OPINION DRIVEN DECISION SUPPORT SYSTEM

BY

KAVITA ANNAPOORANI GANESAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor ChengXiang Zhai, Chair & Director of Research
Professor Jiawei Han
Associate Professor Kevin C. Chang
Associate Professor Ellen Riloff, University of Utah
Dr. Evelyne Viegas, Microsoft Research

ABSTRACT

Opinions on the web present a wealth of information that can be leveraged in our day to day decision making tasks ranging from which product to purchase to which doctor to consult for a particular ailment. Due to the large volume of opinions available from different sources across web, digesting all the available opinions is a time consuming process which can severely impair user productivity. As a result, these valuable opinions become more of a hindrance than a help in decision making scenarios especially those involving a large number of entities.

Most existing work on solving this general problem has been focused on summarizing opinions to help users better digest all the opinions. Unfortunately, in many decision making scenarios, the number of entities in consideration could be quite large. Thus, making decisions by reading summaries alone would still be inefficient as you would need to read summaries of different entities thoroughly. Further, as most of the opinion summarization systems focus on generating summaries that are highly structured, these summaries lack details that can aid decision making.

In this thesis, we propose a more efficient way of leveraging opinions, that is to combine the strengths of search technologies with opinion analysis and mining tools to provide a powerful decision making platform. This special platform is called an Opinion-Driven Decision Support System (ODSS) - a platform that enables users to *find* and *analyze* entities of interest based on opinions of other web users.

We study three important problems of the ODSS, encompassing search, analysis and data acquisition. First, in providing a useful search capability, we study the problem of *Opinion-Based Entity Ranking* - where entities are ranked based on a set of user specified opinion preferences. Then, in providing analysis tools to aid decision making, we study *Abstractive Summarization of Opinions*, where unstructured summaries highlighting key opinions are generated on any arbitrary topic. In order to enable the search and analysis components, opinionated content is imperative. Hence, in the third part of this thesis, we attempt to study the problem of *Opinion Acquisition*.

In the first part of this thesis, we investigate the use of robust retrieval models and extensions of it for the task of Opinion-Based Entity Ranking. Our evaluation, in two different domains, shows that the proposed methods can be directly applied to rank different types of entities for which opinions are available. Our

user study further shows that the proposed evaluation strategy used for this ranking task is effective and can be used in future evaluations.

In the second part of this thesis, we study two flavors of summarization techniques for generating unstructured opinion summaries. We focus on using unsupervised techniques to generate *abstractive summaries* that are concise, fairly well-formed and convey key opinions in text. Through a series of experiments, we have shown that the summaries generated through the proposed techniques are indeed compact, readable and informative. Our techniques are also practical as we rely very little on external resources and the methods are not bound to the domain they were tested in.

As part of the final research question, we focus on automatic collection of online reviews. We propose a lightweight, unsupervised framework for discovering review pages of arbitrary entities leveraging existing Web search engines. We use a novel information network called the FetchGraph to help with collecting review pages in an efficient manner. The proposed methods were evaluated in three domains and results show that the proposed approach is capable of finding entity specific review pages with reasonable accuracy and efficiency.

To my dad, for helping me realize my true potential.

ACKNOWLEDGMENTS

I would first like to express my heart felt gratitude to my advisor Prof. ChengXiang Zhai who is a great mentor and an excellent teacher. When I first joined UIUC I had very limited experience on how to actually conduct research and come up with research ideas. As time went on, Professor Zhai continuously taught me to think like a researcher by showing me how to come up with practical but elegant solutions to interesting problems and pushing me to think years ahead of time. Prof. Zhai was also extremely understanding with my personal situation and I am very grateful to him for providing me with as much flexibility that I needed to be a successful student and researcher during the course of my Ph.D. His guidance has positioned me to where I am in my career today.

I wish to also express my thanks to the other members of my committee, Prof. Jiawei Han, Prof. Kevin Chang, Prof. Ellen Rilof and Dr. Evelyne Viegas for all their useful suggestions in improving my thesis work. They were all extremely co-operative and flexible in helping me set a date for my Preliminary and Final exam and I truly appreciate their support. I would also like to thank Prof. Andrew Gordon from the University of Southern California for introducing me to the world of research in Natural Language Processing and Text Mining. I thank my fellow lab mates in the TIMan group for all the stimulating discussion and knowledge sharing on various topics over the years. Special thanks to Parikshit Sondhi and V.G. Vinod Vydiswaran for helping me out with remote participation in meetings.

Finally, I am very grateful to my family, whose love and support kept me motivated in order to reach the end. I especially thank my husband for helping me out tremendously in many areas of our lives in order for me to complete my Ph.D. project on time.

TABLE OF CONTENTS

1 INTRODUCTION

The deployment of Web 2.0 technologies has led to rapid growth of various opinions and reviews on the web, such as reviews on products and opinions about people. The vast amount of opinions expressed by experts and ordinary users can be very useful to help people make all kinds of decisions, ranging from what to buy to what treatment to choose for a disease. For example, shoppers at Amazon[1] typically would read the reviews about a product before buying it, and travelers may rely on opinions about hotels on Tripadvisor[2] to help them choose an appropriate hotel at the destination. It has been shown that 77% of online shoppers use reviews and ratings when making a purchase decision[3].

Unfortunately, the vast amount of online opinions has also made it difficult for users to digest all the opinions about a specific topic or entity. Consider the task of a user shopping online for an mp3 player to purchase. Not only is there a large selection of mp3 players, each mp3 player may have hundreds of associated customer reviews which can be found in a variety of different sources. This makes it a challenge for users to actually understand the underlying sentiments about each product and as a result user productivity is impaired. Thus, the task of developing computational techniques to help users digest and exploit all the opinions is a very important and interesting research challenge.

Most existing work on leveraging opinions has been focused on *summarizing* opinions about an entity or a topic to help users better digest all the opinions. Unfortunately, in many decision making scenarios, the choices are a plenty and summaries alone will not be sufficient as this demands users to explore the summaries for all entities. This becomes especially problematic when the user's selection space is quite large to start with (e.g. selecting a hotel in New York City). Further, since most of the opinion summarization systems focus on generating summaries that are highly structured, these summaries lack sufficient details to actually aid decision making tasks. Thus, a much broader set of tools and techniques are needed to leverage online opinions in a more *effective* and *efficient* manner so as to aid decision making.

Search technologies have long helped users find all sorts of documents and entities based on topics. However, with respect to decision making, a user is often interested in finding entities based on key attributes (e.g. price, brand) and the opinions about each entity. While searching based on structured attributes

[1] http://www.amazon.com
[2] http://www.tripadvisor.com
[3] http://www.mediapost.com/publications/

is supported by some vertical search engines, searching based on the opinions of other users has not been previously explored. We believe that a more efficient way to leverage online opinions is to extend search technologies to find entities based on opinions and combine this with opinion analysis and mining tools to provide a powerful opinion-driven decision making platform. Such a platform is useful in several ways. First, a search capability based on a user's *opinion requirements* would enable users to find entities of interest without the hassle of exploring *all* opinions or opinion summaries. This would also help narrow the user's selection space, making it more manageable. Users can then focus on using the analysis tools to select the one entity that satisfies all their requirements. For example, in the case of a user finding a hotel at a destination, the user may only want to consider hotels that are said to be *clean* and have *good breakfast*. Such a requirement can be specified using a specialized search technology that would attempt to find hotels that satisfy this requirement. Using this smaller list of hotels, user's can further narrow down into hotels of their choice by using the available analysis tools or adding more requirements to their search criteria. We call this synergistic platform an Opinion-Driven Decision Support System (ODSS) - a system that enables users to find and analyze entities of interest (e.g. products, people and businesses) based on the opinions of other web users.

Such a decision platform has not been used in the commercial world nor has it been previously studied by researchers in the field. This proposed platform is very different from traditional search engines such as Google which can only help people find *documents* based on *topics* rather than *entities* based on *opinion oriented requirements* on those entities. Further, since traditional search engines are not designed to aid decision making, these search engines do not offer any type of analysis tools beyond a summary of the search results. One would think that analysis tools would be readily available on more specialized search engines such as Google Product Search[4] and Hotels.com[5]. However, these systems only provide search filters based on established attributes (e.g. *price*, *name*, *brand*) and fall short of any opinion related analysis or comparison tools. The ODSS thus fulfills deficiencies of current systems by providing a special platform to facilitate decision making.

Our envisioned ODSS platform is made up of four core components: *(1) Search capabilities*, *(2) Analysis tools*, *(3) Data* and *(4) Presentation*. These components would enable the development of a large-scale opinion driven decision support platform. The ODSS opens up a variety of interesting research challenges in the areas of Information Retrieval, Text Mining and Natural Language Processing. Figure 1.1 outlines some of these challenges. In this thesis, we focus on three important problems of the ODSS, encompassing search, analysis and data. First, in providing a useful search capability, we study the problem of *Opinion-Based Entity Ranking* - where entities are ranked based on a

[4]http://www.google.com/prdhp
[5]http://www.hotels.com

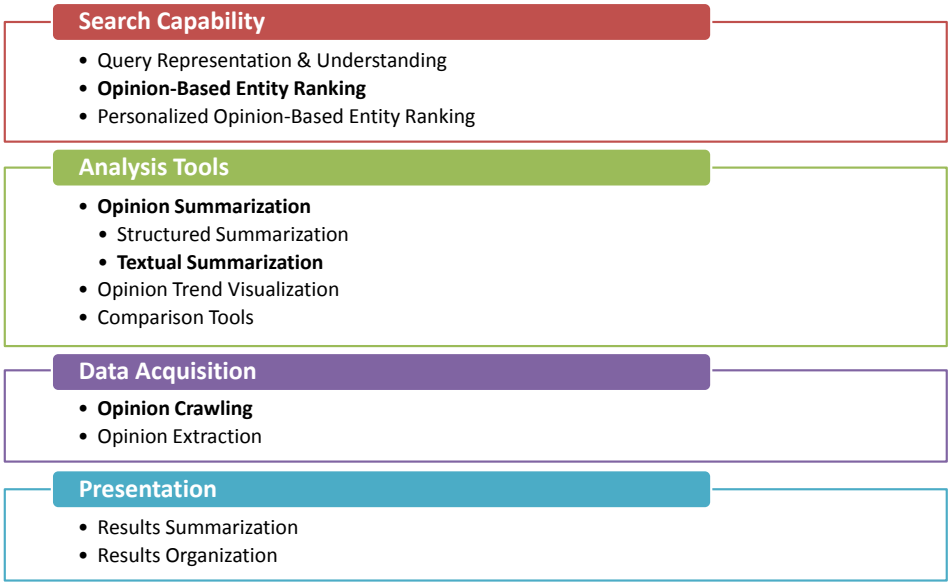

Figure 1.1: Research questions related to building an Opinion-Driven Decision Support System. Bolded fonts indicate questions that are addressed in this thesis.

set of user specified opinion preferences. Second, in providing analysis tools to further aid decision making, we study *Abstractive Summarization of Opinions*, where we propose practical approaches to generating unstructured summaries highlighting key opinions in text. In order to enable both the search and analysis components, opinionated content is imperative. Hence, in the third part of this thesis, we attempt to study the problem of *Opinion Acquisition*. Figure 1.2 shows how each of these research questions fit into the framework of an Opinion-Driven Decision Support System. In the next Section, we discuss the details of these research questions.

1.1 Research Questions

This thesis is divided into four sub-parts where the first three parts are research questions mentioned in the previous section and the final part is a web prototype. The first part of this thesis is about enabling the search for entities based on the user's requirements. This capability is achieved through the study of *Opinion-Based Entity Ranking* which enables users to find entities based on a set of keyword based preferences. Then, in the second part of this thesis, the goal is to provide analysis tools to aid decision making. For this, we study several techniques related to *Abstractive Summarization of Opinions*. The third part is about enabling large-scale opinion collection that will support the search and analysis components. For this, we propose to study the crawling of opinions

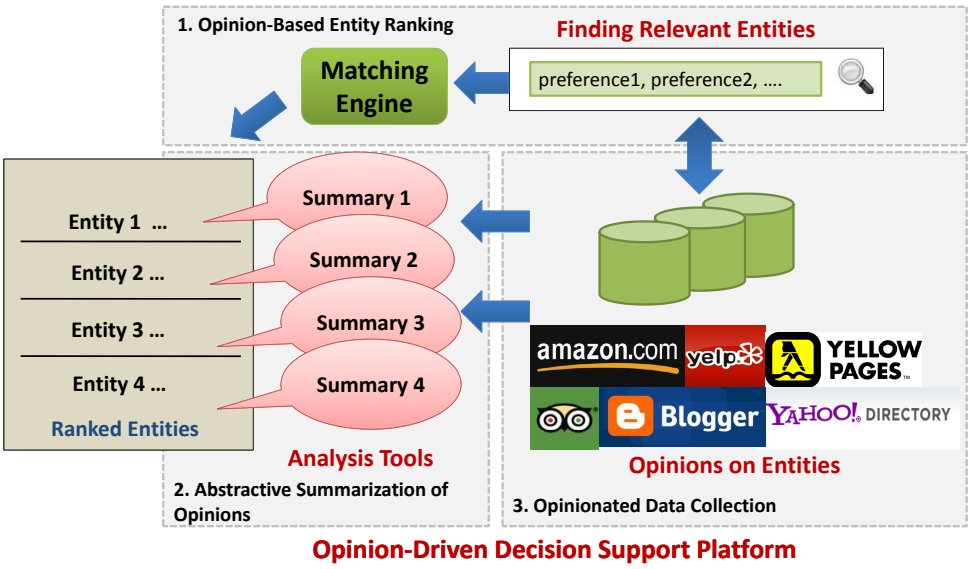

Figure 1.2: The core components of this thesis (numbered). Diagram shows how each component fits into the framework of an Opinion-Driven Decision Support System (red fonts).

in the form of reviews. As the final part of this thesis, we combine ideas from the first three parts and develop a web-based prototype in the context of hotel search. Figure 1.2 shows how each of the research questions in this thesis fits into the framework of an Opinion-Driven Decision Support System.

1.1.1 Part 1: Opinion-Based Entity Ranking

The goal of the search functionality in the envisioned platform is to help users find entities of interest based on their key requirements. Since a user is often interested in choosing an entity based on opinions on that entity, a system that ranks entities based on a user's personal preferences would provide a more *direct support* for a user's decision-making task. For example, in the case of finding hotels at a destination, a user may only want to consider hotels where other people thought was *clean*. By finding and ranking hotels based on how well it satisfies such a requirement would significantly reduce the number of entities in consideration, facilitating decision making. Unlike traditional search, the query in this case is a set of preferences and the results is a set of entities that match these preferences. The challenge is to accurately match the user's preferences with existing opinions in order to recommend the best entities. While extensions of existing opinion mining techniques can be leveraged to rank entities based on opinions, most of these techniques pose practical limitations where most techniques are domain dependent, some require supervision and some only allow a limited set of aspects of an entity to be queried. Our goal is to propose a solution that would easily work in any domain and would scale to large amounts

of data. Thus in this thesis, we explore the use of robust information retrieval models and extensions of it for this Opinion-Based Entity Ranking task [1].

1.1.2 Part 2: Abstractive Summarization of Opinions

Opinion summaries play a critical role in helping users analyze the entities in consideration. Users are often looking out for major *concerns* or *advantages* in selecting an entity. Thus, a summary that can quickly highlight the key opinions about the entity would significantly help exploration of entities and aid decision making. Although opinion summarization has been long explored, most techniques have focused on generating structured summaries on a fixed set of topics, making the summaries rather restrictive thus lacking the level of detail needed to aid decision making. Further, existing opinion summarization techniques are inflexible in that they are mostly domain dependent and often need some form of supervision. To address these limitations, we explore *unsupervised* and *domain-independent* techniques [2, 3] to generate textual summaries of opinions. Our goal is to generate summaries that are concise, readable and informative. We focus on leveraging the redundancies in opinionated content to generate summaries that are more *abstractive* as we believe that abstractive summaries have the potential of being more concise and less redundant than extractive summaries. Further, we aim at making these techniques flexible such that the summaries can be displayed on various screen sizes as well as not be restricted to a set of fixed topics.

1.1.3 Part 3: Opinion Acquisition

To support accurate search and analysis based on opinions, opinionated content is imperative. Relying on opinions from just one specific source not only makes the information unreliable, but also incomplete due to variations in opinions as well as potential bias present in a specific source. Although many applications rely on large amounts of opinions, there has been very limited work on collecting and integrating a complete set of opinions. In this thesis, we focus on crawling a comprehensive set of opinion containing pages pertaining to an entity. As opinions in the form of user reviews alone make up a big portion of online opinions, we narrow our focus of opinion crawling to review crawling. To make this crawler usable in practice, our goal is to ensure that the crawler is (1) general enough to collect reviews on any type of entity, (2) scalable to a large number of entities and (3) useful to client applications even after the crawl process is complete (e.g. to answer application related questions).

1.1.4 Part 4: Demo

To showcase the value of an Opinion-Driven Decision Platform, we build a prototype system (`http://www.findilike.com`) [4] that implements some of the ideas presented in this thesis. This prototype will be implemented in the context of hotel search. The system that we build will enable future research and can open up new research problems for further investigation. Here are some ways in which the system will benefit the research community:
(1) Query logs from the system can be used to form a real *test collection*. This test collection will be helpful in further improving the methods for ranking entities based on preferences.
(2) The system can be used as a *test platform* to test new research ideas such as new tools for opinion analysis, new retrieval models and so on.
(3) User interactions with the system will bring to light the *features* that are important to end users and which interfaces are most effective for decision making.

1.2 Thesis Materials

The content of this thesis is based upon a number of publications, self compiled data sets and demos. The related materials are as listed below:

Publications:
[3] Kavita Ganesan, ChengXiang Zhai and Evelyne Viegas. Micropinion Generation: An Unsupervised Approach to Generating Ultra-Concise Summaries of Opinions, Proceedings of the 21st International Conference on World Wide Web (WWW '12), 2012

[4] Kavita Ganesan and ChengXiang Zhai. FindiLike: Preference Driven Entity Search, Proceedings of the 21st International Conference on World Wide Web (WWW '12), 2012

[1] Kavita Ganesan and ChengXiang Zhai. Opinion-Based Entity Ranking, Information Retrieval, 2012

[2] Kavita Ganesan, ChengXiang Zhai and Jiawei Han. Opinosis: A Graph Based Approach to Abstractive Summarization of Highly Redundant Opinions, Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10), 2010

Released Datasets:
[5] Opinosis Dataset. Dataset containing sets of related sentences extracted from user reviews.

[6] OpinRank Review Dataset. Dataset containing full reviews for cars and hotels collected from Tripadvisor.com (259,000 reviews) and Edmunds.com (42,230 reviews).

Demos:

[2] Opinosis Summarizer: Java based text summarizer for opinions.
`http://kavita-ganesan.com/opinosis-summarizer-library`

[4, 7] FindiLike Web System: Web prototype of envisioned Opinion-Driven Decision Support System used in the context of hotel search.
`http://www.findilike.com`

2 BACKGROUND

In this thesis, we propose an Opinion Driven Decision Support System comprising of *search*, *analysis* and *data collection* components. We focus on solving key problems related to building such a decision support platform ranging from providing the ability to rank entities based on opinions to opinion acquisition tasks for compiling a comprehensive set of opinions. In this section, we will systematically survey related work in conjunction with the three main components of this thesis. We start with the search component, where we draw similarities and point out differences of our search task with existing work on entity retrieval, expert finding, opinion retrieval and multifaceted search (Section 2.1). Then, we move on to the second component on analysis where we propose practical approaches to abstractive summarization of opinions (Section 2.2). Here, we briefly describe existing methods in opinion summarization and then discuss text summarization techniques that are closest to our task. Then, for the third component on data collection (Section 2.3), we outline common web crawling strategies and describe how our task on opinion crawling differs from existing work.

2.1 Opinion-Based Entity Ranking

One of the core components of the proposed ODSS platform, is the ability to find and rank entities based on the opinions expressed on these entities. While searching based on structured attributes is supported by some vertical search engines such as Google Product Search[1], searching based on the opinions of other users has not been previously explored commercially or by any existing work. We believe that an efficient way to leverage online opinions is to extend existing search technologies to enable entity finding based on opinions. Conceptually, this task may seem similar to other tasks such as expert finding, opinion retrieval, multi-faceted search and so on. However, there are several key differences that makes our ranking task unique. In the next few sub-sections, we briefly describe some of the related work in existing literature and discuss the important similarities and differences between our ranking task and these areas of study.

[1] http://www.google.com/prdhp

2.1.1 Expert Finding

Expert finding is a special case of entity retrieval where the goal is to retrieve a ranked list of people or experts who are knowledgeable on a given topic using information retrieval technology. Expert finding has been addressed from different viewpoints, including expertise retrieval, which takes a mostly system-centered approach, and expertise seeking, which studies related human aspects.

To reflect the growing interest in entity ranking in general and expert finding in particular, the Text REtrieval Conference, TREC introduced an expert finding task at its Enterprise track in 2005 [8]. At this track, it emerged that there are two principal approaches to expert finding [8, 9, 10, 11] - the candidate model and the document model. Candidate-based approaches (also referred to as profile-based methods) build a textual representation of candidate experts, and rank them based on a query/topic, using traditional ad-hoc retrieval models. With the document model on the other hand, the goal is to first find documents which are relevant to the topic, and then locate the associated experts. Over the years, various refinements have been accomplished by building on either the candidate or the document models [12, 13, 14, 15].

While some of the ideas related to expert finding can be applied to the opinion-based entity ranking task, our ranking task is different in that each entity is represented by its corresponding textual opinions rather than a set of user expertise. Further, in our ranking task, the query is not just a topic of interest but rather a set of opinion related preferences such as *long battery life and excellent sound quality* (in the case of finding an mp3 player to purchase). Thus, we can expect the query to contain opinion indicating words such as 'excellent'. To accurately match the user's preferences with the existing opinions, special query understanding techniques would thus be essential. In addition to all of that, unlike expert finding, where the goal is to rank a special type of entity (people or experts), with opinion-based entity ranking, we can rank any arbitrary entity type as long as it is supported by opinion containing texts.

2.1.2 Opinion Retrieval

Opinion retrieval was first explored in the TREC Enterprise Track on email search. The goal of opinion retrieval is to identify and rank blog posts expressing an opinion regarding a given topic. The idea is to test the ability to find opinion expressing posts as this is essential to specialized search engines such as blog search engines. Opinion retrieval systems [16, 17, 18] are very similar to traditional retrieval systems in that they are usually built on top of standard retrieval models. The standard retrieval models are used to find documents with relevant content, and then opinion analysis is done on the retrieved content to identify the opinion containing documents.

While opinion retrieval is about identifying documents containing opinions

about a certain topic, opinion-based entity ranking assumes that we already have the opinionated content for a given entity. The task is to thus use all the available opinions to rank entities based on a user's preferences in the order of likelihood that the entity matches the user's preferences.

2.1.3 Multifaceted Search

Faceted search, also called *faceted navigation* or *faceted browsing*, allows users to explore and find information that they need by filtering or navigating with the help of some pre-determined facets [19]. The users often provide a very general query (some systems do not support queries), and then they use the various facets to navigate through the results until the items of interest are found. In other words, the goal is to connect users to items that are of most interest to them.

While the goal of faceted search and ours is similar, the paradigm is different. First of all, in our setup, users find entities based on unstructured text containing opinions of other users rather than structured or categorical data (often used in faceted navigation). In addition, our focus is more on the keyword based preferences in the query that allows users to specify their interest on various facets. For example, a user who is looking for a laptop with a specific criteria, would provide a query such as *Lenovo, very light, bright screen.* In such a query, the facets are actually implicit where in this case the facets being queried are *brand*, *weight* and *screen*. In traditional faceted navigation, these facets are explicitly defined and are usually fixed. Thus, our idea can be considered an *ad-hoc faceted navigation* or a *personalized faceted navigation*[20] system. This new idea can be combined with 'traditional' faceted navigation to provide a powerful search system that can greatly improve user productivity.

2.1.4 Rating Prediction and Decomposition

In recent years, there has been much work in trying to decompose reviews to make aspect based rating predictions [21, 22, 23]. This line of work is closely related to ours as, once we obtain ratings on different aspects, we would be able to rank entities based on their ratings in the aspects interesting to a user. With this setup, the problem would then be to rank the entities by its aspect based ratings depending on which set of aspects the user is interested in. This approach, however, has some practical limitations. First, most of these approaches assume a fixed number of aspects on a given entity. It is not only impractical to define or mine a set of aspects for each category of entities (e.g. politicians: *approval rating, character*; laptops:*battery life, screen*), but a fixed number of aspects would also severely limit the type of queries a user could issue. More importantly, all the work in this line, require some supervision in that they require the availability of ratings associated with reviews, which is not

always present. This is especially true if the opinions are to be obtained from completely unstructured sources such as blog or micro-blog sites.

Since the platform envisioned in this thesis is meant to be scalable and flexible, we assume limited knowledge on the queriable aspects and focus on leveraging robust retrieval models to find entities that closely match a user's unstructured preferences.

2.2 Abstractive Summarization of Opinions

Research in opinion summarization has been quite extensive with much of the focus being on generating structured summaries of opinions. Early opinion summarization systems focused on predicting the overall sentiment class (positive or negative) on a given piece of text [24, 25, 26, 27]. In later years, this definition was generalized to a multi-point rating scale where ratings are predicted on a set of aspects [22, 28, 23, 29, 30, 31, 32]. In contrast to structured summarization approaches, studies addressing unstructured summarization of opinions are notably fewer and *abstractive* approaches are less common than *extractive* approaches [33]. In extractive based techniques, the sentences or phrases deemed most important by the system are included in the summary. Abstractive summarization in the strictest sense involves condensing and rephrasing sections of the source document and is often much harder to achieve.

As mentioned in Section 1.1.2, in the second part of this thesis, we focus on generating textual summaries of opinions. Our goal is to use unsupervised techniques (i.e. to provide greater flexibility) to generate summaries that are concise, readable and can highlight key opinions in text. To the best of our knowledge, no previous work has studied the generation of *concise* and *abstractive* summaries using *unsupervised techniques* such as those presented in this thesis. Also, unlike existing methods that use a purely extractive approach or a highly complex abstractive approach, we attempt to use a shallow *abstractive* approach using only the existing text and minimal dependency on external resources.

The work closest to ours is perhaps the work of Branavan et. al [34] where a keyphrase extraction model was implemented to generate a set of opinion summarizing phrases. This model is based on a hierarchical Bayesian model, reusing existing pros and cons phrases available from the web to train the keyphrase extraction model. Unlike this supervised approach, our methods are unsupervised and domain independent where we try to generate concise and informative summaries using only the existing text, the inherent redundancies in opinions and some additional resources such as a publicly available Web N-Gram model.

Carenini et. al [35, 36] compared both the use of extractive and abstractive summarization methods for opinion summarization. A key difference between our abstractive approach and theirs is that we focus on generating *concise sum-*

maries (set of short phrases) using a domain-independent and lightweight approach, while they focus on generating paragraph level summaries made up of *full length sentences* using a complex natural language generation architecture. Other types of textual opinion summaries include contrastive summarization [37, 38] where the summaries are meant to highlight contradicting sentence pairs in opinionated text. Note that these summaries are not meant to summarize key opinions in text.

2.3 Opinion Acquisition

While much research has been done on mining and summarizing existing opinions [39, 40, 41, 42, 2, 3], there is limited work on automatically collecting a comprehensive set of opinions. Any form of opinion analysis is currently performed on a small portion of opinion containing documents. The problem with relying on opinions from just one or two sources is data sparseness and source related bias which could result in inaccuracies in information presented to the end user. It is thus crucial to have an automatic method to collect a large number of opinions from a variety of sources.

The concept of focused crawling was introduced by Chakrabarti et. al. [43] where the goal is to download only web pages that are relevant to a query or set of topics. Rather than collecting and indexing all accessible web documents, a focused crawler analyzes its crawl boundary to find the URLs that are likely to be most relevant for the crawl, and avoids irrelevant regions of the web. While early focused crawling methods were based on simple heuristics [44, 45], most topical crawlers in literature are predominantly supervised machine learning based methods [43, 46, 47, 48, 49, 50]. There are some key commonalities amongst these topical crawlers. First, topical crawlers are primarily interested in the relevance of a page to a topic. While initially topical relevance was determined using simple heuristics, topical crawlers generally rely on classifiers that require labeled examples. Next, in most topical crawlers, URLs on pages considered 'relevant' are prioritized for further crawling. The three most common concepts used for URL prioritization are (1) *link context* - the lexical content that appears around the given URL in the parent page (2) *ancestor pages* - the lexical content of pages leading to the current page and (3) *web graph* - the structure of the Web subgraph around the node (page) corresponding to the given URL.

While conceptually our task is similar to a traditional topical crawling task, there are some key differences that makes our task unique. In traditional topical crawling, relevance has mostly to do with how relevant a page is to the *topic of interest* regardless of content type. In our task however, the goal is to collect review pages specific to a set of entities, and thus relevance is about (1) if a candidate page is a review page and (2) if a candidate page is relevant to one

of the target entities. Next, in order to collect reviews for arbitrary entities the approach should be general enough to work across domains. Most topical crawlers however are domain dependent as they are trained on data from the domain of interest.

Perhaps the work of Vural et. al [51] is closest to our work where the broad goal is to discover opinion containing documents on the web. However, in their work there is no need for an opinion containing page to be relevant to a specific topic or entity as long as the content is subjective. The key idea used by Vural et. al is to prioritize discovered URLs based on their predicted sentiment scores so that the crawl is focused towards subjective content. The crawl path is similar to that of general web crawling (long crawl). We focus on a short crawl because we use search engine results that are specific to the target entity. Thus, the results are already quite close to what we need and the challenge is more about finding relevant review pages around the neighborhood of the search results with reasonable accuracy and efficiency. In summary, none of the previous methods was designed to solve our problem, and none of them can crawl reviews about an arbitrary entity efficiently.

3 OPINION-BASED ENTITY RANKING

In this chapter, we investigate the problem of Opinion-Based Entity Ranking, that will provide a search functionality to help users find entities of interest based on their personal preferences. Unlike traditional search, the query in this case is a set of preferences and the results is a set of entities that match these preferences. The challenge is to accurately *match the user's preferences* with *existing opinions* in order to recommend the best entities. This type of search capability significantly reduces the user's initial selection space thus making the decision making task more manageable.

3.1 Introduction

The era of Social Computing has kindled massive growth of opinions and reviews on the web, including reviews on businesses, products and opinions about people. Let us just consider reviews of movies. On yahoo's directory listing[1], the number of movie review sites alone is nearing two hundred. This number does not even include the growing number of blogs or social networking sites where people have the ability to freely express opinions about movies.

The vast amount of opinions expressed by experts and ordinary users can be very useful to help people make all kinds of decisions, ranging from what to buy to what treatment to choose for a disease. For example, shoppers at Amazon[2] typically would read the reviews about a product before buying it, and travelers may rely on opinions about hotels on Tripadvisor[3] to help them choose an appropriate hotel at the destination. It has been shown that 77% of online shoppers use reviews and ratings when making a purchase decision[4].

Unfortunately, the abundance of opinions also poses challenges in digesting all the opinions about an entity or a topic. For example, a popular product such as the *iPhone* may have hundreds of reviews on Amazon.com, and popular hotels like *Marriott* or *Hilton* may have over five hundred reviews on Tripadvisor. Thus, the task of developing computational techniques to help users digest and exploit all the opinions is a very important and interesting research challenge.

Most existing work on tackling this general challenge has focused on integrating and summarizing opinions to help users better digest all the opinions (see

[1] http://dir.yahoo.com/
[2] http://www.amazon.com
[3] http://www.tripadvisor.com
[4] http://www.mediapost.com/publications/

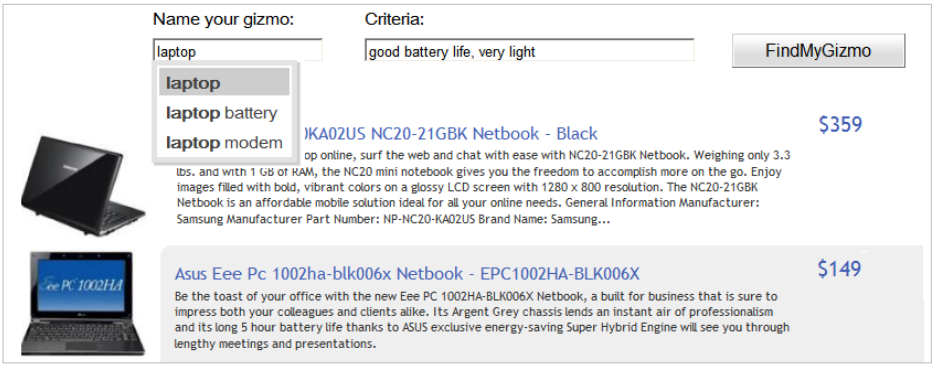

Figure 3.1: An ideal Opinion-Based Entity Ranking System that accepts keyword preferences as a natural keyword query.

Section 2 for a detailed review of related work). In this thesis, we propose a different way of leveraging opinionated content, that is to *directly* rank interesting entities based on how well the opinions on these entities match a user's preferences. Since a user is often interested in choosing an entity based on the opinions on the entity, ranking entities in this way provides a more *direct support* for a user's decision-making task. For example, the decision-making task in the case of a *user shopping for a product* is to decide which product out of the many to buy. Thus, it would be very helpful for such a user if we can take a keyword query from the user expressing his/her preferences for the product (e.g.,"comfortable seats, cheap and reliable" for a car), and return a ranked list of cars in the order of likelihood that a car matches the users preferences. With such a capability, the user is no longer overwhelmed by all the reviews available on all cars, but rather the user can now analyze a much smaller set of cars that roughly matches his/her preferences based on the judgment of other users. Further, this type of ranking is flexible in that it can be applied to any entity for which opinionated content is available.

To rank entities in this way, our idea is to represent each entity with the text of all the reviews of that particular entity, often available from various websites. Given a user's keyword query that expresses the desired features of an entity, we can then rank the relevant entities based on how well its reviews match the user's preferences. An ideal setup for an Opinion-Based Entity Ranking system is as shown in Figure 3.1, where the user can freely express preferences as a natural keyword query.

It is natural for a user to specify preferences on various aspects of an entity in the envisioned entity ranking task. Thus we can expect a user's query to consist of preferences on multiple aspects; for example, a preference query for a car might be "good gas mileage, cheap, reliable", which consists of preferences on three different aspects (i.e., efficiency, price, and reliability). In general, if a user enters a query in a single query box, we would need to parse a query to obtain preferences on different aspects. In this thesis, we focus on studying effectiveness

Figure 3.2: One scenario of Opinion-Based Entity Ranking applications where keyword preferences are expressed on a set of aspects.

of different ranking methods, so we assume that the multiple aspects in a user's query have already been segmented in order to factor out the influence of query segmentation on retrieval accuracy. Such a query can also be naturally obtained by providing a multi-aspect query form or asking a user to use a delimiter (e.g., a comma) to separate multiple preferences. For example, in Figure 3.2, we show a system interface where the users can find hotels in any city by stating their preferences on the various aspects of hotels.

Although this ranking problem closely resembles an information retrieval problem where the reviews of an entity can be regarded as an "entity document," there are two important differences. First, the query is meant to express a user's preferences in keywords; thus it is expected to be longer than regular keyword queries on the Web. More importantly, the query generally would contain preferences on multiple aspects of an entity. As we will show later in the paper, modeling these aspects can improve ranking accuracy. Second, the ranking criteria are to capture how well an entity satisfies a user's preferences rather than the relevance of a document to a query as in the case of regular retrieval. Therefore, the matching of opinionated words or sentiment would be important. We will show that although traditional query expansion works reasonably well in some cases, expanding a query with similar opinion words can significantly improve ranking accuracy on different types of data.

In addition to evaluating the effectiveness of standard text retrieval models for this task, we further propose several extensions of these models to better solve this special ranking problem. Specifically, we propose two heuristics: (1) *query aspect modeling* where we use each query aspect to rank entities and then aggregate the ranked results from the multiple aspects of the query; and (2) *opinion expansion* where we expand a query with related opinion words found in an online thesaurus. Our approach is light-weight, scalable and flexible as we avoid the need for costly information extraction and data mining.

Evaluation of this ranking task is a challenge since no existing test collection can be used for evaluation. We thus opted to create a benchmark data set by leveraging existing rating information. While it is not hard to collect reviews for different entities, it is a significant challenge to obtain reasonable queries and also to evaluate ranking accuracy quantitatively. We propose to solve this problem by leveraging the ratings of different aspects of cars and hotels available

on Edmunds.com[5] and Tripadvisor.com[6], and created two different data sets as a gold standard for quantitative evaluation. The data sets are available at `http://sifaka.cs.uiuc.edu/ir/downloads.html`.

