In other words, the probability of selecting a concept for a synthetic review of a doctor is a mixture of probability of its existence across the whole database and within the reviews of that doctor. In our experimental configuration, we set α to be equal to 0.5.

We follow a similar approach to compute a rating for each synthetic review by combining the global rating distribution in the real dataset and the rating distribution in the reviews of the item. The rating in the real dataset is a number from a set of 12 discrete numbers between 0 and 1. In the synthetic dataset, the ratings are more granular and they are in a set of 101 discrete numbers between 0 and 1.

In addition to doctor reviews, we use Amazon product reviews dataset [82]. This dataset is collected from Amazon.com and is a fairly comprehensive collection of English language product data in a wide variety of categories such as books, clothing and movies. Each product is provided with its reviews including textual reviews and star ratings. Table 4.1 summarizes the characteristics of the datasets used in this study. In our experiments, we treat textual Amazon product reviews as a bag of keywords and only perform the ex-

Table 2.4: Datasets' characteristics.

Dataset	#Items	#Reviews	Reviews per Item	Concepts per Review
Real doctors	248580	726996	Min: 1 Max: 249 Mean: 2.9 Median: 5	Min: 0 Max: 121 Mean: 3.29 Median: 7
Synth doctors	248580	38802836	Min: 1 Max: 10000 Mean: 156 Median: 3209	Min: 0 Max:121 Mean: 3.59 Median: 4
Amazon	9743974	82037337	Min: 1 Max: 25260 Mean: 8.4 Median: 2	Min: 1 Max:1719 Mean: 26.65 Median: 15

Table 2.5: Total disk space consumed by Cassandra indexes.

	Real doctors	Synth doctors	Amazon
Baseline Jaccard	44M	1.4G	33.17
Baseline Path-Length	30M	0.97G	–
RA-IRanker	59M	1.75G	37.46
NRA-IRanker	44M	1.4G	33.17

periments using Jaccard similarity of the query and reviews. One can simply use general ontologies in order to use the Path-Length similarity in such general domains.

Database setup: The indexes are stored in a Cassandra data store [74] on a single node. We chose a NoSQL store, because we only need an efficient mechanism to look up index lists and no SQL capabilities. Other key-value stores can also be used. Table 2.5 summarizes the amount of disk space that each algorithm consumes for storing necessary indexes in Cassandra. We partition all the index lists into 8 KB chunks to allow for incremental reading of the lists. That is, each index list is a row in Cassandra, and each chunk in the list is a column in this row.

Query workload generation: For the doctor reviews datasets, we generate queries by choosing a set of query concepts, where each concept is selected with probability proportional to its frequency in the reviews. For every experiment, we present results averaged over 100 queries. The queries for the Amazon dataset are selected from the queries suggested by the website in different categories. For example, ”spray face child sun screen” is a search query suggested by amazon.com.

Parameters: We evaluate the performance of our algorithms by varying the number of top-k items, the query size and the distance threshold T for the Path-Length similarity measure.

Experimental Results

In this section, we compare the run-time performance of our algorithms against the baselines for both Jaccard and Path-Length similarity measures using several experiments on multiple real and synthetic datasets.

Jaccard Similarity: Figure 2.11 shows the experimental run-time for real and synthetic doctor reviews and Amazon datasets for different values of k, the number of top items, given queries of size 3. Figures 2.11.(a), 2.11.(b) and 2.11.(c) show the average retrieval time of different algorithms for different datasets. In Figures 2.11.(d), 2.11.(e) and 2.11.(f) we demonstrate the total number of read operations made to necessary indexes stored in Cassandra by various algorithms.

The results of our experiment show that NRA-IRanker returns top-k items faster than other algorithms; on an average NRA-IRanker is 5.5x faster than the RA-IRanker for the real doctors dataset, 11.5x faster for the synthetic doctors dataset and 43x faster for Amazon dataset. Base-J has the worst performance amongst other method, specially in Amazon dataset.

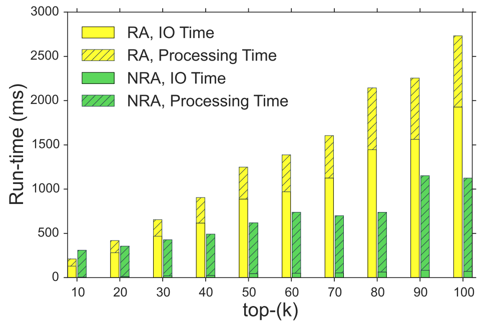


Figure 2.12: Cassandra read access time and processing time for Jaccard-based RA-IRanker and NRA-IRanker algorithms using synthetic dataset.

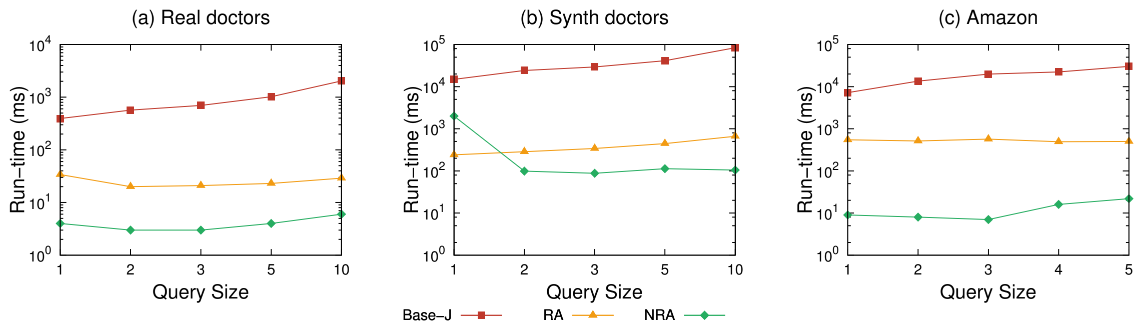


Figure 2.13: Query time using Jaccard similarity measure to find top-20 items.

We demonstrate the IO and processing times separately for RA-IRanker and NRA-IRanker in Figure 2.12. Due to the space limitations, we only report the results for the synthetic dataset since the same pattern is observed in experiments with other datasets. As it is shown in Figure 2.12, processing takes more time than IO in NRA-IRanker because of computing and maintaining the minimum and maximum possible score of partially seen items as well as the rating of partially seen reviews. In RA-IRanker, the increase in the number of reviews per item leads to increase in processing time while computing the score of each item. As a result, with the increase in the number of top items, the processing time increases with a faster pace than IO time.

We also show the query response time by changing the query size in Figure 2.13 for real and synthetic doctors datasets as well as Amazon dataset. As shown in this figure, Query time of the baseline increases with increase in query size because it reads more review lists per query concept and process more data. Amazon product queries that we use have 5 terms at most. That explains the different range of query size for Amazon dataset. In the case of very small query size, more reviews had to be read from one list to terminate. That costs more processing time for NRA-IRanker. As a result, the run-time for small query size is higher.

Path-Length Similarity: Similarly as for Jaccard similarity measure, we compare the query response time for Path-Length similarity measure for real and synthetic doctor reviews datasets as shown in Figure 2.14. In this experiment IRanker algorithms are compared to Base-P for two different Path-Length thresholds of 2 and 3. In both cases, NRA-IRanker outperforms other algorithms significantly. Figure 2.15 shows the experiment result for different query sizes for real and synthetic doctors reviews datasets using Path-Length measure with the threshold equal to 2. Query response time increases for all the algorithms with the increase of the number of the query concepts.

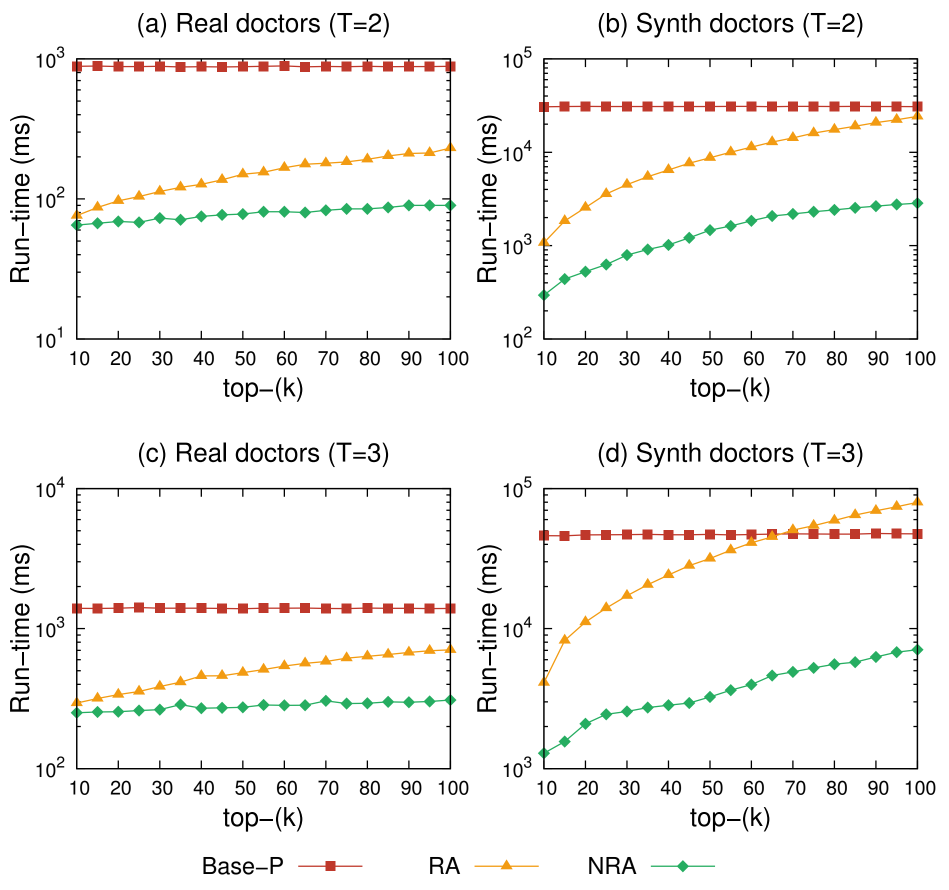


Figure 2.14: Query time using Path-Length Similarity measure for a query of size 3. (a) and (b) are based on Path-Length with threshold 2. (c) and (d) are based on Path-Length with threshold 3.

Table 2.6: An example of doctors returned by each algorithm for query ”skin cancer“. NRA-IRanker makes a better choice by weighing the non-relevant review lower while Baseline simply select the item with higher average ratings.

doctor-id	Selected by	review-id	Review Rating	Text
D1	Baselines	R1_1	0.3	Misdiagnosed **skin cancer** as dermatitis. despite multiple visits regarding spots for 15 months.
		R1_2	1	his practice, including the nurses, assistants, and Dr. X herself, are all wonderful
D2	NRA-IRanker	R2_1	0.2	Worse clinic I have ever been to in my entire life! Very rude and unprofessional staff!!!!!
		R2_2	1	He is extremely knowledgeable. He identified start of **skin cancer** very quickly and help me to stop progress and curing it.

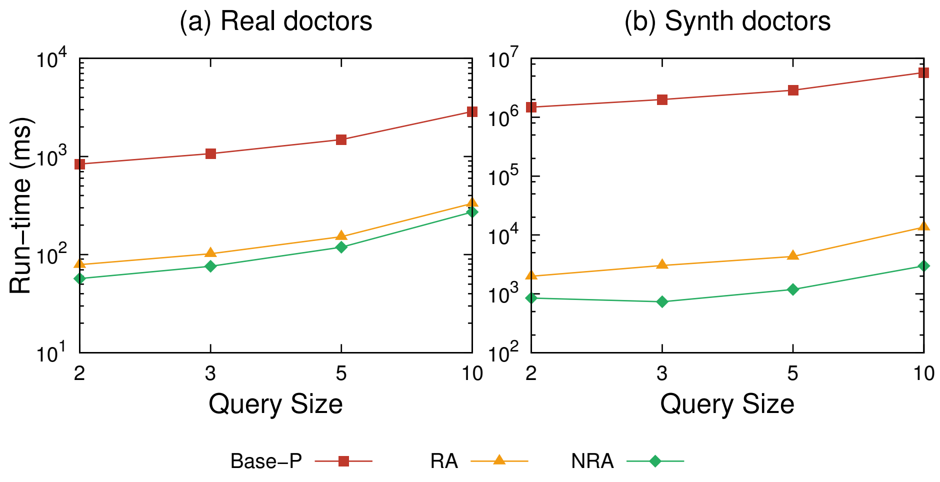


Figure 2.15: Query time using Path-Length similarity with $T = 2$ measure to get top-20 items.

2.5.2 Qualitative User Study

To measure the accuracy of our proposed methods, we conducted a user case study as follows.

We set up a user study to evaluate the quality of the search results using our proposed Jaccard and Path-Length based algorithms and two baseline methods. In particular, we want to evaluate if the relevance-based weighted average ranking formula (Equation 2.1) is effective, and if considering neighboring concepts (Path-Length distance semantics) is improving the search quality.

Baseline1: baseline1 selects all items (doctors) that include all query concepts in their reviews (e.g., the first query concept may be contained in one review and the second in another). Then, the items are ranked by the average rating of all their reviews (not only the ones matching the query).

Baseline2: baseline2 considers, in addition to the items that Baseline1 considers, items that include the direct neighbors of the query concepts in the SNOMED-CT ontology. The ranking is as in Baseline1.

Data selection and setup: We selected 50 queries, each including one or two popular medical concepts such as “acne scar”, “Rheumatoid arthritis” and “weight loss and wrinkled skin”. We randomly select 60 doctors that have at least one of the query concepts or a concept with distance one from query concepts in their reviews. For each doctor, we select three reviews, using the following method to ensure diversity in terms of rating score. The first review is randomly selected from the set of relevant reviews of the doctor. The second review is a random positive review ($rating > 0.5$) if at least 10% of doctor’s reviews are positive, and similarly, for the third review if at least 10% of doctor’s reviews are negative ($rating \leq 0.5$) we randomly select a negative review. Note that if needed, additional random reviews are selected to have three reviews.

For each query, we show the union of the top-3 items returned by NRA-IRanker with Jaccard similarity, NRA-IRanker with Path-Length similarity (distance threshold = 1), Baseline1 and Baseline2 to the users. For each query, we show three reviews for every doctor to the raters and ask them to mark the doctors that they think are good matches. We recruited 15 students as the raters, and each student was asked to complete all 50 queries. We use the users’ majority vote to decide on the relevance of each item.

Table 2.6 shows an example of two doctors returned for ”skin cancer“ in this case study (only two reviews are shown here). Doctor D1 is ranked higher by the baseline algorithms (scored by average rating of R1_1 and R1_2) while Doctor D2 is selected by our

Table 2.7: Search result quality for IRanker Jaccard, NRA-IRanker Path-Length, Baseline1 and Baseline2.

