Design of Web Mining Techniques to Answer Quantity Queries and to Map Documents to Topics

THESIS

Submitted in partial fulfilment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Somnath Banerjee
(Roll No. 2005PHXF041P)

Under the Supervision of
Dr. Krishnan Ramanathan

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
(RAJASTHAN) INDIA
2009

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
(RAJASTHAN)

CERTIFICATE

This is to certify that the thesis entitled **"Design of Web Mining Techniques to Answer Quantity Queries and to Map Documents to Topics"** and submitted by Somnath Banerjee, ID No. 2005PHXF041P for award of Ph. D. Degree of the Institute, embodies original work done by him under my supervision.

Signature in full of the Supervisor: [illegible]
Name in capital block letters: KRISHNAN RAMANATHAN
Designation: SENIOR RESEARCH SCIENTIST
HP LABS

Date: 7th December, 2009

Acknowledgments

I am immensely thankful to Prof. L. K. Maheshwari, Vice-Chencellor, BITS, Pilani for providing me this opportunity to pursue the off-campus PhD of the Institute. I express my gratitude to Prof. Ravi Prakash, Dean, Research and Consultancy Division (RCD), BITS, Pilani for his constant official support, encouragement and making the organization of my research work through the past few years easy.

I thank Dr. Hemanth Jadav, Mr. Dinesh Kumar, Ms. Monica Sharma, Mr. Sharad Shrivastava, Mr. Gunjan Soni, Mr. Amit Kumar and Ms. Sunita Bansal, nucleus members of RCD, BITS, Pilani, without whose cooperation and guidance it would not have been possible for me to pursue such goal oriented research during each of the past few semesters.

I also express my gratitude to the office staff of RCD whose secretarial assistance helped me in submitting the various evaluation documents in time and give pre-submission seminar smoothly.

I thank my Doctoral Advisory Committee (DAC) members, Prof. Navneet Goyal and Prof. Shan Balasubramaniam, who spared their valuable time to go through my draft thesis and were audience to my pre-submission seminar in order to provide several valuable suggestions that immensely helped in improving the quality of my PhD thesis report.

I am deeply indebted to Prof. Soumen Chakrabarti of IIT Bombay for giving me the opportunity to work with him. The time I spent at IIT Bombay was the best part of my PhD.

I am grateful to my advisor, Krishnan Ramanathan for his valuable guidance and support during my studies. I am thankful to Ajay Gupta for providing tremendous encouragement, support and numerous opportunities throughout my PhD. I am grateful to Anjaneyulu Kuchibhotla for providing guidance, encouragement and sorting out various issues during my PhD. I am thankful to Geetha Manjunath and Sitaram Ramachandrula for helping me in writing the thesis and for all the discussions we had.

Throughout my doctoral studies, I benefited from numerous discussions with my friends and colleagues. I am particularly thankful to Matrin, Sayali, Amit, Avinava and Jinesh for their time and assistance.

Finally I am thankful to my friends: Chirantan, Surajit, Sanjib and Samiran. Without them my Bangalore days would not have been such fun filled and memorable.

Date: 7th December, 2009.

Somnath Banerjee

Somnath Banerjee

To my grandparents,

Smt. Kiran Bala Devi and Sri Kalipada Banerjee

Abstract

There is an explosion of electronic information on the Internet and elsewhere. Intelligent information processing tools, e.g. automated question answering, text classification, text clustering, etc., are therefore imperative to be able to utilize this overwhelming amount of information effectively. An information source, in turn, plays varied and important roles in building those tools. For example, an automated question answering system needs an information source to answer questions. Similarly, a text classification or clustering system can perform better if an information source can provide the context in which various terms, concepts and entities are generally used.

The World Wide Web is the largest source of information and also provides an up-to-date view of the world. But information on the web is mostly unstructured or at best semi-structured since most information is in the form of web pages. The enormous size, diversity and ever evolving nature of the web also pose significant challenges for a system to use it as an information source. In this thesis, we focus on techniques which use the information on the web for answering quantity queries and carrying out text classification and clustering.

Quantity queries are a special class of queries where the expected answers are numeric values. Some examples include queries for determining *the price of a product*, *the driving time between cities*, *the battery life of a laptop*, etc. Quantity queries are known to be an important class of queries as evident from their presence in any search engine's query log. Answers for most quantity queries are likely to be available in the web but in most cases the answer may only be available in unstructured form. There could also be noisy data in the web that could lead to incorrect answers. Therefore determining the correct answer to quantity queries from the web is a challenging task. In our work we present two novel algorithms to answer quantity queries by aggregating evidences from multiple promising answer candidates. Our algorithms achieve around 20% higher accuracy in answering such queries as compared to the best-known approaches.

The other part of our work focuses on using several web resources for text classification and clustering tasks. Text classification and clustering are important tools for any large scale textual information processing system. In both text classification and clustering, the "topics" of documents provides valuable inputs to the system. The topic information can be used to improve the accuracy of classification/clustering algorithms as well as to automatically build a classification model for documents.

We develop methods to map documents to topics using the information available on the web. The goal is both to generate *topic features* of the documents and to get a sample set of documents for a given *topic*. The topic features of a document give an indication about the topics covered in the document. Since the topic features are obtained from a large web resource they provide information beyond what is available in the documents. We demonstrate that the use of topic features improves accuracy of text classification/clustering algorithms by more than 30%. The sample set of documents for a given topic can be used to construct training examples to build a classifier for that topic. A classifier for a topic can discriminate between the documents that belong to the topic and those which do not. Constructing training data is also a major bottleneck in building a classifier. We show that several web sources can be used to automatically obtain sample set of documents for a topic. These documents can then be used to construct training examples to build a highly accurate classifier for that topic.

Contents

List of Tables

List of Figures

List of Abbreviations

BOW	Bag of Words
IR	Information Retrieval
LDA	Latent Dirichlet Allocation
NLP	Natural Language Processing
QA	Question Answering
QCQ	Quantity Consensus Query
SVM	Support Vector Machine
TC	Text Classification
web	World Wide Web

Chapter 1

Introduction

Access to information is an important ingredient for the advancement of human society. There is a wealth of information available on the Internet and on other electronic media sources. It requires the usage of intelligent information processing tools to access and consume this overwhelming amount of information. “Search engine”, “automated question answering (QA)”, “text classification”, “text clustering” are some of the very important tools to consume and manage a large volume of information. Without such tools it is impossible to find and consume information from the Internet or other such large media sources.

Such information processing tools in turn either require or can perform better by having access to a large information source. For example, an automated QA system uses an information source to answer questions. The size of the information source is the asymptotic limit of the number or variety of questions the QA system can answer. If a QA system can use a large information source it can possibly answer wider variety of questions. Information management tools, e.g. text classification and clustering, have to deal with various terms, entities or concepts discussed in natural language texts. The basic goal of these tools are to organize documents based on whether they contain same or similar terms, entities or concepts. Two terms, entities or concepts are considered as same if they are often used in similar contexts in a natural language. A large information source that can provide such context information can potentially help the classification/clustering algorithm to perform better.

The World Wide Web is an enormous information source. A significant portion of collective knowledge of mankind is now available on the web. As of November, 2009, there are more than 20 billion web pages that are publicly accessible [UrlSizeOfWeb]. These billions of pages contain a huge volume of information on a large variety of topics, entities, facts, etc.

Most of the information available on the web are in unstructured form and as a collection of web pages. In recent times, users' participation in web content creation has increased dramatically, thanks to the growth of Web 2.0 applications. More then half of the content available on the web today is user generated (e.g. personal web pages, blogs, reviews, comments, profiles and messages in social network sites etc) [Ramakrishnan and Tomkins, 2007]. Such user generated contents are also a very valuable source of information. Web 2.0 applications and user participation have also given birth to little more structured and specialized sources for information. The most prominent example of such source is Wikipedia [UrlWikipedia], the gigantic and free online encyclopedia. Other examples are web directory (e.g. Dmoz [UrlDmoz]) that organizes millions of web pages under a large and hierarchical taxonomy and social tagging sites (e.g. Delicious [UrlDelicious]) where users add meta-data to the web contents. In summary, the collection of web pages and different web 2.0 sources (e.g. Wikipedia, Dmoz, Delicious, etc.) constitute a tremendous source of information. The dynamic and ever growing nature of the web keeps this information refreshed and provides an up-to-date view of the world.

As mentioned before many of the information processing tools can become more useful or effective by using the gigantic web as an information source. Consider the following three example tasks which are to be performed by automated information processing tools.

Task 1 *To respond to queries like "driving time" (between two cities), "product price", "product attributes" (e.g. battery life, screen resolution, etc.), "population of a city", "travel cost" etc.*

Task 2 *Associate a piece of text like "Steve Jobs Video Dreams" with the concept "Apple Inc"*

Task 3 *To filter or retain only pertinent content from incoming news, blogs or other kinds of web pages that belong to any given arbitrary topic (e.g. "German Car")*

The Task 1 is to answer factoid questions about various entities, ranging from driving time, attributes related to product, travel, city, etc. It might be easier and more accurate for an automated QA system to use structured or custom information sources to answer questions. But any structured or custom build source is unlikely to be large and scalable enough to contain answers for wide variety of questions targeted in the Task 1. For example, one could use product inventory databases of some dealers to (easily) answer queries related to product attributes. But that restricts the system to the product domain and to the information provided by only those dealers. The web on the other hand, being an enormous information source, is most likely to contain answers for all such questions. Therefore, a QA system can become more useful if it can answer questions from the web.

In the Task 2, a machine has to identify whether the piece of text "Steve Jobs Video Dreams" belongs to the category "Apple Inc" or not. This is an instance of text classification where the task is to identify the class or category of a given text document. Note that in the input text the terms "Apple" or "Inc" do not appear. The contexts in which the specific terms of the text are generally used can provide such information. For example, if we can figure out that "Steve Jobs" often appears in the context of "Apple Inc" then it might be easier to classify the given piece of text. In many text classification or clustering tasks such need for context information arises. The web is an excellent source to get the context information about any concept or entity. A text classification/clustering algorithm can possibly achieve higher accuracy by making use of such information obtained from the web.

The goal in the Task 3 is to automatically build a classifier for any given arbitrary concept (e.g. *German Car*). Building a text classifier requires training data. Training data in this example will constitute of some sample pages about *German Car* and some sample pages that are not about *German Car*. The process of obtaining training data for a classifier is often manual and expensive. In this case, the web is an excellent source to automatically get sample pages (training examples) for any given concept. Many web resources, e.g. web search engines, Wikipedia, web directory, can be utilized to get sample pages for any given concept. Those sample pages then can be used to construct training examples to build a classifier.

From the above discussion we can see that it is immensely valuable to use the web as the information source in many information processing tasks. Enabling machines to make use of the enormous information available on the web is immensely potential and challenging research topic. This topic is increasingly gaining popularity among various research communities working in the field of information retrieval, machine learning and natural language processing. Apart from question answering, text classification and text clustering, in various other areas also web resources are being extensively used. Web search [Milne et al., 2007], named entity recognition [Cucerzan, 2007], statistical language translation [Zhang and Vines, 2004, Huang et al., 2005], image categorization [Fergus et al., 2005], etc, are examples of such areas.

1.1 Some Challenges in using the Web as an Information Source

Utilizing the web as an information source posses multiple challenges.

1. *Information available on the web is mostly unstructured or at best semi-structured.*

   Information on the web is mostly available as natural language texts. Answering queries from natural language texts is a challenging task. Fixing the domain of the query or the web sources to be used can reduce the complexity of the problem. But that limits the application domain and also the available information. For example, many product search engines (e.g. [UrlGoogleProduct]) find and populate product information (e.g. "price", "battery life", "screen resolution" etc.) from a fixed set of dealer websites (e.g. amazon.com, ebay.com etc.). Such systems are explicitly designed to work only in the product domain. Also, generally they rely on the structure information (e.g. HTML tag or web service API) and therefore cannot utilize the wealth of information available in the user generated contents (e.g. product reviews, blogs etc).

   Because of this unstructured nature of the web it is not easy to get the context information about the concepts described in document (Task 2). Similarly it is also difficult to obtain a representative sample of pages from the web for any given concept.

2. *Information on the web is noisy and could be even inaccurate.*

   There are many web pages mentioning different (and could be even misleading) driving times between two given cities say "Paris" and "Nice". "Steve Jobs" can possibly refer to different persons and therefore can appear in many different contexts in the web. There is no reliable source to obtain good quality sample pages belonging to an arbitrary topic like *German Car.* Web directory (e.g. Dmoz [UrlDmoz] or Yahoo directory [UrlYahooDirectory]) could be a source for obtaining pages belonging to a given concept or topic. But as we will see in this thesis that even a web directory is also noisy and may not be able to provide good representative sample set of pages.

3. *Web is enormous and ever evolving.*

   Everyday almost 12-15 gigabytes of contents are being added to the web [Ramakrishnan and Tomkins, 2007]. Thousands of websites come alive or go offline everyday [UrlDomainCounts]. Even just in Wikipedia, more than 1300 pages get added per day [UrlSizeOfWikipedia]. It is infeasible to build a fixed set of rules to mine an information source that changes so rapidly. For example, a system designed to obtain product information by parsing the HTML structure of amazon.com is likely to become obsolete or requires substantial maintenance effort. Even the huge sources like web directory is gradually becoming outdated. Therefore, an algorithm designed to seek information from a particular source or depends on structural schema of sources can become stale over time.

Therefore, to use the information available on the web, a robust and scalable technique should have the following desirable properties. (a) It should use the vast natural language texts available on the web rather restricting itself to a few structured sources of information. (b) No site specific schema information should be used as the schema keeps changing. (c) The technique has to be statistical rather than using hard coded rules in any form. (d) The technique should focus more on getting aggregated information rather trying to deploy sophisticated methods to directly retrieve specific piece of information.

In this thesis, we develop techniques to use the unstructured and noisy information available on the web in several textual information processing tasks. In particular, we explore the use of the web information source in three different tasks, answering quantity queries, text classification and text clustering. As we will discuss shortly that these are very important tasks in the area of information retrieval and are building block of numerous applications. The web information source served different utility in these tasks. To answer quantity queries, the web information source is used to collect evidences. For text classification/clustering, the web sources are used to find mappings of documents to topics. Mapping of documents to topics are done to get *topic features* of the documents and to get sample documents for a given *topic*. The *topic features* helps in improving the accuracy of test classification/clustering algorithms and the sample documents of a *topic* can be used to construct training data to build a classifier for that topic. Broadly the problems addressed in the thesis can be divided into the following three sub-problems.

1. Answering quantity related queries (e.g. "driving time", "price", "battery life", etc.) from the web

2. Generating topic features of documents from a web corpus. We demonstrate that these topic features of documents help in improving text classification/clustering accuracy.

3. Gathering training data from web sources for any given topic. This enables the possibility of automatically building text classifier for an arbitrary topic.

In the remainder of this chapter we will discuss each of these sub-problems, the specific challenges and our contribution in that sub-problem.

1.2 Specific Sub-problems Addressed in the Thesis

1.2.1 Answering Quantity Consensus Queries (QCQs)

The first sub-problem we address in the thesis is answering quantity queries. A quantity may be a unit-less number or may have an associated unit like length, mass, weight, temperature, etc. As against "spot queries" seeking unique answers like date of birth, etc, in this thesis we are specifically interested in what we call *quantity consensus queries* (QCQs), where there is an uncertainty about the answer quantity. Examples of QCQ are, *"driving time from Paris to Nice"*, *"battery life of Lenovo X300"*, *"price of Canon Powershot SD700 IS"*, *"population density of Mumbai"*, etc.

A QCQ system that can answer quantity queries from the web has many important use cases. Firstly, quantity queries are frequent in any search engine query log and also they are commercially important. Therefore, a QCQ system can add value over standard web search by collecting consensus information from hundreds to thousands of pages, something that would take a user from minutes to hours. Secondly, a QCQ system can also support web sites that offer comparison of prices and features related to products, services and travel.

Answering questions from the huge web corpus requires a different approach than answering questions from a small corpus [Brill et al., 2002]. As we discussed earlier, restricting the sources to be utilized only limits the scope and domain of the application. The scale and noisy data of the web also often prohibits using relatively expensive and rigid NLP techniques. Therefore, QA from web generally focuses on exploiting the redundancy of information on the web than using sophisticated techniques to directly retrieve the correct answers. Answer candidates are retrieved using simple but noisy information retrieval (IR) techniques. Then the most promising answer is detected from the answer candidate set assuming that the correct answer will appear many times in the candidate set (due to the redundancy of information in web).

Formally, a standard factoid QA (or entity search or entity ranking) system from the web largely proceeds along the following line [Cheng et al., 2007, Ko et al., 2007]: Given a query, first a set of *snippets* (defined in the next paragraph) that are likely to contain the answer are

In the remainder of this chapter we will discuss each of these sub-problems, the specific challenges and our contribution in that sub-problem.

1.2 Specific Sub-problems Addressed in the Thesis

1.2.1 Answering Quantity Consensus Queries (QCQs)

The first sub-problem we address in the thesis is answering quantity queries. A quantity may be a unit-less number or may have an associated unit like length, mass, weight, temperature, etc. As against "spot queries" seeking unique answers like date of birth, etc, in this thesis we are specifically interested in what we call *quantity consensus queries* (QCQs), where there is an uncertainty about the answer quantity. Examples of QCQ are, *"driving time from Paris to Nice"*, *"battery life of Lenovo X300"*, *"price of Canon Powershot SD700 IS"*, *"population density of Mumbai"*, etc.

A QCQ system that can answer quantity queries from the web has many important use cases. Firstly, quantity queries are frequent in any search engine query log and also they are commercially important. Therefore, a QCQ system can add value over standard web search by collecting consensus information from hundreds to thousands of pages, something that would take a user from minutes to hours. Secondly, a QCQ system can also support web sites that offer comparison of prices and features related to products, services and travel.

Answering questions from the huge web corpus requires a different approach than answering questions from a small corpus [Brill et al., 2002]. As we discussed earlier, restricting the sources to be utilized only limits the scope and domain of the application. The scale and noisy data of the web also often prohibits using relatively expensive and rigid NLP techniques. Therefore, QA from web generally focuses on exploiting the redundancy of information on the web than using sophisticated techniques to directly retrieve the correct answers. Answer candidates are retrieved using simple but noisy information retrieval (IR) techniques. Then the most promising answer is detected from the answer candidate set assuming that the correct answer will appear many times in the candidate set (due to the redundancy of information in web).

Formally, a standard factoid QA (or entity search or entity ranking) system from the web largely proceeds along the following line [Cheng et al., 2007, Ko et al., 2007]: Given a query, first a set of *snippets* (defined in the next paragraph) that are likely to contain the answer are retrieved from the web. The snippets are then represented using a feature vector and scored using a learned model. Figure 1.1 shows the steps involved in collecting and scoring snippets for a query. Individual snippet scores obtained at this stage are often not sufficient to identify the relevant snippets. This is mostly because the snippets are retrieved using simple IR techniques and the snippet features are very noisy indicator of relevance. Therefore one more step is generally deployed to exploit the redundancy of information on the web. Given the redundancy property of the web, the correct answer is likely to be repeated across multiple snippets. To exploit this property, the scores of the snippets containing same or similar answers are then aggregated. The final score of an answer is this aggregated score and the top K answers are presented to the user.

A snippet is a fragment of text that contains some query keywords and an entity of the expected answer type for the given query. Some example snippets are shown in the Figure 1.1. The text fragment could be defined as a sentence, paragraph or a window of tokens. The expected answer type is the entity type of the correct answer. For factoid questions, the expected answer type could be "person", "organization" or "location" etc., for QCQ, the expected answer type is a quantity type (e.g. "height", "weight" or "USD" etc). Depending on the interface, the expected answer type could be determined by analyzing the question or may be provided by the user. Given this definition of the snippet, the snippets for a query can be retrieved by simply matching the query keywords and the expected answer type to the text fragments in the corpus. Thus existing techniques of fast retrieval from inverted indexes can be directly applied.

The snippet features reflect how good the query matches to the text of the snippet, how far the query keywords are from the candidate entity in the snippet etc. We will discuss the details of the feature design of a snippet in the Section 3.3.2. A learned model is used to score each snippet independently. But the individual snippet scores are generally not sufficient to identify the relevant snippets (snippets containing correct answer entity). Therefore, the scores of the snippets containing same entity are then accumulated. That is, a snippet votes its score for the entity it contains. The entities are then ranked using this accumulated score. Instead of just aggregating the scores of the snippets containing exactly same entity some syntactic variation also generally taken into account. For example, for the question "who is the prime minister of India?", two snippets containing entities "Dr. Singh" and "Manmohan Singh" respectively can vote for the same entity "Dr. Manmohan Singh". This form of score aggregation is basically weighted voting.

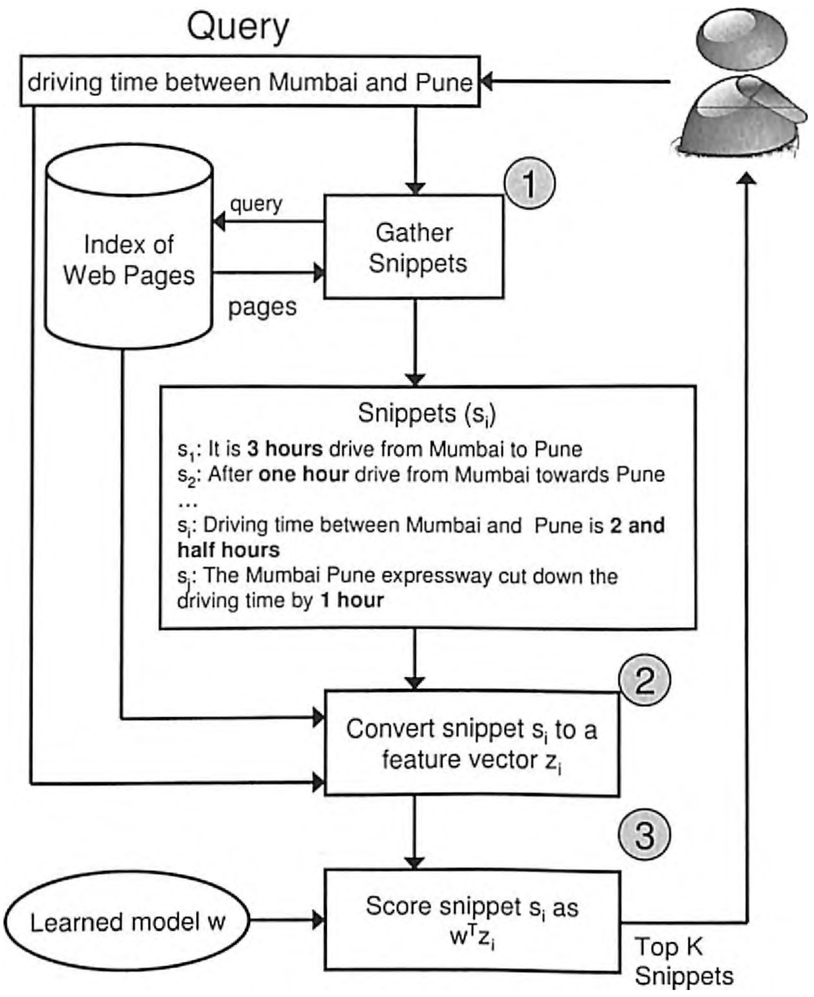

Query
driving time between Mumbai and Pune
1
query
Index of Web Pages
Gather Snippets
pages
Snippets (s_i)
s_1: It is **3 hours** drive from Mumbai to Pune
s_2: After **one hour** drive from Mumbai towards Pune
...
s_i: Driving time between Mumbai and Pune is **2 and half hours**
s_j: The Mumbai Pune expressway cut down the driving time by **1 hour**
2
Convert snippet s_i to a feature vector z_i
3
Learned model w
Score snippet s_i as $w^T z_i$
Top K Snippets

Figure 1.1: Snippet generation and scoring in standard entity rank system. The numbers in the circles show the step sequence. Often the scores of the snippets containing same or similar answer are aggregated and then the answers are ranked based on this score. For clarity this step is not shown in the diagram

The weighted voting score aggregation is inadequate to answer QCQs from the web. Weighted voting, when adopted for QCQ, can only be defined for each distinct quantity. But in QCQ, the snippets containing nearby quantities should reinforce each other. For example, consider the QCQ "price of Canon Powershot SD700 IS". Each retrieved snippet will have a quantity of type "USD" along with some query keywords (e.g. "price", "canon" etc.). It is inadequate to aggregate the scores of distinct quantities. For a QCQ, the correct answer is not a single quantity but rather a distribution over the quantities. The correct price of Canon Powershot could vary from 375 - 400 USD (say). A snippet containing 375 USD as the answer should also vote for the quantities (e.g. 400 USD or 350 USD) near it. Therefore for QCQ, scoring each snippet independently is not going to work and also just taking weighted vote for each distinct quantity is not of much help. Rather we need to design a corroborative and collective scoring mechanism that will score a set of snippets with near by quantities.

Our Contribution

In this work, our first contribution is that we introduce the notion of quantity consensus query (QCQ). We also propose two novel algorithms that can respond to QCQs using the web as the information source. We demonstrate that it is inadequate to adopt a standard entity rank system and score individual snippets or even to aggregate the scores of the distinct quantities. We argue that scoring and ranking mechanism should be corroborative and collective so that the snippets containing near by quantities are scored together. Towards this, we propose novel algorithms that directly score and rank quantity intervals (e.g. 300-400 is a quantity interval for the above QCQ about Canon camera). The quantity intervals are scored using the features and the candidate quantities of the supporting snippets (snippets whose quantities are lying inside the interval). An example output (quantity intervals and supporting snippets) of the QCQ algorithms is shown in the Figure 1.2. The proposed algorithms yield about 20% better accuracy compared to the best known collective ranking algorithms, and are 35% better than scoring snippets independent of each other.

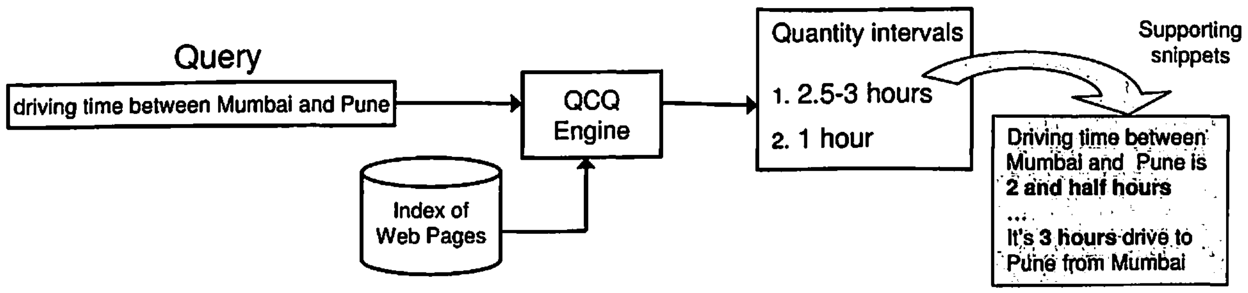

Query
driving time between Mumbai and Pune
QCQ Engine
Index of Web Pages
Quantity intervals
1. 2.5-3 hours
2. 1 hour
Supporting snippets
Driving time between Mumbai and Pune is 2 and half hours
...
It's 3 hours drive to Pune from Mumbai

Figure 1.2: Output of a QCQ engine is a ranked list of quantity intervals and supporting snippets for each interval.

Apart from demonstrating the effectiveness of the proposed algorithms using a set of sample queries and sample snippets, we have developed a fully functional QCQ system that uses a web scale corpus of 500 million pages as the information source. The developed system addresses many scalability issues that a real QCQ system will encounter.

We move ahead further and show one more use of a web information resource in answering QCQs. We use Wikipedia [UrlWikipedia] to disambiguate and accurately identify the entities mentioned in the corpus. The entities mentioned in the 500 million pages are mapped to Wikipedia entities using a recent technique [Kulkarni et al., 2009]. This mapping resolves the ambiguity in the entities mentioned in a natural language text. For example, if the entity "Steve Jobs" in a document refers to the Apple CEO then only this entity will get mapped to the Wikipedia page "Steve Jobs" about Apple CEO. Other "Steve Jobs" will not get mapped to the Wikipedia page about the Apple CEO. So a user can express an unambiguous query (say "Steve Jobs net worth") by pointing to the appropriate Wikipedia entity for "Steve Jobs". Another example is, the mention of entities "HP" and "Hewlett Packard" both will get mapped to the Wikipedia page about the company "Hewlett Packard". So to respond to the QCQ "Hewlett Packard revenue", the snippets containing the entity "HP" (and a quantity of type USD) will also be considered. Note that entity "HP" can refer to the company "Hindustan Petroleum" as well. The mapping ensures that "HP" gets mapped to the Wikipedia entity "Hewlett Packard" only when it is actually referring to that.

1.2.2 Generating Topic Features from a Web Corpus

In this section, we will introduce the second sub-problem addressed in the thesis. Here we develop methods to obtain topic features of documents of a text classification/clustering task from a web corpus. Text classification and clustering are used in wide variety of applications, e.g. news classification/ clustering, spam filtering, clustering search results based on topics etc. Text classification is the task of identifying the class or category of a text document. Text clustering aims to discover natural grouping within a collection of text documents. We demonstrate that the topic features improves the accuracy of text classification and clustering algorithms.

To apply a text classification or clustering algorithms on a set of documents, generally the documents need to be represented in a m dimensional feature space or vector space ($\mathbb{R}^m$). One conventional method of representing documents in a feature space is using *bag of words* technique where each term in the vocabulary is considered as an independent feature. The vocabulary here is the union of all terms appearing in the document collection of the classification/clustering task. In the feature vector representation of a document, a feature gets a non-zero weight if the corresponding term appears in the document. Various weighting schemes have been defined to weigh the features [Salton and McGill, 1986]. The weight of a feature generally captures the frequency of the corresponding term in the document and also the overall importance of the term.

There are several shortcomings of this method of document representation. Firstly, an individual term can be ambiguous (*apple* computer vs *apple* tree) and multiple terms can have the same meaning (synonyms). Secondly, the bag of words approach uses only those pieces of information (terms) that are explicitly mentioned in the documents. A document is often written assuming some background knowledge of the readers (e.g. the piece of news text "Steve Jobs Video Dream" assumes readers' familiarity with the Apple CEO). Document collection used in a classification or clustering task is often small and therefore not sufficient to provide the background knowledge or contexts of all the concepts, entities or terms described in the documents.

Various extensions of bag of words approach have been proposed in the past. This include augmenting (or replacing) the bag of words representation with phrases or n-grams [Lewis, 1992, Bekkerman, 2004], linguistically motivated features e.g. part-of-speech [Sable et al., 2002] etc. More recent works use some external resources to bring in information that is not available in a small document collection. The external resource is generally used to obtain additional features of the documents. Since the additional features are obtained from a much larger (external) resource they provide information that are not readily available in a document. One line of work in this direction uses a dictionary or thesaurus to add synonyms or hypernyms (terms with is-a relation) of terms as features. For example, Scott and Matwin [1999] used synonyms and hypernyms obtained from the largest online thesaurus WordNet [Fellbaum, 1998] as additional features. However, using a manually built thesaurus has scaling issue, e.g. there is no entry for "Steve Jobs" in the WordNet. And ideally we want the features to capture the context in which the concepts, entities or terms of the documents are generally used. For example, for the text "Steve Jobs Video Dream", "Apple Inc" and "Apple CEO" are better features (to determine that the text is related to *Apple Inc*) than feature like "visual communication" that can be obtained from the WordNet hypernym hierarchy of the term "video".

Web is a tremendous information source to obtain context information for any concept, entity or term described in a document. Therefore, recent research is more oriented towards obtaining features from web resources [Gabrilovich and Markovitch, 2006, 2005]. One prominent work in that direction is feature generation from Wikipedia by Gabrilovich and Markovitch [2006]. We give the overview of this method in the contribution subsection (next) as we have used a very similar method to generate topic features from Wikipedia.

Our Contribution

In this work, we develop a method to obtain topic features from a web corpus and demonstrate that topic features improve the accuracy of text classification and clustering algorithms. A topic feature corresponds to a topic from a predetermined set. We will provide formal definition of a topic later. Broadly a topic is a concept like "Apple Inc", "Sports" etc. A topic feature of a document indicates how much the document is aligned to (talking about) the corresponding topic. The set of topics is obtained from a large web corpus. Therefore the topic features of a document provide information that cannot be obtained from that document. For example, the news text "Steve Jobs Video Dreams" is likely to have high value for a topic like "Apple Inc" although that information is not available in the document. The topic features basically capture the contexts in which the terms of the documents generally appear and therefore provide valuable inputs to a classification/clustering algorithm.

To generate topic features we first use Wikipedia and adopt a simple method following Gabrilovich and Markovitch [2006]. We show that such Wikipedia (topic) features help in two different tasks; "short text clustering" and "inductive transfer". Although such Wikipedia features are helpful in improving the accuracy of various tasks, we observe some shortcomings in this method of feature generation. We then propose a more principled approach of feature generation that addresses some of those shortcomings.

Generating Topic Features from Wikipedia In this approach, a topic is a title of a Wikipedia article. A topic feature of a document broadly indicates whether the document is related to the corresponding Wikipedia article or not. For a given document, we obtain these topic features (also referred as Wikipedia features) using a very simple technique. We first built an inverted index of the Wikipedia dump. Then a set of top matching Wikipedia articles are retrieved from that inverted index using the text of the given document as a query. The titles of the top matching Wikipedia articles are considered as the topic features of the document. For example, some topic features, that are obtained from Wikipedia, for the news text "Steve Jobs Video Dreams" are "Steve Jobs", "Apple Inc" etc. Figure 1.3 shows the block diagram of the process of generating topic features from Wikipedia. We demonstrate that when these topic features are used in addition to the bag of words features it helps in improving the accuracy of two different tasks; i) short text clustering [Banerjee et al., 2007] and ii) inductive transfer for text classification [Banerjee, 2007]

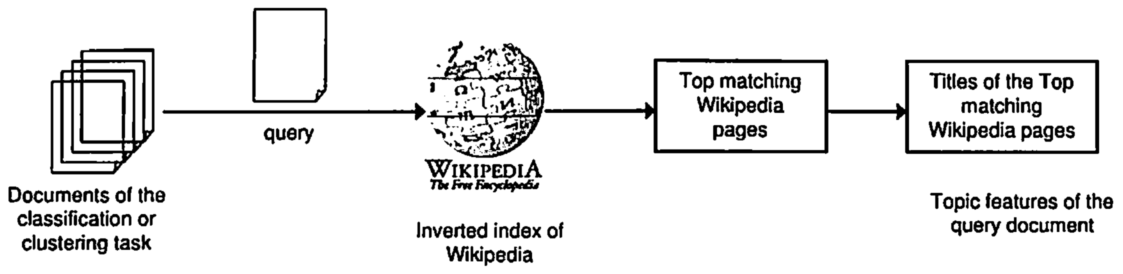

query
Documents of the classification or clustering task
WIKIPEDIA
The Free Encyclopedia
Inverted index of Wikipedia
Top matching Wikipedia pages
Titles of the Top matching Wikipedia pages
Topic features of the query document

Figure 1.3: The steps of topic feature generation from Wikipedia

Although the Wikipedia features help in improving the accuracy of different tasks, this method has shortcomings in terms of its ad-hoc nature and performance issues. For example, what ranking function to use to retrieve Wikipedia articles from the inverted index, how many retrieved Wikipedia articles to be used as the topic features and how the method behaves as Wikipedia evolves? There are no possible guideline to address these questions. Also, since Wikipedia has more than 3 million pages, the number of Wikipedia features obtained could be very large. This can have an adverse impact on the performance of the downstream classification/clustering algorithms. In this thesis, we propose a novel method of feature generation that addresses some of these shortcomings.