Experimental results on these two data sets show that the proposed extensions over standard retrieval models are effective for the task of opinion-based entity ranking. The focused expansion technique (i.e. opinion expansion) is shown to be particularly effective. Modeling the aspects in a user's query as opposed to just treating the query as a "long keyword query" is also beneficial, especially for longer queries with more aspects.

3.2 Methods for Opinion-Based Entity Ranking

In this section, we present several methods for ranking entities based on how well its opinions match a user's preferences, including both standard retrieval models, which we treat as baselines, and some extensions of these models that we propose. To facilitate the discussion, we first introduce some notation. Let $E = \{e_1, ..., e_n\}$ be a set of entities to be ranked. For each entity e_i, we assume that we can collect a set of review documents $R_i = \{r_{i1}, ..., r_{in_i}\}$ that contain the opinions about the entity expressed by users or reviewers, where r_{ij} is a review document. Let D_i be the concatenation of all the review documents of an entity e_i. For convenience, we call D_i the opinion document for entity e_i. To solve the entity ranking problem, we cast it as a text retrieval problem where the text collection $\mathcal{C}$ consists of all the opinion documents for all the entities. That is, $\mathcal{C} = \{D_1, ..., D_n\}$.

From a user's perspective, the easiest way to express preferences for an entity would be to use keywords to describe desirable properties in various aspects. For example, a query for cars may look like "good gas mileage, small size, reliable." We denote such a keyword query by Q. On the surface, our problem is very similar to a regular retrieval problem. However, as discussed in Section 3.1, there are some important differences, which we will leverage to extend a regular retrieval model to improve ranking accuracy. In particular, our queries semantically consist of a set of sub-queries each describing preferences for one separate aspect of an entity, and we will show that it is indeed beneficial to model these semantic aspects. We will also show that emphasizing matching of opinion words through opinion expansion is very effective because it captures the desired matching criteria of relevance better for this ranking task. We now present three baseline standard retrieval models and then we present the two extensions mentioned.

[5] http://www.edmunds.com/
[6] http://www.tripadvisor.com/

3.2.1 Standard retrieval models

By casting the entity ranking problem as a problem of preference matching, we can directly use any standard retrieval model to solve the problem. Here we present three state-of-the-art standard retrieval models that we will experiment with; they are known to be most effective [52, 53] for the task of text retrieval.

BM25 (Okapi)

The BM25 (or Okapi) retrieval function was proposed by Robertson et. al [54] and has been shown to be quite effective and robust for many tasks. Although it was derived based on probabilistic models, it can also be regarded as a variant of the popular vector space model since it provides a term frequency-inverse document frequency (TF-IDF) weighting-based ranking formula. Formally, the score of an opinion document D in collection $\mathcal{C}$ (with n documents) and a query Q is given by:

$$S_{BM25}(D,Q) = \sum_{t \in Q \cap D} \frac{k_1 c(t,D)}{c(t,D) + k_1(1 - b + b * |D|/|\widetilde{\mathbf{D}}|)} \times \log \frac{n+1}{n_t}$$

where $c(t,D)$ and $c(t,Q)$ are the count of term t in document D and query Q, respectively, $|D|$ is the length of document D, $|\widetilde{\mathbf{D}}|$ is the average document length in the collection, n_t is the number of documents containing term t, and b, k_1, and k_3 are parameters that are typically set as $b = 0.75$, k_1 between 1.0 to 2.0, and k_3 between 0 and 1000. We replaced the IDF in the original Okapi formula with the normal IDF because the original one causes negative weights [53] and also performs significantly worse than the normal one in our experiments.

Dirichlet prior

The Dirichlet prior retrieval function is one of the most effective language models for retrieval [55]. It is derived based on query likelihood scoring [56] and Bayesian estimation of document language model [57], but its weighting formula also resembles TF-IDF weighting and document length normalization. Formally, the score of document D and query Q is:

$$S_{Dir}(D,Q) = \sum_{t \in Q \cap D} c(t,Q) \log(1 + \frac{c(t,D)}{\mu p(t|\mathcal{C})}) + |Q| \log \frac{\mu}{\mu + |D|}$$

where the notations are as in Okapi, $p(t|\mathcal{C})$ is the probability of term t according to a background collection language model, and μ is a smoothing parameter to

be empirically set.

PL2

PL2 is one of the most effective functions in the family of divergence from randomness retrieval (DFR) models [52]. Its scoring formula is based on basic statistics similar to those used in other retrieval functions and is formally defined as:

$S_{PL2}(D,Q) = \sum_{t \in Q \cap D} c(t,Q)$

$\times \frac{tfn_t^D \cdot \log_2(tfn_t^D \cdot \lambda_t) + \log_2 e \cdot (\frac{1}{\lambda_t} - tfn_t^D) + 0.5 \cdot \log_2(2\pi \cdot tfn_t^D)}{tfn_t^D + 1}$

where $tfn_t^D = c(t,D) + \log_2(1 + c \cdot \frac{|\widetilde{\mathbf{D}}|}{|D|})$, $\lambda_t = \frac{n}{c(t,\mathcal{C})}$ ($c(t,\mathcal{C})$ is the count of term t in the collection $\mathcal{C}$) and $c > 0$ is a retrieval parameter.

All these three standard retrieval models have corresponding pseudo feedback methods that can take some top ranked documents in an initial retrieval result as if they were relevant documents to extract additional terms to expand a query. Since we use the Terrier[58] toolkit for our experiments, we leverage the pseudo feedback mechanism implemented in this toolkit. Terrier provides various DFR[52] based term weighting models for query expansion. We specifically use the Bose-Einstein 1 (Bo1) model, which is based on Bose-Einstein statistics [59] and is similar to Rocchio[60].

Although standard retrieval models can be used to solve the opinion-based entity ranking problem, these models do not consider the multiple aspects in the query. It also does not consider the special notion of "relevance" when matching an opinion document with a query. Below, we present two extensions of a standard retrieval function to model query aspects and expand a query with opinion words.

3.2.2 Query Aspect Modeling (QAM)

In our setup, we assume that separate query fields would be provided for each aspect, thus the query would naturally consist of multiple aspects. However, a standard retrieval model would not distinguish these multiple aspects; as a result, it is possible that an entity may be scored high just because of matching one of the many aspects extremely well. Thus, one way to improve a standard retrieval function is to use each aspect query to score an opinion document (equivalently an entity) and then combine the scores of an entity in all the query aspects. This way, we can ensure that an entity matches all the aspects. Another potential advantage of modeling aspects in a query, though not explored in this thesis, is the ability to add expansion terms that are relevant to the aspect. For example, say we have a two aspect query - '*good gas mileage*' and '*extremely comfy*'. If we distinguish this query based on aspects, for *'good gas mileage'*, terms like *'mpg','mileage', 'fuel'* can be potentially added. However, if we treat

the user's preferences as long query, without distinguishing aspects, we have to be very careful on the type of terms added as we may end up retrieving items that are better in one aspect compared to the other.

While we have assumed separate query fields for different aspects, the aspects in a query can also be obtained explicitly by asking a user to use a special delimiter such as a *comma* to separate multiple aspects. These aspect queries can also be obtained from a regular keyword query using query *parsing* or *segmentation* techniques as shown in the work of [61]. Thus, by capturing multiple aspects in the query, we may now denote a query with $Q = \{Q_1, ..., Q_k\}$ where $k \geq 1$ and Q_i is a keyword query for an aspect of the entity, which we will refer to as an *aspect query*.

We now present several methods for leveraging this aspect structure. Let $S(D, Q)$ be any retrieval function. We can use the function to compute a score for each document with respect to each aspect query Q_i (i.e., $S(D, Q_i)$), and then combine the scores to generate an overall score for each document. Depending on how we combine the scores, we have several variants of this query aspect modeling (QAM) strategy. In particular, we can either combine the scores directly or combine the ranks of documents according to their scores in each query aspect. Moreover, we can also use different ways to aggregate the scores or ranks. In our experiments, we tested the following QAM scoring methods:

Average Score: $S_{AvgScore}(D, Q) = \frac{1}{k} \sum_{i=1}^{k} S(D, Q_i)$

Average Rank: $S_{AvgRank}(D, Q) = \frac{1}{k} \sum_{i=1}^{k} Rank(D, Q_i)$

Median Rank: $S_{MedRank}(D, Q) = Median_{i \in [1,k]} Rank(D, Q_i)$

Min Rank: $S_{MinRank}(D, Q) = Min_{i \in [1,k]} Rank(D, Q_i)$

Max Rank: $S_{MaxRank}(D, Q) = Max_{i \in [1,k]} Rank(D, Q_i)$

Here, $Rank(D, Q_i)$ refers to the rank of document D in the ranked list of documents for aspect query Q_i. Note that we did not consider other variations of score combination because of the concern that scores of a document in different aspects may not be comparable.

3.2.3 Opinion Expansion

Another limitation of the standard retrieval models for opinion-based entity ranking is that matching an opinion word and matching an ordinary topic word are not distinguished. Intuitively, since we would like to reward an opinion document where a query aspect is favorably reviewed, it is important to match

opinion words in the user's query. However, since topic words are expected to be much more common in review documents and have less variation than opinion words, we hypothesized that expanding a query with additional "equivalent" opinion words may help in emphasizing the matching of opinion words.

Consider a query like *'fantastic battery life'*. Due to the non-uniform way in which people express opinions, for the same expression, some may say *'awesome battery life'* while others may say something brief such as *'good battery'*. Therefore, it would be beneficial to expand such a query by adding synonyms of the word *fantastic*.

We thus propose the following opinion expansion method to expand a query with related opinion words. We use a controlled online dictionary[7] to first extract two classes of words from the query: (1) **intensifiers**, which are adverbs such as *very, really, extremely* and (2) **common praise words**, which are adjectives such as *good, great, fantastic*. In the case of intensifier words, we use only words that are neutral, where the orientation of the word would depend on the word or phrase following. This is to avoid changing the intended orientation of the query. For example, for the query 'extremely comfortable car', related intensifiers such as *exaggeratedly* and *excessively* can change the actual meaning of the user's preference as both these words have negative connotation. Such words would thus not be included in our list or expanded on when opinion expansion is performed. Table 3.1 shows the complete list of intensifiers and praise words used for opinion expansion.

For a given query Q, we can add synonyms of query terms to the query to enrich the representation of opinions and accommodate flexible matching of opinions. Formally, let t_i be a term in a given query Q. Let $syn_{a_1}, ..., syn_{a_{35}}$ be the set of synonyms for praise words and $syn_{b_1}, ..., syn_{b_{23}}$ be the set of synonyms for intensifier words. If t_i matches an intensifier term or a praise term, the corresponding synonyms will be appended to the query. Even if there are multiple praise words or intensifiers in a query, the expansion is done only once.

3.3 Data Set

Since the task of opinion based entity ranking as we defined has not been studied previously, no test collection exists for this task. This makes it a challenge to quantitatively evaluate the proposed methods. In this section, we describe how we address this challenge by creating a benchmark data set from two different domains. While review documents are easy to obtain from the Web, it is unclear how we can obtain queries and create a gold standard to quantitatively evaluate the proposed methods for entity ranking. We propose to use seed aspect queries to generate synthetic longer queries and leverage the available numerical aspect ratings as if they were relevance judgments. We believe that the creation of

[7] thesaurus.com

praise words	**intensifiers**
acceptable	absolutely
admirable	acutely
agreeable	amply
amazing	astonishingly
awesome	certainly
commendable	considerably
decent	dearly
excellent	decidedly
exceptional	deeply
fantastic	eminently
favorable	emphatically
genius	extensively
good	extraordinarily
gratifying	extremely
great	highly
honorable	incredibly
lovely	really
marvelous	substantially
nice	tremendously
pleased	truly
pleasing	very
premium	
remarkable	
satisfactory	
satisfying	
sound	
splendid	
stupendous	
super	
superb	
superior	
terrific	
tremendous	
wonderful	
worthy	

Table 3.1: List of praise words and intensifiers used for opinion expansion

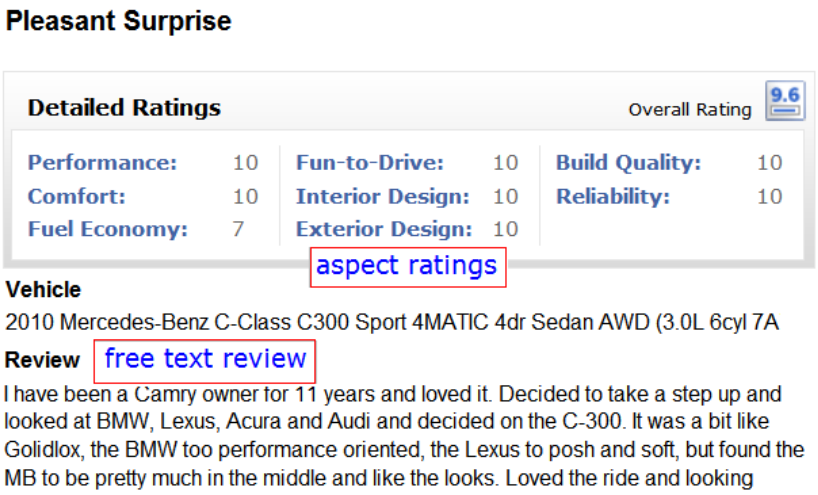

Figure 3.3: A sample car review from Edmunds.com.

this first test data set and the associated evaluation methodology for ranking entities, is one of the important contributions of this work. The data set is available at `http://sifaka.cs.uiuc.edu/ir/downloads.html`.

3.3.1 Review Document Collection

Our task is to return a set of entities based on how well the user's keyword preferences match the opinions on these entities. Therefore, we need a reasonable sized opinion data set supporting each entity. Although our idea is to allow the retrieval of any entity with supporting opinions, we chose to limit to sources that have free-text opinions accompanied by numerical ratings on individual aspects. We restricted our search to such sources to facilitate the evaluation of our task (explained in detail in Section 3.3.3).

With careful analysis, we chose to use reviews from two different domains that represent **different types of reviews**. The first is car reviews from Edmunds.com and the second is hotel reviews from Tripadvisor.com. Both sources have free-text reviews accompanied by numerical ratings on several aspects (all provided by users).

The nature of car reviews on Edmunds.com differs from hotel reviews on Tripadvisor.com. The hotel reviews are far more verbose than the car reviews. Most reviews on cars are only 4-5 sentences long, while the hotel reviews can span several paragraphs with detailed explanation of the reviewer's experience. Figure 3.3 shows an example of a car review from Edmunds.com. The section titled *Detailed Ratings* provides us with the discrete aspect ratings for each review.

To construct our data set, we collected reviews on cars for model-year 2007, 2008, and 2009 and hotel reviews for hotels in 10 major cities internationally.

Car Data Set						Hotels Data Set					
		average aspect ratings						average aspect ratings			
year	# of cars	max	min	mean	var	city	# of hotels	max	min	mean	var
07'	**227**	10.00	5.13	8.72	0.54	beijing	98	5.00	2.56	4.10	0.25
08'	228	10.00	3.79	8.75	0.63	chicago	116	4.92	1.70	4.02	0.31
09'	**143**	10.00	6.03	8.85	0.41	dubai	148	5.00	1.60	3.92	0.49
						las-vegas	154	5.00	1.12	3.70	0.47
						london	**727**	4.96	1.00	3.53	0.71
						montreal	98	4.97	1.10	3.79	0.57
						new-delhi	**80**	5.00	1.58	3.55	0.51
						new york city	246	4.98	2.58	4.09	0.19
						san-francisco	186	4.94	1.32	3.78	0.52
						shanghai	92	4.93	2.09	3.95	0.27

Table 3.2: Basic statistics on collected review data used in experiments. Columns labeled min, max and mean are based on the averaged per aspect user ratings for each entity.

This includes hotels in London, Beijing, Shanghai, Montreal, New Delhi, Dubai, New York City, Chicago, San Francisco and Las Vegas. In creating our data set, we avoided reviews that were too sparse as there would not be sufficient opinion text to test the effectiveness of a ranking method. Thus, we ensured that we only considered cars/hotels that had at least least 10 reviews.

The accompanying aspect ratings on Edmunds.com are on 8 different aspects, namely *fuel economy, comfort, performance, reliability, interior design, exterior design, build* and *fun to drive*. These ratings are on a scale of 1-10. As for hotel reviews, there are 5 aspects and this includes *cleanliness, value, service, location* and *room*. These ratings are on a scale of 1-5.

Table 3.2 provides a summary of the collected data. Columns labeled *min* and *max* show the absolute minimum and maximum aspect ratings for a given model-year/city, where the aspect ratings have been averaged across reviews of the same entity. The mean aspect ratings and variance are also shown in this table. Overall, the variance in ratings in both data sets is small.

3.3.2 Query Generation

The queries expected in an opinion-based entity ranking system are very different from a regular query one would issue to a typical vertical search engine, like a product search engine. If a user were looking for a laptop on Google Product Search[8], the user would typically type short keywords like *laptop* or *dell laptops*. Such systems generally return a list of entities without any specific order to start with, allowing the user to narrow down into the items of interest using different filters or through faceted navigation.

In our case, assuming that the type of entity (e.g. people, cars, hotels, restaurants) being searched for is known, users can then state their preferences for that entity using a set of descriptive keywords. These keywords would indicate what

[8]http://www.google.com/products

the user desires in the different aspects of that entity. For example, for a laptop we can have a query such as *'dell, good battery life, bright screen, very portable'*. The system would then return a ranked list of entities in the order of likelihood that the entity matches the user's preferences. Queries issued to a system such as this would thus have two important properties: (1) the query lengths can vary greatly - from short queries like *'good battery life'* to longer queries like *'excellent battery life, bright screen, lightweight'* and (2) the queries may contain opinion indicating words and intensifiers (e.g. very, extremely, good, super, excellent).

While there are many vertical search systems like Google Product Search, there exists no system that currently takes a set of keyword based preferences as shown in Figure 3.1. This makes it hard for us to obtain a natural sample of queries. We thus constructed our test queries from a set of seed queries. Since we expect the user to express his/her preferences on a fixed number of aspects, for the purpose of evaluation, we assume that these aspects would correspond to the aspects that have associated numerical ratings in our data set. We manually obtained a set of seed queries for each of these aspects and then we randomly combined the seed queries from different aspects to form longer multi-aspect queries that we call *generated queries*.

Specifically, we asked three average users to provide a few queries that they would issue on the various aspects of entities in our data set, to 'find' those that match their preferences. So, a user who desires a comfortable car with good gas mileage may issue a query such as '*comfortable seats, excellent mpg*', where 'comfortable seats' corresponds to the *comfort* aspect and 'excellent mpg' corresponds to the *fuel economy* aspect. The user thus specifies both the aspect being queried and the query keywords for that aspect. This is to simulate the behavior of obtaining queries from a query interface such as the one in Figure 3.2. With this, we obtained an average of six seed queries per aspect (5 for hotels and 7 for cars) for the two domains. We ignored one aspect, *'exterior design'* as it was not a popular topic of discussion within the car reviews, and hence may not help in evaluating retrieval methods that rely on keyword matching. In Table 3.3, we show the estimated aspect mentions in the car dataset. These numbers were obtained by counting the number of times the representative words in each aspect were mentioned.

Through random combination of seed queries from different aspects, we generated 10,000 queries per data set. These queries are to be used with entities in each city (for hotels) and model-year (for cars). The shortest query is one aspect long and the longest query can be a query that touches each aspect of the car/hotel. Each *generated query* can have at most one seed query from a given aspect. Table 3.4 shows some sample seed queries defined on 2 different aspects of cars and hotels and Table 3.5 shows some sample *generated queries* for the car data set.

Since the seed queries were obtained without a real system in place, it is

Aspect	Words used for keyword matching	Mentions
comfort	comfort	15530
interior	interior	13068
fuel economy	gas, fuel	10924
performance	performance	5013
build	built, build	4156
reliability	reliability, reliable	4119
exterior	exterior	3122

Table 3.3: Approximate aspect mentions in the car dataset.

important to ensure that these seed queries indeed represent typical user queries in our evaluation domain. Queries submitted to a car or a hotel search engine would not be useful because such systems are typically very structured and have limited support for natural keyword queries. However, users tend to use the major search engines like Bing[9], Yahoo![10] and Google[11] as a starting point to many of their search activities. Since the *query suggestion* feature of search engines is based on what other users have searched on, and the *related searches* feature is typically mined from query logs [62], we use both these features to determine how representative our seed queries are in these two domains.

We append the entity type to each seed query (for e.g., 'very clean' + '*hotel*' for the cleanliness aspect of hotels) and use that as a query into the major search engines. We then note the related searches and query suggestions for each seed query. We call these the *common aspect queries*. For example, a query like 'clean hotels' may yield in common aspect queries like 'clean hotels in Las Vegas' and 'clean hotels NYC cheap'. With this, we know that the seed query indeed reflects a natural user query. Almost all seed queries (in both domains) returned a set of common aspect queries on the major search engines. Table 3.6 and 3.7 show some of the seed queries with corresponding common aspect queries for each aspect in the two domains. The *build* aspect from the cars domain and the *service* aspect from the hotels domain are the only ones that had limited or no related queries (in all three search engines). This makes sense as some aspects are relatively more subjective or opinion oriented. So, it is not very likely that users would search for 'hotels with polite staff' on the major search engine sites. However, given a system like the one we envision, it would be more likely that such queries would be encountered. Therefore, these seed queries provide a nice mix of what a user typically looks for in these domains and what users could potentially search for in the future given an opinion-based search system. For further analysis, we looked into the Microsoft Live Labs query logs (released in 2006) to see what the most frequently mentioned aspects of preferences are in these two domains. This query log has 15 million queries, from US users, sampled over one month. Although this is a relatively small query

[9] www.bing.com
[10] www.yahoo.com
[11] ww.google.com

Cars	
Aspects	**Sample Seed**
fuel economy	good gas mileage, great mpg
comfort	comfortable, very comfy

Hotels	
Aspects	**Sample Seed**
cleanliness	very clean, clean
value	cheap, affordable

Table 3.4: Sample *seed queries* used to generate longer *multi-aspect queries*

Aspects	**Generated Queries**
comfort	*comfortable* *very comfy*
comfort, fuel	*comfortable, good gas mileage* *very comfy, great mpg*
comfort, reliability, fuel	*comfortable, reliable, good gas mileage* *comfy, dependable, great mpg*

Table 3.5: Example of *generated queries* for the car data set

log, it is sufficient enough to show some word distribution in these domains. For this, we used the words 'cars' and 'hotels' to retrieve all related queries from the query logs. For each domain, we then collect the counts of terms in these retrieved queries and sort them in decreasing order of their frequencies. The top 50 query words related to the purchasing of a car and the top 30 query words related to finding a place to stay are shown in Table 3.8. We see that all these words can be mapped into the aspects that we considered in generating our queries (the mappings are shown in parentheses in the table). Furthermore, in both domains, most of the aspects that we used for evaluation (i.e., aspects with known ratings from reviewers in Tripadvisor.com and Edmunds.com) were indeed queried by users. The aspects not well covered in these top query words are the *fun* and *comfort* aspects for cars and the *cleanliness* aspect for hotels. We believe that this does not necessarily indicate a lack of interest by users in these aspects, but rather, it is likely that users would not expect the current search engines to return meaningful results for such aspects, thus they would not even try such queries. Overall, the query log analysis results indicate that the queries we generated indeed represent typical aspects of preferences that users are interested in when ranking cars and hotels.

3.3.3 Relevance Judgments Generation

One of the most important task in our evaluation is to determine how well the retrieved entities match the user's preferences. Ideally, for a subjective task like this, given a user's preference query, we would need a human judge to read the related reviews and provide a judgment score of how well the retrieved entities match the user's preferences. This would involve understanding the underlying opinions in the reviews of each retrieved entity for each aspect involved in the user's query. This process is not only time consuming but can also be over-

Cars		
Aspect	**Sample Seed**	**Related User Queries from Google, Yahoo, Bing**
fuel	good gas mileage good fuel economy decent gas mileage excellent fuel economy	cars with high mpg cars with great gas mileage fuel efficient cars good fuel economy trucks cheap good gas mileage cars best fuel economy cars
comfort	comfortable very comfortable comfortable to drive	top 10 comfortable cars comfortable cars for back pain best comfortable cars small comfortable cars most comfortable ride
fun	fun driving fun to drive easy to drive	most fun driving cars most fun to drive cars 2010 fun to drive cars fun to drive sedans
build	well built good build solid build	well built cars most well built car
reliability	reliable very reliable durable dependable	most reliable car reliable used car dependable used car most dependable cars 2008 cheap dependable cars top ten durable cars cheap durable cars high reliability cars
performance	good overall performance good performance high performance	high performance cars performance cars for sale performance cars and trucks high performance used cars high performance electric cars
interior	quiet interior comfortable interior	cars with quiet interior quiet cars 2010 most quiet cars cars with quietest rides comfortable interior cars cars comfortable seats

Table 3.6: Seed queries and corresponding related user queries (about *cars*) on major search engines like Yahoo!, Bing and Google.

Hotels		
Aspect	**Sample Seed**	**Related User Queries from Google, Yahoo, Bing**
value	cheap affordable good value reasonable price	hotels downtown chicago reasonable prices cheap downtown chicago hotels cheap hotels affordable hotels in nyc good value new york city hotels good value hotels cheap very cheap hotels in new york
cleanliness	clean place clean good cleanliness	hotel nice clean cheap clean hotels nyc clean hotels in hershey pa clean hotel rooms cheap clean hotel clean hotel hong kong clean hotel singapore
room	spacious room comfortable room nice room cozy rooms	cozy hotels in chicago comfortable hotels in paris comfy hotels dublin comfortable hotel rooms in las vegas spacious hotel rooms in new york really nice hotel rooms cheap nice hotel rooms nice hotel rooms in las vegas
location	great location nice location great view nice view	great location hotels london paris hotels in great location new york hotels with great views hotels with great views in washington hotels with nice views san francisco hotels with nice views in nj
service	helpful staff polite staff good service	N/A

Table 3.7: Seed queries and corresponding related user queries (about *hotels*) on major search engines like Yahoo!, Bing and Google.

Top 50 query words related to cars (p=performance, g=mileage, i=interior, e=exterior, a=affordability, r= reliability)		
454 seat (i)	96 mileage (g)	35 alarms (i)
433 cheap (a)	93 diesel (g)	32 light (e)
352 muscle (e)	89 video (i)	31 speed (p)
217 hybrid (g)	79 performance (p)	31 efficient (g)
217 fast (p)	78 carseat (i)	31 compact (i)
211 seats (i)	64 safety (r)	31 cheapest (a)
190 sports (e)	64 fastest (p)	30 coupons (a)
173 gas (g)	63 small (e)	29 japanese (r)
172 electric (g)	50 convertible (e)	29 ipod (i)
171 fuel (g)	45 economy (g)	28 milage (g)
157 cool (e)	42 storage (i)	28 charger (i)
139 luxury (e)	41 alarm (i)	26 player (i)
130 stereo (i)	35 tv (i)	25 sound (i)
123 big (e)	35 miles (g)	
101 price (a)	35 dvd (i)	

Top 30 query words related to hotels (l=location, p=price, r=room, s=service)	
576 cheap (p)	38 niagara (l)
324 airport (l)	37 oceanfront (l)
305 island (l)	34 sea (l)
200 downtown (l)	32 university (l)
186 discount (p)	30 worth (p)
168 pet (s)	28 beachfront (l)
166 friendly (s)	24 romantic (l)
165 ocean (l)	24 coast (l)
161 lake (l)	23 rates (p)
113 luxury (r)	22 budget (p)
95 beach (l)	16 service (s)
78 falls (l)	16 pools (s)
52 water (l)	14 honeymoon (l)
45 jacuzzi (r)	
41 close (l)	
40 around (l)	

Table 3.8: List of most frequent co-occurring terms in queries "cars" and "hotels" in the Microsoft Live Labs query logs and their corresponding aspects of preferences.

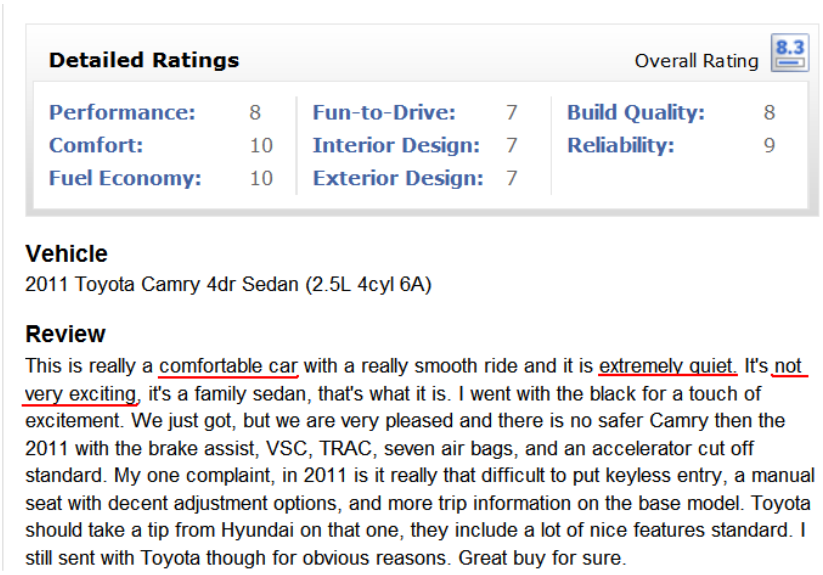

Figure 3.4: A car review with accompanying aspect based score ratings. There are mentions of the car being comfortable and quiet and accordingly a high score was given to the *comfort* aspect. There is also a mention of the car not being very exciting and as can be observed only a moderate rating was given to the *fun* aspect.

whelming and it may be hard for the human judges to keep track of the 'key opinions'. We thus need a reasonable way to approximate human judgment. To solve this problem, we propose to leverage the existing aspect ratings that come with the user reviews in our two data sets.

Both our data sets come with free-text reviews accompanied by a set of numerical ratings on several aspects. Some of the mentions in the free-text reviews directly reflect on the aspect score that an entity receives. Figure 3.4 shows a car review with corresponding aspect ratings. In this review, there are mentions of the car being 'comfortable and quiet' and accordingly a very high score was given to the *comfort* aspect. There was also a mention of the 'car being not too exciting' and accordingly, a moderate rating was given to the *fun* aspect. As in most user reviews, users tend to write about aspects that stands out most to them either in a good way or a bad way. In our two data sets, users are also allowed to provide aspect scores that may be reflective of some of their free-text comments. These aspect scores can thus serve as a relevance judgment score that indicates how well an entity performs on each of its aspects. We believe that this is a good approximation to human judgment. For example, if most users find that a particular car has excellent gas mileage, then the *fuel economy* aspect would have a high aspect score. In the other extreme, apart from negative mentions about the *fuel economy*, the score for this aspect would also be low. So, if a user is looking for a car with 'very good mpg' then ideally we should return all cars that have very high scores on the *fuel economy* aspect or

otherwise the system should be penalized. However, such a judgment is based on average ratings of a group of users, thus it may not reflect the real preferences of any particular user. As a result, the evaluation results using such judgments are only meaningful for relative comparison of different ranking methods, which is our goal.

Judgment scores are needed on individual aspects (to evaluate how well an entity matches one query aspect) and also on a combined set of aspects (to assess how well an entity matches the entire query). To compute judgment scores for individual aspects, we use the ratings provided by each user on a given aspect and average it. We call this score the Average Aspect Rating (AAR).

For queries that span multiple aspects, we take individual AAR scores of the aspects involved and average it. This, we call the Multi-Aspect AAR (MAAR). Let $Q = Q_1, ..., Q_k$ be a query with k aspects and E be an entity. Let $r_i(E)$ be the AAR of E in aspect i. Thus, $MAAR(E, Q)$ is defined as:

$$MAAR(E, Q) = \frac{1}{k} \sum_{i=1}^{k} r_i(E)$$

We assume that an ideal ranking of entities for query Q would correspond to ranking E in the descending order of $MAAR(E, Q)$, and this enables us to quantify how close a retrieval result is to this ideal ranking.

3.4 Experiments

In this section, we describe our experimental setup and present the experiment results on the two test sets.

3.4.1 Experimental Setup

Evaluation Measures

Since our gold standard has multiple levels of ratings for a car, we used the Normalized Discounted Cumulative Gain (nDCG) [63] measure as the evaluation metric of our ranking task. In an opinion-based entity ranking system, only the *top-k* items ($k = 10$ in our case) that closely match the user's preferences are deemed critical. Thus, we used nDCG of the top 10 entities (denoted as nDCG@10) as a main measure.

The Discounted Cumulative Gain (DCG) accumulated at a particular rank position p is defined as:

$$DCG_p = MAAR_1(E, Q) + \sum_{i=2}^{p} \frac{MAAR_i(E,Q)}{\log_2 i}$$

To allow the DCG to be comparable across queries and search results, it is normalized by its ideal ranking, which is obtained by sorting documents based on their MAAR values available from our gold standard. Let the DCG at

position p of the ideal ranking be denoted by $IDCG_p$. The nDCG is then computed as:

$$nDCG_p = \frac{DCG_p}{IDCGp}$$

Data Pre-processing

To evaluate the effectiveness of the proposed methods, we retained only the text segments of the reviews, dropping all HTML overhead and numerical ratings. The ratings were removed from our data set so that our experiments are in no way influenced by them. So, in essence, each document in our collection is a concatenation of text based reviews about a car/hotel. The length of each document varies greatly based on the number of reviews and also the size of individual reviews.

Implementation of retrieval methods

We use the three retrieval models (i.e., BM25, Language Modeling, and PL2) implemented in the Terrier 2.2 [58] toolkit for our experiments. We, however, had to make a few implementation changes to support Dirichlet Prior based Language Models [55] and fix the IDF problem of Okapi BM25 model discussed in [53].

3.4.2 Experiment Results

Standard Retrieval Models

We first look into the performance of the three state of the art standard text retrieval models. We used the default model parameters for Okapi BM25 (*b=0.75, k3=8, k1=1.2*) on both data sets as varying them did not make much difference in performance. PL2 uses a parameter c, a value for the term frequency normalization. This value was set to 1000 for both the car and hotels data set. We varied this value and found that a large value works well for the type of collection that we have. For the language modeling based retrieval, we set $\mu = 1000$ for both data sets as has been done in some previous work [64] and this value works well in our experiments.

	Hotels			**Cars**		
	PL2	**LM**	**BM25**	**PL2**	**LM**	**BM25**
StdNoFb	0.890	0.889	0.847	0.926	0.926	0.924
StdFb	0.897	0.896	0.869	0.926	0.923	0.923
change	**0.81%**	**0.74%**	**2.48%**	**-0.03%**	**-0.32%**	**-0.08%**

Table 3.9: nDCG@10 using standard (Std and StdNoFb) retrieval models.

The nDCG values based on 10,000 queries (for each data set) averaged across queries is reported in Table 3.9, where, in addition to comparing the three methods, we also compare these methods using the pseudo feedback mechanism explained in section 3.2. Based on Table 3.9, we can make several observations: (1) It appears that, overall, PL2 is most effective, followed by Dirichlet prior LM and then BM25. Interestingly, as we will show later, BM25 appears to perform the best with the proposed extensions. (2) We further see that pseudo feedback consistently helps improve the ranking of hotels but deteriorates the ranking performance of cars. Since the hotel reviews are much denser, the use of pseudo feedback is effective as the terms added to expand the query are more meaningful for the ranking process. Upon analysis of the pseudo feedback for the ranking of cars, it becomes clear why performance is degraded. For the query *'good fuel efficiency'*, some of the words added are *4cycl, jeep* and *kia*, and these words have no relation to fuel efficiency being good, resulting in the wrong cars being ranked highly. Even though pseudo feedback seems promising for this task, it only helps when the reviews are verbose. We will show later that our proposed opinion expansion is consistently effective and improves performance on both data sets.

Opinion Expansion

We now look into the question of whether the proposed opinion expansion method helps improve ranking accuracy. To test the idea of opinion expansion, we alter a query if it contains a praise word or an intensifier, and add the corresponding opinion synonyms to expand the query (explained in section 3.10). Table 3.10 shows the results obtained using opinion expansion on top of standard models and models that use query aspect modeling (to be discussed in the next section). From this table, it is indeed clear that opinion expansion helps all models in generating better ranking of hotels and cars. The performance improvement for BM25 is especially clear. With the use of opinion expansion, BM25 proves to be most effective amongst the three retrieval models. (We will further compare the three retrieval models in Section 3.4.2. The Wilcoxon signed rank test [65] shows that all the improvements in Table 3.10 are *statistically significant with a very low p-value ($p < 10^{-6}$)*. This indicates that enriched opinion words in the query can indeed accommodate flexible matching of opinions, which is needed for the opinion based entity ranking task; in contrast, the standard pseudo feedback-based query expansion is only effective in some cases (see Table 3.9). Moreover, the improvements observed with pseudo-feedback are not as high as can be achieved with opinion expansion.