	AP@1	**AP@2**	**AP@3**
Baseline1	0.29	0.3	0.31
IRanker (Jaccard)	0.33	0.41	0.42
Baseline2	0.5	0.58	0.69
IRanker (PathLength)	0.81	0.89	0.94

proposed methods (scored only based on R2_2). The second column is not shown to the study users.

Results: The mean average precision (MAP) at various positions – @1, @2 and @3 – for the four ranking methods is reported in Table 2.7. Based on the raters judgement, MAP for NRA-IRanker using Path-Length measure is significantly higher than the other methods, as it leverages both the ontological relationships (as does Baseline2) and the relevance-based weighing (as does the Jaccard variant of NRA-IRanker).

2.5.3 Discussion

Based on our experiments, we observe that the performance of NRA-IRanker in all cases except very small query size is significantly better than the other algorithms. In general, the query response time using Jaccard similarity measure is much less than the Path-Length similarity measure due to the simpler similarity computation as well as the smaller number of concepts to consider while looking for top-k items relevant to a query.

In terms of disk space, as shown in Table 2.5, for Jaccard similarity, Base-J and NRA-IRanker have the same requirements, which makes NRA-IRanker even more desirable given its far superior execution time. For Path-Length, Base-P needs 33% less space than NRA-IRanker. All in all, NRA-IRanker has the next smaller space requirement.

In addition, we showed through the user study that (a) a relevance-based weighing of reviews and (b) ontological query expansion both improve the quality of ranking results.

2.6 Conclusion

In this chapter, we proposed a solution to efficiently compute the top-k reviewed items based on a query-specific scoring function. We used semantic ontologies to improve the quality of search results. Using experiments on a real world dataset and a large augmented synthetic dataset, we showed that the performance of our algorithm is significantly better than baseline methods. Amongst the three proposed solutions, NRA-IRanker has the best performance in most settings and also needs the least disk space to store necessary indexes.

A possible future direction is to study to take into account the item's review rating distribution in computing the item score while weighting the review ratings. Another improvement may result from considering a finer rating granularity of concepts within a review. That is, instead of having the same rating score for all concepts of a review, one could employ NLP methods to compute a rating for each concept. This requires detecting context bounds in the text, analyzing the sentiment, handling negations, and more.

Chapter 3

Define and Extract Domain-Specific Ontological Concepts

The Multiple Listing Service, commonly known as the MLS, is the singularly most important database where real estate agents and brokers list real estate properties for sale. It is common that agents include textual comments pertinent to the property. Although the information content of comments varies, it is usually expressed in good faith and in many cases is helpful in shedding light on the overall condition and the value of the property. Therefore, it seems reasonable that semantic text analysis would be useful to evaluate properties, or aspects thereof. As far as we're aware of, no methodology to effectively extract insight from the MLS textual portion exists. In this chapter we demonstrate how textual descriptions may be exploited for property ranking. The proposed methodology, which combines supervised and unsupervised methods, identifies domain-specific concepts and combines their contributions to assign a score to a listing. We evaluate the proposed methods using both human evaluators and data-driven evaluation metrics on real datasets (complied from actual listings), and compare them to baseline approaches.

3.1 Introduction

Real estate agents and other real estate professionals have to sift through hundreds or thousands of listings per day to locate the ones that they should focus on to satisfy their clients, for rent, sale or investment purposes. For example, at HomeUnion [2], expert real estate investment consultants have to carefully read through hundreds of listings per day to pick the most investment-worthy ones. In addition to structured attributes like year-built and number of bedrooms, agents have to read through listing's description, typically a few paragraphs-long as well as related comments from other agents. This is clearly time consuming, translating into significant labor costs.

Evidently, the ranking the listings solely based on the structured attributes is sub-optimal as many significant pieces of information are only reflected in the text description; this includes things like remodeling information (new granite counter-top), financial conditions (short sale, foreclosure), etc. Conversely, for one reason or another, an agent will disclose that the home has "foundational issue", clearly a negative home attribute. To our best knowledge, current property valuation methods do not utilize this kind of information for ranking properties.

In this chapter, we propose a novel methodology to assign a “goodness” score based on the textual description of a listing. This score can then be combined with other structured attributes to generate an overall goodness score for a property whereby enabling the agent to focus on other, perhaps more pertinent home characteristics.

Specifically, we first build a collection of real-estate-specific concept, a phrase that describes a real-world entity, such as “granite counter top,” to use for annotation. Each concept in this collection is manually labeled with a numeric goodness value by experts. For example, “sell as-is” is a negative concept for ’turn-key’ investors not wishing to further invest to enhancing the condition of a property (like replacing old carpets.) To that end we’ve created a continuous vector representations for vocabulary words by analyzing a corpus of real estate descriptions in order to annotate the property’s textual information with scored concepts.

Existing real estate lexicons include a limited real estate vocabulary, i.e., concepts, as they are mainly designed for structured data aggregation rather than text analysis [5]. Additionally, these lexicons aren’t up-to-date with the current real estate market trends. For example “keyless entry” concept is a newer trend in housing market. As a result it’s important to have a method to build a comprehensive concept collection while keeping it up-to-date.

Considering all vocabulary words and n-grams (a phrase of n consecutive words) extracted from a corpus of domain-specific data is an expensive manual labeling task, involves assigning goodness scores to all of them. Instead, we’ve identified the most useful n-grams by using a measure based on their frequency, their ’chances’ to appear together.

We then filter the n-grams by removing the non frequent words and phrases with low mutual information. We group similar words and phrases together so that domain experts label similar groups of phrases together as a concept and assign a goodness score to it.

Even if a comprehensive scored lexicon of concepts were available, it will still remain a challenge to identify approximate matches for these concepts, e.g., "kitchen countertop" should match "kitchen counter." In this chapter we introduce an unsupervised method for text annotation using word2vec, a neural network modeling framework.

This chapter has the following contributions:

- We (intelligently) select a relatively small set of candidate phrases out of a large corpus of real estate text descriptions for domain experts to label as domain concepts, in Section 3.3.1.
- We extract exact and approximate scored real estate concepts from real estate description text and assign a score to the text using word2vec generated word vectors and matching techniques, in Section 3.3.2.
- We evaluate our description scores computed by our algorithms with both human annotators' judgements, in Section 3.4.1, and a data-driven approach, in Section 3.4.3.

Related work is presented in Section 3.2 and we conclude in Section 3.5.

3.2 Related Work

Real Estate Appraisal: There has been research on estimating real estate appraisal, the process of valuing the propertys market value. These studies focus on feature design

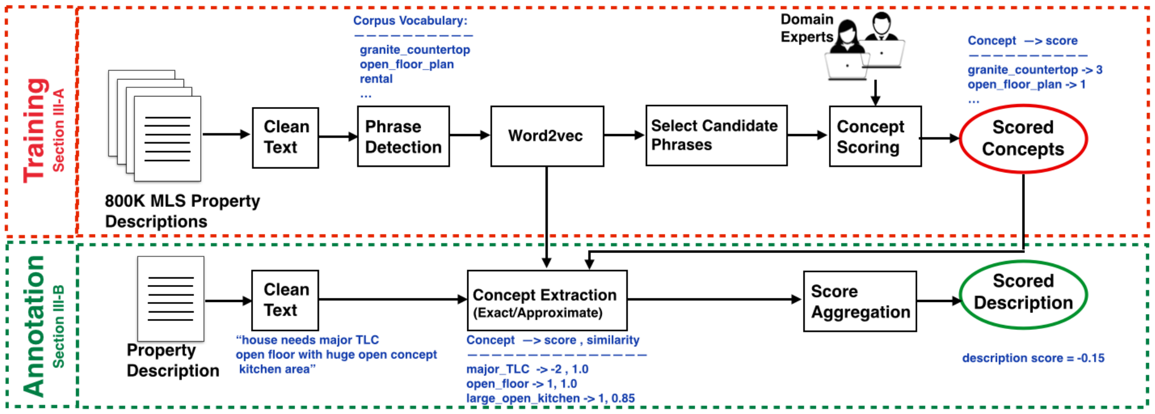


Figure 3.1: Text annotation architecture. Example texts in the diagram are colored blue.

and price tracking by comparing similar properties or change of price over time. [47] [49] use a learn-to-rank model to predict the ranking (in terms of its potential investment value) of a residential real estate based on features extracted from disparate datasets, such as taxi trajectories, road networks, and online social media ratings. Other studies work on price-rate ratio and price-income ratio for evaluating property values [70]. Some studies rely on financial time series analysis by analysing the trend, periodicity and volatility of house prices [89]. More traditional works are based on repeat sales methods that construct a predefined price index based on properties sold more than once during the given period [105]. The characteristic based methods assume the price of a property merely depends on its characteristics and location [108]. Downie et al. [40] studied the automated valuation models which aggregate and analyze physical characteristics and sales prices of comparable properties to provide property valuations.

More recent works [69], [28] apply general additive mode, support vector machine regression, multilayer perceptron, ranking and clustering ensemble method to computational house valuation. Another study focus on exploiting the mutual enhancement between ranking and clustering to model geographic utility, popularity and influence of latent business area for estimating estate value [49]. They identify and jointly capture the geographical individual, peer, and zone dependencies as an estate-specific ranking objective for enhancing prediction of estate value.

Multiple Listings Service, MLS, is a real estate listing database that provides real estate listings located all across the USA through advertised real estate by real estate agents. It contains real estate MLS listings for rent or sale by Realtors and other realty professionals that are members of local MLS Multiple Listing Service [4]. Currently there are 51 local MLS databases for different USA states. The data for each listing includes a set of structured attributes such as number of bedrooms and square feet that are used as features in real estate valuation techniques. In addition to these attributes, there are two main textual fields that contain property description and real estate agent's remarks about the listed property. These textual data often includes key points regarding the property that affects an agent's judgement while pricing the property. To our knowledge, there has not been any research done regarding extracting these features and analysing them along with other property features. In this project, we employ natural processing and text mining methods to extract key concepts from textual property descriptions and compute an additional numerical feature value describing the property value based on the textual data.

Concept Extraction Methods: Although automatic annotation of online textual resources has been studied extensively in research communities, it still remains a challenging task [28]. Several research studies focus on incorporating natural language processing techniques to do annotation tasks. Most of these studies rely on pre-annotated training examples where they tag sub-strings of the document with pre-defined annotations by identifying the distribution of vocabulary words for different annotation topics [38]. Other studies and tools rely on extra knowledge bases such as regular expression based rules [68] [80] [94] [95] [81] [15]. Another category of studies on text annotation are using machine learning methods to automatically learn the patterns for a text annotations [73]. MnM [115] is a system that retrieves patterns and rules for semantic annotation from a corpus of pre-annotated text. [54] [51] [42] are other examples of automatic rule finding approaches. two main challenges of the existing methods is their dependence on predefined rules or pre-annotated training data as well as assumption of existing concept repository.

Domain specific concepts collection (ontology): Building a collection of labels to use for text annotation has been a challenging task. Constructing ontology is a domain specific task and varies in different domain and contexts. Popescu et al. [93] introduce OPINE, a review-mining system that uses relaxation labeling to find the semantic polarity of words in the context of given product features in online reviews. It first finds features and their attribute and then uses relaxation labeling to extract their polarity in a textual review. Further research is done towards enhancing the ontology quality by refinement to better suit the target domain [100] [102] [18].

3.3 Concepts Extraction and Description Scoring

In this section we describe the steps we take towards assigning a goodness score to each description text (retrieved from MLS property listings). Our approach consists of two main steps. Figure 3.1 shows the flow of our approach towards extracting concepts from real estate description text.

First, in Section 3.3.1, we build a collection of real estate-related concepts along with their goodness score. Specifically, after cleaning a corpus of text descriptions of MLS listings and identifying key phrases, the vocabulary is further pruned based on mutual information and frequency. Then, to facilitate the definition of concepts (several phrases may map to the same concept), we cluster the phrases using word2vec vector representations. This allows human labelers to view similar phrases next to each other and mark the ones that should be part of the same concept. In addition, human experts assign a goodness score to each concept.

Next, in Section 3.3.2, given a description text, we extract the scored concepts and compute an aggregate goodness score for the input text. In this phase, a scored concept collection (product of training phase) as well as numerical vector representations are used to annotate a given text with exact or approximate matching of the scored concepts. The aggregation of the concept goodness scores that are extracted from the text generates a single score for the given description.

3.3.1 Building Scored Concepts Collection

In this section we show how, given a collection of descriptions, we generate a set of phrases that represent concepts that the real estate experts will score. The goal is to minimize the human effort, while at the same time capturing a large percentage of the concepts mention in property listings.

We use a corpus of 800K property listing descriptions and remarks retrieved from MLS database. we clean the text in corpus by lemmatization, lower-casing, tokenizing and removing the stop words (a set of 30 words that we collected) for each property. Next, we identify the phrases by merging the words that have a frequently happen together and infrequently happen in other context with a simple data-driven approach [85]. To be precise, for each consecutive pair of words (bigram) we compute a score using Equation 3.1 defined as follows:

$$score(a\ b) = \frac{(p_{ab} - min\_count)}{p_a \times p_b} \tag{3.1}$$

where p_a, p_b are frequency of terms “a”, and “b” respectively, and p_{ab} is the frequency of phrase “a b” in the corpus. min_count is a parameter to account for minimum term frequency. If a pair has a score that is greater than a threshold score (a parameter for phrase generation) then words will be attached using a hyphen and a phrase in the corpus is created. Algorithm 3 shows how word2phrase works. We run word2phrase three times in order to with decreasing min_count threshold to allow longer phrases consisting multiple words. The outputs of fist and second runs are the inputs for second and third runs respectively. Figure 3.2 shows an example of running word2phrase twice. In first round “steel” and “appliances” meet phrase score threshold and get attached as a phrase. In second round, “stainless” and “steel-appliances” are the two words that are grouped together create a phrase. After identifying phrases and merging them into one unit using word2phrase method, we use word2vec [86] tool to learn the space of continuous word and phrase representations from the preprocessed corpus. word2vec provides an efficient implementation

Algorithm 3 word2phrase algorithm. T is the score threshold for generating a phrase by attaching two words.

```
1: procedure WORD2PHRASE($text$)
2:     text = clean(text)
3:     List tokens = tokenize(text)
4:     FOR ($i = 1; i <= tokens.size(); i++$) DO
5:         IF $Score(tokens[i-1], tokens[i]) > T$ THEN
6:             new-phrase = $tokens[i-1]$ + "_" + $tokens[i]$
7:         end IF
8:     end FOR
9: end procedure
```

Round 1: Kitchen has stainless steel_appliances.
Round 2: Kitchen has stainless_steel_appliances.

