USING NAMED ENTITIES IN POST-CLICK NEWS RECOMMENDATION

by

Arash Koushkestani

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
June 2015

To my parents.

Table of Contents

List of Tables

List of Figures

Abstract

With the growth of online news readers, many news websites use different signals to attract users' initial clicks. However, the problem of keeping users in the web site through *post-click news recommendation* is relatively under explored. To address this problem, we try to find the news articles related to the one that a user is currently reading based on the content of the articles while no history or user profile is assumed. The problem is very similar to a typical information retrieval problem in which the system finds related documents to a given query ranked by a similarity function which produces a relatedness score between a document and the query. However, we conducted experiments to show that "relatedness" is not equivalent to similarity as in information retrieval. As a relatedness function, we used the semantic similarity of named entities extracted from the body of news articles in a combination with lexical similarity functions available through information retrieval systems. A new system called Tulip was used as the named entity recognition and disambiguation system and the word skip-gram model was used for finding similarity of named entities. Tulip provides precise recognition of named entities and very fast response time. Additionally, a stochastic keyword extraction algorithm based on the Chinese restaurant process and the word skip-gram model was proposed to capture topical similarity of two articles. To solve problem practically, we proposed using the cosine similarity of TF-IDF vectors of articles as a filter to narrow down the search space, given one article as a query. Then we applied the relatedness function to the results returned by cosine similarity. In other words, we proposed a relatedness function to re-rank the results extracted from a typical retrieval system. Due to the nature of the problem and available datasets, we proposed a graph based approach as an unsupervised approach for labeling pairs of documents during both training and testing. We trained and tested our method on two datasets against the cosine similarity of TF-IDF vectors as the baseline before testing it by domain experts. The model trained on our proposed features is demonstrated to outperform the baseline. Finally we conducted a series of experiments to rank the importance of different features. Based on our observations, semantic similarity of named entities along with Information Based lexical similarity (included in Lucene) are more effective than other lexical features and provide better ranking for the related news.

List of Abbreviations Used

ANOVA	Analysis of variance
CF-IDF	Concept frequency-inverse document frequency
CFS	Correlation based feature selection
CRP	Chinese Restaurant Process
DCG	Discounted cumulative gain
GBRT	Gradient boosted regression tree
IB	Information based
IR	Information retrieval
JVM	Java virtual machine
LDA	Latent Dirichlet allocation
LM	Language model
LSI	Latent semantic indexing
MAP	Mean average precision
MVC	Model, View, and Controller
NDCG	Normalized discounted cumulative gain
NERD	Named entity recognition and disambiguation
ORM	Object-relation mapping
PMI	Pointwise mutual information
RDBMS	Relational database management system
SF-IDF	Synset frequency-inverse document frequency
TF-IDF	Term frequency-inverse document frequency

Acknowledgements

First, I would like to thank my supervisor, Evangelos Milios. Thank you for taking me as a master student at the first place. Thank you for your guidance, patience, and the great project you have assigned to me. You taught me how to do research and organize my thought when I was lost in the middle of so many ideas and tasks.

I am also grateful to my mentor, Marek Lipczak. Thank you for teaching me a new way of thinking when working with big data problems. You had many great ideas for solving hard problems which I could never think of before meeting you. Thank you for trusting me and giving me different projects, I learned a lot. I can never forget the one year time that I have spent constantly learning from you.

I am fortunate to meet and work with wonderful people in The Chronicle Herald. Thank you Jason Hurst, Bryan Cave, and Sheryl Grant for all of your time and efforts. You trusted me, you spent time for me, and you provided me your invaluable data that I could never find anywhere else. Thank you Bryan and Jason to spend so much time preparing all the tests and interfaces required for evaluating the recommender. You always gave me positive energy. I must thank all of the other people in The Chronicle Herald for their cooperation in this project from the first place.

I would like to thank Dalhousie University to provide such a great environment to do research and allowing me to finish my work. Coming to Dalhousie was a great opportunity that has changed my life forever. It was my first time being away from home, and I am happy that beautiful Dalhousie University was my first destination far away from family.

I am proudly grateful to my supportive and loving family. To my mom and dad, I am blessed to be your child. To my sister, you always made me happy. Thank you all for encouraging me to pursue my interests and your unconditional love and support. I am grateful to my friends who never let me be alone. Thank you for letting me not thinking about my problems while I was with you.

Chapter 1

Introduction

Online news readership has experienced an explosive growth in the last decade. According to the Newspaper Association of America[1], 173 million unique visitors have visited digital newspaper web sites in only January 2015 which is 27 million more than the number of visitors in January 2014. This number only shows news papers and not many famous news websites such as Yahoo! News and Google News. The Halifax based news paper "The Chronicle Herald" has 1000 online users on average at a given moment on weekdays. All these numbers confirm the importance of recommending related news articles to users.

One obvious advantage of online news websites over traditional media is the augmentation of information such as links to encyclopedias such as Wikipedia, and adding voice or video clips to the news. One other advantage is the ability to show related news articles to the user. When she is done with one news article, she may also like to explore other related news articles to the current article on the screen. There are tools like Hermes[2] for personalizing news articles to solve this problem, but practically, a large fraction of users do not log in to news web sites and are referred to a news article from another domain. More than 30 percent of Facebook users get their daily news from the social media website [3] and are redirected to news websites from their personal account (their Facebook wall or Twitter timeline). In this situation, the user had been already in the website and the goal is to feed him or her with as much related news as possible to keep the user interested. We follow the work done by Lv et al. [2011] and call this problem post-click news recommendation.

There are two research areas related to post-click news recommender systems: the first is document pair similarity and the second is news personalization. In the document pair similarity area, the goal is to quantify the similarity of two documents

[1] http://www.naa.org/
[2] http://hermesportal.sourceforge.net/
[3] http://www.pewresearch.org/fact-tank/2014/09/24/how-social-media-is-reshaping-news/

Figure 1.1: An article from "The Chronicle Herald". The red box represents the recommended news to the selected article.

from different aspects (Huang et al. [2012]). If we look at the news articles as two text documents, then finding related news articles is finding other articles with the highest similarity score. The goal of the news personalization is to suggest news articles based on user's interest. This field has grabbed a lot of attention in recent years (Borsje et al. [2008]).

Effective post-click news recommendation is crucial to news web sites, because recommending unrelated news articles will hurt user interest in the web site. So recommended articles must be logically and topically related. They must represent one coherent story through time to keep the user interested. They must also be about the same person or entity.

In the best case scenario, news recommendation is offered by editors by going through the entire corpus and selecting related news before publishing a news article.

This task is very time consuming and is not always complete because editors do not remember all the related news articles from the past and one editor may not be an expert in all topics. Therefore it is highly desirable to have a recommender system for both editors and users: it can be seen as a decision support system for news editors and as a content-based recommender system for visitors of the web site.

The notion of relatedness between two news articles is difficult to define precisely. Two news articles can be related because they happened in the same geographical location, the main subject person is the same, or they follow one developing story. In contrast, researchers in information retrieval have defined different relevancy measures through the years to find relevant documents for a given query in information retrieval area. Each relevancy measure produces a slightly different result. In our initial experiments, we took one article and used its entire body as a query string. Then we passed the query to an IR system which contained all of the news article in the repository and examined the returned results. We discovered that although relevancy is not equal to relatedness, the produced results by different information retrieval systems can be seen as related news. Since each of them can capture different aspects of relatedness, we decided to combine them and use machine learning to learn the concept of relatedness using different relevancy scores.

Semantic similarity of named entities can also capture relatedness. A good semantic similarity measure can assign high similarity score to a pair of entities which are related to each other from different aspects, like geographic location or affiliation of people. Hence, we used semantic similarity of named entities in a combination with other lexical similarity measures in an effort to formulate relatedness.

A relatedness model will produce a score for a pair of news articles in a way that more related pairs get higher scores. A very important issue of a recommender system is its performance in terms of response time. Having a trained model of relatedness, it is not practical to apply it on all possible pairs of articles in the repository. All major search engines that implement typical information retrieval algorithms have reduced their response time over the last decade and that's one of the main reasons many of them are being used as content based recommended systems in different applications especially in news domain. We introduced a filtering idea using a simple cosine similarity to narrow down the search space while keeping the accuracy of the model

as high as possible. For a given query document, instead of checking the relatedness score with all available articles, we first pass the query article to a retrieval system and get a number of articles. The number of returned documents are much fewer than the available articles and we set it in a way to minimize the loss of recall as much as possible. After applying the filter, a relatedness function will score each pair of news articles consisting of query article and each of the returned articles. By narrowing down the search space and number of times we apply relatedness function, we can solve this problem practically.

1.1 Research Objectives

The main goal of this research is providing a model that has better performance than Lucene (baseline system) in content-based news recommendation. In order to achieve our goal, we performed following studies:

1. Using named entities and studying their effect in news recommendation.
2. Using word vectors to provide semantic relatedness between two news articles.
3. Using word vectors and Chinese restaurant process to provide topical similarity between two news articles.

Our proposed method must outperform baseline in both accuracy of recommendation and response time. To achieve this goal, we implemented a fast approach for recommending news by introducing lexical filtering and re-rank its results.

1.2 Contributions

Here is the list of contributions of this thesis:

1. Using Tulip as the state of the art system in named entity recognition and disambiguation for post-click news recommendation. We used skip-grams to calculate semantic similarity of named entities and we showed that features based on semantic similarity are more important than lexical similarities.
2. A new keyword extraction method which helps summarizing news articles and creating a keyword based profile. We first cluster the words of a news article

based on their semantic similarities by using a one-pass stochastic algorithm. Each cluster shows one of the topics of the text and the one that represents dominant topic of text is selected. The keywords of the selected cluster will be used to find similar news articles and we claim that using this features beside other lexical similarity features improves the accuracy of the model since it can capture topical similarity.

3. We adapted a graph representation of data to model relatedness scores by only having binary relatedness information, the only available information that can be extracted from datasets. The nodes represent news articles and each edge represents binary relatedness between two news article. We used the distance of nodes in the graph as the score of news article pairs. We claim that using inferred scores improves the performance of any regression model in this task as opposed to binary regression of classification. It also shed light to consider story of news articles instead of isolated pairs of news articles.

In addition to the main contributions, we argued that our proposed solution for news recommendation is practical. We proposed an architecture that instead of finding relatedness score for all news pairs, tries to re-rank the returned results from a TF-IDF search engine. This approach narrows down the search space required for the system to find related news articles.

Our relatedness function is a combination of different similarity measures. The idea of combining similarity measures is not novel. But some individual features are novel and are used for the first time in this domain including features that are based on Tulip and skip-gram model.

Finally we provided two publicly available datasets for news recommendation which can be later used by researchers to improve news recommender systems. The first one belongs to The Chronicle Herald newspaper and the second one is parsed from Wikinews. It is the first time we used a free-collaborative source of news like Wikinews for the post-click news recommendation task.

Chapter 2

Background and Related Work

In this chapter we review previous work in related fields. To the best of our knowledge there is a little work in the area of post-click news recommender systems. However there are many research studies performed in related areas such as personalized news recommendation using different methods. First two different paradigms of recommender systems are explained. As this thesis is a content based recommender system, some work in this area will be covered. After that news personalization systems which work based on user profiles will be briefly discussed. Then there is an overview of the under-explored area of post-click news recommender systems. Since our proposed method maps the problem to an information retrieval task, a brief overview in this field is also discussed. Additionally, we discuss the importance of named entities and similarity measures based on Wikipedia. A good semantic similarity measure for named entity pairs can capture semantic relatedness of two text documents better than lexical similarities. At the end, we explore learning to rank and their applications in recommendation systems.

2.1 Recommender Systems

The problem of recommending items to users has been studied quite extensively throughout the last decade and two main paradigms of problem solving have emerged: collaborative filtering and content-based filtering. In most of the studies, both entities of users and items exist. Users are usually registered into the website and their activity can be recorded to build an informative user profile Lops et al. [2011].

In collaborative filtering, the recommender identifies users whose preferences are similar to a given user and recommends items they have liked (Balabanović and Shoham [1997]). The preferences can be mined by looking at whether a user has bought an item or not. More advanced solutions analyze history of users' activity such as checking if a user listened to a music track, watched a movie, rated an item

or wrote a comment. In many cases such as online shopping or multimedia web sites in which contents are rarely available, common collaborative filtering algorithms outperform existing content-based methods by a relatively large margin.

In the content-based paradigm, the recommender tries to construct a user profile based on what she preferred previously. In this case, the recommender does not depend on what other users liked or disliked about a particular product and user profile will be compared to existing item profiles to find similar items. In this way, the recommendation problem can be seen as a personalization task in which users will find the type of the information they are mainly interested in. Users are usually allowed to provide some information about themselves to guide the system further. Content-based filtering does not suffer from the major drawback of collaborative filtering which is called the *cold-start* problem. Whenever a new item is added to the system there is no rating or preference available for that and it takes some time for the system to collect enough information for making useful inference. On the other hand, in content-based systems, recommending new items can be done by matching its content to the existing user profiles.

2.1.1 Personalized News Recommendation System

News are naturally not a suitable domain for collaborative filtering since new articles are being published every day, which introduces sparseness in the user-item matrix. Also, news article have relatively much shorter lifetime and they lose their importance after a very short time compared with items in an online shopping web site. So the most sensible choice for a recommender system is a content-based system. The key problem to solve is to create a representative profile for both articles and users.