The Proposed Method of Feature Generation The proposed method takes more principled approach of feature generation from a web corpus. It deploys a topic modeling technique [Blei et al., 2003] to extract K topics from a given web corpus. The definition of a topic here is the probability distribution over the words in the vocabulary. If the i^{th} topic is denoted as z_i then z_i is the probability distribution $p(w|z_i)$ where w is a word in the vocabulary. This probability distribution actually defines the topic. For example, if the topic z_i is about "sports" then a sports related word w (e.g. "football", "cricket" etc.) are likely to have high value for $p(w|z_i)$. A topic modeling technique also provides methods to estimate the probability distribution of the K topics for a given document d (i.e. $p(z_i|d)|z_i \in \{z_1 \dots z_K\}$). The proposed feature generation method uses topic modeling and works as follows (Figure 1.4)

1. Given a web corpus W it applies a topic modeling technique (in particular Latent Dirichlet Allocation [Blei et al., 2003]) to extract K topics $(z_i \dots z_K)$ from W. A topic z_i is the

probability distribution over the words of the vocabulary.

2. Now lets assume a classification or clustering task is to be performed on a document collection $\{D\}$. For each document $d \in D$, the probability distribution ($p(z_i|d)$) of the K topics is then estimated using topic modeling technique. $p(z_i|d)$ is then used as the i^{th} topic feature of the document d.

3. The documents are then represented in feature space consisting of bag of words features as well as the K topic features. Once the documents are represented in a feature space standard classification/clustering algorithms can be applied directly.

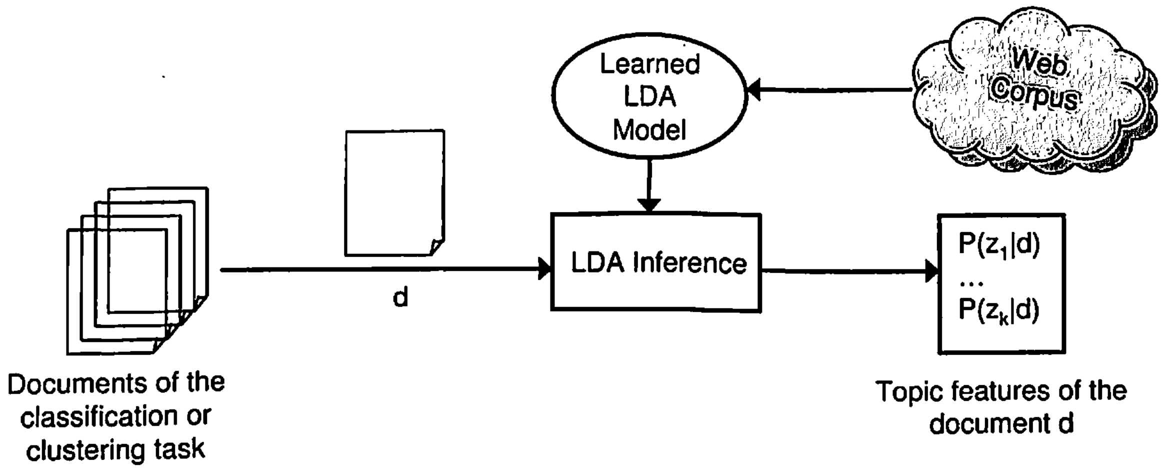

Learned LDA Model
Web Corpus
d
LDA Inference
P(z1|d)
...
P(zk|d)
Documents of the classification or clustering task
Topic features of the document d

Figure 1.4: The proposed feature generation algorithm

Our experiments show that these topic features can help in improving text classification accuracy by more than 30% on average. Also, the proposed feature generation method addresses many of the limitations observed in the retrieval based feature generation method from Wikipedia described earlier. For example, it does not depend on the titles of the Wikipedia and therefore can work with any other web corpus. It uses only K features, where K is typically in the range of 200-300.

1.2.3 Gathering Training Data for Web Page Classification

The third sub-problem addressed in the thesis is to use the web information sources to gather training data to build a web page classifier for any given concept. A web page classifier for a concept can discriminate between the pages the belong to the concept and those which do not. Web page classifier is an important tool to deal with the problem of information overload. Better visualization and navigational capability can be added by categorizing the web pages under different categories. Web pages are very diverse. To build a web page classifier that can perform well across different types of pages, a substantial amount of training data is generally required. Moreover, in some settings the target categories could be user defined and therefore not predetermined. For example, a user may want all the incoming news/blog articles related to "German Car" to be put under a certain folder. In this work, we develop methods to automatically gather training data from different web sources for any arbitrary category or concept. Obtaining labeled training data is generally a manual and expensive step in building a classifier. Here we demonstrate that it is possible to automatically build highly accurate classifiers for arbitrary concepts by leveraging different web sources.

Labeled training data for a text classifier is a set of text documents along with the correct class labels. A correct class label is an element of a predetermined set of classes (also called concepts or categories in text classification). For example, let say the set of classes are *World*, *Science & Technology*, *Entertainment* and *Other*. Then the labeled training data for the corresponding text classification task will be a set of documents where each document is associated with one or more of those four classes. A supervised classifier (e.g. Support Vector Machine [Vapnik, 1995]) learns a model from the labeled training data. This learned model can then be used to classify new documents that were not there in the training set. These new documents are also referred to as test data as they are used to evaluate the classifier accuracy.

One essential step in building a supervised classifier is gathering labeled training data. Generally it is assumed that the training data is drawn from the same distribution as the test data. The accuracy of the learned classifier is highly dependent on the quality and the amount of the training data. In the case of web page classification, the web pages are very diverse. Therefore obtaining a training set that reflects the distribution of the test data is difficult. Also, if the target classes are arbitrary and not predetermined then it is not feasible to obtain training data manually.

The problem of gathering training examples to build a web page classifier was recognized in the past as well [Davidov et al., 2004]. But most prior approaches in this area assume web directory like Dmoz [UrlDmoz] as a good source of training data [Davidov et al., 2004]. Dmoz is a human edited web directory with five million pages categorized under nearly six hundred thousand categories. The categories in Dmoz form an hierarchy. Examples of Dmoz category hierarchy are *Top/Arts/Movies/...*, *Top/Recreation/Food/...* etc. Given that the web pages in Dmoz are manually categorized and the huge size of Dmoz, it is natural to assume Dmoz as a source of good quality training data for those categories. In fact when the test data is also drawn from Dmoz, the accuracy achieved by a classifier, trained on Dmoz data, is quite high. But we show that this cross validation accuracy over the data obtained from Dmoz is misleading. Surprisingly even many recent papers related to web page classification just report the cross validation accuracy [Bennett and Nguyen, 2009, Xue et al., 2008] on Dmoz data.

The pages in Dmoz do not reflect the distribution of pages obtained from other sources in the web. Therefore, a classifier built using training data obtained from Dmoz actually performs badly while classifying the web pages of other sources. We need to generate more robust set of training data so that the trained classifier performs well in classifying pages from different sources in the web.

Our Contribution

In this work, we demonstrate the possibility of building web page classifiers without the hassle of manually labeling training examples. We propose methods to automatically gather training data utilizing different web sources. Given a concept name (e.g. *German Car*) several web sources are utilized to obtain training data to build a classifier for that concept. In particular, we use four different web sources, Wikipedia, Google, Dmoz and Delicious [UrlDelicious], to obtain labeled training data. Figure 1.5 shows the high level block diagrams of the process of gathering training data from different web sources.

First we establish that any single web source is not sufficient to build a highly accurate classifier that can perform well across test data of different sources. Therefore, we need to exploit more than one sources to construct our training data to build a classifier. We then demonstrate that the sources differ considerably from one another. The underlying distribution of the pages or the definition of the concept are different in each source. Therefore, blindly mixing the training data obtained from different sources actually creates a far noisier, heterogeneous training data that actually hurts the classifier performance. We then develop several intelligent methods to combine the training data obtained from these different sources. We show that when the training data is created by intelligently combining the different sources, the classifier achieves more than 13% higher accuracy when compared with the baseline approach of blindly mixing the data of different sources together. Finally, we demonstrate that by leveraging different web sources it is possible to build classifiers that can perform well across different types of web pages.

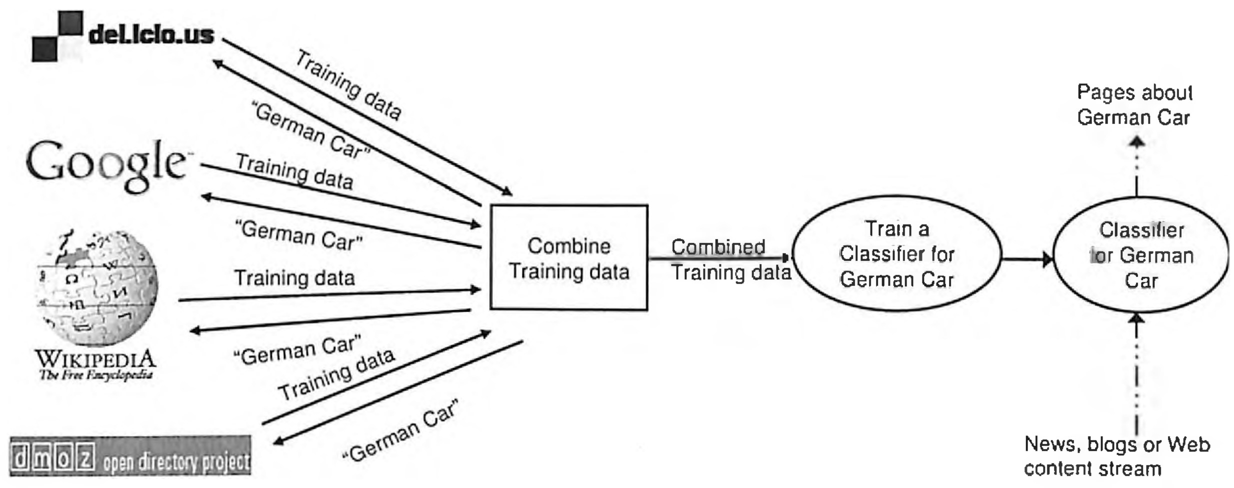


Figure 1.5: Training data generation from different web sources

1.3 Overall Contributions

The key contributions of the thesis are

1. We introduce and formalize the notion of quantity consensus query (QCQ) and propose novel algorithms that can provide answer to QCQs by gathering evidences from the web.

2. We demonstrate that the topic features obtained from Wikipedia can improve the accuracy of clustering short text documents and also are valuable in a setting of inductive transfer for text classification

3. We propose a novel method of feature generation from an additional corpus that addresses some of the shortcomings of a recently proposed feature generation method from Wikipedia

4. We develop methods to automatically obtain training data from different web sources. We demonstrate that it is possible to build highly accurate classifiers by combining training data obtained from multiple web sources.

Chapter 2

Literature Review

The major contributions of the thesis are the novel techniques of mining the web to answer quantity consensus queries, obtaining topic features and gathering training data for a given topic. But several methods and experiments described here use many state of the art machine learning algorithms. It is important to give an overview of the machine learning algorithms used in the thesis to set the notations and terminology. Therefore we divide this chapter into two sections. The first section describes different document representation techniques and machine learning algorithms used in the thesis. The second section reviews the prior work related to the contribution of the thesis.

2.1 Machine Learning Algorithms Used in the Thesis

This section first reviews techniques for text document representation. The document representation techniques include vector space model [Salton and McGill, 1986] and different topic modeling techniques. We then review the state of the art machine learning algorithms for text clustering, text classification and learning to rank. The notations used here are mostly standard, so that the reader familiar with any of these techniques can skip the corresponding subsection.

2.1.1 Document Representation

Most machine learning and information retrieval algorithms work on vector space or feature space representation of the documents. Once the documents are represented in a vector space, the similarity or distance between two documents can be computed using standard methods like cosine similarity or euclidean distance between the corresponding vectors. Most popular method of representing documents in a feature space is bag of words method. Although simple and widely used, bag of words representation has some limitations. In particular, bag of words representation cannot model well the ambiguity of individual terms or the relation between different terms. Topic modeling technique addresses some of these shortcomings by representing the documents in a low dimensional space. A brief overview of these two techniques are given next.

Bag of Words Representation

In this method, each dimension of the vector space corresponds to a separate term in the vocabulary. Sometimes the terms are *stemmed* by collapsing the morphological variants. If there are m terms in the vocabulary then each document is represented as a vector of m dimensions ($\mathbb{R}^m$). If a term appears in the document then the corresponding element in the vector gets a non-zero weight. There are several ways to compute this weight of a term in a document. The most popular weighting method is TF*IDF where, TF is frequency of the term in the document and IDF is the inverse of the document frequency of the term in the whole corpus. Definition of TF (Term Frequency) and IDF (Inverse Document Frequency) are given below.

$$\text{TF of term t in the document d} = \frac{\textit{Number of times t appears in d}}{\textit{Total number of terms in d}}$$

$$\text{IDF of term t} = \log \frac{\textit{Total number of documents in the corpus}}{\textit{Number of documents where the term t appears}}$$

This way of document representation with TFIDF weighting is also known as vector space model and was first presented in [Salton et al., 1975]. More details can be found in [Salton and McGill, 1986]. In chapter 4, we will be using the bag of words representation of documents as the baseline representation to compare against the proposed algorithms. The proposed algorithms represent the documents in an enriched feature space containing topic features in addition to the term features and thereby addresses some of the shortcomings of bag of words representation.

Topic Modeling

When the dimensions of the vector space are individual terms then the cosine similarity between the documents just captures the syntactic matching between the individual terms. The similarity value in this case is basically the lexical matching between the terms of the two documents. It ignores the connection between the terms and the ambiguity of individual terms in a language.

Topic modeling techniques try to model the connection between the terms. It assumes that two terms are related if they are often referred in similar contexts [Jones, 1971]. A topic modeling technique basically groups the terms appearing in similar contexts. Such group of terms are referred as topics. The documents are then represented in a vector space where each dimension corresponds to a topic. The value of a feature (or the value of an element in the vector) represents the prominence of the corresponding topic in the document. Number of topics is far less than the number of terms in the vocabulary as each topic is a group of (related) terms. Therefore, the number of dimensions in this topic based representation is far less compared to the bag of words method. Since each topic is a group of related terms, this representation can model the connection between the terms. Also, if a term is ambiguous (has multiple meanings) it can belong to multiple topics. The estimated topics for a document is expected to capture the meanings of the terms as used in the document. For example, the term "play" could belong to the topic "Theatre" or to "Sports". If a document is talking about "Theatre" then the term "play" in the document will contribute to the topic "Theatre" and the corresponding feature will have higher value. So the ambiguity of individual terms is also addressed in some sense.

The first work in the area of topic modeling was Latent Semantic Analysis (LSA) [Deerwester et al., 1990] which was published in early 90s. But only recently the term "topic modeling" has been coined and it has become an active area of research [Steyvers and Griffiths, 2007]. Now there are several methods for topic modeling (e.g. pLSA [Hofmann, 1999], LDA [Blei et al., 2003], hLDA [Blei et al., 2004], etc.). These topic modeling methods differ in the way they define and estimate the topics (groups of related terms). Next we briefly describe LDA, the topic modeling technique most popular in the community and used in this thesis.

Latent Dirichlet Allocation (LDA) Latent Dirichlet Allocation (LDA) [Blei et al., 2003] is probably the most widely used topic modeling technique in the literature. LDA is a generative model and assumes that the words in a document are generated from K hidden topics. In LDA, each document d is viewed as a multinomial distribution θ over the K topics. Also for each topic z there is a multinomial distribution over the words (β_z). Each word of a document is generated by first sampling θ from a Dirichlet distribution with parameter α, then a topic z from θ and then the word w from β_z. The steps of the document generation process is shown below.

- For each document d in the corpus
  - Draw a topic distribution $\theta \sim Dir(\alpha)$, where $Dir(\alpha)$ is the Dirichlet distribution with parameter α.
  - For each word w in the document
    * Draw a topic assignment $z \sim Mutinomial(\theta)$
    * Draw a word $w \sim Multinomial(\beta_z)$, where β_z is the distribution of words given the topic z.

Graphical model representation of LDA is shown in the Figure 2.1. Note that here a topic z is a probability distribution over the words (β_z). The meaning of the topic is interpreted from this probability distribution. For example, if in β_z, sports related words (e.g. "football", "play", "win" etc.) have high probability then the topic z can be interpreted as sports. Similarly, each document is viewed as a probability distribution (θ) over the topics. If the document is about sports then the topics related to sports is likely to have higher probability in θ.

TH 6121

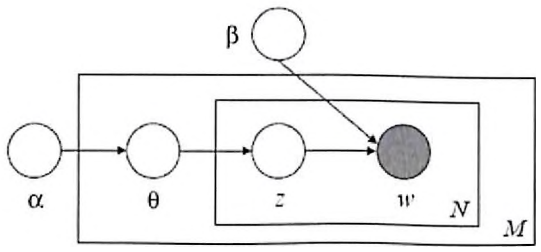


Figure 2.1: Graphical Model Representation of LDA

Learning in LDA involves estimating the parameters θ (for every document) and β_z (for every topic z) from a corpus. LDA also has an inference procedure for estimating the topic distribution θ for any given document. The given document may not be part of the corpus from where the LDA parameters are learned. That is, using LDA inference we can estimate the topic distribution even for a new document. Several algorithms, including variation approximation [Blei et al., 2003], Gibbs Sampling [Griffiths and Steyvers, 2004] have been proposed for learning and inference in LDA. Gibbs sampling method is easy to setup and is used in our implementation of LDA.

LDA model is essentially a Bayesian version of the pLSA model and addresses many shortcomings of pLSA. In fact it can be shown that the pLSA model is a special case of LDA under uniform Dirichlet prior distribution [Girolami and Kaban, 2003]. Having the Dirichlet prior often generates more reasonable topic distribution in practice. LDA has got quite popularity in the research community and many follow up papers are regularly being published in the recent conferences.

In this thesis, we show another use of LDA rather than using it to represent documents in a topic space. In the Section 4.2, we demonstrate that LDA can be used to generate topic features to improve text classification/clustering accuracy. For that purpose, we first use LDA to extract K topics from a web corpus. These K topics are then used as K topic features of the documents of the given classification/clustering task. For each document in the given classification/clustering task, we estimate the distribution of K topics (θ) using the inference procedure of LDA. Now this K dimensional probability distribution acts as K topic features of the document. We show that these additional topic features can improve the accuracy of text classification often by huge margin.

2.1.2 Text Clustering

Text clustering is an important machine learning tool and has many different applications. Text clustering is used in organizing and visualizing a document collection, e.g. grouping news of same topic or clustering search results for better visualization. Text clustering is an unsupervised machine learning tool. It has been widely studied in the literature and an overview of different clustering algorithms can be found in [Chakrabarti, 2002]. This section reviews the clustering algorithms which will be used later to test our hypothesis that topic features can help in improving the text clustering accuracy.

Clustering Algorithms

The goal of text clustering is to partition a set of documents into K clusters $(C_1, \ldots, C_K)$. Clustering is commonly done by optimizing an objective function that measures the quality of the clusters. The cluster quality can be measured in terms of intra-cluster similarity or inter-cluster distance. Intra-cluster similarity denotes an aggregated similarity between the elements (documents) belonging to the same cluster. Inter-cluster distance is a measure of aggregated distance between the elements of different clusters. Some examples of cluster quality functions are shown below.

Intra-cluster similarity I $= \sum_{i=1}^{K} \frac{1}{n_i} \sum_{d_i,d_j \in C_i} sim(d_i, d_j)$

Inter-cluster similarity E $= \sum_{i=1}^{K} n_i sim(C_i, C)$

$C_i =$ Centroid of cluster i $= \frac{1}{n_i} \sum_{j=1}^{n_i} d_j$ and $C =$ centroid of the corpus $= \frac{1}{n} \sum_{j=1}^{n} d_j$

Here sim is a similarity function and can be cosine similarity, n_i is the number of elements in the cluster i and n is the total number of elements in the corpus. The documents need to be represented in the vector space to compute the cosine similarity. Also note that E is the inter-cluster similarity and generally the reciprocal of E is considered as the inter-cluster distance. The cluster quality functions I and E can both be combined to create a single objective

$$maximize \quad H = \frac{I}{E} \tag{2.1}$$

The above formulation is just one way to formulate the clustering objective. There are many other ways to define an objective function for document clustering [Zhao and Karypis, 200]. Once the objective is defined the partitioning approach of clustering can proceed in two ways, *bottom-up* or *top-down*.

bottom-up This approach starts with each document as a cluster and then in each iteration collapse two clusters into a single cluster. The two clusters are chosen by optimizing the clustering objective. The iterations continue until there are only K clusters. This *bottom-up* approach is also known as *Agglomerative Clustering*.

top-down A well known member of this family is k-means clustering. K-Means clustering starts with random assignment of documents into K clusters. Then in each iteration documents are assigned to the cluster with most similar centroid (or to cluster for which the clustering objective is optimized). Once the documents are assigned to clusters the cluster centroids are re-computed. The iterations continues until the cluster assignment of documents or the clustering objective do not change much.

Agglomerative clustering often yields better cluster quality but it is slow as it needs to do exhaustive pair wise comparison in each iteration. K-Means clustering is fast but the cluster quality depends on the choice of initial cluster assignments.

The above techniques are known as partitioning approach to clustering. There are others approaches, for example generative models or probabilistic approaches [Chakrabarti, 2002]. Also, the above clustering assignment is known as hard cluster assignment as a document is assigned to a single cluster. There are soft clustering algorithms where a document is assigned to multiple clusters with some weights (or probability distribution). For example, pLSA and LDA are soft clustering techniques where the documents are assigned to K cluster with a probability distribution (θ in LDA).

Stopping Criteria Most clustering algorithms, including the partitioning approaches described above, require the user to specify the number of clusters K to be obtained. One possible way to overcome this problem is to start with $K = 1$ and continue to explore $K + 1$ as long as some cluster quality function improves. One need to be cautious that the cluster quality function does not have any singularity. That is the cluster quality function should not encourage a single cluster or number clusters equal to the number of elements. For example, intra-cluster similarity will be the maximum value when the number of clusters is equal to the number of elements. On the other hand the clustering objective H defined in (2.1) can be used to choose the optimum number of clusters.

Evaluation A clustering algorithm can be evaluated against a gold standard data using a measure called *accuracy*. The gold standard data should contain the ideal clustering, i.e. the exact number of clusters and cluster assignment of the documents. Let say the gold standard clustering is $C_1^*, \ldots, C_K^*$ and the clustering algorithm comes up with clustering $C_1, \ldots, C_L$. Here, K and L are the ideal number of clusters and number of clusters discovered by the clustering algorithm respectively. K and L could differ. To compute the accuracy of the clustering algorithm, first each element in $\{C_1, \ldots, C_L\}$ need to be exclusively mapped to an element in $\{C_1^*, \ldots, C_K^*\}$. Let say C_i is mapped to C_j*. Then the overlap between the elements of C_j* and C_i can be thought of as the correct assignment. Accuracy of a clustering algorithm is the fraction of total elements that are correctly assigned to the clusters.

The computed accuracy depends on the mapping between C_i and C_j^*. Therefore the mapping is done such that the accuracy is maximized. That is the mapping of C_i and C_j* need to be done by maximizing the total overlap of elements between C_i and C_j*. This is an instance of assignment problem or maximum weighted matching in a bipartite graph and can be solved in polynomial time.

Cluto [UrlCluto] is a clustering tool that provides six different clustering algorithms under partitioning approach. It also has a wide range of objective functions to select. We use Cluto in our experiments with text clustering. In the Section 4.1.1 we show that when documents are represented in a feature space containing topic features obtained from Wikipedia, the accuracy of different clustering algorithms improve significantly.

2.1.3 Text Classification

Text classification (TC) (a.k.a text categorization) is the task of labeling natural language texts with *class labels* from a predefined set. Formally, given k class labels ($C = c_1 \dots c_k$) the task is to assign one or more class labels from C to a document d. In *Single-Label* classification, only one class label is assigned where as in *Multi-Label* classification, more than one class labels can be assigned to a document d. Initially a set of documents $d_1 \dots d_l \in D$ and the ground truth class labels for each document is given as the training data. The training data is used to learn a classifier (or model). The learned classifier is then used to do the actual TC task on any given (new) document d. Often the number of possible class labels $k = 2$ and the classifier has to assign either of the class labels to a document. This type of classification task is known as binary classification task and the class labels are denoted as $+1, -1$ also called as positive and negative class respectively. Binary classification is fundamental to the classification task and can be extended to build a multi class classification system.

The performance of a learned classifier is evaluated on a test set. A test set, like the training set, is again a collection of documents $d_1 \dots d_{\hat{l}} \in D$ with ground truth class labels of each document. The classifier is shown only the documents but not the class labels. The classifier is evaluated by comparing the classifier predicted class labels and the ground truth class labels of the documents. Common evaluation measures are *accuracy* (fraction class labels predicted correctly) and *F-Measure* (harmonic mean of precision and recall). Details of these evaluation measures can be found in [Sebastiani, 2002].

TC dates back to the early '60s, but only in '90s it became an active research topic among information retrieval and machine learning communities [Sebastiani, 2002]. TC is now being used in a range of applications, including news/blog content categorization, spam filtering, personalization, word sense disambiguation etc. TC is still a very active field of research with many new techniques are coming up for web page classification [Qi and Davison, 2008], classification under a taxonomy hierarchy (instead of flat set of class labels) [Xue et al., 2008] and several others.

A number of different classification techniques exists in the literature. The most commonly used text classifier is *Support Vector Machine* (SVM), first introduced by Boser, Guyon and Vapnik in COLT92 [Boser et al., 1992]. It is often observed that SVM outperforms other classifiers like Naive Bayes, Decision Tree, Neural Network etc. in the tasks of text classification [Joachims, 1998, Yang and Liu, 1999]. As argued by Joachims [1998], SVM has two important advantages in TC.

- In traditional feature space (i.e. BOW) representation of the document, the number of dimensions is very high (i.e. size of the vocabulary). SVM is fairly robust to over fitting and can scale up to very high dimensions.
- Not much parameter tuning is required and the theoretically motivated default choices of parameters generally work well in TC.

In this thesis also we used SVM as the base classifier to establish our hypotheses. In particular we show that

- Topic features can help in improving the accuracy of SVM classifier in many different text classification tasks (chapter 4).
- Different web sources can be used to get training data to automatically build a SVM classifier for web page classification (chapter 5)

Next we give a brief description of the SVM classifier.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a linear classifier with a decision surface of the form $w \cdot x + b = 0$, where x is the suitable feature vector representation (e.g. BOW) of the document. The weight vector w and the bias parameter b is determined using the training data. Document x is classified to the class positive (+1) or negative (-1) based on whether the quantity $w \cdot x + b$ is greater or less than 0.

$w \cdot x + b = 0$ is an equation of a hyperplane with the two different sides of the hyperplane are denoted by $w \cdot x + b > 0$ or $w \cdot x + b < 0$. SVM chooses the hyperplane (i.e. the parameter w and b) that maximally separates the positive and negative training examples. Assume that there are n training examples (represented as vectors in $\mathbb{R}^m$) from the two classes. SVM seeks a hyperplane that separates the positive and negative examples by the widest possible margin. This can be written as

$$\begin{aligned} &Minimize \quad \frac{1}{2} w \cdot w \\ &subject \quad to \quad c_i(w \cdot x_i + b) \geq 1 \qquad \forall i = 1 \dots n \end{aligned} \tag{2.2}$$

Where $x_1 \dots x_n$ are the training document vectors and $c_1 \dots c_n$ are the class label of the corresponding document. Each class label c_i has value in $\{+1, -1\}$. The obtained hyperplane is orthogonal to the shortest line connecting the convex hull of the positive and negative classes.

In the equation (2.2), the assumption is that it is possible to draw an hyperplane separating the positive and negative examples of the training data. That is, the positive and negative examples of training data are separable which is often not the case. To handle the general case where the training data is not separable, slack variables $\xi_1 \dots \xi_n$ are introduced. The slack variable ξ_i measures the degree of misclassification of the training example x_i. With the slack variables the equation (2.2) becomes

$$\begin{aligned} &Minimize \quad \frac{1}{2} w \cdot w + C \sum_i \xi_i \\ &subject \quad to \quad c_i(w \cdot x_i + b) \geq 1 - \xi_i \qquad \forall i = 1 \dots n \\ &and \quad \xi_i \geq 0 \qquad \forall i = 1 \dots n \end{aligned} \tag{2.3}$$

Equations (2.2) and (2.3) are instances of (convex) quadratic objective function with linear constraints and therefore can be solved using a quadratic programming package. For n documents, time taken to train a SVM is n^a, where a is typically between 1.7 and 2.1 [Chakrabarti, 2002]. The recent development in this area has made it possible to train SVM in linear time using *cutting plane* method of convex optimization [Joachims, 2006].

2.1.4 Learning To Rank

So far we have studied the machine learning techniques to cluster similar documents or to predict the class labels of a document from a predefined set of labels. But in many applications the central issue is *ranking*. For example, in document retrieval, a ranked list of documents need to be presented given a keyword query. Other such applications are collaborative filtering, product rating etc. *Learning to rank* is a machine learning technique that learns to *rank* set of items from a given set of training examples.

Learning to rank is an active field of research with a strong motivation in document retrieval. The ranking function of earlier generation search engine was a combination of small set of features (e.g. TF, IDF and document length). Since the number of features were small it was possible to manually tune the ranking parameters. Today's search engines use hundreds of features to produce the ranked list of search results. Apart from TF, IDF type features, the features used today are PageRank [Page et al., 1998] and other link based measures, users' feedbacks (e.g. clicks on search results), user's browsing history, general reputation of the site etc. Manually tuning the parameters of ranking function that combines all such features is an extremely difficult and unreliable task. Recently the IR community started exploring the use of machine learning techniques to learn the ranking function. For a given query, a set of documents are first retrieved from the index. Each document is then represented in a feature space where the features are obtained using query as well as the document. The ranking function usually scores each document of the query as a linear or non-linear combination of the feature values. The documents are then ranked in the decreasing order of the scores.

The job of a machine learning algorithm here is to learn a ranking function from a set of manually labeled training data. Training data for learning to rank algorithm is prefect or partial ordering defined among a set of documents for a set of queries. The task then is to learn a ranking function that maximally fits the given ordering of documents for the training queries subject to some regularization. Next we describe a simple training and evaluation scenario of learning to rank to provide more clarity on standard learning to rank algorithms.

Simple Training and Evaluation Scenario

Training data for learning to rank consists a set of query Q. Each query has a set of documents with relevance labels assigned to them. The relevance labels define the partial ordering among the documents of a query. Suppose q is a query from the set Q and $X^{(q)} = \{x_{q1} \ldots x_{qn}\}$ is the set of n documents for that query. Here the document x_{qi} is represented in a d dimensional feature space $\mathbb{R}^d$. Unlike classification and clustering, the features here are not BOW features but computed from the query-document $\{q, x_{qi}\}$ pair. The features denote various similarity measure of the query to the document (e.g. BM25 [Salton and McGill, 1986]) or document quality (e.g. PageRank [Page et al., 1998]).

The training data comes with relevance labels of the documents. The relevance label y_{qi} of a document x_{qi} is in general given in a ordinal scale $\{r_1 \dots r_k\}$. There exists a total order between the relevance labels $r_1 > r_2 > \dots > r_k$. Therefore the relevance labels define the partial order among the documents of a query. For example, if $y_{qi} > y_{qj}$ then the document x_{qi} is assumed to be more relevant than the document x_{qj}. Such pairs of $\{x_{qi}, x_{qj}\}$ are called *preference pair* and denoted as $x_{qi} \succ x_{qj}$. If x_{qi} is preferred over x_{qj} (i.e. $x_{qi} \succ x_{qj}$) then x_{qi} need to be ranked higher than x_{qj}.

Let $Y^q = \{y_{q1} \dots y_{qn}\}$ is the relevance labels of the documents of the query q. Then the training data of learning to rank algorithm can be denoted as

$$D = \{X^q, Y^q\}_{q=1}^{Q}$$

The task is to learn a ranking function $f : \mathbb{R}^d \mapsto R$ that takes the feature vector of a document (x_{qi}) as input and returns a score. The documents of a query q are then ranked in the decreasing order of their scores. The ranking function is learnt such that it mostly obeys the pair preference ($\succ$) orders in the training data. That is,

$$x_i \succ x_j \Leftrightarrow f(x_i) > f(x_j)$$

At the time of deployment or testing, given a test query t_q a set of documents $X^{t_q} = \{x_{t_q 1}, \dots, x_{t_q n}\}$ are first retrieved from the index and then the documents are ranked in the decreasing order of the scores $f(x_{t_q i})$.

RANKSVM There exists many algorithms to learn the ranking function f from the training data. A comprehensive list of ranking algorithms can be found in [UrlLetor]. In this thesis, we used RANKSVM [Herbrich et al., 2000] to rank the snippets of a given query in QCQ. RANKSVM scores the documents as the linear combination of the features. That is, the scoring function f is defined as $f(x) = w \cdot x$. Training procedure in RANKSVM learns the weight vector w by minimizing the violations in the pair preference. That is, if $x_{qi} \succ x_{qj}$ in the training data then RANKSVM tries to learn a w such that $w \cdot x_{qi} > w \cdot x_{qj}$. This can be written as the following quadratic optimization problem.

$$
\begin{aligned}
Minimize \quad & \frac{1}{2} w \cdot w + C \sum_{qij} \xi_{qij} \\
subject \quad to \quad & w \cdot (x_{qi} - x_{qj}) \geq 1 - \xi_{qij} \qquad \forall q \in Q \quad \forall x_{qi} \succ x_{qj} \\
and \quad & \xi_{qij} \geq 0 \qquad \forall q, i, j
\end{aligned}
\tag{2.4}
$$

The above optimization formula looks very similar to the optimization formula of SVM (equation (2.3)). In fact one way to look at RANKSVM is that it uses the SVM classification formulation over pair of instances. For each preference pair $\{x_{qi}, x_{qj}\}$, it creates an instance $(x_{qi} - x_{qj})$. If $x_{qi} \succ x_{qj}$ then the instance is labeled as +1 otherwise -1. The advantages of the RANKSVM includes the simplicity of the model. It is also shown to be very effective in document retrieval [Joachims, 2002].