Figure 3.2: Example of word to phrase transformation. Round 1 identifies ”steel” and ”appliances” to be connected as phrase. In Round 2, ”steel_appliances” is considered as one word and it is identified as a phrase in combination with ”stainless”.

of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. Word2vec allows us to train models on a large data sets (up to hundreds of billions of words). Word2vec computes a vector representation for each word using a recurrent neural network. Figure 3.3 shows two main approaches that word2vec uses for training the model to learn word representations. The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a text window. While in CBOW (continuous bag of words), the model is trained such that it predicts a word given its context (surrounding words).

The input of the skip-gram model is a single word w_I and the output is the words in w_I's context $\{w_{O,1}, ..., w_{O,C}\}$ defined by a word window of size C. For example, consider the sentence ”I drove my car to the store”. A potential training instance could be the word ”walking-distance” as an input and the words “curb-appeal”,“propery”,“located”,“school”,

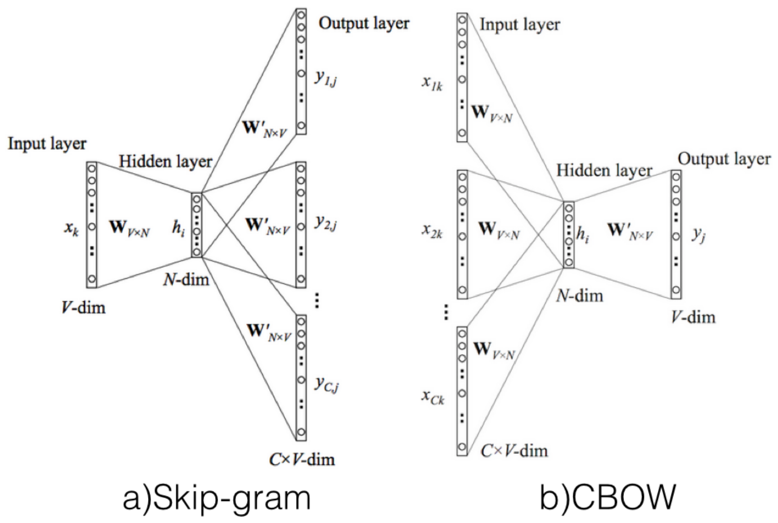


Figure 3.3: Two word2vec learning approaches. a) skip-gram model learns a model that given a word, guesses the context. b) CBOW (continuous bag of words) model learns to guess the word given a context. In these diagrams, V is the number of vocabulary words and N is the size of word vectors.

"church","shopping-mall" as outputs. All of these words are one-hot encoded, meaning they are vectors of length V (the size of the vocabulary) with a value of 1 at the index corresponding to the word and zeros in all other indexes. As we can see, Word2vec is essentially creating training examples from plain text which means that we can have a virtually unlimited number of training examples at our disposal. In CBOW version, the input layer consists of the one-hot encoded input context words $\{x_1, ..., x_C\}$ for a word window of size C and vocabulary of size V. the output layer is output word y in the training example which is also one-hot encoded. The word vector representations are the weights of the neural network and they are learned after the training cycle is complete.

After training high dimensional word vectors on a large amount of data, the resulting vectors can be used to answer very subtle semantic relationships between words [86]. More specifically, the words that are semantically related such as synonyms or the words of the same category tend to have very similar vectors because they appear in the same context. Cosine similarity measure is used to quantify the similarity of the words based on their vectors.

We use Skip-gram model in training phase to compute word vector representations. Using cosine similarity measure, we compute word clusters with KNN method to group the relevant words together. The goal of clustering is to identify groups of relevant concepts and make the labeling task easier for the domain experts by putting all the relevant concepts together. We further prune these clusters by removing non frequent words and phrases with low phrase scores (ex. mutual information or Equation 3.1) to make a smaller set for the domain experts to label the goodness score.In this project, after processing 800K property descriptions, we come up with approximately 3000 candidate concepts grouped in 500 clusters to be scored by domain experts.

Labeling the goodness of concepts: We asked two real estate experts to merge together phrases with the same meaning, in the context of homes evaluation, to form concepts, that is, “kitchen counter” and “kitchen countertop.” Further, we asked them assign a goodness score between -10 and 10 to each of the 3000 concepts. For example, since “foundation_issue” is costly for the property, it get a -8 while “mior_cosmetics” get -1 as their goodness score. The average score of multiple expert’s opinion was recorded for each concept.

3.3.2 Scoring the Property Listings

Now that we have the scored collection of concepts, in order to score a property’s textual description, we need to detect the real estate concepts in the text and compute the overall score based on their aggregated goodness scores. In the previous section, we explained how we build a collection of concepts that we use to annotate the text. An exact concept may not be exactly present in the text but a semantically similar term may be present (e.g., synonyms, acronyms, etc.). In this case, the word2vec word vectors can be used to capture the relevance of the words to the real estate concepts in our concepts collection. In this section, we describe three methods to extract the concepts from a given text.

The goal is to extract labeled concepts and assign a score to the property description based on the aggregation of the concepts' goodness scores. We propose three variations of our solution with different trade-off's between computation time and precision.

Exact concept matching (ECM)

Given a property description, we find all the scored concepts that exactly appear in the cleaned text. In this variation, we define the property score as the summation of the goodness score of the found concepts:

$$score(text) = \sum_{concept \in text} goodness(concept)$$

Nearest Concepts Matching (NCM)

In this variation, we find all the vocabulary words/phrases that exist in the input text string. Then, for each word/phrase, we find the nearest concept with the maximum cosine similarity amongst the scored concepts. If the cosine similarity is greater than a threshold parameter, then the concept will be considered as found with a weight equal to its similarity to the existing word/phrase. Similarity threshold is a parameter in our system.

The textual score of the property is computed by summing up the goodness scores of the found concepts weighted by their similarity to the vocabulary word/phrase exist in the text (in terms of cosine similarity). Formally, we define the score of textual property description as follows:

$$score(text) = \sum_{p \in text} goodness(nearest\_concept(p)) \times weight(p)$$

where p is a vocabulary word/phrase that is in the text and weight of the word/phrase is defined as follows:

$$weight(p) = cosine\_similarity(nearest\_concept(p), p)$$

Non Redundant Nearest Concepts Matching (NRNCM)

In NRNCM, we first extract the concepts using nearest concept matching. Then, in a greedy way, we iterate over the set of matched concepts and compare each concept with the rest of concepts. If two concepts of the matched set have a similarity greater than the similarity threshold then we remove the concept with the smaller similarity weight from the set of concepts and continue. The purpose of this pruning is to avoid scoring the property multiple time for the same concept.

3.4 Evaluation Results

3.4.1 Experimental Setup and Measures

Data Set: Our corpus includes 800K property descriptions and agent remarks that are inserted by real estate agents for property listings (fetched from MLS database). These descriptions are retrieved from the property listings fetched from the MLS database. After pre-processing the corpus to clean the text we get a vocabulary of size 45K and 34485K words in total.

Experimental Setup: We set min_count from Equation 3.1 to be equal to 100. For the phrase score threshold, we chose different values during our experiments and show how the results changes. Finally, We evaluate our scoring algorithms using both CBOW and skip-gram models to train word vectors.

3.4.2 Evaluation Using Human Labeled Text

We randomly selected a set of 100 property listings. The real estate experts assign a score of -1, 0 or +1 to each description if it is a negative, neutral or positive description. We then sort the descriptions with scores that are computed using three variations of our proposed solution.

We measure the correlation of the rankings using normalized Kendall-Tau [65] measure. Kendal_Tau measures the ranking agreement of two different ranking schemes by comparing each pair of rankings for a set of ranked objects. Assume that o_1 and o_2 are two objects and their ranks by ranking method R^1 has $R^1_{o_1} < R^1_{o_2}$ relation meaning that o_2 is ranked higher than o_1. If the ranks by R^2 agrees with R^1, such that $R^2_{o_1} < R^2_{o_2}$, then o_1

Table 3.1: Example of ranking of 5 property listing text with human annotator (ground truth) and automatic machine annotator. Optimal ranking in last column is based on assumption that all ranks are distinct and they are in perfect correlation with human ranking. In case of tie in scores, an equal ranking is assigned to all of them.

Human Score	**Human Ranking**	**Machine Score**	**Machine Ranking**	**Machine Optimal Ranking**
+1	2	23.0	3	1
+1	2	45.3	2	2
+1	3	51.0	1	3
0	3	3.0	4	4
-1	4	-10.0	5	5

and o_2 are concordant pairs. On the other hand, if $R^2_{o_1} > R^2_{o_2}$, o_1 and o_2 are a discordant pair. based on this definition, Kendall_Tau is defined as following.

$$Kendall\_Tau = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$$

where n_c and n_d are the number of concordant and discordant pairs respectively. n_0 is the number of possible pairs. n_1 and n_2 are number of pairs with a tie using ranking methods R^1 and R^2 respectively.

We normalize rank correlation measures by their value for perfect ranking and we call it optimal machine ranking. We assume in optimal machine ranking all the objects have distinct rank and are ordered such that the ranking completely agrees with human ranking. Table 3.1 shows an example of ranking using human judgement and our machine score based ranking. Based on these definitions, figures 3.5 and 3.4 show the Kendall-tau rank correlation of three proposed algorithms based on CBOW and Skip-gram strategies for training word vectors respectively. We compare our algorithms score based rankings to domain expert judgements. We evaluate the rank correlation for different cosine similarity thresholds between 0.5 and 0.9 inclusive. As shown in figures 3.5 and 3.4, the ECM algorithm has a constant correlation for different similarity thresholds since it is independent of this parameter. The correlation of NCM is similar or better than NRCM for all similarity thresholds. This observation implies that redundant mentioning of similar concepts should not be ignored. Another observation is that maximum correlation for NCM and NRNCM is happening in lower threshold while using CBOW based word2vec learning comparing to Skip-gram based learning. The peak correlation also is slighty lower using CBOW. The reason is that using CBOW, similar words' vectors have higher distance than Skip-gram which causes more concepts to be filtered using higher thresholds.

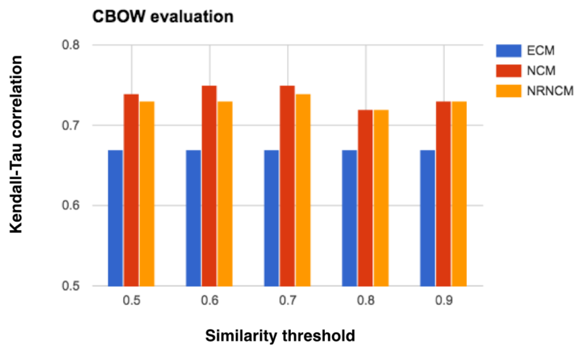


Figure 3.4: Kendall_Tau evaluation of ranking algorithms for different cosine similarity thresholds. Word vectors are trained using CBOW model.

3.4.3 Evaluation Using Price Variation

We did a more extensive evaluation by counting the properties that are expected to have similar listing price but they don't. We assume the reason should be explained in the description text and agent remarks. If two properties are similar in terms of key features including location, year built, lot-size and square feet, we expect them to have the

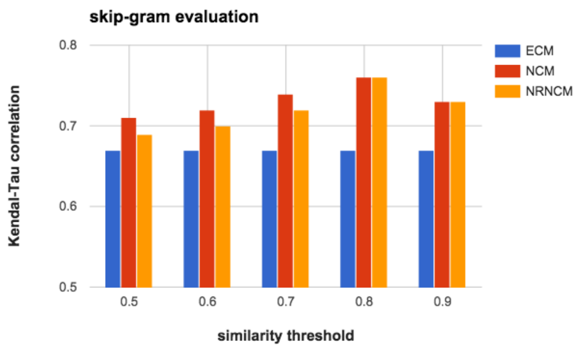


Figure 3.5: Kendall_Tau evaluation of ranking algorithms for different cosine similarity thresholds. Word vectors are trained using Skip-gram model.

same listing price. For location similarity, we consider properties with similar zipcode and community subdivision. In terms of the square feet and lot size, we consider them similar if their difference is less than 10% of their average.

We use 5000 property listings and find the pairs that are similar for all the property features that we mentioned. for each property we compute a score based on the property description and agent remarks. We found 310 similar pairs out of our sample set.

For each pair, if one of the properties' price is greater than the other property's price with a difference greater than 10% of their price average and its text score is also greater, we say that scoring function and listing price have agreement.

Table 3.2 shows the results of the evaluation for different proposed algorithms. For NCM and NRCM, Skip-gram based word vectors are used to match approximate concepts with a similarity threshold equal to 0.8. For each algorithm the number of pairs with agreement is listed in the table. This evaluation also shows that performance of NCM is slightly better than NRCM as shown in human judgement base evaluation.

Table 3.2: Data-driven ranking evaluation.

	# of properties	**# of similar pairs**	**# price/score agreement**
ECM	5000	310	42
NCM	5000	310	73
NRNCM	5000	310	70

3.5 Conclusion

In this chapter we propose a semi-supervised method for building a comprehensive real estate concept collection using a corpus of property descriptions from MLS. We propose an effective unsupervised method to annotate property descriptions and extract real estate concepts. We use the extracted exact and approximate concepts goodness scores to compute a score for property description. The calculated score is used to rank the property descriptions. We use both human judgements and a data-driven approach to evaluate our algorithms. Our results indicate that the ranking by NCM algorithm (weighted aggregation of exact and approximate concepts) is the most effective method, which has the highest rank correlation of 0.76 with human judgements.

Chapter 4

Finding Similar Documents using Keyword-Based Search Interfaces

Web search tools commonly provide keyword-based query interfaces to allow user search for documents in enormous Web collections. However, they are limited by design to expect a limited number of keyword query and do not let the user input a set of documents to look for more similar documents. Selecting effective keyword queries that mostly find documents that are similar to input documents is crucial because users don't have direct access to the underlying collection and issuing search queries through interfaces is costly and rate-limited. Here, we propose a solution to detect query keywords that are effective in finding documents similar to the set of input documents by taking into account the collection statistics. Our experiments show that our proposed solution outperforms the state of the art and baseline methods in finding effective keyword queries.

4.1 Introduction

A common problem in Information Retrieval is that a user wants to retrieve documents similar to a given set of relevant documents. For example, a patent attorney may have a few documents provided by a client describing an invention and would like to search for patents similar to these input documents. Similarly, a scientist may search for related work in an area and may have access to a few documents related to this area. A Web user may also have found a set of documents related to a topic, e.g., related to the topic of academic scandals, and may be looking for more similar documents on the Web.