TF-IDF is a basic way to represent documents in the vector space model. User profiles are built by aggregating TF-IDF scores of words in the articles Lops et al. [2011]. News article will later be compared to the created profile by cosine similarity and the most relevant ones will be reported. But lexical similarity of words is not informative enough for user or article profiles. Systems like Hermes Borsje et al. [2008] used WordNet to enrich the representation and provided a new value for each word called Concept Frequency - Inverse Document Frequency (CF-IDF) Goossen et al. [2011] in a vector space model. Other methods such as Synset Frequency -

Inverse Document Frequency (SF-IDF) Moerland et al. [2013] follows the same idea as well. Using ontologies was not a new idea in news recommendation since many news outlets have their own category hierarchy and provide similar news articles based on what category a user might be interested in. Named entities were also used for news personalization (Gabrilovich et al. [2004]). In these systems, however, semantic similarity among named entities was not considered.

2.1.2 Post-Click News Recommendation System

All of the mentioned systems rely on the fact that there exists a user profile and a descriptive model of users and items can be constructed. However, they do not take into consideration a major difference between news web sites and other web sites: many readers do not have any user account so building a profile of user preferences or her activity logs is impossible. News websites usually rely on item to item similarity to recommend similar news articles to what a user is reading at a given moment. In this situation, techniques in information retrieval are used to find relevant news articles. These techniques require an accurate definition for similarity or relatedness in order to produce acceptable results.

The main motivation is to keep the user in the news web site as long as possible by offering interesting news articles after she enters the web site. This area is called post-click news recommendation Lv et al. [2011] and is fairly under-explored. In the state of the art, post-click recommendation is done by editors manually searching through the entire corpus. The process in both expensive and limited to the editors' knowledge about the subject.

The same problem was addressed by Bogers and van den Bosch [2007] through combining three different relevancy measures, and created a regression model out of their proposed combinations. They also demonstrated that using the entire body of news articles can improve the performance of their algorithms as opposed to using a fraction of it like using first paragraphs or selected passages. The only comparable work that differentiated between relevancy and relatedness in the news domain was done by Lv et al. [2011]. They followed the idea of combining different relevancy scores, however they argued that relevancy scores do not represent relatedness between two news articles. They proposed using a modified topic model method

(LDA) to capture higher level similarity between two articles helps recommending news articles which may be very related topically but have few words in common. They also demonstrated that relevancy scores are among the most important features. They used learning to rank for this task and used ranking measures to evaluate their proposed method.

Other work use modified information retrieval methods which often rank articles by calculating relevancy score. Due to performance constraint of news recommendation, many methods will not be suitable for this task. Because the entire process of finding related news articles must be finished when the user finishes reading one news article. Lexical information retrieval methods are usually fast enough and are easily fit into the requirements of this domain. An example of an alternative method is document clustering which its output can be used to find similar documents to a query article. However the performance of clustering algorithms is worse than lexical IR methods.

Trevisiol et al. [2014] studied post-click recommendation of news articles in a completely different paradigm. They studied the behavior of users in different news web sites and constructed a graph which indicated how users switched their attention between news articles. Then they made their probabilistic graph-based model to guess which news article user will read after reading one article.

Capelle et al. [2013] used semantic similarity of WordNet concepts in their news recommendation task. Although they used their method in a personalization task (as opposed to our post-click problem task) it was one of the first methods to use named entities and their similarities. They calculated similarities between concepts using Bing search engine by finding number of web pages that a concept is appeared in. They used these statistics to calculate PMI co-occurrence similarity measure.

In this thesis, the problem of post-click news recommendation is addressed by introducing new document pair similarity measures based on semantic relatedness of named entities. We followed the previous work that combined different similarity measures to capture relatedness by introducing semantic similarity of named entities. We used a state of the art named entity recognition and disambiguation system to extract entities and then used the method of skip-gram to calculate similarity of two named entities.

In most systems, datasets are created manually for the specific task. For example authors usually collect some news articles and ask people to rank them for their relatedness or relevance to use them in ranking or regression models. In our work, we created a graph of relatedness between news articles using explicit feedback from news editors. The nodes represent news articles and each edge represent binary relatedness between two news article. We used the distance between nodes to calculate their relatedness. This approach can be used for any news website by simply extracting binary relationship among news articles. We later show that using this approach reduces learning noise and improves our results for a good extent. We used this approach for parsing Wikinews[1] as a collaborative news outlet. In fact Wikinews was used for the first time in this thesis and can be a benchmark for comparing different recommendation systems in the future.

2.2 Information Retrieval

Information retrieval is the activity of obtaining information resources relevant to an information need from a collection of data. The input is typically a textual query and the output is the ranked list of relevant document. There is always and scoring strategy which plays an important role in information retrieval as it defines which document is relevant to the query. Returned documents are usually sorted based on their relevancy scores. This area has been studied extensively and many successful statistical methods and tools have been proposed. TF-IDF is the most famous method. Okapi BM25, probabilistic language models, and information based retrieval models are also available. Casting the news article recommender problem into an information retrieval problem can be done easily by considering the content of one document as query and pass it to an IR system. The returned results can be seen as relevant documents and are sorted by their relevancy scores.

2.2.1 Semantic Search

Statistical information retrieval methods only consider lexical relevancy and do not take into account the meaning of the words. Although their performance is acceptable in many situations, in some tasks which require deeper understanding of the

[1] http://en.wikinews.org

underlying topic, they fail to provide accurate results. There are some works that use an ontology specifically WordNet to calculate semantic similarity of text documents (Leacock and Chodorow [1998]). However the performance overhead of non-statistical methods is usually high and does not scale well.

LSI (Landauer et al. [1998]) is one of the techniques in IR which is believed to capture semantic relatedness between queries and documents. It is an indexing and retrieval method that uses a mathematical technique called singular value decomposition to identify the relationship between the terms and the concepts contained in an unstructured collection of text. LSI is based on the fact that words within a context tend to have similar meanings. It has the ability of correlating semantically related terms in the text.

LDA (Blei et al. [2003]) is a topic modeling algorithm that can be used in information retrieval to provide semantic analysis of words. Similar to LSI, it can identify sets of words that are semantically related. It also produces a topic model for documents by assigning topic weights to its containing words. In other words, the output of LDA can be vectors of topics for a given document. One typical way to measure similarity of two documents based on LDA is to calculate the Kullback-Leibler divergence of their topic distribution. An alternative approach is calculating cosine similarity of two topic distributions and use them as two vectors in topic space.

Explicit Semantic Analysis (Gabrilovich and Markovitch [2007]) is a vectorial representation of text that uses a document corpus as a knowledge base. Each text can be represented in a high dimensional space of concepts derived from a knowledge base like Wikipedia. The similarity between two texts will be calculated using conventional methods such as cosine similarity of their corresponding vectors in the new vector space. ESA is a proven method in text categorization. The provided similarity measure can be directly used in information retrieval tasks to perform semantic search.

2.2.2 Named Entity Recognition and Disambiguation

A correct mapping from words to concepts in an external knowledge base, automatically solves the problem of synonyms in information retrieval. Beside that, many valuable relationships between concepts in a knowledge base can help with semantic

search and identifying underlying topics. For example in FreeBase[2] it is easy to infer that Halifax is a Canadian city in the Atlantic coast line and is the capital city of the province of Nova Scotia. Such valuable information can lead to a better inference in similarity.

Semantic similarity between two concepts in knowledge bases received a lot of attention since the emergence of reliable and very big ones like Wikipedia Medelyan et al. [2008] Witten and Milne [2008]. There are also works like DBPedia Spotlight[3] which follows a context-based scoring strategy. Recently, Mikolov et al. [2013] introduced Skip-grams which are currently state-of-the-art in semantic similarity between any given pair of words or consequently any concepts. We used skip-gram model extensively in this thesis.

In order to use semantic similarities of named entities in semantic information retrieval tasks, named entities must be recognized and disambiguated from texts. The first task is to identify named entities within the text. In other word, all the words that can possibly be a named entity in the knowledge base must be identified. After that the disambiguation task selects the correct meaning for each word from a given knowledge base. As mentioned before, the knowledge base is usually Wikipedia. The task of named entity recognition and disambiguation has been studied broadly in recent years due to their importance. There are a few effective methods such as WikiMiner (Milne and Witten [2008]), Wikify (Mihalcea and Csomai [2007]), DBPedia Spotlight (Mendes et al. [2011]), AIDA (Hoffart et al. [2011]), and a novel fast and accurate method called Tulip (Lipczak et al. [2014]). Tulip is in fact the state-of-the-art system in this area based on the results in Named Entity Recognition and Disambiguation Challenge 2014.

The most recent work to use semantic similarity of concepts in document pair similarity belongs to Huang et al. [2012] who used Wikiminer for named entity recognition and disambiguation and used different similarity features for document pairs such as average, minimum, and maximum similarity of concept pairs in two text documents. Then a regression model was trained and the output of the model was used as similarity of two documents.

To the best of our knowledge, there is no such a system to use state-of-the-art

[2]http://www.freebase.com/
[3]spotlight.dbpedia.org/

methods in named entity recognition and disambiguation along with an accurate similarity measure. In this work, Tulip was used for named entity recognition and disambiguation (NERD) task and skip-grams were used to provide semantic similarity among identified named entities for the first time. Extracted named entities and all features based on them are among the most important features.

2.2.3 Learning to Rank

Learning to rank refers to machine learning techniques for training the model in a ranking task. Learning to rank is useful for many applications in information retrieval, natural language processing, and data mining (Li [2011]). In this work, we cast the problem of finding related news into a ranking problem and measured its performance against traditional regression models.

The input of a learning to rank algorithm is lists of items with some partial ordering between items within each list. The output would be a model, which takes a query and a document and returns either a score or a binary decision. The score represents relevancy in information retrieval, but based on our goal and our features, we can see the scores as relatedness degrees. Relevancy score denotes how well returned documents meets the information need of the user which is expressed through a query string. However relatedness score represents similarity of two documents, which in this thesis are two news articles.

Although this is a fairly new area, there are quite a lot of works in this domain. The goal of many of the works is introducing a new method for learning to rank such as MART(Friedman [2000]), RankNet (Burges et al. [2005]), RankBoost (Freund et al. [2003]), SVMRank (Joachims [2002]), and LambdaMart (Wu et al. [2010]). There are also many works on applications of learning to rank in other domains than their original information retrieval tasks such as construction of email threads (Dehghani et al. [2013]) and recommender systems. Since recommender systems outputs are always a ranked list of relevant results, learning to rank has been used in recommender systems for a while. In post-click news recommender system, only Lv et al. [2011] used learning to rank and they did not offer any comparison with traditional regression models. In contrast, we are going to compare learning to rank methods with regression model to justify using one method over others.

2.3 Keyword Extraction

Keyword extraction is a well-studied area of natural language processing. The main goal is to extract meaningful words and phrases from text files. There are many motivations behind developing such systems including text summarization, document tagging, and compact representation of documents. Methods are either supervised or unsupervised. Unsupervised methods are usually based on statistical information and co-occurrence of the words in text while supervised methods usually require a sufficient number of text documents with keywords extracted by experts (Medelyan et al. [2009]).

In this work we present a new method for keyword extraction based on the semantic relatedness of words in skip-gram model and the Chinese restaurant process. Our extracted keywords are used as one of the features to capture relatedness and works alongside other features to capture new aspects of relatedness.

Chapter 3

Features, Training Set, and Relatedness Model

In this work, we are trying to answer two research questions:

1. Is news article relatedness the same as similarity in information retrieval?
2. If they are different, can we define a relatedness function?

First we performed some experiments with typical information retrieval tools to answer our first question. Then we focus on introducing a new relatedness function for news. There is not a clear definition for relatedness of two news articles. Therefore, we follow the works by Lv et al. [2011] to learn an unknown concept of relatedness by combining different similarity measures. In our work, each similarity measure provides a score which is used as a numeric feature. So all of the definitions of features are for a pair of documents.

We model relatedness of a pair of documents as a regression problem, in which a set of similarity measures is treated as a set of input features to a regression model, which maps them to a single similarity value. The main challenges here are selecting input similarity measures and finding a good regression model. The **relatedness model** that we are defining is similar to the similarity functions in information retrieval tasks in which a function scores the similarity of a query to a given document. Hence, we follow that terminology and call a given news article **query document**. For each query document, the trained model will be capable of giving higher scores to **related documents**.

In this chapter, we begin with answering our first research question by doing a simple experiment. Then we explain different features that are used in order to create a regression model which will finally become our relatedness model. We will begin describing a keyword extraction method which will help us define our first feature in section 3.3. Then we focus on the features that captures semantic similarity of two news articles in section 3.4. After that we explain a number of lexical features that

K	Precision @ K	Recall @ K
3	0.339	0.453
5	0.225	0.563
15	0.089	0.727
25	0.057	0.785
35	0.042	0.816
45	0.033	0.831

Table 3.1: Performance of Lucene search engine with default TF-IDF scoring function considering top k returned results.

were used in the regression model. In section 3.6 we discuss different approaches to use available datasets to train the relatedness model. Finally, we explain different regression models that we used to model relatedness.

3.1 Similarity as Relatedness

Our first research question is knowing if similarity measures that are used in information retrieval capture relatedness in news articles. To answer this question, we used the common information retrieval tool called Lucene[1]. This tool acts like a document database and is extensively used in many software products or websites that have searching service. Given its popularity and efficiency, we used it as our baseline. Lucene also provides very fast and reliable indexing and retrieval APIs in various programming languages.

We indexed the body of the news articles in "The Chronicle Herald" dataset by Lucene using its standard analyzer which removes very common stop words and makes all the text lower-cased. After that we extracted a subset of news articles that had at least one related news article based on what news editors marked as related. For each of those articles, we created a query using its body and passed it to Lucene (more details about this dataset is in section 5.1). We measured precision and recall of retrieved documents by considering different number of returned articles as shown in table 3.1. The returned articles for each query article are matched with the ground truth to calculate precision and recall.

This experiment gives us an estimation of how well an information retrieval system performs in finding related news articles. In this experiment, the similarity function

[1]https://lucene.apache.org/core/

that we used was cosine similarity of TF-IDF vectors of documents which is widely used in IR tasks. According to results of this experiment, although similarity is not equal to relatedness in news domain, lexical similarity is one of its important aspects. In the following sections, we define additional features to capture other aspects of relatedness.