It is possible that the improvement of opinion expansion may have come from simply favoring entities with more 'positive' reviews. That is, it is possible that the System selects entities that are positive overall, which would naturally have higher MAAR scores, thus yielding better nDCG than the baseline method. To

	Hotels			Cars		
	PL2	**LM**	**BM25**	**PL2**	**LM**	**BM25**
StdNoFb	0.890	0.889	0.847	0.926	0.926	0.924
+ OpinExp	0.921	0.918	0.923	0.936	0.932	0.950
change	**3.38%**	**3.17%**	**8.18%**	**1.06%**	**0.48%**	**2.73%**
AvgScoreQAM	0.898	0.894	0.848	0.926	0.927	0.924
+ OpinExp	0.924	0.920	0.928	0.936	0.934	0.951
change	**2.77%**	**2.85%**	**8.61%**	**1.08%**	**0.67%**	**2.75%**

Table 3.10: nDCG@10 using opinion expansion

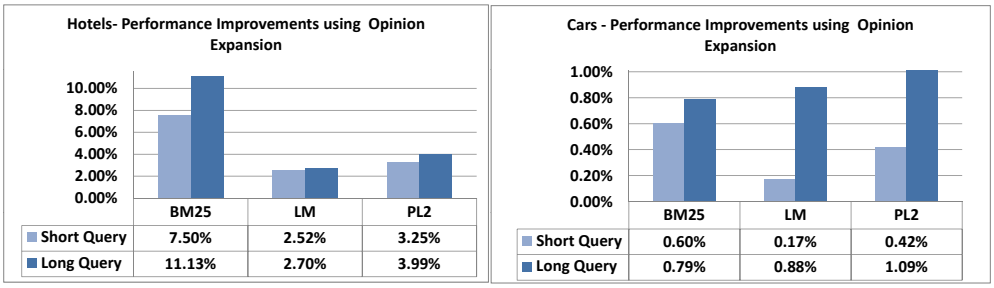

Figure 3.5: Performance improvements over the AvgScoreQAM model with the use of opinion expansion for long and short queries. Better improvements are achieved on longer queries than shorter queries.

analyze the actual behavior, we look into the performances of two subgroups of queries, short queries and long queries. Short queries are those that touch 1-2 aspects, while long queries are those touching 4-5 aspects for hotels and 6-7 aspects for cars. If the System was only picking out entities that were more positive in general, the improvements on shorter queries should be just as high or in fact higher (since it is less affected by score combination across aspect queries). This is however not the case as can be seen in Figure 3.5. The graphs show that the improvements achieved on longer queries is considerably higher than that achieved on shorter queries, which means that the system is not just favoring entities that are simply more positive.

Query Aspect Modeling

Another extension we proposed is to model the multiple aspects in the query explicitly and then combine the scores from multiple aspects to generate an overall score for a document. We now examine the effectiveness of this extension.

Table 3.11 summarizes results obtained with the query aspect modeling approach when the aggregation method is "Average Score" (i.e., $S_{AvgScore}(D, Q)$), which, as will be shown later, is the best among all the four ways of aggregation when used with opinion expansion. From this table, we see that query aspect modeling improves performance of ranking on both data set. Even though opinion expansion significantly improves the performance of the standard method

	Hotels			Cars		
	PL2	**LM**	**BM25**	**PL2**	**LM**	**BM25**
StdNoFb	0.890	0.889	0.847	0.926	0.926	0.924
AvgScoreQAM	0.898	0.894	0.848	0.926	0.927	0.924
change	**0.97%**	**0.58%**	**0.12%**	**0.00%**	**0.16%**	**0.00%**
StdNoFb + OpinExp	0.921	0.918	0.923	0.936	0.932	0.950
AvgScoreQAM + OpinExp	0.924	0.920	0.928	0.936	0.934	0.950
change	**0.35%**	**0.25%**	**0.58%**	**0.00%**	**0.18%**	**0.00%**

Table 3.11: nDCG@10 of using standard models against QAM models.

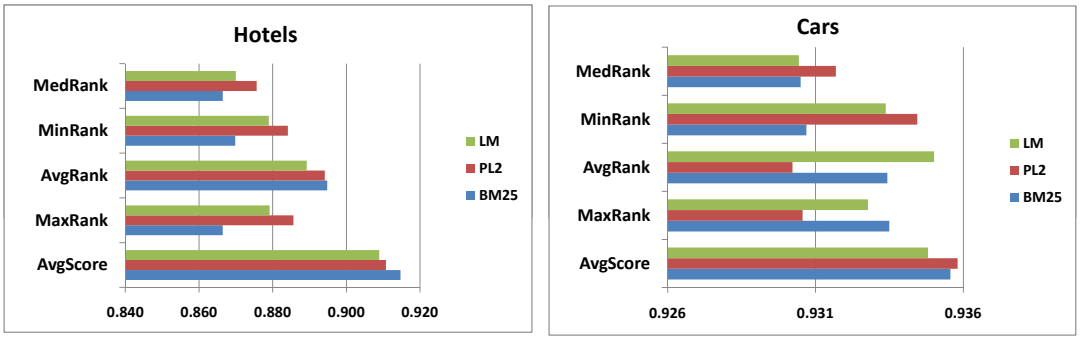

Figure 3.6: nDCG@10 using different ranking strategies with QAM+OpinExp

(as shown in Table 3.10), introducing query aspect modeling provides further improvements. Wilcoxon signed rank test [65] shows that all the improvements above 0.1% in Table 3.11, are statistically significant with a very low p-value ($p < 10^{-6}$).

In Figure 3.6, we further provide a comparison of performance results using the different ranking strategies. This comparison is essential as the ranking strategy has a direct impact on how the entities are ranked. Based on this graph, we can say that the average score (AvgScore) based strategy works the best on the whole. The use of the actual ranks like AvgRank only works well in some cases as can be seen in the graph.

One advantage of our evaluation method is that we can easily analyze queries of different numbers of aspects. Since this factor is intuitively related to effectiveness of query aspect modeling, we further looked into how well the base method compares to the aspect modeling method on queries of different numbers of aspects.

Users who provide short queries are typically flexible users who have limited preferences. Queries that such users issue could be short queries like *'good mpg'*. There are also the "picky" or "rich" users who have very specific preferences on many aspects. These users will typically issue long queries like *"excellent fuel economy, comfortable interior, solid build, highly reliable"*. For both the data sets, we manually selected some of the shortest queries (covering 1-2 aspects) and some of the longest queries (covering 6-7 aspects for cars and 4-5 aspects for

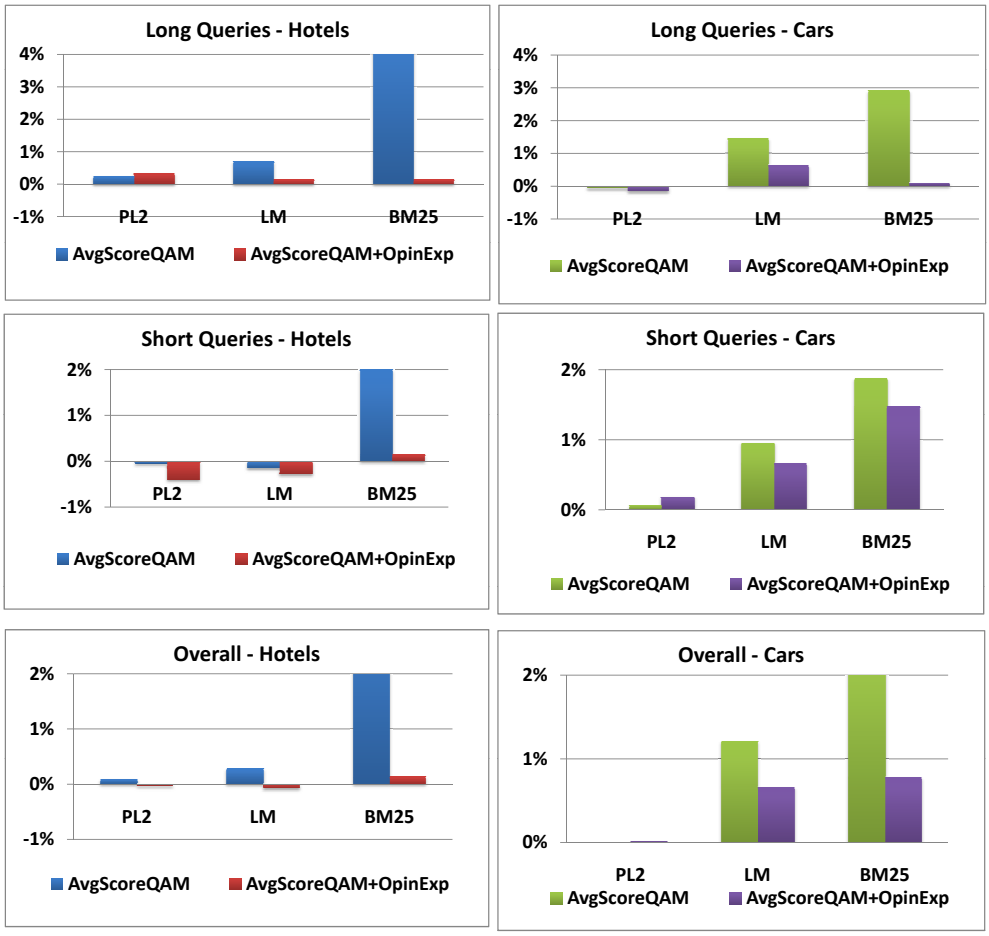

Figure 3.7: Performance change of AvgScoreQAM over StdNoFb and AvgScoreQAM+OpinExp over StdNoFb+OpinExp on queries of different length

hotels). We compare the performance of the QAM runs with its corresponding standard run on these queries. The percentage of change in performance is shown in Figure 3.7.

On the car data set, it can be seen that the aspect modeling of queries consistently yields performance improvement on very short queries. On longer queries however, performance improvements can only be seen with the LM and BM25 models. The reverse is the case for hotels. Modeling aspects in short queries seems to be effective only with BM25. On longer queries however, all three models benefit from the use of query aspect modeling. Overall, the use of QAM shows to be most beneficial with the BM25 model with consistent performance improvements on both data sets and for both long and short queries.

Behavior of Retrieval Models with Opinion Expansion

While all three retrieval models show performance improvements with the use of opinion expansion, BM25 consistently outperforms its counterparts with the use of this expansion technique. To understand why, we looked deeper into the details of the rankings. Specifically, we compared these three models in two subgroups of queries (short vs. long) and three subsets of review documents with different sizes. Each city (for hotels) and model-year (for cars) has a set of review documents, where each review document represents a distinct real-world entity. For the purpose of this discussion, we will refer to all review documents in a given city or model-year as a *collection*. As shown in Table 3.2, each collection can have a varying size of review documents.

Figure 3.8 shows the AvgScoreQAM and AvgScoreQAM+OpinExp performance on the hotels data set at different collection sizes for both long queries and short queries. Here, we see that for both types of queries, when no opinion expansion is used, the LM approach is most stable to variation in the collection size, but as the collection size grows, the other two models suffer a degradation in performance. In particular, BM25 is worse than the other two methods in all cases. With the use of opinion expansion, it is interesting that we now see a different pattern: the BM25 model performs the best overall, and in particular, it does much better than the other two models when the collection size is large (i.e., more entities to rank). A similar behavior was also observed with the cars data set. This means that BM25 gains much more than the other two models from opinion expansion.

Analytically, a major difference between BM25 and the other two models is that BM25 has an upper bound on the score contribution that can be made by each matched query term, no matter how frequently the term occurs in the document [66], while the other two do not have this property. Thus intuitively, BM25 would favor documents that match more query terms, while the other two models would be more prone to favoring non-relevant documents that match just a few query terms many times. Since opinion expansion would introduce many additional opinion and intensifier words, we hypothesize that the reason why BM25 gains more from opinion expansion is because PL2 and LM cannot properly handle the additional words added to the query, which could occur frequently in the review documents. The mistakes that it makes in terms of ranking become far more apparent when the collection size is large. However, with BM25, any one term's contribution to the document score cannot exceed a saturation point.

To validate this hypothesis, we looked into the result set of a query that yielded in high discrepancies in the rankings between the competing paradigms. The query is 'very clean, cozy rooms, excellent staff'. For this query, we took the first ranked entity of each result set (PL2 and LM ranked the same entity as the first) and plotted a graph that shows the query terms (after expansion),

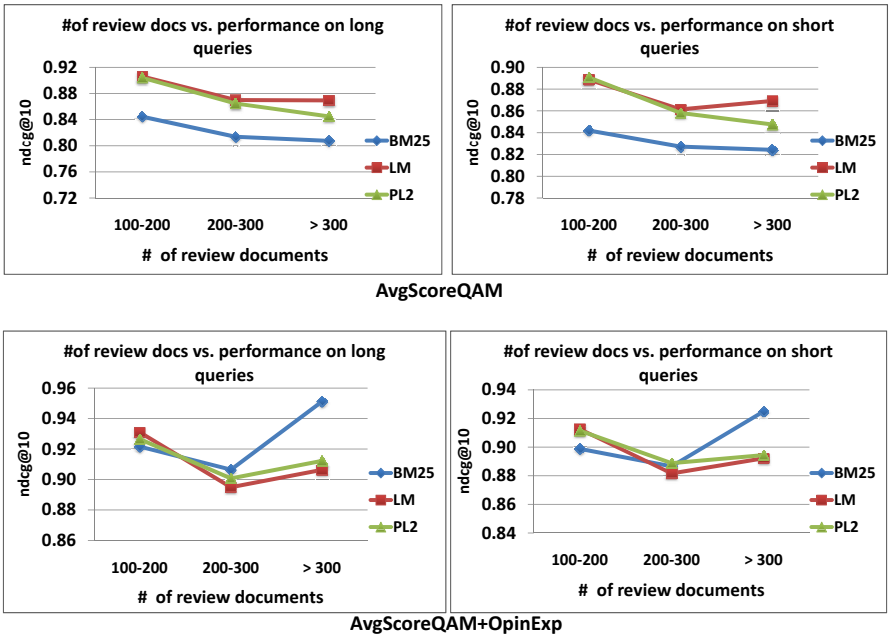

Figure 3.8: Performance of **AvgScoreQAM** and **AvgScoreQAM+OpinExp** vs the number of review documents in each city from the hotels data set.

against the average term frequencies of the query terms in its corresponding entity document. The resulting graph is as shown in Figure 3.9. The MAAR score of the first ranked entity by PL2 and LM is 4.54 (denoted by A), while the one by BM25 is 4.83 (denoted by B). The highest MAAR from the gold standard for this query is 4.87.

Figure 3.9 shows that the top ranked entity by BM25 indeed has a more balanced matching of all query terms, while the top ranked entity by PL2 and LM has more skewed frequencies of query terms. For example, A has a very large number of occurrences of the term 'very', while an important original query term 'cozy' has a very low average frequency. In contrast, B matches the query terms in a more balanced fashion, where the original query terms (labeled in the graph) and the expanded terms have average frequencies that are not extremely high or extremely low.

Such a concern about the skewness of matched query terms becomes more serious after opinion expansion as an expanded query would contain many redundant words, increasing the chance of a non-relevant document to dominate the ranking result. Similarly, when the collection size is large, the problem also becomes more serious as there is a higher chance of having such a distracting non-relevant document.

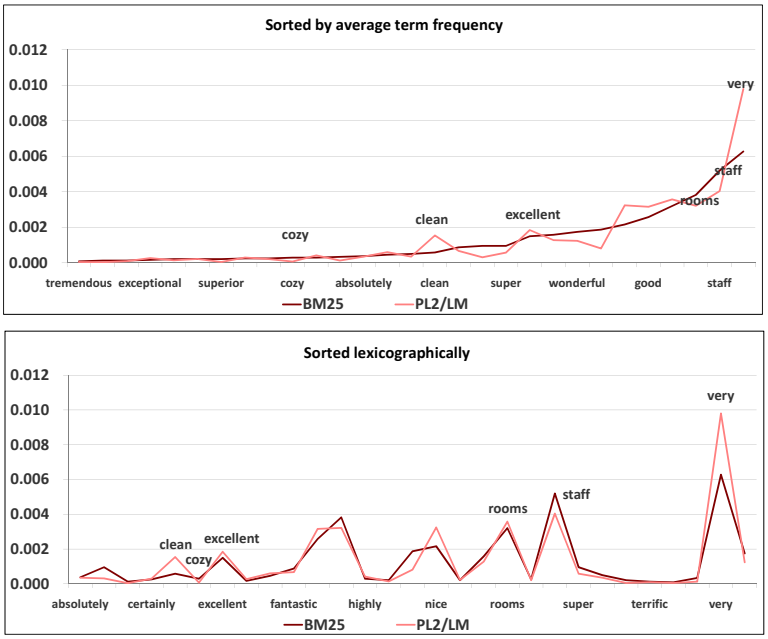

Figure 3.9: Average term frequency of query words of the first ranked entity for the query 'very clean, cozy rooms, excellent staff'. The labeled terms are the original query terms. All other terms are the result of opinion expansion. PL2 and LM ranked the same entity as the first.

Influence of the availability of review data

One assumption in our problem setup is that we have enough review data to represent opinions about an entity. We now try to understand how much data we actually need to get a reasonable ranking of entities. This will also help us understand if the proposed extensions can be expected to perform better and better as we accumulate more review data. To understand this, we varied the amount of reviews used by selecting a different percentage of reviews for ranking. We ran the best performing configuration, (which by far is the AvgScoreQAM+OpinExp run) on these different sizes of reviews.

Figure 3.10 illustrates the performance versus the amount of review data used. Notice that for the hotels data set, the performance peaked when we used only 60%-70% of the data, after which there was a slight degradation in performance. On the car data, performance consistently improved after about 60% of the data was used.

The quick performance improvement for the hotels data set is likely due to the verbose nature of this data set. While for the car data set, due its sparse nature, almost the entire data set was needed for the performance to peak. The trend of this curve indicates that there could be more improvements if more reviews were introduced. It is possible that the quality of reviews used would also play a role in how much review data is actually needed for this task.

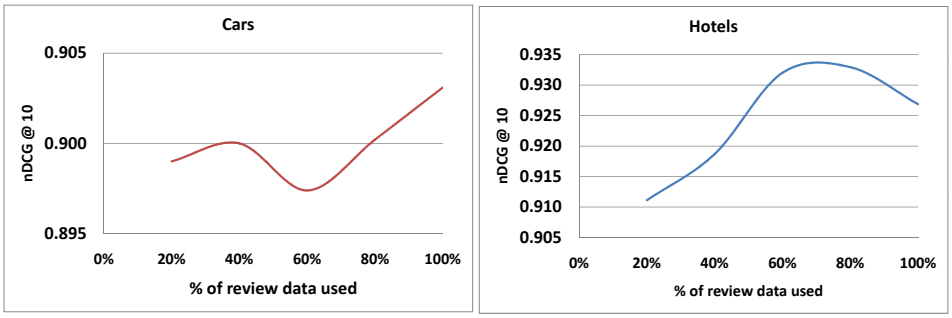

Figure 3.10: Performance vs % review data used

Sample Results

To illustrate some sample results of ranked hotels and cars, we show results from the two domains. First we show how a ranked list of hotels change as aspect queries are added to it. Then, we show the top ranked cars for an interesting query. The results shown were obtained using the AvgScoreQAM+OpinExp configuration.

Table 3.12 shows the top 10 ranked hotels in Dubai (with corresponding AAR) that match the query, *'very clean'*. Then, in Table 3.13, we show how this ranked list changes as a new aspect query, *'great views'* is added to the original query. From Table 3.12 we can see that the lowest AAR for the *cleanliness* aspect (for all hotels in Dubai), is 2.71 and the highest is 4.951. The AAR scores of all the top 10 hotels that match this query are above the average AAR for this aspect. This clearly shows that the users are indeed getting reasonable matches. However, the ordering of these entities are still not perfect. For example, the first ranked hotel, *Hatta Fort Hotel*, has an AAR score that is lower than that of *Burj Al Arab*, the hotel that ranks second in this list.

Next, when a new aspect query, *'great views'*, is added to the current query, there is a noticeable change in the ranking of hotels (as shown in Table 3.13). The *Burj Al Arab* which previously ranked second, now ranks first with the addition of this new aspect query. The *Le Royal Meridien Beach Resort* which ranked third, now ranks tenth in the second ranked list. The *Hatta Fort Hotel* that previously ranked first, is not even in the top 10 of this new ranked list. This is reasonable because the AAR of the *Hatta Fort Hotel* on the *location* aspect is only 4.107 compared to 4.745 for the *Burj Al Arab*. Most entities in this list have AAR scores that are well above the average in their respective aspects.

Here are some interesting review snippets for *Burj Al Arab* with regards to *cleanliness* and *location*:

"The rooms are really huge and spotlessly clean, the gym is state of the art with great sea views from the tread mills and the Spa is fantastic...."

System Rank	Hotels	'cleanliness' AAR
1	hatta fort hotel	4.607
2	**burj al arab**	**4.920**
3	**hilton dubai creek**	**4.642**
4	**le royal meridien beach resort spa**	**4.914**
5	renaissance dubai hotel	4.600
6	the ritz carlton dubai	4.693
7	al manzil hotel	4.915
8	**le meridien dubai**	**4.586**
9	**hilton dubai jumeirah**	**4.762**
10	bel ali golf resort spa	4.620
Highest possible AAR		4.951
Lowest possible AAR		2.710
Average AAR		4.220

Table 3.12: Top 10 ranked hotels for the query 'very clean'. This ranking has an nDCG of 0.960. All hotels in this list have AARs above 4.5, which is above the average AAR for this aspect.

"The rooms are all suites and very spacious. They are all 2 floors with beautiful views...The rooms are clean and the hotel is well situated."

"...the hotel itself is just beautiful, and in a lovely location, with fantastic views from all the floor to ceiling windows in our suite (13th fllor) across the marina..."

The second illustration of results is based on the query *'very reliable'* on the car data set, a query that most people can relate to. The top 10 cars that match this query is shown in Table 3.14. As can be seen in this list, the cars returned are mostly Japanese cars which are known for their reliability[12]. While these cars have high AAR scores on the *reliability* aspect, the overall ratings of these cars are not necessarily high. This shows that the system is not simply retrieving cars that are positive overall. The following snippets show some of the supporting comments for the first ranked car, *2007 Honda Accord*.

"...Solid, reliable car with low cost of ownership. Nice computerized maintenance notification system. Comfortable heated leather seating..."

"...I had to find something reliable, with good resale. This car is incredible....."

"...My experience with this vehicle has been as follows - the engine & transmission provide a smooth, powerful and reliable ride. The suspension is awful though..."

[12]http://www.independent.co.uk/life-style/motoring/motoring-news/japanese-cars-are-still-the-most-reliable-2016405.html

System Rank	Hotels	'cleanliness' AAR	'location' AAR
1	**burj al arab**	**4.920**	**4.745**
2	jw marriott hotel dubai	4.373	3.608
3	**hilton dubai creek**	**4.642**	**4.112**
4	al qasr at madinat jumeirah	4.833	4.817
5	mina a salam at madinat jumeirah	4.918	4.881
6	dar al masyaf at madinat jumeirah	4.951	4.848
7	grand hyatt dubai	4.895	4.289
8	**le meridien dubai**	**4.586**	**4.069**
9	**hilton dubai jumeirah**	**4.762**	**4.312**
10	**le royal meridien beach resort spa**	**4.914**	**4.694**
Highest possible AAR		4.951	4.881
Lowest possible AAR		2.710	1.900
Average AAR		4.222	3.767

Table 3.13: Top 10 ranked hotels for the query 'very clean' and 'great views'. This ranking has an nDCG of 0.944. The bolded hotels appear in the result set of the query 'very clean' shown in Table 3.12.

3.5 User Study

We performed a small user study to further understand the effectiveness of our proposed method in retrieving entities and also assess the effectiveness of our evaluation strategy. In this study, we asked users to judge the relevance of entities retrieved by our best performing system (BM25 with AvgScore-QAM+OpinExp). These relevance scores were then used for various analysis.

3.5.1 Procedure

We recruited two undergraduate students (referred to as *User1* and *User2*) who were asked to act as 'real users' of a system that enables them to search for entities based on a set of preferences. These users were presented with a query, and corresponding results (i.e. the ranked list of entities that satisfy the query) along with its respective reviews. The users were informed that the query is meant to be a set of user preferences and the entities presented as results should ideally match these preferences based on the reviews. With this in mind, for each query, the users were asked to *analyze the reviews* of the top 10 entities and then assign a relevance score to those entities based on how well it satisfies the query. This judgment is based on a 3-point rating scale defined as follows:

System Rank	Cars	'reliabillity' AAR	overall ratings
1	2007 honda accord	9.350	8.846
2	2007 honda civic	9.280	8.870
3	2007 toyota camry	9.720	8.115
4	2007 toyota yaris	9.690	9.275
5	2007 toyota corolla	9.360	8.700
6	2007 honda fit	9.580	9.079
7	2007 honda cr-v	9.380	8.933
8	2007 toyota tundra	9.170	8.871
9	2007 ford fusion	9.460	9.101
10	2007 toyota tacoma	9.090	8.790
Min		6.320	6.888
Max		9.940	9.790
Average		8.951	8.722

Table 3.14: Top 10 ranked cars from model-year 2007 that match the query 'very reliable'. Most cars have AAR scores that are above average.

Score 1: Poor match. The entity does not satisfy the query well.
Score 2: Reasonable match. The entity satisfies the query reasonably well.
Score 3: Good match. The entity is a very good match for the query.

For each relevance score that the user assigns, the user was also asked to provide a brief justification for those scores. For example *Score (1) - Does not match most preferences* or *Score (2) - Matches only some preferences really well*. This study was performed on 25 queries which were randomly selected from both our car and hotel dataset. Our goal is to obtain a representative set of queries of different characteristics. In total, we had 12 long queries (touching > 2 aspects) and 13 short queries (touching 1-2 aspects). The entities presented as results were generated by our best performing system (BM25 with AvgScoreQAM+OpinExp).

3.5.2 Analysis of Relevance Ratings

In Table 3.15 we report the average relevance ratings assigned by User1 and User2. On average, it can be seen that both users thought that the entities retrieved by the system were a reasonable match to the queries. Notice that in the majority of cases, both users thought the entities were either a *reasonable match* (User1 - 110 entities; User2 - 81 entities) or a *good match* (User1 - 84 entities; User2 - 140 entities), rather than a *poor match*(User1 - 56 entities; User2 - 29 entities). This shows that our proposed *retrieval based method* for this special task is actually quite effective, with an average rating of above 2.0.

We further look into the entities that were assigned a low score. In Table 3.16, we summarize the most common justification provided by User1 and User2 on

	User 1	User 2
Average Rating	2.14	2.44
Std. Dev	0.40	0.25
# Entities rated 1	56	29
# Entities rated 2	110	81
# Entities rated 3	84	140
Total	**250**	**250**

Table 3.15: Average user judgment scores.

	User1	User2
Score (1)	-does not match one or more preference -does not match any of the preferences well	-no preference matched except one -no preferences are matched
Score (2)	-matches all preferences, but not too much -match most preferences well, but some do not match that well	-all preferences are matched, but some conflicting opinions -all preferences are matched to some extent -not much information about one preference
Score (3)	-matches all preferences well	-matches all preferences well -matches all preferences well, except one

Table 3.16: Summary of relevance score justification given by User1 and User2

their rating assignments. As can be seen, a score of 1 is typically assigned when the reviews do not contain any mentions about one or more preferences within the query. A score of 2 is assigned when (1) there is limited evidence in the reviews about the preferences or (2) only some preferences are matched well or (3) there are conflicting opinions about a preference. A score of 3 is only assigned when most of the preferences are matched well (with sufficient evidence).

The agreement in terms of relevance ratings assigned by User1 and User2 is shown in Table 3.17. As can be seen, the kappa scores show that the agreement is quite low with most of the disagreement happening when the users were to choose between a rating of 2 and 3. Also, the disagreement is higher on longer queries than on shorter ones. This may be because with longer queries, we have more preference criteria, which amplify the variances of subject judgments. The results also suggest that User1 seems to have used a different rating strategy than User2 and this is also quite clear from the justification summary provided in Table 3.16.

Deeper analysis into the rating assignments reveals that User1's strategy is

overall agreement				**short queries**				**long queries**			
	1	**2**	**3**		**1**	**2**	**3**		**1**	**2**	**3**
1	5	14	37	**1**	2	13	9	**1**	3	1	28
2	2	22	94	**2**	1	15	43	**2**	1	7	51
3	0	5	71	**3**	0	5	42	**3**	0	0	29
kappa	0.09			kappa	0.12			kappa	0.07		

Table 3.17: Agreement on relevance ratings between User1 and User2

to look into both the number of matched aspects as well as how many people praised the relevant aspect. The user first checks if all preferences in the query are matched in the reviews. If all preferences are matched and if the user feels that there is 'enough' evidence for each of those preferences, then User1 assigns a rating of 3. Otherwise, the user only assigns a rating of 2. User2's strategy is to look at the bigger picture. On short queries, if all preferences are matched well then a rating of 3 is assigned. If all the preferences are matched well but there are some conflicting opinions, then a score of 2 or 1 is assigned depending on the severity of conflict. On longer queries however, if just one preference is not matched well, the entity is still considered a good match and a score of 3 is assigned. A score of 2 or 1 is only assigned when there are either conflicting opinions or more than one preference does not match well.

These differences are indeed very interesting as this tells us that different users have different criteria in judging the relevance of an entity. Some users may prefer entities ranked based on the level of evidence (positive mentions) on an aspect. Other users may prefer entities with no conflicting opinions even though not all preferences are matched well. This suggests that the ranking of entities can be further personalized according to what matters most to the user.

While the individual ratings provided by User1 and User2 do not agree all that well, it is quite possible that correlation exists in their relative preferences of entities. We thus measured rank correlation using the relevance ratings provided by both users. In particular, we computed the average Gamma correlation coefficient [67] between the rankings. The Gamma statistic was preferred over Kendall τ as ties are taken into account explicitly. Note that ties are common in the rankings of User1 and User2 as they were only allowed to use a 3-point rating scale. The correlation ranges between -1 and +1. A value of 0 means that there is no correlation; 1 is perfect positive correlation; -1 is perfect negative correlation. Based on the 25 queries, we obtained an average correlation score of 0.69. This correlation score shows that the two users actually agree reasonably well on the relative rankings of the entities even though the actual score assignments may be different.

3.5.3 Effectiveness of Gold Standard Rankings

In our evaluation, we have assumed that the average numerical ratings provided by review writers (on various aspects), would reflect the best ordering of entities. These ratings were thus used as the gold standard rankings. To validate this assumption, we compare the nCDG of the gold standard rankings and system rankings using the relevance ratings provided by User1 and User2. Specifically, we assume that the *actual* ideal ordering of entities is based on the ratings provided by User1 and User2 (as opposed to our gold standard rankings). Then, to compute the system nDCG, the relevance ratings provided by User1 and User2 are re-ranked according to the system rankings. Similarly, to compute

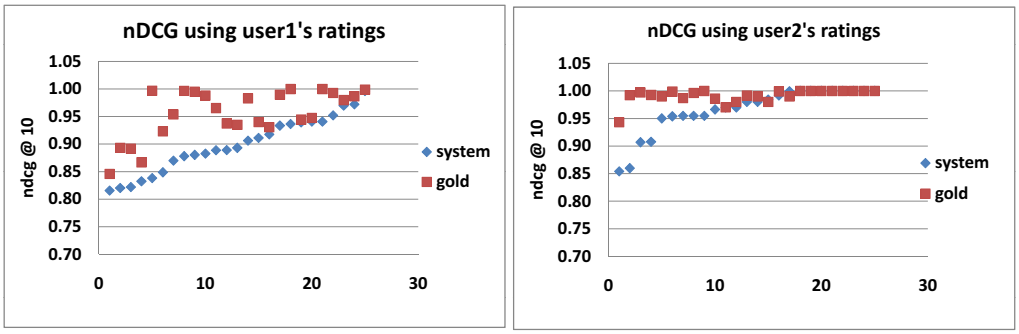

Figure 3.11: nDCG @ 10 scores of system rankings and gold standard rankings using judgments provide by user1 and user2

User1		**User2**	
System Avg.	Gold Avg.	System Avg.	Gold Avg.
0.865	**0.910**	0.923	**0.950**

Table 3.18: Average nDCG @ 10 scores of system rankings vs. gold standard rankings using judgments provide by user1 and user2.

the nDCG of our gold standard rankings, these relevance scores are re-ranked according to the gold rankings. The intuition here is that, if our gold standard ranking is indeed an accurate measure of relevance, it should have stronger agreement with human rankings than the system rankings would. In other words, compared to the system, the gold standard should be better at recovering human rankings.

Figure 3.11 shows the resulting nDCG scores of system rankings and gold standard rankings using the relevance ratings provided by User1 and User2. In Table 3.18, we report the average scores. Based on Figure 3.11, we see that in many cases (especially for User1), the resulting nDCG scores of the gold standard rankings is higher than that of system rankings. The cases where the scores overlap almost perfectly was due to ties in the rankings. As an example, when a rating of 3 is assigned to all entities, this results in the same nDCG scores for both the system rankings and gold standard rankings regardless of any ordering. As can be seen, this mainly happens to entities ranked by User2. On average however (see Table 3.18), it is clear that the gold standard agrees more with the two users than does the system. Thus, our assumption that the average numerical ratings given by web users can be a good approximation to human judgment is indeed reasonable.

3.6 Discussion

Overall, our experiments show that the idea of ranking entities based on a user's keyword preferences and the opinions of other users is promising and opens up

a new application area of retrieval models. Even the simple extensions that we made to the standard retrieval models have already shown promising results, and there are many possibilities to further optimize a retrieval model for this task.

In this thesis, we only studied the effectiveness of our proposed method in two specific domains and on a fixed set of aspects (to facilitate evaluation). However, our idea itself can be expanded to a variety of real world domains which includes ranking people, products, businesses and services using a set of keyword based preferences expressed on any arbitrary aspect. The basic requirement in setting up such an opinion-based entity ranking system is the need for a large number of opinion containing documents. For example, using all the mentions about different politicians in blog articles, news articles from CNN[13] and BBC[14] and micro-blogging sites such as Twitter, we can rank these politicians based on a user's preferences. These preferences can be attributes such as 'honest' and 'liberal' or the politician's promises such as 'better health care plan' and 'against child abortion', etc. Similarly, using all the reviews from e-commerce sites like Amazon.com[15], BestBuy.com[16] and Walmart.com[17], we can rank products based on the user's preferences. For example, if the user is interested in purchasing a laptop, the user could find laptops based on his/her personal tradeoffs using a set of keywords such as 'lightweight', 'bright screen','highly reliable', 'long battery life' and so on. Thus, instead of reading many reviews for a large number of laptops (to check if the laptop actually satisfies the user's preferences), the entity ranking system tries to shortlist a set of laptops that match these preferences. With this, the user would only need to analyze the laptops ranked by the system.

In terms of accepting a user's preferences, different types of user interfaces may be used. The most general interface would be a single text field that would allow users to express preferences using a natural keyword query. Aspects in the query can then be obtained using query segmentation techniques. Another approach is to ask users to specify a special delimiter to separate their preferences. While this would require just one additional character between two preferences, users could find this requirement rather unnatural to their usual browsing and searching pattern. A more practical user interface would be to provide separate text fields to represent the different preferences. While all these are reasonable suggestions, the question with regards to the best user interface for an entity ranking task such as this remains open until a full user study has been performed.

Our use of retrieval models for this task represents a shallow but general solution to the problem. If we assume that users will only express preferences on

[13] www.cnn.com
[14] www.bbc.com
[15] www.amazon.com
[16] www.bestbuy.com
[17] www.walmart.com

a set of common aspects, then it is possible to leverage existing work in rating prediction [21, 22, 23] to rank entities more accurately based on a user's preferences. Although such a refined approach could lead to more accurate ranking, as we have mentioned in Section 2, these approaches pose practical limitations. With the rating prediction approach, scaling up to different domains would involve a lot more text processing compared to our retrieval based approach. For example, aspect discovery in each domain would be a necessity and once found, users are tied to these limited number of aspects. Further, the rating prediction approaches require some form of supervision such as the presence of overall ratings, which severely limits the type of textual content that can be utilized.

3.7 Conclusions and Future Work

In this thesis, we proposed a novel way of utilizing opinion data - that is to directly rank entities like people, businesses and products based on a user's preferences and existing opinions on those entities. We studied the use of several state-of-the-art retrieval models for this task and propose some new extensions over these models. We also leverage rating information associated with some car and hotel reviews to create a benchmark data set for quantitative evaluation of opinion-based entity ranking.

Experimental results show that the use of opinion expansion is especially effective for improving the ranking of entities according to the user's preferences. We also show that the aspect modeling of queries as opposed to treating queries as set of keywords, is effective on longer queries. While all three state-of-the-art retrieval models show improvement with the proposed extensions, the BM25 retrieval model is most consistent and works especially well with these extensions.

Our evaluation, in two very different domains (cars and hotels), shows that the proposed methods can be directly applied to rank different types of entities for which we have reviews available. We thus believe that this is a very promising line of study with good prospects of practical applications. Our user study shows that the ranking results of entities from the proposed methods have high NDCG values based on human judgments and can be very useful for users to help them choose entities based on opinions.