Although there are powerful keyword search interfaces on top of various collections – LexixNexis for patent search [3], Google Search API for the Web [1], etc. – a common limitation they have is that they do not allow the user to input a set of documents (one or thousands), but expect a relatively small number of terms as a query. Further, these interfaces typically charge a fee for every page of query results. For example, the LexisNexis Statistical Gateway charges \$0.30-\$0.40 per query, and Google Enterprise charges \$100 for 20,000 queries, where each query returns one page of results. Note that a common property in all these collections is that the user may not have access to the underlying collection through any way other than through the provided search APIs. Hence, to use these interfaces, one has to extract sets of important terms from the input documents, to formulate queries. These queries should ideally return many similar documents in high-rank positions of the results.

Given that a collection is only accessible through a search interface, this work proposes effective techniques to generate queries from a set of input documents, which return similar documents to the input documents in high positions. We refer to this problem as *Docs2Queries*.

The Docs2Queries problem has received limited attention by the community [106, 124]. The state-of-art works focus on extracting good terms from the input documents, given a basic understanding of the ranking formula, which is generally tf-idf (term frequency and inverse document frequency) based. Specifically, they select terms with high tf-idf score. This is a reasonable heuristic, but its drawback is that it ignores the language model of the collection and limits the heuristics to use only information from the input documents.

We propose a more principled approach to select the best queries, which also considers the language model of the collection (specifically, an estimation of it, as the full collection is not available to us). Our key *hypothesis* is that the best queries to return the input documents in high-rank positions will also return other similar documents in high-rank positions. That is, we focus on how to compute queries that will return our given input documents in high positions. Our experimental results confirm the validity of our hypothesis, and also the superiority of our method compared to the state-of-the-art methods. Specifically, our approach outperforms the state-of-the-art by up to 40%.

More specifically, we follow a probabilistic approach, where for each candidate query, we compute the probabilistic distribution of positions of the input documents and pick the queries where the expected average position of the input documents is minimized. Our approach does not require exact knowledge of the ranking formula used by the search API, but assumes that tf-idf is a key factor, as does previous work [124]. Note that most popular IR ranking formulas – vector space cosine similarity, language model, probabilistic model – include a significant tf-idf factor [34].

A key challenge is the huge search space of candidate queries. Another challenge is that the collection is not available to us (except via the search API), and hence we do not have accurate statistics to allow us to compute the positions' probability distributions. For that, we employ various sampling techniques. Another challenge is to account for the overlap between selected queries, so they do not return many common documents, given that each page of results has an access cost.

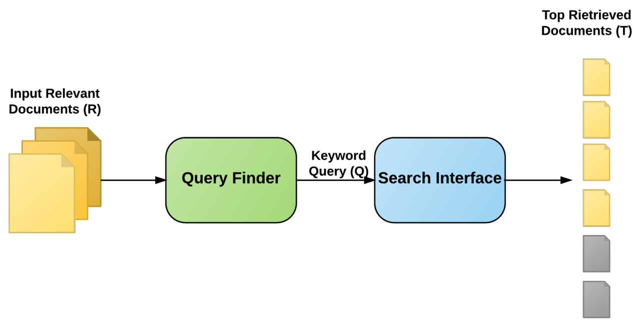


Figure 4.1: System architecture: Docs2Queries module finds queries that return documents similar to the input documents in top positions.

Our contribution in this study are as follows:

- Propose a principled solution to the Docs2Queries problem of extracting the best queries given a set of input documents (Section 4.3 and Section 4.4).
- Use two different collection sampling methods to estimate the collection statistics (Section 4.5).
- Confirming our hypothesis by comparing our proposed algorithms with baseline and state of the art approaches (Section 4.5).

4.2 Related Work

In this project, the goal is to detect queries that are effective in finding similar documents to an input set of example documents using a keyword-based search interface. Here, we describe the related work on query refinement and finding similar documents.

Relevance feedback (RF): RF is a commonly used feature in information retrieval systems that uses user's interaction with the system to refine the query for improving the search results [99]. RF methods are mostly based on Rocchio's algorithm [99, 27]. In Rocchio's approach, documents and query are represented in a vector space. Every time user feedback is available, the query is refined by adding the relevant documents with a positive weight and the non-relevant documents with a negative weight.

RF is the closest line of work to our problem; however, the assumption of having the iterative improvement does not apply here because, in our case, there is only a predefined set of documents as relevant examples and users are not generating further feedback on returned results.

Key-phrase extraction: Extracting key phrases from a document is a well-studied problem. These key-phrases could be suited to tasks like relevance filtering or browsing in retrieval [76]. Supervised approaches find key-phrases by training machine learning models for identifying key-phrases, using training documents where the key-phrases are known [121, 111, 64]. In [46], authors use dynamic query modeling to find related content based on a textual stream.

On the other hand, unsupervised methods rely on statistical information to select key-phrases [87]. A basic approach that comes to mind is to rank terms or n-grams in the input document by frequency or tf-idf score [127, 76]. Another unsupervised approach suggested in [79] involves clustering the candidate key-phrases in a document into topics, such that each topic is composed of all and only those candidate key-phrases that are related to that topic. In [124], authors select key-phrases by identifying noun-phrases using part of speech tagging. Most of the key-phrase generation methods, order the key-phrases based on a form of tf-idf score.

Search by document: Search for similar documents given an input document is a directly relevant research area to our work [106, 124]. Yang et al. in [124] proposed *Query by Document* (QBD) addressing the problem of cross-referencing on-line information in the context of blogs. They select noun phrases from an input document to use as keyword queries to search for pages similar to that document. In [36], Dasdan et al. propose an approach to the covering test problem where the goal is to find out if there is a near-duplicate of a certain document in a corpus of documents that is accessible using a rate-limited keyword query interface.

In both cases, the generated queries are merely based on the input document and not taking into account much of the collection statistics that matter in ranking function while computing the score of the candidate queries. Further, they rely on query generation methods to select the candidate queries.

Vidhya et al. proposed a personalized query formulation method based on identifying key phrases from an input document [53]. The key-phrases are used to query a search engine and the results are evaluated for similarity to the original document. They find the key-phrases by taking the co-occurrences within the input document. They use Jaccard similarity to measure the similarity of the retrieved document to the input document.

Document similarity functions In Document Filtering, several methods to define the similarly of a document to a user's profile (defined as a set of documents) have been proposed. One of them, which we also use in our experiments, is the KL divergence [33, 32].

4.3 Definitions

We begin by defining the key data types in our problem definition and proposed algorithms.

Let C be the collection of documents that are indexed by a search engine. We consider search engines that provide a keyword-based query interface to the users for accessing the documents in the indexed collection. The *keyword-based query interface* is the user interface to the Web collection that inputs a set of keywords and outputs a ranked list of documents. .

A *keyword query* $Q = \{q_1, q_2, .., q_n\}$ is a set of n uni-grams. As we mentioned earlier, we study the problem of finding keyword queries from a set of input documents provided by the users. Let define $R = \{d_1, d_2, ..., d_k\}$ as a set of k example input documents to extract keyword queries that will result in retrieving similar documents.

Search engine providers do not reveal the exact ranking function of their system as it is an asset of the business; however, it is fair to assume the text-based relevance ranking is a function of commonly used "term frequency inverse document frequency" (tf-idf) score.

As a result, we explain our proposed solution by assuming the score of document d given keyword query Q is computed as follows:

$$score(d, Q) = \sum_{q \in Q} tf(d, q) \times idf(q) \tag{4.1}$$

where $tf(d, q)$ is the term frequency of keyword q in document d and $idf(q)$ is the inverse document frequency of q.

Similar documents problem: Given a set of input documents, a search interface, number T of search results per query, number m of queries and maximum query length n, find as many similar documents to input documents as possible.

Note that T and m represent the budget of our problem. For example, one need to \$100 for 20,000 queries in Google enterprise search [1].

Our hypothesis is that queries that will return the input documents in high-rank positions, also return similar documents. Next, we define the problem of *Docs2Queries* with regard to our hypothesis.

Minimum position problem: Given R, the set of example input documents find Q, a query with up to n keywords such that the average rank position of input documents given Q is minimized. Formally, we define this objective function as follows:

$$Q = \arg\min_{Q} \frac{1}{|R|} \sum_{d \in R} position(d, Q) \tag{4.2}$$

We extend the definition to consider finding $O = \{Q_1, Q_2, \cdots, Q_m\}$, a set of m queries, for which the average minimum position of the input documents across all queries is minimized. Formally, we define:

$$O = \arg\min_{O} \frac{1}{|R|} \sum_{d \in R} \min_{Q \in O}(position(d, Q)) \tag{4.3}$$

There are different ways to find document similarity. In our experiments, we use KL divergence that has been used for document filtering [32]. Our approach is not limited to this distance metric and we expect that it still work with other document similarity metrics.

Our theoretical model solves the minimum position problem and our experiments support our hypothesis that it also works for similar document problem.

4.4 Algorithms

In this section, we explain our approach to solve the problem of finding search queries to retrieve similar documents. First, we show how to solve the problem for finding the best single query with a single keyword based on Equation 4.2. Then, we extend our solution to finding the best single query with multiple keywords. Finally, we explain our approach for the solution case with multiple queries including multiple keywords with respect to Equation 4.3. As mentioned earlier, we assume that search engine's score function for document d with respect to Query Q is computed using Equation 4.1. Thus, the rank position of d within the collection given Q is defined as follows:

$$position(d, Q) = probability(X_Q > score(d, Q)) \times |C| \tag{4.4}$$

Where X_Q is the score of a random document in the collection with respect to Q. Next, we will explain how to solve the problem starting by the easiest case paving the road to solving the general form of the problem definition in Equation 4.3.

4.4.1 Single-keyword query

We start by describing our solution for the simplest case where the goal is to find the best query $Q = \{q\}$ with a single keyword. Given this condition, the following stands:

$$\begin{aligned} & probability(X_Q > score(d, Q)) \\ & \qquad = probability(Y_q > tf(d, q)) \\ & \qquad = 1 - \sum_{k < score(d,q)} probability(Y_q = k) \end{aligned} \tag{4.5}$$

Where Y_q is the term frequency of keyword q for a random document in collection. This equation reduces the computation of document score probability to term frequency probability. Thus, the position of document is defined as follows:

$$position(d, q) = \sum_{k > score(d,q)} p_q(X = k) \times |C| \tag{4.6}$$

If the access to corpus or a representative sample of the corpus is possible, term frequency probabilities can be obtained from the data. However, in absence of complete data-driven term probability distribution, according to Theobald et al. [109], Poisson probability distribution is an appropriate estimator of the term frequency distribution within the corpus. As a result, the only parameter necessary to compute the term frequency probability is the average term frequency. According to this assumption, the term frequency distribution is defined as follows:

$$probability(X_q = k) = e^{-\lambda_q} \frac{\lambda_q^k}{k!} \tag{4.7}$$

Where λ_q is Poisson parameter and represents the average term frequency of keyword q in the collection.

We can now define the objective function to find a single-keyword query based on term frequency distribution as follows:

$$q = \arg\min_q \sum_{d \in R} \sum_{k > score(d,q)} probability(Y_q = k) \tag{4.8}$$

where the term frequency probability can be computed using data distribution information or based on Poisson distribution in Equation 4.7. In practice, only up to top_T documents are return for every query. By incorporating a threshold T that is defined as the maximum returned documents for a query, we alter the objective function in Equation 4.8 as follows:

$$q = \arg\min_{q} \sum_{d \in R} Minimum\{\frac{T+1}{|C|}, \sum_{k > score(d,q)} probability(X_q = k)\} \tag{4.9}$$

In other words, we treat all the documents that are ranked lower than position T equally and place them in $T+1$th position in the ranking.

In practice, to avoid having to evaluate Equation 4.9 for all terms in the input documents R, we only consider the l terms with the highest tfi-df score for concatenated input documents, where tf is computed in R and idf in the whole collection C. In the experiments, we set $l = 100$, and found that the same query is selected as when all terms are considered.

4.4.2 Multi-Keyword Query

In order to compute the position of a document given a query with multiple keywords, we assume that term occurrences are independent within a document. By making independence assumption, we define the objective function to find the best query as follows:

$$\begin{aligned} &Q = \arg\min_{Q} \sum_{d \in R} probability(X_Q > score(d,Q)) \\ &Where\ we\ assume, \\ &probability(X_Q > score(d,Q)) \approx \sum_{k_1,\ldots,k_n | \sum_{i=1}^{n} k_i \times idf(q_i) > score(d,Q)} \prod_{i=1}^{n} probability(Y_{q_i} = k_i) \end{aligned} \tag{4.10}$$

The exact solution to this equation is computed by considering all keyword combinations to form queries of size n from the keywords that are within vocabulary of input documents. However, for practical purposes, we relax the problem in Equation 4.10 by limiting the query space only consider keywords with top tf-idf score in input documents. In addition, we only consider frequencies up to a constant z in frequency combination space. As a result, Equation 4.10 with regard to constant z changes as follows:

$$probability(X_Q > score(d,Q)) \approx \sum_{k_1,\ldots,k_n \mid \sum_{i=1}^{n} k_i \times idf(q_i) > score(d,Q), k_i < z} \prod_{i=1}^{n} probability(Y_{q_i} = k_i) \tag{4.11}$$

The time complexity is affected by the time to search in the query space with $O(l^n)$, and to compute score probabilities with $O(z^m)$. In our experiments, we set $z = 5$ and $l = 100$, and found that the outcome is not that affected given the significant computational cost decrease comparing to using larger values for the parameters. Note that, so far we explained how to find a query with n keywords. We will repeat the computation for queries with one keyword up to n and select the best query among them.

4.4.3 Multiple Multi-Keyword Queries

Finally, to find m queries with up to n keywords, to be practical, we propose the following heuristic that finds m queries by partitioning the input documents into m clusters by K-Nearest-Neighbor (KNN) algorithm. Because the objective is to find queries that best position input documents in high-rank positions, we call our algorithm Best Position (BP).

In the next section, we define variants of our algorithms based on the way we compute the term frequency distribution.

Best Position (BP) Algorithm:

Repeat the following steps m times:

1. Select a random document from input set.
2. Find the $(|R|/m) - 1$ closest documents to the selected document.
3. Find a query with up to n keywords for the selected $(|R|/m)$ documents.
4. Remove the selected documents from the input set.

Note that we use the cosine similarity of the document vectors as the similarity measure. Also, in our experiments, we use the tf-idf vector of the vocabulary words as the continuous distributed document representation. We experimented the effectiveness of several heuristics and found that the above method works best.

4.5 Evaluation Results

4.5.1 Experimental Settings

We use Lucene search engine with a tf-idf based scoring function as the search engine with keyword query interface in our experiments. In this section, we describe the experimental results comparing our solution that is based on best position (BP), with the state of the art (QBD) and a baseline method.