This experiment also shed light to an important design decision regarding implementation of recommender system. The final recommender system must be responsive enough to handle all concurrent users who are visiting the news website. If we assume that we have a relatedness function which produces a score for two given news articles, a trivial idea to use that for finding related articles would be comparing a given news article to all others. This idea is definitely impractical considering the number of news articles in a news website. Instead, if we look at the results produced with minimum overhead by Lucene, we can use it as a filter before our model. In other words, we will use Lucene retrieved documents to narrow down our search space. Based on our results, if we get 45 results, then we only need 45 comparisons with the cost of losing 17 percent of recall. In fact considering only 45 results means that 17 percent of pairs are related without having exact match of words which is still acceptable considering the performance gain of the system. We also used the subset that was prepared for this experiment for our other experiments including training and testing of the models. We separated this subset to a train and test set with size of 0.75 and 0.25 of the original set respectively.

In the rest of this thesis, when we talk about a relatedness model, we mean a model which will be applied to the Lucene's returned results. In other words, the relatedness model will be used to re-rank the retrieved results based on other similarity measures.

3.2 Model of News Articles and Labels

In our model, without loss of generality, each input news article must contain a title, a news body, and a publish date. There is also an optional field which represents editors' choices of related news articles. This optional field is used to train our system, but is not used in production environment. This is a minimalistic view to a news article and enables us to define various similarity measures suitable for news articles which can be used in many news websites based on similarity of the news body, title, and

date. We will later define all of our features based on these properties.

The training labels are provided for a fraction of news article pairs by human judgment. The labels indicates if a pair of news articles are related, but does not provide any information about whether a pair is unrelated. In "The Chronicle Herald" data set, the labels are set by editors while in Wikinews data set, everybody can provide such labels for article pairs. These labels can be used in various ways as it will be discussed later. In both of our datasets, almost one percent of news articles have a related news section. The data related for this section comes from editors of the news. So we look at them as ground truth for modeling relatedness. Although this fraction of data is small compared with entire dataset, they are invaluable since the editors are the experts of this domain. However, the problem is that they cannot relate all the news articles to each other; they can remember a limited number of news articles at a moment, and more importantly, they have a limited time which prevents them from going through their entire news corpus to extract all related articles. As a result we will be sure about the label of a pair of news articles if they are labeled as related. But it is wrong to assume that unlabeled news pairs are unrelated.

3.3 Keyword Extraction

In this section we explain how we obtain one of our features for the regression model. We believe that this feature helps us to find similar documents that are about one specific topic. This feature assigns a numeric similarity value to a pair of documents and the value is greater for the pairs that have a shared dominant topic. We used semantic similarity of words to find word groups or clusters that are closer to each other semantically. Then we pick the group which represents the dominant topic of the document and use only those words to find similar articles.

In this thesis and this particular module of the thesis, we used skip-gram model for word representation(Mikolov et al. [2013]) that was pre-trained on Google news corpus[2]. In this model, each word is a vector in a hypothetical, high-dimensional space. This model preserves relationship of the words through the vectors. For example, words that have closer meaning to each other are placed closer to each other in the space. Also, if we add two vectors of two words, we will get a vector

[2]https://code.google.com/p/word2vec/

similar to a word which covers meaning of both words. For example, if we add vectors of words "man" and "woman", we get a vector very similar to the word "human". Each element of the vector defines one hypothetical aspect of similarity between words, so cosine similarity of the vectors of the words represents semantic similarity of words. The word vectors are learned from massive text datasets which is described in appendix A.

The problem of finding important words in a single document is called keyword extraction. Our goal is to find topical keywords of a text and use them to measure similarity of two documents using different similarity measures. The main usage of keyword extraction is in summarization. However, in information retrieval, it sometimes improves the precision of retrieved documents if noisy words are removed. One of the widely used approaches for removing non-informative words is removing stop words. The keywords must be selected very fast, and with the least amount of information loss. We propose a novel one-pass algorithm using word clustering and a stochastic model to eliminate words that are not related to the main topic of a single text document. We claim that having only important words instead of normal bag of word model is a better representation toward modeling relatedness if the extracted words represent the core topic of a document.

Our solution is to remove noisy words from a single document and keep only words that are truly relevant to the main topic of the text and use those words to measure similarity with other documents. For example, if an article is about environment, it is likely that it also touches on economic issues. If we extract keywords representing only the dominant topic, then we can find related documents about only that topic.

3.3.1 Modified Chinese Restaurant Process

In our method, we use Chinese restaurant process (CRP) to perform clustering on all of the words of a text, because we need a one pass linear algorithm in which each word is seen only once during clustering. In this model, clusters are tables, and words are people. In the Chinese Restaurant Process or CRP, we have an infinite number of tables, each with infinity capacity. At each time step only one customer enters the restaurant. The first customer will be seated at an unoccupied table with probability 1. At time n + 1, a new customer arrives and has to decide where to sit: either a

new table will be assigned to the new customer or she will choose to sit next to some people who are already at an occupied table. The former happens with a probability α which is called CRP hyper-parameter. The later will happen with a probability based on number of people who are already seated on the table. Here is the formal definition:

$$P(z_i = k|z_{1:i-1}, \alpha) \propto \begin{cases} n_k, & \text{if } k \leq K \\ \alpha, & \text{otherwise } k = K + 1 \end{cases} \tag{3.1}$$

where k is the index of selected table, n_k is the number of people sitting at table k, α is the hyper-parameter, and z_i is the new person entering the restaurant while there are $i - 1$ people in the restaurant.

We modified the formal definition of CRP to use similarity of word vectors that are coming from skip-gram model in our cluster assignment. In our model, we have to keep track of one extra item which is the sum of all vectors within a cluster which we call cluster vector. The cluster vector represents what a cluster generally means based on the words it contains. In this method, for each new word, we look into cluster vectors of all current clusters and find the one which has the maximum cosine similarity with the new word vector. If the similarity is greater than α then we assign the new word to the target cluster. Otherwise, we consider creating a new cluster with probability α. It means that if the new word is not very related to the cluster, there is still a chance, to assign it to that cluster. This property prevents creating too many word clusters. There is one distinct difference to traditional CRP: if the new word does not go to an empty table, it deterministically goes to the most similar table.

Using the Chinese restaurant process to cluster words was already done in other domains. In one of the works, a hierarchical clustering for words was created by a nested Chinese restaurant process (Blei et al. [2010]). They also proposed another method for a distance dependent Chinese restaurant process Blei and Frazier [2011]. But our work is different since we are working with cluster vectors and not word vectors alone. To the best of our knowledge, it is the first time that both skip-gram model and CRP are used for keyword extraction.

The intuition behind using the Chinese restaurant process is that each table or

Algorithm 1: Clustering words of one document into different topical groups using modified Chinese restaurant process

Data: An array A representing all tokenized words

Result: A set of arrays, each array represents a cluster of words

Remove stopwords from input array A;

Create an empty cluster set clusterSet;

Create an empty set representing each cluster vector clusterVec;

Create a cluster with the first word and put it in clusterSet;

Set the first cluster's vector to the vector of the first words;

p = 1.0;

forall the *Word w in A* **do**

    pNew = p / (number of clusters);

    maxSim = find the similarity of most similar vector to w in clusterVec using cosine similarity;

    if *maxSim* > *pNew* **then**

        Add w to the most similar cluster;

        Add vector of w to the vector of most similar cluster;

    else if *pNew* > *randomNumber* **then**

        Create a new cluster;

        Put w into the new cluster;

        Set new cluster's vector as w's;

    else

        Add w to the most similar cluster;

        Add vector of w to the vector of most similar cluster;

return clusterSet;

cluster represents a topic and each word in that cluster is related to the topic as well. Since each word contributes to the meaning of topic by influencing the cluster vector, an incoming word is likely to be assigned into the proper topic. In this way, words will be grouped topically using skip-gram vectors while losing the ability to assign more than one topic to each word (e.g. soft clustering of words as LDA). One of the good properties of this method as opposed to LDA is that it does not require the knowledge of the number of cluster (topics) in advance, and clusters are created dynamically. Since we are going to focus on only words of one topic later, there is no point to have a soft clustering.

3.3.2 Selecting Keywords of Text

The word clusters for an article about economy of province of Alberta and its current workforce wage are extracted in figure 3.1 (the raw text is available in appendix D). Having the word clusters of a given document, the task of keyword extraction is now reduced to selecting the cluster which contains the most important words. We use standard TF-IDF to assign scores to words, and then for each cluster, calculate the average TF-IDF score of its words. At the end, we choose the cluster with highest score to represent keywords of the input document (Figure 3.2). We will later use literal presence of the words of the selected cluster to retrieve similar articles. The retrieved articles will be mainly about the topic represented by the selected cluster since we are not using words which may be either general words or about other topics.

Having a keyword representation of a document, we believe is informative enough to be used for obtaining similarity of documents beside other lexical features. In order to use extracted keywords in calculating relatedness, we concatenated the keywords of each document and formed a query string. Then we passed it to Lucene search engine to extract documents which contains the exact keywords. We use cosine similarity of our keywords and the entire body of documents to find related documents and the corresponding scores. In this way, we only get high similarity values for documents which are about one specific topic. There are other ways of using the extracted keywords. One of the them is to do a summation of the vectors of keywords of a document and then represent documents with that vector. In this way, we can measure semantic similarity of two documents by computing cosine similarity of their

Run 1	Run 2
jim,happy,feel,noises,loud,argument,drum,lots,sounding,grass,springtime,tend,charlie	springtime,noises,lots,happy,sounding,loud
teachers,private,people,company,worker,governments,firms,employees,corporations,increasingly,global,giant,government,public	province,jim,oilpatch,national,pay,drum,provincial,glen,oil,canadaalberta,charlie
alberta,province,oilpatch,canada,national,glen,provincial,oil	western,reopening,exist,happened,close
earns,vs.,lower,bonus,stores,statistics,typically,cut,costs,expected,widens,income,car,competitive,household,expecting,brings,bonuses,workers,outlook,economist,cent,recession,sector,economic,slip,budget,pressure,year,upscale,wages,revenues,job,energy,wage,sustain,accumulated,compensation,fiscal,increases,slashes,ways,conference,contracts,executive,weekly,hurting,position,turns,nominal,loss,spending,average,pay,hole,added.,freezes,high,median,numbers,higher,predicts,dealerships,unions,growth,eke,rollbacks	board,outlook,company,bonus,feel,argument,employees,stores,statistics,budget,income,increasingly,car,competitive,contracts,household,bonuses,workers,economist,cent,recession,giant,sector,economic,pressure,wages,lower,firms,counter,revenues,job,energy,wage,fiscal,compensation,increases,private,teachers,conference,people,ways,alert,executive,government,worker,weekly,upscale,position,nominal,spending,public,rollbacks,control,corporations,high,median,numbers,chief,higher,governments,global,costs,dealerships,unions,growth,hurting
	freezes,predicts,accumulated,average,eke,widens,turns,slip,expecting,slashes,added.,expected,vs.,loss,year,cut,typically,sustain,earns,brings,tend

Figure 3.1: Word cluster results after two runs on the same article.

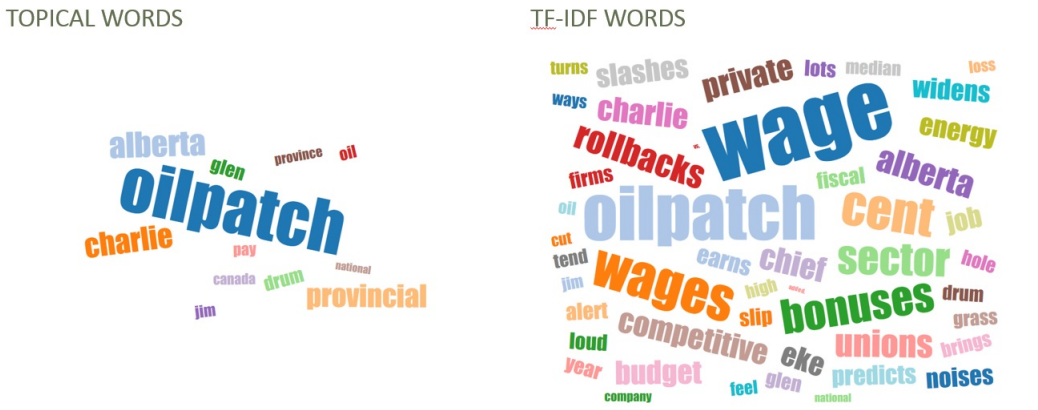

Figure 3.2: Topical word profile vs. Tf-idf keywords profile of an article

document vectors, even when they have no words in common. However, we observe that using this method will assign high scores to documents which are similar in a general domain like politics or sports, whereas in our task of news recommendation, we need a similarity measure which is more specific.

One other application of keyword extraction for the news domain is tagging. The keyword model for documents represents words from the text that are very important. Using them for summarizing news article is one of the directions of future work. This feature is called *Keywords* in the evaluations.

Our initial experiment to extract an informative subset of words was to apply clustering algorithms to words of a document and using their vector distance as the

cluster distance criteria. We used K-Means as our clustering algorithm and preliminary results were promising. But there was two major problems with this approach. The first problem was that we were using Euclidean distance of words to put them into clusters. Although related words are close to each other based on their vector representation, it is their cosine similarity which captures the semantic meaning of two words and K-Means is not well-suited for this type of similarity. The second problem is that we need to know the number of clusters; I.e, the number of topics in a text which is usually unknown.

We used skip-gram similarity, which is a state-of-the-art semantic similarity of words, to create a new representation for documents and used it for our similarity task. Additionally, unlike other keyword extraction methods, our method is unsupervised; It does not require training on a corpus and it operates only on a single document. The skip-gram model requires training on large datasets to model vectors of words. In this thesis, we used a pre-trained model from Google News which is available online[3]. To the best of our knowledge, this method is a novel approach for using skip-gram model for modeling documents.