Our work opens up many interesting future research directions. First, in this thesis, we only explored techniques that are unique to the problem of opinion-based entity ranking. We believe that many of the existing techniques and refinements in information retrieval especially in areas like expert finding can further help in improving the performance of this task. Also, in both query aspect modeling and opinion expansion, we explored some simple ideas in this thesis. The fact that these simple techniques are effective suggests that more sophisticated methods such as structured query language models [68] and sentiment analysis techniques can be potentially leveraged to further improve per-

formance. The data set and evaluation methodology introduced would greatly facilitate further exploration in this direction.

In the next chapter, we will look into our proposed methods for summarization of opinions as part of the analysis tools to facilitate decision making.

[17]The work done in this chapter has been published in (Ganesan & Zhai 2012) [1].

4 GRAPH-BASED APPROACH TO ABSTRACTIVE SUMMARIZATION OF OPINIONS

In providing fine-grained analysis tools as part of the ODSS platform, we propose several abstractive summarization methods. In this Chapter, we describe a graph based approach to abstractive summarization of opinions. This approach takes advantage of the structural redundancies in text, modeled using graphs to generate summaries that are concise, readable and informative. Unlike existing methods in abstractive summarization, this approach is domain independent and lightweight making it suitable for practical use.

4.1 Introduction

Summarization is critically needed to help users better digest the large amounts of opinions expressed on the web. Most existing work in Opinion Summarization focus on predicting sentiment orientation on an entity [24, 69] or attempt to generate aspect-based ratings for that entity [22, 70, 28].

Such summaries are very informative, but it is still hard for a user to understand why an aspect received a particular rating, forcing a user to read many, often highly redundant sentences about each aspect. To help users further digest the opinions in each aspect, it is thus desirable to generate a concise textual summary of such redundant opinions.

Indeed, in many scenarios, we will face the problem of summarizing a large number of highly redundant opinions; other examples include summarizing the 'tweets' on Twitter or comments made about a blog or news article. Due to the subtle variations of redundant opinions, typical extractive methods are often inadequate for summarizing such opinions. Consider the following sentences:

1. *The iPhone's battery lasts long, only had to charge it once every few days.*
2. *iPhone's battery is bulky but it is cheap..*
3. *iPhone's battery is bulky but it lasts long!*

With extractive summarization, no matter which single sentence of the three is chosen as a summary, the generated summary would be biased. In such a case, an abstractive summary such as '*iPhone's battery is cheap, lasts long but is bulky*' is a more complete summary, conveying all the necessary information. Extractive methods also tend to be verbose and this is especially problematic

when the summaries need to be viewed on smaller screens like on a PDA. Thus, an informative and concise abstractive summary would be a better solution.

Unfortunately, abstractive summarization is known to be difficult. Existing work in abstractive summarization has been quite limited and can be categorized into two categories: (1) approaches using prior knowledge [71, 72, 73] and (2) approaches using Natural Language Generation (NLG) systems [74, 75]. The first line of work requires considerable amount of manual effort to define schemas such as frames and templates that can be filled with the use of information extraction techniques. These systems were mainly used to summarize news articles. The second category of work uses deeper NLP analysis with special techniques for text regeneration. Both approaches either heavily rely on manual effort or are domain dependent.

In this paper, we propose a novel flexible summarization framework, Opinosis, that uses graphs to produce abstractive summaries of highly redundant opinions. In contrast with the previous work, Opinosis assumes no domain knowledge and uses shallow NLP, leveraging mostly the word order in the existing text and its inherent redundancies to generate informative abstractive summaries. The key idea of Opinosis is to first construct a textual graph that represents the text to be summarized. Then, three unique properties of this graph are used to explore and score various subpaths that help in generating candidate abstractive summaries.

Evaluation results on a set of user reviews show that Opinosis summaries have reasonable agreement with human summaries. Also, the generated summaries are readable, concise and fairly well-formed. Since Opinosis assumes no domain knowledge and is highly flexible, it can be potentially used to summarize any highly redundant content and could even be ported to other languages. (All materials related to this work including the dataset and demo software can be found at `http://timan.cs.uiuc.edu/downloads.html`.)

4.2 Opinosis-Graph

Our key idea is to use a graph data structure (called Opinosis-Graph) to represent natural language text and cast this abstractive summarization problem as one of finding appropriate paths in the graph. Graphs have been commonly used for extractive summarization (e.g., LexRank [76] and TextRank [77]), but in these works the graph is often *undirected* with *sentences as nodes* and similarity as edges. Our graph data structure is different in that each node represents a *word unit* with *directed edges* representing the structure of sentences. Moreover, we also attach positional information to nodes as will be discussed later.

Algorithm 1 (A1): *OpinosisGraph*(Z)

1: **Input:** Topic related sentences to be summarized: $Z = \{z_i\}_{i=1}^{n}$
2: **Output:** $G = (V, E)$
3: **for** $i = 1$ to n **do**
4: $w \leftarrow Tokenize(z_i)$
5: $sent\_size \leftarrow SizeOf(w)$
6: **for** $j = 1$ to $sent\_size$ **do**
7: $LABEL \leftarrow w_j$
8: $PID \leftarrow j$
9: $SID \leftarrow i$
10: **if** $ExistsNode(G, LABEL)$ **then**
11: $v_j \leftarrow GetExistingNode(G, LABEL)$
12: $PRI_{v_j} \leftarrow PRI_{v_j} \cup (SID, PID)$
13: **else**
14: $v_j \leftarrow CreateNewNode(G, LABEL)$
15: $PRI_{v_j} \leftarrow (SID, PID)$
16: **end if**
17: **if not** $ExistsEdge(v_{j-1} \rightarrow v_j, G)$ **then**
18: $AddEdge(v_{j-1} \rightarrow v_j, G)$
19: **end if**
20: **end for**
21: **end for**

Our graph representation is closer to that used by Barzilay and Lee [78] for the task of paraphrasing, wherein each node in the graph represents a unique word. However, in their work, such a graph is used to identify regions of commonality and variability amongst similar sentences. Thus, the positional information is not required nor is it maintained. In contrast, we maintain positional information at each node as this is critical for the selection of candidate paths.

Algorithm **A1** outlines the steps involved in building an Opinosis-Graph. We start with a set of sentences relevant to a specific topic, which can be obtained in different ways depending on the application. For example, they may be all sentences related to the *battery life* of the *iPod Nano*. We denote these sentences as $Z = \{z_i\}_{i=1}^{n}$ where each z_i is a sentence containing part-of-speech (POS) annotations. (A1:4) Each $z_i \in Z$ is split into a set of *word units*, where each unit, w_j consists of a word and its corresponding POS annotation (e.g. "*service:nn*", "*good:adj*"). (A1:7-9) Each unique w_j will form a node, v_j, in the Opinosis-Graph, with w_j being the label. Also, since we only have one node per unique word unit, each node keeps track of all sentences that it is a part of using a *sentence identifier* (SID) along with its *position of occurrence* in that sentence (PID). (A1:10-16) Each node will thus carry a *Positional Reference Information* (PRI) which is a list of {SID:PID} pairs representing the node's membership in a sentence. (A1:17-19) The original structure of a sentence is recorded with the use of directed edges. Figure 4.1 shows a resulting Opinosis-Graph based on four sentences.

The Opinosis-Graph has some unique properties that are crucial in generating abstractive summaries. We highlight some of the core properties by drawing examples from Figure 4.1:

Property 1. ***(Redundancy Capture)***. *Highly redundant discussions are nat-*

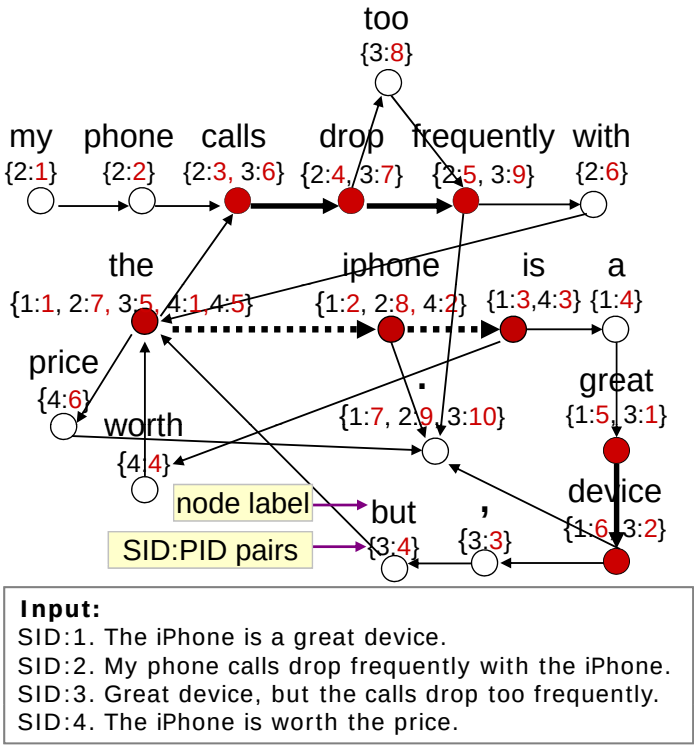

Figure 4.1: Sample *Opinosis-Graph*. Thick edges indicate salient paths.

urally captured by subgraphs.

Figure 4.1 shows that although the phrase '*great device*' was mentioned in different parts of sentences (1) and (3), this phrase forms a relatively heavy sub-path in the resulting graph. This is a good indication of salience.

Property 2. *(**Gapped Subsequence Capture**). Existing sentence structures introduce **lexical links** that facilitate the discovery of new sentences or reinforce existing ones.*

The main point conveyed by sentences (2) and (3) in Figure 4.1 is that *calls drop frequently*. However, this is expressed in slightly different ways and is reflected in the resulting subgraph. Since sentence (2) introduces a *lexical link* between '*drop*' and '*frequently*', the word '*too*' can be ignored for sentence (3) as the same amount of information is retained. This is analogous to capturing a *repetitive gapped subsequence* where similar sequences with minor variations are captured. With this, the subgraph *calls drop frequently* can be considered redundant.

Property 3. *(**Collapsible Structures**). Nodes that resemble hubs are possibly collapsible.*

In Figure 4.1 we see that the subgraph '*the iPhone is*', is fairly heavy and the 'is' node acts like a '*hub*' where it connects to various other nodes. Such a

structure is naturally captured by the Opinosis-Graph and is a good candidate for compression to generate a summary such as '*The iPhone is a great device and is worth the price*'. Also, certain word POS (e.g. linking verbs like 'is' and 'are') often carry hub-like properties that can be used in place of the outlink information.

4.3 Opinosis Summarization Framework

In this section, we describe a general framework for generating abstractive summaries using the Opinosis-Graph. We also describe our implementation of the components in this framework.

At a high level, we generate an abstractive summary by repeatedly searching the Opinosis graph for appropriate subgraphs that both encode a valid sentence (thus meaningful sentences) and have high redundancy scores (thus representative of the major opinions). The sentences encoded by these subgraphs would then form an abstractive summary.

Going strictly by the definition of true abstraction [79], our problem formulation is still more extractive than abstractive because the generated summary can only contain words that occur in the text to be summarized; our problem definition may be regarded as a word-level (finer granularity) extractive summarization. However, compared to the conventional sentence-level extractive summarization, our formulation has flavors of abstractive summarization wherein we have elements of *fusion* (combining extracted portions) and *compression* (squeezing out unimportant material from a sentence). Hence, the sentences in the generated summary are generally not the same as any original sentence. Such a "shallow" abstractive summarization problem is more tractable, enabling us to develop a general solution to the problem. We now describe each component in such a summarization framework.

4.3.1 Valid Path

A valid path intuitively refers to a path that corresponds to a meaningful sentence.

Definition 1. *(**Valid Start Node - VSN**). A node v_q is a valid start node if it is a natural starting point of a sentence.*

We use the positional information of a node to determine if it is a VSN. Specifically, we check if $Average(PID_{v_q}) \leq \sigma_{vsn}$, where σ_{vsn} is a parameter to be empirically set. With this, we only qualify nodes that tend to occur early on in a sentence.

Definition 2. *(**Valid End Node - VEN**). A node v_s is a valid end point if it completes a sentence.*

We use the natural ending points in the text to be summarized as hints to which node may be a valid end point of a path (i.e., a sentence). Specifically, a node is a *valid end node* if (1) the node is a punctuation such as *period* and *comma* or (2) the node is any coordinating conjunction (e.g., *'but'* and *'yet'*).

Definition 3. ***(Valid Path)****. A path $W = \{v_q...v_s\}$ is valid if it is connected by a set of directed edges such that (1) v_q is a* ***VSN****, (2) v_s is a* ***VEN****, and (3) W satisfies a set of well-formedness POS constraints.*

Since not every path starting with a VSN and ending at a VEN encodes a meaningful sentence, we further require a valid path to satisfy the following POS constraints (expressed in regular-expression) to ensure that a valid path encodes a well-formed sentence:

1. $.*(/nn)+.*(/vb)+.*(/jj)+.*$
2. $.*(/jj)+.*(/to)+.*(/vb).*$
3. $.*(/rb)*.*(/jj)+.*(/nn)+.*$
4. $.*(/rb)+.*(/in)+.*(/nn)+.*$

This also provides a way (if needed) for the application to generate only specific type of sentences like *comparative sentences* or *strictly opinionated sentences*. These rules are thus application specific.

4.3.2 Path Scoring

Intuitively, to generate an abstractive summary, we should select a valid path that can represent most of the redundant opinions well. We would thus favor a valid path with a high redundancy score.

Definition 4. ***(Path Redundancy)****. Let $W = \{v_q...v_s\}$ be a path from an Opinosis-Graph. The path redundancy of W, $r(q,s)$, is the number of overlapping sentences covered by this path, i.e.,*

$$r(q,s) = n_q \bar{\cap} n_{q+1} ... \bar{\cap} n_s,$$

where $n_i = PRI_{v_i}$ and $\bar{\cap}$ is the intersection between two sets of SIDs such that the difference between the corresponding PIDs is no greater than σ_{gap}, and $\sigma_{gap} > 0$ is a parameter.

Path redundancies provide good indication of how many sentences discuss something similar at each point in the path. The σ_{gap} parameter controls the maximum allowed gaps in discovering these redundancies. Thus, a common sentence X between nodes v_q and v_r, will be considered a valid intersect if $(PID_{v_{r_x}} - PID_{v_{q_x}}) \leq \sigma_{gap}$.

Based on path redundancy, we propose several ways to score a path for the purpose of selecting a good path to include in the summary:

1. $S_{basic}(W) = \frac{1}{|W|} \sum_{k=i+1}^{s} r(i,k)$
2. $S_{wt\_len}(W) = \frac{1}{|W|} \sum_{k=i+1}^{s} |v_i, v_k| * r(i,k)$

3. $S_{wt\_loglen}(W) = \frac{1}{|W|}(r(i,i+1) + \sum_{k=i+2}^{s} log_2|v_i, v_k| * r(i,k))$

v_i is the first node in the path being scored and v_s is the last node. $|v_i, v_k|$ is the length from node v_i to v_k. $|W|$ is the length of the entire path being scored. The S_{basic} scoring function scores a path purely based on the level of redundancy. One could also argue that high redundancy on a longer path is intuitively more valuable than high redundancy on a shorter path as the former would provide better coverage than the latter. This intuition is factored in by the S_{wt_len} and S_{wt_loglen} scoring functions where the level of *redundancy* is *weighted* by the *path length*. S_{wt_loglen} is similar to S_{wt_len} only that it scales down the path length so that it does not entirely dominate.

4.3.3 Collapsed paths

In some cases, paths in the Opinosis-Graph may be collapsible (as explained in Section 4.2). In such a case, the collapse operation is performed and then the path scores are computed. We will now explain a few concepts related to collapsible structures. Let $\widehat{W} = \{v_i...v_k\}$ be a path from the Opinosis-Graph.

Definition 5. *(**Collapsible Node**). Node v_k is a candidate for collapse if its POS is a* verb.

We only attempt to collapse nodes that are *verbs* due to the heavy usage of verbs in opinion text and the ease with which the structures can be combined to form a new sentence. However, as mentioned earlier other properties like the outlink information can be used to determine if a node is collapsible.

Definition 6. *(**Collapsed Candidates, Anchor**). Let v_k be a collapsible node. The collapsed candidates of v_k (denoted by $CC = \{cc_i\}_{i=1}^{m}$) are the remaining paths after v_k in all the valid paths going through $v_i...v_k$. The prefix $v_i...v_k$ is called the anchor, denoted as $C_{anchor} = \{v_i...v_k\}$. Each path $\{v_i...v_n\}$, where v_n is the last node in each $cc_i \in CC$, is an individually valid path.*

Table 4.1 shows a simplistic example of anchors and corresponding collapsed candidates. Once the anchor and collapsed candidates have been identified, the task is then to combine all of these to form a new sentence.

Definition 7. *(**Stitched Sentence**) A stitched sentence is one that combines C_{anchor} and CC to form a combined, logical sentence.*

We will now describe the stitching procedure that we use, by drawing examples from Table 4.1. Since we are dealing with verbs, C_{anchor} can be combined with the corresponding CC with commas to separate each $cc_i \in CC$ with one exception - the correct sentence connector has to be used for the last cc_i. For C_{anchor_a}, the phrases *really good* and *clear* can be connected by 'and' due to the same sentiment orientation. For C_{anchor_b}, the collapsed candidate phrases

C_{anchor}	CC	Connector
a. the sound quality is	cc_1 : really good cc_2 : clear	and
b. the iphone is	cc_1 : great cc_2 : expensive	but

Table 4.1: Example of anchors, collapsed candidates and suitable connectors

are well connected by the word 'but'. We use the existing Opinosis-Graph to determine the most appropriate connector. We do this by looking at all *coordinating conjunction* (e.g. 'but', 'yet') nodes (v_{cconj}) that are connected to the first node of the last collapsed candidate, cc_m. This would be the node labeled '*clear*' for C_{anchor_a} and '*expensive*' for C_{anchor_b}. We denote these nodes as v_{0,cc_m}. The v_{cconj}, with the highest *path redundancy* with v_{0,cc_m}, will be selected as the connector.

Definition 8. ***(Collapsed Path Score)*** *The final path score after the entire collapse operation is the average across path scores computed from v_i to the last node in each $cc_i \in CC$.*

The collapsed path score essentially involves computing the *path scores* of the individual sentences assuming that they are not collapsed and then averaging them.

4.3.4 Generation of summary

Once we can score all the valid paths as well as all the collapsed paths, the generation of an abstractive summary can be done in two steps: First, we rank all the paths (including the collapsed paths) in descending order of their scores. Second, we eliminate duplicated (or extremely similar) paths by using a similarity measure (in our experiments, we used Jaccard). We then take the top few remaining paths as the generated summary, with the number of paths to be chosen controlled by a parameter σ_{ss}, which represents summary size.

Although conceptually we enumerate all the valid paths, in reality we can use a redundancy score threshold, σ_r to prune many non-promising paths. This is reasonable because we are only interested in paths with high redundancy scores.

4.4 Summarization Algorithm

Algorithms **A2** and **A3** describe the steps involved in Opinosis Summarization. A2 is the starting point of the Opinosis Summarization and A3 is a subroutine where path finding takes place, invoked from within A2.

Algorithm 2 (A2): *OpinosisSummarization*(Z)

1: **Input:** Topic related sentences to be summarized: $Z = \{z_i\}_{i=1}^{n}$
2: **Output:** $\mathcal{O}$ ={Opinosis Summaries}
3: $g \leftarrow OpinosisGraph(Z)$
4: $node\_size \leftarrow SizeOf(g)$
5: **for** $j = 1$ to $node\_size$ **do**
6: **if** $VSN(v_j)$ **then**
7: $pathLen \leftarrow 1$
8: $score \leftarrow 0$
9: $cList \leftarrow CreateNewList()$
10: **Traverse**$(cList, v_j, score, PRI_{v_j}, label_{v_j}, pathLen)$
11: $candidates \leftarrow \{candidates \cup cList\}$
12: **end if**
13: **end for**
14: $\mathcal{C} \leftarrow EliminateDuplicates(candidates)$
15: $\mathcal{C} \leftarrow SortByPathScore(\mathcal{C})$
16: **for** $i = 1$ to σ_{ss} **do**
17: $\mathcal{O} = \{\mathcal{O} \cup PickNextBestCandidate(\mathcal{C})\}$
18: **end for**

(A2:3) Opinosis Summarization starts with the construction of the Opinosis-Graph, described in detail in Section 4.2. This is followed by the *depth first* traversal of this graph to locate *valid paths* that become *candidate summaries*. (A2:6-12) To achieve this, each node v_j in the Opinosis-Graph is examined to determine if it is a *VSN* and, if it is, path finding will start from this node by invoking subroutine A3. A3 takes the following as input: *list* - a list to hold candidate summaries; v_i - the node to continue traversal from; *score* - the accumulated path score; $PRI_{overlap}$ - the intersect between PRIs of all nodes visited so far (see Definition 4); *sentence* - the summary sentence formed so far; *len* - the current path length. (A2:7-10) Before invoking A3 from A2, the path length is set to '1', path score is set to '0' and a new list is created to store candidate summaries generated from node v_j. (A2:11) All candidate summaries generated from v_j will be stored in a common pool of candidate summaries.

(A3:3-4) Algorithm A3 starts with a check to ensure that the minimum path redundancy requirement is satisfied (see definition 4). For the very first node sent from A2, the path redundancy is the size of the raw PRI. (A3:5-10) If the redundancy requirement is satisfied, a few checks are done to determine if a *valid path* has been found. If it has, then the resulting sentence and its final score are added to the list of candidate summaries.

(A3:11-31) Traversal proceeds recursively through the exploration of all neighboring nodes of the current node, v_k. (A3:12-16) For every neighboring node, v_n the PRI overlap information, path length, summary sentence and path score are updated before the next recursion. (A3:29) If a v_n is not collapsible, then a regular traversal takes place. (A3:17-27) However, if v_n is collapsible, the updated sentence in A3:14, will now serve as an *anchor* in A3:18. (A3:21) A3 will then attempt to start a recursive traversal from all neighboring nodes of v_n in order to find corresponding collapsed candidates. (A3:22-26) After this, duplicates are eliminated from the *collapsed candidates* and the *collapsed path*

Algorithm 3 (A3): *Traverse*(...)

1: **Input:** $list$, $v_k \subseteq V$, $score$, $PRI_{overlap}$, $sentence$, len
2: **Output:** A set of candidate summaries
3: $redundancy \leftarrow SizeOf(PRI_{overlap})$
4: **if** $redundancy \geq \sigma_r$ **then**
5: **if** $VEN(v_k)$ **then**
6: **if** $ValidSentence(sentence)$ **then**
7: $finalScore \leftarrow \frac{score}{len}$
8: $AddCandidate(list, sentence, finalScore)$
9: **end if**
10: **end if**
11: **for** $v_n \in Neighbors_{v_k}$ **do**
12: $PRI_{new} \leftarrow PRI_{overlap} \ \widetilde{\cap} \ PRI_{v_n}$
13: $redundancy \leftarrow SizeOf(PRI_{new})$
14: $newSent \leftarrow Concat(sentence, label_{v_n})$
15: $L \leftarrow len + 1$
16: $newScore \leftarrow score + PathScore(redundancy, L)$
17: **if** $Collapsible(v_n)$ **then**
18: $C_{anchor} \leftarrow newSent$
19: $tmp \leftarrow CreateNewList()$
20: **for** $v_x \in Neighbors_{v_n}$ **do**
21: **Traverse**$(tmp, v_x, 0, PRI_{new}, label_{v_x}, L)$
22: $CC \leftarrow EliminateDuplicates(tmp)$
23: $CCPathScore \leftarrow AveragePathScore(CC)$
24: $finalScore \leftarrow newScore + CCPathScore$
25: $stitchedSent \leftarrow Stitch(C_{anchor}, CC)$
26: $AddCandidate(list, stitchedSent, finalScore)$
27: **end for**
28: **else**
29: **Traverse**$(list, v_n, newScore, PRI_{new}, newSent, L)$
30: **end if**
31: **end for**
32: **end if**

score is computed. The resulting *stitched sentence* and its *final score* are then added to the original list of candidate summaries.

(A2:14-18) Once all paths have been explored for candidate generation, duplicate candidates are removed and the remaining are sorted in descending order of their path scores. The best σ_{ss} candidates are 'picked' as final Opinosis summaries.

v_i is the first node in the path being scored and v_s is the last node. $|v_i, v_k|$ is the length from node v_i to v_k. $|W|$ is the length of the entire path being scored. The S_{basic} scoring function scores a path purely based on the level of redundancy. One could also argue that high redundancy on a longer path is intuitively more valuable than high redundancy on a shorter path as the former would provide better coverage than the latter. This intuition is factored in by the S_{wt_len} and S_{wt_loglen} scoring functions where the level of *redundancy* is *weighted* by the *path length*. S_{wt_loglen} is similar to S_{wt_len} only that it scales down the path length so that it does not entirely dominate.

4.5 Experimental Setup

We evaluate this abstractive summarization task using reviews of *hotels*, *cars* and various *products*[1]. Based on these reviews, 2 humans were asked to construct 'opinion seeking' queries which would consist of an *entity name* and a *topic of*

[1]Reviews collected from Tripadvisor, Amazon, Edmunds

interest. Example of such queries are: *Amazon Kindle:buttons*, *Holiday Inn, Chicago: staff*, and so on. We compiled a set of 51 such queries. We create one review document per query by collecting all review sentences that contain the query words for the given entity. Each review document thus consists of a set of unordered, redundant review sentences related to the query. There are approximately 100 sentences per review document.

We use ROUGE [80] to quantitatively assess the agreement of Opinosis summaries with human composed summaries. ROUGE is based on an n-gram co-occurrence between machine summaries and human summaries and is a widely accepted standard for evaluation of summarization tasks. In our experiments, we use ROUGE-1, ROUGE-2 and ROUGE-SU4 measures. ROUGE-1 and ROUGE-2 have been shown to have most correlation with human summaries [81] and higher order ROUGE-N scores ($N > 1$) estimate the fluency of summaries.

We use multiple reference (human) summaries in our evaluation since it can achieve better correlation with human judgment [82]. We leverage Amazon's Online Workforce[2] to get 5 different human workers to summarize each review document. The workers were asked to be concise and were asked to summarize the major opinions in the review document presented to them. We manually reviewed each set of reference summaries and dropped summaries that had little or no correlation with the majority. This left us with around 4 reference summaries for each review document.

To allow performance comparison between humans, Opinosis and the baseline method, we implemented a Jackknifing procedure where, given K references, the ROUGE score is computed over K sets of K-1 references. With this, average human performance is computed by treating each reference summary as a 'system' summary, computing ROUGE scores over the remaining K-1 reference summaries.

Due to the limited work in abstractive summarization, no natural baseline could be used for comparison. The existing work in this area is mostly domain dependent and requires too much manual effort (explained in Section 4.1). The next best baseline is to use a state of the art extractive method. Thus, we use MEAD [83] as our baseline. MEAD is an extractive summarizer based on cluster centroids. It uses a collection of the most important words from the whole cluster to select the best sentences for summarization. By default, the scoring of sentences in MEAD is based on 3 parameters - *minimum sentence length*, *centroid*, and *position in text*. MEAD was ideal for our task because a good summary in our case would be one that could capture the most essential information. This is exactly what centroid-based summarization aims to achieve. Also, since the *position in text* parameter is irrelevant in our case, we could easily turn this off with MEAD.

We introduce a **readability test** to understand if Opinosis summaries are in fact readable. Suppose we have N sentences from a system-generated summary

[2]https://www.mturk.com

Recall				
	ROUGE-1	ROUGE-2	ROUGE-SU4	Avg # Words
Human	0.3184	**0.1106**	0.1293	17
Opinosis	0.2831	0.0853	0.0851	15
Baseline	**0.4932**	0.1058	**0.2316**	75
Precision				
	ROUGE-1	ROUGE-2	ROUGE-SU4	Avg # Words
Human	0.3434	0.1210	0.1596	17
Opinosis	**0.4482**	**0.1416**	**0.2261**	15
Baseline	0.0916	0.0184	0.0102	75
F-score				
	ROUGE-1	ROUGE-2	ROUGE-SU4	Avg # Words
Human	0.3088	**0.1069**	**0.1142**	17
Opinosis	**0.3271**	0.0998	0.1027	15
Baseline	0.1515	0.0308	0.0189	75

Table 4.2: Performance comparison between Humans, Opinosis$_{best}$ and Baseline.

and M sentences from corresponding human summaries. We mix all these sentences and then ask a human assessor to pick at most N sentences that are *least readable* as the prediction of system summary.

$$readability(\mathcal{O}) = 1 - \frac{\#CorrectPick}{N}$$

If the human assessor often picks out system generated summaries as being least readable, then the readability of system summaries is poor. If not, then the system generated summaries are no different from human summaries.

4.6 Results

The baseline extractive method (MEAD) selects 2 most representative sentences as summaries. For a fair comparison, we fix the Opinosis summary size, $\sigma_{ss} = 2$. We also fix $\sigma_{vsn} = 15$. The best Opinosis configuration with $\sigma_{ss} = 2$ and $\sigma_{vsn} = 15$ is called Opinosis$_{best}$ ($\sigma_{gap} = 4, \sigma_r = 2$, S_{wt_loglen}). ROUGE scores reported are with the use of *stemming* and *stopword removal*.

Performance comparison between humans, Opinosis and baseline. Table 4.2 shows the performance comparison between humans, Opinosis$_{best}$ and the baseline method. First, we see that the baseline method has very high recall scores compared to Opinosis. This is because extractive methods that just 'select' sentences tend to be much longer resulting in higher recall. However, these summaries tend to carry information that may not be significant and is clearly reflected by the poor precision scores.

Next, we see that humans have reasonable agreement amongst themselves

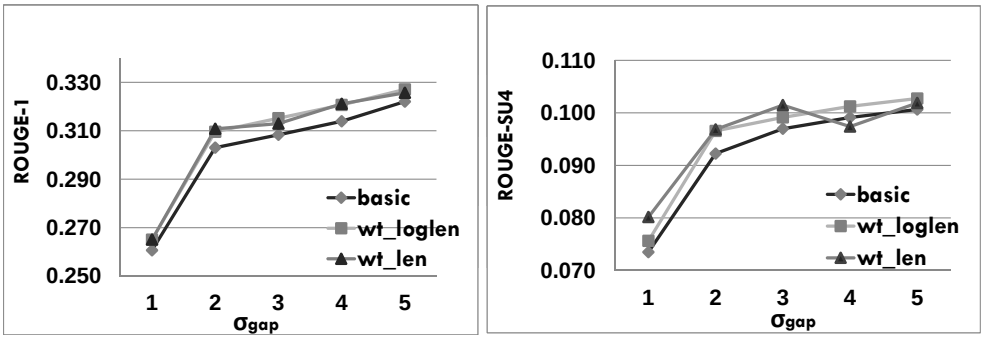

Figure 4.2: ROUGE scores (f-measure) at different levels of σ_{gap}, $\sigma_r = 2$.

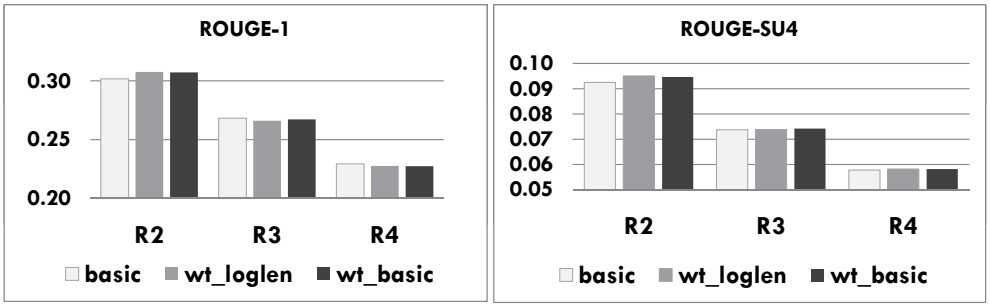

Figure 4.3: ROUGE scores (f-measure) at different levels of σ_r averaged across $\sigma_{gap} \in [1, 5]$

given that these are independently composed summaries. This agreement is especially clear with the ROUGE-2 recall score where the recall is better than Opinosis but comparable to the baseline even though the summaries are much shorter. It is also clear that Opinosis is closer in performance to humans than to the baseline method. The recall scores of Opinosis summaries are slightly lower than that achieved by humans, while the precision scores are higher (Wilcoxon test shows that the increase in precision is statistically more significant than the decrease in recall). In terms of f-scores, Opinosis has the best ROUGE-1 score and its ROUGE-2 and ROUGE-SU4 scores are comparable with human performance. The baseline method has the lowest f-scores. The difference between the f-scores of Opinosis and that of humans is statistically insignificant.

Comparison of scoring functions. Next, we look into the performance of the three scoring functions, S_{basic}, S_{wt_len} and S_{wt_loglen} described in Section 4.3. Figure 4.2 shows ROUGE scores of these scoring methods at varying levels of σ_{gap}. First, it can be observed that S_{wt_basic} which does not use *path length* information, performs the worst. This is due to the effect of heavily favoring redundant paths over longer but reasonably redundant ones that can provide more coverage. We also see that S_{wt_len} and S_{wt_loglen} are similar in performance with S_{wt_loglen} marginally outperforming S_{wt_len} when $\sigma_{gap} > 2$. Since S_{wt_len}

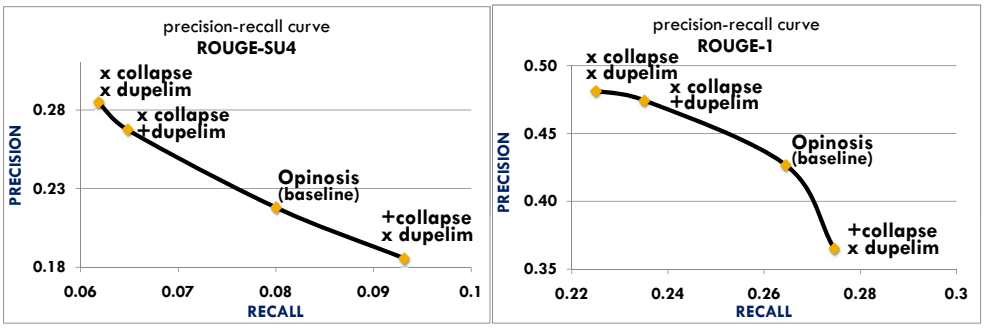

Figure 4.4: Precision-Recall comparison with different Opinosis features turned off.

uses the raw *path length* in its scoring function, it may be inflating the path scores of long but insignificant paths. S_{wt_loglen} scales down the path length, thus providing a reasonable tradeoff between redundancy and the length of the selected path. The three scoring functions are not influenced by different levels of σ_r as shown in Figure 4.3.

Effect of gap setting (σ_{gap}). Now, we will examine the effect of σ_{gap} on the generated summaries. Based on Figure 4.2, we see that setting σ_{gap}=1 yields in relatively low performance. This is because σ_{gap}=1 implies immediate adjacency between the PIDs of two nodes and such strict adjacency enforcements prevent redundancies from being discovered. When σ_{gap} is increased to 2, there is a big jump in performance, after which improvements are observed in smaller amounts. A very large gap setting could increase the possibility of generating ill-formed sentences, thus we recommend that σ_{gap} is set between 2-5.

Effect of redundancy requirement (σ_r) . Figure 4.3 shows the ROUGE scores at different levels of σ_r. It is clear that when $\sigma_r > 2$, the quality of summaries is negatively impacted. Since we only have about 100 sentences per review document, $\sigma_r > 2$ severely restricts the number of paths that can be explored, yielding in lower ROUGE scores. Since the scoring function can account for the level of redundancy, σ_r should be set according to the size of the input data. For our dataset, $\sigma_r = 2$ was ideal.

Effect of collapsed structures and duplicate elimination. So far, it has been assumed that all features used in Opinosis are required to generate reasonable summaries. To test this hypothesis, we use Opinosis$_{best}$ as a baseline and then we turn off different features of Opinosis. We turn off the *duplicate elimination feature*, then the *collapsible structure feature*, and finally *both*. Figure 4.4 shows the resulting precision-recall curve. From this graph, we see that without duplicate elimination and when collapsing is turned off, the precision is highest but recall is lowest. No collapsing implies shorter sentences and thus lower recall, which is clearly reflected in Figure 4.4. On top of this, if duplicates are allowed, the overall information coverage is low, further affecting the recall.

"About food at Holiday Inn, London"

Human summaries:

[1] Food was excellent with a wide range of choices and good services.

[2] The food is good, the service great. Very good selection of food for breakfast buffet.

Opinosis abstractive summary:

The food was excellent, good and delicious. Very good selection of food.

Baseline extractive summary:

Within 200 yards of leaving the hotel and heading to the Tube Station you have a number of fast food outlets, highstreet Restautants, Pastry shops and supermarkets, so if you did wish to live in your hotel room for the duration of your stay, you could do.....