Dataset: In our experiments, we use "TREC-9 Filtering Track" test collection that is a set of 348,566 abstracts of references from MEDLINE, the on-line medical information database, consisting of titles and/or abstracts from 270 medical journals over a five-year period (1987-1991). The available fields include title, abstract and few other fields [57]. As the set of input documents, we use the filtering topics that are a subset of 63 of the original query set developed by Hersh et al.xi for their IR experiments [57]. As a second dataset, we use TREC-8, "Ad hoc Test Collections". These collections include 1.6M English articles from news media such as Wall Street Journal and Financial Times Limited. Table 4.1 shows the details on dataset characteristics after filtering documents based on minimum length of 10 and terms with minimum document frequency of 10.

Table 4.1: Datasets characteristics.

	TREC-9	TREC-8
#Documents	233444	1634243
Average Document Length	84	200
Vocabulary Size	38849	254102
#Query Sets	63	50
Average Size of Query Sets	30	237
Average Document Frequency	505	1287

Baseline techniques:

State of the art (QBD): QBD is the technique that is proposed by Yang et al. to to automate the process of cross referencing on-line information content [124]. They first extract noun-phrases (i.e. sequences of nouns and adjectives) from the input document as the candidate queries using part of speech tagging tools. Then, they sort candidates based on the following scoring function for phrase P given input document d as follows:

$$score(P, d) = \sum_{t \in P} tf(d, t) \times idf(t) + \alpha * coherence(P) \tag{4.12}$$

Where $coherence(P)$ is defined as follows:

$$coherence(P) = \frac{tf(d,P) \times (1 + logtf(d,P))}{\frac{1}{|P|} \times \sum_{t \in P} tf(d,t)} \tag{4.13}$$

Authors propose another scoring function based on co-occurrence information of the terms in a phrase; but we omit that method because it results in worse performance than the one in Equation 4.12.

Baseline: In addition to QBD, the state of the art solution, we define the following baseline to compare against our proposed solution for finding m queries with up to n keywords:

1. Select top $n * m$ keywords with the highest sum of tf-idf in concatenated relevant documents
2. Order the keywords by the tf-idf score defined in step 1 and partition to m cluster of keywords where each cluster is considered as one query.

BP-QBD: We also apply our Best Position (BP) score to noun phrases identified by the part of speech tagging to compare against QBD. In experiments results, we refer to this method as BP-QBD.

Estimating corpus statistics: All proposed algorithms and baselines require some statistics from the collection for computing $probability(X_q)$. We propose to estimate the term frequency distribution either based on Poisson distribution (*BP-P*) or based on the distribution that is driven from sample set (*BP-D*). Table 4.2 shows different statistics required by each algorithm.

Table 4.2: My caption

	Inverse Document Frequency (idf)	**Average Term Frequency (λ_q)**	**Term Frequency Distribution ($probability(X_q)$)**
TF-IDF	x		
QBD	x		
BP-QBD	x		x
BP-P	x	x	
BP-D	x		x

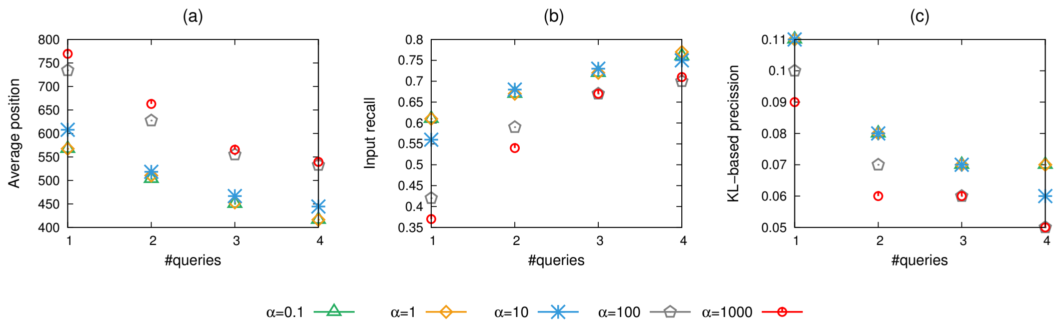


Figure 4.2: Query effectiveness wiith different number of queries for QBD vs. different values of coherence parameter α based on (a) Average position, (b) Recall, and (c) KL-based precision. idf is estimated with query-based sampling technique (S2).

We evaluate the effectiveness of different algorithms using two different ways of estimating the corpus statistics.

Random sample of the collection (S1): The first approach is to assume that a representative subset of the underlying collection is available. For instance, Wikipedia dataset could be used as a sample of web collection indexed by general-purpose search engines such as Google. In our experiments, we generate this subset, by randomly selecting a set of samples documents from datasets that are used in our experiments.

Sampling by submitting keyword queries (S2): We follow the proposed approach by Ipeirotis et al. to compute document frequencies corpus statistics using et al. work on estimating document frequency [62]. The sampling for collecting a set of documents is an iterative process with the following steps:

Start by choosing a seed query keyword. While the number of sampled documents is less than required (sample size is a parameter), do:

1. Select a random keyword from sampled documents vocabulary.
2. Issue a query using selected keyword to search interface and add the top 3 (system parameter) documents to the sample set.

Figure 4.3 shows the correlation of document frequencies estimated by this method with actual document frequency of the terms as well as the Relative error of estimations to the actual values. As the size of sample set increases, we cover more words from the vocabulary of the collection but at the same time, the error increases because of increase in the number of terms with low document frequency. However, as it is shown in Figure 4.3(a), the rank correlation of terms ordered by document frequency increases. In other words, the accuracy of the relative order of document frequency estimation increases. We further extend this method and compute term frequency distributions by considering sample distribution scaled by estimated collection size divided by sample size. We assume the corpus size is equal to the largest estimated document frequency.

Note that S2 is a more practical way to sample the collection because it does not need to have access to the underlying collection.

Evaluation metrics: For each input set, we use half of the documents for finding queries and the other half for testing the effectiveness of the queries. We measure the effectiveness

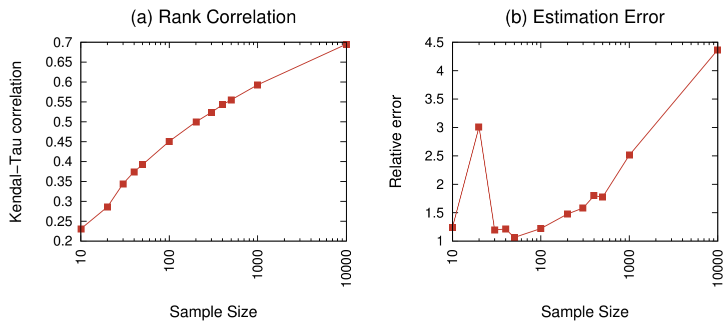


Figure 4.3: Evaluating document frequency estimation for sampling TREC-9 dataset by querying search interface (S2) [62]

Table 4.3: Parameters

Parameter	**Value**
Maximum number of keywords (n)	2
Number of top retrieved document threshold (T)	1000
Maximum term frequency threshold (z)	5
Number of pruned vocabulary terms (l)	100
Hybrid method threshold (d)	3

of the queries selected by different solutions based on three different metrics:1) Average position of test input documents, 2) Recall of the test input documents, and 3) Precision of the returned top documents based on KL distance of the documents with the concatenated test input documents. More precisely, we consider a retrieved document as similar to input documents, if the KL distance is less than a threshold, kl_ths. Where kl_ths is the average KL divergence of every input document from the aggregation of the rest of relevant documents and computed as shown in Equation 4.14. KL function computes the KL divergence of the two input documents.

$$kl\_ths = mean(KL(r, concat(R - r))) \qquad ,\text{for } \ r \in R \tag{4.14}$$

4.5.2 Experimental Results:

In this section, We compare the effectiveness of the queries selected by each method. First, we evaluate the effectiveness versus the number of queries. The constant factors for this experiment are set as shown in Table 4.3.

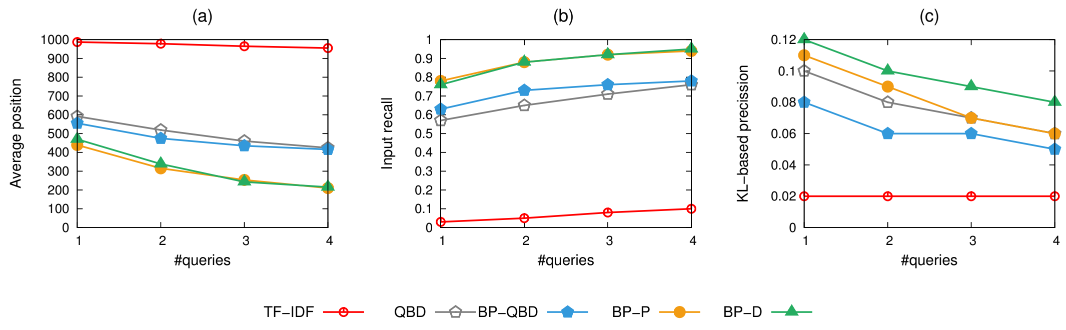


Figure 4.4: TREC-9 dataset; Query effectiveness vs. number of queries based on (a) Average position, (b) Recall, and (c) KL-based precision. Collection statistics of this experiment are based on random sampling of the documents in collection (S1).

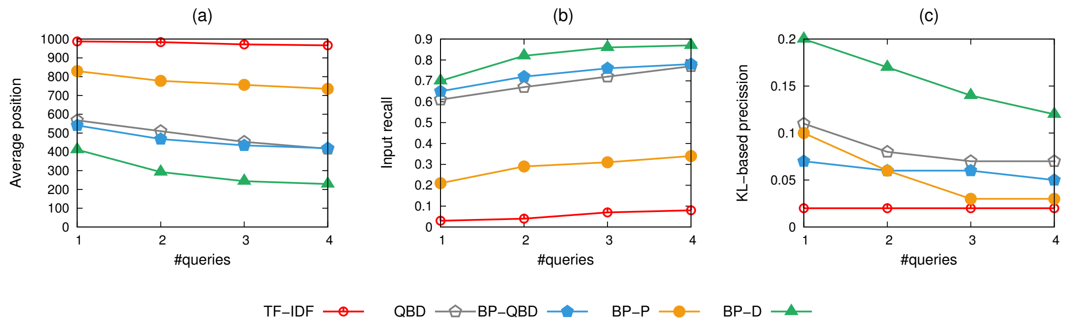


Figure 4.5: TREC-9 dataset; Query effectiveness vs. number of queries based on (a) Average position, (b) Recall, and (c) KL-based precision. Collection statistics of this experiment are based on sampling the collection by submitting queries through search interface (S2).

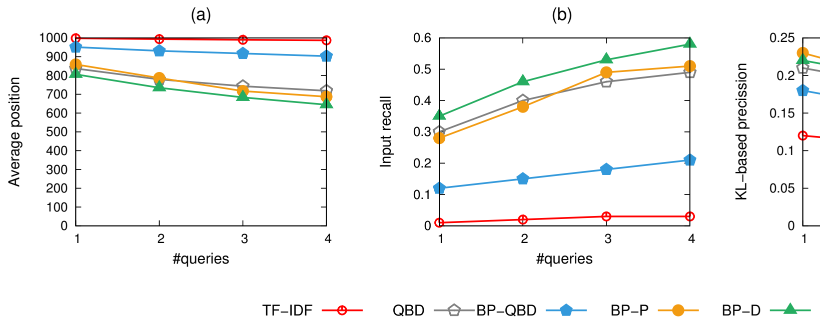


Figure 4.6: TREC-8 dataset; Query effectiveness vs. number of queries based on (a) Average position, (b) Recall, and (c) KL-based precision. Collection statistics of this experiment are based on random sampling of the documents in collection (S1).

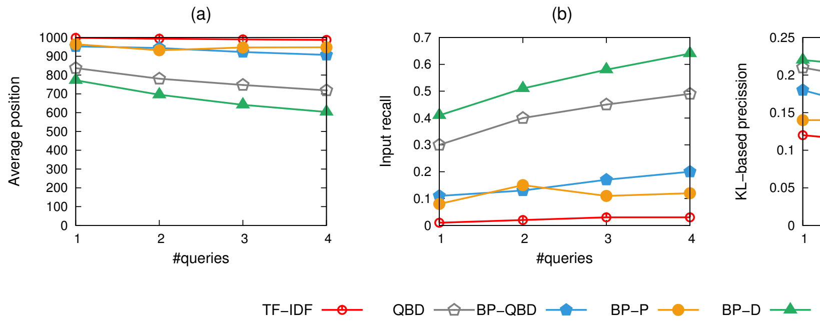


Figure 4.7: TREC-8 dataset; Query effectiveness vs. number of queries based on (a) Average position, (b) Recall, and (c) KL-based precision. Collection statistics of this experiment are based on sampling the collection by submitting queries through search interface (S2).

We first evaluate different values of parameter α in Equation 4.12 to obtain the best value to use as the coherence factor in QBD method. Figure 4.2 shows that the best result is achieved by setting α to less than or equal 1. Based on this observation, for the rest of the experiments in this section, we use $\alpha = 1$.

Figures 4.4 and 4.5 show the effectiveness results on TREC-9 dataset. Figure 4.4 shows that our method, BP-D, that finds best queries with respect to minimizing rank positions has the best performance among different methods. More precisely, we observe that average rank positions decreases by 49% using QBD relative to the tf-idf baseline and decreases by 68% using BP-D. Our experiments show that with random sampling technique, the effectiveness of QBD and BP-QBD are comparable as well as the effectiveness of BP-D and BP-P. The methods with lower average precision have a higher recall for test input documents. In terms of KL-based recall, we observe that BP-D has the highest recall. BP-P is the second best and unlike the recall, it has a lower recall than BP-D. Also, although BP-QBD has a higher recall of input test documents than QBD, it has a lower kl-based recall.

In Figure 4.5, we repeat the same experiments with S2 sampling technique. The experiment's outcome for almost all methods does not show a noticeable change except for BP-P. The effectiveness of BP-P becomes worse due to a much less accurate estimation of the average term frequency. In S2, sampling by submitting queries to search interface, the main goal is to have accurate estimations of document frequency and it is not necessarily a good estimator for average term frequencies. All in all, in TREC-9 that is a domain-specific collection, S2 sampling leads to overall more accurate results for BP-D than S1.

In Figures 4.6 and 4.7 we show our evaluation results using TREC-8 dataset. TREC-8 is a more general dataset comparing to TREC-9 and includes articles in a veriety of domains. Our experiments in Figure 4.6 show that BP-D is still the most effective method; However, on average, the effectiveness comparing to the state of the art is 6% higher using S1 and 12% higher using S2. In addition, the recall of similar test documents as well as kl-based recall of similar documents is higher using S2 in TREC-8.