3.4 Semantic Similarity Features

This section introduces a set of features based on named entities and their semantic similarity for our regression model. We use named entities that are extracted from the text to calculate semantic similarity of a pair of news articles. At first we briefly introduce Tulip as our named entity recognizer and disambiguation system. Then we define functions that calculate similarity of two named entities. Finally, we used those functions to define different similarity measures of two news articles which are the features that are used in this thesis.

The named entities are extracted using Tulip (see appendix C). Tulip is a named entity recognition and disambiguation (NERD) system which takes one text document as input and annotates the named entities by linking them to Freebase[4] knowledge base. By annotating the text, an article can be represented as a set of entities. In knowledge bases, especially Freebase, we can infer valuable information about

[3]https://code.google.com/p/word2vec/
[4]http://www.freebase.com/

relatedness of two entities by considering various information.

The task of recognition and disambiguation of named entities in a text is a fundamental task in the systems that rely on knowledge base information. Since it sits at the bottom of the stack of modules in such systems, any errors will propagate through the system and reduce performance of later components. Tulip is currently stat of the art among NERD systems and won the first prize in named entity recognition and disambiguation challenge 2014 Lipczak et al. [2014]. In addition to its high precision, the response time of the system is very short compared with other system. This property makes Tulip suitable for our recommender system in which we have many articles that are required to be annotated. In this thesis, each article is passed to Tulip and the set of returned entities are used to build an entity profile for each article.

A group of our proposed features to measure semantic similarity of two documents are based on semantic similarity of their extracted named entities. In this section, we introduce five features to capture semantic similarity of two documents. We followed the work by Huang et al. [2012] and assume that we have a named entity profile for a given document. They demonstrated that by introducing different numerical features and combining them, we can capture semantic similarity of two articles. Their numerical features are extracted from sets of named entities by applying a similarity function which takes two named entities and returns a similarity values. They used a similarity measure based on graph structure of Wikipedia (Witten and Milne [2008]).

3.4.1 Semantic Similarity of Two Named Entities

Although graph based similarity based on Wikipedia is a proven approach, we decided to use a newer state of the art skip-gram model to calculate similarity of named entities (Mikolov et al. [2013]). We also used a new system called Sunflower (see appendix B) which provides similarity considering Wikipedia categories and compared it with skip-gram model in terms of its performance in the final recommender system. One advantage of using skip-gram over the graph based method is that it can be trained on any text corpus and since we are working on news domain, we used vectors that were trained on news domain same as the ones that were used in section 3.3. But

in this case, the vectors were trained by considering the mentions of named entities instead of only words. The semantic similarity of a pair of named entities, e_1 and e_2 is defined by using cosine similarity of their corresponding vectors:

$$sim(e_1, e_2) = cos(V_{e_1}, V_{e_2}) \tag{3.2}$$

in which V_{e_1} and V_{e_2} are vector representation of e_1 and e_2 respectively.

Sunflower is a system developed at Dalhousie University by Marek Lipczak[5]. This system is based on different language versions of Wikipedia and provides a category profile for each Wikipedia article. Since each named entity in Freebase has a corresponding article in Wikipedia, we can use Sunflower to find the vectorial representation of the entity in Wikipedia category space. Then we use equation 3.2 to calculate similarity of two named entities by using category vectors.

The category graph of Wikipedia is very complex and huge, but Sunflower provides accurate and sparse representation of articles by limiting the amount of generalization. According to our preliminary experiments, in our news recommendation task, we have to stay specific on the subject of news, so we set the Sunflower parameters to provide the most specific representation as possible to calculate similarity by setting its *width* to 1 and its *height* to 3 to get the best results (more details in appendix B).

3.4.2 Semantic Similarity of Two Articles

We introduce 5 features inspired by Huang's work. As we will discuss in chapter 5, the skip-gram model gives us better similarity function. Therefore, in experiments that we do not specify Sunflower similarity, we are using skip-gram model as default choice. However, all of the semantic features are defined regardless of the similarity function. They can be replaced with other similarity methods at any time.

If we have $sim(S_{Ai}, S_{Bj})$ as a semantic similarity function of two named entities in two sets and S_A and S_B representing two sets of entities of documents A and B respectively, the first feature is the average of all possible similarity values between entities of two sets. Formally, the value for this feature is:

[5]This work has not been published yet.

$$sim(S_A, S_B) = \frac{\sum_i^{|S_A|} \sum_j^{|S_B|} sim(S_{Ai}, S_{Bj})}{\mid S_A \mid . \mid S_B \mid} \tag{3.3}$$

We call this feature *AvgSim*. We add a prefix to describe which similarity function was used for this feature (either Sunflower or Skip-gram).

The next feature is calculated with a slight change to previous formula. The intuition behind this feature is that similarity of two sets of entities is usually dependent of the maximum similarity score of their entities:

$$sim(S_A, S_B) = \frac{\sum_i^{|S_A|} max(sim(S_{Ai}, S_{Bj} \in S_B))}{\mid S_A \mid} \tag{3.4}$$

In other words there is usually at least one entity in another set which has the highest similarity score. The average value of all maximum scores can remove the effect of less important and less related entities. Two other features are also used which represent minimum and maximum similarity between two sets. This feature will be called *SumOfMax* in evaluations.

The other two features are trivial. They are maximum and minimum similarity score between all pairs of entities between two sets (*MaxSim* and *MinSim* in evaluations respectively). We also used Jaccard index of two entity sets as an additional feature. This feature captures exact presence of entities in two documents. The Jaccard index of two sets is defined by following formula:

$$J(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \tag{3.5}$$

Where S_A and S_B are the sets of named entites in two documents. We call this feature in evaluations *JaccardSim*.

3.5 Other features

We discussed some of our features so far based on named entities and skip-gram model. As we mentioned earlier, we try to combine the semantic features through a regression model with other well known features to model relatedness. In this section, we start by introducing 6 features that are based on lexical similarity of news articles. Then we introduce LDA-based similarity which uses a topical distribution of words to capture relatedness. Finally we introduce a feature based on the publication date of news articles.

3.5.1 Lexical Similarity

Although relatedness of two news article is not as same as their lexical similarity, having the same word can give us a lead about relatedness. Both Lv et al. [2011] and Bogers and van den Bosch [2007] argued that standard information retrieval similarity measures can capture important aspects of relatedness which are novelty and relevancy.

We defined 6 lexical similarity features. The first one is cosine similarity of two documents using their term frequency vectors. The reason we did not use TF-IDF is that this aspect of relatedness has been already captured according to our architecture by using Lucene as a filter. So it would be redundant to calculate it again. Instead we only calculate TF vectors and cosine similarity of two documents based on TF. We call this feature *CosineTF* in our evaluations.

$$CosineTF(A,B) = \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2} \cdot \sqrt{\sum_i B_i^2}} \tag{3.6}$$

The second feature is Jaccard index of terms in body of two news article. It can be calculated using formula 3.7 by having A and B as the sets of terms in two documents. We call this *JaccardBody* in evaluations. If we use formula 3.7 with sets representing terms in title of two articles, then we can obtain numeric values for another feature which indicates similarity of titles of two article. This feature is called *JaccardTitle*.

$$J(A,B) = \frac{|S_A \cup S_B|}{|S_A \cap S_B|} \tag{3.7}$$

Okapi BM25 is one of the standard scoring methods in information retrieval. It is based on the probabilistic retrieval framework developed in 1970s and it works well in many IR tasks. This score can be obtained using following formula:

$$score(Q,D) = \sum_i^n IDF(q_i).\frac{f(q_i,D).(k_1+1)}{f(q_i,D) + k_1.(1-b+b.\frac{|D|}{avgdl})} \tag{3.8}$$

where IDF is inverse document frequency, $avgdl$ is average length of all documents in corpus, and k_1 and b are free parameters. In previous works, BM25 was one of the most important feature. We call this feature *BM25* in the evaluations.

Another lexical feature is based on language modeling. To calculate this feature, a language model is created for each document. Then for each query, the probability

of the query text given the language model is calculated. The main property of this type of similarity is that it partially models the local dependency of the word in the text. There are two famous smoothing for this type of similarities: Jelinek-Mercer and Dirichlet smoothing. According to Lv et al. [2011] the former shows similarity much better than the later. Also our preliminary results suggested using Jelinek-Mercer smoothing. So our language model similarity features uses only this type of smoothing. We used the implementation of this type of similarity available in Lucene package. This feature is called *LM* in the evaluations.

Our last lexical feature is information based similarity score. This similarity model is based on the work by Clinchant and Gaussier [2010] and is based on the fact that the difference of behavior of a word at the document and collection levels brings information on the significance of the word for the document. They showed in their paper that their model leads to a simpler and more effective similarity measure for ad-hoc information retrieval. We used the implementation of this formula in Lucene software package. In evaluations this feature is called *IB*.

3.5.2 LDA-based Similarity

Topic models are a group of methods that uncover the hidden topical and thematic structure in document collections. Blei et al. introduced Latent Dirichlet Allocation in 2003 and it quickly became one of the most famous and effective methods for topic modeling. Given a collection of documents which in our case is the total news corpus, LDA returns distribution of topics for each document and distribution of topics of each words. For each document pair, Jensen-Shannon divergence of their distributions of topics can be seen as a measure of dissimilarity. To apply LDA on our corpus, we used MALLET[6] library. LDA needs to know the number of clusters beforehand. We set it to 200 after doing empirical tests on the corpus to get the best results.

As explained before, Jensen-Shannon divergence represents the difference between two documents in terms of their distribution over each topic. Since LDA returns a vector representation of documents in topic space, we can use cosine similarity of the vectors to represent similarity of two documents. Our experiments revealed that cosine similarity is more powerful and informative than divergence measure to be a

[6]http://mallet.cs.umass.edu/

feature for relatedness. Hence, instead of KL or Jensen-Shannon divergence, we used the equation 3.6 and used cosine similarity of topical distribution of two articles. This feature is called *LDASim*.

3.5.3 Time Similarity

The last feature represents how close are two news articles in terms of their publish time. The intuition behind that is that when news articles are covering a story, that story develops quickly and related news articles about that story are published one after another. This feature is simply the absolute value of the time difference between two news articles. This feature is called *TimeSpan*.

3.6 Labeling of Training Examples

In this section, we explain how we calculate the relatedness score of two news articles to be used during training the system. To train our regression model for relatedness, we need to have both positive and negative examples. For positive examples, we can use explicitly labeled related pairs. However, we have no information about negative examples. In this section we discuss different strategies to find negative examples.

For our scoring model, we have the option of creating a binary classifier to assign a class label as related or not related to a given pair of news article. However, since we need to provide a ranked list of related news articles, we focus on a regression model to produce relatedness score. Sorting output list by their relatedness score will help us generate the desired output ranked list. When working with regression models, we convert positive labels to numeric value 1.0 and negative labels to numeric value 0.0. Applying a threshold on the final output will keep only related documents. The threshold will be based on user's need: lower threshold will provide more news about other topics.

3.6.1 Data Model 1: Random Selection of Negative Examples

In this model, positive examples are those labels that are explicitly labeled as related. For negative examples, we select pairs randomly and those that are not in our positive set of pairs, are assigned as negative. The number of these randomly selected are the

same as positive examples to keep the dataset balanced.

3.6.2 Data Model 2: Using Lucene for Generating Negative Examples

In section 3.1 we introduced the idea of using Lucene as a filter. If we use that strategy in production, then Model 1 would not be compatible with it. In other words, our relatedness model is going be to used to re-rank the output of Lucene results. So the data set must be prepared in a way to assist the system doing the re-ranking task. The trained model will still be a relatedness model, but the data points will resemble the final production environment during training.

In order to have a compatible data model, we used Lucene as a filter in the process of making dataset in exactly the same way as we use it in production environment. In this setting, each article in our training set is passed to Lucene to get top 45 results. Then for each of results, we check if they form a positive pair or not according to the gold standard. Finally we sub-sample our negative instances to make it a balanced training set.

3.6.3 Data Model 3: Graph Representation of Relatedness

One problem that arises in our ground truth dataset, is that sometimes article A is related to article B and B is related to C but there is no labeling for the AC pair. In situations like this, it is not rational to label AC pair of news articles as a negative or not-related. Also, we cannot say they are surely related, because it is not rational to label real related articles (i.e labeled by editors) and these partially-related articles as related. To handle both situations, we proposed making a graph of relatedness from the existing related pairs in the dataset. In the proposed graph, each node represent a news article and there is an unweighted non-directional edge between two nodes if they are labeled as related. If we have V as set of nodes which represents set of news articles and E as set of edges, the formal definition of each element of adjacency matrix of relatedness graph $G(V, E)$ is as follows:

$$a_{i,j} = \begin{cases} 1 & \text{if } V_i \text{ and } V_j \text{ is a positive pair in training set} \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

Figure 3.3: A sample view of connected components in the training set of The Chronicle Herald dataset.

In the relatedness graph G, the relatedness of two nodes can be represented by the shortest path between those nodes. That is if two nodes are closer to each other, they are more related than two nodes farther than each other. We used this scoring scheme of relatedness to label document pairs in order to use them to train our models.

We also observed that there are a lot of isolated communities in the relatedness graph. Fig 3.3 shows the relatedness graph of The Chronicle Herald dataset. If we look inside of some communities, we can easily observe that they are related news articles about one story. We will call each of these communities a news **story thread** since they are about one subject like a discussion thread in forums of email threads. We will later consider these story threads in our evaluations.