RANKSVM view each preference pair as a training instance. This type of learning to rank algorithms are called *pairwise* approach. There are two other approaches: *pointwise approach [Li et al., 2007]* where individual document is used as an instance, *listwise approach [Joachims, 2005, Chakrabarti et al., 2008]* where entire ranked list of documents of a training query is used as a training instance.

In the chapter 3, we use RANKSVM to rank the snippets of a given QCQ. We have experimented with other learning to rank algorithms (e.g. [Yue et al., 2007]). But in our experiments RANKSVM was giving better accuracy. In standard learning to rank formulation individual snippets are ranked independently. In the chapter 3, we argue that for QCQ we need design a ranking algorithm that ranks a set of snippets collectively. We propose novel methods towards that and showed huge improvement in accuracy when compared to standard learning to rank algorithms.

Evaluation Measures Standard evaluation measures of IR are used to evaluate a learning to rank algorithm. A learning to rank algorithm reports a ranked list of items (documents or snippets) for a given query. For evaluation, a manually labeled data set (similar to the training data) is used. The evaluation measures credit an algorithm for ranking the more relevant items on top of the less relevant items.

The commonly used evaluation measures in IR are mean reciprocal rank (MRR) [Voorhees, 2001], mean average precision (MAP) [Yue et al., 2007] and normalized discounted cumulative gain (NDCG) [Järvelin and Kekäläinen, 2000]. Assume there are Q queries in the test set. For each query q we consider only the top n documents reported by the ranking algorithm. Then MRR, MAP and NDCG are defined as follows.

Mean Reciprocal Rank (MRR) For some queries, e.g. navigational queries ("Hewlett Packard"), there is mainly one relevant URL (`http://www.hp.com/`). Therefore, it is important to evaluate the ranking algorithm based on where it ranks the relevant URL. MRR is a suitable evaluation measure for that purpose. MRR evaluates an algorithm based on the top most rank of the relevant documents. If the topmost rank at which a relevant document for q is found is r_q, then reciprocal rank for q is $\frac{1}{r_q}$. Averaging over all $q \in Q$ MRR is defined as follows

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q} \tag{MRR}$$

Mean Average Precision (MAP) MAP evaluation measure can be used when the relevance labels of the items are binary.

Precision at rank r is defined as

$$P@r = \frac{\#\{\text{relevant items up to rank r}\}}{r}$$

The average precision for a query q is defined as

$$AP(q) = \frac{\sum_{r=1}^{n} P@r * 1[\text{item at rank r is relevant}]}{\text{number of relevant items}}$$

MAP is averaged over all queries Q in the test set.

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{MAP}$$

Normalized Discounted Cumulative Gain (NDCG) NDCG evaluation measure is of recent interest in Information Retrieval and Machine Learning community. The DCG of a query q and ranked list of documents is defined as

$$DCG(q) = \sum_{i=1}^{n} (2^{r(i)} - 1)/log(1 + i) \tag{DCG}$$

Where $r(i)$ is the relevance label of the item ranked at i. NDCG is normalized DCG and the normalization is done by the maximum achievable DCG for q. Maximum DCG (DCG^*)

is obtained when the documents are ranked in the decreasing order of their relevance labels. NDCG for a test set is defined as

$$NDCG = \frac{1}{|Q|} \sum_{q \in Q} \frac{DCG(q)}{DCG^*(q)} \tag{NDCG}$$

2.2 Review of Related Work

So far we have reviewed the machine learning techniques used in the thesis. Next we review the work that are related to the specific contribution of the thesis. In section 2.2.1, we review prior work addressing quantity query and question answering from web in general. Section 2.2.2 describes prior approaches of feature generation from WordNet and Wikipedia. In section 2.2.3, we discuss work related to gathering training data from web corpora.

2.2.1 Answering Quantity Consensus Queries (QCQs)

As such there is no prior work addressing "quantity consensus queries" or even "quantity related queries". Quantity search is a special case of entity search. In entity search, a user searches for a particular type of entity (e.g. "person", "organization", "place" etc.). From internal system perspective entity search is same as factual question answering. The accuracy of entity search or factual question answering from web often improves by exploiting the redundancy of information in the web [Clarke et al., 2001]. Aggregating evidences found in multiple websites often improves the ranking accuracy in entity search. But evidence aggregation techniques used in standard entity search is not adequate for quantity queries. To answer quantity queries we need to design a ranking algorithm that collectively scores a set of snippets (text segments) containing near by quantities. But in the past there has been relatively little work on collective ranking of this form. One recently proposed approach that can be adapted to do collective ranking for quantity query is Laplacian consensus. In this section, we first discuss evidence aggregation techniques used in entity search. Then we give an overview of the Laplacian consensus technique.

Evidence Aggregation in Entity Search

Entity search or answering factoid questions from web require different approach than answering questions from a small collection of documents. A small corpus is not expected to have much redundancy. Therefore, often sophisticated natural language processing is required to detect the single or few mentions of the correct answer in the small corpus. On the other hand, while answering questions from web it is often profitable to exploit the redundancy of information in web [Brill et al., 2002, Clarke et al., 2001]. Therefore evidence aggregation is an important step of question answering from web.

Standard entity search system is designed to answer general factoid questions (e.g. who, what, where, when etc.). The evidence aggregation in such system is commonly defined as weighted voting. Given a query, first a set of snippets are retrieved from the corpus. Each snippet here contains an entity as the candidate answer. Retrieving snippets of a query commonly involves entity extraction and standard information retrieval. Each snippet is then scored independently using a learned model or some heuristics (e.g. how good the snippet text matches to the query, confidence of the entity extractor on the entity of the snippet etc.). The score of an entity is defined as the aggregated scores of the snippets containing that particular entity [Clarke et al., 2001, Brill et al., 2002, Wu and Marian, 2007]. Some syntactic variation in the entity mention can also possibly be detected (Barack Obama vs Mr. Obama). In such case the scores of those (syntactically varied) entities are aggregated.

As argued in the introduction section, such weighted voting is not adequate to answer quantity queries. In quantity search, we need to reinforce evidence from nearby quantities. For example, if 480 USD is a correct price of a Canon camera then it is likely that 470 or 490 USD are also correct prices. In the paper published by Wu and Marian [2007], they talked about answering quantity queries. But again simple weighted voting was used for evidence aggregation.

Some researchers [Moriceau, 2006, Prager et al., 2007] have devised type theories with rule systems to conflate syntactic and quantitative variations of candidate answers, aggregate evidence across these variations, and perhaps explain them. Substantial handcrafting of type systems and conflation rules are required in this approach.

Recently Ko et al. [2007] have proposed a probabilistic graphical model to do *collective ranking* of entities. An undirected graphical model (specifically a Boltzmann machine [Hinton and Sejnowski, 1986]) is defined where the nodes are the answer candidates and the edges denote the similarity between the answer candidates. The joint prediction model over the graph can estimate both the correctness of individual answers as well as their correlations. Although a principled approach but the joint prediction model has exponential time complexity. Also when applied to the quantity search, defining edge weights will become a problem. The difference of quantity values is not a proper indicator of quantity similarity. The similarity function between quantities are query dependent. For example, if the query is distance between Sun and Pluto then "10 billion" and "12 billion" may be similar quantities where as they might be less similar for other queries.

Laplacian Consensus Another way to do collective ranking is using Laplacian consensus as shown in [Qin et al., 2008]. This Laplacian method was originally proposed for pseudo relevance feedback scenario of information retrieval. When adapted to quantity search, the Laplacian method defines a graph over the snippets similar to the probabilistic graphical model discussed above. But it has tractable inference procedure and therefore used in the thesis for comparison (section 3.5).

The Laplacian consensus method used in the thesis works in the following way. Given a query, a graph is defined where the retrieved snippets are the nodes. The edge weight between two snippets denotes some similarity between those two snippets. The similarity could be defined to capture either the textual similarity of the snippets (as done in [Qin et al., 2008]) or the similarity between the quantities of the snippets. The idea of Laplacian consensus is that if two snippets are similar then we are reluctant to say one snippet is relevant and the other snippet is irrelevant. At the same time the relevance of each snippet can be judged independently based on the features of the snippet. In Laplacian consensus, the final scores of the snippets are assigned by considering these two factors. In section 3.5.5, we will give the exact details of the scoring function for this approach.

As in the case of probabilistic graphical model, defining edge weight is problematic for Laplacian method. Also, as we will see later that another major problem of Laplacian is that it tries to smooth the scores across all edges of the graph. Given a query there are many irrelevant quantities. We need not to smooth the scores between the irrelevant quantities (snippets). In this thesis, we propose asymmetric scoring functions that pay more attention to a quantity interval where most likely the relevant snippets belong to (section 3.6 and 3.7).

2.2.2 Generating Topic Features from a Web Corpus

The second major contribution of the thesis is obtaining topic features from a web corpus. Here we review the prior works that address the limitations of bag of words representation by obtaining features from different sources. Earlier approaches in this direction was using a dictionary to get the synonyms or hypernyms (higher level terms in the "IS-A" hierarchy) of terms. In most cases, the WordNet [Fellbaum, 1998] was used as the dictionary. The growth of Web 2.0 has created tremendous knowledge resources like Wikipedia. The scale and quality of Wikipedia corpus has prompted many recent research to look at Wikipedia resource for feature generation. In this section, we first review the feature generation techniques from WordNet and then describe several recently proposed methods to obtain features from Wikipedia.

Feature Generation from WordNet

WordNet [Fellbaum, 1998] is a large online thesaurus that captures the synonyms and hypernyms relationship between the words. Hypernyms, i.e. "IS-A" relationships (e.g. Einstein is-a Physicist) between the terms, form a hierarchy in the WordNet. Each node in this hierarchy is a synset (Synonym set). A synset is a set of words or phrases that are synonyms to each other with respect to a particular meaning. If a word or phrase has multiple meanings then it will appear in multiple synsets. Scott and Matwin [1999] proposed an approach where they used the synsets and hypernyms of the words as features. Given a document to be represented in the feature space, they first take the words of the document that has entries in the WordNet. For each word, the synset id hypernyms are then added as the feature of the document. Not much gain was observed in text classification with this new features.

Feature Generation from Wikipedia

Wikipedia is a classic example of the power of collaborative editing. Wikipedia is the largest and fastest growing encyclopedia in the world. With more than 3 million pages, 10 million registered users and 18 edits per page [UrlWikiStat], the coverage and the quality of Wikipedia articles are enormous. In a sense, Wikipedia contains a significant fraction of the knowledge of the world. Moreover Wikipedia corpus is freely available for download[1]. This has prompted many researchers to use Wikipedia as a knowledge source in various tasks including feature generation for document representation.

Gabrilovich's Method Gabrilovich and Markovitch [2006, 2007] first used Wikipedia in feature generation. The feature generation method works as follows. For each document first

[1]at `http://download.wikimedia.org/`

a set of contexts are extracted. The contexts of a document are individual words, sentences, paragraphs and the entire text of the document. Then for each context, 10 best matching Wikipedia articles are retrieved. The matching is done based on the textual similarity of the context and the Wikipedia article. The title of the retrieved Wikipedia articles are then used as the Wikipedia features of the document. The Wikipedia features are used in addition to the bag of words features. They showed that Wikipedia features can improve text classification accuracy [Gabrilovich and Markovitch, 2006] and can also help in determining semantic relatedness [Gabrilovich and Markovitch, 2007].

It was claimed that the Wikipedia features give more background information of the document. For example (as mentioned in the paper), for the document title "Apple Patents a Tablet MAC", the retrieved Wikipedia titles were "MAC OS", "LAPTOP", "AQUA" (the GUI of MAC OS X), iPOD etc. It was argued that this type of features contain relevant information and can also help in disambiguating the ambiguous contexts. Empirical results were shown to corroborate this claim.

In section 4.1, we will apply a very similar method of feature generation from Wikipedia and show that those Wikipedia features can help in "clustering short texts" [Banerjee et al., 2007] and "doing inductive transfer over text classifiers" [Banerjee, 2007]. Our feature generation method is slightly simpler than the method proposed by Gabrilovich et al. but still able to improve the accuracy of the both the tasks by a good margin.

Exploiting the Structure of Wikipedia in Feature Generation Wikipedia is not just a collection of pages but it has lots of structure information as well. For example, Wikipedia articles contain hyperlinks to other Wikipedia articles, each article is placed under one or more categories or sub-categories, the category sub-category graph of Wikipedia has a hierarchical structure. This structure information of Wikipedia is quite rich and the method proposed by Gabrilovich does not make use of it. More recent researches show how to use this structure information of Wikipedia in feature generation [Hu et al., 2008, Wang et al., 2007, Wang and Domeniconi, 2008]. The method shown in [Hu et al., 2008, Wang et al., 2007] first builds a WordNet like thesaurus from Wikipedia. An entry of the thesaurus is a Wikipedia article. The entry of the thesaurus or the Wikipedia article is referred as a concept. They define several relations between the concepts based on the content and the structure information of Wikipedia articles. The relations are described below

Synonym: The *redirect* hyperlinks of Wikipedia connects the concepts that are synonyms. For example, the acronyms U.S.A., U.S., USA, US redirect to the article "United States". This redirect links along with the *anchor texts* were used to define the synonyms.

Polysemy: The Wikipedia disambiguation page was used to identify the polysemy relation between the concepts. A disambiguation page of Wikipedia lists multiple senses of a concept. For example, the disambiguation page of Apple contains several different uses of the term Apple, including "Apple Inc", "Apple (album)", persons with name "Apple" etc.

Hyponym: Hyponym is the opposite of the hypernym in the "IS-A" relation (Einstein is-a Physicist, here Einstein is the hyponym of physicist whereas physicist is the hypernym of Einstein). But in that paper the term hyponym is used to mean hypernym only. Hyponym relation between the Wikipedia concepts can be identified using the category hierarchy of Wikipedia. Each article in Wikipedia belongs to some categories or sub-categories. Categories and sub-categories links form a hierarchy in Wikipedia.

Associative Relation: Associative relation defines similarity between the Wikipedia articles. Associative relation between two concepts is defined based on the textual similarity of the corresponding articles, overlap of the categories they belong and the path distance between those categories in the category hierarchy.

Given a document, they first select a candidate set of Wikipedia concepts that are mentioned in the document. Then for each selected concept the synonyms, hyponyms and the associated concepts are added as features. If the new concepts that are added have multiple meanings (i.e. the concept has an entry in a Wikipedia disambiguation page) then a disambiguation process takes place. The disambiguation process selects the closest concept from the list of entries in the Wikipedia disambiguation page. Here the closest concept is identified by the textual similarity of the document to the corresponding Wikipedia article of the concept and the link distance from the other unambiguous concepts added to the document. They reported modest gain in text classification and clustering accuracy.

Proposed Feature Generation Method The feature generation methods described above are quite ad-hoc in nature and can have performance issues. In the Gabrilovich method, Wikipedia articles were retrieved by textual "similarity" of the contexts to the Wikipedia articles. Top 10 retrieved Wikipedia articles were used as the Wikipedia features. It is not clear how the quality of the Wikipedia features depend on the "similarity" function being used or the number (here it is 10) of top articles kept for features. In both the above described methods a new feature is a Wikipedia article. Since the number of Wikipedia articles is 3 million, the new representation of documents can have millions of features. This can have performance penalty on the downstream learning algorithms. The algorithms heavily depends on the Wikipedia corpus (the text, title and even the structure of Wikipedia). Wikipedia is fast growing and evolving corpus. The impact of this evolution on the above feature generation methods is unknown and hard to measure or predict.

In this thesis we propose a more principled approach of feature generation using topic modeling (section 4.2). The number of new features added in this method is controlled by a parameter K where K is typically of the order of 100. The proposed method does not use any Wikipedia specific characteristics (e.g. Wikipedia titles or link structure) to generate features. It automatically extracts the topic features from the given corpus using topic modeling technique. Therefore, it can extract features from a corpus other than Wikipedia as well.

2.2.3 Gathering Training Data for Web Page Classification

Our third major contribution is automatically gathering training data to build web page classifier for any arbitrary concept. Training data generation is often manual and an expensive process. Gathering training data to build web page classifiers is even more challenging. Web pages are diverse and therefore collecting a representative sample may involve huge effort. Also in the scenario targeted in this thesis the target categories are not predefined. A user may define an arbitrary category and the system needs to automatically build a classifier. To build classifiers without involving any manual step we need to device a way to gather training examples automatically. In this thesis, we develop methods to utilize different web sources to automatically gather training data.

Surprisingly there is not much research in this direction. Most research in web page classification assumes Dmoz as the source for labeled training data [Qi and Davison, 2008, Xu et al., 2007, Xue et al., 2008]. Dmoz [UrlDmoz] is a web directory maintained by a community of users. It has 0.6 million categories and the categories are organized in a hierarchy. 4 million web pages are manually classified under these categories. Given the pages are categorized by human editors it is natural to use Dmoz as the source of training data. Even the cross validation accuracy reported in the past [Qi and Davison, 2008, Xu et al., 2007, Xue et al., 2008] is quite high. Davidov et al [Davidov et al., 2004] directly addressed the question of generating labeled training data. But they also used Dmoz as the source and reported cross validation accuracy.

But as we will see later, that the distribution of pages in Dmoz is not representative of the distribution of the pages obtained from other sources in the web. Therefore, the cross validation accuracy using Dmoz as the training as well as the test corpus does not necessarily reflect the actual accuracy of the classifier in all different types of web pages. Here we are interested in building a classifier that performs well across web pages of different sources. Therefore we need to utilize sources other than Dmoz as well. In this thesis, we use three other sources apart from Dmoz to obtain training data. The sources are Wikipedia, search engine Google and the social bookmarking site Delicious [UrlDelicious]. We develop methods to obtain training data from these sources.

In the past, there has been relatively little work in utilizing web sources like Google, Wikipedia or Delicious to generate training data. Fergus et al [Fergus et al., 2005] used the image search engine of Google to automatically gather training examples for image object category recognition. Given an image category name (e.g. "airplane"), they retrieve top few images from the image search engine of Google using the category name as the query. Then an extended version of pLSA is applied to extract K topics from the these images. Using a validation set they choose a particular topic that indicates the target category (here "airplane") of the images. A new image can be classified by estimating the probability of the selected topic in the image. This approach is unlikely to bring much success in the text domain as the data is much more noisy and constructing a good validation set in an unsupervised manner can become problematic.

Although Google, Wikipedia, Delicious sources were not used much for the purpose of training data generation, they have been used in many other information retrieval and text mining related tasks. For example, Wikipedia was used in named entity disambiguation [Cucerzan, 2007], improving the quality of search results [Milne et al., 2007], keyword extraction and word sense disambiguation [Mihalcea and Csomai, 2007], question answering [Buscaldi and Rosso, 2006] etc. Google search engine had been used for the tasks like obtaining n-gram statistics [Keller and Lapata, 2003], meaning discovery [Cilibrasi and Vitanyi, 2007]. It was shown that the "tags" available for web sites in the site like Delicious can be used to improve the quality of search results [Heymann et al., 2008, Yanbe et al., 2007].

In this thesis, we used four different sources, namely Google, Wikipedia, Dmoz and Delicious to obtain training data for an arbitrary concept. We develop methods to combine the noisy training data obtained from each individual source. There is a large volume of work in the name of semi-supervised learning that shows methods to combine labeled and unlabeled data [Blum and Mitchell, 1998, Nigam et al., 2000]. But our problem is different as we need to combine noisy labeled data obtained from heterogeneous sources.

Chapter 3

Answering Quantity Consensus Queries (QCQs)

In this chapter we give the details of the first sub-problem addressed in the thesis - answering quantity consensus queries from the web. We first introduce the notion of quantity consensus query (QCQ). We then discuss the system architecture that we have designed to respond to QCQs (Section 3.3). We then study the performance of several possible approaches to answer QCQs and establish that existing ranking algorithms are not sufficient to answer QCQs (Section 3.5). Next we propose two novel ranking algorithms to collectively score and rank the quantity intervals in respond to a QCQ (Section 3.6 and 3.7).

Apart from proposing novel and more accurate ranking algorithms we have also designed a fully functional quantity search engine operating over a multi-terabyte web crawl (Section 3.9). The designed system addresses many scalability issues that a large scale QCQ system will have to solve. Finally, we show the use of Wikipedia annotations in improving the quantity search experience (Section 3.10).

3.1 Introduction

3.1.1 Entity Search and Corroboration

Search engines are getting increasingly sophisticated in extracting and exploiting structured data from unstructured and semistructured Web pages. Most major search engines identify mentions of people, places, organizations, street addresses, ZIP codes, dates, prices, disease names, and several other types of named entities mentioned on the Web pages they crawl.

Entity search has become a standard task in the research community as well. INEX[1] features a track where the aim is to return entities that satisfy a query. The TREC enterprise track[2] includes an expert search task, an important special case of entity search. Answers to TREC-style factoid questions (TREC-QA[3]) are frequently named entities.

Approaches to entity and expert ranking include probabilistic generative models that capture relations between the query, documents, latent topics [Fang and Zhai, 2007, Balog et al., 2009], and lexical proximity between query words and candidate entities [Petkova and Croft, 2007, Cheng et al., 2007, Balog et al., 2009].

Corroboration of an entity, mentioned redundantly across multiple sites, often increases ranking accuracy and robustness [Clarke et al., 2001]. Syntactic variations ("Washington" vs. "George Washington") may exist in candidate mentions, and each mention may have a score based on query, language, topic and proximity considerations.

3.1.2 Quantity Consensus Queries (QCQs)

In this chapter we focus on *quantity search*, an important special case of entity search. A quantity may be a unitless number or have an associated unit like length, mass, temperature, currency, etc. TREC-QA[4] 2007, 2006, and 2005 have 360, 403 and 362 factoid queries, of which as many as 125, 177, and 116 queries seek quantities. As against "spot queries" seeking unique answers like date of birth, we are specifically interested in what we call *quantity consensus queries* (QCQs), where there is uncertainty about the answer quantity ("driving time from Paris to Nice" or "battery life of Lenovo X300"). TREC-QA 2007, 2006, and 2005 have at least 61, 39 and 28 such queries. To learn a reasonable distribution over the uncertain quantity, the user may need to browse thousands of pages returned by a regular search engine. A QCQ system reduces this cognitive burden by zooming down from document to snippet to quantity level. QCQ engines can also support sites that offer comparison of prices and features related to products, services and travel.

In the information extraction, integration and warehousing literature, a *curate-and-query* approach is popular; it assumes the existence of entity and relationship extractors [Agichtein

[1]`http://inex.is.informatik.uni-duisburg.de/`
[2]`http://trec.nist.gov/pubs/trec15/`
[3]`http://trec.nist.gov/data/qamain.html`
[4]`http://trec.nist.gov/data/qamain.html`

and Gravano, 2000, Bunescu and Mooney, 2005] for limited domains, which populate (possibly probabilistic) relational databases [Cafarella et al., 2007, Wu and Weld, 2007]. We argue that open-domain ad-hoc QCQs cannot leverage the curate-and-query strategy, because the queries are too diverse and the sources are too unstructured for a priori schema design or information extraction. Our hypothesis is that some combination of string-oriented IR and structured aggregation is essential at query time.

3.1.3 Our Contributions

We introduce QCQs (Section 3.2) and give novel algorithms that aggregate evidence in favor of candidate quantities and quantity intervals from *snippets* in a collective and corroborative fashion, without attempting deep NLP on snippets.

For a given query, the ith snippet is a segment of text tokens, centered around the mention of a quantity x_i. A *quantity* is a number or a range accompanied by an (optional) unit of measurement. To the left and right of the central quantity mention are other *context* tokens of the snippet.

As baseline, we first consider (Section 3.5) algorithms that learn to rank items (documents or snippets) represented as feature vectors [Joachims, 2002, Joachims et al., 2007, Yue et al., 2007] [5]. An item (here, a snippet) is usually represented as a feature vector $z_i \in \mathbb{R}^d$ in response to a query. In our case, z_i will encode the presence of query words in the snippet context, lexical proximity between query words and quantity x_i, and rarity of matched query words in the corpus (IDF). Using manually-provided snippet relevance labels $y_i \in \pm 1$, these algorithms learn a *model vector* $w \in \mathbb{R}^d$ such that the score of the ith test snippet is $w^\top z_i$, and snippets are then sorted by decreasing score. We show that scoring using z_i performs poorly, because z_i by itself is a very noisy relevance signal.

We then evaluate a recent technique [Wu and Marian, 2007] that aggregates evidence across snippets i, j only if x_i, x_j match exactly. This fails in the face of close but not identical quantities in dominant clusters. Next we adapt a graph Laplacian smoothing technique [Ko et al., 2007, Qin et al., 2008] that balances between individual snippet score evidence $w^\top z_i$ and quantity proximity, say, $|x_i - x_j|$. This formulation cannot ignore quantity proximity among irrelevant snippets, and gives only modest gains.

These trials and observations prompt us to propose (Section 3.6) new scoring mechanisms

[5] for a comprehensive list visit `http://research.microsoft.com/users/LETOR/`

for entire *intervals* of x values, instead of individual snippets, as was done in prior work. We show how to aggregate snippet scores into candidate interval scores, and then pick the best intervals. This dramatically boosts accuracy.

In Section 3.7, we give another algorithm: it represents an interval I with novel feature vectors $\hat{z}_I$, where some features are aggregated from snippet-level scores $w^\top z_i$. Note that i indexes individual snippets and I represents an interval. We use max-margin methods [Joachims, 2002] to learn a "stacked" model $\hat{w}$. During testing, $\hat{w}^\top z_I$ is used to sort candidate intervals.

Our stacked ranker further enhances accuracy compared to interval scoring using w alone. It achieves over 20% relative improvements in snippet-level MAP and NDCG compared to Laplacian smoothing, which in turn is 10–15% better than independent snippet ranking. We compare favorably with the best TREC-QA participants wrt precision-at-1. We also present a new way to evaluate sequences of quantity intervals, as against snippet lists.

Providing snippet labels y_i is more tedious than providing ground truth x_i values per query. In Section 3.8, we propose a very simple alternative to training w and $\hat{w}$ using only ground truth x_i, with a very small drop in quality.

Given the extreme diversity and noise in snippets, it is astonishing that clear and often correct consensus can be mined without the help of deep NLP, even for completely ad-hoc queries.

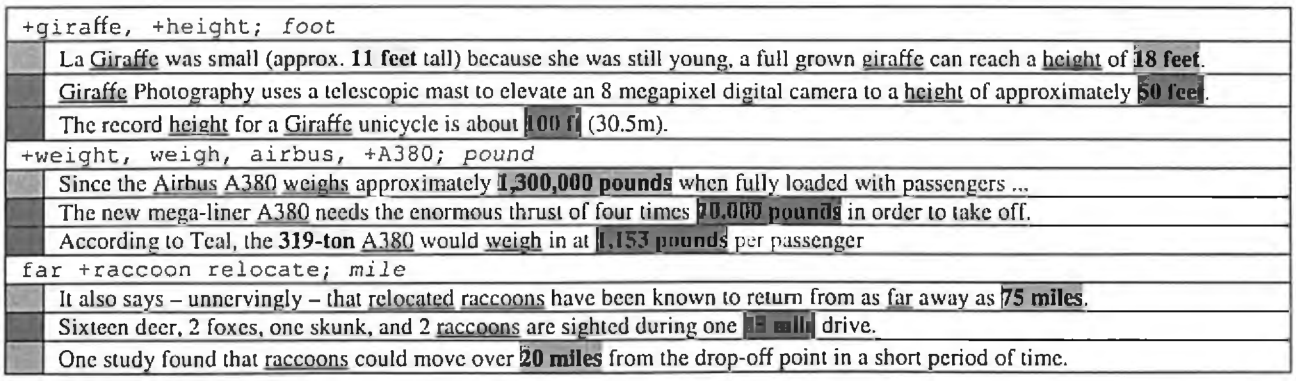
Finally, we address some scaling issues of building a quantity search engine over a multi-terabyte web corpus. We also show the use of Wikipedia in enriching the quantity search experience.

3.2 Terminology

3.2.1 Query

A QCQ has two main parts: a set of words or phrases, and a quantity type specifier. Some words or phrases may be marked compulsory with a prefixed '+'. The latter may be unitless, if a count is desired, or have an unit. Some example QCQs are shown in Figure 3.1. As with ordinary Web queries, the onus of getting better snippets, through the use of '+' and phrases, lies with the user.

A third optional component of QCQs that gives additional control is a user-defined *relative*



`+giraffe, +height;` *`foot`*	
░	La Giraffe was small (approx. **11 feet** tall) because she was still young, a full grown giraffe can reach a height of **18 feet**.
▓	Giraffe Photography uses a telescopic mast to elevate an 8 megapixel digital camera to a height of approximately **50 feet**.
▓	The record height for a Giraffe unicycle is about **100 ft** (30.5m).
`+weight, weigh, airbus, +A380;` *`pound`*	
░	Since the Airbus A380 weighs approximately **1,300,000 pounds** when fully loaded with passengers ...
▓	The new mega-liner A380 needs the enormous thrust of four times **70,000 pounds** in order to take off.
▓	According to Teal, the **319-ton** A380 would weigh in at **1,153 pounds** per passenger
`far +raccoon relocate;` *`mile`*	
░	It also says – unnervingly – that relocated raccoons have been known to return from as far away as **75 miles**.
▓	Sixteen deer, 2 foxes, one skunk, and 2 raccoons are sighted during one **[illegible] mile** drive.
░	One study found that raccoons could move over **20 miles** from the drop-off point in a short period of time.

Figure 3.1: QCQs with snippets (matches underlined; quantities in boldface, good, maybe, bad).

width parameter r, where $0 \leq r \ll 1$, meaning that the user is looking for a quantity interval $[x, x']$, such that $x' \leq (1 + r)x$, which has strong collective evidence from snippets. r is necessarily user-defined: a QCQ about Olympic record times has a fundamentally different expectation of precision compared to a QCQ about the distance between the Sun and Pluto. Only the user can provide that domain knowledge. In practice, a large number of QCQs run well with a default setting like "$r = 0.05$". In any case, r is an *upper bound* on the relative width, and our system will tighten the interval if it can.

3.2.2 Snippet (x_i, z_i)

A snippet is a suitably large window of tokens around a candidate quantity which matches the unit specified in the QCQ. A quantity scanner (Section 3.3.1) identifies token segments that express quantities. The quantity, including unit, is called x_i for the ith snippet for a given query. The surrounding text is turned into a suitable feature vector representation $z_i \in \mathbb{R}^d$ (z_i depends also on the query).

The design of z_i must consider the proximity between the central quantity mention to snippet tokens that match query tokens, and is described in detail in Section 3.3.2. Any snippet that has one or more token matches with the query is potentially a relevant snippet, and its quantity a *candidate quantity*. Some sample relevant and irrelevant snippets for the above QCQs are shown in Figure 3.1. These snippets make clear the great variety of contexts in which plausible quantities appear close to significant query words.

3.2.3 Consensus

As is clear from the examples, QCQs are characterized by the absence of an absolute or single truth. Our first impulse was to model the quantity of interest as a random variable, and build a system to return a distribution over it. But the event space is too complex: it involves natural language usage and extraction accuracy, among other uncertainties. We therefore avoid generative models for quantities, and explore discriminative, collective ranking techniques for snippets. Informally, a *consensus interval* is a tight range $[x, x']$ of quantities that enjoys strong collective support from high-scoring snippets. We will give more precise proposals in Sections 3.6 and 3.7. There, we will see that this simple notion of consensus performs very well.

To be fair, consensus is not the only form of useful aggregation; in some cases, it may be limiting or misleading. E.g., plutonium has multiple isotopes with diverse half lives, and a name may refer to many people with diverse birth years. Our QCQ system performs reasonably despite such ambiguity, because it reports (snippets from) not one but a number of top-scoring x-intervals. Time-variant quantities offer another challenge. E.g., the QCQ `+"bill gates", assets, worth;` `USD` may give an outdated answer, depending on Web coverage. A complete solution would require "carbon dating" each snippet, which appears even more challenging than reliable timestamping of whole Web pages. The causes of multi-valued answers have been analyzed in some detail [Moriceau, 2006, Prager et al., 2007].

3.3 QCQ System

The QCQ system architecture is shown in the Figure 3.2. The functionality of different stages are as follows:

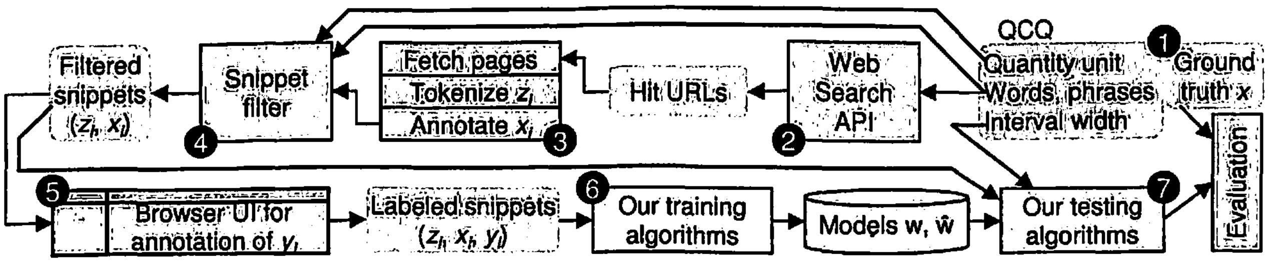


Figure 3.2: Sketch of our QCQ system prototype. Processing stages are numbered from 1 onward.

Stage 1 The input to the system is a QCQ. As mentioned in the previous section, a QCQ consists of query keywords and/or phrases, desired unit type and an interval width parameter (r). For the purpose of evaluation, each QCQ (q) in our testbed comes with a ground truth quantity set (X^q). The ground truth quantity set of a QCQ are the set of correct correct quantities for the QCQ.

Stage 2 & 3 The query keywords and phrases are sent to a search engine and a set of pages corresponding to the top matched URLs are fetched. Each page is then tokenized and the quantity mentions in the page are annotated by a quantity scanner. The quantity scanner detects the text segments that are likely to be quantity mentions and also identifies the type of the quantity. For example, text segments like "15 km" or "fifteen to sixteen kilometers" will be detected as quantities of type "kilometer". We will discuss the details of the quantity scanner shortly. Once the quantity mentions are detected, the snippets for the given QCQ are generated by taking a suitable window of token around the quantities with desired unit type. The suitable window is defined either by fixed number of tokens or the sentence boundary whichever is shorter. Some example snippets are shown in the Figure 3.1.

In this particular implementation we took help from web search engine to get a set of pages from where the snippets are extracted. Later in this chapter we will discuss the scaled version of this system where the snippets are extracted from our own corpus of 500 million pages.

Stage 4 We filter the snippets that does not contain at least one query word/phrase because those snippets are unlikely to be relevant to the QCQ. At this stage we have set of snippets for the given QCQ. Each snippet s_i is a window of tokens containing a quantity x_i of the desired unit type and some query words/phrases. A snippet (s_i) is then represented using a feature vector (z_i). The features of a snippet is extracted using both the query as well the snippet. We will describe the features of the snippets shortly.