"What is free at Bestwestern Inn, San Francisco"

Human summaries:

[1] There is free WiFi internet access available in all the rooms.. From 5-6 p.m. there is free wine tasting and appetizers available to all the guests.

[2] Evening wine reception and free coffee in the morning. Free internet, free parking and free massage.

Opinosis abstractive summary:

Free wine reception in evening. Free coffee and biscotti and wine.

Baseline extractive summary:

The free wine and nibbles served between 5pm and 6pm were a lovely touch. There's free coffee, teas at breakfast time with little biscotti and, best of all, from 5 till 6pm you get a free wine 'tasting' reception which, as long as you don't take……

Figure 4.5: Sample results comparing Opinosis summaries with human and baseline summaries.

Notice that the presence of duplicates with the collapse feature turned on results in very high recall (even higher than the baseline). This is caused by the presence of similar phrases that were not eliminated from the collapsed candidates, resulting in long sentences that artificially boost recall. The Opinosis baseline which uses duplicate elimination and the collapsible structure feature, offers a reasonable tradeoff between precision and recall.

Readability of Summaries. To test the readability of Opinosis summaries, we conducted a *readability test* (described in Section 4.5) using summaries generated from Opinosis$_{best}$. A human assessor picked the 2 least readable sentences from each of the 51 test sets (based on 51 summaries). Collectively, there were 565 sentences out of which 102 were Opinosis generated. Out of these, the human assessor picked only 34 of the sentences as being least readable, resulting in an average readability score of 0.67. This shows that more than 60% of the generated sentences are indistinguishable from human composed sentences. Of the 34 sentences with problems, 11 contained *no information* or were *incomprehensible*, 12 were *incomplete* possibly due to false positives when the sentence validity check was done, and 8 had *conflicting information* such as '*the hotel room is clean and dirty*'. This happens due to mixed feelings about the same topic and can be resolved using sentiment analysis. The remaining 3 sentences were found to contain *poor grammar*, possibly caused by the gaps allowed in finding redundant paths.

Sample Summaries. Finally, in Figure 4.5 we show two sample summaries on two different topics. Notice that the Opinosis summaries are concise, fairly well-formed and have closer resemblance to human summaries than to the baseline summaries.

4.7 Conclusion

In this chapter, we described a novel summarization framework (Opinosis) that uses textual graphs to generate abstractive summaries of highly redundant opinions. Evaluation results on a set of review documents show that Opinosis summaries have better agreement with human summaries compared to the baseline extractive method. The Opinosis summaries are concise, reasonably well-formed and communicate essential information. Our readability test shows that more than 60% of the generated sentences are no different from human composed sentences.

Opinosis is a flexible framework in that many of its modules can be easily improved or replaced with other suitable implementation. Also, since Opinosis is domain independent and relies on minimal external resources, it can be used with any corpus containing high amounts of redundancies.

Our graph representation naturally ensures the coherence of a summary, but such a graph emphasizes too much on the surface order of words. As a result, it

cannot group sentences at a deep semantic level. To address this limitation, we can use a similar idea to overlay parse trees and this would be a very interesting future research.

[2]The work done in this chapter has been published in (Ganesan et al. 2010) [2]

5 LEVERAGING WEB-NGRAMS TO GENERATE ULTRA-CONCISE SUMMARIES OF OPINIONS

In the previous chapter, we described our first abstractive summarization method using a graph data structure to model the structural redundancies in text. In this chapter, we study a more generative approach to summarization. We propose to leverage publicly available Web-Ngrams to generate ultra-concise summaries of opinions using a formal optimization framework. We give special importance to the conciseness of summaries (captured by the optimization framework) so that summaries can be displayed on various screen sizes. This approach just as the previous one, is lightweight and general, and requires no linguistics or domain knowledge.

5.1 Introduction

In this chapter, we explore the task of generating a set of very concise phrases, where each phrase (micropinion) is a summary of a key opinion in text. The ultra-concise nature of the phrases allows for flexible adjustment of summary size according to the display constraints. Our emphasis on generating **concise** abstractive summaries (rather than extractive summaries), makes this a unique summarization problem which has not been previously studied. In Table 5.1, we show examples of envisioned micropinion summaries.

On the surface, our summarization task appears to be similar to a keyphrase extraction problem. However, since the goal is to help users digest the underlying opinions, there are some important aspects that are unique to this task. In traditional keyphrase extraction, the goal is primarily to select a set of phrases to characterize documents. Thus, phrases such as *battery life* and *screen* from a set of reviews about a phone may be selected as keyphrases. For our task, such keyphrases are meaningless without the associated opinions. In addition, as we want readers to *understand* the opinions in the summary, the phrases in the

Mp3 Player Y	**Restaurant X**
Very short battery life. Big and clear screen. **(8 words)**	Good service. Delicious soup dishes. Very noisy at nights. **(9 words)**

Table 5.1: Example of micropinion summaries on two different topics given a constraint of 10 words.

summary need to be fairly well-formed and grammatically sound. Consider a phrase such as *'short battery life'* in contrast to one such as *'life short battery'*. Even though both phrases contain the same words, the ordering is different, changing their meaning, where the former is readable and the latter is not. This readability aspect is less of a concern in traditional keyphrase extraction as the phrases are only used to 'tag' documents.

In this chapter, we present a novel *unsupervised* approach to generating *micropinion* summaries for different display constraints. Our idea is to start with a set of high frequency seed words from the input text, gradually forming meaningful higher order n-grams. At each step the n-grams are scored based on their *representativeness* and *readability*. We frame this problem as an optimization problem, where we attempt to find a set of concise, non-redundant phrases (not necessarily occurring in the original text) that are readable and represent the major opinions. We propose heuristic algorithms to solve this optimization problem efficiently.

Evaluation results using a set of product reviews shows that our approach is effective in generating micropinion summaries and outperforms other summarization approaches. Further, the proposed approach is lightweight and general, requiring no linguistics or domain knowledge. It can thus be used in a variety of domains and could even be used with other languages. The dataset and demo would be publicly available upon publication.

5.2 An Optimization Formulation for Micropinion Summarization

The goal of our task is to generate a compact and informative summary using a set of micropinions. A micropinion is a short phrase (between 2 and 5 words) summarizing a key opinion in text. The minimum phrase length of 2 is based on the observation that a meaningful opinion is often targeted towards an object [84] (e.g. *clear screen* vs. *clear*). The maximum phrase length of just 5 is to allow for flexibile adjustment of summary size according to the display requirements. For example, a small phone may have stricter display requirements than a full sized PDA. Thus, compared with most existing work in text summarization, a unique aspect of our goal is to maximize information conveyed in the given constraints.

Formally, given a set of sentences $Z = \{z_i\}_{i=1}^{n}$ from an opinion document, our goal is to generate a micropinion summary, $M = \{m_i\}_{i=1}^{k}$, where $|m_i| \in [2, 5]$ words and each $m_i \in M$ conveys a key opinion from Z. Note that while we require m_i to use words that have occurred at least once in Z, we do *not* require m_i to be an exact subsequence of any of the sentences in Z. This makes our task setup more of an *abstractive summarization* problem. In contrast to the predominantly popular extractive summarization task, this raises a new

interesting challenge in how to *compose* concise and meaningful summaries using only words from the original text. This requirement is not restrictive since user generated content such as opinions have the benefit of volume and with this there would be many choices of words to describe a major opinion.

Intuitively, we would like the generated micropinion summary, $M = \{m_i\}_{i=1}^{k}$ to be: (1) *representative* (i.e., each m_i should reflect the major opinions in the original text), (2) *readable* (i.e., each m_i should be well formed according to the language's grammatical rules), and (3) *compact* (i.e., M should use as few words as possible to convey the major opinions). Thus, in theory, we can solve this new summarization problem by enumerating all possible summaries and evaluating each one to see how well it satisfies these three criteria, which suggests that we can formulate micropinion summarization as the following optimization problem:

$$M^* = \arg\max_{M=\{m_1,\ldots,m_k\}} \sum_{i=1}^{k} [S_{rep}(m_i) + S_{read}(m_i)]$$

$$\begin{aligned} \textit{subject to} \quad & \textstyle\sum_{i=1}^{k} |m_i| \leq \sigma_{ss} \\ & S_{rep}(M) \geq \sigma_{rep} \\ & S_{read}(M) \geq \sigma_{read} \\ & sim(m_i, m_j) \leq \sigma_{sim} \forall, i, j \in [1, k] \end{aligned}$$

where

1. $S_{rep}(m_i)$ is a scoring function measuring the representativeness of m_i;
2. $S_{read}(m_i)$ is a scoring function measuring the readability of m_i;
3. $sim(m_i, m_j)$ is a similarity function measuring the similarity of m_i and m_j;
4. σ_{ss}, σ_{rep}, σ_{read}, and σ_{sim} are four thresholds for the maximum length of the summary, minimum representativeness, minimum readability and maximum similarity.

The rationale for this optimization formulation is the following: First, the objective function captures the intention of optimizing both *representativeness* and *readability*. Second, the compactness is captured by setting a threshold on the maximum length of the summary and a threshold on the similarity between any two phrases in the summary so as to minimize redundancy. Finally, we impose two thresholds for minimum representativeness and readability respectively for two reasons. First, this ensures efficiency by not exploring non-promising candidate phrases. In turn, this also ensures a summary to be both representative and readable (i.e., avoid "skewed tradeoffs").

Clearly, the main challenge now is to define the representativeness function $S_{rep}(m_i)$, the readability function $S_{read}(m_i)$, and the similarity function

$sim(m_i, m_j)$. In this chapter, we focus on studying how to define $S_{read}(m_i)$ and $S_{rep}(m_i)$ as they represent interesting new challenges raised by micropinion summarization. For $sim(m_i, m_j)$, we use the Jaccard similarity [85] to measure and eliminate redundancy.

In Section 5.2.1, we will explain the proposed method to measure the representativeness of m_i, $S_{rep}(m_i)$, based on the pointwise mutual information of the words in m_i. This captures how well m_i aligns with major opinions in the original text. Then in Section 5.2.2, we describe how we estimate readability, $S_{read}(m_i)$, of phrases based on a general n-gram language model with the assumption that m_i is more readable if m_i is more frequent according to the n-gram model.

5.2.1 Representativeness

In general, opinions are often redundant and may contain contradicting viewpoints. Hence, generating a few highly representative phrases is a challenge. Since we are mainly interested in summarizing the *major* opinions in text, a representative summary would be one that can accurately bring to surface the most common *complaint*, *praise* or *critical information*. For example, assuming we have 10 sentences in the input document that talks about "battery life being short" and one about "battery life being excellent", by our definition, the former would be the representative opinion phrase.

In determining the representativeness of a phrase m_i, we have defined two key properties of a highly representative phrase: (1) the words in each m_i, should be strongly associated within a narrow window in the original text and (2) the words in m_i should be sufficiently frequent in the original text.

The first property ensures that only a set of related words are used in the generated phrases to avoid conveying incorrect information. As an example, if we were to generate a phrase containing the word *short*, it is important that *short* is used with the right set of words or we may convey information not present in the original text (e.g. *the phone is short* instead of *battery life is short*). While this is not a problem for methods that use existing n-grams from the input document, it is important for our method as we form n-grams from seed words. This first property of strong association can be captured by computing the *pointwise mutual information* (PMI) of words in m_i based on its alignment with the original text. Note that PMI was shown to be the best metric to measure strength of association of word pairs [86]. For a set of strongly associated words to be considered representative, these words should also be significant in the input text. The second property thus rewards n-grams containing words that occur frequently in the original text. Formally, suppose $m = w_1...w_n$ is a candidate phrase. We define $S_{rep}(m)$ as follows:

$$S_{rep}(w_1...w_n) = \frac{1}{n}\sum_{i=1}^{n} pmi_{local}(w_i) \tag{5.1}$$

where $pmi_{local}(w_i)$ is a local pointwise mutual information function defined as:

$$pmi_{local}(w_i) = \frac{1}{2C}\sum_{j=i-C}^{i+C} pmi'(w_i, w_j), i \neq j \tag{5.2}$$

where C is a contextual window size. The $pmi_{local}(w_i)$ measures the average strength of association of a word w_i with all its C neighboring words (on the left and on the right). So, for the phrase *short battery life*, assuming $C = 1$, for *short* we would obtain the average PMI score of *short* with 'battery' and for *battery* we would obtain the average PMI of *battery* with 'short' and *battery* with 'life'. When this is done for each $w_i \in m$, this would give a good estimate of how strongly associated the words are in m, which is the rationale for Equation 5.1. We use a modified PMI scoring referred to as pmi' where the pmi' between two words, w_i and w_j is defined as:

$$pmi'(w_i, w_j) = \log_2 \frac{p(w_i, w_j) \cdot c(w_i, w_j)}{p(w_i) \cdot p(w_j)} \tag{5.3}$$

where $c(w_i, w_j)$ is the frequency of two words co-occurring in a sentence from the original text within the context window of C (in any direction) and $p(w_i, w_j)$ is the corresponding joint probability. We later show the influence of C on the generated summaries. The co-occurrence frequency, $c(w_i, w_j)$ which is not part of the original PMI formula is integrated into our PMI scoring to reward frequently occurring words from the original text (based on property (2)). The problem with the original PMI scoring is that it yields in high scores for low frequency words. By adding $c(w_i, w_j)$ into the PMI scoring, we ensure that low frequency words do not dominate and moderately associated words with high co-occurrences have relatively high scores.

5.2.2 Readability

For a summary to be readable, it will have to be fairly well-formed and grammatically sound according to the language's grammatical rules. The readability aspect is an important requirement in any summarization task as this allows a reader to easily digest information. In extractive summarization, the readability of a summary is less of a problem as *existing sentences* or *phrases* are reused to form summaries. In our approach, we do not reuse sentences or phrases directly from Z, but rather attempt to generate new phrases using existing words from Z. Hence, there is no guarantee that the generated phrases would be well-formed and readable. For example, without readability scoring, it will be hard to distinguish the grammatical difference between the phrases *The good iPhone is* and *The iPhone is good*.

Without any form of supervision, measuring the readability of a phrase is difficult. We address this problem by leveraging the publicly available Microsoft N-gram service[1] [87], to score all our system generated phrases. The abundance of textual content on the web which includes blogs, news articles, user reviews, tutorials, etc makes an n-gram model estimated based on all such content an ideal judge of how readable the system generated phrases are. The intuition is that if a generated phrase occurs frequently on the web, then this phrase is readable. This approximation to determining readability is fair, since the web as a corpus, is extremely large and there would be enough evidence to segregate a well-constructed phrase from a poorly constructed one.

Specifically, we use the Microsoft's trigram language model trained on the body text of documents to obtain conditional probabilities of the candidate phrases. In scoring each phrase, we first obtain the conditional probability of different sets of trigrams in the phrase. These scores are combined and averaged to generate the final readability score. Suppose $m = w_1...w_n$ is a candidate phrase, $S_{read}(m)$ is thus defined as follows:

$$S_{read}(w_i...w_n) = \frac{1}{K} \cdot \log_2 \prod_{k=q}^{n} P(w_k|w_{k-q+1}...w_{k-1}) \tag{5.4}$$

where q represents the n-gram order of the model used and in our case, $q = 3$. K represents the number of conditional probabilities computed.

Equation 5.4 is essentially the chain rule used to compute the joint probability in terms of conditional probabilities, which is then averaged. Averaging the scores allows us to set cutoffs which helps in pruning non-promising candidates.

5.2.3 Parameter Settings

The optimization formulation involves several parameters to be empirically set. Some of these have to be set in an application-specific way based on tradeoffs between multiple criteria of summarization. For example, σ_{ss} indicates the desired total length of a summary and can be set, e.g., based on the screen size of a mobile device if the summary is to be displayed on such a device. σ_{sim} controls the amount of redundancy allowed in the summary (less redundant with a smaller σ_{sim}). Finally, σ_{rep} and σ_{read} help with the efficiency of the optimization algorithm and also help in ensuring minimum representativeness and readability of phrases; we will later examine the influence of these two parameters in our experiments. With this setup, the remaining challenge is to solve the optimization problem efficiently, which we will discuss in the next section.

[1] http://web-ngram.research.microsoft.com

5.3 Summarization Algorithm

Since we want to *compose* new phrases using words from the original text, the potential solution space for our optimization problem, is huge. In practice, it is infeasible to enumerate all the possible summary candidates and score each one of them. In this section, we propose a greedy algorithm to solve the optimization problem by systematically exploring the solution space with heuristic pruning so that we only touch the most promising candidates.

At a high level, we start with a set of high frequency unigrams from the original text. We then gradually merge them to generate higher order n-grams as long as their *readability* and *representativeness* remain reasonably high. This process of generating candidates stops when an attempt to grow an existing candidate leads to phrases that are low either in readability or representativeness (i.e. does not satisfy σ_{rep} or σ_{read}).

Specifically, the input to our summarization algorithm is a set of sentences from an opinion containing document. For example, all review sentences about the *iPhone*. We denote these sentences as $Z = \{z_i\}_{i=1}^{n}$. The output is a micropinion summary with a set of n-gram phrases $M = \{m_1, ..., m_k\}$, where the number of micropinions is determined based on the constraints of the optimization problem. The summarization algorithm consists of the following three steps:

Step 1. Generation of seed bigrams: The first step takes the original text, Z as input and generates a set of promising bigrams based on combinations of high frequency unigrams.

Step 2. Generation of scored n-grams: The second step takes the seed bigrams as input and further grows them into a set of promising n-grams by concatenating bigrams that share an overlapping word. While generating n-grams, we also compute their representativeness and readability scores S_{rep} and S_{read}, and prune all the cases where any of these scores is below the corresponding threshold. We further check redundancy of the generated candidates, and if two phrases have a similarity higher than the σ_{sim} threshold, we would discard the one with a lower combined score of $S_{rep} + S_{read}$.

Step 3. Generation of micropinion summary: The final step is to sort all the candidate n-grams based on their objective function values (i.e., sum of S_{rep} and S_{read}) and generate a micropinion summary M by gradually adding phrases with the highest scores to our summary until the accumulated summary length reaches the length threshold σ_{ss}.

We will now focus on elaborating steps 1 and 2. Step 3 is straightforward and thus will not be discussed.

5.3.1 Generation of seed bigrams

As redundancies are inherent in opinionated documents, this property can be leveraged to shortlist a set of seed words that can be used to generate higher order n-grams. This way, we avoid having to try every combination of words. Our assumption is that if a word is not frequent in the original text, it is unlikely a good candidate word to be included in any phrase of a micropinion summary (presumably we will have other better candidate words to work with).

Assuming that the input text is a set of sentences, we shortlist a set of high frequency unigrams from the input text, where the unigrams should have counts larger than the *median* count (after ignoring words with frequency of 1). This ensures that the cutoff is adjusted according to the input size. The low threshold serves as an initial reduction in search space; further reduction happens when S_{rep} and S_{read} are computed later.

Each of these high frequency unigrams is then paired with every other unigram to form bigrams. For example, if we have the words 'battery' and 'life', the bigrams generated would be 'battery life' and 'life battery'. We then compute the representativeness score S_{rep} of each bigram and keep only those bigrams whose S_{rep} passes the threshold σ_{rep}. Although the combination of words could be quite random, the S_{rep} function helps in pruning invalid combinations (since it demands co-occurrences of two words in a small window of the original text).

5.3.2 Depth-first search for candidate generation

Using the seed bigrams described in the previous section, we attempt to generate higher order n-grams that will finally serve as candidate micropinions. If there are a large number of seed bigrams (for large input documents), the starting seeds can further be shortlisted by their representativeness scores. For example, using only the top 500 seeds as the starting point. Algorithm 4 outlines the steps involved in candidate micropinion generation.

Algorithm 4 *GenerateCandidates*(p)

```
1: Input: A candidate phrase (bigrams initially)
2: Output: A set of micropinions
3: if (S_read(p) < σ_read || S_rep(p) < σ_rep) then
4:   return
5: end if
6: if ValidCandidate(p, candList) then
7:   candList ← {candList ∪ p}
8: end if
9: joinList ← GetJoinList(seedBigrams)
10: for bigram ∈ joinList do
11:   if NotMirror(p, bigram) then
12:     newPhrase ← Merge(p, bigram)
13:     Score(newPhrase)
14:     GenerateCandidates(newPhrase)
15:   end if
16: end for
```

First, for any incoming phrase, p, a check is done to determine if p is a promising phrase. This is done by checking the $S_{read}(p)$ and $S_{rep}(p)$ scores

with the corresponding thresholds (line 3). If the score constraints are not fulfilled, then p will not be further expanded. This step ensures that we explore only reasonable phrases, thus improving efficiency.

If p fulfills the score constraints, then a check is done to determine if it can become a candidate micropinion (line 6). This is based on the similarity between p and the existing candidate phrases from the pool of candidates. If p is not similar to any of these phrases, then p will automatically be added to this candidate pool. If p is similar to a phrase X in the pool, p will replace X if $S_{rep}(p) + S_{read}(p) > S_{rep}(X) + S_{read}(X)$. In other words, at any given time, we will have a set of non-redundant candidate phrases.

Once the validity of a phrase has been determined, the algorithm proceeds recursively in a depth first fashion in an attempt to expand p to a higher order n-gram (line 9-16). We expand phrases using concepts used for pattern growth as shown in [88]. In particular, we impose a merge requirement between a candidate phrase p and a bigram, B (from the set of seed bigrams) as follows: (1) the ending word in p should overlap with the starting word in B and (2) p should not be a mirror of B. With this, all seed bigrams that satisfy this requirement will be merged with p. Consider a phrase *very long* ***battery*** and a seed bigram ***battery*** *life*. The overlapping word *battery*, connects the two phrases and since one is not a mirror of the other, the two phrases can be merged to form *very long battery life*. Such a pattern growth approach eliminates the need for an exhaustive search and it also avoids exploration of n-grams that are extremely random and unlikely useful. The newly merged phrases are then scored for their readability and representativeness prior to being further expanded (line 13).

5.4 Experimental Setup

5.4.1 Dataset

To evaluate this micropinion generation task, we leverage user generated reviews from CNET[2] for 330 different products. Each product has 5 associated reviews at the minimum. The CNET review structure is such that a user writes a *full length review* about a product followed by a brief summary about the *pros and cons* (PC). The PCs are usually a set of short phrases such as "*bright screen*, *fast download*, etc" where these phrases tend to summarize the full reviews and hence are quite redundant. In evaluating our summarization task, we ignore the PCs and use only the *full reviews* for a product to make the summarization task more realistic (i.e we eliminate obvious redundancies). Thus, for a given product X, we generate a *review document*, r_x, where r_x holds all sentences from the full reviews related to X. On average, there were 500 sentences per review document. Out of the 330 review documents generated, we use only

[2]http://www.cnet.com

230 documents for evaluation as 100 were withheld to train one of the baseline approaches.

While the PCs seem ideal as the gold standard summary, in some cases the PCs are just an enumeration of features that the user likes or dislikes and contain no specific opinions. For example, "**Pros**: *battery*, *sound*; **Cons**: *hard disk*, *screen*". Since we need a set of opinion phrases that are more complete such as "*improved battery life*; *crystal clear sound*", we leverage these PCs to help human summarizers compose meaningful summaries. Specifically, for each product X, we find the top 10 high frequency phrases from the PCs which are then presented as hints to the human summarizers. Since we have a large number of review documents to summarize, such hints help the human summarizer with topic coverage, thus reducing bias in the summaries. Two human summarizers, were asked to read the reviews of each product presented to them and then compose a set of phrases summarizing key opinions on the product. Out of the 330 products, 165 were assigned to one summarizer and the remaining to the other. With this, for each r_x, we obtained a corresponding human summary, h_x.

5.4.2 Quantitative Evaluation

In demonstrating the effectiveness of our approach, we need to quantify to what extent our summaries are representative of key opinions and how readable these summaries are so that they can be understood by human readers. One way of assessing the quality of summaries is to measure how well system summaries resemble human composed summaries. Keyphrase extraction tasks are typically evaluated based on the number of overlapping keyphrases between system generated keyphrases and the gold standard ones. This requires exact matches which is unlikely, as there could be many ways to describe one opinion and subtle variations may result in an unfair 'no match'.

We thus chose to use ROUGE [80], an evaluation method based on n-gram overlap statistics found to be highly correlated with human evaluations. ROUGE was also the standard measure used in the DUC 2004 summarization task[3] on generating very short summaries such as headline summaries ($<$ 10 words). ROUGE is ideal for our task as it does not demand exact matches but it can measure both representativeness and readability of summaries. As an example, ROUGE-1 measures the overlap of unigrams between system summaries and human summaries, thus measuring **representativeness**. Higher order ROUGE-N ($N > 1$) captures the match of subsequences, which measures the fluency or **readability** of summaries. In our experiments, we primarily use ROUGE-1, ROUGE-2 (bigram overlap) and ROUGE-SU4 (skip-bigram with maximum gap length of 4).

[3] http://duc.nist.gov/pubs.html#2004

5.4.3 Qualitative Evaluation

In addition to the automatic ROUGE evaluation, we also performed a manual evaluation to assess the potential utility of the micropinion summaries to real users. Specifically, two assessors were asked to read the micropinion summaries presented to them (using the original reviews as reference) and then fill out a questionnaire assessing the summary on several aspects related to its effectiveness. Note that these assessors are different from those that composed the gold-standard summaries. The questionnaire consisted of three key questions rated on a scale from 1 (Very Poor) to 5 (Very Good). The questions are as follows:

Grammaticality: Are the phrases in the summary readable with no obvious errors? Score (1) - None of the phrases are readable or comprehensible; Score (5) - I don't see any issues with the phrases in the summary.

Non-redundancy: Are the phrases in the summary unique with unnecessary repetitions such as repeated facts or repeated use of noun phrases? Score (1) - All the phrases mean the same thing; Score (5) - The phrases are very unique, summarizing very different topics or issues.

Informativeness: Do the phrases convey important information regarding the product? This can be positive/negative opinions about the product or some critical facts about the product. Score (1) - None of the phrases contain useful or accurate information about the product. Score (5) - All the phrases contain accurate opinions or critical information about the product.

Note that the first two aspects, *grammaticality* and *non-redundancy* are linguistic questions used at the 2005 Document Understanding Conference (DUC) [89]. The last aspect, *informativeness*, has been used in other summarization tasks [90] and is key in measuring how much users would learn from the summaries.

For this evaluation, we used the micropinion summaries for 70 different products that were randomly selected from our dataset. The assessors were not informed about which method was used to generate the summaries.

5.4.4 Baselines

To assess how well our approach compares with existing approaches, we use three representative baselines. As our first baseline, we use TF-IDF, an unsupervised language-independent method commonly used for keyphrase extraction tasks. To make the TF-IDF baseline competitive, we limit the n-grams in consideration to those that contain at least one adjective (i.e. favoring opinion containing n-grams). Note that the performance is much worse without this selection step. Then, we used the same redundancy removal technique used in our approach to allow a set of non-redundant phrases to be generated.

For our second baseline, we use KEA[4] [91], a highly cited, state-of-the-art keyphrase extraction method. KEA builds a Naive Bayes model with known keyphrases, and then uses the model to find keyphrases in new documents. The KEA model was trained using the 100 review documents withheld from our dataset (explained in Section 5.4.1). With KEA as a baseline, we would gain insights into the applicability of existing keyphrase extraction approaches for the task of micropinion generation. Note that since KEA uses training data and our method does not, the comparison is, strictly speaking, an unfair comparison.

As our final representative baseline, we use Opinosis [2], an abstractive summarizer designed to generate textual summary of opinions. Opinosis was shown to be more effective in generating concise opinion summaries compared to extractive approaches like MEAD [83]. We turned off the optional *collapse* feature in Opinosis which attempts to merge several short phrases into longer ones to simulate the task of micropinion generation. All other settings were set at their default values.

In addition to these baselines, we also performed a run to examine the benefit of our strategy of composing potentially new phrases as opposed to relying solely on phrases that occurred in the original text. For this run (WebNgram$_{seen}$), we force our search algorithm to return n-grams that have occurred at least once in the reviews. To give a fair comparison, all the n-grams in each of our baseline are 2-5 words long. Our approach is referred to as *WebNgram* in all our experiments.

5.5 Results

By default, the efficiency parameters in our approach are set as follows: minimum readability score, $\sigma_{read} = -2$ (log of probabilities); minimum representativeness score, $\sigma_{rep} = 4$. As will be shown later, the performance of summarization is not very sensitive to the settings of these parameters. The contextual window size is set to $C = 3$, which is the optimal setting. The user adjustable parameter for redundancy control (using Jaccard) is set to $\sigma_{sim} = 0.40$, for reasonable diversity in phrases.

Comparison of summarization strategies. First, we assess the effectiveness of our approach (WebNgram) in comparison with other representative approaches (KEA, Opinosis, TF-IDF). The performance comparison is shown in Table 5.2 for different σ_{ss} settings. Since the summary size is constrained, we are primarily interested in the gain in recall as the precision is proportional to recall when summary length is fixed. First, based on Table 5.2, we see that overall, WebNgram has the highest ROUGE-1 and ROUGE-2 scores, outperforming all baseline methods. The statistical significance of improvements (based on Wilcoxon test) is indicated in Table 5.2. As the summary size increases, Web-

[4]http://www.nzdl.org/Kea/

ROUGE-1							ROUGE-2						
σ_{ss}	**5**	**10**	**15**	**20**	**25**	**30**	σ_{ss}	**5**	**10**	**15**	**20**	**25**	**30**
Tfidf	0.032	0.062	0.085	0.104	0.122	0.137	Tfidf	0.006	0.010	0.014	0.016	0.018	0.020
KEA	**0.046**	0.079	0.104	0.122	0.142	0.164	KEA	0.010	0.021	0.026	0.029	0.033	0.038
Opinosis	0.034	0.094	0.130	0.157	0.185	0.209	Opinosis	0.012	0.031	0.044	0.049	0.055	0.056
WebNGram$_{seen}$	0.033	0.092	0.121	0.151	0.173	0.190	WebNGram$_{seen}$	0.013	0.035	0.043	0.050	0.056	0.060
WebNGram	0.037	**0.102**	**0.135**	**0.173**	**0.195**	**0.219**	**WebNGram**	**0.017**	**0.047**	**0.055**	**0.069**	**0.074**	**0.080**
% Improvement							% Improvement						
WebNgram over **TfIdf**	+13.25*	+39.90**	+36.61**	+40.00**	+37.32**	+37.77**	WebNgram over **TfIdf**	+65.55**	+77.80**	+74.22**	+77.41**	+75.52**	+75.07**
WebNgram over **KEA**	-25.36	+22.38*	+22.74**	+29.31**	+27.33**	+25.37**	WebNgram over **KEA**	+39.33*	+56.59**	+52.95**	+58.18**	+55.66**	+52.34**
WebNgram over **Opinosis**	+7.48	+8.19	+3.41	+9.17	+5.02	+4.89	WebNgram over **Opinosis**	+27.73	+33.57**	+19.71*	+29.95**	+26.09**	+29.19**

Table 5.2: ROUGE-1 & ROUGE-2 recall scores at different σ_{ss}; *statistically significant with p-value$<$ 0.01, **p-value $<$ 0.001

σ_{ss}	**Web Ngram**	**Opinosis**	**TF-IDF**	**Kea**
5	**1.00**	1.14	1.63	1.90
10	**2.91**	3.00	3.40	4.01
15	**4.53**	4.76	5.00	6.04
20	**5.85**	6.56	6.66	8.01
25	**7.36**	8.43	8.32	9.86
30	**8.74**	10.01	9.93	11.75

Table 5.3: Average # of generated phrases in summary.

Ngram consistently gains both in terms of ROUGE-1 and ROUGE-2. This shows that representative phrases are being included in the summary and these phrases are also readable as shown by the consistent gain in ROUGE-2 (the other baseline methods do not gain as much with ROUGE-2).

Next, we see that the ROUGE-1 scores of Opinosis and WebNgram are quite similar, but the ROUGE-2 scores are very different. The similarity in ROUGE-1 scores indicates that Opinosis is able to capture similar topics and opinions as WebNgram. However, the ROUGE-2 score of Opinosis indicates that the generated phrases are less fluent. Through manual inspection, we found that Opinosis generates many short phrases (< 4 words), generally fragmented and thus less fluent. This explains the lower ROUGE-2 scores. This is further confirmed by the average number of phrases generated by Opinosis which is higher (i.e more fragmented) than that of WebNgram (shown in Table 5.3). One possible reason as to why Opinosis has problems generating longer and more complete phrases is lack of structural redundancies between sentences in the CNET reviews.

Amongst all the approaches, TF-IDF performs the worst as shown in Table 5.2. This is likely due to insufficient redundancies of meaningful n-grams. When there are subtle differences in common expressions, it is often difficult to discover redundant n-grams. This is where our method excels as we do not directly rely on the structure of the input text, but rather expand high frequency seed words into longer and meaningful phrases. Finally, notice that KEA does only slightly better than TF-IDF even though KEA is a supervised approach. While KEA is suitable for characterizing documents, such a supervised approach proves to be insufficient for the task of generating micropinions. It might be that the model needs a more sophisticated set of features for it to generate more meaningful summaries. Note that varying the size of training data had minimal effect on KEA.

Seen N-grams vs. System Generated N-grams. In our candidate generation approach, we form longer n-grams from shorter ones using the procedure described in Section 5.3. One may argue that with sufficiently redundant opinions, searching the restricted space of all *seen n-grams* may be sufficient for generating micropinion summaries and thus such a search procedure may not be necessary. We thus performed a run by forcing only seen n-grams to appear

σ_{ss}	WebNgram +Bias	Web Ngram	Opinosis	TF-IDF	KEA
10	2.80	2.11	2.08	2.13	1.16
20	5.48	4.19	4.56	4.01	2.33
30	7.81	6.10	6.77	6.07	3.39

Table 5.4: Average # of opinion words in summary

as candidate phrases. The results are shown in Table 5.2 under WebNgram$_{seen}$. From this, it is clear that this approach yields lower performance compared to using system generated n-grams (WebNgram), suggesting that our search algorithm actually helps discover useful new phrases. When opinions are not sufficiently redundant, observed n-grams tend to be less representative than our system generated n-grams which are constructed by combining related words and phrases (e.g. *big and clear screen* from *"..the phone has a big screen.."* and *"..the screen was clear.."*). This is one good example of why a more abstractive approach is suitable for generating such concise opinion summaries.

Well-formedness of phrases. Intuitively, a good summary phrase is one that is fairly well-formed and clearly conveys the intended meaning. Thus, a few readable phrases is more desirable than many fragmented phrases. Consider two micropinion summaries, $M = \{very\ clear, screen\ is\}$ and $M' = \{very\ clear\ screen\}$. In this example, it is obvious that M' is a more desirable summary than M. Thus, we now look into the average number of phrases generated for different summary sizes which is shown in Table 5.3. We can see that the WebNgram approach generates the fewest phrases for any given σ_{ss} constraint. This shows that the WebNgram phrases are longer on average (i.e. more well-formed) and also readable as previously shown by the ROUGE-2 scores and further validated by our manual evaluation. KEA seems to favor very short phrases and on average, KEA generates the most number of phrases.

Opinion coverage and effect of summary biasing. An important point to note is that each of the baselines used (KEA, Opinosis and TF-IDF) has some form of opinion biasing built-in; *KEA* with the training examples from human composed summaries, *Opinosis* through selection of phrases with certain POS tags and *TF-IDF* through selection of adjective containing n-grams. In our current model, we do not use any form of biasing to evaluate the effectiveness of our method as is. To estimate the actual coverage of opinions, we count the average number of opinion words in the summary using a general set of adjective words from General Inquirer[5] (1,614 positive and 1,982 negative). The results are reported in Table 5.4. Considering only the base WebNgram model and the other 3 baselines discussed earlier, Opinosis has the highest number of opinion words. However, notice that even without opinion specific refinements, WebNgram has comparable number of opinion words to that of Opinosis and TF-IDF (which have built-in biasing).

[5]http://www.wjh.harvard.edu/ inquirer/

	ROUGE-1	ROUGE-2	ROUGE-SU4
WebNgram	0.219	0.079	0.069
WebNgram+Bias	0.240	0.085	0.088
change	**9.3%**	**7.8%**	**27.9%**

Table 5.5: Recall scores with the use of biasing ($\sigma_{ss} = 30$).

	Recall			**F-score**		
Rouge	**R-1**	**R-2**	**R-SU4**	**R-1**	**R-2**	**R-SU4**
$+S_{rep}$-S_{read}	0.192	0.057	0.056	0.120	0.035	0.019
-S_{rep}+S_{read}	0.199	0.061	0.062	0.127	0.035	0.020
$+S_{rep}$+S_{read}	**0.219**	**0.080**	**0.073**	**0.140**	**0.046**	**0.024**

Table 5.6: Performance of scoring components ($\sigma_{ss} = 30$).