The relatedness graph enabled us to prevent assigning an unrelated label to two articles in same news thread by introducing **partial relatedness**. In order to improve the precision, we still need to learn characteristics of a non-related pair of news articles to improve accuracy of the data model. To solve this problem, we randomly select nodes from two different news threads and consider them as not-related. We limit

the number of times we do the random selection to have a fairly balanced dataset. In order to see the effectiveness of this method of constructing a training set, we call this method *Model 3*.

Algorithm 2: Finding distance of nodes in the graph which represents how similar two node are in the relatedness graph. Calculated scores can be converted to relatedness scores directly. In the relatedness graph, nodes are articles and edges represent explicit relatedness in the dataset. Edges are not weighted.

Input: Set of news article pairs A in which each pair is a related pair of news articles

Output: Set of relatedness scores for each pair in A.

nodes = make a graph node for each news article in A;

edges = make an edge for each pair in the A;

Create graph g with *nodes* and *edges*;

returnSet = new empty set;

forall the *News article a in A* **do**

node = find the corresponding node to a in g;

(*reachableNodes*, *distanceToRoot*) = apply BFS on g with *node* as the root;

forall the *Node r in reachableNodes* **do**

add (*node*,*r*,*distanceToRoot*(r)) to *returnSet* ;

return *returnSet*;

3.7 Relatedness Model

In this section we discuss various strategies to use our different training datasets for creating relatedness model. We used two open source libraries: Weka[7] and RankLib[8]. Weka provides different algorithms including feature selection, pre-processing tools, classifiers and clustering tools. RankLib is a collection of different learning to rank algorithms implemented in Java.

[7]http://www.cs.waikato.ac.nz/ml/weka/

[8]http://sourceforge.net/p/lemur/wiki/RankLib/

3.7.1 Feature Selection

Not all the features that we introduced are used to calculate relatedness of two news articles. Although combining different similarity measures means combining different aspect of relatedness, some features may introduce noise instead of helping other features. For example some features may capture very general theme of news articles that is not useful.

Since testing features by user experience is very expensive, feature selection must be performed before introducing model in the production environment. So most of the experiments regarding the performance of feature extraction methods were performed offline. We collected all feature value and then used Weka package to apply different feature values and test different combination of features. Weka application provides all required tools for calculating importance of features.

In order to find the importance of individual features, we used Correlation-based feature subset selection (CFS) (Hall [1999]). This method assigns a merit value for a subset of features which is on the basis of the following hypothesis: good feature subsets contain features highly correlated with the regression value, yet uncorrelated to each other. In other words, features that are independent from other features and correlates more with regression values get better scores. The correlation score with the regression values can be calculated using Pearson's correlation coefficient or Spearman's ρ. However, Hall used three different measures of relatedness in his method: minimum description length, symmetrical uncertainty and relief.

In addition to CFS feature evaluation, we conducted feature selection based on two other methods. The first one is ranking features based on information gain. This method evaluates the worth of an attribute by measuring the information gain with respect to the regression value. In other words, it ranks the attributes by the order of their separating power. The second method is called *Relief* which assigns a score to the attribute based on its ability to separate close instances. If two instances are close in the feature space, for a given attribute, it is desirable to get *close to equal* values; otherwise the a negative score will be added to the overall score of the attribute.

3.7.2 Regression Models

The problem of training a regression model has been broadly studied in the machine learning community. There are a lot of options available to pick among many regression algorithms. Since we wanted to show the power of introduced features and data model, we focused on a few well known algorithms. One property of the features used in this research is that we have only one feature (time difference) that may get a value out of the range of what there is in training set and all other feature values will remain in a fixed range. So practically, the feature space created by training data will not change in the future and any regression algorithm that divides the feature space meaningfully will perform well for the future data points. Based on our experiments, we found that regression trees provide best performance for the system. So our main focus is on regression trees.

The concept that the regression tree is trying to learn and describe is news relatedness. It will separate feature space into subspaces in which most of data points have the same value. The relatedness score of a given pair in the future will be calculated based on the subspace it goes to. Given this definition, it looks like an interpolation problem and tree based algorithms are usually preferred for this kind of problem since they can create a complex set of subspaces in the feature spaces. In other words, the regression tree will transform a set of attributes, each representing one aspect of relatedness, into a relatedness score:

$$relatedness\_score = f(\overrightarrow{X}) \tag{3.10}$$

where X is the input feature vector and f is the learned relatedness model. The way that the regression tree obtains values for f is completely based on how it divides the feature space into smaller subspaces. During evaluation, a feature vector will end up being placed in one of the subspaces and its value will be calculated based on other data points in that subspace.

There are many parameters to consider for selecting a algorithm for regression. The first important property of an algorithm is its speed both for training and testing. Given the size and novelty of the problem, training time is important and allows us run many experiments with different parameters. An example of very slow algorithm that needs tuned parameters for training is SVR (Support Vector Regression) with

some complicated kernel functions.

Decision or regression trees easily fit to the problem. They rarely require any parameter, and they are trained very fast. The only downside of using them is their problem of overfitting to the training data. As mentioned earlier, this is an interpolation problem, so overfitting will not be a serious issue. Also, there are ensemble methods which solve the overfitting problem.

The first regression model which was trained on our data models was a simple regression tree. We used the REPTree implementation from WEKA package which is a simple incremental regression tree with greedy backward pruning. The loss function in this tree is based on variance reduction, exactly the same as CART (Olshen et al. [1984]). This implementation allows to apply pruning using back fitting to reduce error on a percentage of learning data as verification set.

When using data model 1 or 2, the scores in training set is either 0.0 or 1.0. However, to use our 3rd data model, we had to divide range of 0 to 1 into equal intervals. The number of intervals are calculated based on longest possible path in the graph of relatedness. It means that if the longest path is 4, then we score relatedness of pairs using values 1, 0.80, 0.60, 0.40, 0.20, and 0 for very related to completely unrelated. Note that in this case, the 0.20 will be assigned to a news article which is four steps away from the query article. The 0 score is assigned to a completely irrelevant article based on the graph structure.

The second tree based method that we used for regression is gradient boosted regression trees (Friedman [2000]) or GBRT. Like other ensemble methods, this method combines some weaker regression algorithms which are typically regression trees to solve a general regression problem. It works based on reducing mean square prediction error by fitting a new tree at each iteration of gradient descent algorithm. The only required parameters for this method are maximum number of trees and maximum number of leaves of trees. We used an implementation in RankLib[9] package.

We did not use SVR methods since it requires finding and tuning kernel and regularization parameters. Our preliminary experiments suggested that the data points are not separable at all. Statistical data from size of the decision trees also suggested that it is a very hard problem to solve for methods requiring separable data

[9]http://sourceforge.net/p/lemur/wiki/RankLib/

points.

3.7.3 Learning to Rank

The news-article recommendation problem can be seen as a learning to rank problem. For each query document, we have a list of related documents with its relatedness score. The relatedness scores are not assigned by human judgment, but we can use our graph-based data mode to train a ranking model to re-construct the ranked list. In this way we can evaluate how well the ranking model performed not only for finding explicit related pairs, but also for finding related articles in the same story thread. In other words, we can consider each node of the graph as a query article and each node that is in the same story thread is related, but each related news must be ranked based on their relatedness score. This way of evaluation will gives us insight about the performance of the system as an exploratory system.

Learning to rank algorithms use different loss functions to optimize their performance of learning. Their general behavior can be categorized in three groups: pointwise, pairwise, and listwise.

Pointwise algorithms are the same as regression model on the entire dataset. So each feature vector is used independently of other vectors, and is used to minimize a loss function. It is the same as supervised regression or classification tasks and the final relatedness model tries to predict the relevancy scores as closely as possible.

Pairwise methods use the relevancy scores to create a set of pairwise constraints. Then constraints are classified using existing supervised algorithms. Pairwise algorithms do not predict relevancy scores, but produce a ranked list of results. So there is no way to apply any sort of threshold to distinguish between highly related and slightly related results. On the other hand, pairwise algorithms are always preferred over pointwise algorithms when final ranking is more important than relevancy scores, especially if click data is used as major source of training.

Listwise methods take the advantage of the structure of the ranked list in the training data and try to optimize any given evaluation measure as opposed to other methods which optimize an internal loss function. Like pairwise methods, the output is a list of ranked results and there is no meaningful scoring available. Using different loss functions mandates us to apply threshold on only top k results. For example if

the algorithms loss function optimizes precision of the top two results (P@2) then only first two results must be considered in evaluation. These methods work great in traditional recommendation situation in which there is a user and a list of preferred items that must be recommended to her. These systems are usually optimized for a certain measure which is defined by the business value of the recommender.

Loss functions of listwise methods are based on evaluation metrics of learning to rank algorithms. This is important since the trained ranking model will be evaluated based on exactly what it is supposed to do in production environment. Usual evaluation measures are Mean Average Precision(MAP), NDCG@k and Precision@k. We used these measures to evaluated our method for the situations where a ranked list of results are preferred.

We trained and evaluated different learning to rank algorithms and it turned out that pointwise methods (regression models) are always outperforming others. The main reason can be their ability to provide interpretable relatedness score which can be used directly for removing less related articles. In our task, for each query document, we have a variable number of related news. Unlike traditional ranking problems which are often based on top k results, a meaningful threshold must be applied to prune unrelated articles. The other reason that pairwise and listwise methods did not perform well was that our similarity features are designed to capture relatedness globally. Those two methods are usually successful when there is a user-item matrix and items are ranked relative to the user by defining local similarity functions.

Chapter 4

Implementation

In this chapter we briefly discuss some important aspects of the implementation. The main purpose of this chapter is to discuss how different components are connected together, what the bottlenecks are and demonstrate possible ideas for future extension of the application.

4.1 Implementation Overview

The recommender system that is being discussed in this thesis is implemented as a web application which can be deployed at any news web site. The system must be responsive enough to handle a large number of concurrent people reading different news articles and getting recommended articles. Also, it must accept and index newly written news articles to recommend them in response to an old article, or recommend older articles for them.

There are two tasks executing in such a system. The first one is the main task of the system, which is recommending related articles, given a new article. The second task is indexing each news article when is it being published and put it into the repository of the recommender. The former requires three sub tasks:

1. Extracting features that represent relatedness of a given news article to all other articles.

2. Calculating similarity of the given news article and all other articles based on the features.

3. Sorting other articles based on their relatedness score, and apply required thresholds to prune unrelated results.

The most important property of such recommender system is being as quick as possible in producing a ranked list of result for a given article. The number of newly

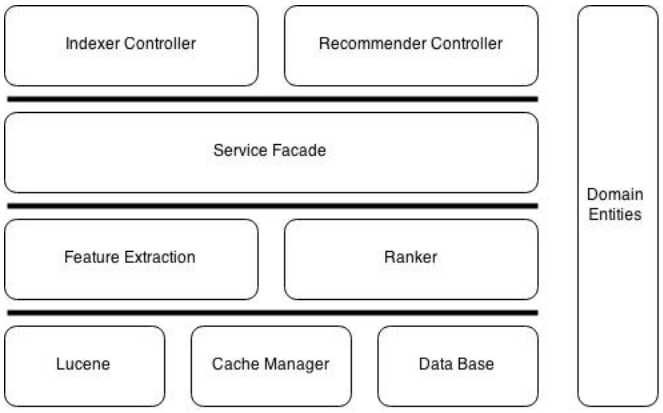

Figure 4.1: Layered architecture of the system.

written articles is far smaller than the number of visits. So all the expensive tasks of indexing and extracting features can be done while the news article is being published. However, producing a ranked list of related news articles must be done quickly and dynamically. It must be dynamic, because new articles are published every day and there can be a new article more related to a given article than currently identified related ones. It must also be as quick as possible, because users do not like to wait to see related news articles.

We used a layered architecture for our software architecture (figure 4.1). There are two controller components in this system which harmonize the tasks required for each scenario: indexer and recommender. Indexer is mainly responsible for getting a raw and newly published news article and put it into the system in a proper way for later use. The recommender gets one news article identifier at a time and returns a ranked list of related news articles. Other layers deal with feature extraction, scoring, and managing repository data.

In our layered architecture, the interaction with the environment is performed only through controllers. The controllers' main responsibilities are preparing input for the rest of the system and preparing output in a usable format for external use. The controller is placed on top of a business layer. The business layer represents the domain model of the system which contains all the objects required for all different tasks. This layer is composed of many components including feature extractor, ranker, indexer, and recommender. Feature extractor is responsible for extracting features for a given pair of articles, ranker uses the extracted features to generate a relatedness

score, recommender uses ranker to generate output, and indexer persists a new article, the results of feature extractor, and ranker. The entire business layer is placed on top of data layer which is responsible for transactional persistence and retrieval of information to and from database. We used Hibernate[1] object-relation mapper (ORM) technology in our data layer. The underlying data base can be different and tuned for better performance for a news web service.

Since features are extracted for each pair of documents, the amount of space required for storing feature values is $O(N^2)$ given N articles in the system. On the other hand, having all the article pairs and their features in a repository can improve the performance of the system and eliminates the need for recalculating costly features. Based on our architecture, whenever a new article is begin published, it will be persisted into a database itself. Then this article is compared to all other articles in the database and for each pair, a feature vector is extracted and persisted. All these works should be done during indexing. So the recommendation task can be done by passing feature vectors of an article to a scoring model and extracting the produced score by models. In this way the performance of the recommender is dependent only on finding and extracting feature vectors as assigning scores to feature vectors is usually a cheap task. Fig 4.2 shows the major component of the architecture.

As we discussed about our first research question in chapter 3, before working on different features, a prototype system was created with the following simplified model using Lucene search engine:

1. When a new article is being published, it is indexed by Lucene.

2. When an article is being passed to the system to collect other related articles, it creates a query from the body of the input article and return Lucene generated results given that query.

This prototype had all the components including an indexer and recommender controller. In this case, the indexer controller was only calling the Lucene indexing service and the recommender was responsible for creating a query, passing it to Lucene, and returning Lucene results.