Stage 5 & 6 For the purpose of training and evaluation we need ground truth relevance labels of the snippets. To collect ground truth relevance, labels we developed a browser based GUI where manual relevance feedback can be provided against the snippets of a QCQ. Using this browser GUI we collected manual feedback on a set of snippets of a set of sample QCQs. The details of this labeled data set is described in the section 3.4. This labeled data set is used to train a model (w or $\hat{w}$) and to evaluate different algorithms.

Stage 7 Given a test QCQ we follow stage 1 to stage 4 to generate the snippets of the test QCQ. We then apply the learned model of stage 6 to rank the snippets of the QCQ. The ranked list of snippets are then presented to the user or used to evaluate the ranking algorithm. As we will see later that our algorithms can output a ranked list of quantity intervals not just ranked list of snippets.

Next we describe the quantity scanner that annotates a text document with quantity occurrences (Section 3.3.1) and the design of feature vector of a snippet (Section 3.3.2).

3.3.1 Quantity Scanner for Annotating x_i

A *quantity scanner* annotates character spans that are likely to be quantity mentions, which come in diverse forms. Some have unit prefixes, like currency symbols. Some have unit suffixes, like scientific measures. Some have exponent modifiers, like "10 million liters" or "€50 million". Units are expressed diversely, e.g., '$' vs. USD, 'm' vs. meter vs. metre. Even the numerals are written in diverse styles. Scientific quantities may be written without commas, commas after every third digit, or at irregular spacing, as in "Rs 1,20,000". There may be spurious spaces before or after commas. Periods may end sentences or be decimal points. Very large or small quantities may be written in mantissa-exponent form. Small numerals like 1, 2, 30 may be written as words. 1889 might be a unitless count or a year. '$' may indicate different currencies. x_i may also be a range, e.g., **10–20 feet.**

We used the rule-based JAPE engine, which is part of the well-known GATE NLP package (`http://gate.ac.uk/`). We compiled about 150 rules covering mass, mileage, power, speed, density, volume, area, money, time duration, time epoch, temperature length and so on. Augmenting our rule base to capture more types of quantities should be straightforward. Manual spot checks on our annotator led to estimates of precision, recall and F1 as 0.92, 0.97, 0.95. Luckily, ranking intervals using consensus is robust to this small rate of scanner glitches.

Unit normalization: In the example QCQs above, each query has an associated specific unit (unless the answer is a count). In a deployed system, more generic units should be allowed, such as *length* in place of *mile* or *km*, or *time interval* in place of *hour* or *year*. This would also assist collecting consensus across candidate quantities expressed in different units. Our prototype does not handle this issue, except identifying different standard forms of a unit (e.g. foot, feet, ft), but it can be added on easily.

3.3.2 Feature Vector Design for z_i

We defined two families of features on (the query and) snippet text: first, standard vector-space ranking features [Liu et al., 2007, Liu, 2008], and second, features that encode lexical proximity between query word matches and quantity tokens [Petkova and Croft, 2007, Cheng et al., 2007, Balog et al., 2009].

Standard ranking features

Each snippet was characterized by the tokens in five *fields* F: snippet, a window of 10 sentences above and below the snippet, the text of the page from where the snippet is originated, the HTML title of the page, and the URL of the page. For each of the five fields F, three features were added to feature vector z_i:

TFSum: $\sum_{t \in q \cap F} \text{TF}(t, F)$

IDFSum: $\sum_{t \in q \cap F} \text{IDF}(t)$

TFIDFSum: $\sum_{t \in q \cap F} \text{TF}(t, F)\text{IDF}(t)$

TF(t, F) is the term frequency of t in F and IDF(t) is the standard IDF of t with respect to a reference corpus (union of all documents over all queries). In addition, we used:

- Jaccard similarity[6] between query and snippet tokens. Jaccard similarity between two sets (A and B) are defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- Number of tokens in the snippet.

Lexical proximity features

Guided by the work on locality or proximity based ranking [de Kretser and Moffat, 1999, Cheng et al., 2007, Petkova and Croft, 2007, Balog et al., 2009], we defined the *proximity* between the mention of quantity x_i and a query token match t in its vicinity as the *reciprocal of the number of tokens between the mention of* x_i *and* t (zero if no t exists).

Queries have a variable number of tokens. Therefore we define four proximity features aggregated over query tokens:

[6]`http://en.wikipedia.org/wiki/Jaccard_index`

- Maximum proximity of x_i to any query token.
- Proximity of x_i to the rarest (largest IDF) query token.
- Proximity of x_i to the smallest IDF query token.
- IDF-weighted average of proximity to all query tokens.

The weights in w corresponding to these proximity features were among the highest when w was learnt using RANKSVM [Joachims, 2002]. To keep our system robust and scalable, we avoided deeper NLP techniques like learning to spot relations from dependency parse trees.

Altogether, we used 21 features: 4 proximity, 5×3 similarity features and 2 other features.

3.4 Testbed

3.4.1 QCQs with ground truth

We collected 162 QCQs from diverse sources. Some examples of QCQs used in the experiments here are shown in Appendix A. Each QCQ q was collected along with ground a truth quantity set X^q. Most X^qs contained multiple values or ranges. Unless noted otherwise, we report performance on the union of these QCQs.

Infobox: We created 40 QCQs by sampling Wikipedia Infoboxes for numeric attributes of Wikipedia entities.

TREC-QA: We chose TREC-QA queries that had non-unique quantity answers: 16 from TREC-QA 2004 and 61 from TREC-QA 2007.

Misc.: 9 queries were contributed by W&M [Wu and Marian, 2007]. 36 QCQs were contributed by volunteers, who found ground truth X^q through careful Web search.

Growing our QCQ set is limited only by snippet-labeling effort (described next).

3.4.2 Snippet Label y_i Collection

We used Web search APIs to collect snippets (stage 2 & 3 in the QCQ system architecture, Figure 3.2). Unlike QA-oriented text indices, major Web search APIs do not allow us to ask for documents containing, say, a distance in feet within 20 tokens of the word *elephant*. This necessitated a two-step filtering approach. In the first step, we sent words, phrases and unit names in the QCQ to the engine. Response URLs were fetched, tokenized, and quantities annotated. Quantities that matched the QCQ unit, and were within one sentence (or a maximum token window) from a query word were retained, with their snippet context.

For training and evaluation, a selection of 100 snippets per QCQ were presented, using a browser-based GUI, for manual labeling of $y_i \in \pm 1$, the relevance of snippet i. Six volunteers, including the author, annotated the snippets. There were (infrequent) inconsistencies between the contributed answer quantities and y_i labels. I.e., snippets with quantities not in the ground truth ranges were sometimes marked relevant, mostly because the Web has a more up-to-date ground truth. We did not attempt to make these consistent, insisting that a robust algorithm must take this in stride. As a whole we have provided manual relevance labels to about 15,000 snippets of 162 queries. We have made this data available in the public domain[7]. Unless specified otherwise the experiments in this chapter are conducted in this data set.

3.4.3 Response and Comparative Evaluation

QCQ systems may return a ranked list of snippets, with the quantities highlighted (Figure 3.1). The advantage is that the user can glance over and judge the snippets directly. Traditional criteria [Liu, 2008], such as Mean Average Precision (MAP) or Normalized Discounted Cumulative Gain (NDCG) can then be used directly. (Mean Reciprocal Rank or MRR is not appropriate for QCQs because it does not give credit for comprehensive coverage of consensus values.)

Alternatively, to display many promising quantities within scarce real estate, QCQ systems may report a list of x-intervals, each subject to the user-provided relative width constraint. Evidence snippets can be shown if an interval is clicked. Evaluating a list of intervals, or comparing a system that ranks snippets with one that ranks intervals, are new challenges. We will discuss these in Section 3.7.4

3.5 Prior approaches and insights

We describe existing approaches that can be adapted for QCQs, culminating in a comparison shown in Figure 3.5.

[7]can be download from `http://www.cse.iitb.ac.in/soumen/doc/QCQ/`

3.5.1 Using Web search directly

The minimal baseline (that any useful QCQ system must beat) is to send QCQ words/phrases to a search engine, get the top snippets, scan them for qualifying quantities with proper units, and list them if they appear within a stipulated distance of at least one query token. A listed quantity x is judged correct if it matches (or is contained by) a ground truth quantity (or interval).

Such snippet-level evaluation gives very poor MAP and NDCG (below 0.15), partly because search engines have no mechanism to promote to top ranks those snippets that contain quantities of specified types and query words. We can be generous and give credit for unsupported but correct quantities anywhere on the pages (not just reported snippets), which is what we show in Figure 3.5. We use two major engines (called Web1 and Web2). Our algorithms are better at promoting relevant snippets to top positions, comfortably beating the generous evaluation of Web search engines.

3.5.2 Snippet-level RANKSVM

We tried several techniques for learning [Joachims, 2002, Yue et al., 2007, Liu, 2008] a snippet-level w given snippets (z_i, y_i) (x_i is ignored here), with the score $w^\top z_i$ used for ranking snippets. We found standard pairwise RANKSVM [Joachims, 2002] (formulation given below) as good as direct optimizers of MAP [Yue et al., 2007] or NDCG [Chapelle et al., 2007].

$$\min_{w,\xi} \tfrac{1}{2} w^\top w + C \sum_{i:y_i=1} \sum_{j:y_j=-1} \xi_{ij} \quad \text{subject to} \tag{3.1}$$

$$\forall i \text{ s.t. } y_i = 1, \forall j \text{ s.t. } y_j = -1, \begin{cases} \xi_{ij} \geq 0 \\ w^\top z_i + \xi_{ij} \geq w^\top z_j + 1 \end{cases}$$

$\sum_{i,j} \xi_{ij}$ upper bounds the number of pair preferences violated and C balances between violations and $|w|$. Details of RANKSVM algorithm is mentioned in Section 2.1.4.

Figure 3.5 compares the accuracy of various baseline algorithms. (For all RANKSVM-style learning algorithms in this paper, five-fold cross validation was used and the best value of C in (3.1) was picked from among $\{10^{-3}, 10^{-2}, .1, 1, 10\}$.). Figure 3.5 shows that RANKSVM is generally better than Web1 and Web2. As for MAP, remember that Web1 and Web2 are given massive advantage while RANKSVM snippets are evaluated stringently. However, a closer look at RANKSVM (next) provides key actionable insight.

3.5.3 Vertical Bands in $w^\top z_i$ vs. x_i Scatter

RANKSVM considers only $w^\top z_i$ scores, but how do these relate to corresponding x_is? Figure 3.3 plots scatters of $w^\top z_i$ (y-axis) against x_i (x-axis) for three representative queries. For visual uniformity across queries, both axes have been scaled to $[0, 1]$. Snippets are also called "points". If optimization (3.1) were perfect, all good points would lie higher along the y-axis than all bad points. This is rarely the case: although z_i was designed with considerable care, decent separation between relevant and irrelevant snippets is never achieved on the basis of $w^\top z_i$ alone.

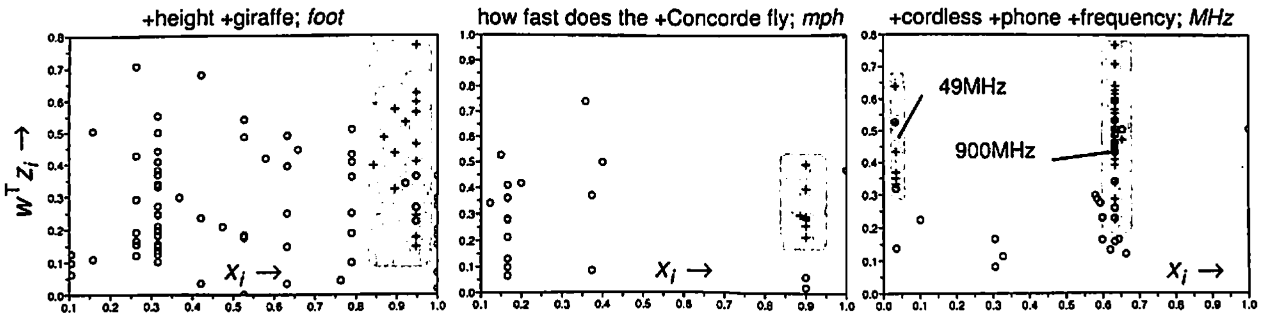


Figure 3.3: Scatter of $w^\top z_i$ against x_i for representative QCQs; relevant (irrelevant) points marked '+' ('o').

But the scatter plots also show a valuable clue: *relevant points often cluster in vertical bands.* From Figure 3.3 and the simplified sketch in Figure 3.4, it seems that for each query, one or few *upward-open rectangular strips* capture most good snippets with very few bad snippets.

It is natural to ask at this point why we cannot use decision trees (which naturally find rectangle discriminators) or SVMs with nonlinear kernels. The reason is that the width, location and even the number of semi-open rectangles (equivalently, parameters of non-linear kernels) change from query to query. Parameters learnt by decision trees or nonlinear SVMs will not generalize across diverse queries. We need a more non-parametric approach.

3.5.4 Wu and Marian's System (W&M)

A first approach to integrating x from snippets is to take weighted voting, similar to exploiting redundancy in QA. W&M accumulates a score for each distinct x from snippets where x occurs. The snippet score is determined by the following considerations:

- It decays geometrically with the rank assigned by the search engine to the source page.

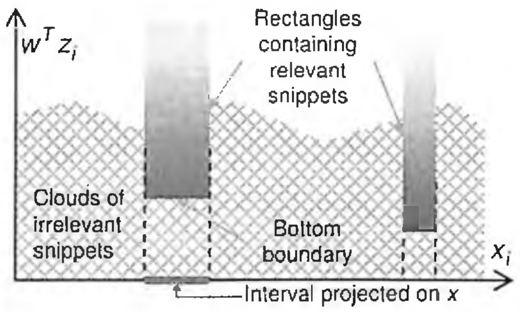

$w^T z_i$
Rectangles containing relevant snippets
Clouds of irrelevant snippets
Bottom boundary
x_i
Interval projected on x

Figure 3.4: Our proposed "hypothesis class" of semi-open rectangles.

- It decreases reciprocally with the number of candidate quantities on the source page.
- It decreases exponentially with the number of duplicate/mirror pages and pages from the same domain. (Search engines already enhance diversity and eliminate duplicates, so this rarely fires.)
- It decreases reciprocally to the shortest distance between the quantity and a query token (lexical proximity).

Score aggregation happens only on exact equality of x. Figure 3.5 shows that W&M is consistently worse than RANKSVM. Often, relevant snippets are found at quite poor ranks, because the whole-page ranking imposed by Web1 and Web2 are often not suited for QCQs. Recall again that Web1 and Web2's accuracy may be substantial overestimates.

3.5.5 Laplacian Smoothing

A second way to combine x_i and $w^\top z_i$ is via a graph Laplacian approach [Qin et al., 2008]. Each snippet is made a node in a graph $G = (V, E)$. Each node/snippet has an associated feature vector z_i as before, inducing a (noisy) *local score* $w^\top z_i$. Meanwhile, the x_i values at nodes are used to define edges weights $R(i, j)$, inversely related to $|x_i - x_j|$.

The formulation seeks a model w while assessing a loss $(f_i - w^\top z_i)^2$ for deviations between *final scores* f_i and local scores, and a roughness loss $\sum_{\{i,j\}\in E} R(i,j)(f_i - f_j)^2$, where $f \in \mathbb{R}^{n\times 1}$ is the column vector of final scores. Finally, there is the usual training loss if the final score of a good snippet is less than the final score of a bad snippet. The training loss is expressed using standard hinge loss $\sum_{g,b} \max\{0, 1 + f_b - f_g\} \geq \sum_{g,b}[\![f_g \leq f_b]\!]$. Here f_a and f_b denotes the scores of good (relevant) and bad (irrelevant) snippets. The weight vector w is learnt by minimizing these three loss functions.

Minimizing this three loss functions turns out to be a quadratic program with linear constraints [Qin et al., 2008]. During testing or deployment we have the learned model w. So the final scores of the snippets of the test query can be computed by minimizing the deviation and roughness losses. The snippets are then ranked in the decreasing order of their final scores.

The design of edge weights R critically determines the algorithm, but there is no generic guideline. We tried the following reasonable definitions:

$x_i = x_j$ **equality:** Following W&M's weighted voting semantics, we define $R(i,j) = 1$ if $x_i = x_j$ and 0 otherwise.

$|x_i - x_j|$ **distance:** $R(i,j) = \max\left\{0, 1 - \frac{|x_i - x_j|}{|x_i| + |x_j|}\right\}$

$|x_i - x_j|$ **decay:** $R(i,j)$ is defined as $\exp(-s\|x_i - x_j\|)$ or $\exp(-s(x_i - x_j)^2)$, where s is a tuned **spread parameter** (inverse variance).

Snippet cosine: Following the pseudorelevance feedback [Qin et al., 2008] setting, we ignore x_i, x_j and use cosine similarity between the text of snippets i and j as $R(i,j)$. Snippet text is represented as a binary vector over token space. The intuition is that if snippet texts for i and j are similar, they should have similar score.

		MAP	NDCG@1	NDCG@5	NDCG@10
	Web1	0.375	0.338	0.362	0.380
	Web2	0.350	0.413	0.357	0.377
	RankSVM	0.369	**0.450**	0.412	0.406
	W&M	0.306	0.247	0.303	0.322
Laplacian	Equality	0.384	0.369	0.353	0.382
	Distance	0.407	0.413	0.401	0.420
	Decay	**0.421**	0.433	**0.422**	**0.435**
	Cosine	0.375	0.438	0.396	0.405

Figure 3.5: Initial results (bold $\Longrightarrow$ max in column).

Figure 3.5 summarizes accuracies of all approaches discussed thus far. Laplacian smoothing with the "decay" option gives modest gains over Web1, Web2, RANKSVM, and W&M. The gains are limited by two factors. First, the Laplacian formulation assesses the roughness penalty on *all* edges, even those between snippets putatively labeled irrelevant. For QCQ, we should favor smoothness of f_i only among relevant snippets. Second, there is no ready way to tune the width parameter s reliably across diverse queries and associated quantities. Our algorithms get around these issues.

```
1: **inputs:** snippet set $S$ with $x_i$ and $w^\top z_i$ values, interval width tolerance parameter $r$
2: sort snippets $S$ in increasing $x_i$ order
3: **for** $i = 1, \ldots, n$ **do**
4:   **for** $j = i, \ldots, n$ **do**
5:     **if** $x_j < (1+r)x_i$ **then**
6:       let $I = [x_i, x_j]$
7:       *merit* $\leftarrow$ *GetIntervalMerit*$(S, I)$
8:       maintain intervals with top-$k$ merit values
9: **for** surviving intervals $I$ in decreasing merit order **do**
10:   present snippets in $I$ in decreasing $w^\top z_i$ order
```

Figure 3.6: Interval merit enumeration.

3.6 Listing and Scoring Intervals

Instead of scoring and ranking snippets, we shift our focus to quickly enumerating and scoring rectangular regions as shown in Figure 3.4. We begin with searching for the position and width of a promising rectangle on the x-axis, i.e., searching over intervals $I = [x, x']$, with $x' \le (1+r)x$ as specified in Section 3.2.1. We will overload I to also mean a *set* of snippets. A snippet $s_i = (x_i, z_i)$ is said to belong to I if $x_i \in I$. In case a snippet mentions a range (such as **10–20 feet**), the snippet belongs to I if the range is contained in I.

For a query q with n_q snippets, there are at most $\binom{n_q+1}{2}$ functionally distinct (in terms of the snippets they contain) intervals on the x axis. Some of these intervals $I = (x, x')$ are too wide ($x' > (1+r)x$) and can be discarded. Usually $r \ll 1$, so the enumeration of valid candidates $I \in \mathcal{I}_r$ can be done efficiently using a left-to-right sweep that takes close to linear time in practice. For simplicity Figure 3.6 shows a naive $O(n_q^2)$ enumeration of intervals.

Figure 3.4 suggests that we should also search over all possible bottom boundaries of I. In practice, this makes negligible difference. Our results in Section 3.8 may explain why this is the case.

3.6.1 Merit Functions *GetIntervalMerit*(S, I)

As we enumerate over intervals I, we need to use the signal from $w^\top z_i$ for $i \in I$ and potentially $i \notin I$, to evaluate *GetIntervalMerit*(S, I). If there is any useful signal in $w^\top z_i$, we should prefer intervals I such that points in I have generally larger values of $w^\top z_i$ than points not in I. Accordingly, we provide three choices of merit (to maximize over I):

$$\sum_{i:x_i \in I} w^\top z_i \tag{Sum}$$

$$\sum_{i \in I} \sum_{j \notin I} (w^\top z_i - w^\top z_j) \tag{Diff}$$

$$\sum_{i \in I} \sum_{j \notin I} \max\left\{0, w^\top z_i - w^\top z_j\right\} \tag{Hinge}$$

Observe that terms in (Diff) can be positive or negative; favorable and unfavorable score pairs can cancel out. This is prevented in (Hinge). In machine learning, one *minimizes* hinge *loss* rather than *maximize* hinge *gain*, but in QCQ, the former leads to tiny proposed relevant clusters that are often incorrect.

3.6.2 Snippet-level Evaluation Experiments

We compare three algorithms: the best two approaches from Figure 3.5 (RANKSVM and Laplacian Decay) and interval merit enumeration (for which the snippet-level model w was trained using RANKSVM). For MAP (Figure 3.7), we vary interval width tolerance r (shown as a percentage). For NDCG (Figure 3.8) we hold $r = 8\%$ and report NDCG at ranks $1 \ldots 10$. Note that $r = 0$ means an interval of width zero, but this can contain multiple snippets if they mention the exact same quantity. RANKSVM and Laplacian Decay do not depend on r.

Interval merit beats all baselines First, interval enumeration with (Diff) beats all other approaches by a wide margin. Interval enumeration with (Hinge) is second, still beating all others.

Effect of r (Diff) and (Hinge) show significant boost in accuracy as r is increased beyond 0. (Diff) is stable between $r = 3\%$ and 9%. This is direct evidence that robust aggregation over x_i values is critical to success.

(Diff) **better than** (Hinge) Occasionally, avoiding deep NLP leads to systematic pollution from irrelevant but dense intervals. E.g., for the QCQ `+giraffe +height: foot`, an irrelevant cluster (as per predominant human interpretation) develops around 6 feet thanks to

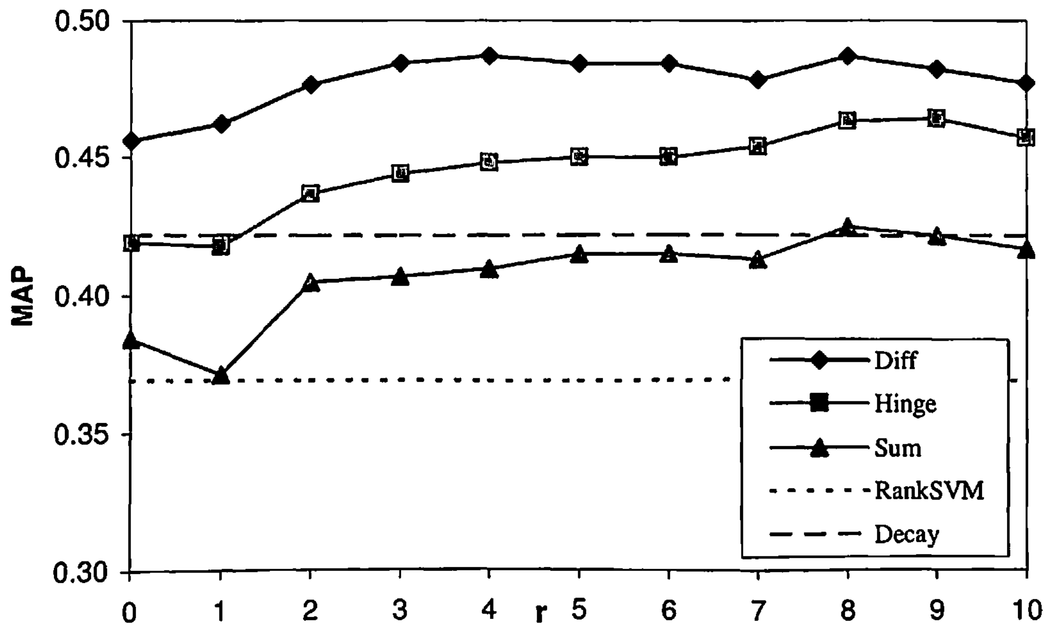


Figure 3.7: Interval merit evaluation (MAP).

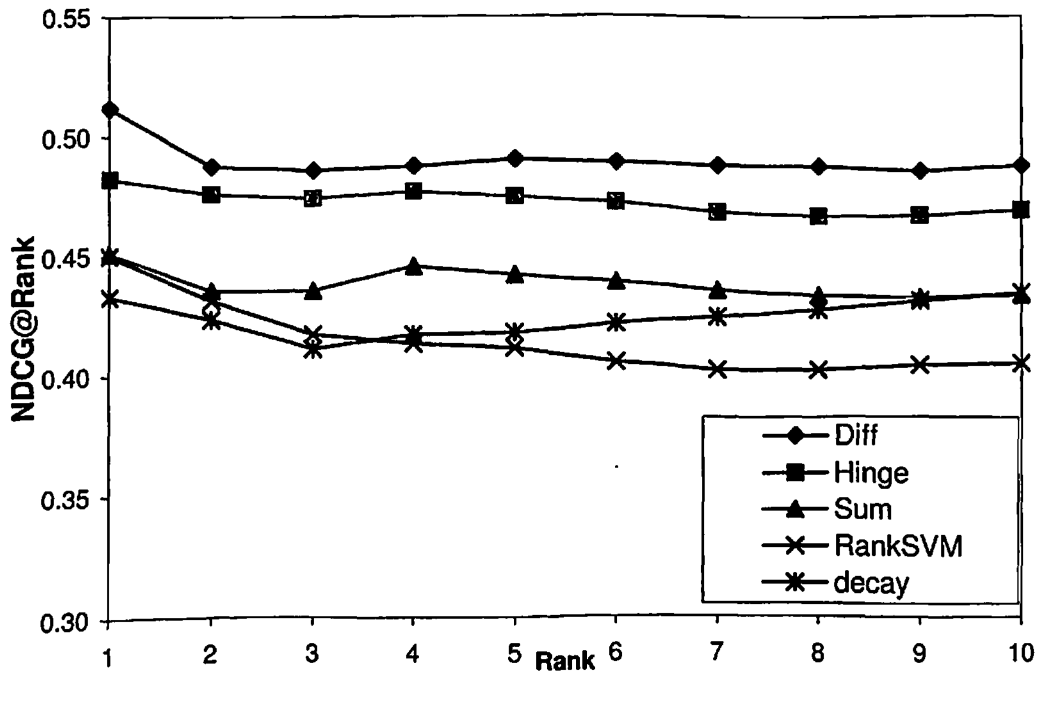


Figure 3.8: Interval merit evaluation (NDCG).

snippets like this: "newborn *giraffe* calves begin their lives by falling from a *height* of **6 feet**", "A young *giraffe* has to survive a fall of **six feet**", or "A *giraffe*'s legs alone are taller than many humans—about **6 feet**". These intervals have a lower average $w^{\top} z_i$ and (Diff) reveals this better than (Hinge).

3.7 Learning to Rank Intervals

In the previous section we proposed a way to score intervals, based on aggregating $w^\top z_i$ scores of snippets inside and outside the intervals. In this section we design a learner that directly learns to rank intervals instead of individual snippets.

As in Section 3.6, we will use relative tolerance r to define $\mathcal{I}_r$, the set of candidate intervals satisfying r. We already know that $|\mathcal{I}_r| = O(n_q^2)$.

Every candidate interval $I \in \mathcal{I}_r$ will be represented by an *interval feature vector* $\hat{z}_I$. The interval ranker will learn a corresponding scoring model vector $\hat{w}$.

3.7.1 Interval Features $\hat{z}_I$

Unlike in snippet-level RankSVM, we are at liberty to define *collective* features of intervals, rather than just aggregate $\{z_i : x_i \in I\}$, in simple ways as in Section 3.6. Specifically, a simple average of feature vectors may fail to capture certain significant clustering in the z_i space. There may be much stronger clues to guess how good an interval is.

For example, an interval is good if most of the points in the interval are relevant to the query, if the interval has high merit (as defined in Section 3.6.1) and most of the points in the interval have consensus on a quantity or there are relatively few distinct quantities. We capture these clues by designing a set of additional features that are collective across an interval. We call them *interval features*:

1. Whether all snippets in I contain some query word
2. Whether all snippets in I contain the minimum IDF query word
3. Whether all snippets in I contain the maximum IDF query word
4. Number of distinct words found in snippets in I
5. Number of words that occur in all snippets in I
6. One minus the number of distinct quantities mentioned in snippets in I, divided by $|I|$
7. Number of snippets in I, divided by n_q
8. Three features corresponding to the three merit functions defined in Section 3.6.1, which require w to compute.

Apart from the above interval features we also append to $\hat{z}_I$ the vector average of the feature vectors z_i with $i \in I$.

3.7.2 Interval Relevance and Preferences

Recall that we want to learn to compare intervals, but our ground truth y_i is collected over snippets. The next piece is to define a relevance score over each interval $I \in \mathcal{I}_r$. We assign a relevance score to an interval I based on the fraction of relevant snippets in I. I.e., if I has n_I^+ relevant snippets and n_I snippets overall, then its relevance score is defined as n_I^+/n_I. Thus, snippet-level y_i labels determine the relevance score of intervals.

For two intervals I and I', if the relevance score of I is larger than that of I', we assert a pairwise preference $I \succ I'$ between the intervals. These interval comparisons will replace individual snippet comparisons in (3.1). (Other algorithms [Yue et al., 2007, Chapelle et al., 2007, Liu, 2008] may be used in place of RANKSVM.)

```
1: inputs: snippets $s_i$ with labels $y_i$, tolerance $r$
2: for each interval $I \in \mathcal{I}_r$ do
3:     compute the relevance of $I$ using snippet labels $y_i$
4:     compute feature vector $\hat{z}_I$
5: generate interval pair preferences $I \succ I'$
6: set up a RANKSVM problem involving intervals:
```

$$\min_{\hat{w},\hat{\xi}} \tfrac{1}{2}\hat{w}^\top\hat{w} + C\sum_{I\succ I'}\hat{\xi}_{I,I'} \quad \text{s.t.} \qquad \text{(IntervalRank)}$$

$$\forall I \succ I': \quad \hat{w}^\top\hat{z}_I - \hat{w}^\top\hat{z}_{I'} \geq 1 - \hat{\xi}_{I,I'}; \quad \hat{\xi}_{I,I'} \geq 0$$

```
7: train using RANKSVM to get $\hat{w}$
8: return $\hat{w}$
```

Figure 3.9: Interval training algorithm.

Initial experience with the algorithm shown in Figure 3.9 suggested that we were generating too many preference pair constraints based on insignificant interval relevance differences. We improved both training speed and accuracy by discretizing interval relevance to an ordinal scale of 0–10. In other words, the relevance of an interval was defined as $\lfloor 10n_I^+/n_I \rfloor$. We tried between 5 and 10 ordinal levels and the accuracy was not very sensitive to the number of levels.

Suppose the interval ranker learns model $\hat{w}$. Given a test query, $\mathcal{I}_r$ is enumerated as before. Then the intervals in $\mathcal{I}_r$ are ranked by decreasing $\hat{w}^\top \hat{z}_I$. If a snippet list must be provided, we run down the intervals in decreasing $\hat{w}^\top \hat{z}_I$ order, and order snippets within each interval using snippet score $w^\top z_i$.

Results We compare the best algorithm from Section 3.6, viz., (Diff) merit score for intervals, against the (IntervalRank) algorithm presented in this section.

Figure 3.10 compares MAP obtained by IntervalRank vs. Diff as width tolerance r is varied. IntervalRank is better, reaching a MAP of 0.511 against 0.421 by Laplacian smoothing and 0.369 by RANKSVM. The story with NDCG (Figure 3.11) is almost similar, the gains increasing with rank. IntervalRank achieves NDCG@10 of 0.513 against 0.435 by Laplacian smoothing and 0.406 by RANKSVM. The improvement in accuracy in IntervalRank is due to the fact that it scores an interval based on various features and using a learned model. Whereas in Diff the intervals are scored using a single pre-defined merit function.

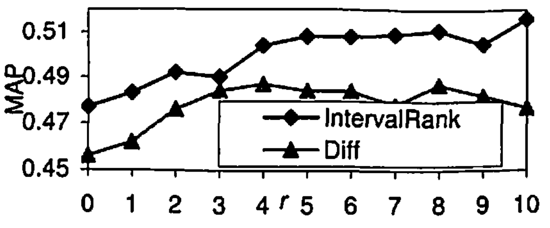


Figure 3.10: Comparison of Merit-Diff and interval ranking algorithms (MAP).

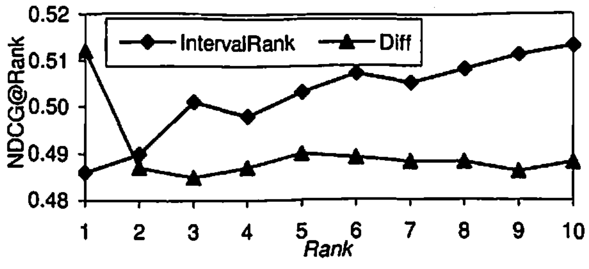


Figure 3.11: Comparison of Merit-Diff and interval ranking algorithms (NDCG).

We did an ablation study by removing one feature from all $\hat{z}_I$ at a time. The maximum MAP reduction was for feature #6 (one minus number of distinct quantities mentioned in the interval). This shows that quantity consensus is an important feature of the interval.

3.7.3 Comparison with TREC-QA Participants

We have used quantity queries of the question answering track of TREC (of the year 2004 and 2007) in our experiments. From the TREC website we obtained the performance of the participants of the QA track. Direct comparison is not possible as we used web to extract the answers whereas the corpus used in TREC was different.

TREC QA track uses precision-at-1 as the evaluation measure. So we just compare the precision-at-1 value of our IntervalRank algorithm against the precision-at-1 achieved by TREC participants on our sample of quantity queries of those two tracks. For our sample queries of TREC-QA 2007, we are second-best and for our sample queries of TREC-QA 2004, we are at rank 5 out of 63 teams. While not very meaningful for QCQ, this shows that our system is competitive wrt precision-at-1.