To gain insights into how opinion specific refinements can help our summaries, we explore a simple heuristics biasing. Specifically, in our final candidate selection step we only considered phrases that contain at least one adjective. This run is called **WebNgram+Bias**. From Table 5.4 and Table 5.5, it is clear that the addition of opinion biasing improves agreement with human summaries and the average number of opinion words in the summaries is also much higher than the base model. This suggests that our model can be further refined to generate task specific summaries.

Effectiveness of two-part scoring. So far, it has been assumed that both components of our scoring function, S_{rep} and S_{read} are required to generate reasonable summaries. To test this hypotheses, we generate summaries with different scoring components turned off at a given time. The ROUGE scores are reported in Table 5.6. Overall, the performance is lowest when S_{read} is not used (row 1). Without S_{read} it is likely that ungrammatical phrases with high representativeness scores appear as good candidates, thus resulting in poor performance. While S_{read} is important, by itself, it does not perform as well (row 2) as when used in conjunction with S_{rep} (row 3). This is because S_{read} favors highly readable phrases but these phrases may not be representative of the underlying opinions. Thus, it is clear that S_{read} and S_{rep} work in synergy to generate meaningful summaries.

Stability of parameters. It is critical to understand the effect of non-user

	TF-IDF		**WebNgram**		**Human**	
Question	**Avg.**	**Dev.**	**Avg.**	**Dev.**	**Avg.**	**Dev.**
Grammaticality	2.01	0.63	4.16	0.86	4.71	0.65
Non-redundancy	2.34	0.87	3.92	0.69	4.53	0.71
Informativeness	1.67	0.71	3.22	0.76	3.61	1.01
Overall	**2.00**	**0.74**	**3.77**	**0.77**	**4.29**	**0.79**

Table 5.7: Results of manual evaluation with $\sigma_{ss} = 25$. Score 1 (Very Poor) to 5 (Very Good).

dependent parameters in our model (namely σ_{read}, σ_{rep} and C), so that these parameters can be set correctly for a new dataset. The primary function of σ_{read} and σ_{rep} is to prune non-promising candidates, thus improving efficiency and as a result also ensuring minimum readability and representativeness. Without these two parameters, we will still arrive at a solution, but the time to convergence would be much longer and the results could be skewed (e.g. very representative but not readable). Figure 5.1 (a) and (b) show how different settings of σ_{read} and σ_{rep} affect the overall performance. The values in Figure 5.1 (a) are averaged across $\sigma_{rep} \in [1, 5]$ and $C \in [2, 6]$; The values in Figure 5.1 (b) are averaged across $\sigma_{read} \in [-4, -1]$ and $C \in [2, 6]$. Notice that these two parameters are actually stable across different values and do not affect performance except in extreme conditions (thresholds that are too high). When $\sigma_{read} \leq -1$, phrases are expected to have high readability scores from the start and this requirement is too restrictive in finding good candidates. Similarly, when $\sigma_{rep} \geq 5$, the candidates are expected to have extremely high representativeness scores at every point, and again restricting discovery of good candidates. The fact that σ_{read} and σ_{rep} do not affect performance (except when the thresholds are too high) suggests that the objective function already ensures phrases to be both representative and readable. It is thus safe to set these values to be small enough (between -2 and -4 for σ_{read}; between 1 and 4 for σ_{rep}) to ensure reasonable efficiency and meaningful summaries. Note that the low 'min' curves as seen in Figure 5.1 (a), (b) and (c) is caused by the extreme values, $\sigma_{rep} = 5$ and $\sigma_{read} = -1$.

The third parameter, C is a window size used to compute the representativeness score of a phrase. The requirement is that two words in a *candidate phrase* should occur within a context window of size C in the original text (see Section 5.2.1). Figure 5.1 (c) shows performance at different C values (averaged across $\sigma_{rep} \in [1, 5]$ and $\sigma_{read} \in [-4, -1]$). On average, the best performance is achieved when $C = 3$, which is quite reasonable. Intuitively, when C is large, certain important words that are spread out can now be seen in context (e.g. *the Nokia phone that I bought was cheap*). At the same time, wrong pairs of words may also be considered related and this is evident with lower performance when $C > 3$.

To further show that the suggested parameter settings would hold true on new datasets, we obtained the optimal parameter values tuned on the 100 review documents withheld to train KEA. The values are: $C = 3$, $\sigma_{read} = -4$ and $\sigma_{rep} = 4$. Note that all these values comply with the suggested settings. In fact, the σ_{rep} and C values are the same as our default settings. The only difference is that $\sigma_{read} = -4$. This new σ_{read} value on our evaluation dataset, did not show any significant difference in terms of performance (ROUGE-1 gain:-0.004, ROUGE-2 gain: +0.002). The only difference was in efficiency, which in this case was slower due to the more relaxed setting. It is thus clear, that the efficiency parameters have little effect on performance of summarization as long

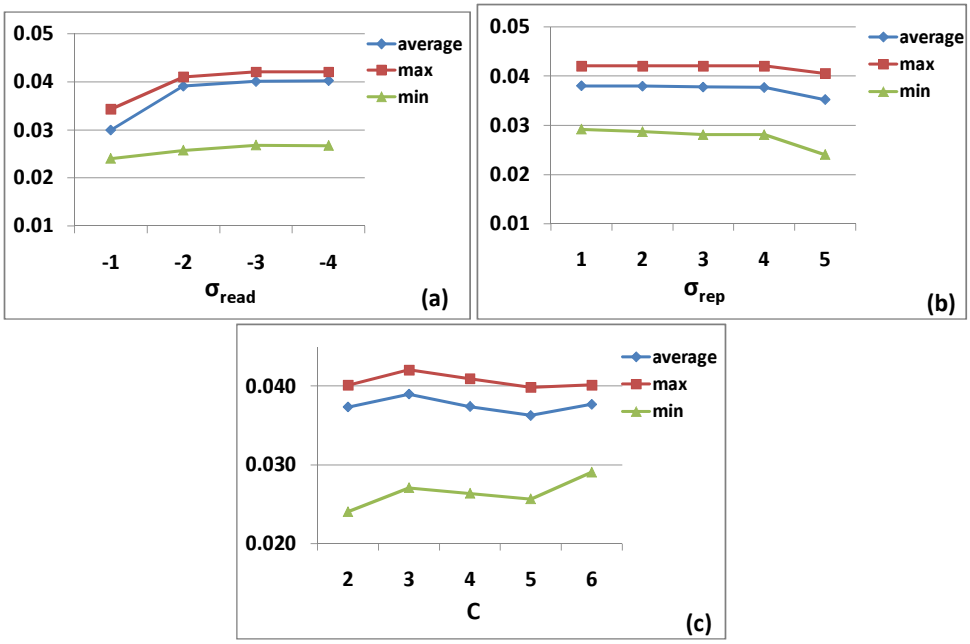

Figure 5.1: ROUGE-2 with varying σ_{read}, σ_{rep} and C. Labeled as (a), (b) and (c) respectively.

as the values are not too restrictive.

Manual evaluation. To assess potential utility of the micropinion summaries to real users, a subset of the summaries were manually evaluated using the procedure described in Section 5.4.3. The average *grammaticality*, *non-redundancy* and *informativeness* scores (along with respective standard deviation scores) for three methods are reported in Table 5.7. The results on human summaries serves as an upper bound for comparison. A score *below* 3 is considered 'poor' and a score *above* 3 is considered 'good'.

Earlier, we showed that TF-IDF had the lowest ROUGE scores amongst all the approaches, indicating that the summaries may not be very useful (see Table 5.2). The scores assigned by the human assessors on the TF-IDF summaries agree with this conclusion. On average, TF-IDF summaries received poor scores (below 3) on all three dimensions compared to WebNgram and human summaries. WebNgram's average scores are above 3 on all dimensions and are quite close to human scores.

In terms of grammaticality, WebNgram summaries received an average score of 4.16, which means that the WebNgram phrases are mostly grammatically sound with a few exceptions. This number actually correlates well with our ROUGE-2 scores discussed earlier. Next, the average non-redundancy score of 3.92 tells us that the WebNgram phrases are fairly unique with a few cases of overlapping facts or opinions. Although our σ_{sim} setting requests a diverse set of phrases, some of the redundancies were caused by the different ways in which the same information can be conveyed. For example, the phrases *Excel-*

lent sound quality and *Great audio* may seem different but actually mean the same thing. Such differences can be resolved with the use of opinion or phrase normalization or clustering of similar words and concepts prior to summarization. While the average informativeness scores of WebNgram and humans are quite close, notice that these scores are just slightly above 3. This suggests that the informativeness aspect is rather subjective and a score above 3 is actually quite encouraging. The WebNgram summaries that had informativeness scores between 3-4 mostly had meaningful and representative phrases along with some false positives that did not have any real information (e.g *I bought this for Christmas*). The informativeness aspect can be improved in different ways, one of which is through a much stricter selection of phrases. This is something we would like to study in detail in the future.

5.6 Conclusion

In this chapter, we proposed an unsupervised summarization approach that leverages Web N-grams to generate ultra-concise summaries of opinions. We frame the problem as an optimization problem with an objective function capturing representativeness and readability and constraints that ensures compactness of a generated summary. We propose a heuristic search algorithm to solve the optimization problem efficiently. Our evaluation using a set of user reviews shows that our summaries can convey essential information with minimal word usage and is more effective than other competing methods.

The proposed approach is practical, lightweight and general as it does not rely on any linguistic annotations (e.g. POS tagging or syntactic parsing) and is not designed or optimized for a specific domain. It only uses the existing text and a web scale n-gram model to generate meaningful summaries. Thus, our approach can be used in a variety of domains (e.g. blogs, twitter, etc) and can be used to generate summaries in other languages.

In the future, we would like to explore further refinements such as including an opinion component into the model and investigate the use of a domain specific n-gram model to improve the quality of summaries.

[5]The work done in this chapter has been published in (Ganesan et al. 2012) [3]

6 OPINOFETCH: AN UNSUPERVISED APPROACH TO COLLECTING OPINIONS ON ARBITRARY ENTITIES

To support the search and analysis tasks of the ODSS platform, opinionated content is imperative. While previous works have been focused on mining and summarizing online opinions, there is limited work on exploring the automatic collection of opinion containing documents. In this chapter, we propose a *lightweight* and *unsupervised* framework for collecting opinions namely reviews for arbitrary entities. We show how we leverage existing web search engines and use a novel information network called the FetchGraph to efficiently obtain review pages for entities of interest.

6.1 Introduction

With the surge of Big Data capabilities, the abundance of opinions expressed by experts and ordinary users on the Web is now becoming vital to a wide range of applications. For example, market research tools may use opinions to determine if a product is worth developing or marketing. Business intelligence applications may use online opinions to understand what users like or dislike about a recently launched product. Another important usage is with online shopping sites where these sites can utilize existing opinions on the web to help users make purchase decisions. While there is a clear need for a large number of opinions about a topic or entity (e.g. person, product or business), access to such content is very limited as opinions are often scattered around the web and the web is inherently dynamic. Consider the task of collecting opinions about the *iPhone 5*; one can find related opinions on well known e-commerce sites such as Amazon.com and BestBuy.com within popular review sites such as CNET.com and Epinions.com and on less mainstream sites such as Techradar.com and personal blog sites. It is clear that there is no central repository to obtain all the opinions about an entity or topic. Moreover, the set of sites that contain reviews about one entity may not contain reviews about another similar entity. This makes the task of developing computational techniques for *collecting online opinions* at a large scale a new and interesting research challenge with a pressing need. '

Online opinions are typically present in user generated reviews, personal and professional blog sites, forums, tweets, Facebook status updates and more. In this chapter, we focus primarily on *user reviews* as reviews alone make up a big portion of online opinions. For example, user generated reviews can be found in most e-commerce applications (e.g. Amazon.com, Walmart.com and

eBay.com), specialized user review websites (e.g. Yelp.com), vertical search engines (e.g. Hotels.com) and online directories (e.g. Yellowpages.com). Thus, a comprehensive set of user reviews alone would be highly valuable to many opinion dependent applications.

Intuitively, the simplest way to collect online reviews, is simply to crawl the entire web and then identify opinion containing pages on entities of interest. While in theory this method seems feasible, in practice it is actually intractable as (1) visiting and downloading all pages on the web would be very time consuming and places high demands on network and storage resources and (2) it would become very expensive to perform relevance classification on each page from the web. Thus, a more reasonable method to solving this problem would be to use a focused crawling strategy.

Existing focused crawlers [92] are primarily designed to crawl all documents pertaining to a topic such as *databases* and *cycling*. Thus, the *type* of page or document (e.g. review page, news page, blog article, etc.) is not as important as the content of the page itself. In contrast, our goal is more specific as we are interested in a particular type of page (i.e. review pages) and are more concerned about the relevance of a page to entities of interest. Moreover, most of the existing focused crawlers are either supervised relying on large amounts of training data or rely heavily on external help, making these crawlers rather restrictive.

With this, we propose OpinoFetch a practical unsupervised framework for collecting online reviews for arbitrary entities. Our key idea is to first obtain am initial set of candidate review pages for a given entity using an existing Web search engine. We then expand this list by exploring links around the neighborhood of the search results. All these entity specific candidate pages and its supporting components are modeled in a *heterogenous graph data structure* called the *FetchGraph* which helps with efficient lookup of important statistics and helps answer important application questions. The FetchGraph is pruned in the end based on relevance scores (computed using the FetchGraph itself) leaving behind a set of relevant, entity specific review pages.

In contrast to all previous work, our approach is unsupervised and assumes *no domain knowledge* and thus can work across any domain (e.g. hotels, cars, doctors, etc.). This is to provide a more general and practical approach to finding online opinions. Evaluation results in three different domains (i.e. attractions, hotels and electronics) show that our approach to finding entity specific review pages far exceeds Google search and we are able to find such review pages with reasonable *efficiency* and *accuracy*. The dataset and demo of this system will be available at `http://timan.cs.uiuc.edu/downloads.html`. In summary, the contributions of this work are as follows:

1. We propose a lightweight, unsupervised framework for discovering review pages of arbitrary entities leveraging existing Web search engines.
2. We introduce FetchGraph, a novel information network for efficient data

collection

3. We create the first test set for evaluating this review page collection problem.
4. We run experiments to test the proposed methods in three domains and show that our approach is capable of finding entity specific review pages with reasonable accuracy and efficiency.

6.2 A General Unsupervised Framework for Review Page Collection

Given a set of entities, $E = \{e_1, ..., e_n\}$, from a domain, $\mathcal{D}$, the goal of our task is to find a complete set of online review pages denoted as $R_i = \{r_{i1}, ..., r_{in}\}$, where R_i contains a set of relevant review pages about e_i. Entities refer to objects on which reviews are expressed. This can be *businesses* such as hotels and restaurants, *people* such as politicians and doctors and *products* such as smart phones and tablet computers. The domain is a broad category and the granularity of the domain (e.g. smart phones vs. all mobile devices) is decided by the client application. Review pages are pages containing user composed opinions about a target entity. This includes *user generated reviews* (e.g. as found in Amazon.com) and *expert reviews* (e.g. as found in CNET.com).

Our problem set-up allows for flexible adjustment of entities of interest according to the application needs and with this, the proposed framework would mainly focus on finding opinions about these target entities. This is to support a typical application scenario where a large number of reviews is needed for a specific set of entities (e.g. all hotels in the United States).

There are several challenges to solving this special task of review page collection for arbitrary entities. One important problem is the task of matching entities of interest with crawled pages. Typically, to do this accurately, we would need large amounts of training data. However, this is not practical as it requires training data for each target entity and the list of entities can vary and can get large depending on the application.

Another problem is that there is no easy method for obtaining starting points for the crawl. Unlike commonly crawled domains such as news and blog domains where published RSS feeds are easily obtainable for use as starting points, obtaining an initial set of seed pages for review page collection is not as easy. If we used links from one specific review site as seeds, aside from being able to crawl all reviews from that site, there is no guarantee that this would take the crawler to other review sites. We also cannot rely only on a fixed set of sites to obtain entity specific reviews because review containing sites are often incomplete. If one site has reviews for entity A this does not guarantee reviews for entity B even if they are closely related (e.g. reviews on iPhone 4s and iPhone 5).

To address these challenges, we propose a general *unsupervised* review page collection framework capable of collecting review pages for arbitrary entities, by leveraging an existing Web search engine. The framework consists of the following steps:

Step 1. Obtain initial Candidate Review Pages (CRP): Given an entity e_k, we obtain an initial set of Candidate Review Pages (CRP) using a general Web search engine. This is used in conjunction with an *entity query*, a special query requesting review pages related to the target entity. We use σ_{search} to control the number of search results.

Step 2. Expand CRP list: We then expand the CRP list by exploring links around the neighborhood of each initial CRP, building a larger and more complete list of potential review pages. σ_{depth} controls how far into the neighborhood the exploration should happen. Intuitively, the more pages we explore, the more chances of recovering relevant review pages.

Step 3. Collect Entity Related Review Pages Next, all the pages in the expanded CRP list along with the initial CRPs are scored based on (1) *entity relevance* and (2) *review page relevance*. Both these scores are used to eliminate irrelevant pages from the CRP list retaining a set of relevant review pages for each $e_k \in E_{\mathcal{D}}$. We will now expand on the details of each of these steps.

6.2.1 Obtaining Initial Candidate Review Pages

Since most content on the web are indexed by modern web search engines such as Google and Bing, review pages of all sorts of entities would also be part of this index. Also, modern search engines are very effective at finding pages about entities. Capitalizing on this fact, we can leverage web search engines to find the initial CRPs. We can do this by using information about the entity as the query (e.g. entity name + address or entity name + brand) along with biasing keywords such as *reviews* or *product reviews*. This is called the *entity query*. For example, the entity query for reviews of a hotel in Los Angeles would be similar to: *Vagabond Inn USC Los Angeles reviews*. The hope is that the results of such a query would consist of pointers to review pages about the target entity or have relevant review pages somewhere in the neighborhood.

We can thus treat the results of the search engine as an entry point to collecting a more complete and accurate set of pointers to entity related review pages. There are several advantages of doing this. First, search engines can help discover *entity specific review sites* as different sites would hold reviews for different subsets of entities even within the same domain. As an example, when we compare the search results for the query *iPhone 3g reviews* and *iPhone 4 reviews* on Google, we will find that there are sites that contain reviews for one of these products but not the other and vice versa. Next, since web search engines are effective in finding pages about entities, the task of matching reviews

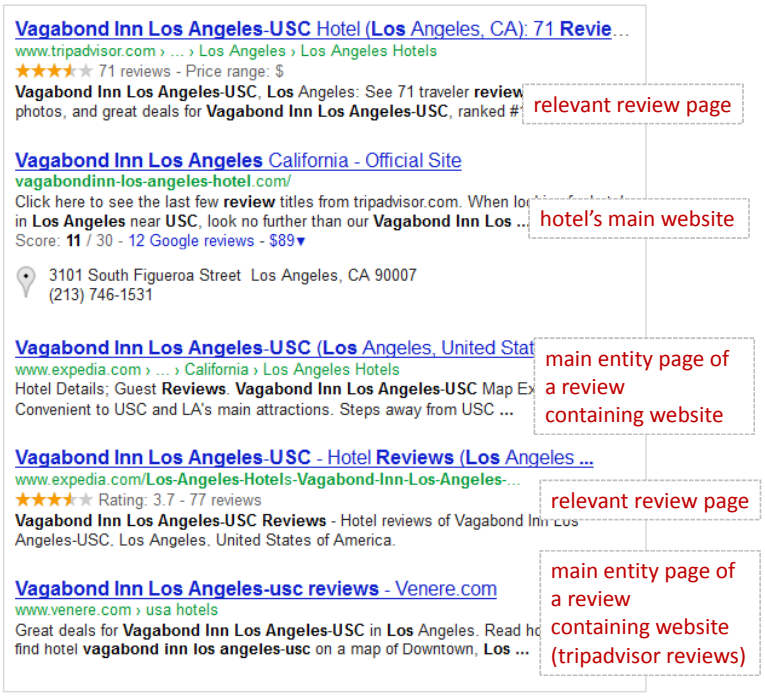

Figure 6.1: Example search results from Google for the query *Vagabond Inn USC Los Angeles reviews*. Only two pointers are to actual review pages.

to an entity is already partially done by the search engine.

Since we use different search results for different entities, our task is more of a *hyper-focused data collection task*, as the search results are already quite 'close' to the relevant content. This is unlike traditional topical crawling where search results are used as a very general starting point of a long crawl. One may argue that the results of search engines alone are sufficient in finding entity related reviews. While some of the search engine returned pointers are valid links to review pages, there are many pointers that are not. In Figure 6.1 we show snapshot of results from Google for the query *Vagabond Inn USC Los Angeles reviews*. From this, we can see that only two out of five items point to valid review pages. The second item is pointer to the hotel's homepage. The third and fifth items point to sites that contain reviews about the hotel, but the link is to the main entity page rather than the actual review page. To address these problems, we propose to expand the CRP list.

6.2.2 Expanding CRP List

While it is possible to expand all URLs in a page for further exploration, this is a waste of resources as some URLs are known to be completely irrelevant (e.g. 'contact us' page and 'help' page). We propose a simple and effective URL prioritization strategy that attempts to bias the crawl path towards entity related pages. To achieve this, we measure the average cosine distance between

(1) terms within the *anchor text* and the *entity query* and (2) *URL tokens* (delimited using special characters) and the *entity query*. Thus, the more the anchor text or URL resemble the entity query, the more likely that this page is relevant to the target entity. In each page, we can use the top N scoring links for further exploration until the chosen depth (σ_{depth}) is reached. N here can be a constant or a percentage of links. While more sophisticated strategies are possible, optimizing this step is not the focus of our work, we thus leave this as a future work.

6.2.3 Collecting Entity Related Review Pages

During the course of expanding the CRP list, we would naturally encounter many irrelevant pages to get to relevant ones. We thus need a method to eliminate the irrelevant pages. A page can thus be scored in terms of (a) *review page relevance* denoted by $S_{rev}(p_i)$ and (b) *entity relevance* denoted by $S_{ent}(p_i, e_k)$ where p_i is a page in the crawled set and e_k is the entity for which p_i appeared as a CRP. With this, to determine if a page p_i is relevant to an entity e_k, we need to check if $S_{rev}(p_i) > \sigma_{rev}$ and $S_{ent}(p_i, e_k) > \sigma_{ent}$ where σ_{ent} and σ_{rev} are two thresholds that range from $[0 - 1]$. While it may be possible to combine scoring of (a) and (b) into a single scoring approach, separating the scores provides more control on how each aspect is scored. Moreover, we may choose to give higher priority to one aspect than the other.

The proposed framework does not put any restriction on how to define the $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$ scores. Below we present a reasonable instantiation that we evaluate in our experiments. Since we would like to do everything in an unsupervised way, these scoring functions can only be defined in a heuristic way.

6.2.3.1 Review Page Relevance

Determining if a page is made up of reviews can be done by using a lexicon consisting of sentiment words or words commonly found in review pages to score a page. Specifically, for a given page p_i, we can heuristically compute a review page relevance score as follows:

$$S_{rev}(p_i) = \frac{\sum_{t \in V} log_2\{c(t, p_i)\} * wt(t)}{normalizer}$$

where t is a term from the defined vocabulary, V and $c(t, p_i)$ is the frequency of t in p_i and $wt(t)$ is the importance weighting given to term t. If we used only raw term frequencies, this may artificially boost the $S_{rev}(p_i)$ score even if only one of the terms in V was matched. Thus, we use log to scale down the frequencies. $wt(t)$ is important because it tells the scoring function which terms are better indicators of a review page. For example, terms like *review* and *rating*

are better indicators than terms like *date*. Intuitively, a review page would have many of the terms in V with reasonably high term frequencies.

Since, we perform a sum of weights over all terms in V, the $S_{rev}(p_i)$ value can become quite large for highly dense review pages and this would make it difficult to set thresholds. To overcome this problem the $S_{rev}(p_i)$ is normalized with a *normalizer*.

Most review pages have similarities to a certain degree both in terms of structure and usage of words. Based on this property, we construct a review vocabulary consisting of the most common *review page indicating* words where the weight contribution of each word depends on its popularity across different web sources. Specifically, we manually obtained URLs to 50 different review pages from 25 distinct web sources covering a wide range of domains such as electronics, software tools, doctors, hotels and others. We discarded all common stop words and rank each remaining term t in the combined vocabulary of the 50 review pages, denoted as R as follows:

$$Rank(t, R) = SiteFreq(t, R) * AvgTF(t, R)$$

$$AvgTF(t, R) = \frac{1}{n} \sum_{i}^{n} \frac{c(t, r_i)}{MaxTF_{r_i}}$$

where $SiteFreq(t, R)$ corresponds to how many web sources the term t occurred in and $AvgTF(t, R)$ is the sum of normalized term frequencies of a term t across all review documents that contain t. n is the total number of review documents containing t. With this, the more popular a term is across sites and the higher its average term frequency, the higher the rank this term would have. This helps review page specific terms to emerge rather than domain specific words. Example of terms from our review vocabulary are as follows: *review*, *helpful*, *services*, *rating*, *thank*, *recommend*. The top 100 terms and their corresponding weights that is the $AvgTF(t, R)$ are included in the final review vocabulary.

To normalize $S_{rev}(p_i)$ we define the following normalizers:

SiteMax Normalizer: If a particular site is densely populated with reviews, then many of the review pages within this site would have high review relevance scores. Similarly, if a site contains limited reviews, then its likely that many of the review pages would have low review relevance scores. Thus, if we normalize the raw $S_{rev}(p_i)$ using the maximum $S_{rev}(p_i)$ score from the site that it originates from, the score of a true review page would always be high regardless of density of the review site.

EntityMax Normalizer: In many cases if an entity is highly popular, the user reviews on that entity would accordingly be abundant. Similarly, if an entity is not so popular, then the amount of reviews on that entity would also be limited. This is usually true across websites. For example, reviews on the *iPhone* is abundant regardless of the review containing site. By using the maximum review page relevance score of all pages related to a particular entity as a normalizer, a review page of an unpopular entity would still receive a high score because the

maximum $S_{rev}(p_i)$ score would not be very high.

GlobalMax Normalizer: To help with cases where the EntityMax or SiteMax normalizers are unreliable, we can use the maximum $S_{rev}(p_i)$ score based on all pages in the FetchGraph. When there is a limited number of collected pages for a given entity (e.g. uncommon entities such as a particular doctor) the EntityMax normalizer could become unreliable. Also, when there is only one page collected from a particular site (e.g. a blog page), the $S_{rev}(p_i)$ score using the SiteMax normalizer would be artificially high as it is normalized against itself. To help with both these cases we incorporate the GlobalMax normalizer which can act as a 'backup' normalizer.

6.2.3.2 Entity Relevance Scoring

Even though a page p_i may be a review page, it may not be relevant to an entity e_k. To eliminate such irrelevant pages, we need an *entity relevance* score measuring how relevant p_i is to e_k. We observed that most review pages have URLs that contain the name of the entity commented on. We thus define the relevance score as the similarity between the entity query and the URL of the candidate review page.

To measure similarity we use Jaccard which is defined as the size of the intersection divided by the size of the union of sample sets. The Jaccard measure is ideal because we are measuring similarity of tokens between two pieces short texts. Also, since the Jaccard similarity score ranges from $[0-1]$ this makes it easy to set the σ_{ent} cutoff. With this, the entity relevance of a page p_i with entity e_k, $S_{ent}(p_i, e_k)$ is defined as follows:

$$S_{ent}(p_i, e_k) = \frac{T_{URL} \cap T_{EQ}}{T_{URL} \cup T_{EQ}}$$

where T_{URL} is a set containing all the URL tokens (tokenized by special characters) and T_{EQ} is a set containing all the terms within the entity query.

6.3 Implementation Challenges

In the previous section, we outlined a very general approach with a reasonable instantiation to finding entity related review pages in an unsupervised manner. While the proposed ideas can be solved in a multitude of different ways, the question now is, how can we make this framework useful to client applications? We define two key aspects to ensure usability in practice: (1) Efficiency and (2) Access to rich information.

1. Efficiency: Our goal is to enable collection of review pages for a large number of entities. Thus, the data collection task should be efficient enough to terminate in a reasonable amount of time with reasonable accuracy without

overusing resources. Efficiency is usually affected by the inability to obtain required information quickly and this usually stems from lack of proper structure. Thus, it is important to manage crawl information properly so that we have quick access to various statistics and related information.

2. Access to rich information: A client application will benefit greatly if the framework can provide access to information beyond the basic crawled pages. In standard crawling tasks, once the crawler is done crawling, the collected pages and sometimes the WebGraph are the only information available to the client application. Consider a query such as *get review pages from top 10 popular sites for entity X*. Such a query is difficult to answer with a database or a WebGraph as none of these can model complex relationships.

We now present a novel data structure called the FetchGraph which can address the challenges mentioned in this Section.

6.4 FetchGraph: Novel Information Network for Review Crawling

We propose a rich data structure called the FetchGraph, which is a *directed heterogenous graph* or an *information network* that can model arbitrary relationships between different components of a data collection problem. Figure 6.2 shows an example of a FetchGraph for the problem of collecting review pages for a single entity, *iPhone 5*. Note that this is a partially complete graph used as an example. The first thing we would notice from Figure 6.2 is that the FetchGraph provides an intuitive view of the entire data collection problem as it models the complex relationships between the different components (e.g. entities, pages, terms, sites). Each component is represented by a simple node in the graph and relationships are modeled using edges. This rich information network has several interesting properties that helps provide balance between efficiency and accuracy and helps with application related querying.

One simple data structure to access various statistics. The FetchGraph provides fast access to all sorts of statistics. For example, based on Figure 6.2, it is obvious that in order to obtain the term frequency of a word in a given page, we only need to lookup the weight of the edge connecting the relevant *content node* and *term node*. We would not have to repeatedly compute the term frequencies nor do we need to maintain a separate data structure in order to track page related terms.

Provides access to global statistics. Since the FetchGraph keeps track of different components over the course of collecting review pages for many entities, we have access to global statistics. For example, for normalization of the $S_{rev}(p_i)$ scores, we can now use global information such as the global maximum term frequency instead of relying on just local information. Using global information to compute key statistics could lead to higher accuracy.

Models complex relationships which can be persisted. As the FetchGraph is able to model complex relationships between different components in a data collection problem, the client application can leverage this information network to answer all sorts of interesting questions. This is because once the graph has been constructed it

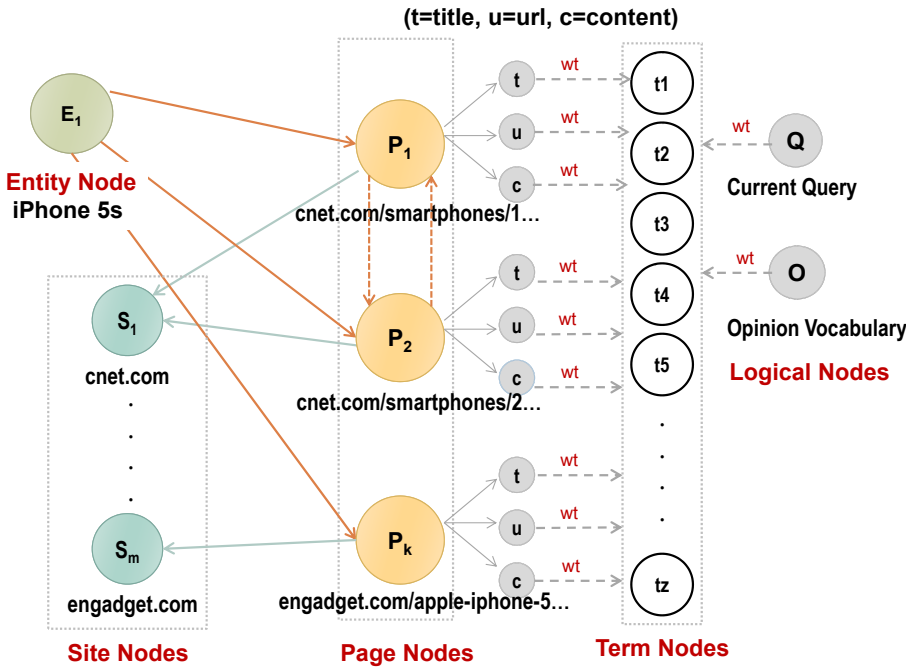

Figure 6.2: Example FetchGraph for a single entity - *iPhone 5*. Dashed edges indicate compound edges. Gray nodes indicate logical (conceptual) nodes. Term nodes represent terms in the combined vocabulary of the data collection task (e.g. page content and URL tokens).

can be persisted and accessed at a later time for use by the client application. For example, for a given entity, the client application may only want to consider the top N relevant review pages from each site. Using the FetchGraph we can retrieve such pages by first accessing all pages related to the given entity. Then, we can find the site that each page belongs to and rank pages from each site using the $S_{rev}(p_i)$ scores.

6.4.1 Components of the FetchGraph

Intuitively, in a web (data) page collection problem, the goal is to collect a set of web pages. Each of these pages originate from a specific site. Since this is a focussed data collection problem, each page is related to a specific topic or entity. The page and its URL, search queries, etc. are at the very core made up of a set of terms. Based on this, we define 5 core node types of the FetchGraph: (1) entity nodes, (2) page nodes, (3) term nodes (4) site nodes and (5) logical nodes and 2 application specific logical nodes: (6) OpinVocab node and (7) query node.

In formal terms, we denote entity nodes as $E_{\mathcal{D}} = \{e_i\}_{i=1}^{n}$ where $\mathcal{D}$ represents a domain type, page nodes as $P = \{p_i\}_{i=1}^{k}$, term nodes as $T = \{t_i\}_{i=1}^{z}$, site nodes as $S = \{s_i\}_{i=1}^{m}$ and logical nodes as L_X, where X represents the type of logical node. Figure 6.2 graphically illustrates how these nodes are connected.

A logical node, is a conceptual node encapsulating a sub-component, a concept or a cluster of information. With this node, we can easily add semantics to the FetchGraph. The OpinVocab node, $L_{OpinVocab}$ is a logical node that encapsulates all terms in the review vocabulary (to help with review relevance scoring). $L_{OpinVocab}$ would thus have

edges to all relevant term nodes with edges holding the weight contribution of each term in the vocabulary. The query node, $L_{Query_{e_k}}$ is a logical node encapsulating the entity query for each e_k (to help with entity relevance scoring). To model the contents of an entity query, there is an edge from $L_{query_{e_k}}$ to all relevant term nodes.

A page is made up of several sub-components (e.g. URL, title, contents). We model the URL and contents of a page using logical nodes as both are used for relevance scoring. $L_{URL_{p_i}}$ represents the URL node and $L_{Content_{p_i}}$ represents the content node. Both these logical nodes link to term nodes to model term composition.

6.5 Efficient Review Crawling with FetchGraph

In this Section we will discuss how we use the FetchGraph for the review page collection task. The key steps in our instantiation of the framework include (1) finding initial CRPs, (2) expanding the CRP List, (3) Growing the FetchGraph and (4) Pruning the FetchGraph to eliminate irrelevant pages. The first two steps are independent of the FetchGraph and is already explained in Section 6.2. The main challenge in using the FetchGraph is how to grow the FetchGraph to include pages and establish relationships and how to compute relevance scores (to prune irrelevant pages) with respect to the FetchGraph. We will now focus on elaborating these two steps.

6.5.1 Growing the FetchGraph

Algorithm 5 outlines the construction of the FetchGraph for review page collection. We start with the set of CRPs collected for a given entity e_k. For any incoming page, we check if the page already exists in the graph (based on page URL). If a page is an existing page, then only the ownership of the page to entity e_k is determined. Entity ownership is revised if the $S_{ent}(p_i, e_k)$ score is larger for the current entity than the existing one (line 3-5).

Algorithm 5 $GrowFetchGraph(CRPList_{e_k}, e_k, G)$

1: **for** $CRP \in CRPList_{e_k}$ **do**
2: $p_i \leftarrow GetPageNode(CRP, G)$
3: **if** $pageExists(p_i, G)$ **then**
4: $S_{ENT}(p_i, e_k) = ComputeEntityRel(p_i, e_k)$
5: $UpdateEntityOwnership(p_i, e_k, S_{ENT}(p_i, e_k))$
6: **else**
7: **if** NOT $isNearDuplicatePage(p_i, G)$ **then**
8: $AddNode(p_i, L_{URL_{p_i}})$
9: $AddNode(p_i, L_{Content_{p_i}})$
10: $AddEdge(L_{URL_{p_i}} \rightarrow T)$ {U}RL tokens is made up of terms
11: $AddEdge(L_{Content_{p_i}} \rightarrow T)$ {c}ontent is made up of terms
12: $S_{rev}(p_i) = ScoreRevRel(p_i)$ {r}eview page relevance
13: $S_{ent}(p_i, e_k) = ScoreEntRel(p_i, e_k)$ {e}ntity relevance
14: $AddEdge(e_k \rightarrow p_i)$ {an entity owns the page}
15: $AddEdge(p_i \rightarrow s_k)$ {page is part of a site}
16: **else**
17: $AddToDuplicateList(p_i, G)$
18: **end if**
19: **end if**
20: **end for**

If a page does not already exist in the graph, a check is done to see if the page is a near duplicate page to an existing page (line 7). This usually happens when there are multiple URLs linking to the same page. If a page is a near duplicate, then this page is added to the duplicate list of the existing page (line 17). Otherwise, a new node p_i, representing this new page is created. We will not discuss how near duplicates are detected since this is not the focus of our work and we also turn this feature 'off' during evaluation. Next, the page related logical nodes described in the previous section are created. These nodes are added to the parent page, p_i (line 8-9). Then, these logical nodes are linked to relevant term nodes based on the textual contents within these components. For each component, there will be one edge for each unique term and the term frequencies are maintained at the edge level (line 10-11).