[1] http://hibernate.org/

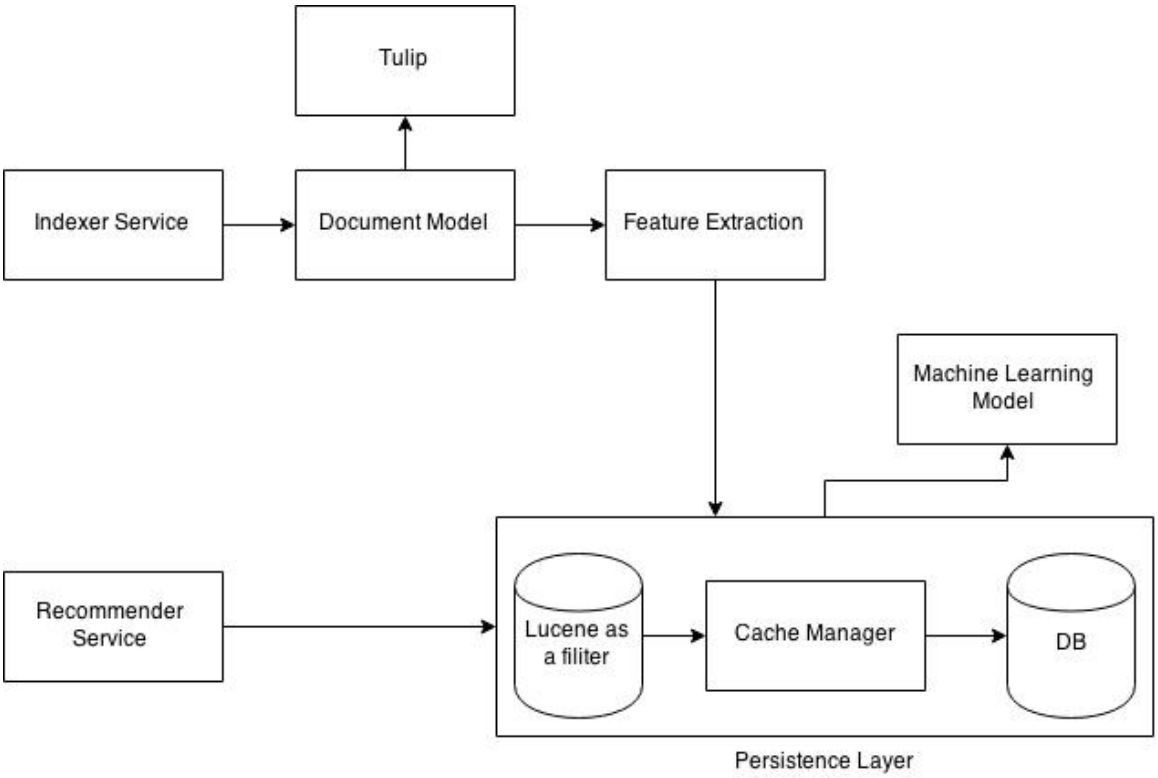

Figure 4.2: Components of the system. Arrows show dependency of components.

Using Lucene as a filter to produce C results for a query document, the space required to keep all feature vectors is reduced from $O(N^2)$ to $O(C \cdot N) = O(N)$, while keeping speed of the system as fast as possible. The goal of this strategy is re-ranking the returned results from Lucene to improve its precision while keeping recall as high as possible. In best case scenario, the precision and recall of the system will be 1.0 and 0.831 respectively. According to table 3.1 C is set to 45.

Using Lucene as a filter leads to idea of using a cache to maintain a list of pre-calculated similarity scores. The cache is responsible to keep track of the top C documents for each articles. The cache can be also used to keep track of the similarity score, so recommendation task would be done by performing a cache check and returning results. During indexing of a given document, the cache for that document is empty. So the article is passed to Lucene, and top C results is retrieved. Then new document and the returned C documents will make C document pairs. The feature extraction will extract the feature values and put them into the cache. A call to our scoring component makes the procedure complete by updating the relatedness scores

in the cache.

When recommender is being called, a call to Lucene is made to get C results. The returned results are compared to the cached values and a cache hit or a cache miss can happen. Those values in the cache that are not in the returned list are removed and all articles that are not in the cache (cache miss) are sent to feature extraction component and persisted in the cache. This type of cache management ensures that new articles are always considered to recommend for a given article. Additionally, this strategy has a huge impact on the response time of the system by limiting all required feature vectors to 45 at each query. We persist the cache inside the database, and do all cache management in the business layer of the application. Indexing information by B-Trees in database makes searching logarithmic and since we have at most 45 entries in the entire cache table, the search and retrieval task happens in sub-linear time.

The summary of what happens in the system during indexing based on figure 4.2 is as follows:

1. A new article treated as the query article is sent to the indexing service.

2. The article is passed to Tulip and a document model (section 3.2) from its named entities and tokenized body is created.

3. The document model is passed to feature extraction component. Since feature vectors represent different similarity measures of a pair of articles, the query article is passed to Lucene and a number of relevant articles are retrieved. For each pair of news articles, containing the query article and a relevant article, a feature vector is extracted.

4. Extracted feature vectors that belong to pairs of articles are inserted in the data base.

5. Feature vectors that belong to pairs of articles that include the query article and relevant articles are sorted by the relatedness score, generated by a relatedness model (section 3.7).

Likewise, the summary of what happens during recommendation is as follows:

1. The article that a user is currently reading is assumed to be indexed and is passed to the recommender service.

2. The article is passed to Lucene as the query article and a number of relevant articles from current state of the system are retrieved. Returned articles are not necessarily the same as what had been returned during indexing, because other articles may have been added in the meanwhile.

3. Retrieved documents are compared with the articles that were found previously for the same query article which are now in the data base.

4. For each article not paired with the query article, the new feature vector is calculated and relatedness score is fetched from the model.

5. The data base table that contains the feature vectors and scores is updated and articles sorted by relatedness score to the query article are returned. The table acts as a cache to reduce number of times the feature extraction is done.

4.1.1 Architecture

The implementation of the system should be adaptable to different changes including adding or removing features and modification of layers. It must also generate recommended results rapidly. So the binary code itself must not produce any bottleneck. Additionally, beside performance needs of the project, maintainability is a driving factor in implementing this project. In addition to external libraries and tools which can change in future, domain specific features can be added later to the system like news topics and tags.

For implementation, we used JavaEE technology. As mentioned before, we used Hibernate as our ORM for the data layer. The entire system is a Maven[2] project which provides an easy to use dependency management. To manage layers of our web application, we used Spring MVC[3] framework. In web application development Model-View-Controller pattern is different than traditional publisher-subscriber pattern and is equivalent to a three layer application stack. In traditional MVC, controller

[2]http://maven.apache.org/
[3]http://spring.io/

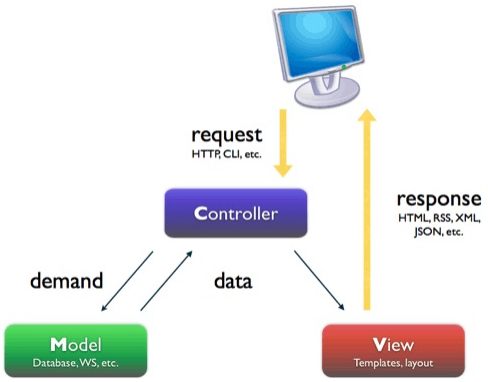

Figure 4.3: A high-level view of the MVC model.

handles the input and updates the model, then model informs view about changes that happened, so the view can update itself accordingly. In web MVC, the view is at client side, so there is no way to see the changes of model all the time. Instead, controller gets the input, updates the model, and informs view about changes. In this way, view and controller are in top layer, and business and data layers come afterward. This is equivalent to our layered design of the system with the difference that there is no view for this application.

This application has many external dependencies. The external components used in this system are shown in table 4.1. All of these components were subject to changes many times in the development phase. They can also be changed during production. For example changing underlying database system can happen at any time. To handle external dependencies, we used Spring dependency injection. It allows us to configure all these dependencies through external configuration file and let the application work by only an interface representing the functionality of these components. The concrete implementation of the components are linked to the application dynamically based on the configuration file.

4.1.2 Performance and bottleneck

The response time of the system during indexing and recommendation depends on many factors. During indexing, we need to persist document on the secondary storage device of the system (usually a hard drive) using Lucene. After that, we need to access database in order to retrieve, modify, or store new feature vectors. Since in both of

External Component	Used system
Data Base	MySQL
Regression Model	Weka
Learning to Rank	RankLib
Indexing and statistical text analysis	Lucene
Semantic Similarity	Word2Vec
Named entity recognition and disambiguation	Tulip Web Service

Table 4.1: List of external dependencies of the application

these tasks, we are accessing to the slowest component of the system, minimizing number of accesses will improve overall performance of the system. The other slow component of the system, is network access to services like Tulip which is performed only once per document during indexing. In case of hard disk access of Lucene and database system, we can use their internal caching system which depends on their configuration.

Regarding the implementation, the right choice of collections have a huge effect on the overall performance. In Java, there are various implementations for Set and Map data structures. We used Trove[4] collection which is a high performance implementation of Java collections and has better performance in terms of memory and speed.

To find the bottlenecks of the system other than the mentioned components, we did a simple performance test for execution of two scenarios and used a simple JVM profiling tool called Visual VM[5]. In Visual VM, we can see the amount of time the process is spending in each method. Figure 4.4 represent a sample output from VisualVM for recommendation task. In this example, the process spend most of its time inside method *handleCache* which is in *Recommender* class. It simply depicts hot spots in the application which gives us a clue about where to begin optimization. The task of optimizing the application is not a part of this thesis. But having an insight about hot spots can shed light into the future work directions.

During profiling, it became clear that the application is spending most of its time in Lucene related function. There are huge number of invocations for Lucene methods as keeping cache up-to-date requires finding similar articles for every single query. In

[4]http://trove.starlight-systems.com/
[5]http://visualvm.Java.net/

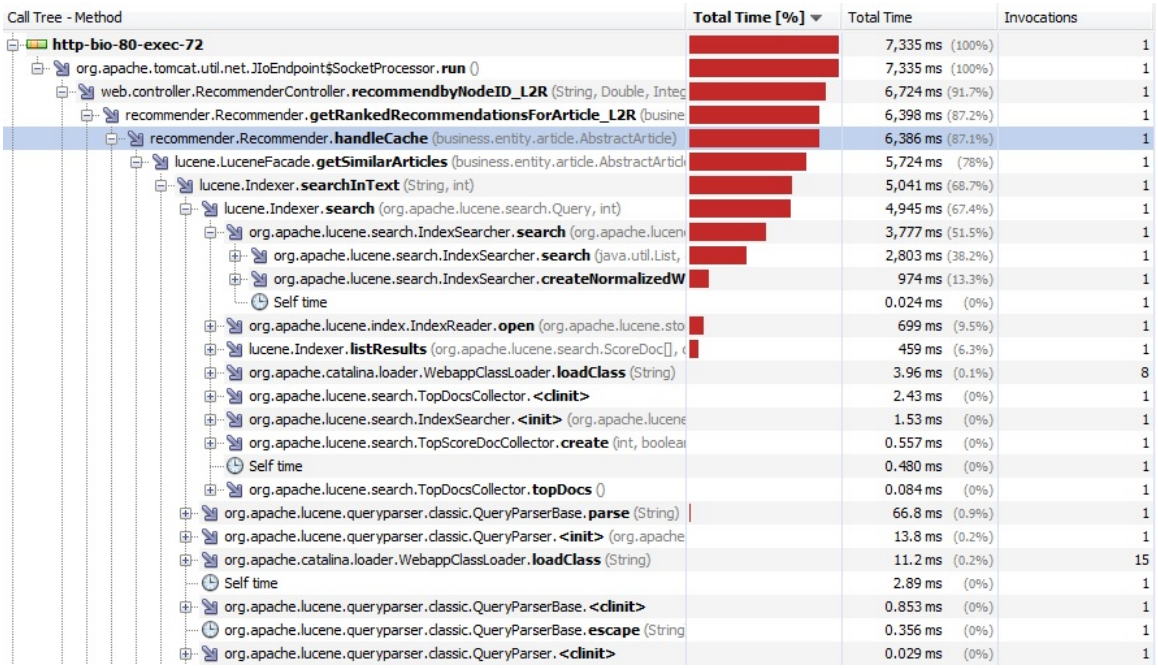

Figure 4.4: An example of profiling. The most time consuming method is cache handling.

figure 4.5, there are useful information regarding number of times a method is called and how much time is spent in that method sorted by their time consumption. As it shows, the performance of the application is heavily based on method in Lucene package.

The scalability of the architecture allows us to run the application on more than one hardware instance and get better response time. The application can be replicated on more than one server and the only shared point of the replicas would be Lucene index and database. Lucene index can be replaced with Solr to support horizontal scalability which can distribute hotspots on different machines. Database layer which uses standard RDBMS like MySQL can be horizontally scaled too since all major RDBMSs support this feature. One of the directions of future work is to find a better way to perform tasks related to these bottlenecks and eliminate them by modifying the algorithms or using parallelism techniques.