4.6 Conclusion

We explored the problem of *Docs2Queries* and proposed a principled way to extract best queries for the input documents. Our proposed solution is based on the collection's language model. In our experiments, we considered two different simple ways of sampling the collection to estimate the language model. Our experimental results based on TREC-9 and TREC-8 datasets show that our BP-D solution comparing to the state of the art excels in effectiveness by being on average 40% and 10% more successful in positioning similar documents, respectively. We showed that the best results are achieved with BP-D using sampling by submitting queries to search interface that is a more practical sampling approach.

Chapter 5

Content analysis on Anonymous Social Networks

Yik Yak is a location-based social media platform that lets users post and read messages anonymously. In view of recurring reports of harassment and abuse among college undergraduates, we conducted an exploratory study of Yik Yak messages from 19 universities across four U.S. states. We found that prosocial messages were about five times as frequent as bullying messages. Significant geographic variation in the relative frequency of messages offering support and bullying messages included a potentially problematic pattern of increased bullying paired with low levels of social support at Texas universities. Relationships/sex was the most frequently discussed topic and school/classes was the least popular topic among college students. A correlation analysis revealed that at schools where students sought help more often in their Yik Yak messages, messages offering support were also more frequent. Two findings illuminate the social-contextual conditions of cyberbullying. First, the frequency of bullying messages was positively related to the popularity of messages seeking help. Second, more bullying occurred on campuses where students were more likely to discuss politics.

5.1 Introduction

College attendance marks an important period of psychosocial development with significant implications for a healthy and productive adulthood. The academic and social demands of college life are often strenuous and pose a risk to students health and well-being. One problem among college freshman, for example, is poor sleep [113] which has been linked to a number of adverse consequences in this population, including higher rates of depressive symptoms and stress [120, 50], weight gain [98], and poor academic performance [35]. Another relatively recent problem among college students, due to technologies like social media, is cyberbullying, which can lead to depression and suicide.

In light of the various health-related risks associated with college attendance, it is important to monitor student health in order to provide adequate services and support. Traditionally, methods for monitoring students health have focused on case reports and surveys.While these methods can offer detailed insights into health-related attitudes and behaviors, their application tends to be time and cost intensive. Ideally, one would have the capacity to collect and analyze health-relevant data continuously and in real time so as to address student health needs in a flexible and timely manner. Researchers have therefore explored the feasibility of using social media platforms to identify and predict health-related events. In one study, for example, Young, Rivers, and Lewis screened geolocated Twitter messages for keywords that suggest HIV-risk related behaviors and found a significant relationship between the frequency of use of these keywords and HIV prevalence at the level of U.S. counties [125]. De Choudhury, Gamon, Counts, and Horvitz were able to predict the onset of major depressive disorder with 70% accuracy on the basis of peoples Twitter messages, relying on indicators such as linguistic style, use of depression terms, and social-network characteristics [37]. Because of the prevalent use of social media among college students, these technologies might be used to monitor student health and well-being.

Yik Yak is a relatively recent addition to the social media world that quickly grew in popularity among U.S. college students after its inception in 2013. It functions as an online bulletin board on which users within the same geographic area (e.g., a college campus) can post and read messages anonymously. Critics of the social network argueaided by anecdotal evidence relayed through media reportsthat anonymous posting encourages harassment and bullying [63, 7, 8, 117]. In a recent content analysis of Yik Yak conversations, Black, Mezzina, and Thompson did not find evidence for a pervasive culture of harassment and abuse, but they did observe derogatory and incendiary comments, arguably racist and sexist messages, and several likely instance of bullying [19].

Based on these findings, we carried out an exploratory study of posts on the Yik Yak social network. Our goal was to help provide insights for school administrators and public health researchers on the prevalence and popularity of messaging behaviors such as bullying and social support, and of topics discussed on the network. Knowledge of these activities on Yik Yak can then be used to improve student well-being, for example, by guiding interventions that promote healthy and prosocial behaviors.

Table 5.1: Characteristics of Universities Included in the Study

State	University	Public/Private	Enrollment	Ranking
CA	California Polytechnic State University	Public	19,226	221
	CSU Chico	Public	16,535	467
	CSU Los Angeles	Public	20,353	700
	CSU San Bernardino	Public	17,167	700
	University of California Irvine	Public	25,001	153
FL	Florida International University	Public	53,525	550
	Florida State University	Public	36,575	226
	University of Central Florida	Public	59,894	445
	University of Florida	Public	36,731	56
	University of South Florida	Public	35,035	396
NY	Cornell University	Private	14,706	9
	CUNY Hunter College	Public	20,582	350
	CUNY John Jay College of Criminal Justice	Public	15,845	700
	SUNY Buffalo State	Public	10,665	700
	SUNY New Paltz	Public	7,756	423
TX	Tarleton State University	Public	11,008	800
	Texas Tech University	Public	29,342	550
	University of Houston	Public	36,128	388
	University of Texas Rio Grande Valley	Public	27,560[1]	

Note. Source of enrollment and ranking data: Wall Street Journal/Times Higher Education College Rankings 2017. CA = California; FL = Florida; NY = New York; TX = Texas; CSU = California State University; SUNY = State University of New York; CUNY = City University of New York.

5.2 Methods

From April 6th, 2016 to May 7th, 2016, we collected anonymous conversations posted on the Yik Yak social network at 19 universities located in California, Florida, New York, and Texas (Table 5.1). We randomly selected 100 conversation threads from each of the 19 universities, for a total of of 16,966 messages, with a mean of 893 messages per university (SD = 128). We analyzed the messages with respect to the type of messaging behavior, message content, and popularity of message type and content.

5.2.1 Messaging Behaviors

For each Yik Yak message we determined whether it displayed, if any, one of four predefined behaviorsseeking help, offering support, being funny, and bullying. These predefined behaviors were decided based on the literature on bullying on Yik Yak, as well as topics that were decided to be relevant to undergraduates from a team of 3 undergraduate students. Initially, 90 randomly selected messages were coded independently by two undergraduate raters.

5.2.2 Message Topics

For the analysis of message content, we applied latent Dirichlet allocation (LDA) to the message corpus in order to identify themes that were discussed on Yik Yak [20]. LDA is a topic modeling algorithm that models the messages as a mixture of underlying topics. Each topic, in turn, is probabilistically associated with various words. Since a topic is defined purely in statistical terms, its semantic interpretationthat is, its labelis chosen by the user on the basis of the word probabilities for the topic.

Next, we sought to identify those topics for which the LDA message classifications aligned most closely with human judgment. This was done on the basis of a subset of 1200 randomly selected messages to which the LDA assigned a topic with a probability greater than 7. For each of these messages, a team of three raters decided whether or not the LDA topic assignment was correct (i.e., does the message discuss Topic X?). Based on these results, we selected the four topics with the highest classification accuraciesrelationships/sex, college living, politics, and school/classes.

In the final step, two undergraduate raters independently applied the four-topic classification scheme to 90 randomly selected messages. We found that their inter-rater agreement was high (Cohens kappa = .78), so all remaining messages were coded by either one of the two raters.

5.2.3 Message Popularity

The popularity of a message was determined by the aggregate score of +1 votes (upvotes) and -1 votes (downvotes) the message had received from Yik Yak users up to the time of data collection. Of note, if a message on Yik Yak reaches a sum score of -5, it is automatically deleted from the social network. Thus, the lowest possible popularity score for a message in our data set was -4.

5.3 Results

In all statistical analyses the significance criterion was $\alpha = .05$.

5.3.1 Frequency of Messaging Behaviors

Our first set of analyses concerns the relative frequency of four messaging behaviors seeking help, offering support, bullying, and humor. We included messages that were uniquely classified as one of the four messaging behaviors and messages that did not fall into any of the categories. 51 messages (0.3%) did not meet these criteria and were not analyzed further.

Overall, 12% of messages were assigned to one of the four message categories. Specifically, the percentages of messages with which Yik Yak users sought help, offered support, intended to be funny, or bullied were 1.8, 5.2, 3.1, and 1.9, respectively. We then asked whether this pattern of usage varied across the U.S. Table 5.2 (top) breaks down the relative frequency of message types by state. Separate Fishers exact tests for each messaging behavior showed that there were significant differences between states in the relative frequency of messages offering support, $p < .001$, and of bullying messages, p< .001, but we found no geographic differences for messages seeking help, $p = .2$, or for funny messages, $p = .4$.

The relative frequency of supporting messages ranged from 2.5% in Texas to 8.1% in Florida. Multiple pairwise comparisons between states using Bonferroni-corrected Fishers exact test showed significant differences for every possible combination of states, all $ps < .05$. In a comparable analysis of bullying messages, we found that the two states with the lowest rates of bullying, California and Florida, differed significantly from the states with the highest rates, New York and Texas, all $ps < .05$, but there was no difference between California and Florida, $p = 1.0$, or between New York and Texas, $p = 1.0$.

5.3.2 Frequency of Topics

To assess the relative frequency of topics discussed on Yik Yak, we used messages that raters uniquely assigned to one or to none of four LDA-derived topics (relationships/sex, college living, politics, school/classes). This led to the exclusion of 117 messages (0.7%) from the frequency analysis. 26% of the remaining messages dealt with either relationships and sex (14.9%), college living (3.8%), politics (3.6%), or school and classes (4.0%). In Table 5.2 (bottom), these numbers are broken down further by state.

By conducting separate Fishers exact tests, we found significant regional differences for each topic. We followed up on these significant effects with Bonferroni-corrected Fishers exact tests for all pairwise comparison between states for each topic. Relationship messages were less frequent in New York, the state with the lowest rate, than in California, $p < .001$, and Texas, $p = .048$. For messages about college living, significant differences emerged between all states (all $ps < .05$), except for California and Texas, the two states with the highest prevalence of this topic ($p = 1.0$). Yik Yak users in the four states also messaged about politics more or less frequently, all $ps < .001$, with the exception of California and Florida, the two states with intermediate levels of political messaging ($p = 1.0$). Finally, school and classes were talked about with different frequencies in the four states (all $ps < .05$), save for California and Texas, where this topic garnered the most interest.

5.3.3 Popularity of Messaging Behaviors

In this and the following section we report findings on the popularity of the different messaging behaviors and topics, based on the aggregate of +1 votes (upvotes) and -1 votes (downvotes) each message elicited from Yik Yak users. We refer to this total as the popularity score of a message. In order to protect our analyses from an undue influence of a few massively popular messages, we flagged messages with a score greater than three standard deviations above the grand mean. 305 messages (1.8%) were identified as popularity outliers and excluded from further analysis.

Table 5.3 (top) displays the mean popularity scores for the four messaging behaviors at the state level. We submitted the individual message scores to a State (CA, FL, NY, TX) x Behavior (seeking help, offering support, bullying, humor) analysis of variance (ANOVA). Both main effects were significant, $F(3, 1940) = 5.11$, $MSE = 4.1$, $p = .002$, for State, and $F(3, 1940) = 25.19$, $MSE = 4.1$, $p < .001$, for Behavior. The interaction between the two factors was not significant, $F(9, 1940) = 1.16$, $MSE = 4.1$, $p = .319$.

In order to determine which states exhibit significantly different mean popularity scores, we used Tukeys range test as a single-step multiple comparison procedure. This analysis revealed that, on average, Yik Yak messages received lower popularity scores in Texas than in Florida and New York, both $ps < .05$. Following up on the significant main effect of Behavior, Tukeys test showed that bullying messages were the least popular, differing significantly from messages seeking help, offering support, and funny messages, all $ps < .01$. By contrast, funny messages were liked the most. Their popularity score was significantly greater than the scores for messages seeking help, offering support, and for bullying messages, all $ps < .001$.

5.3.4 Popularity of Topics

Table 5.3 (bottom) summarizes the mean popularity scores of messages that discussed one of the four topics identified through LDArelationships and sex, college living, politics, and school and classes. A State (CA, FL, NY, TX) x Topic ANOVA revealed main effects of State, $F(3, 4293) = 11.23$, $MSE = 4.9$, $p < .001$, and of Topic, $F(3, 4293) = 6.03$,

Table 5.2: Frequency of messaging behaviors and topics by State

Frequency of Messaging Behaviors and Topics by State

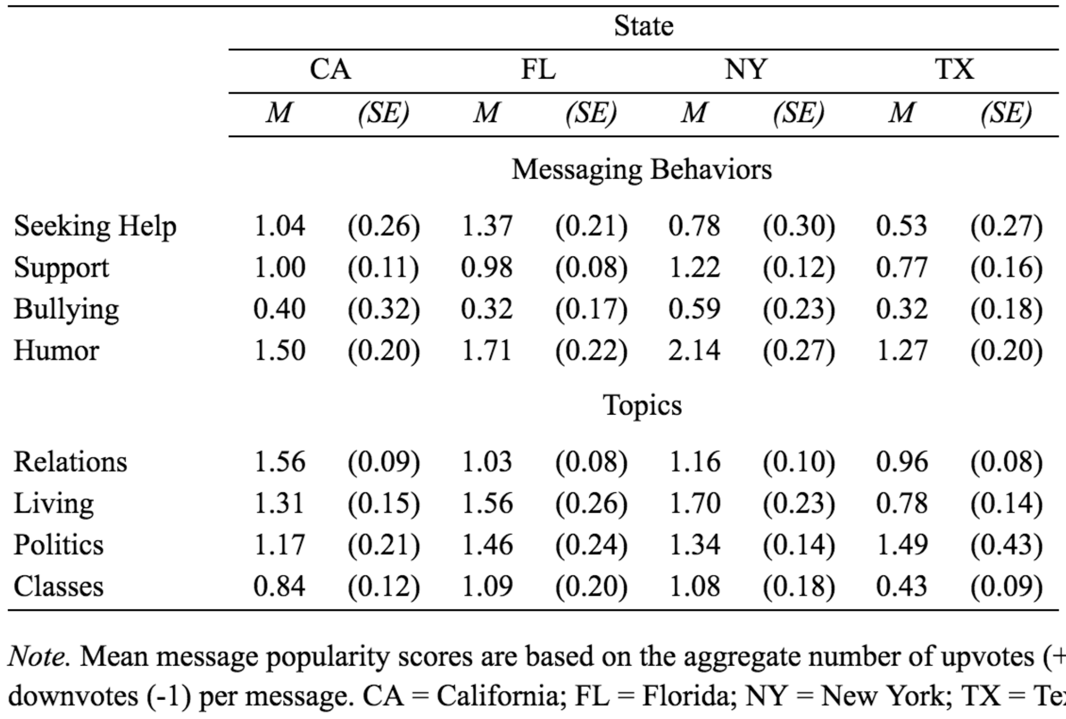
	State								Fisher's Exact P
	CA		FL		NY		TX		
	%	*(n)*	*%*	*(n)*	*%*	*(n)*	*%*	*(n)*	
	Messaging Behaviors								
Seeking Help	1.6	(70)	2.0	(94)	1.5	(65)	2.0	(70)	0.2
Support	4.2	(183)	8.1	(381)	5.5	(234)	2.5	(88)	<0.0001
Bullying	1.4	(61)	1.5	(68)	2.3	(98)	2.7	(93)	<0.0001
Humor	3.1	(140)	2.9	(134)	3.4	(144)	2.8	(98)	0.4
	Topics								
Relations	16.4	(730)	14.8	(689)	13.2	(562)	15.4	(535)	0.0004
Living	5.0	(224)	1.8	(83)	3.7	(157)	5.2	(180)	<0.0001
Politics	3.0	(133)	2.6	(122)	7.5	(317)	1.0	(35)	<0.0001
Classes	4.7	(208)	2.4	(114)	2.5	(150)	5.7	(198)	<0.0001

Note. CA = California; FL = Florida; NY = New York; TX = Texas

$MSE = 4.9$, $p < .001$, and a significant State-by-Topic interaction, $F(9, 4293) = 2.95$, $MSE = 4.9, p = .002$. We carried out Tukeys test to further investigate the significant main effects. For the effect of State, we found that Texas, the state with the lowest popularity scores overall, differed significantly from California, Florida, and New York, all $ps < .05$. Regarding the popularity of topics, school and classes was a significantly less popular topic than relationships and sex, college living, and politics, all $ps < .01$.