3.7.4 Interval-oriented Evaluation

Our algorithms rank intervals, but to evaluate them wrt snippet-level y_i ground truth, we iterated through intervals by decreasing $\hat{w}^\top z_I$, listing snippets $i \in I$ by decreasing $w^\top z_i$. Snippet-level NDCG or MAP is suitable when users inspect snippet lists [Robertson, 2008]. If a QCQ system presents a list of intervals, the user may inspect at most a small number of evidence snippets per interval, so snippet-level MAP or NDCG may not accurately reflect cognitive burden. We propose recall and precision criteria that recognize an interval, not a snippet, as a unit of attention. Suppose there are n^+ snippets marked relevant for a QCQ, and our algorithm A outputs $I_1, \ldots, I_m$, where I_j contains n_j snippets, of which k_j are good. The *interval-oriented precision* of A at interval rank j is defined as $(k_1 + \cdots + k_j)/(n_1 + \cdots + n_j)$. The *interval-oriented recall* is defined as $(k_1 + \cdots + k_j)/n^+$. To compare with a snippet-listing algorithm A' we simply line up the first $n_1 + \cdots + n_j$ snippets, assume that A' reported intervals $I'_1, \ldots, I'_m$, and evaluate similar to $I_1, \ldots, I_m$. Note that IntervalRank cannot cheat at recall using arbitrarily large r, because precision will plummet. Results in Table 3.1 show that collective interval scoring and presentation can increase both recall and precision, particularly for the top few intervals. Laplacian decay is between RANKSVM and IntervalRank.

	# Intervals →				
Algo, measure	1	2	3	4	5
IntervalRank recall	0.521	0.581	0.637	0.647	0.685
Lapl. decay recall	0.510	0.569	0.614	0.634	0.655
RANKSVM recall	0.458	0.514	0.554	0.596	0.618
IntervalRank prec	0.443	0.432	0.416	0.388	0.371
Lapl. decay prec	0.382	0.367	0.350	0.330	0.316
RANKSVM prec	0.330	0.312	0.298	0.294	0.284

Table 3.1: Interval-oriented evaluation.

3.8 Quantity-imputed Labeling

We have assumed throughout that the label y_i is known for each training snippet ("complete" supervision). However, it is much more natural and efficient to train a QCQ system based on ground truth quantity set X^q and z_i. Another advantage of this form of "partial" training is that we can semi-automatically glean training data from social media, such as Wikipedia Infoboxes.

Suppose we sloppily impute y_i values using X^q: any snippet with $x_i \in X^q$, or contained in a range in X^q, is considered relevant. These imputed $\tilde{y}_i$s may conflict with "true" y_is (if available). How drastically might $w, \hat{w}$ deteriorate because of using $\tilde{y}_i$s to train our system in place of y_is?

We sampled queries leading to 14,562 y_i-labeled snippets. $\tilde{y}_i$ gave only 571 false positives and 395 false negatives. These modest fractions may explain why modeling the bottom boundary of rectangles in Figure 3.4 did not make a significant difference. Table 3.2 shows the effect of imputed training on test MAP score. The drop in test accuracy is very mild. Our algorithm continues to beat all baselines.

	y_i known	y_i imputed
RANKSVM	0.369	0.361
Merit-Diff	0.487	0.475
IntervalRank	0.511	0.480

Table 3.2: Effect of imputation on test MAP.

3.9 Scaling QCQ to Terabyte Corpus

So far in this chapter we used a search engine to get a set of pages for a QCQ (stage 2 in the Figure 3.2). The snippets for the QCQ were extracted from those set of pages. The major limitation in using a search engine is that it does not support quantity type as a query term. For example, web search engines does not allow us to ask for documents containing the terms "price", "Canon" or "camera" within a distance of 20 tokens from a quantity of type USD. Another limitation of using search engine is the performance issue. Given a QCQ we need to fetch the top result pages. It takes substantial time even to download top few (say 500) pages.

To support more precise query language and to build a live quantity search engine we have build our own index of 500 million pages. The index supports querying for quantity types as well. Having our on index also makes it possible to respond to queries in real time. In the implemented live QCQ system a user can enter a QCQ through a search box and get a list of quantity intervals (reported by our algorithms described above) as the response. The quantity intervals can be expanded to view the supporting snippets (Figure 3.12).

We had to address several engineering challenges in building such a large scale system.

Quantity Scanner The JAPE rule based quantity scanner described in the Section 3.3.1 has a throughput of 10 pages per second. Annotating 500 million pages with that throughput is clearly unacceptable even after reasonable parallelization.

Distributed Search Index of this huge corpus has to be hosted in multiple machines. Therefore we had to create multiple indexes containing a subset of pages. We also had to develop a search system that can perform the search over the indexes distributed across multiple machines

To tackle the first problem, we had to redesign our quantity scanner to reduce the dependency on NLP techniques and JAPE engine of GATE. The second problem was mostly tackled using standard tools. Although we had to modify those tools for performance reasons. Next we first give an overview of the implemented live QCQ system and then describe the improved quantity scanner that can annotate our corpus in acceptable speed and accuracy.

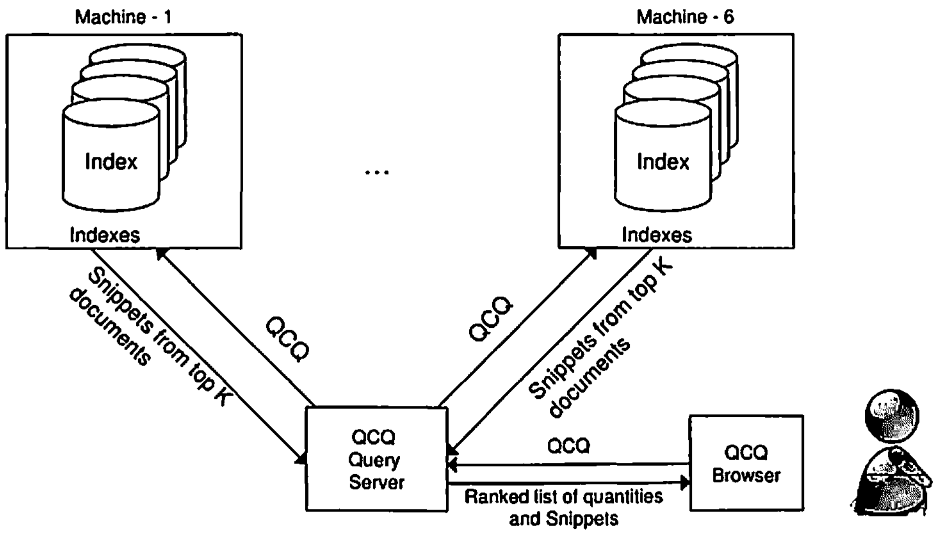
The corpus of 500 million pages is stored in highly compressed format and divided into several smaller files. The total size of the corpus is about 4.5 terabytes divided into 60 smaller files each containing around 8 to 10 million pages. Each smaller file is separately indexed using

Lucene [UrlLucene]. Each index is of size about 32GB. The 60 Lucene indexes are then hosted in 6 machines. These machines are called as index server. A user sends a QCQ to the QCQ server (Figure 3.12). The QCQ server then forwards the QCQ to the index servers. Each index server does a local search on the indexes it is hosting and retrieves top K pages containing the query terms in the proximity of a quantity with desired unit type. The top K is computed using Lucene's scoring function. Lucene's scoring functions require IDF of the query terms. IDF is a global property and cannot be computed locally by the index servers. Therefore, the QCQ server first accumulates the document frequency (DF) of each query term from all the index servers. It then broadcasts the IDF ($log(N/DF)$) of the query terms to the index servers.

Snippets (as defined in section 3.2.2) are then extracted from this top K pages. Only the snippet feature vector (z_i) and the quantity (x_i) are extracted as only those two things are used in our algorithms. The actual text of the snippets are only required while displaying it to a user and not used by the algorithms. Lucene index contains the term vector of the pages. Computing feature vector (z_i) only requires access to the term vector, query terms and query terms IDF. Therefore, can be computed locally by the index servers. The index server then sends back the snippet information (z_i, x_i, *docid*, start and end character offsets of the snippet) to the QCQ server. The QCQ server enumerates and ranks the quantity intervals using our algorithms and then shows it to the user. A user can click on a quantity interval to see the supporting snippets (Figure 3.14). Once a quantity interval is clicked then only the actual snippet texts need to be shown to the user. The snippet texts are retrieved from a froward index which can return the text segment given docid and start/end character offsets.

3.10 Quantity Search over Wikipedia Annotated Corpus

In this section, we show one more use of a web resource in answering quantity queries. We show the use of Wikipedia to disambiguate concepts and to support more specific query language. We used a recently developed system at IIT Bombay that annotates the entities mentioned in a corpus with entity IDs from Wikipedia [Kulkarni et al., 2009]. For example, the entity mentions like "Michael Jordon" (or even "Michael" referring to *Michael Jordon*) will get mapped to the Wikipedia entity "Michael Jordon". But there are seven different "Michael Jordon's" in Wikipedia referring to basketball player, footballer, machine learning researcher etc. So the challenge is to map the entity mentions (referred as *spot*) to the proper Wikipedia



Machine - 1
Index
Indexes
...
Machine - 6
Index
Indexes
Snippets from top K documents
QCQ
QCQ
Snippets from top K documents
QCQ Query Server
QCQ
Ranked list of quantities and Snippets
QCQ Browser

Figure 3.12: Distributed indexes in live QCQ system

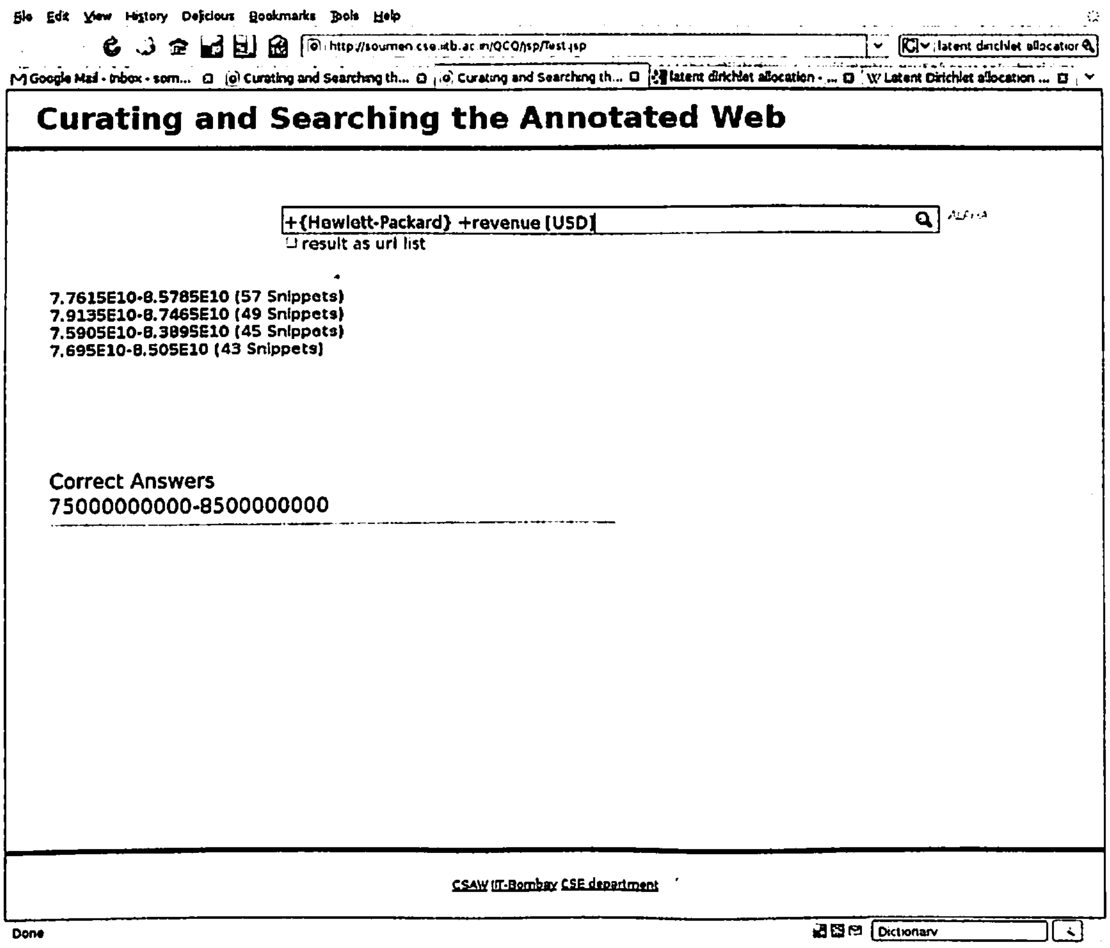
entity. The guiding premise of the algorithm was that a document largely refer to topically coherent entities. While *Michael Jordon* and *Stuart Russell* can refer to seven and three persons respectively in Wikipedia, a page where both *Michael Jordon* and *Stuart Russell* is mentioned is almost certainly talking about computer science and machine learning researcher *Michael Jordon*. They developed a collective annotation algorithm to accurately annotate the *spots* with Wikipedia entities.

One major advantage of annotating entity mentions with Wikipedia entity IDs is that it helps in disambiguation. Basketball player *Michael Jordan* and machine learning researcher *Michael Jordan* get linked to the respective Wikipedia entities and that helps in distinguishing them. Another scenario is "HP" and "Hewlett Packard" will both get linked to the Wikipedia entity about the company "Hewlett Packard". Note that because of the same disambiguation the instances of "HP" that are referring to "Hindustan Petroleum" will not get linked to "Hewlett Packard".

We have annotated our corpus of 500 million pages with Wikipedia entity IDs using the collective annotation algorithm. The inverted indexes of the corpus now contains the Wikipedia annotations as well. Therefore, one can use a Wikipedia entity ID as a query term. In the implemented system, a user can enter a Wikipedia entity with braces, e.g. {*Luxembourg (city)*}. A drop down menu is developed to assist a user in typing the Wikipedia entity.

In Figure 3.13 and 3.14 we show a sample query and output quantity intervals of the query.

In this example, a user is looking for the revenue of the company "Hewlett Packard". Wikipedia entity '*{Hewlett Packard}* is used as a query term. Since mentions of both "HP" and "Hewlett Packard" get linked to this Wikipedia entity, both of them will be considered as query terms.


File Edit View History Delicious Bookmarks Tools Help
http://soumen.cse.iitb.ac.in/QCQ/jsp/Test.jsp
Done

Curating and Searching the Annotated Web

+{Hewlett-Packard} +revenue [USD]

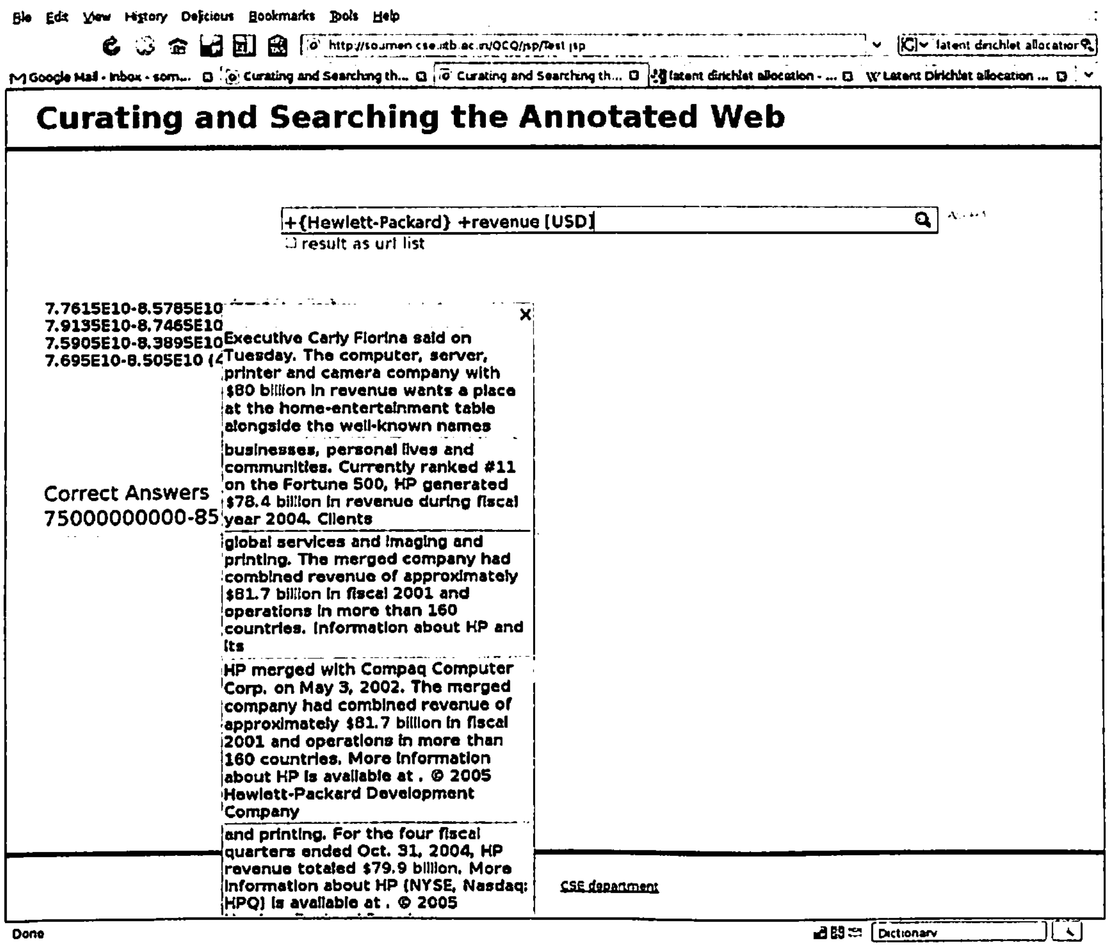
result as url list

7.7615E10-8.5785E10 (57 Snippets)
7.9135E10-8.7465E10 (49 Snippets)
7.5905E10-8.3895E10 (45 Snippets)
7.695E10-8.505E10 (43 Snippets)

Correct Answers
75000000000-8500000000

CSAW IIT-Bombay CSE department

Figure 3.13: QCQ search interface. Wikipedia entities in the query are specified using '{' (e.g. {Hewlett Packard}). A ranked list of quantity intervals are shown as the response. The number of snippets supporting the quantity interval is shown within bracket. "Correct answers" are the answers that could possibly be extracted by manually browsing the corpus (data of 2006)



Curating and Searching the Annotated Web

+{Hewlett-Packard} +revenue [USD]

result as url list

7.7615E10-8.5785E10
7.9135E10-8.7465E10
7.5905E10-8.3895E10
7.695E10-8.505E10 (4

Correct Answers
7500000000-85

Executive Carly Fiorina said on Tuesday. The computer, server, printer and camera company with $80 billion in revenue wants a place at the home-entertainment table alongside the well-known names

businesses, personal lives and communities. Currently ranked #11 on the Fortune 500, HP generated $78.4 billion in revenue during fiscal year 2004. Clients

global services and imaging and printing. The merged company had combined revenue of approximately $81.7 billion in fiscal 2001 and operations in more than 160 countries. Information about HP and its

HP merged with Compaq Computer Corp. on May 3, 2002. The merged company had combined revenue of approximately $81.7 billion in fiscal 2001 and operations in more than 160 countries. More information about HP is available at . © 2005 Hewlett-Packard Development Company

and printing. For the four fiscal quarters ended Oct. 31, 2004, HP revenue totaled $79.9 billion. More information about HP (NYSE, Nasdaq: HPQ) is available at . © 2005

CSE department

Figure 3.14: The expanded list of supporting snippets for a quantity interval is shown here

3.11 Conclusion

We introduced QCQs, and proposed algorithms for returning consensus intervals in response to QCQs. We showed that corroborative ranking of intervals is more accurate than ranking snippets independently. The experiments show that the proposed ranking algorithms achieve about 20% higher accuracy than most state of the art approaches. We have made our QCQ system more powerful by replacing search APIs with our own quantity index on Web-scale corpora. We have also shown that how Wikipedia annotations can be used to enrich the QCQ system.

Chapter 4

Generating Topic Features from a Web Corpus

In the last chapter, we discussed techniques for mining the web to answer quantity queries. In this chapter, we will discuss the use of web resources to improve the accuracy of text classification and clustering tasks. In particular, we will discuss the second sub-problem addressed in the thesis, namely obtaining topic features from a web corpus. Broadly we will describe the methods to obtain topic features from a web corpus and will experimentally show that the topic features helps in improving the accuracy in various tasks of text classification and clustering.

The organization of the chapter is as follows; we first study a simple method of feature generation from Wikipedia (Section 4.1). The feature generation method largely follow the Gabrilovich's method described in the Section 2.2.2. We then show that features generated from Wikipedia can help in clustering short texts and doing inductive transfer for text classification. Next we point out some shortcomings of the Gabrilovich's method of feature generation and propose a novel method of feature generation from a web corpus (Section 4.2). The proposed method addresses some of the shortcomings of the Gabrilovich's method and also improves the accuracy of text classification tasks.

4.1 Generating Topic Features from Wikipedia

In this section, we deploy a very simple method to generate topic features from Wikipedia. The method is very similar but slightly simpler to the method of feature generation proposed by Gabrilovich and Markovitch [2006]. The deployed feature generation method works as follows; an inverted index of the Wikipedia pages is first created from the XML dump of Wikipedia. Given a text document, a set of Wikipedia pages are retrieved from the inverted index using a query (or set of queries) constructed from the text of the document. The titles of the top K retrieved Wikipedia pages are used as the K topic features of the given document. These K topic features are also called as Wikipedia features. The documents are then represented in a features space consisting of "bag of words" as well as the Wikipedia features.

Wikipedia XML dump is freely available for download [1]. Only the dump of the English language Wikipedia pages are used in the experiments here. The dump used in the experiment was obtained on November 26, 2006. A preprocessing step was used to filter out not so informative Wikipedia pages and thereby reducing the size of the index. The pages removed were Template pages (starts with "Template:") or other pages describing the Wikipedia features (starts with "Wikipedia:"). A stop word filter was then applied on the Wikipedia pages. Short articles in Wikipedia are either under construction or about overly specific concept. Therefore, any page that contains less than 50 words at this stage were also removed. At the end there were 1,174,107 pages. An inverted index of these pages were created using Lucene [UrlLucene]. Lucene is an open source java based indexing and search technology. To obtain Wikipedia features of a document, a set of top matching Wikipedia article was retrieved from this Lucene index using the queries constructed from the text of the document.

The following example shows how Wikipedia features are obtained and used to enrich the document representation. Consider the following two news headlines as two text documents.

docl: *Sony to Slash PlayStation3 Price*

doc2: *Jittery Sony Knocks $100 Off PS3 Price Tag*

Table 4.1 shows the bag of words representation of these two documents. Each row in the table is the feature vector representation of the corresponding document. For clarity, binary weighting is used instead of TF*IDF weighting. In binary weighting, if a word appears in the document the corresponding feature gets weight 1 otherwise it gets weight 0. Note that the words are converted to lowercase in the feature. Lowercase conversion, stop word removal (and sometime setmming) are standard preprocessing steps of document representation. Table 4.1 shows that there are very few term overlaps between the two documents although the documents are talking about the same news topic. The readers of this thesis can easily figure out that the

[1]available at `http://download.wikimedia.org/`

follows; an inverted index of the Wikipedia pages is first created from the XML dump of Wikipedia. Given a text document, a set of Wikipedia pages are retrieved from the inverted index using a query (or set of queries) constructed from the text of the document. The titles of the top K retrieved Wikipedia pages are used as the K topic features of the given document. These K topic features are also called as Wikipedia features. The documents are then represented in a features space consisting of "bag of words" as well as the Wikipedia features.

Wikipedia XML dump is freely available for download [1]. Only the dump of the English language Wikipedia pages are used in the experiments here. The dump used in the experiment was obtained on November 26, 2006. A preprocessing step was used to filter out not so informative Wikipedia pages and thereby reducing the size of the index. The pages removed were Template pages (starts with "Template:") or other pages describing the Wikipedia features (starts with "Wikipedia:"). A stop word filter was then applied on the Wikipedia pages. Short articles in Wikipedia are either under construction or about overly specific concept. Therefore, any page that contains less than 50 words at this stage were also removed. At the end there were 1,174,107 pages. An inverted index of these pages were created using Lucene [UrlLucene]. Lucene is an open source java based indexing and search technology. To obtain Wikipedia features of a document, a set of top matching Wikipedia article was retrieved from this Lucene index using the queries constructed from the text of the document.

The following example shows how Wikipedia features are obtained and used to enrich the document representation. Consider the following two news headlines as two text documents.

doc1: *Sony to Slash PlayStation3 Price*

doc2: *Jittery Sony Knocks $100 Off PS3 Price Tag*

Table 4.1 shows the bag of words representation of these two documents. Each row in the table is the feature vector representation of the corresponding document. For clarity, binary weighting is used instead of TF*IDF weighting. In binary weighting, if a word appears in the document the corresponding feature gets weight 1 otherwise it gets weight 0. Note that the words are converted to lowercase in the feature. Lowercase conversion, stop word removal (and sometime setmming) are standard preprocessing steps of document representation. Table 4.1 shows that there are very few term overlaps between the two documents although the documents are talking about the same news topic. The readers of this thesis can easily figure out that the

[1]available at `http://download.wikimedia.org/`

above two documents are about the same topic. This is because the readers have the background knowledge that "PlayStation3" and "PS3" refers to the same object and also a human can easily understand that "knocking off price" and "slashing price" have same meaning.

	sony	slash	playstation3	price	jittery	knocks	ps3	tag
doc1	1	1	1	1	0	0	0	0
doc2	1	0	0	1	1	1	1	1

Table 4.1: BOW representation (binary weighting)

Figure 4.1 shows the titles of the retrieved Wikipedia articles for these two documents. Note that there is quite a bit of overlaps in the retrieved title lists although the word overlaps of those documents was few. This is because in Wikipedia articles the synonyms (e.g. PlayStation3 and PS3) are used interchangeably therefore they are able to retrieve similar lists of articles. The titles of the top K Wikipedia articles are then added as topic features of the document. The new feature space consists of bag of words as well as the topic features. This features space is referred here as the enriched feature space. Table 4.2 shows the representation of these two documents in the enriched feature space.

	sony	slash	...	Playstation Network Platform	Playstation 2	Ken Kutaragi	Playstation 3	...
doc1	1	1	...	1	1	1	1	...
doc2	1	0	...	1	0	1	1	...

Table 4.2: Enriched representation with Wikipedia features

4.1.1 Clustering Short Texts

Now we show how the Wikipedia features can help in improving the accuracy of clustering short text documents. Text clustering is an important tool for content management and dealing with the problem of information overload. In many applications, the items to be clustered are short segments of texts, e.g., search result snippets, forum & chat messages, blog & news feeds, product reviews, books & movie summary etc. The short texts consists of few words to few sentences. Because of the short length the individual text items are unable to provide enough contexts or word co-occurrence statistics for a good similarity measure between the items. The general problem of bag of representation (e.g. synonyms, polysemy) are also more prominent

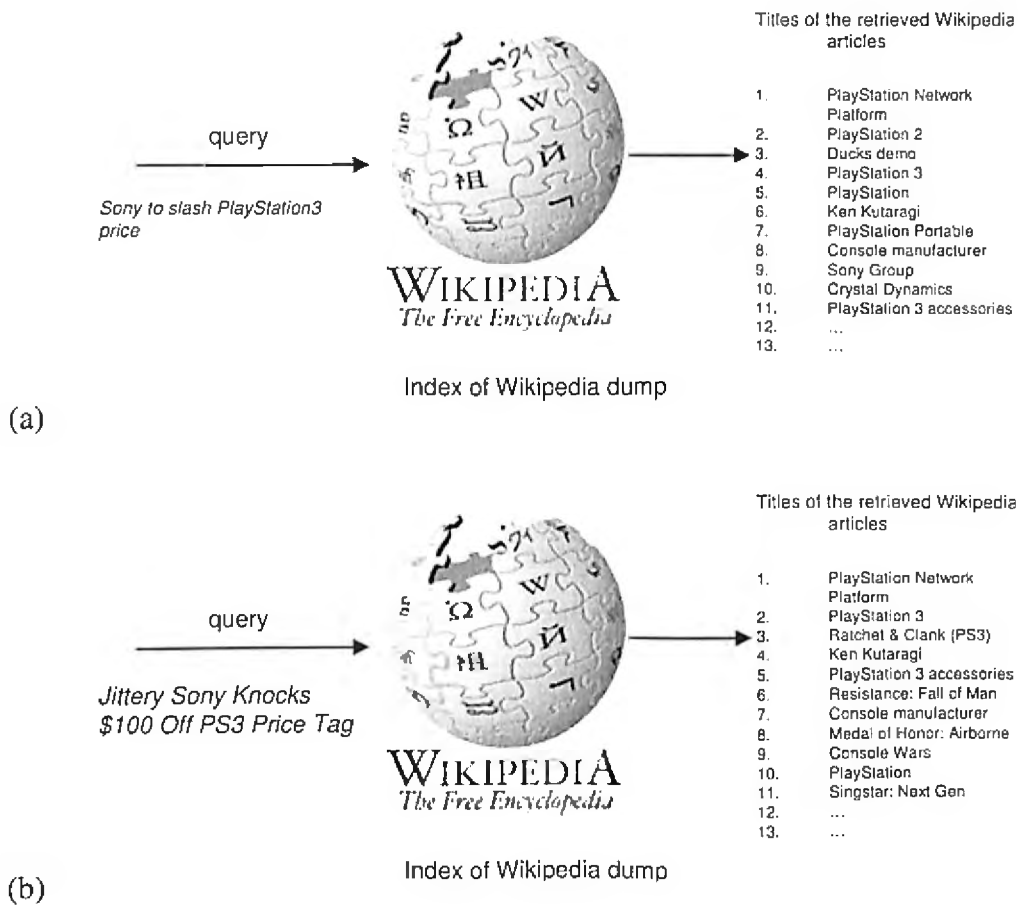

Titles of the retrieved Wikipedia articles
query
Sony to slash PlayStation3 price
WIKIPEDIA The Free Encyclopedia
Index of Wikipedia dump
1. PlayStation Network Platform
2. PlayStation 2
3. Ducks demo
4. PlayStation 3
5. PlayStation
6. Ken Kutaragi
7. PlayStation Portable
8. Console manufacturer
9. Sony Group
10. Crystal Dynamics
11. PlayStation 3 accessories
12. ...
13. ...
(a)
Titles of the retrieved Wikipedia articles
query
Jittery Sony Knocks $100 Off PS3 Price Tag
WIKIPEDIA The Free Encyclopedia
Index of Wikipedia dump
1. PlayStation Network Platform
2. PlayStation 3
3. Ratchet & Clank (PS3)
4. Ken Kutaragi
5. PlayStation 3 accessories
6. Resistance: Fall of Man
7. Console manufacturer
8. Medal of Honor: Airborne
9. Console Wars
10. PlayStation
11. Singstar: Next Gen
12. ...
13. ...
(b)

Figure 4.1: Feature generation from Wikipedia

when the text items are of short lengths. Therefore, clustering (or even classification) short texts is a challenging problem.

The examples in the Figure 4.1 show that the Wikipedia features can provide context information that are not even there in the documents. For example, the Wikipedia features suggest that both the documents are related "PlayStation Network Platform", "PlayStation 3", "PlayStation" etc although such terms may not appear in the document. The hypothesis here is clustering short texts with Wikipedia features will improve the clustering accuracy. To test this hypothesis we conduct an comparative study of several clustering algorithms between BOW representation and the enriched representation of the documents. The results indicate that for many clustering algorithms, significantly higher accuracy is obtained when the documents are represented in the enriched feature space. Next we discuss the datasets and the results of the experiment.

Dataset

The labeled dataset for the experiments was were created from Google News [UrlGoogleNews]. The goal was to collect a set of news articles belonging to a set of topics with known topic assignment. A topic is considered as a cluster[2] and the news belonging to a topic are considered as the members of the corresponding cluster. We run the different clustering algorithms on these news articles and compute clustering accuracy against the ground truth cluster memberships.

Google News homepage contains short articles (news clips) on different news topics. For each article there is a link pointing to the other articles on the same topic. Figure 4.2 shows one such article and the associated link (circled red) to the other articles of the same news topic. Following the red circled link in the Figure 4.2 will lead to a page that contains 30 news clips on the same topic. If there are more than 30 news clips available on that topic then there will be a link to the "Next" page. Therefore following the link to the related article and the subsequent "Next" pages one can gather news articles of a particular topic. Repeating this procedure starting from another news clip of some other topic we can gather a set of news clip belonging to a set of topics. Since we know the topic memberships of the news clips, we can use this data to evaluate a clustering algorithm.

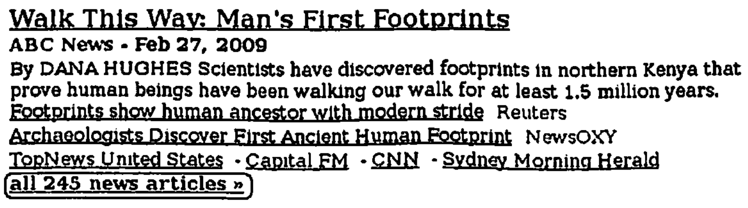

Walk This Way: Man's First Footprints
ABC News - Feb 27, 2009
By DANA HUGHES Scientists have discovered footprints in northern Kenya that prove human beings have been walking our walk for at least 1.5 million years.
Footprints show human ancestor with modern stride Reuters
Archaeologists Discover First Ancient Human Footprint NewsOXY
TopNews United States - Capital FM - CNN - Sydney Morning Herald
all 245 news articles »

Figure 4.2: A news clip from Google News

We took a snapshot of the Google News homepage on February 16, 2007. That page had 26 news articles on 26 different topics and so 26 links pointing to the other articles of the corresponding topics. We crawled those 26 links and the "Next" page from each link and then extracted the news clips by parsing the crawled pages. This way we gathered 1557 news clips belonging to the 26 different topics (i.e., approximately 60 articles per topic). Each news clip here consists of a title and one line of description. Note that the original 26 links of the Google News homepage served as the cluster identifiers.

[2]There could be other notion of clusters in a set of news articles. But topic based clustering is common in the literature (e.g.. [Allan, 2002]) and has practical importance

News articles of the above dataset were then represented in two different ways and six different clustering algorithms were run using each of the representation. The first representation is the bag of words representation (referred as *baseline*) and the second one was the in the enriched representation with Wikipedia features (referred as *wiki_method*). As mentioned earlier, the Wikipedia features of a news clip was obtained by querying the Lucene index of the Wikipedia dump. Two separate query was used to get the Wikipedia features of a news clip. One query is the title and the other one is the description sentence of the news clip. For each query, 10 top matching Wikipedia articles were retrieved from the Lucene index; i.e. total 20 Wikipedia articles were retrieved. The titles of these 20 Wikipedia articles were then used as the features. Note that in this list of 20 titles some titles may appear twice since two queries were used separately. The weight of the corresponding Wikipedia feature was set appropriately.

The title of a news article is generally more important than the description. We obtained better results by giving more importance to the terms that appear in the title. In the *baseline* representation, this was achieved by giving double weights to the terms appearing in the "title" of the news article. Similarly in the *wiki_method*, we double the weights of the Wikipedia features that were retrieved by the title of the news clip.