Hot Spots - Method	Self Time [%]	Self Time	Total Time	Invocations
org.apache.lucene.store.ByteBufferIndexInput.**readByte** ()		1,972 ms (26.9%)	1,972 ms	10,688,949
org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.**run** ()		609 ms (8.3%)	7,335 ms	1
org.apache.lucene.search.TermScorer.**docID** ()		550 ms (7.5%)	550 ms	80,071,137
org.apache.lucene.search.DisjunctionScorer.**heapAdjust** (int)		462 ms (6.3%)	908 ms	4,950,968
org.apache.catalina.loader.WebappClassLoader.**findResourceInternal** (Strin...		301 ms (4.1%)	427 ms	1,001
com.mysql.jdbc.util.ReadAheadInputStream.**fill** (int)		253 ms (3.5%)	253 ms	5,495
org.apache.lucene.store.ByteBufferIndexInput.**readInt** ()		176 ms (2.4%)	176 ms	405
org.apache.lucene.search.DisjunctionSumScorer.**countMatches** (int)		172 ms (2.4%)	424 ms	9,891,280
org.apache.lucene.store.DataInput.**readVInt** ()		145 ms (2%)	2,097 ms	9,616,691
org.apache.catalina.loader.WebappClassLoader.**findClassInternal** (String)		138 ms (1.9%)	572 ms	1,001
org.apache.lucene.store.ByteBufferIndexInput.**readBytes** (byte[], int, int)		121 ms (1.7%)	121 ms	560,305
org.apache.catalina.loader.WebappClassLoader.**loadClass** (String, boolean)		118 ms (1.6%)	692 ms	1,032
org.apache.lucene.store.ByteBufferIndexInput.**readLong** ()		102 ms (1.4%)	102 ms	518
org.apache.lucene.codecs.lucene3x.SegmentTermEnum.**next** ()		64.6 ms (0.9%)	476 ms	558,253
org.apache.lucene.search.TermScorer.**score** ()		64.2 ms (0.9%)	185 ms	4,945,640
org.apache.lucene.codecs.lucene3x.SegmentTermDocs.**next** ()		56.9 ms (0.8%)	1,398 ms	4,949,184
org.apache.catalina.loader.WebappClassLoader.**getURI** (java.io.File)		50.5 ms (0.7%)	50.5 ms	668
org.apache.lucene.codecs.lucene3x.TermBuffer.**read** (org.apache.lucene.store...		48.4 ms (0.7%)	273 ms	558,233
org.apache.lucene.store.DataInput.**readVLong** ()		42.2 ms (0.6%)	67.1 ms	1,144,799
org.apache.lucene.store.FSDirectory.**listAll** (java.io.File)		41.5 ms (0.6%)	42.2 ms	1
com.mysql.jdbc.Util.**handleNewInstance** (java.lang.reflect.Constructor, Obje...		41.5 ms (0.6%)	310 ms	4,222
org.apache.naming.resources.FileDirContext.**file** (String)		39.2 ms (0.5%)	39.3 ms	1,007
org.apache.lucene.codecs.lucene3x.Lucene3xNormsProducer$NormsDocValues$1.(		38.3 ms (0.5%)	38.3 ms	4,945,627
com.mysql.jdbc.MysqlIO.**send** (com.mysql.jdbc.Buffer, int)		34.2 ms (0.5%)	35.9 ms	2,703
org.apache.lucene.codecs.lucene3x.Lucene3xFields$PreDocsEnum.**freq** ()		32.3 ms (0.4%)	32.3 ms	4,945,627
org.apache.lucene.search.similarities.DefaultSimilarity.**tf** (float)		30.1 ms (0.4%)	30.1 ms	4,945,640
org.apache.lucene.util.WeakIdentityMap.**put** (Object, Object)		27.2 ms (0.4%)	35.0 ms	22,918
org.apache.lucene.search.similarities.DefaultSimilarity.**decodeNormValue** (long)		26.4 ms (0.4%)	26.4 ms	4,945,640

Figure 4.5: Hot Spots of application.

4.1.3 Load Test

We used a load testing tool called Grinder[6] to find out the responsiveness of the system during heavy traffic. We assumed that all the articles had already been indexed and users are only requesting recommended results. This will involve Lucene and database and requires high level of concurrency.

Grinder provides users a scenario script written in Python which contains the name of a URL, the expected response from the server and following actions. Then the number of threads or processes that are going to send request to server will be set. This tool is heavily used to understand the behavior of web application under heavy loads and if we combine it with a profiling tools, valuable information about bottlenecks can be understood.

As our experiments indicated, the maximum number of handled requests by the recommender was limited by the performance of Lucene as a filter. under normal load (100 requests per second), the system was able to respond to 30 requests per second on its peak. On average, each recommendation query will take 2 seconds assuming that the server is not fully loaded. In these experiments, the advanced caching capabilities

[6]http://grinder.sourceforge.net/

of Hibernate was not enabled. One of the directions of the future works will be using more advanced solutions for retrieval like Solr and compare its performance to the current system.

Chapter 5

Experiments and Results

In this chapter, all the offline and online experiments on the proposed system are discussed. To measure the performance, we relied on well-known performance criteria of recommender systems. For each of the datasets, we did two different experiments: the first was based on explicit labels of the relatedness in the dataset for the purpose of measuring ability of the system to recommend such news pairs, while in the second set of experiments, we focused on our proposed graph model of relatedness to measure its ability to find all related news in a story thread. The baseline in all of the experiments is Lucene which is the standard search engine used in industry.

5.1 Datasets

To perform our experiments, we relied on two datasets. The first dataset was constructed from a set of news articles belonging to "The Chronicle Herald" news paper. it contains 116,384 news articles for a period of 4 years. Out of all news articles, there are 1472 pairs of news articles that are tagged as related news by web editors. One problem with this labeling is that the pairs are tagged in a directional way. It means that there are situations in which A is related to B but the opposite direction is not labeled. We consider all pairs as non-directional. Before separating the set into training and testing subsets, we made sure that an article will never appear in both sets. It means if A is related to B and B is related to C, then both of these pairs have to appear in either training or testing set and not in both. Considering all pairs of documents, we created a query-document style dataset in which for each query article, there is a set of related articles. We divided training and testings subsets by 60-40 ratio which resulted in 889 training query articles and 583 testing query articles.The average related document per query is 1.86.

The second dataset that was used in the following experiments was extracted from Wikinews. Like the first dataset, editors labeled a tiny fraction of documents as

related pairs. The dataset contains 16723 news articles and only 2035 articles have at least one related news article. We followed the same procedure to build training and testing subsets. The training set contains 1216 query articles and the test set contains 819 query articles. The average number of related document per query is 1.87.

5.2 Main Measurement Criteria

In this set of experiments, we used *Precision*, *Recall*, and *F1* to measure performance of the system against the gold standard. To measure precision of the system, we used the following formula:

$$P_q = \frac{|r_q \cap g_q|}{|r_q|} \tag{5.1}$$

where r_q is the set of returned articles by the system and g_q is the set of articles that must be returned according to gold standard.

We calculate recall for a given query article by the following formula:

$$R_q = \frac{|r_q \cap g_q|}{|g_q|} \tag{5.2}$$

$F1$ measure is the harmonic mean of precision and recall and gives us an overall performance of the system and the retrieval power of it:

$$F1_q = \frac{R_q \cdot P_q \cdot 2}{R_q + P_q} \tag{5.3}$$

All these measures are used to measure performance of the system on immediate related news articles. In other words, the gold standard is always the binary relatedness flags as they are represented in the actual dataset unless it is specified differently.

5.3 Feature Selection Results

As discussed in subsection 3.7.1, we conducted a feature importance ranking based on three different feature selection algorithms. In all three experiments, time difference attribute, Jaccard index of named entities, and information based score are getting

high ranks. However using LDA gets very low rank. We conjecture that LDA assigns high similarity score to a pair of documents which are only related by sharing a high level topic like politics.

The topical keywords feature is doing well among lexical features. It stands on top of BM25 and cosine similarity based on CFS (discussed in subsection 3.7.1). According to Information Gain and RELIEF, this feature also performs better than language modeling. Therefore, it is one of the most important lexical features.

To the best of our knowledge, there is no work in news recommendation that used Information Based similarity as a feature. According to the results from our experiments, this feature gets very high rank compared to other lexical features.

Except Jaccard index of named entity sets of two documents, other features that rely on named entities get middle-class ranking scores. There is a close competition between skip-gram similarity and Sunflower similarity features groups as based on what method is used for feature selection, one method seems to be better than the other one. However, if we consider speed of the system during feature extraction, skip-gram model is a clear winner since it only requires a cosine similarity. In terms of the ability of the feature to model relatedness, the best way to find out which one performs better was to train a regression model with and without these features and select the one which produces the best results. For this purpose, we trained and tested a simple regression model by REPTree algorithm in Weka package. In these experiments, we used REPTree for our regression models with a cut-off threshold of 0.8 which was obtained through trial and error.

The results of testing regression models that were trained with different feature sets are demonstrated in table 5.3. As these results show, using LDA decreases the performance of the system. It is so low rank in the regression tree that removing it barely changes the precision and recall of the system (first and second rows). The third row of the table represents results without LDA and two less important features calculated by Sunflower. As it shows, the results are improved by removing those features. The fourth row is the system without using LDA and Sunflower at all and the results are almost the same as keeping two features from Sunflower. Considering the cost of calculating similarity by Sunflower, a system with 13 features is more preferable than 15 features. The last row is a system that uses only Sunflower instead

CFS	Information Gain	RELIEF
TimeSpan	TimeSpan	TimeSpan
IB	JaccardTitle	IB
JaccardSim	JaccardSim	JaccardSim
SunflowerAvgSim	IB	SunflowerAvgSim
SunflowerSumOfMax	SunflowerMax	JaccardTitle
JaccardTitle	SunflowerAvgSim	SkipgramMinSim
SkipgramAvgSim	SunflowerSumOfMax	Keywords
SkipgramSumOfMax	Keywords	sunFlowerMin
JaccardBody	LM	JaccardBody
LM	SkipgramMinSim	LM
LDASim	SkipgramMaxSim	BM25
Keywords	SkipgramSumOfMax	CosineTF
BM25	SkipgramAvgSim	SkipgramMaxSim
SunflowerMax	sunFlowerMin	SunflowerMax
CosineTF	CosineTF	SunflowerSumOfMax
SkipgramMaxSim	BM25	SkipgramAvgSim
SkipgramMinSim	JaccardBody	SkipgramSumOfMax
sunFlowerMin	LDASim	LDASim

Table 5.1: Ranked list of features extracted from three feature selection algorithms.

13 Selected Features
TimeSpan
IB
JaccardSim
JaccardTitle
SkipgramAvgSim
SkipgramSumOfMax
SkipgramMaxSim
SkipgramMinSim
JaccardBody
LM
Keywords
BM25
CosineTF

Table 5.2: Selected features for modeling relatedness.

Feature Set	Precision	Recall	F1 Score
All features (18 features)	0.473	**0.713**	0.569
No LDA (17 features)	0.475	0.708	0.569
No LDA, Min and Max Sunflower (15 features)	0.493	0.705	**0.580**
No LDA and Sunflower (13 features)	**0.499**	0.693	**0.580**
Sunflower instead of skip-gram (13 features)	0.460	0.683	0.550

Table 5.3: Effect of Features in Regression Performance. None of the numbers in this experiment were statistically significant than others. However due to the performance of Skipgram model and Occam's razor principle, we decided to select only 13 features without considering Sunflower and LDA.

of skip-gram model and it clearly indicates that skip-gram gives us better similarity scores for news recommendation task.

5.4 Selecting Best Data Model

In our first experiment, we wanted to select the best data model for training our regression models. As we discussed in subsection 3.6 we had three models for labeling negative data samples. In the first data model, we selected negative examples randomly. In the second model, we selected them based on the Lucene output as a filter, and in the third model we used the graph of relatedness to model partial relatedness scores. We call these models *Random*, *LuceneFilter*, and *GraphData* respectively. We also compared these models to a baseline Lucene search engine which we call *lucene* in our result tables. Since the purpose of this experiment is to select one data model against others, all of the regression models were created using REPTrees with all 13 features selected in previous section.

We report the macro average of the discussed criteria (see subsection 5.2) in this experiment, shown in tables 5.4 and 5.5. In these experiments, we used REPTree for our regression models with a cut-off threshold of 0.8. As these results indicate, the *GraphData* gives us the best negative sampling of data. The training correlation coefficient that is reported in the result tables indicates a very important behavior of the data. When we are dealing with randomly sampled negative instances, the high training correlation coefficient indicates a separable dataset which does not need a very complicated model. In other words, a simple regression tree performs very well in separating positive and negative examples. However, the randomly selected

Model name	Training Correlation Coefficient	Macro Precision	Macro Recall	Macro F1
GraphData	0.789	**0.499**	0.693	**0.580**
Random	**0.917**	0.088	**0.804**	0.159
LuceneFilter	0.714	0.455	0.433	0.443
Lucene(best f1: k = 3)	N/A	0.339	0.453	0.388

Table 5.4: Comparing different data labeling methods on "The Chronicle Herald" dataset. Training correlation coefficient shows the performance of the system on training set while other columns represent the results on test data.

Model name	Training Correlation Coefficient	Macro Precision	Macro Recall	Macro F1
GraphData	0.652	**0.373**	0.589	**0.457**
Random	**0.911**	0.200	**0.775**	0.318
LuceneFilter	0.512	0.366	0.327	0.345
Lucene(best f1: k = 3)	N/A	0.348	0.645	0.452

Table 5.5: Comparing different data labeling methods on "Wikinews" dataset. Training correlation coefficient shows the performance of the system on training set while other columns represent the results on test data.

negative data points do not correspond with actual setting of the problem since the regression model is supposed to re-rank Lucene's output. The lower value of the training correlation coefficient in other data models represents a very complex and non-separable feature space.

5.5 Comparing Different Regression Models

In our second experiment, we wanted to compare performance of different Regression tree models trained on our selected data model which is Graph data (based on the experiment in previous section 5.4). We also measured the effect of our keyword

Model name	Macro Precision	Macro Recall	Macro F1
REPTree (13 features)	0.499	0.693	0.580
GBRT (13 features)	0.471	0.721	0.570
GBRT (12 features)	0.471	**0.723**	0.570
REPTree (12 features)	**0.530**	0.672	**0.593**
Lucene(best f1: k = 3)	0.339	0.453	0.388

Table 5.6: Offline results of four models on "The Chronicle Herald" dataset. Bold values represent the best results in a column. The red numbers indicate those that are not statistically significant from the bold ones. Baseline method is below double lines.