The significant State-by-Topic interaction indicates that states differ with respect to the relative popularity of topics. In order to identify patterns of topic popularity within each state, we carried ANOVAs with Topic as the single factor, separately for each state. These ANOVAs yielded a significant effect of Topic for California, $F(3, 1231) = 5.36$,

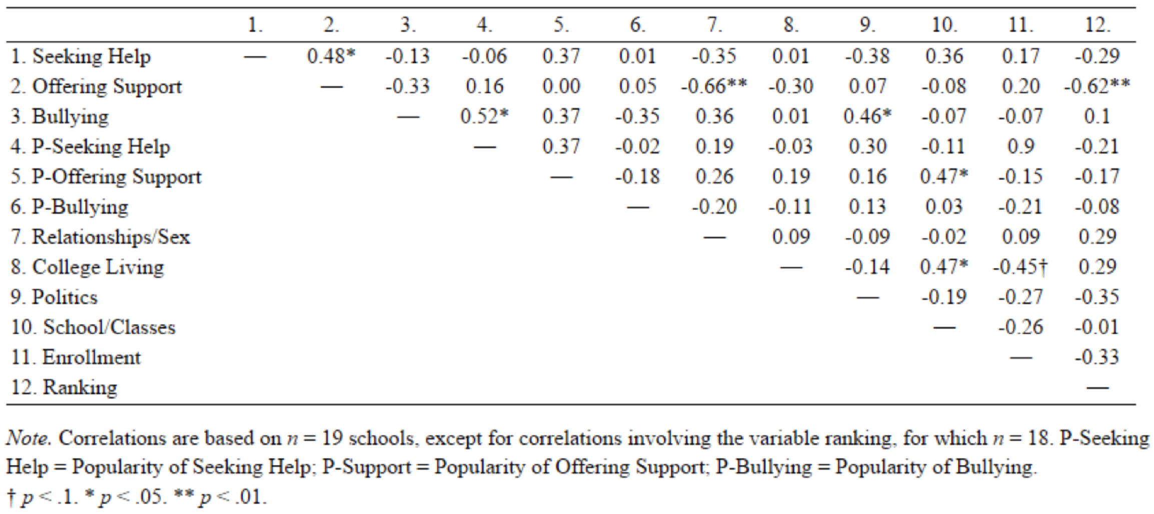
Table 5.3: Popularity of messaging behaviors and topics by State based on aggregation of up/down votes.



	State							
	CA		FL		NY		TX	
	M	*(SE)*	*M*	*(SE)*	*M*	*(SE)*	*M*	*(SE)*
	Messaging Behaviors							
Seeking Help	1.04	(0.26)	1.37	(0.21)	0.78	(0.30)	0.53	(0.27)
Support	1.00	(0.11)	0.98	(0.08)	1.22	(0.12)	0.77	(0.16)
Bullying	0.40	(0.32)	0.32	(0.17)	0.59	(0.23)	0.32	(0.18)
Humor	1.50	(0.20)	1.71	(0.22)	2.14	(0.27)	1.27	(0.20)
	Topics							
Relations	1.56	(0.09)	1.03	(0.08)	1.16	(0.10)	0.96	(0.08)
Living	1.31	(0.15)	1.56	(0.26)	1.70	(0.23)	0.78	(0.14)
Politics	1.17	(0.21)	1.46	(0.24)	1.34	(0.14)	1.49	(0.43)
Classes	0.84	(0.12)	1.09	(0.20)	1.08	(0.18)	0.43	(0.09)

Note. Mean message popularity scores are based on the aggregate number of upvotes (+1) and downvotes (-1) per message. CA = California; FL = Florida; NY = New York; TX = Texas

Table 5.4: Inter-correlations at the school level for messaging behavior frequencies, popularity of messaging behaviors, topic frequencies and school characteristics.



	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
1. Seeking Help	—	0.48*	-0.13	-0.06	0.37	0.01	-0.35	0.01	-0.38	0.36	0.17	-0.29
2. Offering Support		—	-0.33	0.16	0.00	0.05	-0.66**	-0.30	0.07	-0.08	0.20	-0.62**
3. Bullying			—	0.52*	0.37	-0.35	0.36	0.01	0.46*	-0.07	-0.07	0.1
4. P-Seeking Help				—	0.37	-0.02	0.19	-0.03	0.30	-0.11	0.9	-0.21
5. P-Offering Support					—	-0.18	0.26	0.19	0.16	0.47*	-0.15	-0.17
6. P-Bullying						—	-0.20	-0.11	0.13	0.03	-0.21	-0.08
7. Relationships/Sex							—	0.09	-0.09	-0.02	0.09	0.29
8. College Living								—	-0.14	0.47*	-0.45†	0.29
9. Politics									—	-0.19	-0.27	-0.35
10. School/Classes										—	-0.26	-0.01
11. Enrollment											—	-0.33
12. Ranking												—

Note. Correlations are based on *n* = 19 schools, except for correlations involving the variable ranking, for which *n* = 18. P-Seeking Help = Popularity of Seeking Help; P-Support = Popularity of Offering Support; P-Bullying = Popularity of Bullying.
† $p < .1$. * $p < .05$. ** $p < .01$.

$MSE = 5.39$, $p = .001$, and Texas, $F(3, 928) = 5.84$, $MSE = 3.17$, $p < .001$, but not for Florida, $F(3, 985) = 2.41$, $MSE = 4.91$, $p = .066$, or for New York, $F(3, 1149) = 2.34$, $MSE = 5.7$, $p = .072$. We followed up on the significant effects for California and Texas with Tukeys test. In California, school and classes were a less popular topic than relationships and sex, $p < .001$. In Texas, users liked messages about school and classes less than messages about relationships and politics, both $ps < .01$.

5.3.5 Interplay between Messaging Behaviors, Popularity of Behaviors, Topics, and School Variables

In this section we examine the relationship between the frequency of prosocial messages in which users seek help or offer support, the frequency of bullying messages, the popularity of these messaging behaviors, and the frequency of topics. The analysis was carried out at the school level. For each university we calculated mean messaging behavior frequencies, the corresponding mean popularity scores, and mean topic frequencies. The intercorrelations of these variables, together with two additional school variablesnumber of students enrolled and ranking ("Wall Street Journal/Times Higher Education College Rankings 2017," 2016) are summarized in Table 5.4.

We found that schools with a greater frequency of help-seeking messages also exhibit a greater frequency of messages offering support. Messages offering support were also sent more often on campuses where students post less about relationships and sex. Moreover, messages offering support were more popular at higher-ranking schools and at schools where students post more about classes. Two results concern the frequency of bullying messages. First, the frequency of bullying messages on Yik Yak was positively related to the popularity of messages seeking help. Second, bullying occurred more often on campuses where users post more about politics. Finally, we found that the frequency of posts about classes was positively related to the popularity of messages that offered support and to the frequency of posts about college living.

5.4 Discussion

Thanks to the growing popularity of social media across all segments of society, researchers now have a plethora of data sources at their disposal from which they can derive new insights about peoples attitudes, behaviors, and beliefs. The ability to observe social media users in near real-time holds particular promise in the domain of public health where rapid detection of health-relevant events and timely intervention are of essence. The aim of the present study was to explore whether information pertaining to college students health and well-being can be gleaned from their conversations on the anonymous Yik Yak social network. To this end, we analyzed the frequency and popularity of prosocial messages (seeking help, offering support, being funny) and bullying messages as well as the frequency and popularity of topics discussed online.

In our data set, prosocial messages were more frequent than bullying messages (10.1% vs. 1.9%). We observed significant regional differences in the frequency of messages offering support and of bullying messages. Most notable, Yik Yak users at Texas colleges sent the fewest supporting messages and the most bullying messages. This finding needs to be interpreted with caution in light of the relatively small number of messages and universities in our study. Nevertheless, it highlights a potentially problematic pattern of social media use among college students that future research may link to adverse health outcomes. Unsurprisingly, bullying messages were the least popular and funny messages the most popular among Yik Yak users, independently of what state they lived in.

In order to identify topics of Yik Yak messages, we relied on statistical modeling as an alternative to the subjective classification scheme recently used by Black and colleagues [19]. A subsequent analysis of topic prevalence revealed that relationships/sex was the most frequently discussed topic among college students. School and classes turned out to be the least popular topic, as measured by the number of up- and downvotes messages of this kind received. From an intervention point of view, regional differences in topic frequency and popularity matter because they offer campus representatives and health professionals clues on how to best engage a student population both online and offline. While the relative popularity of topics was similar across states, we found greater regional variation in the relative frequency of topics. For example, 7.5% of Yik Yak messages in the state of New York dealt with politics, but only 1% in Texas, and college living was addressed in 5% of messages in California, but in only 1.8% of messages in Florida.

With our final correlational analysis we wanted to learn more about factors that promote prosocial online behaviors and prevent cyberbullying at U.S. colleges. Several findings are noteworthy. At schools where students sought help more often in Yik Yak messages, messages offering support were also more frequent. Students may offer support in response to requests for help, but the reverse relationship is also conceivable: At schools where support is offered frequently, students may feel encouraged to ask for help. A higher prevalence of supporting messages also appears to be a characteristic of higher-ranking universities.

Why messages of support were sent more often at schools where relationships and sex were discussed less frequently is difficult to interpret and requires further investigation. It is also not clear why the popularity of messages offering support was positively related to the frequency of the school/classes topic.

Two results speak directly to the frequency of cyberbullying on college campuses. First, bullying was positively related to the popularity of messages seeking help. One interpretation for this finding is that students react prosocially to a higher prevalence of bullying by encouraging help-seeking behavior, although they did not appear to actually offer more support (the correlation between the frequency of supporting and bullying messages was negative and not significant). An alternative hypothesis is that certain prosocial messaging behaviors can trigger cyberbullying. Second, there was a higher incidence of bullying at schools where students discussed politics more frequently.

The major limitations of this study are to be seen in the small number of colleges and universities and the modest number of Yik Yak messages per school. We therefore caution against generalizing our findings until they can be replicated with larger samples. Our main intentions with this study were to learn about students online behaviors and interests from their posts on the Yik Yak social network, and, more specifically, to garner initial insights into conditions affecting prosocial and antisocial uses of social media. We believe that the findings reported here can be a stepping stone for further research.

Bibliography

[1] Google enterprise search. `https://enterprise.google.com/search/`.

[2] HomeUnion Inc. wensite. `http://www.homeunion.com`.

[3] Lexixnexis. `https://www.lexisnexis.com/totalpatent/`.

[4] Multiple Listing Services. `http://www.mls.com`.

[5] Real Estate Standards Organization. `http://www.reso.org`.

[6] Wall Street Journal/Times Higher Education College Rankings 2017. `www.timeshighereducation.com/rankings/united-states/2017`.

[7] Why a girl who was viciously bullied on Yik Yak now believes in the anonymous app's future. `www.businessinsider.com/elizabeth-long-was-bullied-on-yik-yak-2015-3`.

[8] Yik Yak chat app stirring up trouble in high schools. `www.cnn.com/2014/03/07/tech/yik-yak-app-high-school-problems/`.

[9] Alan R Aronson. Effective mapping of biomedical text to the umls metathesaurus: the metamap program. In *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 2001.

[10] Anastasios Arvanitis, Matthew Wiley, and Vagelis Hristidis. Efficient concept-based document ranking. In *EDBT*, 2014.

[11] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, 2010.

[12] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[13] Montserrat Batet, David Sánchez, and Aida Valls. An ontology-based measure to compute semantic similarity in biomedicine. *Journal of biomedical informatics*, 2011.

[14] Joachim Baumeister, Jochen Reutelshoefer, Fabian Haupt, and Karin Nadrowski. Capture and refactoring in knowledge wikis coping with the knowledge soup. 2008.

[15] Joachim Baumeister, Jochen Reutelshoefer, and Frank Puppe. Knowwe: a semantic wiki for knowledge engineering. *Applied Intelligence*, 35(3):323–344, 2011.

[16] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *VLDB*, volume 1, pages 119–128, 2001.

[17] Robert Baumgartner, Oliver Frölich, and Georg Gottlob. The lixto systems applications in business intelligence and semantic web. In *The Semantic Web: Research and Applications*, pages 16–26. Springer, 2007.

[18] Rokia Bendaoud, Yannick Toussaint, and Amedeo Napoli. Pactole: A methodology and a system for semi-automatically enriching an ontology from a collection of texts. In *Conceptual Structures: Knowledge Visualization and Reasoning*, pages 203–216. Springer, 2008.

[19] Thompson L. Black E, Mezzina K. Anonymous social media understanding the content and context of yik yak. In *Computers in Human Behavior*, 2016.

[20] Jordan MI. Blei DM, Ng AY. Latent dirichlet allocation. In *Journal of Machine Learning Research*, 2003.

[21] Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 2004.

[22] Sergey Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[23] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434. ACM, 2003.

[24] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *ACM ICML*, 2005.

[25] Chia Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *IEEE TKDE*, 2006.

[26] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic ranking of database query results. In *VLDB*, 2004.