Once a page is added to the FetchGraph, the next step is to score the new page in terms of *review page relevance*, $S_{rev}(p_i)$ and *entity relevance*, $S_{ent}(p_i, e_k)$ (line 12-13). Note that for $S_{rev}(p_i)$, only the unnormalized scores are initially computed.

After scores have been computed, other relationships in the FetchGraph are established. An edge from the entity node to the page node is added to indicate entity ownership (line 14). Entity ownership of a page depends on which entity the page appeared as a CRP. We also link the page with the site that it originates from (line 15).

Once all CRPs for all entities have been added to the FetchGraph, the review page relevance score, $S_{rev}(p_i)$ is normalized using dependency information from the graph. Then the graph is pruned based on the $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$ relevance scores . This leaves us with a set of high confidence relevant review pages. Although pruning here is done at the very end, it can also be performed periodically as the graph grows.

6.5.2 Computing $S_{rev}(p_i)$ using FetchGraph

In Section 6.2.3.1 we presented our proposed method for scoring a page in terms of review page relevance using a review vocabulary and several normalization strategies. This vocabulary is modeled in the FetchGraph as described earlier where the terms and their contributing weights are encapsulated by $L_{OpinVocab}$.

Without the FetchGraph, to compute term frequencies we can use an in memory table. For this, we would first need to parse the page to obtain textual contents of the page and then term frequencies can be maintained using a table with a unique term as the key and frequencies as the entry. Assuming no database is maintained, to normalize the raw scores we further need to compute statistics such as the maximum term frequencies for all entity related pages (EntityMax normalization) which would require access to more pages and term frequencies of those pages. While parsing and computing these statistics just once may not seem too expensive, we need repeated access to some of this information (e.g. to compute normalizers) and repeated computation of the same information is a waste of time and resources. One may argue that we can keep track of all pages along with all sorts of statistics with just an in memory table. While this is feasible for a few pages, for large number of pages and entities, this would quickly become unmanageable and memory intensive. One possibility is to maintain an inverted index for each page collected. However, inverted indexes can only provide access to limited statistics. With the FetchGraph however, a page is loaded

into memory once for construction of the FetchGraph. After that, page related term frequencies can be directly accessed from edges linking to relevant term nodes.

Formally, given a page node p_i, its content node, $L_{Content_{p_i}}$ and the OpinVocab node $L_{OpinVocab}$, let T_c be all term nodes connected to $L_{content_{p_i}}$ and let T_{ov} be all term nodes connected to $L_{OpinVocab}$. For simplicity we refer to $L_{content_{p_i}}$ and $L_{OpinVocab}$ as L_c and L_{ov}. With this, the unnormalized $S_{rev}(p_i)$ score with respect to the FetchGraph is computed as follows:

$$S_{rev}(p_i) = \sum_{t \in T_c \cap T_{ov}} log_2[wt(L_c \rightarrow t)] * wt(L_{ov} \rightarrow t)$$

where $L_c \rightarrow t$ and $L_{ov} \rightarrow t$ refer to the connecting edges from L_c to t and L_{ov} to t.

To normalize the raw $S_{rev}(p_i)$ scores, we have several options as proposed in Section 6.2.3.1. Given P as all pages in the FetchGraph, let P_{e_k} be all pages connected entity node e_k and let P_s be all pages from a particular site s. With this the normalized scores are defined as follows:

GlobalMax : $S_{rev_{GM}}(p_i) = \frac{S_{rev(p_i)}}{\max\limits_{p \in P}(S_{rev(p)})}$

EntityMax : $S_{rev_{EM}}(p_i) = \frac{S_{rev(p_i)}}{\max\limits_{p \in P_{e_k}}(S_{rev(p)})}$

SiteMax : $S_{rev_{SM}}(p_i) = \frac{S_{rev(p_i)}}{\max\limits_{p \in P_s}(S_{rev(p)})}$

SiteMax + GlobalMax: $S_{rev_{SM+G}}(p_i) = 0.5 * S_{rev_{SM}}(p_i) + 0.5 * S_{rev_{GM}}(p_i)$

EntityMax + GlobalMax: $S_{rev_{EM+G}}(p_i) = 0.5 * S_{rev_{EM}}(p_i) + 0.5 * S_{rev_{GM}}(p_i)$

The subscripts GM, EM and SM represent GlobalMax, EntityMax and SiteMax normalization respectively.

6.5.3 Computing $S_{ent}(p_i, e_k)$ using FetchGraph

In Section 6.2.3.2, we propose to compute $S_{ent}(p_i, e_k)$ based on the similarity between the page URL and entity query using Jaccard similarity. Since a given page can appear in the CRP list of different entities, we will be computing the $S_{ent}(p_i, e_k)$ scores for the same page with different entities. Therefore, there will be repeated access to URL terms. To manage the URL terms without re-tokenizing it each time, we can maintain a table in memory with the URL of a page as the key and the list of URL terms as the entries. While this approach will work very well for a small number of pages, as the list of URL's grow (as more and more pages are crawled), the table will become huge as we maintain separate lists of tokens for each page and the terms can be repetitive across lists. In the FetchGraph however, there are no duplicate terms as we maintain one unique node for a given term. The term make-up of a URLs is modeled using an edge to the relevant term nodes. With this, the growth of the graph is much more manageable (we show later that the FetchGraph's growth is linear to the number of pages).

Given an entity node e_k and page node p_i, where p_i is connected to e_k, the $S_{ent}(p_i, e_k)$ with respect to the FetchGraph is computed as follows:

$$S_{ent}(p_i, e_k) = \frac{T_{URL} \cap T_Q}{T_{URL} \cup T_Q}$$

where T_{URL} contains all term nodes connected to $L_{URL_{p_i}}$ (logical node representing the URL) and T_Q contains all term nodes connected to $L_{Query_{e_k}}$ (logical node representing the entity query).

6.6 Experimental Setup

We evaluate our proposed OpinoFetch framework in terms of accuracy, give insights into efficiency and provide examples of application related queries that the FetchGraph can answer. For this we use three different domains - *electronics*, *hotels* and *attractions* where each of these domains is quite different from one another.

Dataset and Queries. The electronics domain is highly popular with reviews in a variety of sources ranging from personal blog sites to expert review sites. The hotels domain while not as popular as electronics, has abundance of reviews on well known travel sites such as Hotels.com and Tripadvisor. The attractions domain is least popular on the web and the available reviews in each source is often incomplete. Even on big sites like Tripadvisor, the reviews in the attractions domain only covers a small portion of all available attractions. In our dataset, we have a total of 14 entities for which reviews are to be collected (5 hotels; 5 electronics; 4 attractions).

In finding the initial set of CRPs, we use Google as the general search engine. Other search engines can certainly be used to find initial CRPs. However, our focus is on how we can identify relevant review pages from such starting points accurately and efficiently. In expanding the initial CRP list, we explore the top 10 ranked URLs (details in Section 6.2.2). For the entity query (search keywords) used with the search engine we used a descriptive query consisting of the *full entity name*, *address* (if location specific) and the term *reviews*. Thus, for an attraction such as Universal Studios in Florida the resulting query will be *Universal Studios, Orlando Florida Reviews*. Throughout our evaluation, we primarily use the top 30 Google results for each entity query.

Evaluation Measures. As our task is more of a *hyper-focused data collection task*, the actual pages that need to be collected are already close to the starting points. Thus, the difficulty is in finding the actual relevant content around the vicinity of these starting points. With this, we focus on a short range crawl rather than a long crawl. We show later that distant URLs yield in much lower gains in recall. Topical crawlers are usually evaluated by *harvest rate* which is the ratio between number of relevant and all of the pages retrieved [92, 43]. While it is interesting to see shifts in precision over number of retrieved pages for a long crawl, this is not so interesting for a short crawl where the number of pages crawled per entity is not very large. Thus, we measure precision and recall after the FetchGraph has been pruned.

Defining true recall for this task is extremely difficult as there is no mechanism to obtain all relevant review pages about an entity from the web, nor is it easy to crawl the entire web and identify review pages pertaining to the entities in our evaluation set. Given that most pages on the web are indexed by well known search engines, we approximate recall by constructing a gold standard judgment set that looks deeper into entity specific search results.

Specifically, to construct our gold standard judgments, for each entity query in our evaluation set, instead of using the top 30 results, we explore the top 50 results and

follow links up to a depth of 3 in order to build a link repository. We then ask human judges (through crowdsourcing) to judge if a given URL points to a review page for the named entity. We had a total of 57,154 unique (entity query + URL) pairs which called for 171,462 judgment tasks using 3 human judges for each task. The majority voting was used as the final judgment. To control the quality of the judgments, we introduced 50 gold standard judgment questions where for every few judgment tasks presented to the workers there will be a hidden gold standard question. If a worker misses too many of these gold standard tasks, then the contribution of this worker will be excluded. Precision and recall for an entity e_k are computed as follows:

$$Prec(e_k) = \frac{\#RelPages(e_k)}{\#RetrievedPages(e_k)}$$

$$Recall(e_k) = \frac{\#RelPages(e_k)}{\#AllRelPages(e_k)}$$

While the constructed judgments can provide a good estimate of precision and recall, the actual precision and recall is actually higher. This is because, first, there are many URLs with minor differences that point to the same content. The precision and recall would be higher if we capture all of these URLs even though in reality, capturing at least one is equally good. Resolving duplicates for each URL in this judgment set would be expensive and is unnecessary because ultimately what matters is the relative improvement in performance. Next, other than prioritizing the URLs to be crawled, we do minimal filtration on the specific URLs. Therefore, our judgment set can include pages in other languages. While it is easy for a human to judge if some of the pages in other languages contain reviews or not, our framework will most likely prune pages that are non-English and this lowers performance values.

Baseline. Since there is no other relevant work that has explored the collection of entity specific review pages, we do not have a similar competing method for comparison. We thus use Google results as a baseline as these search results are deemed relevant to the entity query and are 'close' to the actual information need.

During evaluation, we turned off several extended features: We turned off the *duplicate elimination* module so we do not tie duplicate pages together (which would improve precision); We place no restrictions on the type of URLs followed as there could be an many file types that can be eliminated; We also do not force crawling of English only content to enable future work in other languages.

6.7 Results

By default, the following are the settings used throughout our evaluation unless otherwise mentioned. Number of search results, $\sigma_{search} = 30$; CRP expansion depth, $\sigma_{depth} = 1$; Review normalization method: EntityMax+GlobalMax; Minimum review relevance score: $\sigma_{rev} = 0.1$, Minimum entity relevance score: $\sigma_{ent} = 0.1$; σ_{rev} and σ_{ent} are the only two thresholds to be empirically set where

Recall					
# of search results	**10**	**20**	**30**	**40**	**50**
Google	0.083	0.030	0.041	0.051	0.058
OpinoFetch	0.017	0.136	0.179	0.209	0.236
OpinoFetchUnnormalized	0.069	0.111	0.142	0.161	0.184
Precision					
Google	0.301	0.271	0.252	0.247	0.229
OpinoFetch	0.311	0.287	0.274	0.261	0.255
OpinoFetchUnnormalized	0.402	0.364	0.343	0.322	0.311
# of pages shortlisted					
Google	10	20	30	39	49
OpinoFetch	53	95	127	155	180
OpinoFetchUnnormalized	36	66	86	104	121

Table 6.1: Performance at different search sizes. OpinoFetch & OpinoFetchUnnormalized are based on best $F_{0.5}$ scores.

higher values yield in better precision and lower values favor recall. Our method is referred to as OpinoFetch.

Google Search vs. OpinoFetch. In Table 6.1, we report performance comparison of OpinoFetch, OpinoFetchUnnormalized and Google at different search result sizes. *OpinoFetch* uses *EntityMax+GlobalMax* normalization of $S_{rev}(p_i)$ and *OpinoFetchUnnormalized* does not use any normalization. Both these runs are based on the best yielding $F_{0.5}$ scores.

Based on Table 6.1 we see that the precision of Google search is low even though the number of search results is not very large (between 10 - 50). If we just considered the top 10 search results (where the precision is highest), on average, 7 out of 10 results do not link to relevant review pages. This shows that most of the search results are not direct pointers to review pages or are completely irrelevant to the entity query. Then, we can also see that with Google results there is limited gain in terms of recall even with increasing number of search results. This goes to show that there exists many more relevant pages in the vicinity of the search results then what the search engine sees as relevant. One may argue that an entirely different query would improve these results. However, general search engines like Bing and Google serve typical users who want results fast and tend to use less descriptive queries than what was used in our evaluation. Therefore, we expect the search results using our entity query to be more accurate compared to a non-descriptive query (e.g. *Universal Studios Reviews* which shows mixed results between the one in Florida and Hollywood).

From Table 6.1, we can also see that the performance of OpinoFetch (both the normalized and unnormalized versions) is significantly better than plain Google search. The recall steadily improves when more and more search results are used. This shows that there is a lot of relevant content around the vicinity of the search results and our approach that looks for such relevant content is effective in that we are able to identify a lot of these relevant review pages. As we pointed out in Section 6.6, the actual recall and precision values would be

higher if we discount redundancies and language barriers.

Does normalizing $S_{rev}(p_i)$ yield in better performance? In this work, we have assumed that by leveraging the dependencies in the FetchGraph, we can make more accurate review page relevance predictions. Our review page relevance score, $S_{rev}(p_i)$, uses dependencies in the FetchGraph to obtain normalization scores. From Table 6.1, we see that the use of normalization maintains higher recall than without normalization. Observe that OpinoFetchUnnormalized prunes many more pages than OpinoFetch (including many relevant pages), artificially increasing the precision, but the recall is adversely affected. By using dependency information from the FetchGraph, we actually avoid pruning pages that seem irrelevant with unnormalized scores but are actually relevant. This this why OpinoFetch has better recall than OpinoFetchUnnormalized even though both have the highest $F_{0.5}$ scores.

How many levels to explore? By default, in our evaluation we use $\sigma_{depth} = 1$. Now, we look into how much improvement we see in terms of recall by following links at different σ_{depth} levels. We fix $\sigma_{search} = 30$ and compare gain in terms of recall at different search depths. The results are shown in Figure 6.3. Notice that we gain the most in terms of recall by just analyzing links within the search results ($\sigma_{depth} = 1$) and as we follow links that are further away from the search results, the gain in recall keeps dropping. While the search results itself may not be direct pointers to review pages, there are actually many relevant review pages that are close to the search results and as these links are discovered, recall significantly improves. On the other hand, as the crawler digs deeper and deeper, the relevance of the links followed to the target entity (i.e. entity query) declines and therefore the gain in recall is also much lower. Thus, the best crawl depth is $\sigma_{depth} = 2$ as crawling further does not improve recall significantly.

Also notice that the attractions domain gains the most in terms of recall at every level. This is because reviews in this domain are sparse and any additional links followed yields in more review pages compared to just the search results which had very low precision and recall to start with.

Is one domain harder than another? While collecting review pages may seem like a generic task for all entities, the difficulty in collecting reviews in one domain can be quite different from another. In our evaluation, we have observed that collecting reviews from the *attractions* domain was most difficult with lowest precision and recall as shown in Figure 6.4. One reason for this is because the attractions domain has relatively fewer reviews and review sites compared with the other two domains. Thus, there is a higher likelihood of collecting and following links to completely irrelevant pages. More importantly, we have observed a lot more ambiguity in the attractions domain compared to the electronics or hotels domain. For example, one of our entities in the attractions domain is *Disneyland Park Anaheim California*. Based on our investigation, we noticed many unrelated entities with their own review pages that carry the name of this entity. Examples are review relates to *Space Mountain Disneyland*

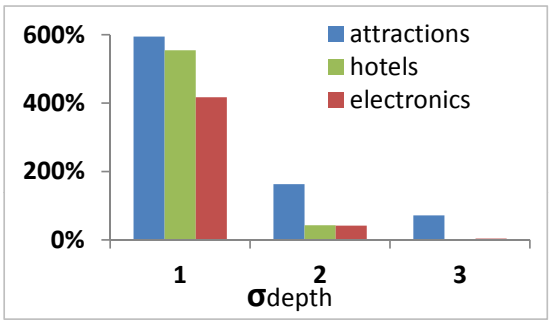

Figure 6.3: Gain in recall at different depths using OpinoFetch.

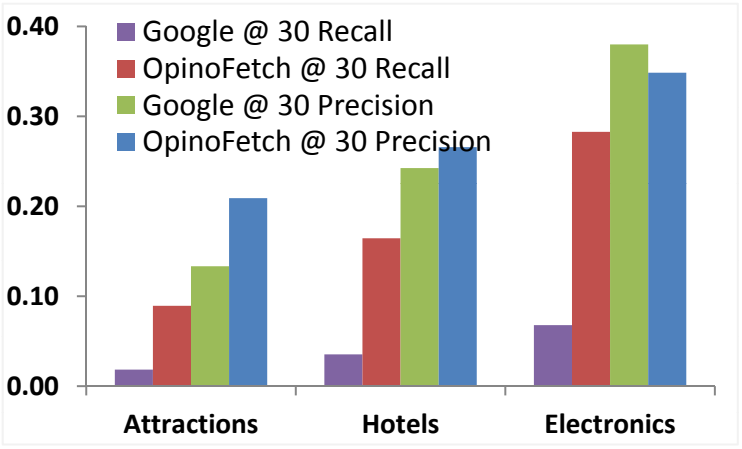

Figure 6.4: Precision and recall of Google and OpinoFetch in different domains with $\sigma_{search} = 30$

Park (a fun ride) and *Fairfield Inn Anaheim Disneyland Park Resort* (a hotel).

Best normalization method for computing $S_{rev}(p_i)$. To determine the best normalization strategy of $S_{rev}(p_i)$, we look into the precision and $F_{0.5}$ scores using different strategies across $\sigma_{rev} \in \{0.1, 0.3, 0.5\}$. We set $\sigma_{ent} = 0$ to turn off pruning based on entity relevance. The results are summarized in Table 6.2. First, notice that all normalizers improve precision of the results and is especially clear for attractions domain. Next, we see that the methods that incorporate EntityMax have higher levels of precision than the ones that incorporate SiteMax. This is reasonable, because a popular site like Tripadvisor would cover entities from different domains (e.g. hotels and attractions). Thus, the maximum the $S_{rev}(p_i)$ score from such a site may be too high for sparsely populated domains such as attractions resulting in unreliable normalized scores. This is why the attractions domain has the lowest precision when we use the SiteMax normalizer.

EntityMax uses the maximum score of pages related to one entity and thus the score gets adjusted according to entity popularity. Interestingly, EntityMax+GlobalMax performs slightly better than EntityMax in terms of precision

	Hotels		Attractions		Electronics		Average	
	P	$F_{0.5}$	**P**	$F_{0.5}$	**P**	$F_{0.5}$	**P**	$F_{0.5}$
EM + GM	**0.356**	0.218	**0.346**	0.152	**0.378**	0.316	**0.351**	0.229
SM + GM	0.261	0.226	0.201	0.156	0.338	**0.318**	0.264	0.234
EM	0.350	**0.229**	0.311	**0.162**	0.374	0.315	0.337	**0.235**
SM	0.238	0.222	0.161	0.145	0.325	0.317	0.240	0.228
No Pruning	0.218	0.220	0.115	0.124	0.294	0.302	0.209	0.215
Change in precision over no pruning								
EM + GM	**+63.03%**		**+200.28%**		**+28.38%**		**+97.23%**	
SM + GM	+19.47%		+74.31%		+14.91%		+36.23%	
EM	+60.34%		+169.87%		+26.94%		+85.72%	
SM	+8.87%		+39.56%		+10.44%		+19.62%	

Table 6.2: Avg. precision (P) and $F_{0.5}$ across $\sigma_{rev} \in \{0.1, 0.3, 0.5\}$ with different normalizers. EM=EntityMax; SM=SiteMax; GM=GlobalMax

likely because we also use the global maximum which boosts the scores of densely populated review pages and reduces the scores of sparsely populated ones.

6.7.1 Site Coverage

One could argue that it is possible to obtain reviews about all entities in a particular domain (e.g. hotels, electronics, etc) just by crawling a few major opinion sites. However, based on our observation, even entities within the same domain can have a very different set of review sources and thus just a handful of opinion sites would not cover all reviews about an entity. We would thus like to show that OpinoFetch can reach out to long-tail reviews that we would not be able to obtain by just crawling a few major opinion sites. We refer to this analysis as site coverage.

For the site coverage analysis, we run OpinoFetch (with a crawl depth of 2) using the top 100 search results from Google for 4 entities within the electronics domain. We then compile a list of sites for URLs deemed relevant by OpinoFetch for each of the 4 entities. In creating the review site list, we eliminate all redundancies and normalize international sites and sub-domains (e.g. asia.cnet.com, reviews.cnet.com and www.cnet.com would be converted into cnet.com). With this, we have a unique list of review sites for each entity. Given this list, we categorize all sites that appear in the top 20 search results of each entity as major opinion sites. Since users typically only look at the first few pages of the search results, Google tends to rank all the sites deemed relevant and important before other 'less important' sites. Thus, this strategy of considering sites that appear in the top 20 search results as the major opinion sites is reasonable. All other sites found using OpinoFetch are then regarded as long-tail sites.

Table 6.3 shows the distribution of sites for all 4 entities and Table 6.4 shows examples of major opinion sites and long tail review sites for two of the entities. Based on Table 6.3, we can see that in all cases, more than 50% of the relevant review pages are from long-tail sites. This goes to show that

Entity Type	Major Opinion Sites	Long Tail Sites	# Unique Relevant Sites
Apple iPhone 64GB 4S	28.26%	71.74%	46
Garmin Nuvi 205	25.00%	75.00%	20
HP Touchpad Tablet 16GB	32.35%	67.65%	34
Nikon D5100	18.84%	81.16%	69

Table 6.3: Distribution of major opinion sites vs. long tail sites. Note that all sites are unique accounting for sub-domain differences, internationalization and any form of redundancies.

Entity Type	Major Sites	Long Tail Sites
Nikon D5100	target.com reviews.bestbuy.com ebay.com costco.com consumerreports.org pcmag.com	cameras.pricedekho.com club.dx.com digital-photography-school.com kenrockwell.com nikondslrtips.com photographylife.com
HP Touchpad 16G Tablet	reviews.officemax.com newegg.com computershopper.com engadget.com expertreviews.co.uk pcworld.co.uk wired.com	webosnation.com anandtech.com pcpro.co.uk pocket-lint.com forum.tabletpcreview.com winnipeg.kijiji.ca tabletconnect.blogspot.com

Table 6.4: Example of major opinion sites and long tail review sites for Nikon D5100 camera and HP Touchpad 16GB Tablet.

there are a lot of reviews that exist in a variety of different sources than just the major opinion sites. Also, note that the number of sites containing relevant reviews about the entities are very different even though they all are electronics. An example of a review page from a long tail site is `http://www.kenrockwell.com/nikon/d5100.htm` for the Nikon D5100 camera. Another example is `http://www.webosnation.com/review-hp-touchpad` for the HP Touchpad 16GB Tablet. Both these sites contain personal reviews on the corresponding products which will be a value add when aggregated with reviews collected from all other sources.

6.7.2 Time and Memory Analysis

The OpinoFetch framework is developed in Java. For all experiments we use a 2x 6-core @ 2.8GHz machine with 64GB memory.

One key advantage of using the FetchGraph is its ability to keep most information related to the data collection problem encapsulated within a single heterogenous network. This can range from representing information about entities to individual terms within the data collection vocabulary. With this, it is quite possible for the network to grow too large too fast and not fit in memory for processing. In Figure 6.5, we can see that the FetchGraph's growth is ac-

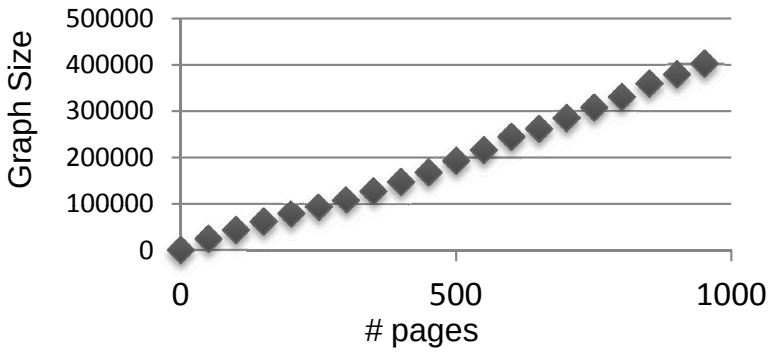

Figure 6.5: Growth of FetchGraph with respect to number of pages collected.

	+FetchGraph	**-FetchGraph**
$S_{rev}(p_i)$ **(unnormalized)**	0.085ms	8.62ms
EntityMax Normalizer	0.056ms	4.39s

Table 6.5: Average execution time in computing $S_{rev}(p_i)$ & EntityMax normalizer with and without the FetchGraph.

tually linear to number of pages collected and this is without any code related optimization or special filters (which can decrease overall nodes created). If we added 1 million pages to the FetchGraph and assume that each node and edge are represented by objects of size 50 bytes (base object is 8 bytes), the resulting FetchGraph would be approximately 20GB, which is still manageable in memory and would only reduce in size with various optimizations.

Another advantage of using the FetchGraph is efficiency in information access. When we need access to dependency information (e.g. in computing normalizers for $S_{rev}(p_i)$) or repeated access to various statistics (e.g. page related term frequencies), it is not possible to obtain such information easily or efficiently without a proper data structure. Due to the versatility of the FetchGraph, once a page gets added to this information network, it becomes easy to access all sorts of information from the network.

Table 6.5 shows execution time of computing the unnormalized $S_{rev}(p_i)$ score and execution time for computing the EntityMax normalizer using the FetchGraph and without it (averaged across all domains). It is clear that even to compute the unnormalized $S_{rev}(p_i)$ it would be quite expensive to repetitively compute and recompute these scores without any supporting data structure. This becomes worse when we normalize the scores as seen in the time to compute the EntityMax normalizer without the FetchGraph. The execution time utilizing the FetchGraph is notably lower as the page is only loaded into memory once and all other statistics can be obtained by accessing the FetchGraph directly. While it is feasible to use a database for some of these tasks, the FetchGraph is an in memory data structure and thus is much faster than accessing the database especially when large joins are expected. Also, since we can sep-

Electronics (3.92 ms)			Attractions (2.60 ms)		
Site	$S_{rev}(p_i)$	**# Ent.**	**Site**	$S_{rev}(p_i)$	**# Ent.**
Amazon	57.90	5/5	Yelp	41.65	4/4
Bestbuy	19.48	5/5	Tripadvisor	32.76	4/4
Ebay	20.71	5/5	Yahoo! Travel	1.92	4/4
Cnet	17.94	4/5	Rvparkreviews	14.21	2/4
Digitaltrends	5.37	2/5	Virtualtourist	6.24	2/4
Techradar	5.83	2/5	Igougo	5.06	2/4

Table 6.6: Snapshot of results for the query *select PopularSites(10) from FectchGraph(D) order by EntityCount*; *D=Electronics* and *D=Attractions*. $S_{rev}(p_i)$ represents the cumulative $S_{rev}(p_i)$ score for the site.

arate the data collection problem (e.g. by domain), we only need to load the required networks into memory.

6.7.3 Sample Query & Results

One of the important uses of the FetchGraph is to answer application related questions. Assuming we have a special query language to query the FetchGraph, one interesting question is: *What are the popular review sites in a given domain?*.This query is quite typical of business intelligence applications that perform analysis on subsets of data. Using the FetchGraph this information can be obtained by ranking the sites based on *indegree* information and cumulative per site $S_{rev}(p_i)$ scores which would result in popular and densely populated review sites to emerge at the top.

Table 6.6 shows a snapshot of results requesting top 10 popular sites for the *electronics* and *attractions* domain. In total, there were 77 sites for attractions and 100 for electronics. First, it is obvious that the list of review websites vary greatly from domain to domain. Then, we also see that not all sites within a given domain contain reviews for all the entities. This is intuitive as some sites may be very specific to a subset of entities (e.g. only cell phones) or some sites may contain incomplete directory listings or product catalogs. The more striking fact is that all this information (including score aggregation and ranking) can be obtained very quickly from the FetchGraph (3.29 ms for electronics and 2.60 ms for attractions).

6.8 Discussion

In our current approach, we rely solely on entity relevance and review page relevance scores in order to find review pages about an entity. While it is easy and fairly effective to score the pages in this manner, setting thresholds can be a bit of a problem as different entities and domains may behave differently with different thresholds. When this is a concern, our heuristics based approach can

actually be extended and used in a semi-supervised setting.

The idea is as follows; Instead of treating all search results from a general web search engine equally, we can use relevant review pages from the top N search results as training examples. For example, using our current scoring approach, we can use the top 10 search results from Google to find a set of high confidence review pages (i.e. review pages that score highly in terms of entity relevance and review page relevance). These high confidence seed review pages can then be used as training examples in order to find other relevant review pages from the search results (and vicinity of the search results). We can use a simple classifier such as nearest neighbor for this purpose. Training examples can be incrementally added several times by applying our heuristics scoring method on pages classified as relevant, again treating the very high scoring pages as training examples. One advantage of this approach is that we only use heuristics to find very high scoring pages to serve as training examples. All other pages do not have to use these heuristics and the relevance in this case would be how close all other pages are to the sample pages.

6.9 Conclusion

In this chapter, we proposed an *unsupervised* framework for collecting online opinions namely reviews for *arbitrary entities*. We leverage the capabilities of existing Web search engines and a rich information network called the FetchGraph to efficiently discover review pages for arbitrary entities.

Our evaluation in three interesting domains show that we are able to collect *entity specific review pages* with reasonable accuracy in an unsupervised manner without relying on large amounts of training data or sophisticated Named Entity Recognition tools. We also show that our approach performs significantly better than relying on just search engine results and we can achieve higher accuracy using the dependency information from the FetchGraph. Our analysis shows that the FetchGraph supports efficient lookup of various statistics and helps answer interesting application questions, making it a queriable network.

Compared with existing approaches in topical crawling, our approach is practically oriented, unsupervised and is domain independent, and is thus immediately usable in practice. The proposed FetchGraph is also highly flexible and can be extended to different data collection problems such as collecting news articles about specific topics.

7 DEMO - FINDILIKE: PREFERENCE DRIVEN ENTITY SEARCH

To showcase the power of some of the proposed ideas in this thesis, I developed a web demo system called FindiLike. This system is capable of finding hotels based on preferences of the user (structured or unstructured opinion preferences) and beyond search this system also provides analysis capabilities in the form of opinion summaries as well as tag cloud visualization of reviews. This system was demonstrated in the context of hotel search in the WWW 2012 demo session [4]. This chapter provides a brief overview of the demo system as well features that were demonstrated.

7.1 Introduction

Web search engines enable users to find all sorts of documents based on a topic of interest. However, with the growth of online content, more and more people are interested in finding entities or objects instead of just documents. This is especially true in decision making scenarios where a user would often like to find entities such as hotels, restaurants and doctors based on their personal requirements. While current search engines like Google are able to recognize certain types of entities (e.g. products and location) these search engines have limited capability in assisting users with decision making. Thus, in a decision making scenario such as choosing a place to eat or a doctor to see, users would turn to sites like Yelp which have better support for decision making where users can select entities of interest based on attributes such as *price*, *location* and *service* and also by reading the unstructured *reviews* of these entities.

Similar to Yelp, vertical search systems like Amazon, Hotels.com, and Bing Shopping facilitate decision making by providing domain specific navigation capabilities in the form of search filters. These filters which are often based on structured information (e.g. *price*, *brand* and *color*), help users to quickly narrow into entities of interest. However, filters based on only structured information, limit the capability for selecting entities based on the unstructured opinions of other users, which is another important factor in decision making. The closest to an 'opinion filter' is the ability to limit entities by the *overall user ratings* which would still force users to read the reviews to ensure that the opinions within these reviews fulfill their requirements. Suppose, a user was looking for a place to eat and wanted a *quiet* restaurant with *good service*. In this case, just limiting the entities by the overall ratings would clearly not be useful. The

user would still need to digest the reviews of all restaurants in consideration to find those that satisfy this criteria. Further, not all sites have the 'overall ratings' feature which makes it even harder to leverage existing opinions.

In reality, the opinions of other users is an important influencing factor in our day to day decision making tasks, ranging from which doctor to see to which of the many smart phones to purchase. However, digesting all the available opinions is time consuming and can become confusing over time due to the sheer volume of available opinions. To truly facilitate decision making, opinions should be leveraged in a more efficient manner and should be tightly integrated with the core decision making components of a system.

Existing works [93, 22, 23] have attempted to resolve this problem through summarization of opinions to help users better digest all the opinions. However, when dealing with a large number of entities, even summaries would get confusing as users would still need to keep track of how well each entity fulfills their opinion requirements. Thus, to provide a more direct support for a user's decision making task, we have developed FindiLike, a novel system capable of ranking interesting entities such as hotels based on a set of heterogenous preferences with unstructured *opinion preferences* being a major component of the system. The idea behind FindiLike is to allow users to specify key preferences upfront to the system. These preferences include structured preferences (e.g. *price* and *distance*) as well as unstructured *opinion preferences* that can be specified using descriptive keywords (e.g *clean rooms, cheap, good breakfast* in the context of hotels). With this, the system then scores relevant entities based on how well these entities match the specified preferences. What makes this system unique is that unlike faceted navigation which only filters out 'irrelevant' entities, FindiLike ranks entities by how well *key* preferences are matched, giving users the flexibility in selecting entities based on preference tradeoffs. On top of that, FindiLike allows users to analyze ranked entities using opinion summarization tools which is rarely available with other entity search systems. In the long run, FindiLike aims to evolve into a complete decision making platform for different types of entities. We demonstrate our current system in the context of hotel search. Additional information about this demo can be found at: `http://info.findilike.com`.

7.2 Architecture

In this section, we provide a brief overview of the FindiLike system architecture. FindiLike is a web application that enables users to *find* entities based on structured and unstructured set of preferences. Although this ranking task resembles the entity ranking task studied widely by the information retrieval and database communities [94], our task is actually quite different. The goal of entity ranking or entity retrieval is to return relevant entities instead of just

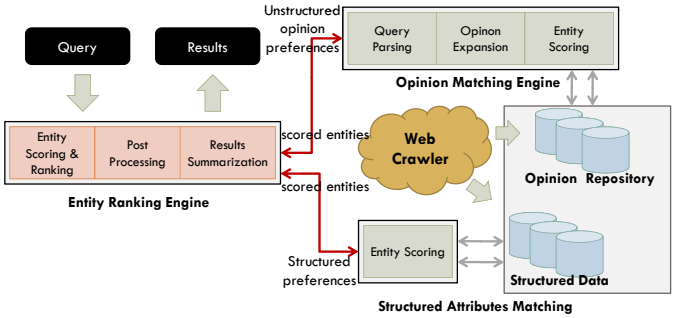

Figure 7.1: Preference driven entity search architecture. The architecture supports both structured and unstructured preferences.

documents. The entities are ranked according to how well these entities satisfy a *topic* described in *natural language text*. While the goal of FindiLike is also to retrieve relevant entities, FindiLike ranks entities in the order of likelihood an entity matches a set of *user preferences* rather than just a topic described in plain text. We thus refer to our special set-up as *preference driven entity search*.

Just like any other entity search engines, the FindiLike system consists of several key components ranging from the user query component to a data collection component as shown in Figure 7.1. In brief, the system takes in user specified preferences and sends these preferences to the relevant scoring engines: opinion preferences to the *opinion matching engine* and structured preferences to *structured attributes matching* module. These scoring engines score a subset of entities (e.g. all hotels in a particular location) based on how well these entities match a given preference. The individual preference scores are then combined and the entities are re-ranked based on these new scores. The summarization module generates a summary of the top N relevant entities which are then displayed to the user. The user then has the option of adding more preferences or has the option of using the analysis tools to further assist them with decision making. In the next few sections, we provide more information about some of the key components of the system.

7.2.1 User Query

The query to the FindiLike system is a set of preferences. These preferences can be structured by nature such as preference for price, preference for distance and etc. and can also include unstructured preferences for opinions (e.g. desiring a clean hotel when finding hotels at a destination). While opinions can be extracted and used as structured preferences, this information extraction task would be very costly on a large scale and would also force users to express preferences on pre-defined aspects of an entity, which is rather restrictive. With FindiLike, we avoid the need for any information extraction by directly using the review texts of each entity as will be explained in Section 7.2.2.