In the experiments, the freely available clustering package SenseClusters [UrlSenseclusters] was used. SenseClusters uses Cluto [UrlCluto] for clustering but provides suitable interfaces for text processing, parameter setting and evaluation. As mentioned in the section 2.1.2 Cluto provides six different clustering algorithms, three partitioning methods (rb, rbr and direct), two agglomerative methods (agglo, bagglo) and one graph based method. The details of these clustering algorithms can be found in the cluto manual. The goal here is to show that the representation in the enriched space can help different types of clustering algorithms. There are some parameters to choose in clustering, e.g. the stopping criterion function. All the parameters were left at the default settings of the SenseClusters.

Results

Table 4.3 shows the accuracy achieved by the *baseline* and the *wiki_method* with the different clustering algorithms. The *wiki_method* achieved the best overall accuracy of 89.56%. It also achieved better accuracy than the *baseline* in 5 out of 6 clustering algorithms. When averaged across the six different clustering algorithms the *wiki_method* achieves accuracy of 74.81%, an improvement of 33.26% from the baseline overall average accuracy of 56.14%.

Only in the case of "rbr" *wiki_method* is little worse than the *baseline*. We could not find exact explanation for this degradation of performance as the Cluto manual does not provide the exact details of the clustering algorithms. One possible reason we believe is over fitting. rbr is the extension of the rb and tries to globally optimize the solution by repeatedly calling the rb algorithm. rb already optimizes the number of clusters and a cluster criterion function. Therefore, the multiple objectives and local optimization of those objectives in a higher dimensional feature space in rbr can potentially over fit the data.

	rb	rbr	direct	agglo	graph	bagglo
baseline	63.20	79.38	67.05	22.03	81.62	23.57
wiki_method	85.42	63.65	82.66	83.88	**89.56**	43.67

Table 4.3: Clustering accuracy

4.1.2 Improving Inductive Transfer for Text Classification

In machine learning literature, inductive transfer "refers to the problem of retaining and applying the knowledge learned in one or more tasks to efficiently develop an effective hypothesis for a new task" [Silver et al., 2005]. A great deal of research on inductive transfer has been done under various names, e.g., learning to learn, life-long learning, transfer learning, multi-task learning, hierarchical Bayes etc. Labeled training data is usually scarce or expensive. But labeled training data might be easily available in some related tasks. Also, in some scenario there might already exist a learned model for a related task. The purpose of inductive transfer is to use the knowledge learned on the related tasks to improve the performance on the target task. For example, Wu and Dietterich [2004] showed that the image classification accuracy can be improved when SVMs are trained on a large number of related images but relatively few target images. In their experiment, the target image was scanned tree leaves whereas the related images where a collection of dried plant specimens easily available from a university Herbarium. It has been observed that transferring knowledge often helps if the target and the related tasks are similar enough. But it can also hinder performance if the tasks are too dissimilar. This later phenomena is known as "negative transfer" [Silver et al., 2005].

In this section, we show a method of improving the performance of inductive transfer in the task of text classification using Wikipedia features. In particular, we target an inductive transfer scenario shown to be useful in daily classification task [Forman, 2006]. The inductive transfer model (described shortly) is basically a classifier re-use model [Bollacker and Ghosh, 1998]. A set of classifiers are built for a related set of tasks. Knowledge transfer from these classifiers is done by using the outputs of these classifiers as inputs in the target task. In our method, the classifiers from which knowledge need to be transferred are trained in the feature space consists of the Wikipedia features. We observed that classifiers trained in such feature space are more effective in transferring knowledge than the classifiers trained on the bag of words features.

Next we describe the daily classification task and the inductive transfer model to reuse the past classifiers in the daily classification task. We then show that the inductive transfer becomes more effective when the past classifiers are trained in the Wikipedia feature space.

Daily Classification Task (DCT)

The daily classification task (DCT) is the task of building a classifier daily to classify news articles. Consider the text categorization problem for a news agency. Everyday the news agency receives lots of news articles and they want to categorize the articles under certain categories or classes (e.g. "sports", "business", "Sci/Tech", etc.). In addition, the news agency has some human annotators who provides ground truth class labels for a subset of news articles received in any particular day. The news articles with ground truth class labels can be used as the training data to build a classifier. The task is to classify the remaining articles (for which the ground truth class labels are not available). The performance is measured by taking the average of the classification accuracy over all days.

Formally, everyday the system receives N news articles. Out of these N articles, ground truth labels of T articles are available for training. The job is to classify the remaining (N-T) articles of that day. Performance will be reported by taking the average performance over 365 days (say).

In the news domain, as time progresses new events occur and old events disappear. Therefore, the underlying concepts are not stable but change over time. This problem is known as concept drift. In such setting, classifiers built in the past are not very effective in predicting the class labels of todays (or futures) news articles. Therefore as time progresses we need to build new classifiers using the labeled training data made available recently.

The strawman approach of solving the DCT problem is to build a classifier everyday using only the T training cases available that day. Then use this classifier to predict the class labels of the remaining (N-T) cases of that day. But we should leverage the training examples made available in the past. A straightforward approach to do that is to use all the training examples available up to the current date. For example, when building a classifier on the i^{th} day $i * T$ training examples are available. We can use all the training examples together to build a classifier. As observed by Forman [Forman, 2006], this approach also does not work well. Due to the change in concepts over time, putting all training examples together actually hurts the performance. In addition, using all training examples has performance penalty as training time will increase and feature space will blow up due whole bunch of new terms in the training set.

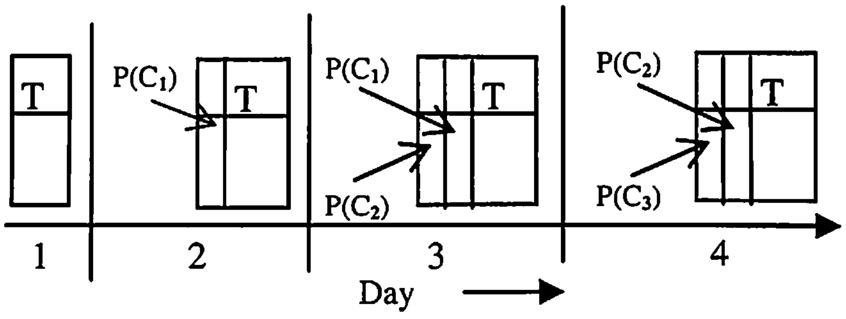
More sophisticated and effective approach in solving the DCT problem is to transfer the knowledge of the previously built classifiers to the new classifier being built. This can be done by using an inductive transfer model.

Inductive Transfer Model for DCT

An inductive transfer for DCT can be designed by using the outputs of the past classifiers as additional input features to a new classifier. Like the strawman algorithm, everyday a new classifier is trained using the T available training cases. Inductive transfer is done by adding P additional binary features (positive/negative) to the cases of today. These P features are the predictions of the P previous days' classifiers on the today's cases. The prediction of a classifier on an article (case) is whether the article belongs to the target class or not (positive/negative). These features are referred here as the prediction features. Here the value of P determines the number of previous days' classifiers we can use. By varying the value of P in our experiment we will see the impact of P. Note that these P prediction features are used in addition to the other features (e.g. bag of words and Wikipedia features) of the articles. Also, these prediction features are added to the todays training as well as test cases.

An example of our inductive transfer model for $P = 2$ is shown in the Figure 4.3. At day 1 there is no past classifier. Therefore, the classifier of day 1 is built using the T training cases without any prediction features. At day 2 there is the past classifier of day 1. For each cases of day 2, the prediction of the classifier of day 1 is added as an additional binary feature. Now the classifier of day 2 is built using the T training cases where each training case has an extra binary feature. At day 3, we have 2 previous days' classifiers available. Therefore, two additional prediction features are added to the each cases of day 3. At day 4 also there are only

2 previous days of classifiers are available as we choose to use P=2 in this example.



- T : Number of Training cases on any day. The boxes are feature vector representation of the today's cases (training and test)
- C_i: Classifier built on day i using the T training cases of that day
- $P(C_i)$: Prediction of the classifier C_i on each of the cases.

Figure 4.3: Inductive Transfer Model

One problem in doing inductive transfer in this fashion is that all the past classifiers remain always in use for any value of $P > 0$. For $P = 1$ today's classifier depends on the yesterday's classifier only. But the yesterday's classifier depends on the one from the day before it, and so on. To break this recurrence while doing inductive transfer from the past classifiers we have to make sure that the past classifiers are independent. This is done by building the past classifiers separately simply using the T training data without adding prediction features to them. That is, each day two different classifiers are built. One is dependent on the P previous days classifiers and this classifier is used in predicting the class label of remaining (N-T) cases of that day. This classifier is referred here as *today's* classifier. Average performance of the *today's* classifiers over a period of time (365 days) is used as the performance measure of the system. The other classifier is trained without any prediction features and therefore independent of any previous classifier. This latter classifier is used in future while doing inductive transfer. Here the term *previous days'* classifiers always refer to this kind of classifiers.

Doing inductive transfer in the above mentioned way has many advantages. Firstly, the past classifiers are used just to provide features to the cases of today. A state of the art classifier should be able ignore those features that are useless. Classifiers, like support vector machine

(SVM) are known to be able to ignore large set of redundant words in the text classification task. Therefore, doing inductive transfer through features with a state of the art classifier reduces the risk of negative transfer. Secondly, the performance overhead of this inductive transfer model is just adding and using P additional features and therefore expected to be negligible.

The inductive transfer model used here is very similar to the temporal inductive transfer (TIX) model described by Forman [Forman, 2006]. But to make the *previous days'* classifiers useful he had to use hindsight. Hindsight is the ground truth labels of a percentage of the past cases that were not in the training set. In his work, while training the *previous days'* classifiers, in addition to the T training cases a percentage of (N-T) cases were included in the training set with proper ground truth labels. The best performance was observed using full hindsight. Full hindsight means *previous days'* classifiers are trained using all the N cases of the corresponding day. In such scenario all the N cases have to be properly labeled. The problem here is that there is no easy way to obtain the hindsight.

In our setting, no hindsight has been used. *Previous days'* classifiers are trained using only the T training cases of the corresponding day.

Experiments

The hypothesis for the experiments here is that the inductive transfer from the classifiers trained using the Wikipedia features are more effective. The Wikipedia features of the news articles of a particular class are more stable, whereas the terms appearing in such news articles can change overtime. Therefore, when the *previous days'* classifiers are trained using the Wikipedia features they are able to predict the class label of *today's* articles better. As a consequence the P prediction features becomes more informative and improves the accuracy of the *today's* classifier.

Dataset We used the news articles of RCV1 corpus [Lewis et al., 2004] to setup the daily classification task. The RCV1 corpus contains more than 800,000 news articles produced over 365 days (from 1996-08-20 to 1997-08-19). The news articles are manually categorized to many topics.

We sorted the news articles by day and everyday only 400 articles were used just to keep the experiment time manageable. Out of those 400 articles, 100 articles were used as training cases and the job was to classify the remaining 300 articles everyday (i.e. as per our terminology

$N = 400$ and $T = 100$). The macro average F-measure over 365 days is used as the measure of performance

Methods Three methods are evaluated in our experiment. All three methods differ only in representing the articles in terms of features. The first method represents the articles only using the bag of words, the next two use the Wikipedia features also. Otherwise, all three methods deploy the same inductive transfer model described above. Each day the *today's* classifier is trained using the 100 training articles and this classifier is used to predict the class labels of the remaining 300 articles of that day. Inductive transfer is done by adding the predictions of P *previous days'* classifiers as additional features of the articles. The *previous days'* classifiers are trained without using these prediction features (to break the recurrence as discussed earlier).

Linear Support Vector Machine (SVM) of Weka library [Witten and Frank, 2005] (version 3.5) was used as the base classifier. Only binary feature weighting has been used with the parameter C of SVM was set to default value 1. Next we describe the different methods in details.

Baseline Each news article is represented only using bag of words. The stop words are removed.

BOW+Wiki The bag of words features of each article is augmented with the Wikipedia features of the article. Therefore, in this representation the features of an article are either the terms appearing in the article or the Wikipedia titles retrieved using the article as a query. The *today's* classifiers as well as the *previous days'* classifiers use this as the base representation of the articles. Note that in addition to these features, P prediction features are added to the articles while training and testing the *today's* classifier.

WikiOnly Our conjecture is that the Wikipedia features are more stable in terms of concept drift as it captures the background knowledge of contents of the articles. To make full use of it we choose to train the *previous days'* classifiers only using Wikipedia features of the articles. That is, everyday we train the classifier that will be used in future for inductive transfer with only the Wikipedia features of the training articles. The other classifier, i.e., the *today's* classifier, is trained using bag of words (and P prediction features) similar to the Baseline method. Also note that while generating the prediction features of today's articles the *previous days'* classifiers use only the Wikipedia features of the todays articles.

Results

The above described methods were tested on the DCT of binary classification for several individual categories in the RCV1 corpus. Figure 4.4 shows the average F-measure (over 365 days) achieved by different methods on four major classes in the RCV1 corpus; ECAT (ECONOMICS), M13 (MONEY MARKETS), GCAT (GOVERNMENT/SOCIAL), GSPO (SPORTS). The results are shown here by varying the value of *P* from 0 to 128.

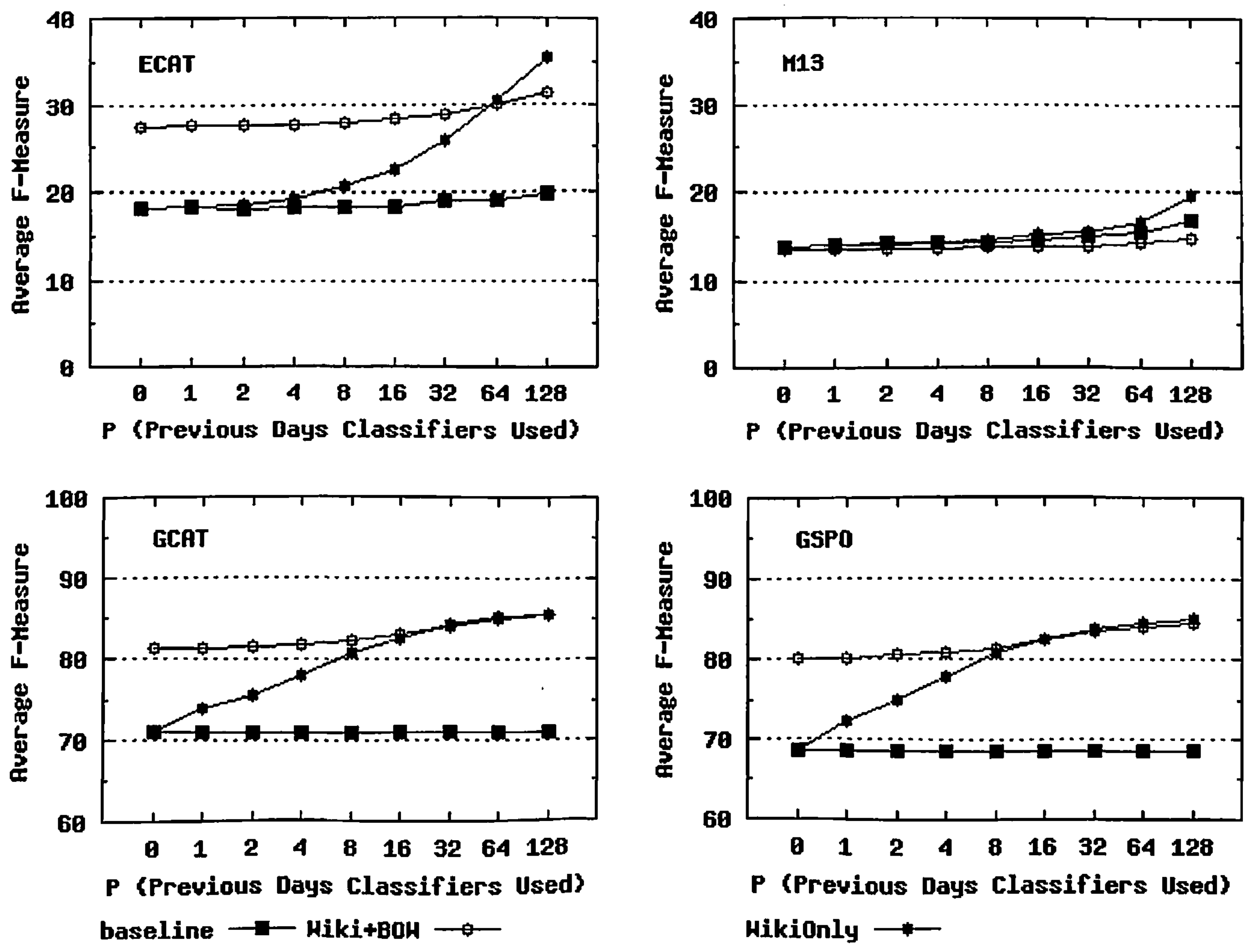


Figure 4.4: Results for 4 major classes in the RCV1 corpus: ECAT and M13 (top), GCAT and GSPO (bottom)

Figure 4.4 shows that the performance of the *baseline* method remains flat with the increase of P. Only in the case of M13 we can see some marginal improvement as P increases.

$P = 0$ means no prediction features from the *previous days'* classifier and therefore no inductive transfer. So the F-measure at $P = 0$ gives the performance of the strawman algorithm. The flat curves of the *baseline* method thus indicate that the inductive transfer from *previous days'* classifiers has almost no effect when the *previous days'* classifiers are trained using only bag of words features

Next we observe that in the most cases the *BOW+Wiki* method yields much higher average F-measure than the baseline for all different values of P. Here also $P = 0$ means no inductive transfer but the articles are represented in the enriched feature space consisting of bag of words as well as the Wikipedia features. That led to a significant increase in F-measure in most cases even for $P = 0$ (almost 10 points for ECAT, GCAT, and GSPO). But in this case also the curves remain almost flat with the increase of P. This implies that in this method also there is not much value in using the predictions of the *previous days' classifiers.*

In the *WikiOnly* method, the *today's* classifiers use only the bag of words representation of the articles. Therefore, the performance at $P = 0$ is the same as the baseline method. But as P increases the average F-measure starts improving. At $P = 128$ this method either surpass the performance of *BOW+Wiki* method (for ECAT and M13) or at least yields same average F-measure. This huge increase in performance is just because of the inductive transfer from the *previous days'* classifiers. Here the *previous days'* classifiers are trained using only the Wikipedia features of the articles.

In news domain the words change very frequently [Forman, 2006]. Therefore, a classifier trained using the word features fails to provide good prediction in the news articles of future dates. That is the main reason for the flat curves of the *baseline* and *BOW+Wiki* approaches. The Wikipedia features of an article captures the broader contexts of the concepts described in the articles. Unlike the word features, the Wikipedia features remain stable for different articles describing same (or nearly same) concepts. Therefore, the classifiers trained only on the Wikipedia features provide more useful predictions on the articles of future dates.

Since the *WikiOnly* method was performing best in our experiment, in Figure 4.5 we show the performance impact of this method on 30 major categories in the RCV1 corpus. In this figure, the length of an arrow indicates the impact on performance for $P = 0$ to $P = 128$. It can be seen that for the majority of the classes there are significant improvements in the average F-measure. Also, there is no negative impact on the performance (no downward arrow) so no "negative transfer".

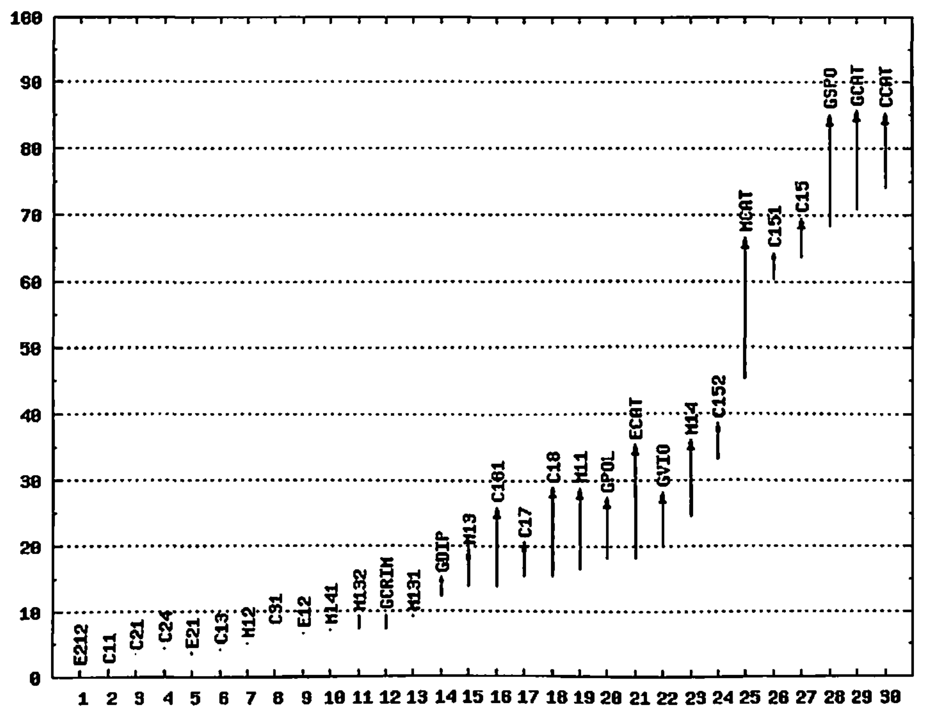


Figure 4.5: Results for 30 major classes in the RCV1 corpus

4.2 Feature Generation using Topic Modeling

So far in this chapter we adopted a simpler version of feature generation method proposed by Gabrilovich et al. We showed that such Wikipedia features are useful in text clustering and inductive transfer over text classifiers. Some earlier work show that such features are useful in other tasks like text classification [Gabrilovich and Markovitch, 2006], computing semantic relatedness between words [Gabrilovich and Markovitch, 2007] and web search [Milne et al., 2007]. Although the results of all these work show that the Wikipedia features are useful in different information retrieval related tasks but the above method of feature generation from Wikipedia has the following limitations.

Ad-hoc retrieval method The above method of feature generation from Wikipedia depends on ad-hoc retrieval technique. A set of queries are constructed from the given document and a set of Wikipedia features are retrieved from the inverted index of the Wikipedia corpus using a ranking function. There is no guidelines about how to construct the queries, what ranking function to use or how many Wikipedia features to add.

Large number of features The total number of Wikipedia features added is the number of distinct Wikipedia pages retrieved by the documents of the classifications/clustering task.

Therefore, the feature space can become very large. For example, consider the scenario where each document retrieves a different set of Wikipedia pages. Wikipedia has millions of pages and therefore the feature space can potentially grow to have millions of dimensions. There is no proper way to control the growth of the feature space. High dimensional feature space can have performance impact on the downstream classifications/clustering algorithms.

Dependency on the Wikipedia corpus This method retrieves the Wikipedia articles using textual similarity to the given document and uses the "titles" assigned to the Wikipedia articles as the additional features. Therefore, it heavily depends on the similarity of the document to the Wikipedia articles and the titles given to the Wikipedia articles. But Wikipedia is a fast evolving corpus. It is not known how this method will behave as the Wikipedia corpus evolves.

In this section, we develop a novel method of feature generation from an additional corpus. The additional corpus can be the Wikipedia corpus or any other corpus available in the web. In this method K topics are extracted from the additional corpus using a topic modeling technique. These K topics then act as K additional features. Instead of ad-hoc retrieval, this method uses more principled inference methods of topic modeling to estimate the topic probabilities in a given document. The parameter K controls the total number of new features to be added. Therefore, the growth of feature space is limited by K which is typically in the range of $100 - 200$. Also, unlike the Wikipedia features, the features here are extracted "topics" and not the "titles" of the articles in the corpus. Therefore, this method can work with any other web corpus not just with Wikipedia like encyclopedia corpus.

4.2.1 Overview of the method

First we recall and show some examples of the topic modeling technique Latent Dirichlet Allocation (LDA) [Blei et al., 2003] introduced in the Section 2.1.1. LDA can extract K topics, $z_1 \dots z_K$, from a corpus. Here a topic is the probability distribution over the words, i.e. topic z_i is the probability distribution $P(w|z_i)$ where w is a word in the vocabulary. Also, for each document d, LDA determines the probability $P(z_i|d)$ that the document contains the topic z_i. LDA has a principled mechanism for inference, i.e. given a new document d_{new}, which was not in the corpus, it can determine the topic probabilities $(P(z_i|d_{new}))$ for the new document.

Some example topics extracted from two different sample collections of Wikipedia pages are shown in the Figure 4.6. Here each sample collection contains 10,000 Wikipedia pages and 200 topics were extracted from each collection using LDA. Figure 4.6 shows 10 extracted topics for each sample collection. In the tables, each column represent a topic (a probability distribution over the words). For each topic z_i, the words are sorted based on the probability $P(w|z_i)$ and only the top 5 words are shown in the Figure 4.6. Here the words are stemmed to keep the vocabulary size manageable.

z_a	z_b	z_c	z_d	z_e	z_f	z_g	z_h	z_i	z_j
band	radio	parti	style	militari	school	airport	engin	game	jewish
music	station	elect	background	unit	univers	airlin	design	player	jew
record	broadcast	vote	color	forc	student	intern	power	video	rabbi
album	televis	parliament	font	armi	colleg	flight	model	dragon	hebrew
rock	channel	minist	border	war	educ	air	electr	plai	ben

(a) random sample - I

z_k	z_l	z_m	z_n	z_o	z_p	z_q	z_r	z_s	z_t
album	radio	parti	style	war	school	air	design	game	israel
music	station	elect	text	armi	student	aircraft	engin	player	jewish
band	broadcast	vote	background	forc	high	airport	power	card	jew
record	televis	box	font	battl	educ	flight	imag	video	isra
song	channel	candid	width	militari	grade	airlin	model	plai	rabbi

(b) random sample - II

Figure 4.6: Topics extracted from Wikipedia

Few interesting things can be noticed in the Figure 4.6. Firstly, lots of extracted topics can be interpreted as high level real world topics (e.g. *music*, *politics*, *education* etc). Secondly, the topics shown in the table a and table b in Figure 4.6 are extracted from two different random samples of Wikipedia pages. But still the topics look very similar. Each corresponding column of table a and b in the Figure 4.6 has at least 3 words in common. It shows that it is possible to extract general real world topics, like *music*, *politics*, *education* etc, from a sample collection of Wikipedia pages. Note that corresponding columns in table a and b of figure 4.6 have different topic numbers (e.b. z_a vs z_k for the first columns). This is to emphasize the fact that even if two extracted topics from two different Wikipedia corpora looks same there is no guarantee that those topics will be found under the same topic id.

The feature generation method proposed here applies LDA over the additional corpus to extract 200 topics. Now given the documents of a classification or clustering task, it uses the LDA provided inference technique to estimate the probability distribution of the 200 topics ($P(z_i|d)$) for each document d. This probability distribution or the 200 probability values are used as the 200 topic features of the documents. As before the topic features are used in addition to the bag of words features of the documents.

We used LDA with Gibbs sampling [Griffiths and Steyvers, 2004] methods for learning (extracting topics $P(w|z_i)$) and inference (estimating topic probability $P(z_i|d)$). Topic extraction is done over the additional corpus and therefore can be done off-line to the current classification/clustering task. The inference in Gibbs sampling is very fast. An iteration during inference using Gibbs sampling involves computation of the order of the number of words in the new document and 50 iterations are good enough for our method. Next we show that these topic features can help in improving the accuracy of text classification tasks.

4.2.2 Improving Text Classification Accuracy

We tested the proposed method of feature generation on the RCV1 corpus [Lewis et al., 2004]. A sample collection of 10,000 random pages of Wikipedia was used as the corpus to generate topics. Only 10,000 pages were taken instead of the whole Wikipedia corpus as our current implementation of LDA cannot run on very large corpus. From this collection of 10,000 Wikipedia pages, 200 topics were extracted using LDA. Then for each document (d) in the RCV1 corpus, the probability that the document contains each of those 200 topics ($P(z_i|d) \forall i \in \{1 \ldots 200\}$) were estimated using the inference technique of LDA. These 200 topic probabilities act as 200 topic features of the document. Topic features are used in addition to the bag of words features.

Figure 4.7 shows the comparison of the F-measures between the bag of words representation (*baseline*) and the document representation with the topic features. Classification accuracy (F-measure) of 30 major categories of RCV1 corpus are shown in the figure. For each category a SVM was trained with the first 10% of the documents in the RCV1 corpus. The remaining 90% documents were used for testing. Since the topic features here are constructed from a random collection of Wikipedia pages, we report the average accuracy of 10 runs with 10 random samples of Wikipedia pages. It can be seen in the Figure 4.7 that for many categories the

F-measure has improved by the topic features. For some categories (e.g. GSPO, GPOL etc) the improvement is huge. Also, the topic features never hurts the performance as the F-measures with topic features is always as good as the F-measures of the baseline.

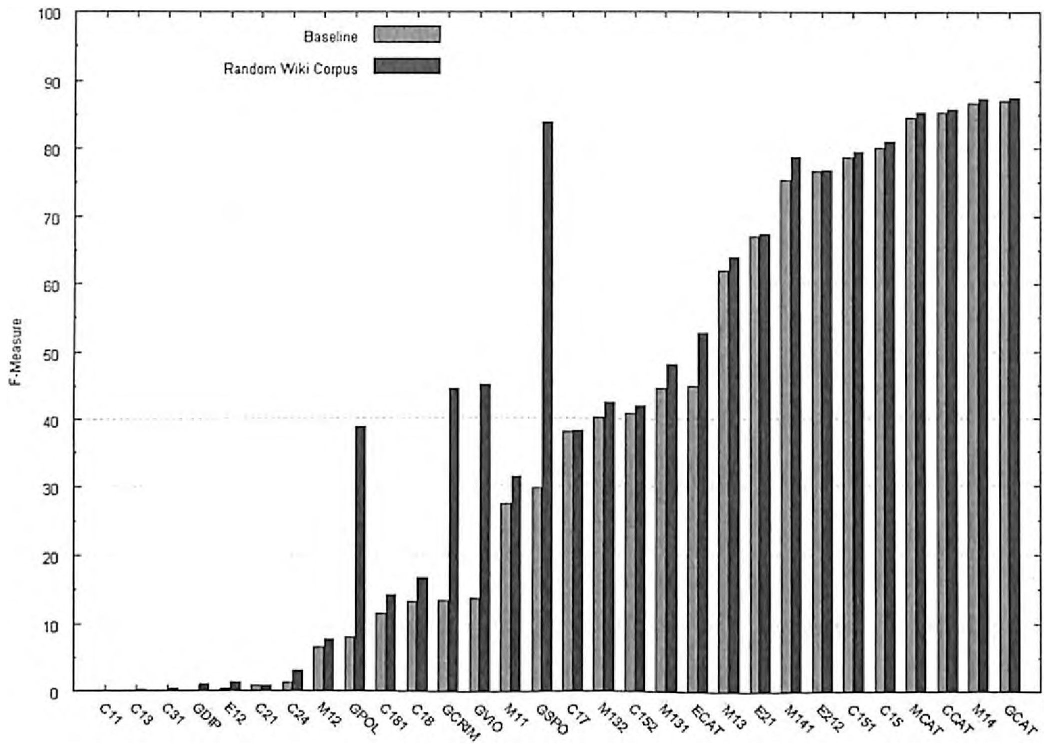


Figure 4.7: Classification accuracy: baseline vs topic features

Next we show the impact on classification accuracy as the amount of training data changes. Learning with few training data is of practical importance as gathering labeled training data is an expensive and often manual process. Figure 4.8 shows how the F-measures changes as we change the training data from 10% to 50%. The F-measure here is the average F-measure of the 30 categories (and 10 runs for the topic features). The interesting observation here is that the improvement with the topic features is more when the training data size is less.

At this point one question arises that instead of using an additional corpus could we have used the training set to extract the topics and hence the topic features. In fact in [Blei et al., 2003] this scenario was considered in the experiment without much improvement in classification accuracy. Training set is often small (due to manual effort involved in creating it) therefore may not be sufficient to apply a statistical technique like LDA to extract the topics. The extracted topics may be of poor quality and hence the topic features might not provide much meaningful information.

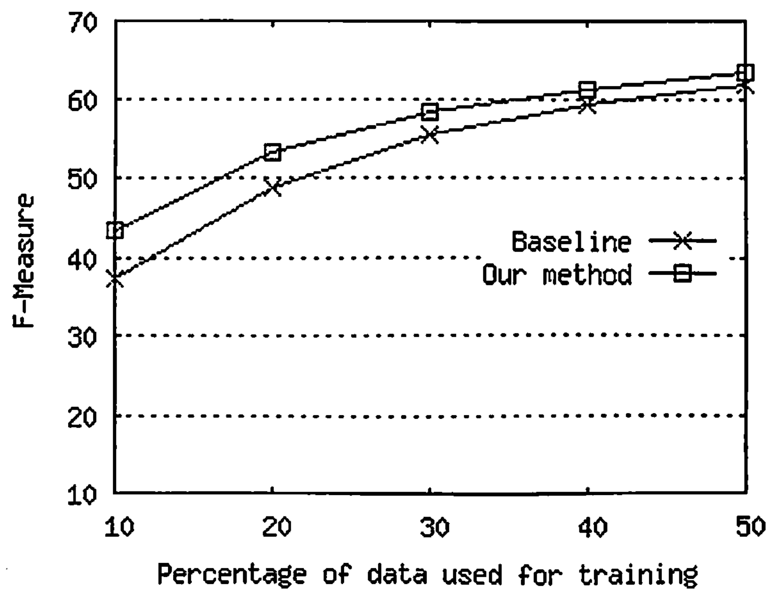


Figure 4.8: F-Measures at different training data size

One major problem of extracting topics from a random collection 10,000 Wikipedia pages is that the most of the meaningful topics are very general. For example, as shown in the Figure 4.6 the extracted topics resembles general topics like e.g. *music*, *politics*, *education* etc. A topic feature of a document indicates how much the document is aligned to the corresponding topic. Since the extracted topics are general, the topics features may not provide meaningful information when the target categories are very specific. This is evident in the results shown in the Figure 4.7. For categories like M11 (*EQUITY MARKETS*) or M132 (*FOREX MARKETS*), the improvements in F-measure are less. Because to categorize the documents under M11 or M132 a topic feature corresponding to a general topic like *music* may not provide much valuable information. On the other hand, major improvements are obtained for general and popular categories like GSPO (*SPORTS*), GPOL (*POLITICS*), GVIO (*WAR*) and GCRIM (*CRIME*).

We therefore conjectured that it will be better to extract the topics from a corpus that contains pages about the target categories of the current classification task. If the target categories are among the discussed topics in the corpus then LDA will most likely be able to extract topics that are (related to) target categories. In that case, the topic features can provide more useful information to the classification task.

4.2.3 Feature Generation from Related Corpus

Now we will describe a method to construct a corpus that are likely to contain pages discussing topics related to the current classification task in hand. We will use this corpus to extract topics using LDA and then apply the proposed method to generate topic features for the documents of the classification task. We will show that improvement in classification accuracy is more when the corpus is related to the classification task in hand.