[27] Hsinchun Chen, Ganesan Shankaranarayanan, Linlin She, and Anand Lyer. A machine learning approach to inductive query by examples: an experiment using relevance feedback, id3, genetic algorithms, and simulated annealing. Technical report, DTIC Document, 1998.

[28] Juan Cigarrán-Recuero, Joaquín Gayoso-Cabada, Miguel Rodríguez-Artacho, María-Dolores Romero-López, Antonio Sarasa-Cabezuelo, and José-Luis Sierra. Assessing semantic annotation activities with formal concept analysis. *Expert Systems with Applications*, 41(11):5495–5508, 2014.

[29] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM, 2004.

[30] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning concept hierarchies from text corpora using formal concept analysis. *J. Artif. Intell. Res.(JAIR)*, 24:305–339, 2005.

[31] Philipp Cimiano, Andreas Hotho, Gerd Stumme, and Julien Tane. Conceptual knowledge processing with formal concept analysis and ontologies. In *Concept Lattices*, pages 189–207. Springer, 2004.

[32] Kevyn Collins-Thompson, Paul Ogilvie, Yi Zhang, and Jamie Callan. Information filtering, novelty detection, and named-page finding. In *TREC*, 2002.

[33] Thomas M Cover and Joy A Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[34] Bruce Croft and John Lafferty. *Language modeling for information retrieval*, volume 13. Springer Science & Business Media, 2013.

[35] Gennaro L. Curcio G, Ferrara M. Sleep loss, learning capacity and academic performance. In *Sleep Medicine Reviews*, 2006.

[36] Ali Dasdan, Paolo D'Alberto, Santanu Kolay, and Chris Drome. Automatic retrieval of similar content using search engine query interface. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 701–710. ACM, 2009.

[37] Counts S Horvitz E. De Choudhury M, Gamon M. Predicting depression via social media. In *ICWSM*, 2013.

[38] Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, R Guha, Anant Jhingran, Tapas Kanungo, Kevin S McCurley, Sridhar Rajagopalan, Andrew Tomkins, et al. A case for automated large-scale semantic annotation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):115–132, 2003.

[39] Alexiei Dingli, Fabio Ciravegna, and Yorick Wilks. Automatic semantic annotation using unsupervised information extraction and integration. In *Proceedings of SemAnnot 2003 Workshop*, 2003.

[40] Mary-Lou Downie and Gill Robson. Automated valuation models: an international perspective. 2008.

[41] David Eppstein and Daniel S Hirschberg. Choosing subsets with maximum weighted average. *Journal of Algorithms*, 1997.

[42] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

[43] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 2003.

[44] Fernando Farfan, Vagelis Hristidis, Anand Ranganathan, and Michael Weiner. Xontorank: Ontology-aware search of electronic medical records. In *IEEE ICDE*, 2009.

[45] Miriam Fernández, Iván Cantador, Vanesa López, David Vallet, Pablo Castells, and Enrico Motta. Semantically enhanced information retrieval: an ontology-based approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2011.

[46] John Foley, Michael Bendersky, and Vanja Josifovski. Learning to extract local events from the web. 2015.

[47] Yanjie Fu, Yong Ge, Yu Zheng, Zijun Yao, Yanchi Liu, Hui Xiong, and Nicholas Jing Yuan. Sparse real estate ranking with online user reviews and offline moving behaviors. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 120–129. IEEE, 2014.

[48] Yanjie Fu, Guannan Liu, Spiros Papadimitriou, Hui Xiong, Yong Ge, Hengshu Zhu, and Chen Zhu. Real estate ranking via mixed land-use latent models. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 299–308. ACM, 2015.

[49] Yanjie Fu, Hui Xiong, Yong Ge, Zijun Yao, Yu Zheng, and Zhi-Hua Zhou. Exploiting geographic dependencies for real estate appraisal: A mutual perspective of ranking and clustering. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1047–1056. ACM, 2014.

[50] Maggs J. Galambos N, Dalton A. Losing sleep over it: daily variation in sleep quantity and quality in canadian students first semester of university. In *Journal of Research on Adolescence*, 2009.

[51] Giorgos Giannopoulos, Nikos Bikakis, Theodore Dalamagas, and Timos Sellis. Gontogle: a tool for semantic annotation and search. In *The Semantic Web: Research and Applications*, pages 376–380. Springer, 2010.

[52] Mona Golestan Far, Scott Sanner, Mohamed Reda Bouadjenek, Gabriela Ferraro, and David Hawking. On term selection techniques for patent prior art search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 803–806. ACM, 2015.

[53] Vidhya Govindaraju and Krishnan Ramanathan. Similar document search and recommendation. *Journal of Emerging Technologies in Web Intelligence*, 4(1):84–93, 2012.

[54] Siegfried Handschuh and Steffen Staab. Authoring and annotation of web pages in cream. In *Proceedings of the 11th international conference on World Wide Web*, pages 462–473. ACM, 2002.

[55] Siegfried Handschuh, Steffen Staab, and Fabio Ciravegna. S-creamsemi-automatic creation of metadata. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 358–372. Springer, 2002.

[56] Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. Semantic similarity from natural language and ontology analysis. *Synthesis Lectures on Human Language Technologies*, 2015.

[57] William Hersh, Chris Buckley, TJ Leone, and David Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *SIGIR94*, pages 192–201. Springer, 1994.

[58] Andrew Hogue and David Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *Proceedings of the 14th international conference on World Wide Web*, pages 86–95. ACM, 2005.

[59] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *ACM SIGKDD*, 2004.

[60] David Huynh, David R Karger, Dennis Quan, et al. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Semantic Web Workshop*, volume 52, 2002.

[61] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *CSUR*, 2008.

[62] Panagiotis G Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 394–405. VLDB Endowment, 2002.

[63] Mahler J. Who spewed that abuse? anonymous yik yak app isnt telling. In *The New York Times*, 2015.

[64] Rong Jin and Alexander Hauptmann. Learning to select good titlewords: An new approach based on reverse information retrieval. 2001.

[65] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, pages 81–93, 1938.

[66] Nadzeya Kiyavitskaya, Nicola Zeni, James R Cordy, Luisa Mich, and John Mylopoulos. Cerno: Light-weight tool support for semantic annotation of textual documents. *Data & Knowledge Engineering*, 68(12):1470–1492, 2009.

[67] Nadzeya Kiyavitskaya, Nicola Zeni, Luisa Mich, James R Cordy, and John Mylopoulos. Annotating accommodation advertisements using cerno. *Information and Communication Technologies in Tourism 2007*, pages 389–400, 2007.

[68] Paul A Kogut and William S Holmes III. Aerodaml: Applying information extraction to generate daml annotations from web pages. In *Semannot@ K-CAP 2001*, 2001.

[69] Vilius Kontrimas and Antanas Verikas. The mass appraisal of the real estate by computational intelligence. *Applied Soft Computing*, 11(1):443–448, 2011.

[70] John Krainer and Chishen Wei. House prices and fundamental value. *FRBSF Economic Letter*, 2004.

[71] Markus Krötzsch, Pascal Hitzler, and Guo-Qiang Zhang. Morphisms in context. In *Conceptual Structures: Common Semantics for Sharing Knowledge*, pages 223–237. Springer, 2005.

[72] Markus Krötzsch, Denny Vrandečić, and Max Völkel. Semantic mediawiki. In *The Semantic Web-ISWC 2006*, pages 935–942. Springer, 2006.

[73] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1):15–68, 2000.

[74] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 2010.

[75] Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 1998.

[76] Jae-Woo Lee and Doo-Kwon Baik. A model for extracting keywords of document using term frequency and distribution. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 437–440. Springer, 2004.

[77] Hugo Liu and Push Singh. Conceptneta practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.

[78] Xiaoyong Liu and W Bruce Croft. Cluster-based retrieval using language models. In *ACM SIGIR*, 2004.

[79] Zhiyuan Liu, Peng Li, Yabin Zheng, and Maosong Sun. Clustering to find exemplar terms for keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 257–266. Association for Computational Linguistics, 2009.

[80] Sanjay Kumar Malik, Nupur Prakash, and SAM Rizvi. Semantic annotation framework for intelligent information retrieval using kim architecture. *International Journal of Web & Semantic Technology (IJWest)*, 1(4):12–26, 2010.

[81] Diana Maynard. Multi-source and multilingual information extraction. *Expert Update*, 6(3):11–16, 2003.

[82] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2015.

[83] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[84] Genevieve B Melton, Simon Parsons, Frances P Morrison, Adam S Rothschild, Marianthi Markatou, and George Hripcsak. Inter-patient distance metrics using snomed ct defining relationships. *Journal of biomedical informatics*, 2006.

[85] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[86] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[87] Amit Kumar Mondal and Dipak Kumar Maji. Improved algorithms for keyword extraction and headline generation from unstructured text. *First Journal publication from SIMPLE groups, CLEAR Journal*, 2013.

[88] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *ACM WWW*, 2012.

[89] Chaitra H Nagaraja, Lawrence D Brown, and Linda H Zhao. An autoregressive approach to house price modeling. *The Annals of Applied Statistics*, pages 124–149, 2011.

[90] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2008.

[91] Catia Pesquita, Daniel Faria, Andre O Falcao, Phillip Lord, and Francisco M Couto. Semantic similarity in biomedical ontologies. *PLoS computational biology*, 2009.

[92] W Pirie. Spearman rank correlation coefficient. *Encyclopedia of statistical sciences*, 1988.

[93] Ana-Maria Popescu and Orena Etzioni. Extracting product features and opinions from reviews. In *Natural language processing and text mining*. Springer, 2007.

[94] Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, and Miroslav Goranov. Kim–semantic annotation platform. In *The Semantic Web-ISWC 2003*, pages 834–849. Springer, 2003.

[95] Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov. Kim-a semantic platform for information extraction and retrieval. *Natural language engineering*, 10(3-4):375–392, 2004.

[96] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 1989.

[97] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

[98] Sharkey KM Reen E Bond TL Raffray T Carskadon MA. Roane BM, Seifer R. What role does sleep play in weight gain in the first semester of university? In *Behavioral Sleep Medicine*, 2014.

[99] Joseph John Rocchio. Relevance feedback in information retrieval. 1971.

[100] Sebastian Rudolph, Johanna Völker, and Pascal Hitzler. Supporting lexical ontology learning by relational exploration. In *Conceptual Structures: Knowledge Architectures for Smart Applications*, pages 488–491. Springer, 2007.

[101] Guergana K Savova, James J Masanz, Philip V Ogren, Jiaping Zheng, Sunghwan Sohn, Karin C Kipper-Schuler, and Christopher G Chute. Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 2010.

[102] Barış Sertkaya. Ontocomp: A protege plugin for completing owl ontologies. In *The Semantic Web: Research and Applications*, pages 898–902. Springer, 2009.

[103] Moloud Shahbazi, Matthew Wiley, and Vagelis Hristidis. Iranker: Query-specific ranking of reviewed items. *http://www.cs.ucr.edu/~mshah008/publications/ItemsRanking.pdf*.

[104] Walid Shalaby and Wlodek Zadrozny. Patent retrieval: A literature review. *arXiv preprint arXiv:1701.00324*, 2017.

[105] Robert J Shiller. Arithmetic repeat sales price estimators. *Journal of Housing Economics*, 1(1):110–126, 1991.

[106] Mark D Smucker and James Allan. Find-similar: similarity browsing as a search tool. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 461–468. ACM, 2006.

[107] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

[108] Laura O Taylor. The hedonic method. In *A primer on nonmarket valuation*, pages 331–393. Springer, 2003.

[109] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 648–659. VLDB Endowment, 2004.

[110] Ivan Titov and Ryan McDonald. Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th international conference on World Wide Web*, pages 111–120. ACM, 2008.

[111] Peter D Turney. Learning algorithms for keyphrase extraction. *Information retrieval*, 2(4):303–336, 2000.

[112] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995.

[113] Becker C Vail-Smith K, Felts WM. Relationship between sleep quality and health risk behaviors in undergraduate college students. In *College Student Journal*, 2009.

[114] Maria Vargas-Vera, Emanuela Moreale, Arthur Stutt, Enrico Motta, and Fabio Ciravegna. Mnm: semi-automatic ontology population from text. In *Ontologies*, pages 373–402. Springer, 2007.

[115] Maria Vargas-Vera, Enrico Motta, John Domingue, Mattia Lanzoni, Arthur Stutt, and Fabio Ciravegna. Mnm: Ontology driven semi-automatic and automatic support for semantic markup. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 379–391. Springer, 2002.

[116] Ellen M Voorhees, Donna K Harman, et al. *TREC: Experiment and evaluation in information retrieval*, volume 1.

[117] Wang T Nika A Zheng H Zhao B. Wang G, Wang B. Whispers in the dark: analysis of an anonymous social network. In *Proceedings of the 14th Internet Measurement Conference (IMC)*, 2009.

[118] Henry Wasserman and Jerome Wang. An applied evaluation of snomed ct as a clinical vocabulary for the computerized diagnosis and problem list. In *AMIA Annual Symposium Proceedings*. American Medical Informatics Association, 2003.

[119] Kyle Williams, Jian Wu, and C Lee Giles. Simseerx: A similar document search engine. In *Proceedings of the 2014 ACM symposium on Document engineering*, pages 143–146. ACM, 2014.

[120] Moroz TL Williams PG. Personality vulnerability to stress-related sleep disruption: Pathways to adverse mental and physical health outcomes. In *Personality and Individual Differences*, 2009.

[121] Ian H Witten, Gordon W Paynter, Eibe Frank, Carl Gutwin, and Craig G Nevill-Manning. Kea: Practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, pages 254–255. ACM, 1999.

[122] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics.

[123] Xiaochun Yang, Yaoshu Wang, Bin Wang, and Wei Wang. Local filtering: Improving the performance of approximate queries on string collections. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 377–392. ACM, 2015.

[124] Yin Yang, Nilesh Bansal, Wisam Dakka, Panagiotis Ipeirotis, Nick Koudas, and Dimitris Papadias. Query by document. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 34–43. ACM, 2009.

[125] Lewis B. Young S, Rivers C. Methods of using real-time social media technologies for detection and remote monitoring of hiv outcomes. In *Preventive Medicine*, 2014.

[126] Zhongwu Zhai, Bing Liu, Hua Xu, and Peifa Jia. Clustering product features for opinion mining. In *ACM WSDM*, 2011.

[127] Zhenya Zhang and Honemei Cheng. Keywords extracting as text chance discovery. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*, volume 2, pages 12–16. IEEE, 2007.

[128] Zhu Zhang. Weighing stars: Aggregating online product reviews for intelligent e-commerce applications. *IEEE Intelligent Systems*, 2008.