The structured preferences provided by the user can vary greatly depending on the application domain. In the case of hotel search, we allow the user to explicitly specify structured preferences on distance from a particular landmark and the desired price range. While various attributes of an entity may be used for preference based ranking, we believe that it is only essential to use the most important attributes for such a ranking feature leaving the rest of the attributes as filters just as in faceted navigation.

As for the unstructured opinion preferences, we ask users to state their opinion preferences using a set of descriptive keywords. These keywords would indicate what the user desires in the different aspects of an entity. For example, to show desire for clean hotels with friendly staff, the user may specify a query such as *clean rooms, friendly staff* or *clean place, friendly service*. The ability to specify preferences using free-form text enables users to express preferences on any arbitrary aspect and for any type of desired opinions. In accepting a user's unstructured preferences, different types of user interfaces may be used. The most general interface would be a single text field that would allow users to express preferences using natural keywords. Aspects in the query can then be obtained using various query segmentation techniques. To make this more practical, in our system, users can specify all their preferences in a single query box using a special delimiter such as 'and' or comma to separate each preference. We also allow users to incrementally add preferences as needed instead of re-entering the entire query.

This type of unstructured expression for opinion related preferences, brings about a new type of query understanding problem. The opinion preferences expressed by users can often be ambiguous, and there can be multiple ways to express similar preferences. For example, the expression "good breakfast" is similar in meaning to "great breakfast". To help with the matching of opinions, it would thus be beneficial to expand such a query by adding synonyms of the sentiment word. To this end, we have implemented ideas from [1] in dealing with some of the query understanding problems namely for the task of opinion expansion.

7.2.2 Entity Ranking Engine

For a given class of entities (e.g. hotels in Chicago), the entity ranking engine takes in a set of preferences and attempts to find entities that match all of these preferences. Each entity in the given class is scored based on how well it matches each preference and then the scores are combined. The top N scoring entities are returned as relevant results. By default, these entities are ranked in the order of likelihood an entity matches a user's preferences. More formally, given a set of preferences, $P = \{p_1, ...p_n\}$, the score of an entity E from class i is computed as follows:

$$S(P, E_i) = \frac{1}{n}[S_{structured}(P, E_i) + S_{opinions}(P, E_i)]$$

where, $S_{structured}(P, E_i)$ is scoring based on structured preferences such as price and distance and $S_{opinions}(P, E_i)$ is scoring based unstructured preferences which in this case is opinions. All preference scorings are on a scale of 5.

7.2.2.1 Scoring of Structured Preferences

The scoring of structured preferences is based on how well an entity fulfills the given criteria. Suppose, the preference for the *maximum price* aspect is set at \$60. If an entity's price is \$70, this entity does not quite fulfill the given criteria. In such a case, instead of completely penalizing this entity, the entity is given a score lower than the maximum possible score depending on how much it violates the criteria. In this example, the entity's price exceeds the maximum price by \$10, so this entity may be assigned a score of 4/5 instead of the maximum score of 5/5 for this specific aspect. On the other hand, if the entity's price falls within the maximum price requirement, then the entity immediately receives a full score on this aspect.There are several advantages to scoring entities in this way as opposed to completely eliminating entities. First, a user's requirement can sometimes be unrealistic and complete elimination could yield in no results being returned, which is not good from the perspective of user experience. In contrast, scoring entities with respect to the level of violation would yield in results that most closely match the specified criteria only that the scores could be much lower, which would then encourage users to change their expectations. Users may also be willing to loosen their requirements on some aspects if other aspects of an entity is in their favor. Using the previous example, while the entity's price exceeds the maximum desired by \$10, this entity may have matched other preferences extremely well. In this case, the user may decide to give in to the higher price as all other aspects of the entity are appealing to the user.

7.2.2.2 Scoring of Opinion Preferences

As we avoid the need for costly opinion mining and information extraction, the scoring of opinion preferences differs from how structured preferences are scored. In FindiLike, opinion preferences expressed using descriptive keywords are scored against the review texts written by both experts and average users. This matching task is quite different from keyword search in databases [95] where the goal is to find objects where any of its fields match the given keywords. Our idea is to represent each entity with the unstructured text of all the reviews of that particular entity, often available from various websites. Given a user's keyword preferences that expresses the desired features of an entity (e.g *clean and safe* for a hotel), we then score the relevant entities based on how well

its reviews match the user's preferences using a retrieval model as described in Ganesan & Zhai [1]. The more relevant mentions there are in the reviews of an entity, the higher the score an entity receives. Since opinions are highly subjective, it is often difficult to clearly determine if the opinions accurately describe an entity. However, if different users express similar dissatisfaction and appreciation about an entity, then its more likely that these opinions reflect accurate descriptions of the entity which is the idea behind our scoring mechanism.

7.3 Demo

We will demonstrate our system in the context of hotel search which is accessible at `http://www.findilike.com`. We will demonstrate the following features of FindiLike:

Ranking hotels by preferences. FindiLike enables users to find hotels using opinion driven preferences and other preferences such as distance and price. Suppose the user needs to find hotels close to the Los Angeles Convention Center and wants a hotel in a safe location. Using the FindiLike system, the user can find relevant hotels based on all these requirements. The preference for proximity to the Convention Center can be specified under the 'distance' tab and the preference for hotels that are said to be safe can be specified using the main search box using natural keywords such as *safe neighborhood*. Once all the requirements have been specified, all hotels in the Los Angeles area are then scored based on how well each hotel matches the specified preferences. The results are then ranked in the descending order of the scores as shown in Figure 7.2 (under 'YourMatch'). The individual preference scores are displayed using 'green stars'.

Summarizing ranked hotels. In traditional web search, a user often navigates into search results based on the relevance of the summary snippets to the query. Since, in our case, the user is looking for a hotel based on a set of preferences, a summary of the selected preferences is displayed to the user (as shown in Figure 7.2). For opinion preferences, snippets from the user reviews are displayed. For the distance preference, distance of the hotel from the selected landmark along with the total driving time are displayed.

Browsing opinions via summaries. To further assist users in their decision making process, we help users navigate the opinion space using automatically generated summaries. In most systems, the closest to an opinion summary is the averaged overall ratings provided by different users. Unstructured summaries which can often be more informative [2, 3], is almost never available in existing systems. FindiLike is capable of generating unstructured summaries of opinions, so as to help users digest the most common praises or complaints within the reviews. A snapshot of summaries generated for a hotel in Los Angeles is shown in Figure 7.3. In addition to summaries, to help users visualize mentions within

opinions, we also provide a tag cloud representation of the common mentions under the 'What's Buzzing' tab as shown in Figure 7.4.

List view vs. map view. Two kinds of presentational views are supported by the system. The list view as shown in Figure 7.2 and a map view as shown in Figure 7.5. In list view, the results are organized in a flat list ranked by the overall preference score. This view provides a detail summary of the results and enables users to easily select links and navigate into other components of the system. In map view, the results are displayed on a Google Map, with a small list type summary of the search results on the left. Individual markers need to be selected to see detailed summaries and to navigate into other components of the system.

Amati2002

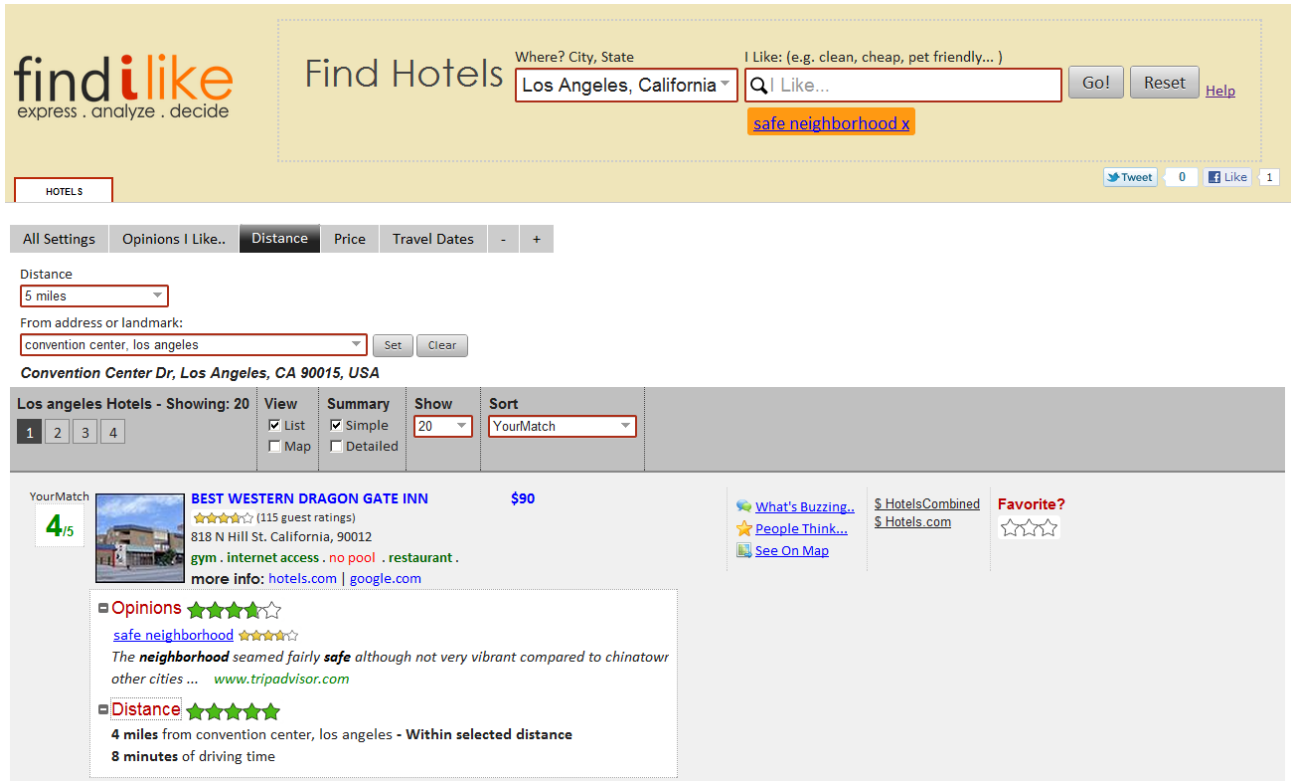

Figure 7.2: FindiLike preference based search. Results shown for the query: "Hotels within *5 miles of the Los Angeles Convention Center* and located in a *safe neighborhood*". *YourMatch* score on far left shows how well all preferences are matched on a scale of 5. Green stars indicate individual preference match levels.

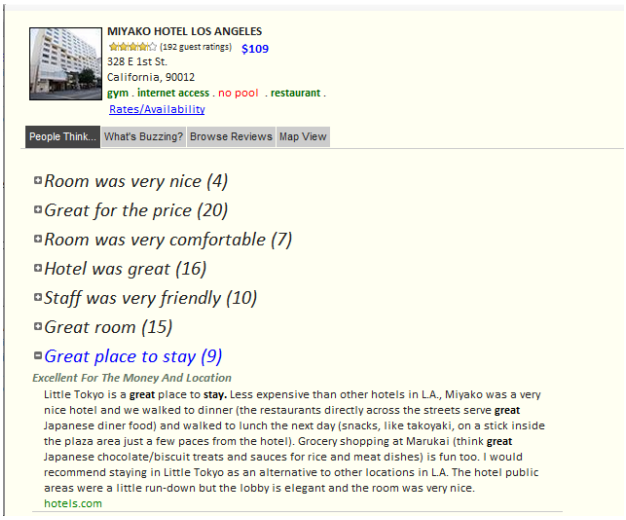

Figure 7.3: Opinion summaries generated by FindiLike for a hotel in Los Angeles. Numbers within parentheses indicate the number of supporting mentions for that particular summary. A click on each summary will display all the supporting reviews.

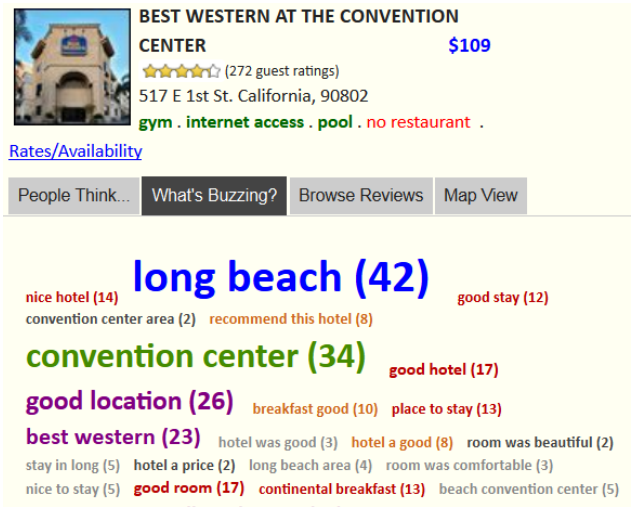

Figure 7.4: 'Clickable' tag cloud visualization of common mentions within reviews.

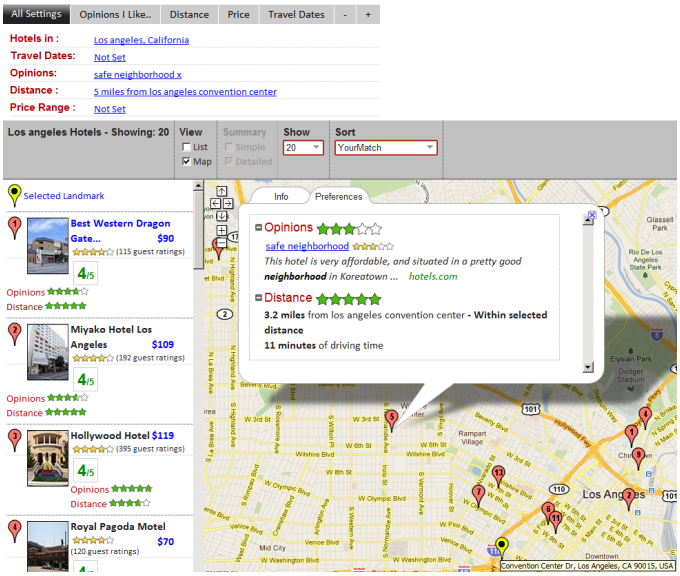

Figure 7.5: Results in map view. Red markers represent relevant results. Yellow marker represents the selected landmark.

8 CONCLUSIONS

8.1 Summary

The main motivation for this thesis was to develop methods towards enabling Opinion Driven Decision Support capabilities. This is an important problem to address as opinions are becoming a mainstream source of information in many of our day to day decision making tasks. Thus, the ability to utilize opinions efficiently to support all sorts of decision making tasks would greatly improve user productivity. While there are many aspects of an Opinion Driven Decision Support System that can be solved, this thesis focusses on areas where essential tools or methods are absent in practice or in existing literature. Further, to enable these tools to be usable in practice and easily integrated into existing applications, the methods proposed in this thesis are made to be general and lightweight. With this, all the proposed methods are unsupervised and rely on limited external resource.

The first aspect addressed in this thesis is the ability to *find entities based on existing opinions*. This is to significantly reduce the number of opinions a user would have to explore and digest in order to find one or more entities of interest. To this end, existing information retrieval models were proposed for this task as information retrieval models are robust, general and can be redefined in many ways. Also these models tend to scale up to large amounts of texts which makes this a highly appealing approach to solving this new search problem.

The second problem that this thesis addresses is methods for generating *concise textual summaries of opinions*. This is to enable users to get a quick understanding about the good and bad about a specific topic or entity and this type of summary also complements the well studied structured summaries. Two different flavors of abstractive summarization approaches were explored. The first approach referred to as *Opinosis* is a graph based summarization framework which relies on structural redundancies between sentences. The second approach *WebNgram* is an optimization framework that attempts to maximize the readability and representativeness of the generated summary. Both these approaches are unsupervised, lightweight and rely mostly on the existing text to generate concise summaries.

The third problem addressed in this thesis is automatic collection of opinions for arbitrary entities. Without a complete set of opinions about an entity, it is

difficult for users to get a complete understanding of the underlying sentiments. Further it would be hard for analysis tools to provide accurate information if the opinions are biased or are very sparse. With this, an unsupervised, lightweight and practical and method called OpinoFetch was proposed in order to collect opinions about any type of entity.

To showcase the power of some of these proposed ideas in a *decision making platform*, a web demo system, FindiLike was developed. This system is capable of finding hotels based on preferences of the user (structured or unstructured opinion preferences) and beyond search this system also provides analysis capabilities in the form of opinion summaries as well as tag cloud visualization of reviews.

8.2 Future Work

The Opinion Driven Decision Support System proposed in this thesis opens up an endless possibility of new research in the areas of Natural Language Processing, Text Mining and Information Retrieval. The tools and techniques proposed as part of this platform are just an initial step towards building a complete ODSS platform. There are many interesting future research directions that can be further pursued:

8.2.1 Opinion Based Entity Ranking

Improving entity relevance through phrasal search. In the current work on Opinion Based Entity Ranking, we have looked into the use of information retrieval models without emphasis on the proximity of keywords. While this presents a highly general approach to querying, there can be cases where this approach would result in false positives. For example, if a user looks for hotels that are 'close to university', it is quite possible that the system would return hotels that are 'close to airport' because of matching most of the terms in the query (except the word 'university') at different positions in the opinion document. If we impose phrase restriction, then we can limit deviation from the actual query because we require that the query words exist and should also exist within close proximity. While in theory, this approach seems to be 'the way to go' for this ranking task, in practice however this approach does not work well because it demands a lot of evidence in which all of the words have occurred in close proximity. One possible way to address this is to use a "back-off" style scoring where you first score entities based on the imposed phrase restriction and then remove this restriction and score based on individual words. With this, even if the words in the phrase never really occurred in close proximity the system does not return empty results.

Using click-through and query logs to improve ranking of entities. Since we are logging the queries and click through information using the FindiLike web system, we can further study how to improve the ranking of entities using the available logs. For example, using previous click through information we can re-rank the search results based on hotels that users have clicked on for a similar query.

Addressing vocabulary gap between query and reviews. In ranking entities based on preferences, we currently rely on surface level keyword matching of words in the query and corresponding reviews. This approach does not consider the true sentiments or semantics within the reviews. While this shallow approach works well with many queries or when there is a high volume of reviews, there are cases where this shallow matching can result in false positives. For example, in finding hotels with mentions of 'clean rooms', a hotel may be ranked highly even if the user had mentioned "rooms not clean".

There are several ways in which this problem can be subdued. First, since users in general would desire entities that have positive ratings, we can first do a basic sentiment analysis on the reviews and rank entities with more positive sentiments before those with more negative sentiments. With this, it would be more likely that the keyword matching on the positively rated entities would yield in a true positive than a negatively rated one. Methods as proposed by Wang et al. [21] can be used to decompose the sentiment ratings within the reviews.

Another way to address this, is to take into account proximity of negative or unwanted words near the actual query words (referred to as *noise words*). If a noise word appears near the query words, then the matching score of the review to the query should be discounted. The closer the proximity of noise words, the more the discounting. This would require that we maintain a lexicon of noise words.

Accounting for review quality in ranking entities. In this thesis, the quality of reviews used for ranking entities was not taken into consideration. However, in actuality the quality or validity of opinions within the reviews can change over time. Consider an example where a very dated review about a car mentions that the car was "very safe to drive". A few years down the road however, the car was recalled due to safety issues. To incorporate such information, we can encode known facts or issues using a 'prior' predicate that updates the opinions within the reviews. For example all instances of "very safe car" or "is a safe car" or anything equivalent can be updated to "not a safe car". This prior should be able to hold any number of facts and can be updated over time.

Another possibility is to discount the keyword matching contribution by taking into the temporal aspect of reviews. With this, the older reviews would have

a lower 'match' contribution (even if the keywords matched perfectly) compared to the newer reviews.

8.2.2 Abstractive Summarization of Opinions

Scaling up summarizers to work on Big Data. While the summarization approaches proposed in this thesis are known to work reasonably fast on a medium pool of review texts, it would be important to understand how well these approaches would scale up to much larger texts. For example, if we need to summarize the electronic health records of all patients diagnosed with diabetes to figure out the major complaints of these patients, then the size of the input text can get quite large depending on the sample size (e.g. all patients in the United States diagnosed with diabetes). It is possible to scale up these summarization systems using the map reduce framework and the challenge would be on how to distribute the summarization tasks and what gain can be expected in terms of speed of summarization.

Can the summarizers work seamlessly on different content types other than reviews? While the proposed summarizers have been shown to work well on review texts, It would also be beneficial to understand if the proposed summarization methods in this thesis would work with other types of texts such as Tweets and Facebook comments which are much shorter and noisier or news articles which are much more well formed. The goal is to see how much adaptation would be required to cater for other types of content other than reviews.

8.2.3 Opinion Acquisition

Is supervision necessary for opinion acquisition? The work on opinion acquisition in this thesis is unsupervised for the purpose of generality and scalability. Most existing focused crawlers however are supervised, requiring large amounts of training data for each topic. Since supervised approaches tend to be more accurate, it would be insightful to know if the performance of a supervised approach is comparable to that of OpinoFetch. If the performances are comparable, then there would be no reason to use supervision especially because we give importance to generality and domain independence.

Improving recall of collected opinions. One of the most critical (and difficult) problems in focused crawling is increasing the recall of relevant content. The same is true for the task of opinion acquisition where we want to collect a comprehensive set of opinions about an entity. Since in OpinoFetch, we use a general web search engine as starting point, we could potentially improve the recall of relevant review pages by exploring a lot more search results (more site

coverage) and also leverage the search results of multiple web search engines as well as social media search engines.

REFERENCES

[1] K. Ganesan and C. Zhai, "Opinion-based entity ranking," *Inf. Retr.*, vol. 15, no. 2, pp. 116–150, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10791-011-9174-8

[2] K. Ganesan, C. Zhai, and J. Han, "Opinosis: A graph based approach to abstractive summarization of highly redundant opinions," in *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, Beijing, China, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1873781.1873820 p. 340–348.

[3] K. Ganesan, C. Zhai, and E. Viegas, "Micropinion generation: an unsupervised approach to generating ultra-concise summaries of opinions," in *Proceedings of the 21st international conference on World Wide Web*, ser. WWW '12, 2012. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187954 pp. 869–878.

[4] K. Ganesan and C. Zhai, "Findilike: preference driven entity search," in *Proceedings of the 21st international conference companion on World Wide Web*, ser. WWW '12 Companion, 2012. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188045 pp. 345–348.

[5] K. Ganesan, "Opinosis Opinion Dataset," http://kavita-ganesan.com/opinosis-opinion-dataset, 2010.

[6] K. Ganesan, "OpinRank Review Dataset," http://kavita-ganesan.com/entity-ranking-data, 2010.

[7] K. Ganesan and C. Zhai, "Findilike: A preference driven entity search engine for evaluating entity retrieval and opinion summarization," in *ACM Conference on Information and Knowledge Management (CIKM '13)*, San Francisco, 2013.

[8] N. C. Msr, N. Craswell, and A. P. D. Vries, "Overview of the trec-2005 enterprise track," in *In The Fourteenth Text REtrieval Conf. Proc. (TREC*, 2005.

[9] I. Soboroff, A. P. de Vries, and N. Craswell, "Overview of the trec 2006 enterprise track," in *TREC*, 2006.

[10] P. Bailey, A. P. de Vries, N. Craswell, and I. Soboroff, "Overview of the trec 2007 enterprise track," in *TREC*, 2007.

[11] H. Shen, L. Wang, W. Bi, Y. Liu, and X. Cheng, "Research on enterprise track of trec 2008," in *TREC*, 2008.

[12] D. Petkova and W. B. Croft, "Proximity-based document representation for named entity retrieval," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*,

ser. CIKM '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1321440.1321542 pp. 731–740.

[13] K. Balog, L. Azzopardi, and M. de Rijke, "A language modeling framework for expert finding," *Inf. Process. Manage.*, vol. 45, pp. 1–19, January 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1460927.1461012

[14] C. Macdonald, D. Hannah, and I. Ounis, "High quality expertise evidence for expert search," in *Proceedings of the IR research, 30th European conference on Advances in information retrieval*, ser. ECIR'08. Berlin, Heidelberg: Springer-Verlag, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1793274.1793311 pp. 283–295.

[15] H. Fang and C. Zhai, "Probabilistic models for expert finding," in *ECIR*, 2007, pp. 418–430.

[16] B. He, C. Macdonald, J. He, and I. Ounis, "An effective statistical approach to blog post opinion retrieval," in *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*. New York, NY, USA: ACM, 2008, pp. 1063–1072.

[17] K. Yang, N. Yu, A. Valerio, H. Zhangm, and W. Ke, "Fusion approach to finding opinions in blogosphere," *ICWSM*, 2007. [Online]. Available: http://www.icwsm.org/papers/2--Yang-Yu-Valerio-Zhang-Ke-new.pdf

[18] K. Yang, N. Yu, and H. Zhang, "Widit in trec 2007 blog track: Combining lexicon-based methods to detect opinionated blogs," in *TREC*, 2007.

[19] D. Tunkelang, *Faceted Search*. Morgan and Claypool Publishers, 2009.

[20] J. Koren, Y. Zhang, and X. Liu, "Personalized interactive faceted search," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 477–486.

[21] H. Wang, Y. Lu, and C. Zhai, "Latent aspect rating analysis on review text data: a rating regression approach," in *KDD '10: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2010, pp. 783–792.

[22] Y. Lu, C. Zhai, and N. Sundaresan, "Rated aspect summarization of short comments," in *Proceedings of the 18th international conference on World wide web*. Madrid, Spain: ACM, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1526728# pp. 131–140.

[23] B. Snyder and R. Barzilay, "Multiple aspect ranking using the good grief algorithm," *In Proceeddings of HLT-NAACL '07*, pp. 300–307, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.4132

[24] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002, pp. 79–86.

[25] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the ACL*, 2005, pp. 115–124.

[26] P. D. Turney, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002. [Online]. Available: http://dx.doi.org/10.3115/1073083.1073153 pp. 417–424.

[27] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005. [Online]. Available: http://dx.doi.org/10.3115/1220575.1220619 pp. 347–354.

[28] I. Titov and R. Mcdonald, "A joint model of text and aspect ratings for sentiment summarization," in *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, June 2008. [Online]. Available: http://www.aclweb.org/anthology-new/P/P08/P08-1036.bib pp. 308–316.

[29] B. Liu, M. Hu, and J. Cheng, "Opinion observer: analyzing and comparing opinions on the web," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 342–351.

[30] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '04. New York, NY, USA: ACM, 2004. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014073 pp. 168–177.

[31] N. Gupta, G. Di Fabbrizio, and P. Haffner, "Capturing the stars: predicting ratings for service and product reviews," in *Proceedings of the NAACL HLT 2010 Workshop on Semantic Search*, ser. SS '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1867767.1867772 pp. 36–43.

[32] H. Wang, Y. Lu, and C. Zhai, "Latent aspect rating analysis on review text data: a rating regression approach," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '10. New York, NY, USA: ACM, 2010. [Online]. Available: http://0-doi.acm.org.millennium.lib.cyut.edu.tw/10.1145/1835804.1835903 pp. 783–792.

[33] H. D. Kim, K. Ganesan, P. Sondhi, and C. Zhai, "Comprehensive review of opinion summarization," 2011.

[34] S. R. K. Branavan, H. Chen, J. Eisenstein, and R. Barzilay, "Learning document-level semantic properties from free-text annotations," in *In Proceedings of ACL*, 2008, pp. 263–271.

[35] G. Carenini and J. C. K. Cheung, "Extractive vs. nlg-based abstractive summarization of evaluative text: the effect of corpus controversiality," in *Proceedings of the Fifth International Natural Language Generation Conference*, ser. INLG '08. Stroudsburg, PA, USA: Association for Computational Linguistics, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1708322.1708330 pp. 33–41.

[36] G. Carenini, R. Ng, and A. Pauls, "Multi-document summarization of evaluative text," in *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006, pp. 305–312.

[37] H. D. Kim and C. Zhai, "Generating comparative summaries of contradictory opinions in text," in *Proceeding of the 18th ACM conference on Information and knowledge management*, ser. CIKM '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1645953.1646004 pp. 385–394.

[38] M. J. Paul, C. Zhai, and R. Girju, "Summarizing contrastive viewpoints in opinionated text," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1870658.1870665 pp. 66–76.

[39] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques," in *Proceedings of EMNLP '02*, 2002, pp. 79–86.

[40] B. Snyder and R. Barzilay, "Multiple aspect ranking using the good grief algorithm," in *In Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL*, 2007, pp. 300–307.

[41] Y. Lu, C. Zhai, and N. Sundaresan, "Rated aspect summarization of short comments," in *Proceedings of the 18th international conference on World wide web*. Madrid, Spain: ACM, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1526728# pp. 131–140.

[42] K. Lerman, S. Blair-Goldensohn, and R. Mcdonald, "Sentiment summarization: Evaluating and learning user preferences," in *12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09)*, 2009.

[43] S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," in *Proceedings of the WWW '99*. New York, NY, USA: Elsevier North-Holland, Inc., 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=313234.313121 pp. 1623–1640.

[44] P. De Bra, G. Houben, Y. Kornatzky, and R. Post, "Information retrieval in distributed hypertexts," in *Proceedings of the 4th RIAO Conference*, 1994, pp. 481–491.

[45] M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur, "The shark-search algorithm. an application: tailored web site mapping," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 317–326, 1998.

[46] S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated focused crawling through online relevance feedback," in *Proceedings of WWW '02*, New York, NY, 2002. [Online]. Available: http://doi.acm.org/10.1145/511446.511466 pp. 148–159.

[47] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," in *Proceedings of the 26th International*

Conference on VLDB, ser. VLDB '00, San Francisco, 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=645926.671854 pp. 527–534.

[48] H. Chen, Y. Chung, M. Ramsey, and C. Yang, "A smart itsy bitsy spider for the web," *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, vol. 49, no. 7, pp. 604–618, 1998.

[49] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "A machine learning approach to building domain-specific search engines." Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999.

[50] J. Johnson, K. Tsioutsiouliklis, and C. L. Giles, "Evolving strategies for focused web crawling," in *ICML*, 2003, pp. 298–305.

[51] A. G. Vural, B. B. Cambazoglu, and P. Senkul, "Sentiment-focused web crawling," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, ser. CIKM '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2396761.2398564 pp. 2020–2024.

[52] G. Amati and C. J. van Rijsbergen, "Probabilistic models of information retrieval based on measuring the divergence from randomness," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 357–389, 2002.

[53] H. Fang, T. Tao, and C. Zhai, "A formal study of information retrieval heuristics," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2004. [Online]. Available: http://dx.doi.org/10.1145/1008992.1009004 pp. 49–56.

[54] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3," in *TREC*, 1994, pp. 0–.

[55] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to information retrieval," *ACM Trans. Inf. Syst.*, vol. 22, no. 2, pp. 179–214, 2004.

[56] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1998, pp. 275–281.

[57] J. Lafferty and C. Zhai, "Document language models, query models, and risk minimization for information retrieval," in *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2001, pp. 111–119.

[58] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma, "Terrier: A High Performance and Scalable Information Retrieval Platform," in *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[59] J. P. B. H. I. O. D. Hannah, C. Macdonald, "University of Glasgow at TREC2007: Experiments in Blog and Enterprise tracks with Terrier," in *Proceeddings of the 16th Text REtrieval Conference (TREC 2007)*, 2007.

[60] G. Salton and C. Buckley, "Improving retrieval performance by relevance feedback," pp. 355–364, 1997.

[61] B. Tan and F. Peng, "Unsupervised query segmentation using generative language models and wikipedia," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 347–356.

[62] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy, "Clustering query refinements by user intent," in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 841–850.

[63] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[64] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to ad hoc information retrieval," in *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2001, pp. 334–342.

[65] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://dx.doi.org/10.2307/3001968

[66] S. Robertson, "The Probabilistic Relevance Framework: BM25 and Beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009. [Online]. Available: http://scholar.google.de/scholar.bib?q=info:U4l9kCVIssAJ:scholar.google.com/&output=citation&hl=de&as_sdt=2000&as_vis=1&ct=citation&cd=1

[67] S. Siegel and Castellan, *Nonparametric statistics for the social sciences*. New York: McGraw-Hill, 1988.

[68] C. Zhai, *Statistical Language Models for Information Retrieval*. Morgan & Claypool, 2009.

[69] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *Proceedings of the ACL*, 2004, pp. 271–278.

[70] K. Lerman, S. Blair-Goldensohn, and R. Mcdonald, "Sentiment summarization: Evaluating and learning user preferences," in *12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09)*, 2009.

[71] D. Radev and K. McKeown, "Generating natural language summaries from multiple on-line sources," *Computational Linguistics*, vol. 24, no. 3, pp. 469–500, 1998. [Online]. Available: http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=972755

[72] S. H. Finley and S. M. Harabagiu, "Generating single and multi-document summaries with gistexter," in *Proceedings of the workshop on automatic summarization*, 2002, pp. 30–38.

[73] G. F. DeJong, "An overview of the FRUMP system," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1982, pp. 149–176.

[74] H. Saggion and G. Lapalme, "Generating indicative-informative summaries with sumum." *Computational Linguistics*, vol. 28, no. 4, pp. 497–526, 2002. [Online]. Available: http://dblp.uni-trier.de/db/journals/coling/coling28.html#SaggionL02

[75] H. Jing and K. R. McKeown, "Cut and paste based text summarization," in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 178–185.

[76] G. Erkan and D. R. Radev, "Lexrank: graph-based lexical centrality as salience in text summarization," *J. Artif. Int. Res.*, vol. 22, no. 1, pp. 457–479, 2004.

[77] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*, July 2004.

[78] R. Barzilay and L. Lee, "Learning to paraphrase: an unsupervised approach using multiple-sequence alignment," in *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*. Morristown, NJ, USA: Association for Computational Linguistics, 2003, pp. 16–23.

[79] D. R. Radev, E. Hovy, and K. McKeown, "Introduction to the special issue on summarization," 2002.

[80] C.-Y. Lin, "Rouge: a package for automatic evaluation of summaries," in *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain*, 2004.

[81] C.-Y. Lin and E. Hovy, "Automatic evaluation of summaries using n-gram co-occurrence statistics," in *Proc. HLT-NAACL*, 2003. [Online]. Available: http://research.microsoft.com/~cyl/download/papers/NAACL2003.pdf p. 8 pages.

[82] C.-Y. LIN, "Looking for a few good metrics : Rouge and its evaluation," *proc. of the 4th NTCIR Workshops, 2004*, 2004. [Online]. Available: http://ci.nii.ac.jp/naid/10020677642/en/

[83] D. Radev, H. Jing, and M. Budzikowska, "Centroid-based summarization of multiple documents: Sentence extraction, utility-based evaluation, and user studies," in *In ANLP/NAACL Workshop on Summarization*, 2000, pp. 21–29.

[84] B. Liu, *Web data mining; Exploring hyperlinks, contents, and usage data.* Springer, 2006.

[85] R. Real and J. M. Vargas, "The Probabilistic Basis of Jaccard's Index of Similarity," *Systematic Biology*, vol. 45, no. 3, pp. 380–385, 1996. [Online]. Available: http://dx.doi.org/10.2307/2413572

[86] E. Terra and C. L. A. Clarke, "Frequency estimates for statistical word similarity measures," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003. [Online]. Available: http://dx.doi.org/10.3115/1073445.1073477 pp. 165–172.

[87] K. Wang, C. Thrasher, E. Viegas, X. Li, and B.-j. P. Hsu, "An overview of microsoft web n-gram corpus and applications," in *Proceedings of the NAACL HLT 2010 Demonstration Session*. Los Angeles, California: Association for Computational Linguistics, June 2010. [Online]. Available: http://www.aclweb.org/anthology/N10-2012 pp. 45–48.

[88] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, pp. 1424–1440, November 2004. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2004.77

[89] H. T. Dang, "Overview of DUC 2005," in *Document Understanding Conference*, 2005.

[90] K. Filippova, "Multi-sentence compression: finding shortest paths in word graphs," in *Proceedings of the 23rd International Conference on Computational Linguistics*, ser. COLING '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1873781.1873818 pp. 322–330.

[91] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning, "Kea: practical automatic keyphrase extraction," in *Proceedings of the fourth ACM conference on Digital libraries*, ser. DL '99. New York, NY, USA: ACM, 1999. [Online]. Available: http://doi.acm.org/10.1145/313238.313437 pp. 254–255.

[92] B. Novak, "A survey of focused web crawling algorithms," SKIDD, 2004.

[93] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *KDD*, 2004, pp. 168–177.

[94] A. P. Vries, A.-M. Vercoustre, J. A. Thom, N. Craswell, and M. Lalmas, "Focused access to xml documents," N. Fuhr, J. Kamps, M. Lalmas, and A. Trotman, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Overview of the INEX 2007 Entity Ranking Track, pp. 245–251. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85902-4_22

[95] J. X. Yu, L. Qin, and L. Chang, *Keyword Search in Databases*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.