Towards this we adopted a very simple method of corpus construction. We use the target category names to obtain a set of pages from the web. Each category in the RCV1 dataset has a category name, e.g. '*SPORTS* (GSPO), *EQUITY MARKETS* (M11) etc. We used the names of the 30 RCV1 categories as queries to the Google search engine. For each query (i.e. the name of the category) we downloaded the landing pages of the top 500 URLs. That way we were able to create a corpus containing around 5,500 pages. We call this corpus as the *Google Corpus*. Since the discussed topics of the Google corpus are related to the RCV1 categories, the LDA extracted topics are also likely to be related to those categories. Therefore the topic features of the documents can provide more useful information to the classifier. The results in the Figure 4.9 confirms this hypothesis.

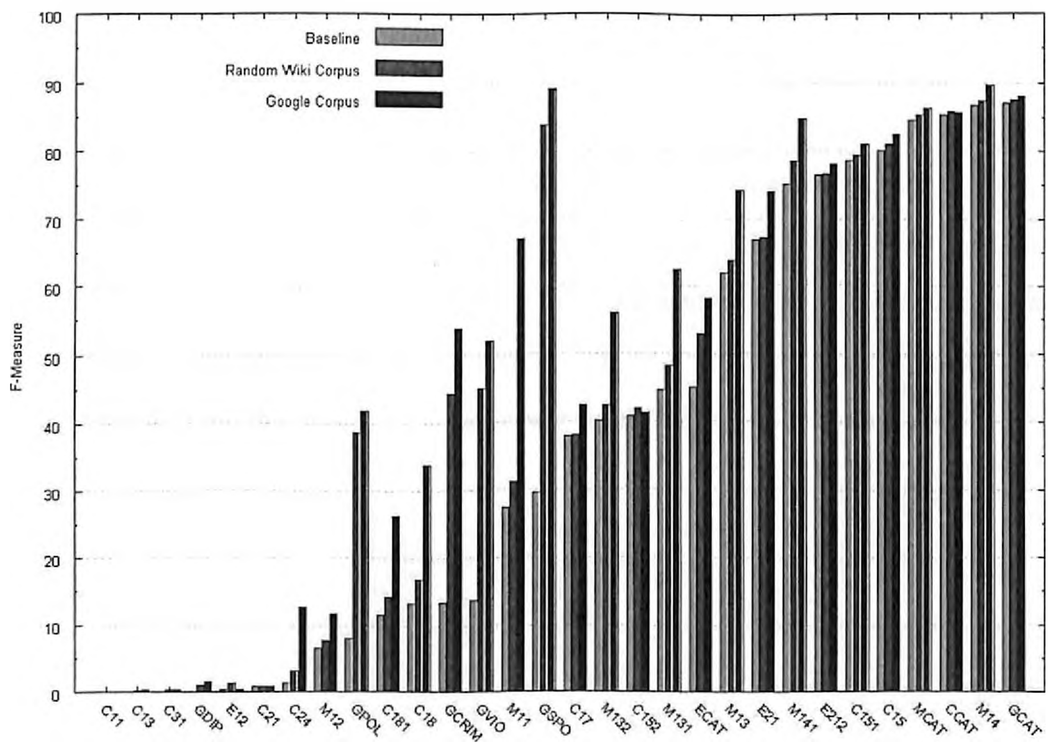


Figure 4.9: Classification accuracy: baseline, Wikipedia random corpus and Google corpus

Figure 4.9 shows the comparison of all the three methods, baseline, representation with topic features obtained from 10,000 random sample of Wikipedia pages and representation with the topic features obtained from the *Google Corpus*. It can be observed in the Figure 4.9, that in most cases the topic features from Google corpus helped a huge gain over the baseline and even over the Wikipedia corpus based approach. Quantitatively, taking average over the 30 categories, the F-measure achieved by the baseline, Wikipedia corpus and Google corpus based methods are 37.23%, 43.45% and 49.2% respectively. Topic features when obtained from random Wikipedia corpus improves the overall average classification accuracy by 16.71%. Whereas, when the topic features are obtained from a custom build corpus (Google corpus here) the improvement in average accuracy is 32.15%, almost double. Only in the cases where baseline classification accuracy is near zero or more than 80% the topic features could not provide more improvement. If the baseline accuracy is near zero or very high then it is very hard to improve anyway.

4.3 Conclusion

This chapter shows that the web resources can be used to obtain information about the topics described in a document. Such topic information, when obtained from a large web resource, provides broader contexts of the concepts described in the document. The broader contexts helps a classification or clustering algorithm to understand the content of the document better and thereby improves their accuracy. In this chapter, we discussed and developed novel methods to obtain topic features from a web corpus. The topic features provide topic information hence broader context information. We conducted several experiments on various text classification and clustering tasks and showed that a classification or clustering algorithm achieves much higher accuracy when the document representation contains topic features.

Chapter 5

Gathering Training Data for Web Page Classification

In the previous chapter, we described how a web corpus can be used to improve the accuracy of various classification and clustering tasks. In this chapter, we will demonstrate how different web sources can be utilized to gather training data to build a web page classifier for any arbitrary concept.

The organization of the chapter is as follows: First we will give the motivation and introduction of the problem (Section 5.1). We then describe methods to automatically obtain training data from different web sources (Section 5.2). Next we show that any one of these sources is itself not sufficient to provide high quality training data (Section 5.4). We need to leverage multiple sources to obtain training data for building a high accuracy classifier. We then develop (Section 5.5) different techniques to utilize multiple sources to build a high accuracy classifier.

5.1 Introduction

The classification of textual documents is a typical and important machine learning task with an enormous variety of applications. Web related examples where such classifiers are successfully applied include the categorization of news and blog content, spam filtering, and the filtering of web content with respect to user-specific interests.

The classical text classification literature focuses on controlled clean corpora, single-label problems, and most noticeably, it comes with the *independent and identically distributed (i.i.d.)* assumption, that is, the assumption that training and test set were sampled from the same underlying distribution. This assumption simplifies matters a lot, because in this case cross-validation results are reliable indicators for how well the target concept has been learned. But many web page classification tasks do not fit this classical setting of classification. Often the goal is to train classifiers that perform well on a broad variety of pages, ranging from clean dictionary entries over Internet shopping sites to noisy blogs. We cannot expect to get an i.i.d. sample from the target distribution, because it may not even be a fixed distribution but depend on the current user, and even the labels may sometimes be debatable. Therefore in practice sampling a training set from the "right" distribution would involve crawling representative sites from the web and manually labeling them, which is too tedious and expensive for many tasks.

In many practical settings of web page classification, the target classes or categories are not known in advance. A user will provide the category name and the system need to come up with a classifier on the fly. One example of such application is feed reader which is typically used to consume news, blogs and other web contents. In a feed reader, a user may like to organize all the news, blogs or other web pages about a particular concept (e.g. *German Car*) under a certain folder. In such scenario we cannot pre-build the classifier.

Building a classifier require labeled training data. Constructing labeled training data is a major bottleneck for supervised text mining applications, which e.g., gave rise to semi-supervised techniques like multi-view learning [Zhou and Li, 2005] that aim for minimizing the demand for labeled data. Like in the scenario of web page classification in many other domain as well, drawing training data following i.i.d. rule is impractically. Just recently, part of the research community started to work on various practically relevant, but much more difficult classification settings in which the classifiers are used with data distributions that do not obey the convenient i.i.d. rule at deployment time. Examples include learning under concept drift (e.g., [Forman, 2006]), where the target concept changes over time, transfer learning (e.g., [Banerjee, 2007]) where classifiers learned for similar but different concepts are exploited to learn the target concept, and learning under sample selection bias [Fan and Davidson, 2006], which subsumes a variety of other problems that involve non-i.i.d. sampled training sets.

In this chapter, we are addressing a practical aspect of web page classification: the construction of training data sets for broadly applicable multi-label web page classifiers. To this end, we propose a general framework that involves gathering and labeling data in an automatic fashion by utilizing various sources on the web. This approach avoids both the process of manually labeling web pages and the complexity of semi-supervised learning, while promising to produce highly accurate classifiers.

Two major contributions are shown in this chapter. First, we demonstrate that it is possible to build accurate web page classifiers without the hassle of labeling examples manually, while avoiding the pitfalls of unrealistic assumptions like stationary and homogeneous data collections that dominate the literature. Second, we show that the different web sources we exemplary use in the experiments in fact follow different underlying distributions, and that the diversity found in these sources cannot be ignored. We then study which sources and combination strategies are able to overcome the discovered problems, and hence allow to build the widely applicable classifiers we aim for.

5.2 Building Training Sets Automatically

The Web 2.0 provides a variety of resources that are promising for data mining problems. It leveraged joint efforts like tagging web contents and building up structured and semi-structured knowledge in electronic form, most prominently the online encyclopedia Wikipedia [UrlWikipedia] and the open directory of web pages, DMOZ [UrlDmoz]. The results are manually labeled, structured, or annotated pages, and – as we will show in the remainder of this chapter – can be used as cheap proxies for self-labeled data.

Our goal is to construct training sets for real-world web page classification. It is often problematic and inappropriate to assume that all web pages exclusively belong to a single category. We hence frame our learning problem as a set of binary classification problems, in which each document can belong to none, one, a few, or all categories. A binary classifier determines the estimated membership function for each category. To build a binary classifier for a category we need training set containing positive (belonging to the category) and negative (not belonging to the category) examples. We use different web sources to gather training corpora. A training corpus here is a (binary) labeled training set from a specific source for a specific category. We will be using the terms training *corpus* and *source* interchangeably.

For brevity, our studies focus on a representative set of four sources and corresponding extraction and labeling techniques. Further sources and different extraction methods can be integrated easily. The selection of sources will be motivated inline. Our extraction techniques were all chosen to be (i) simple enough to be reproducible, (ii) generic enough to cover a vast majority of potentially relevant concepts, and (iii) not to require extensive human interaction during the corpus construction phase. As we shall see, they still allow to build highly accurate classifiers. The specific problems (categories) this chapter exemplary focuses on will be described in more detail in the Section 5.3.1.

Open directory (DMOZ). DMOZ is a human edited web directory that contains almost 5 million web pages, categorized under nearly 600,000 categories. Each category in DMOZ represents a concept, and the categories are organized hierarchically.

The way DMOZ structures the data in terms of natural, human interpretable concepts and the fact that every page is interpreted and classified by a human annotator makes the DMOZ collection the most natural and probably most popular choice for building training sets in the web domain (e.g., [Davidov et al., 2004]). It basically provides a gigantic, manually labeled training set which could roughly reflect the distribution underlying the WWW.

For our experiments, we crawled an RDF dump of DMOZ from November 26, 2006, and we downloaded all pages referenced in that dump. Pages in the sub-tree rooted at any specific category can be thought of as the positive examples of the corresponding class, and the remaining pages as negatives.

We constructed training corpora from DMOZ by selecting 1000 positive examples by "breadth first search" in the corresponding sub-trees of relevant categories, and an equal amount of negative examples chosen at random from pages outside those trees.

Search engine (Google). Search engines provide a simple interface to obtain web pages for any given concept. We simply used the target category name (e.g., *photography*) as a search query to Google search engine [UrlGoogle] and used the landing pages of the first 1000 hits as positive examples; we selected 1000 negatives from DMOZ in the same way as described above. [1]

When considering search results as training examples, one should keep in mind that the pages are relevant in terms of e.g., the PageRank [Brin and Page, 1998] measure, but that they do not necessarily provide definitions for the queried term(s) or any kind of descriptive content. Many of the pages hence have a low signal. For example, start pages of topic-specific portals are legitimate search results, but they often contain more ads and navigational parts than descriptive text.

[1] A similar method was used to construct a related corpus in the Section 4.2.3 of the previous chapter. There we did not require negative examples and the corpus consisted only of the result pages from Google.

However, search engines have recently been recognized as useful for gathering data sets in different contexts, e.g., n-gram statistics [Keller and Lapata, 2003] and meaning discovery [Cilibrasi and Vitanyi, 2007]. Fergus et al. [Fergus et al., 2005] used the image search engine of Google to automatically gather training examples for object category recognition, and most recently a bootstrapping scheme has been suggested for text classification [Guzman-Cabrera et al., 2007].

Social bookmarking site (Del.icio.us). Del.icio.us [UrlDelicious] is a social bookmarking site that allows users to save and tag URLs. The tags used by multiple people for a particular URL are often quite representative of the concept mentioned in the web page of the URL. Pages tagged with a category name can be thought of as the positive examples for that category. Tagging is a very recent activity on the web, so not many text mining approaches utilize tags so far. One example is Yanbe et al. [Yanbe et al., 2007], who showed that tags can be used to improve the results of web search engines. An advantage of social tagging in our setting is that tags capture semantics in a way that resembles human perception at an appropriate level of abstraction, without introducing any unnatural assumptions, like categories being mutually exclusive. The tags are unstructured and freely composable.

Del.icio.us provides an API to obtain web pages with any specified tag. For example, given the category *photography*, we would simply use the Del.icio.us API to obtain pages tagged with the term "photography". Positive examples for Del.icio.us are obtained by crawling 1000 (wherever available) such pages. An equal number of negative examples from DMOZ are chosen as outlined above.

Encyclopedia (Wikipedia). Wikipedia is a community edited encyclopedia containing pages in many different languages. The English version of Wikipedia contains 3+ million pages and has 10+ million registered users [UrlWikiStat]. A recent study states that the coverage as well as the quality of Wikipedia is comparable to encyclopedia Britannica [Giles, 2005].

Important properties of Wikipedia in our context include (i) its semi-structured nature, with no labels being given a priori, but therefore (ii) deeper semantics when comparing to any of the other considered sources, and (iii) very clean pages that provide definitions and refer to related concepts. Wikipedia was recently successfully utilized for various text mining applications, including text categorization [Gabrilovich and Markovitch, 2006, Wang et al., 2007], text clustering [Hu et al., 2008], named entity disambiguation [Cucerzan, 2007], and improving search results [Milne et al., 2007].

To gather positive examples for a given concept, we constructed a Lucene [UrlLucene] index of the Wikipedia dump and used the target concept as the search query, in the same way as described for the Google corpus. Again, we used the top 1000 pages as our positive examples. For selecting negative examples for a concept, we excluded the first 2000 hits (returned by Lucene) from Wikipedia for each query under consideration, and then sampled 1000 negative examples from the remaining pages. We found that it is necessary to also use Wikipedia pages rather than DMOZ pages as negatives examples, because pages from Wikipedia and DMOZ have quite different characteristics.

We also tried graph-based page similarity calculation techniques to retrieve similar pages for a given concept. As long as the given concept can be characterized by a specific Wikipedia page, there are random walk techniques to obtain similar pages. We experimented with *Topic Sensitive PageRank* [Haveliwala, 2002] and *Green Measure* [Ollivier and Senellart, 2007], both of which did not work well. The top few results were usually good, but the noise level (unrelated pages) rapidly increased when going further down the list, so retrieving 1000 relevant pages for our training corpus was not possible with these techniques.

5.3 Experimental Setup

The previous section described ways of obtaining training examples from different web sources for almost any given category. This data can be used to train a set of binary classifiers in the next step. This section describes our test bed and justifies our evaluation scheme, before we move on to the actual experiments in the following section.

5.3.1 Data Sets for Evaluation

We selected a set of 10 diverse categories for our empirical evaluation. For the sake of simplicity and clarity, our concepts were chosen as to satisfy the following constraint: Each concept had to match a category name in DMOZ and a tag in Del.icio.us. This does not narrow down applicability in practice; if there is no exact match then a number of very similar categories/tags can easily be substituted.

We used the categories *health*, *shopping*, *science*, *programming*, *photography*, *linux*, *recipes*, *web design*, *humor* and *music*, which span across three different levels of the DMOZ hierarchy and therefore vary considerably in terms of specificity. For each of these 10 concepts, we constructed a separate training corpus from each of the four different sources. Each corpus contained 1000 positive examples and an equal number of negatives as discussed in Section 5.2.

We preprocessed the raw HTML pages by removing any non-textual content (HTML tags and scripts), tokenized the page, removed stop words, and applied a Porter stemmer. We removed all pages that contained less than 50 words at that point, because they usually did not refer to the concept under consideration. For the experiments, we finally applied the standard TF-IDF weighting and built binary classifiers for each category using the SMO-SVM of the Weka library [Witten and Frank, 2005] with the default settings. Our evaluation measure is classification accuracy averaged over all categories, which is similar to F-Measure in our case, because our corpora have balanced class distributions.

5.3.2 Evaluation Strategy

In the common data mining setup, we would simply cross-validate our learning algorithm on the data sets mentioned above. One round of cross validation involves partitioning the training data, training the classifier in one partition and testing it on the other partition. Cross validation accuracy of a classifier is representative of its true performance only when the test and training data is generated from the same distribution. We do not assume the training corpus to resemble the distribution of web pages at deployment time, however. We will hence only use cross-validation in the specific case where we evaluate classifiers on the same source it was trained on. For the most part, we will switch to evaluation schemes where we evaluate on a single source that is not available during training, and we will compare different strategies for constructing well suited training sets from the remaining (three) sources in this setting. Therefore, our evaluation strategy can be thought of as hold-out evaluation, but at the level of data sources.

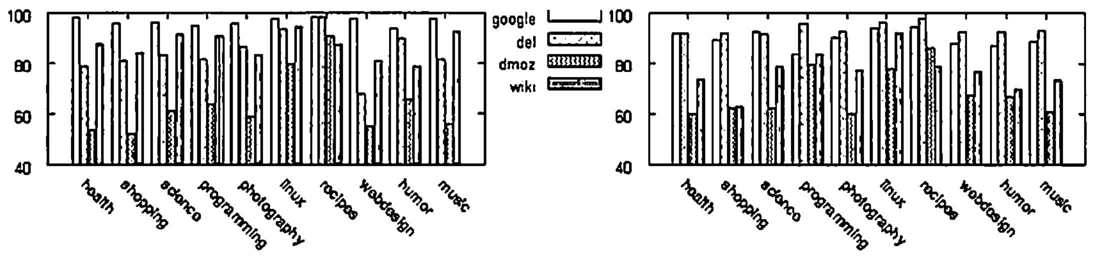
Our rationale is that each corpus will typically contain noise and systematic mistakes. The DMOZ concept of *photography* does not subsume *underwater photography*, for example. The mere fact that a DMOZ category name, a Del.icio.us tag, and a Wikipedia page title are identical does not necessarily imply identical underlying semantics. We assume that there is still agreement between large parts of the different taxonomies, tags, and labels, respectively. This agreement is rooted in a common conceptualization of human annotators. In real deployment setting also a web page classifier will face the challenge of identifying pages belonging to common conceptual class but changed underlying semantics. Therefore, evaluating a classifier on a source that is not used during the training gives better indication of the classifier's performance at deployment. The evaluation of a classifier trained under a different distribution than the one it is deployed on is known from various settings, e.g. classification under sample selection bias [Fan and Davidson, 2006].

Our primary goal is hence to find techniques that allow to learn good classifiers for each source, although that source is not part of the training set. To train the classifier we can use any or all of the remaining three sources. First we will capture and quantify the difference between individual corpora. For that we train the classifier on a single training corpus and test the classifier on the hold-out corpus. That is, we evaluate ordered pairs (i, j) of corpora by training a classifier on corpus i and then measuring its performance on corpus j. We refer to this strategy as *cross-corpus evaluation*. For any pair of corpora that share a common underlying distribution, the expected cross-validation and cross-corpus evaluation results would be identical, whereas, for any pair of highly incompatible corpora applying each of the classifiers to the other corpus would result in much lower classification accuracies. As we will see next that cross corpus evaluation establishes the fact that our training corpora are quite different from each other. Then we will explore different strategies to use multiple different corpora during training and evaluate the classifier on a hold-out (test) corpora.

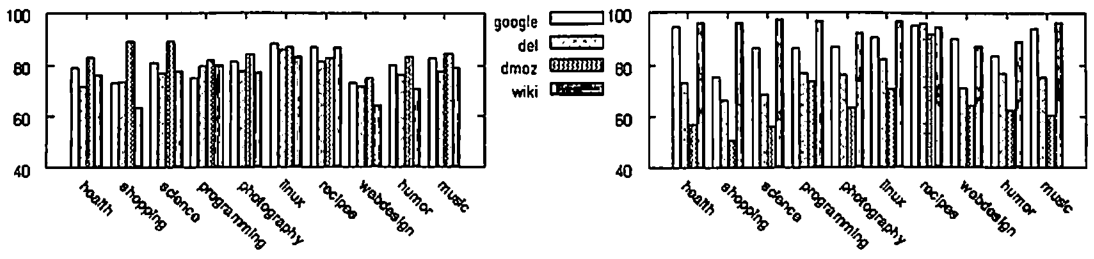
5.4 Experiments with a Baseline Method

We will first discuss the results of the corresponding cross-corpus evaluation. Figure 5.1 shows an overview of all the pairwise performances. For each pair of category and data source we trained a separate classifier and applied it to all data sets of the same category. We substituted tenfold cross-validation results whenever the same source was used for training and testing. We will (for the most part) discuss aggregates, i.e., accuracies averaged over all of our 10 concepts in this paper. Table 5.1 shows such aggregates over all results where the classifier was built from the same source (row) and applied to another source (column). Again, all entries on the main diagonal are averages of the corresponding 10-fold cross-validation accuracies.

For convenience, we set the highest accuracy achieved by any other corpus for each test corpus in bold (maximum of each column) and the lowest accuracy in italics. The numbers in



(a) Training set: Google Training set: Delicious



(b) Training set: DMOZ Training set: Wikipedia

Figure 5.1: Pairwise cross-corpus evaluation results at the category level. Each of the four blocks describes the result for a common training set. The colors of the bars indicate the test sets.

bold are reasonable upper-bounds for how much of the "real" concept is reflected by the corpus; it shows the maximum possible agreement with a classifier built from an independent data set that is only connected to the training set via a common concept name used for both the corpus constructions. The 10-fold cross-validation accuracies (main diagonal) can be referred to as the corresponding upper-bounds of the accuracy we can hope for when given a sample of the same type, because the error rate under these (effectively) i.i.d. samples is an artifact of the learning strategy and not caused by a difference between train and test distribution.

It is clearly evident from the cross-corpus evaluation that the individual sources are quite different. From the table 5.1, we can see that any individual source is not good enough to build a classifier that can perform well across all other sources. Another interesting finding is the poor performance of the community-built DMOZ collection of web pages in our experiments, compared to the much simpler Google corpora. The third row of Table 5.1 shows that the classifiers built on Dmoz corpus generally achieve very low accuracy when tested on other corpora. Although the cross-validation accuracy is moderate (diagonal element). We think this contradicts common beliefs, and that the problems we found illustrate why corpora systematically contradict each other for some concepts: The DMOZ categories are not organized in terms of a taxonomy (tree), but of a directed graph. Care has to be taken when constructing the sub-tree

	Google	Delicious	DMOZ	Wikipedia
Google	(96.44)	**84.17**	*63.42*	**87.12**
Delicious	**90.00**	(93.54)	**68.36**	76.71
DMOZ	*79.98*	77.15	(83.92)	*75.84*
Wikipedia	88.28	*76.21*	65.05	(94.26)

Table 5.1: Results of cross-corpus evaluation. Rows are training sets, columns the test sets.

for a concept, so that all the positive examples of the concept are covered and cannot be sampled as negatives. This is hard, however, because a large number of categories in DMOZ are spread out over the concept graph without any proper path connecting the pieces. For example, the top level categories *Regional*, *Reference*, and *News* cover many categories that reoccur in other parts of DMOZ without any connecting path. In turn, there are debatable links at the level of the concept hierarchy that affect large sub-trees of documents in a systematic way, i.e., they might become false positives. This does not compromise the cross-validation accuracy, because in this case the evaluation happens on uniform sub-samples of the same corpus, so the classifier might capture all these conventions quite well. However, the classifier's concept may not reflect the natural meaning very well, and consequently does not generalize to other corpora. This is why cross-corpus validation is a useful tool. We addressed noise issues during corpus construction, e.g. by respecting the complex link structure and by disregarding known-noisy nodes; still we expect our Dmoz corpora to contain unreliable labels that hurt performance.

Having established that any single source is not sufficient to build a highly accurate classifier, we now explore strategies to exploit more than one training sources to build a classifier. As mentioned earlier, the evaluation strategy will be to set aside one source and build the classifier from the remaining three sources. The baseline method just merges the data of the three corpora that are available for training and fits an SVM classifier to that single training data set. The baseline method simply ignores that our web corpora might all have different characteristics.

The second line in Table 5.2 shows the results of the baseline method that simply aggregates the training corpora. For each category we combined the three different training sources with equal weights and tested on the remaining corpora (column). For example, when a single training corpus is created by combining the data (positive and negative) of Del.icio.us, DMOZ and Wikipedia for each concept, it gives 76.94% accuracy on average on the Google corpora of 10 different concepts. Note that the training sets in this experiment are 3 times larger than in the cross-corpus matrix. Since we do not assume to know the test set at training time, we aim for a strategy that gives us performances that are close to the bold numbers in Table 5.1, repeated in line 1 of Table 5.2. In this light, the performance of the baseline strategy is very poor. Surprisingly, in 3 out of 4 cases this method performs just about as good as the worst single-source classifier, see Table 5.1. When testing on DMOZ, the result is in an acceptable range, but low in absolute terms.

Looking at the Figure 5.1 and inspecting some pages from the different sources confirms the assumptions made in the Section 5.3.2; the corpora differ in fact considerably, and mixing them blindly results in far noisier, heterogeneous, and non-separable corpora that contain examples that sometimes systematically contradict each other because of different concept definitions used by different sources.

5.5 Leveraging Multiple Sources

In response to the bad performance of the baseline method and the demonstrated differences between the distributions underlying different web sources we will now evaluate two different strategies to utilize multiple training corpora. The first one is an ensemble technique, the second one refines the baseline method by introducing example weights before training.

5.5.1 Majority Vote

Leaving Dmoz the other numbers in the first line of the Table 5.2 are respectable accuracy number. Recall that the first row of the Table 5.2 is the bold numbers of the Table 5.1 (maximum achievable accuracy by any other corpus on a test corpus). This means, for a particular test corpus among the remaining three corpora there usually is at least one that would work very well compared to the baseline method. For this reason, we did not mix training data from different sources for the experiments in this subsection, but trained a separate classifier for each training corpus. We applied Platt's scaling [Platt, 1999] to turn SVM outputs into calibrated probability estimates.

As our remaining challenge, it is generally unknown which training set provides good performance on a previously unseen example. The predictive performance of training sets apparently depends on both the concept and the test set. A natural goal is to get an overall predictive performance that is close to the best individual classifier. This goal is known as tracking the best expert in the literature on concept drift [Kolter and Maloof, 2005], where the term *expert* refers to individual classifiers. If true labels were revealed after classifying each instance then we were in the setting of classifying data streams under concept drift.

Instead, we address the harder, but more relevant case, in which the true labels are not revealed, so we cannot dynamically adapt classifier weights. In this case, we can still pick appropriate weights offline. Line 3 in Table 5.2 shows the results of an unweighted majority vote, where we averaged the soft predictions of three classifiers and tested on the indicated (column header) fourth corpus of that category.

It can be seen that – by just keeping the corpora separate during training – we already improved drastically, and got results that are (on average) much closer to the best available individual classifier shown in the first line of Table 5.2. Under our weak assumptions, this implies that these classifiers generalize better to unseen corpora (or different types of web pages). Weighting the classifiers, e.g., by their average cross-corpus performance did not further improve the results in our experiments, so we skip the details.

5.5.2 Weighting Training Data by Confidence

The second strategy is to encourage agreement between the different views on the same concept. It is inspired by multi-view learning, see e.g. [Zhou and Li, 2005], a semi-supervised technique where each data point has multiple representations. For multi-view learning techniques, the generalization error can be upper-bounded surprisingly well if the learner manages to enforce agreement on both the labeled and unlabeled data across the different views. The theory requires unlabeled data and test set to follow the same distribution.

Our setting is different as we do not have an unlabeled sample from the target distribution. But still we can encourage agreement between the classifiers regarding their *training* sets. Before describing this in more detail, it is worth fleshing out the different reasons why an example e_B sampled from source B could be misclassified by a classifier C_A trained on source A:

1. The example e_B is noisy, that is, the labeling process just failed or someone inserted a bad reference between concepts into DMOZ etc.

2. The abstraction from the data collected from A to the function C_A done by the learner is imperfect. Even all the cross-validation results in Table 5.1 are below 100%. So the label of e_B is correct, C_A errs.

3. The classifier and the label of the examples are both correct, but the conventions between the two corpora A and B are inconsistent. This is no "noise" in the classical sense, but a different kind of problem, related to transfer learning: Concepts are similar but not identical across different sources.

Given the lack of reliable information, it should be clear that it is very hard to address point 3, if we have to handle points 1 and 2 at the same time. For this reason, our solutions cannot be as grounded theoretically as techniques in multi-view learning, but still intuitively compelling.

We start with the classifiers that we trained in the last subsection. Our advantage over the setting sketched above is that we have multiple classifiers that we can consult in order to decide on the expected utility of e_B. This allows us to incorporate the confidence of all the different "views" captured by classifiers trained from different, independent sources, which helps to reduce the impact of any source-specific noise.

To this end, we prepare a weighted version of each category-specific training set. Lets assume we have A, B, C sources for training and have set aside the source D for testing. For computing the weight of any specific example e_B, with word vector x_B and label y_B from source B, we consult the classifiers C_A and C_C built from the sources A and C respectively. Each of these classifiers C_A and C_C provides calibrated probability estimate $P(y_B \mid x_B)$ that the example e_B belongs to the category y_B. We use the average of those estimates as the weight for e_B. As desired, examples hence require the agreement of classifiers trained from different sources in order to receive a high weight. In the next step, we join all examples from the three training sources to a single corpus per category, similar to the baseline method. The difference is that we use the weights during the subsequent step of classifier training. Finally, we train a classifier on this weighted training set and apply the classifier to the test corpora of the fourth source (source D). Note that the test corpora are at no point used during training.

Line 4 in Table 5.2 shows the averaged accuracies for this weighting technique. Compared to the equal weight combination (line 2), the accuracy improved drastically. It is now comparable to the performance of majority voting. This indicates that corpus-specific noise is the main reason for the bad baseline performance, and that it can be mitigated by seeking agreement between the various sources.

We want to take this idea one step further. Bad conventions, missing or questionable links between categories, and other kinds of white and systematic noise all share the property that they are not found across multiple sources, but are local problems. As a final refinement of the

	Google	Delicious	DMOZ	Wikipedia
Best single cross-corpus result (5.4)	90.00	84.17	68.36	87.12
Equal weight combination (5.4)	76.94	76.47	68.74	76.79
Majority Vote (5.5.1)	92.95	84.45	65.10	84.44
Weighted training instances (5.5.2)	**93.88**	84.65	66.08	85.73
Weighting & noise elimination (5.5.2)	93.29	**87.52**	**70.21**	**87.50**

Table 5.2: Results of the different methods discussed in Sections 5.4 and 5.5.

weighting strategy, we exclude all examples for which the majority (here: "both") of classifiers predicts the opposite label when making a discrete (boolean) prediction, and assign the highest possible weight of 1 whenever all classifiers predict the label assigned to the example. That is, in this weighting scheme, the example e_B will be excluded from the training set if the both classifiers C_A and C_C predict class label other than y_B. On the other hand e_B will get weight 1 if C_A and C_B both predict the class label as y_B. In the latter case, all the classifiers agree that an example has the correct label. In the former case, the label of the example is very questionable when leaving the context of the specific source it came from. We conclude that it is not helpful to include it at all, but that we are either in case 1 or in case 3 above. Most, but not all of our classifiers confirm that the example has the correct label attached. As before, we weight these examples by confidence that their labels are correct in a general scope.

The last line in Table 5.2 shows the results for this stronger noise reduction and emphasis on agreement. The averaged accuracies are higher for Del.icio.us, DMOZ and Wikipedia compared to just weighting training instances (line 4). In case of the Google data set, the previous results were already quite good ($\approx$ 93%) and we observe no further improvement. Overall, the results of this method are better than all other methods we tried so far. When averaged over of all test corpora (average of the rows in Table 5.2) the baseline achieves 74.74% accuracy. Whereas the weighting & noise elimination method achieves overall average accuracy of 84.63%, an improvement of 13.23%. Note that by utilizing the cross-corpus diversity we were able to outperform the best single-corpus accuracies, see line 1. Even though the best single-corpus method requires knowing the test corpus affront to chose the best training corpus and therefore not realistic.

5.6 Conclusions

In this chapter, we showed that several web sources can be utilized to build highly accurate classifiers with low efforts and costs. Our evaluation assumed that we want a classifier that give good results on a variety of different sources. For that purpose we used hold-out evaluation at the corpus level. The cross-corpus evaluation quantifies the discrepancy between corpora constructed from different sources. It turned out that ignoring the diversity of sources is a surprisingly bad strategy. In response, we developed a cross-corpus example selection and weighting technique based on a set of mild assumptions that takes advantage of this diversity. With our novel learning schemes we show that it is possible to build classifiers that perform consistently well across different sources.

Chapter 6

Conclusion

The World Wide Web is an enormous source of information that can be used by information processing tools. The challenge is that the information is unstructured, noisy and ever evolving. However, it is possible to make sense from this cacophony of information when an aggregated view is taken. The work in this thesis shows that by deploying proper aggregation techniques it is possible to eliminate or greatly alleviate the impact of noise and to gather useful information from the web.

We have developed techniques that use the web as an information source in order to:

- answer quantity queries
- determine topic features of documents
- retrieve sample documents for a given topic

The developed techniques obtain information by aggregating evidences from different web sources. Firstly, for quantity consensus queries our collective ranking algorithms aggregate evidences from multiple snippets in a quantity interval and achieve nearly 20% higher accuracy compared to the state of the art approaches. Secondly, the topic features of a document provide aggregated views of the contexts in which the terms of the document appear across different web sources. Such aggregated contexts capture the usage of the terms of the document in broader contexts and provide valuable inputs to the classification and clustering algorithms. Thirdly, sample documents for a topic were obtained by utilizing multiple web sources and aggregating the documents of different sources with proper weighting. We demonstrated that it is possible to build a high accuracy classifier for a topic from training data constructed from this set of documents.

Our work also contributes to the growing body of work in utilizing the collective human effort in Web 2.0 resources, such as Wikipedia, Dmoz, and Delicious, to improve machine intelligence. We used Wikipedia to disambiguate entities in our corpus and to support a more specific query language in our quantity search. Otherwise, such entity disambiguation and specific querying would have required rigorous semantic information of the documents and substantial user input. We demonstrated that Wikipedia and other web corpora can be used to obtain topic features of documents. The topic features substantially improve the accuracy of text classification and clustering algorithms. The topic features also reduce the amount of training data required in classification to achieve a desired accuracy. Generating training data typically requires substantial manual effort. We demonstrated that Web 2.0 resources and a web search engine can be used to obtain training data for web page classification with minimum manual effort.

Web 2.0 resources are where the efforts of thousands of users are captured and stored. Therefore, Web 2.0 resources provide tremendous opportunity to use the collective human effort in assisting intelligent information processing or other machine learning tasks. The thesis made progress in tapping this collective effort to make various intelligent information processing tools more useful and effective.

6.1 Summary of Results

Quantity Consensus Query (QCQ) We developed novel algorithms to aggregate answers of a QCQ from hundreds of pages across the web. We show that typical signals used in entity ranking, like rarity of query words and their lexical proximity to candidate quantities, are very noisy. Our algorithms learn to score and rank quantity intervals directly, combining snippet quantity and snippet text information. We report on experiments using hundreds of QCQs with ground truth taken from TREC QA, Wikipedia Infoboxes, and other sources, leading to tens of thousands of candidate snippets and quantities. Our algorithms yield about 20% better MAP and NDCG compared to the best-known collective rankers, and are 35% better than scoring snippets independent of each other.

Topic Features from Wikipedia We demonstrated that the topic features obtained from Wikipedia can help in two different tasks.

Clustering Short Texts: The accuracy of clustering short text documents improves when the documents are represented in an enriched feature space consisting of Wikipedia features and the bag of words features. In a clustering task of around 15000 short news clips, we showed that the accuracy of well known clustering algorithms improved by more than 33% on an average when the news clips are represented in this enriched feature space.

Inductive Transfer: The Wikipedia features (i.e. the topic features obtained from Wikipedia) are more stable features of a document than the words of the document. The words may vary even between documents describing the same topic whereas the Wikipedia (topic) features are likely to be similar. Therefore a classifier trained using the Wikipedia features is more effective in classifying documents even when the word distribution of the documents are different from the training set. This property of Wikipedia features was utilized in inductive transfer over text classifiers.

We used the Wikipedia features in an inductive transfer setting for "daily news classification task". In this task, every day there are some labeled news articles for training a classifier. The trained classifier is used to predict the class labels of the remaining news articles of that day. Instead of throwing away the classifiers built in the past, an inductive transfer setting was used to re-use those past classifiers. We demonstrated that when the past classifiers are trained on the Wikipedia features of the news articles, the classification accuracy of the news articles improves significantly. In our experiments, we observed major improvement in terms of F-measure for several categories in the RCV1 corpus. We also observed that doing inductive transfer from classifiers trained on Wikipedia features did not hurt the performance. This means that it did not have a negative transfer effect.

Obtaining Topic Features using Topic Modeling We have proposed a novel feature generation method using a topic modeling technique. The proposed method addresses many shortcomings of the retrieval based feature generation method from Wikipedia. Firstly, it is based on a more principled approach and therefore provides a way to select the parameters used in the system. It only contributes to the limited growth of the feature space and therefore it does not have much of a performance penalty on the downstream learning algorithms. We conducted experiments to demonstrate the usefulness of the topic features obtained using this method. The experiments show that the topic features when obtained using the proposed method can improve the accuracy by more than 32% on an average across 30 different RCV1 categories.

Training Data Generation Web page classification is a challenging task because of the diverse nature of the web pages. Obtaining a good representative sample of training examples is a laborious task while building a web page classifier. We developed methods to automatically obtain training examples from different web sources to build a highly accurate web page classifier. We demonstrated that different sources have different characteristics and it is not possible to blindly mix the data obtained from different sources. We addressed the issue of specificity of different sources by using several techniques to combine the training data from different sources. Our technique for combining the training data achieves 13.33% higher classification accuracy as compared to the baseline technique of blindly mixing the data from different sources.

6.2 Scope for Future Work

The techniques developed in the thesis obtain evidences from the web by simple means and focus more on aggregating the evidences using statistical methods to get useful information. We have demonstrated that aggregating the noisy evidences using statistical methods is definitely a promising approach to obtain useful information from the web. But this approach can also be augmented with the rich literature available in the area of deep semantic understanding of natural language. For example, it might be possible to obtain better results by using natural language processing techniques while obtaining QCQ snippets, Wikipedia features or sample pages for a topic. However many issues, including performance and the free and evolving languages used in web pages, would need to be addressed to do this.

Another direction in which our work can be extended is to apply the developed techniques on media types other than text. The web is an enormous source not only for textual documents but also for images and videos. For example, it will be interesting to see whether the web sources can be used to obtain topic features for image classification/clustering or can be used to obtain training data for image categorization. In general, we believe the web can potentially act as a valuable source of information in processing non-textual information as well.

Bibliography

Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *ICDL*, pages 85–94. ACM, 2000.

James Allan. Introduction to topic detection and tracking. pages 1–16, 2002.

Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. A language modeling framework for expert finding. *IPAM*, 45(1):1–19, 2009. ISSN 0306-4573.

Somnath Banerjee. Boosting inductive transfer for text classification using wikipedia. In *ICMLA '07: Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 148–153, Washington, DC, USA, 2007. IEEE Computer Society.

Somnath Banerjee, Krishnan Ramanathan, and Ajay Gupta. Clustering short texts using wikipedia. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 787–788, New York, NY, USA, 2007. ACM.

Ron Bekkerman. Using bigrams in text categorization. Technical report, Center for Intelligent Information Retrieval (CIIR) , University of Massachusest Amherst, 2004.

Paul N. Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 11–18, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-483-6.

David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:2003, 2003.

David M. Blei, Thomas L. Griffiths, Michael I. Jordan, and Joshua B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems*. MIT Press, 2004.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, New York, NY, USA, 1998. ACM. ISBN 1-58113-057-0.

Kurt D. Bollacker and Joydeep Ghosh. A supra-classifier architecture for scalable knowledge reuse. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 64–72, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X.

Eric Brill, Susan Dumais, and Michele Banko. An analysis of the askmsr question-answering system. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 257–264, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998. ISSN 0169-7552.

Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *EMNLP*, pages 724–731. Association for Computational Linguistics, 2005. URL `http://acl.ldc.upenn.edu/H/H05/H05-1091.pdf`.

David Buscaldi and Paolo Rosso. Mining knowledge from wikipedia for the question answering task. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, 2006.

Michael J Cafarella, Christopher Re, Dan Suciu, Oren Etzioni, and Michele Banko. Structured querying of web text: A technical challenge. In *CIDR*, pages 225–234, 2007. URL `http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p25.pdf`.

Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002. ISBN ISBN 1-55860-754-4. URL `http://www.cse.iitb.ac.in/~soumen/mining-the-web/`.

Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 88–96, New York, NY, USA, 2008. ACM.

Olivier Chapelle, Quoc Le, and Alex Smola. Large margin optimization of ranking measures. In *NIPS 2007 Workshop on Machine Learning for Web Search*, 2007. URL `http://research.microsoft.com/CONFERENCES/NIPS07/papers/ranking.pdf`.

Tao Cheng, Xifeng Yan, and Kevin ChenChuan Chang. EntityRank: Searching entities directly and holistically. In *VLDB*, September 2007. URL `http://www-forward.cs.uiuc.edu/pubs/2007/entityrank-vldb07-cyc-jul07.pdf`.

Rudi L. Cilibrasi and Paul M. B. Vitanyi. The google similarity distance. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):370–383, 2007. ISSN 1041-4347.

Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting redundancy in question answering. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 358–365, New York, NY, USA, 2001. ACM.

Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *EMNLP 2007: Empirical Methods in Natural Language Processing*, 2007.

Dmitry Davidov, Evgeniy Gabrilovich, and Shaul Markovitch. Parameterized generation of labeled datasets for text categorization based on a hierarchical directory. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 250–257, New York, NY, USA, 2004. ACM. ISBN 1-58113-881-4.

Owen de Kretser and Alistair Moffat. Effective document presentation with a locality-based similarity heuristic. In *SIGIR*, pages 113–120, 1999. ISBN 1-58113-096-1.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

Wei Fan and Ian Davidson. Reverse testing: an efficient framework to select amongst classifiers under sample selection bias. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 147–156, New York, NY, USA, 2006. ACM.

Hui Fang and ChengXiang Zhai. Probabilistic models for expert finding. In *ECIR*, pages 418–430, 2007. URL `http://www.eecis.udel.edu/~hfang/pubs/ecir07-expert.pdf`.

Christian Fellbaum, editor. *WordNet: An Electronic Lexical Database (ISBN: 0-262-06197-X)*. MIT Press, 1998.

R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning object categories from google"s image search. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1816–1823, Washington, DC, USA, 2005. IEEE Computer Society.

George Forman. Tackling concept drift by temporal inductive transfer. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 252–259, New York, NY, USA, 2006. ACM.

Evgeniy Gabrilovich and Shaul Markovitch. Feature generation for text categorization using world knowledge. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 1048–1053, 2005.

Evgeniy Gabrilovich and Shaul Markovitch. Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence, Boston, MA*, 2006.

Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of The Twentieth International Joint Conference for Artificial Intelligence*, pages 1606–1611, Hyderabad, India, 2007.

Jim Giles. Internet encyclopedias go head to head. *Nature*, 483:900–901, 2005.

Mark Girolami and Ata Kaban. On an equivalence between plsi and lda. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 433–434, 2003.

T. L. Griffiths and M. Steyvers. Finding scientific topics. *National Academy of Sciences, U S A*, 101 Suppl 1:5228–5235, April 2004. ISSN 0027-8424.

Rafael Guzman-Cabrera, Manuel Montes, Paolo Rosso, and Luis Villasenor. Improving text classification by web corpora. In *Proc. of the 5th Atlantic Web Intelligence Conference (AWIC)*, 2007.

Taher H. Haveliwala. Topic-sensitive pagerank. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 517–526, New York, NY, USA, 2002. ACM.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large margin rank boundaries for ordinal regression.* MIT Press, Cambridge, MA, 2000.

Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Can social bookmarking improve web search? In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 195–206, New York, NY, USA, 2008. ACM.

G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. pages 282–317, 1986.

Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, 1999.

Jian Hu, Lujun Fang, Yang Cao, Hua-Jun Zeng, Hua Li, Qiang Yang, and Zheng Chen. Enhancing text clustering by leveraging wikipedia semantics. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 179–186, New York, NY, USA, 2008. ACM.

Fei Huang, Ying Zhang, and Stephan Vogel. Mining key phrase translations from web corpora. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 483–490, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48, New York, NY, USA, 2000. ACM. ISBN 1-58113-226-3.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM.

Thorsten Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 377–384, New York, NY, USA, 2005. ACM.

Thorsten Joachims. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142. Springer Verlag, Heidelberg, DE, 1998.

Thorsten Joachims, Hang Li, Tie-Yan Liu, and ChengXiang Zhai, editors. *Learning to Rank for Information Retrieval*, Amsterdam, 2007. URL `http://research.microsoft.com/users/LR4IR-2007/`.

Karen Sparck Jones. Automatic keyword classification for information retrieval. 1971.

Frank Keller and Mirella Lapata. Using the web to obtain frequencies for unseen bigrams. *Comput. Linguist.*, 29(3):459–484, 2003. ISSN 0891-2017.

Jeongwoo Ko, Eric Nyberg, and Luo Si. A probabilistic graphical model for joint answer ranking in question answering. In *SIGIR*, pages 343–350, 2007. ISBN 978-1-59593-597-7. URL `http://www.cs.cmu.edu/~jko/paper/sigir.pdf`.

Jeremy Z. Kolter and Marcus A. Maloof. Using additive expert ensembles to cope with concept drift. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 449–456, New York, NY, USA, 2005. ACM.

Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. Collective annotation of wikipedia entities in web text. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 457–466, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9.

David D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50, New York, NY, USA, 1992. ACM.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization reserch. *J. Mach. Learn. Res.*, 5:361–397, 2004. ISSN 1533-7928.

Ping Li, Christopher J. C. Burges, and Qiang Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. MIT Press, 2007.

Tie-Yan Liu. Learning to rank for information retrieval. Tutorial at SIGIR, 2008. URL `http://www.sigir2008.org/tutorials.html#t7`.

Tie-Yan Liu, Tao Qin, Jun Xu, Wenying Xiong, and Hang Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR Workshop*, 2007. URL `http://research.microsoft.com/users/LETOR/`.

Rada Mihalcea and Andras Csomai. Wikify!: linking documents to encyclopedic knowledge. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 233–242, New York, NY, USA, 2007. ACM.

David N. Milne, Ian H. Witten, and David M. Nichols. A knowledge-based search engine powered by wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 445–454, New York, NY, USA, 2007. ACM.

Veronique Moriceau. Numerical data integration for cooperative question-answering. In *EACL Workshop on Knowledge and Reasoning for Language Processing*, pages 42–49, 2006. URL `http://www.aclweb.org/anthology/W/W06/W06-1808.pdf`.

Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Mach. Learn.*, 39(2-3):103–134, 2000. ISSN 0885-6125.

Y. Ollivier and P. Senellart. Finding related pages using green measures: An illustration with wikipedia. In *AAAI Conference*, 2007.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

Desislava Petkova and W Bruce Croft. Proximity-based document representation for named entity retrieval. In *CIKM*, pages 731–740. ACM, 2007. ISBN 978-1-59593-803-9. URL `http://portal.acm.org/citation.cfm?id=1321440.1321542`.

John C. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Adv. in Large Margin Classifiers*, 1999.

John Prager, Sarah Luger, and Jennifer Chu-Carroll. Type nanotheories: a framework for term comparison. In *CIKM '07: Proceedings of the 16th ACM international conference on Information and knowledge management*, pages 701–710. ACM, 2007. ISBN 978-1-59593-803-9.

Xiaoguang Qi and Brian D. Davison. Classifiers without borders: incorporating fielded text from neighboring web pages. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 643–650, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4.

Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, Wen-Ying Xiong, and Hang Li. Learning to rank relational objects and its application to web search. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 407–416, New York, NY, USA, 2008. ACM.

Raghu Ramakrishnan and Andrew Tomkins. Toward a peopleweb. *Computer*, 40(8):63–72, 2007. ISSN 0018-9162.

Stephen Robertson. A new interpretation of average precision. In *SIGIR*, pages 689–690. ACM, 2008. ISBN 978-1-60558-164-4.

Carl Sable, Kathleen McKeown, and Kenneth W. Church. Nlp found helpful (at least for one text categorization task). In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 172–179, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975. ISSN 0001-0782.

Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

Sam Scott and Stan Matwin. Feature engineering for text classification. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 379–388, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002. ISSN 0360-0300.

D. Silver, G. Bakir, K. Bennett, R. Caruana, M. Pontil, S. Russell, and P. Tadepalli, editors. *Workshop on Inductive Transfer: 10Years Later*, 2005.

Mark Steyvers and Tom Griffiths. *Probabilistic Topic Models*. Lawrence Erlbaum Associates, 2007. ISBN 1410615340.

Valdimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlog, New York, NY, USA, 1995.

Ellen M. Voorhees. Overview of the trec 2001 question answering track. In *In Proceedings of the Tenth Text REtrieval Conference (TREC*, pages 42–51, 2001.

Pu Wang and Carlotta Domeniconi. Building semantic kernels for text classification using wikipedia. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–721, New York, NY, USA, 2008. ACM.

Pu Wang, Jian Hu, Hua-Jun Zeng, Lijun Chen, and Zheng Chen. Improving text classification by using encyclopedia knowledge. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 332–341, Washington, DC, USA, 2007. IEEE Computer Society.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

Fei Wu and Daniel S Weld. Automatically semantifying Wikipedia. In *CIKM*, 2007. URL `http://turing.cs.washington.edu/papers/cikm07.pdf`.

Minji Wu and Amelie Marian. Corroborating answers from multiple web sources. In *10th Internal Workshop on Web and Databases (WebDB)*, 2007.

Pengcheng Wu and Thomas G. Dietterich. Improving svm accuracy by training on auxiliary data sources. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 110, New York, NY, USA, 2004. ACM.

Zenglin Xu, Irwin King, and Michael R. Lyu. Web page classification with heterogeneous data fusion. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1171–1172, New York, NY, USA, 2007. ACM.

Gui-Rong Xue, Dikan Xing, Qiang Yang, and Yong Yu. Deep classification in large-scale text hierarchies. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 619–626, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4.

Yusuke Yanbe, Adam Jatowt, Satoshi Nakamura, and Katsumi Tanaka. Can social bookmarking enhance search in the web? In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 107–116, New York, NY, USA, 2007. ACM.

Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49, New York, NY, USA, 1999. ACM. ISBN 1-58113-096-1.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, 2007.

Ying Zhang and Phil Vines. Using the web for automated translation extraction in cross-language information retrieval. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 162–169, New York, NY, USA, 2004. ACM. ISBN 1-58113-881-4.

Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. Technical report, Department of Computer Science, University of Minnesota, 200.

Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, 2005.

URL References

UrlCluto. Cluto - family of data clustering software tools — karypis lab. http://www.cs.umn.edu/~cluto.

UrlDelicious. Delicious. http://delicious.com.

UrlDmoz. Odp - open directory project. http://www.dmoz.org.

UrlDomainCounts. Domain counts & internet statistics. http://www.domaintools.com/internet-statistics/.

UrlGoogle. Google. http://www.google.com.

UrlGoogleNews. Google news. http://news.google.com.

UrlGoogleProduct. Google product search. http://www.google.com/products.

UrlLetor. Letor: Benchmark dataset for learning to rank. http://research.microsoft.com/en-us/um/beijing/projects/letor/index.html.

UrlLucene. Apache lucene - overview. http://lucene.apache.org/java/docs/index.html.

UrlSenseclusters. Ted pedersen - senseclusters. http://senseclusters.sourceforge.net/.

UrlSizeOfWeb. The size of the world wide web. http://www.worldwidewebsize.com/.

UrlSizeOfWikipedia. Wikipedia:size of wikipedia. http://en.wikipedia.org/wiki/Size_of_Wikipedia.

UrlWikipedia. Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Main_Page`.

UrlWikiStat. Statistics - wikipedia. `http://en.wikipedia.org/wiki/Special:Statistics`.

UrlYahooDirectory. Yahoo! directory. `http://dir.yahoo.com`.

Appendix A

List of Quantity Queries Used in the Experiments

The testbed of QCQ experiments in the Chapter 3 contains 162 queries. Each query has a query string that contains the keywords and phrases of the query and also the desired unit type of the answer quantities. The query string was used to search the index (or sent to the search engine). For the purpose of evaluation, the QCQs in the testbed also contain ground truth quantities. The QCQs were collected in a XML file. The format of the XML file and the list of queries collected are shown below.

```
<iitb.QuantitySearch.Query>
  <queryID>A unique string id of the query</queryID>
  <numericID>A unique numeric id of the query</numericID>
  <queryString>Query string that was used to search the index</queryString
  <descriptionString>Detailed description of the query string.
     This was not used in the processing</descriptionString>
  <standardUnitName>The desired unit type of the answer quantity
  </standardUnitName>
  <answerSet>
    <!-- List of ground truth quantities -->
    <string>Ground truth quantity</string>
  </answerSet>
</iitb.QuantitySearch.Query>
```

Figure A.1: The XML format in which QCQs are collected

A sample set of queries used in the experiments are shown below. The full list of 162 queries are available in the website `http://soumen.cse.iitb.ac.in/doc/QCQ/`

```
<object-stream>
```

```
<iitb.QuantitySearch.Query>
  <queryID>Samik-1</queryID>
  <numericID>1</numericID>
  <queryString>What is the age of the universe</queryString>
  <descriptionString>What is the age of the universe</descriptionString>
  <standardUnitName>year</standardUnitName>
  <answerSet>
    <string>13000000000-15000000000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Samik-4</queryID>
  <numericID>2</numericID>
  <queryString>+"half life" of plutonium</queryString>
  <descriptionString>What is the "half life" of plutonium</descriptionString>
  <standardUnitName>year</standardUnitName>
  <answerSet>
    <string>86-88</string>
    <string>24000-25100</string>
    <string>80800000-82000000</string>
    <string>13-15</string>
    <string>373300-380000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Samik-7</queryID>
  <numericID>3</numericID>
  <queryString>+height of a +giraffe?</queryString>
  <descriptionString>What is the height of a giraffe?</descriptionString>
  <standardUnitName>foot</standardUnitName>
  <answerSet>
    <string>16-18</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Samik-8</queryID>
  <numericID>4</numericID>
  <queryString>weight of a +A380 +airbus?</queryString>
  <descriptionString>What is the weight of a A380 airbus?</descriptionString>
  <standardUnitName>lb</standardUnitName>
  <answerSet>
    <string>1200000-1300000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Samik-12</queryID>
  <numericID>7</numericID>
  <queryString>+slant of "Leaning tower of Pisa"?</queryString>
  <descriptionString>What is the +slant of "Leaning tower of Pisa"?</descriptionSt
  <standardUnitName>degree</standardUnitName>
  <answerSet>
    <string>3.97-4</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Anjana1</queryID>
  <numericID>8</numericID>
  <queryString>+revenue of +Microsoft in 2004?</queryString>
```

```
    <descriptionString>What was the revenue of Microsoft in 2004?</descriptionString
    <standardUnitName>USD</standardUnitName>
    <answerSet>
      <string>9000000000-10000000000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Anjana3</queryID>
    <numericID>10</numericID>
    <queryString>+distance between +moon and +earth?</queryString>
    <descriptionString>What is the distance between moon and earth?</descriptionStri
    <standardUnitName>kilometer</standardUnitName>
    <answerSet>
      <string>356410-409000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Anjana4</queryID>
    <numericID>11</numericID>
    <queryString>+mileage of +Bajaj +scooty pep?</queryString>
    <descriptionString>What is the mileage of Bajaj scooty pep?</descriptionString>
    <standardUnitName>kilometer</standardUnitName>
    <answerSet>
      <string>50-55</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Anjana5</queryID>
    <numericID>12</numericID>
    <queryString>+printed +price of +canon +powershot 700IS?</queryString>
    <descriptionString>What is the printed price of canon powershot 700IS?
    </descriptionString>
    <standardUnitName>USD</standardUnitName>
    <answerSet>
      <string>477-499</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Anjana7</queryID>
    <numericID>13</numericID>
    <queryString>average +lifespan of a +chicken?</queryString>
    <descriptionString>What is average lifespan of a chicken?</descriptionString>
    <standardUnitName>year</standardUnitName>
    <answerSet>
      <string>8-13</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Anjana8</queryID>
    <numericID>14</numericID>
    <queryString>maximum +speed of a +"Mercedes Benz"?</queryString>
    <descriptionString>What is the maximum speed of a "Mercedes Benz"?</descriptionSt
    <standardUnitName>KilometersPerHour</standardUnitName>
    <answerSet>
      <string>315-340</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
  <queryID>TREC-2004-7.2</queryID>
  <numericID>19</numericID>
  <queryString>What is the average life span of an +agouti?</queryString>
  <descriptionString>What is the average life span of an +agouti?</descriptionStri
  <standardUnitName>year</standardUnitName>
  <answerSet>
    <string>12-20</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>TREC-2004-12.3</queryID>
  <numericID>20</numericID>
  <queryString>What is the +annual +revenue of +"Rohm and Haas"?</queryString>
  <descriptionString>What is the +annual +revenue of +"Rohm and Haas"?
  </descriptionString>
  <standardUnitName>USD</standardUnitName>
  <answerSet>
    <string>6500000000-8200000000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>TREC-2004-20.4</queryID>
  <numericID>25</numericID>
  <queryString>How fast does the +Concorde fly?</queryString>
  <descriptionString>How fast does the +Concorde fly?</descriptionString>
  <standardUnitName>MilesPerHour</standardUnitName>
  <answerSet>
    <string>1330-1350</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>TREC-2004-21.1</queryID>
  <numericID>27</numericID>
  <queryString>How many +"Club Med" +vacation +spots are there worldwide?</querySt
  <descriptionString>How many +"Club Med" +vacation +spots are there worldwide?
  </descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>80</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Minji-5</queryID>
  <numericID>40</numericID>
  <queryString>+"mini marathon" distance</queryString>
  <descriptionString>+"mini marathon" distance</descriptionString>
  <standardUnitName>mile</standardUnitName>
  <answerSet>
    <string>10-13.1</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>Minji-7</queryID>
  <numericID>42</numericID>
  <queryString>+TV viewing +distance</queryString>
  <descriptionString>+TV viewing +distance</descriptionString>
  <standardUnitName>foot</standardUnitName>
  <answerSet>
```

```
      <string>10</string>
      <string>7</string>
      <string>5</string>
      <string>14</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Live-HowFar-3</queryID>
    <numericID>46</numericID>
    <queryString>How far should +raccoons be +relocated miles</queryString>
    <descriptionString>How far should +raccoons be +relocated miles</descriptionStri
    <standardUnitName>mile</standardUnitName>
    <answerSet>
      <string>10-20</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Live-HowFast-1</queryID>
    <numericID>47</numericID>
    <queryString>how fast can +otters swim speed mph</queryString>
    <descriptionString>how fast can +otters swim speed mph</descriptionString>
    <standardUnitName>MilesPerHour</standardUnitName>
    <answerSet>
      <string>5.5-9</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Live-Frequency-1</queryID>
    <numericID>50</numericID>
    <queryString>+cordless +phone +frequency mhz</queryString>
    <descriptionString>+cordless +phone +frequency mhz</descriptionString>
    <standardUnitName>MHz</standardUnitName>
    <answerSet>
      <string>49</string>
      <string>900</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>Soumen-3</queryID>
    <numericID>54</numericID>
    <queryString>What is the distance between +Paris and +Rome</queryString>
    <descriptionString>What is the distance between +Paris and +Rome</descriptionStr
    <standardUnitName>kilometer</standardUnitName>
    <answerSet>
      <string>1106</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>RU-1</queryID>
    <numericID>55</numericID>
    <queryString>What is the average annual rainfall of Mumbai</queryString>
    <descriptionString>What is the average annual rainfall of Mumbai</descriptionStr
    <standardUnitName>millimeter</standardUnitName>
    <answerSet>
      <string>1800-2200</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
```

```
    <queryID>WikiInfobox-2</queryID>
    <numericID>63</numericID>
    <queryString>Hewlett-Packard employees</queryString>
    <descriptionString>Hewlett-Packard employees</descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>350000-450000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>WikiInfobox-7</queryID>
    <numericID>68</numericID>
    <queryString>Microsoft net income</queryString>
    <descriptionString>Microsoft net income</descriptionString>
    <standardUnitName>USD</standardUnitName>
    <answerSet>
      <string>16000000000-17500000000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>WikiInfobox-18</queryID>
    <numericID>77</numericID>
    <queryString>Mumbai population</queryString>
    <descriptionString>Mumbai population</descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>12000000-13500000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:221.5</queryID>
    <numericID>202</numericID>
    <queryString>+dimes +mint month monthly produce</queryString>
    <descriptionString>How many dimes does the Mint produce each month? U.S. Mint
    </descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>2000000000-2500000000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:234.2</queryID>
    <numericID>206</numericID>
    <queryString>number +songs +"irving berlin" composed wrote</queryString>
    <descriptionString>How many songs did Irving Berlin compose? Irving Berlin
    </descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>800-1500</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:242.5</queryID>
    <numericID>214</numericID>
    <queryString>+"percent" of alcohol by volume in Guinness Brewery Stout</queryStr
    <descriptionString>What is the percent of alcohol by volume in Guinness Stout?
    Guinness Brewery</descriptionString>
    <standardUnitName>Percent</standardUnitName>
    <answerSet>
      <string>3.8-4.3</string>
    </answerSet>
  </iitb.QuantitySearch.Query>

    <numericID>241</numericID>
    <queryString>+"rockets" +"fired" at Israel from Gaza after Israel evacuation of t
    Gaza Strip</queryString>
    <descriptionString>In the three months following the evacuation, how many rockets
    were fired at Israel from Gaza? Israel evacuation of the Gaza Strip
    </descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>230-240</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:269.3</queryID>
    <numericID>242</numericID>
    <queryString>final death toll from this earthquake? Pakistan earthquakes of Octob
    2005</queryString>
    <descriptionString>What was the final death toll from this earthquake? Pakistan
    earthquakes of October 2005</descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>70000-85000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:270.4</queryID>
    <numericID>243</numericID>
    <queryString>+Opportunity +"traveled" on +Mars? The Mars rovers, Spirit and
    Opportunity distance meters</queryString>
    <descriptionString>How many meters has Opportunity traveled on Mars? The Mars
    rovers, Spirit and Opportunity distance</descriptionString>
    <standardUnitName>meter</standardUnitName>
    <answerSet>
      <string>600-650</string>
      <string>13000-14000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:273.1</queryID>
    <numericID>244</numericID>
    <queryString>Limbaugh's listening audience? Rush Limbaugh</queryString>
    <descriptionString>Number of audience Rush Limbaugh</descriptionString>
    <standardUnitName>count</standardUnitName>
    <answerSet>
      <string>13000000-20000000</string>
    </answerSet>
  </iitb.QuantitySearch.Query>
  <iitb.QuantitySearch.Query>
    <queryID>QA07:273.2</queryID>
    <numericID>245</numericID>
    <queryString>Limbaugh's radio show carry +"markets" Rush Limbaugh</queryString>
    <descriptionString>How many markets carry Limbaugh's radio show? Rush Limbaugh
    </descriptionString>
    <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>600</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:273.4</queryID>
  <numericID>246</numericID>
  <queryString>Rush Limbaugh +"married" times</queryString>
  <descriptionString>How many times has Limbaugh been married? Rush Limbaugh
  </descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>3</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:274.5</queryID>
  <numericID>247</numericID>
  <queryString>oil was spilled by the Exxon tanker Valdez? Exxon Mobil Corp
  </queryString>
  <descriptionString>How much oil was spilled by the Exxon tanker Valdez? Exxon Mc
  Corp</descriptionString>
  <standardUnitName>Gallon</standardUnitName>
  <answerSet>
    <string>10000000-11000000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:274.6</queryID>
  <numericID>248</numericID>
  <queryString>cost of the oil clean-up? Exxon Mobil Corp</queryString>
  <descriptionString>What was the cost of the oil clean-up? Exxon Mobil Corp
  </descriptionString>
  <standardUnitName>USD</standardUnitName>
  <answerSet>
    <string>3000000000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:276.3</queryID>
  <numericID>249</numericID>
  <queryString>B-17 bomber built during World War II</queryString>
  <descriptionString>How many B-17's were built? B-17 bomber during Worl War II
  </descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>12500-13000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:276.5</queryID>
  <numericID>250</numericID>
  <queryString>B-17 bomber the Memphis Belle fly missions</queryString>
  <descriptionString>How many missions did the Memphis Belle fly? B-17 bomber
  </descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>25</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:280.6</queryID>
  <numericID>260</numericID>
  <queryString>+people visit Angkor Wat annually</queryString>
  <descriptionString>How many people visit Angkor Wat annually? Angkor Wat temples
  </descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>1000000-2000000</string>
  </answerSet>
</iitb.QuantitySearch.Query>
<iitb.QuantitySearch.Query>
  <queryID>QA07:282.6</queryID>
  <numericID>261</numericID>
  <queryString>Pamuk's +"translated" into +"languages"</queryString>
  <descriptionString>Into how many languages have Pamuk's works been translated?
  Orhan Pamuk</descriptionString>
  <standardUnitName>count</standardUnitName>
  <answerSet>
    <string>20-40</string>
  </answerSet>
</iitb.QuantitySearch.Query>
</object-stream>
```

List of Publications and Presentations

[1] Somnath Banerjee, Soumen Chakrabarti and Ganesh Ramakrihsnan, "Learning to Rank for Quantity Consensus Query", 32nd Annual ACM SIGIR International Conference , 2009, Boston, USA

[2] Somnath Banerjee, Soumen Chakrabarti, Sayali Kulkarni, Ganesh Ramakrhishnan, Amit Singh, "Cacophony to Consensus: Bulding Bridges between the Unstrcutured Web and Structured Knowledge", Technology Review, Indian Edition, August 2009.

[3] Somnath Banerjee, Avinava Dubey, Jinesh Machchhar and Soumen Chakrabarti, "Efficient and Accurate Local Learning for Ranking", In the SIGIR 2009 Workshop of Learning to Rank for Information Retrieval, 2009, Boston, USA

[4] Amit Singh, Sayali Kulkarni, Somnath Banerjee, Ganesh Ramakrishnan, Soumen Chakrabarti, "Curating and Searching the Annotated Web", In the 15th ACM SIGKDD Conference, 2009, Paris, France (System Demonstration)

[5] Krishnan Ramanathan, Somnath Banerjee and Geetha Manjunathan, "Personalization Techniques and Applications", ACM Compute 2008, Bangalore (Tutorial Presentation)

[6] Somnath Banerjee and Martin Scholz, "Leveraging Web 2.0 Sources for Web Content Classification", IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008, Sydney, pp.300-306

[7] Somnath Banerjee, "Improving text classification accuracy using topic modeling over an additional corpus", In the Proceedings of the 31st Annual ACM SIGIR International Conference , 2008, Singapore, pp. 867-868

[8] Somnath Banerjee and Krishnan Ramanathan, "Collaborative Filtering on Skewed Datasets", In the Proceeding of the 17th international conference on World Wide Web, 2008, Beijing, China, pp. 1135-1136

[9] Somnath Banerjee, "Boosting Inductive Transfer for Text Classification Using Wikipedia", In the Proceedings of the 6th International Conference on Machine Learning and Applications, 2007, Cincinnati, USA pp. 148-153

[10] Somnath Banerjee, Krishnan Ramanathan and Ajay Gupta, "Clustering Short Texts using Wikipedia", In the Proceedings of the 30th Annual ACM SIGIR International Conference , 2007, Amsterdam, pp. 787-788

Brief Biography of the Candidate

Somnath Banerjee did his B.E. in Computer Science and Engineering from Jadavpur University, Calcutta and Masters from IIT Kharagpur. He has worked in Microsoft India Development Center, Hyderabad for two years. In 2005, he joined the HP Labs - BITS Pilani PhD Fellowship program. He has spent 9 months at IIT Bombay working with Professor Soumen Chakrabarti and 2 months in HP Labs, Palo Alto working with Dr. Lukose Rajan and Dr. Martin Scholz. His research interests are in machine learning and information retrieval.

Brief Biography of the Supervisor

Dr. Krishnan Ramanathan is a Senior Research Scientist at HP Labs India. He is working on the Simplifying Web Access for the Next billion (SWAN) project. He joined HP labs in January 2005. He joined Hewlett Packard in July 2000 as a software architect with HP India Software Development organization where he worked on the VirtualVault and OpenView Performance products. Prior to joining HP, he was a research scientist with the Defence Research and Development organization(DRDO) India from 1990 to 2000. In DRDO, he worked on diverse topics including shape recognition, decision support systems, neural networks, OCR systems and security applications.

His research interests are in human computer information retrieval (HCIR), personalization and machine learning.

Krishnan has an M.Sc. in Computer Science from University of Poona, India and a Ph.D. in Computer Science (in Machine Learning) from the Indian Institute of Technology, Mumbai.