Model name	Macro Precision	Macro Recall	Macro F1
REPTree (13 features)	0.373	0.589	0.457
GBRT (13 features)	**0.553**	**0.768**	**0.643**
GBRT (12 features)	0.532	0.759	0.626
REPTree (12 features)	0.374	0.595	0.459
Lucene(best f1: k = 3)	0.348	0.645	0.452

Table 5.7: Offline results of four models on "Wikinews" dataset. Bold values represent the best results in a column. The red numbers indicate those that are not statistically significant from the bold ones. Baseline method is below double lines.

extraction feature by comparing a model trained without that feature. In this experiment, we compared two different regression tree approaches: the first is using REPTree and the second is using Gradient Boosted Regression Tree (GBRT) based on CART trees as weak trees (details in section 3.7). We compared these two models against baseline which was Lucene search engine. These models were trained with 13 selected features discussed in section 5.3. We removed the keyword extraction feature and trained the REPTree model with 12 features to measure the effect of this particular feature. Our experiment on two datasets based on precision, recall, and F1 measures is summarized in Tables 5.6 and 5.7. For REPTree, the cut-off values was 0.8. For the GBRT method, the cut-off was set to two steps away form query article.

Based on the results from this experiment, gradient boosted regression trees improved the results in Wikinews dataset to a large extent while on The Chronicle Herald dataset, it almost produced the same results as REPTree. According to Maclin and Opitz [1999] this result suggests that there is a noticeable amount of noise in "The Chronicle Herald" dataset which is not surprising as there are many data points, that are close to each other but with different feature values. In other words, the data

points in the feature space is not easily separable. As mentioned earlier, this dataset is not accurate and is created by minimum effort of editors given a limited time frame which leads to having pairs of articles that are labeled as related while one can find much more related articles instead of the labeled ones. On the other hand, Wikinews dataset is created by many users which virtually means having tens or hundreds of editors per article. This reduces the noise dramatically by assigning more related news articles to one article and allows boosting algorithms improve the results.

The effect of keyword extraction feature was also studied by training models with and without it. In this experiment, according to the gold standard, the decision tree without keyword extraction feature outperforms the one with the keyword extraction feature. However, when using gradient boosted trees, including keyword extraction slightly improves the performance of the system. The intuition behind this feature was to capture topical similarity only based on shared keywords. This feature is making very small impact in The Chronicle Herald dataset. But it improves precision in Wikinews dataset by a large degree. A potential reason is that Wikinews covers broader range of topics and this feature brings news articles within the same category. In other words, this experiment shows the power of this feature for narrowing down the search space when there is a large set of different categories.

5.6 Ranking Results

As we discussed in sub-section 3.6.3, we can see a connected component of the relatedness graph as a thread of a news story. Considering the scoring strategy that we used to model partial relatedness which improved overall accuracy of the system, we can also measure the system as a ranking system to score related articles against each other based on their distance in the graph. In other words, if we consider an article as a query article, then the system should return all the news of the same story thread as related news articles, sorted by their distance in the graph. This is a different application for the system, but it is closely related to the topic.

To measure performance of the system for the ranking scenario, we used *Mean Average Precision* or *MAP*, *Normalized Discounted Cumulative Gain* or *NDCG*, and precision. In contrast to typical information retrieval evaluation scenarios in which human judgment is used as ground truth, we used the distance in the relatedness

graph as the ranking gold standard. In our experiments, given a query document, the ranking of each of the recommender articles is calculated relative to the distance of the query article and recommended article in the graph representation of relatedness. So a recommended article that is the neighbor of the query article gets the highest rank.

Precision and recall are single-value metrics based on the whole list of documents returned by the system. When we have a ranking system, it makes sense to consider order of the returned documents in the computation of precision or recall. By computing the precision and recall at every position of the ranked list of document, we can plot the precision-recall curve. Average precision is the area under the precision-recall curve from $recall = 0$ to $recall = 1$. Instead of going through all the returned results, we can always limit the number of considered articles to any number. Mean average precision is the mean of all average precision values for all query articles.

Discounted cumulative gain[1] is a measure of ranking quality. In information retrieval, it is used to measure the quality of the returned results by a search engine or any retrieval system. DCG measures the usefulness or gain of a document based on its position in the result list. In other words, DCG assumes more relevant documents must be placed higher than less relevant documents. DCG of a query is calculated using the following formula:

$$DCG_n = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{\log_2 i} \tag{5.4}$$

where rel_i represents relatedness score of the result in rank i.

The normalization factor for Normalized DCG is the optimal or ideal setting of the list of retrieved documents. Since we used graph distance values as relatedness scores, we use them as the ideal values for ranking. So given a ranking, we referred to the gold standard (from graph) for the actual ranking score of an element in the list. For example, for a query document, we get d_1, d_2, and d_3 in the results. For each of them, we calculate its distance to the query article. In this example, we get 2, 0, and 6 respectively. We sort these scores in descending order and calculate its DCG. The resulting DCG value is used as the normalization factor.

We also report the precision of the returned list of articles by counting either first

[1] http://en.wikipedia.org/wiki/Discounted_cumulative_gain

3 or first 5 results. In this experiment, we consider all of the news articles in the same story as relevant. Therefore, it is different than the precision scores of the previous experiments which were only based on explicit labels of related articles.

In this experiment, we measured the performance of our models from previous experiments against each other and baseline. In addition, we did the experiment using a few well-known learning to rank algorithms from other learning paradigms (pairwise and listwise) using RankLib package. The summary of the results are in tables 5.8 and 5.9. As these results suggest, the regression models, which are considered as pointwise algorithms in the learning to rank area, are returning the best results.

According to these results, the keyword extraction feature slightly improves the precision of the system in finding related news in the same story. This feature finds similar articles inside a story thread with very small cost of losing a little accuracy on the explicit relatedness of news articles. It also performs much better on "The Chronicle Herald" noisy dataset.

The baseline of this experiment performs well when we measure precision at first ranked result (P@1), specially on Wikinews dataset. It simply means that most of the times, the lexical similarity offered by Lucene can identify the very similar article to the query document. However, other systems outperform baseline when we look at other results in the ranked list. The proposed features are doing well in re-ranking results based on their relatedness score.

On "The Chronicle Herald" dataset, REPTree outperforms other methods in most of the cases. As discussed earlier, this dataset is a noisy dataset. So boosting methods like GBRT do not perform well in noisy situation. This method yields the highest MAP value which means that most of the related news articles in a story thread are ranked on top of the ranked list. This can also be seen by looking at precision results. The keywords extraction feature also plays a very important role when we compare MAP values between models. This experiment demonstrates the importance of this feature in finding related articles in the story thread.

On Wikinews dataset, REPTree is slightly better than baseline. However, GBRT method outperforms both baseline and REPTree by a large margin. It is mainly because that this dataset is not noisy, therefore we see a jump in the results of GBRT as an boosting method. Like previous dataset, the keyword extraction feature

Model Name	NDCG@5	MAP@5	P@1	P@3	P@5
REPTree (13 features)	0.736	**0.596**	**0.671**	**0.570**	**0.546**
GBRT (No threshold)	0.751	0.243	0.628	0.395	0.290
GBRT (13 features, th = 2)	0.724	0.566	0.612	0.540	0.521
GBRT (12 features, th = 2)	0.730	0.478	0.624	0.541	0.520
REPTree(12 features)	0.733	0.499	0.659	0.560	0.532
Lucene(no threshold)	0.614	0.175	0.484	0.296	0.223
LambdaMART	**0.767**	0.24	0.659	0.400	0.287
Coordinate Ascent	0.615	0.185	0.484	0.310	0.234
RankBoost	0.530	0.148	0.397	0.252	0.196

Table 5.8: Ranking results of different models on "The Chronicle Herald" dataset. The performance of ranking is measured using NDCG, Mean Average Precision (MAP) and precision, considering up to 5 top returned results by the system.
Bold values represent the best results in a column. The red numbers indicate those that are not statistically significant from the bold ones. Baseline methods are below double lines.

Model name	NDCG@5	MAP@5	P@1	P@3	P@5
REPTree (13 features)	0.623	0.426	0.531	0.482	0.467
GBRT (No threshold)	**0.843**	0.355	**0.817**	0.512	0.374
GBRT (13 features, th = 2)	0.835	**0.677**	0.810	**0.719**	**0.695**
GBRT (12 features, th = 2)	0.826	0.656	0.799	0.698	0.674
REPTree (12 features)	0.626	0.418	0.530	0.475	0.46
Lucene(No threshold)	0.770	0.306	0.718	0.446	0.337
LambdaMART	0.786	0.320	0.730	0.470	0.356
Coordinate Ascent	0.791	0.321	0.730	0.472	0.353
RankBoost	0.739	0.295	0.656	0.440	0.335

Table 5.9: Ranking results of different models on "Wikinews" dataset. The performance of ranking is measured using NDCG, Mean Average Precision (MAP) and precision, considering up to 5 top returned results by the system. Bold values represent the best results in a column. The red numbers indicate those that are not statistically significant from the bold ones. Baseline methods are below double lines.

Model name	NDCG@5	MAP@5	P@1	P@3	P@5
REPTree(13 features)	0.531	**0.294**	0.389	**0.362**	**0.348**
Baseline	**0.614**	0.175	**0.484**	0.296	0.223

Table 5.10: Results on "The Chronicle Herald" given a model trained on Wikinews.

Model name	NDCG@5	MAP@5	P@1	P@3	P@5
REPTree(13 features)	0.552	0.265	0.427	0.353	0.318
Baseline	**0.770**	**0.306**	**0.718**	**0.446**	**0.337**

Table 5.11: Results on Wikinews given a model trained on "The Chronicle Herald".

improves the ability of the system in finding related news articles in a story thread.

As the results of this experiment show, the learning to rank methods other than typical regression methods are not performing well. The main reason is that the similarity features that are used in this work are producing values regardless of the local relatedness of news articles while other listwise or pairwise (see section 3.7.3) methods require features that represent relatedness considering the entire list of related articles (in listwise method) or a list of relative constraints (in pairwise methods). It is arguable that we could transform our dataset to a pairwise constraint and produce features in that manner. This can be investigated later as a future work.

5.7 Universal News Relatedness Model

In this experiment, we wanted to see if we can train a model once and use it in different situations. Since we had only two datasets, we trained our model on one dataset and tested it on the other one. The results of this experiment are represented in tables 5.10 and 5.11.

At the first glance, we see that the model trained on The Chronicle Herald is not performing well on the Wikinews. The main reason behind it is that Wikinews covers a broader range of news topics and domains while The Chronicle Herald is only a local newspaper covering news mainly about Nova Scotia. On the other hand, The Chronicle Herald is focused on Nova Scotia, so there are plenty of words and named entities which belong to this region. This diminishes the power of the model trained on Wikinews to produce meaningful scores for The Chronicle Herald articles. Although the results show that the model trained on Wikinews is slightly better than

Model name	NDCG@5	MAP@5	P@5
REPTree (13 features)	**0.950**	**0.882**	**0.904**
GBRT (13 features)	0.951	0.875	0.888
REPTree (12 features)	0.899	0.755	0.797
Lucene	0.830	0.768	0.786

Table 5.12: Experts feedback on live system in "The Chronicle Herald". Bold values represent the best results in a column. The red numbers indicate those that are not statistically significant from the bold ones. Lucene was the baseline.

the baseline, this experiment shows that our features are domain dependent. So when we are dealing with regional news, we can not use a model trained on global news.

5.8 Expert Preferences

We made the system available for the editors of "The Chronicle Herald" news paper for a limited time frame. In total, 7 editors participated in this experiment and provided more than 600 binary decisions on whether a recommended result is good or bad. Each editor was able to see 5 recommended results for a given article of their choice. We used their binary judgments to calculate NDCG, MAP and Precision of the system. Like our first experiment, we tested 4 systems against each other. In this test, for a query submitted by each editor, the system selected one of the recommenders randomly and provided related articles as indicated in table 5.12. As these results demonstrate, if the system is being used as a decision support system to help editors with selecting recommended articles, then REPTree with 13 features and GBRT is outperforming Lucene and REPTree with 12 features. These results tell us how the proposed systems are being preferred over Lucene which is the most widely used search engine. The other conclusion from this experiment is the importance of the keyword extraction feature which was not included in the REPTree with 12 features. This feature appears to be very effective in finding related news since the models trained with 13 features are outperforming the one with 12 features. The strength of the keyword feature is that it let the articles about the main topic of the query article get higher ranks. This experiment shows that news editors like this property of the recommender system.

Model Name	NDCG@5	NDCG@3	MAP@5	MAP@3	P@5	P@3
GBRT (12 features)	0.874	0.789	0.599	0.687	0.647	0.447
GBRT (13 features)	**0.892**	**0.862**	**0.644**	**0.766**	**0.679**	**0.496**
Lucene	**0.892**	0.854	0.570	0.685	0.612	0.462

Table 5.13: News recommendation software evaluation using Wikinews dataset. Lucene was the baseline.

5.9 News Recommender Software Evaluation

We asked 15 students to use a web application that uses Wikinews as repository and provide binary feedback about relatedness of recommended news articles, same as the scenario of our experiment with editors in The Chronicle Herald. Each student had 25 news articles about various topics, which were selected randomly from Wikinews test dataset. By selecting each of the 25 articles, a user gets 5 recommended articles which were produced by one of the test systems. The user provided one binary judgment about if a recommended article is related to the selected news article or not. Only title and first 500 words of each article were showed for each article. We asked the participants to skim the news article and make a decision in less than one minute. Therefor, the users made decision based on very little information.

The goal of this experiment was to measure the performance of the proposed models against Lucene baseline and the power of our keyword extraction feature. Thus, there are 3 models in this experiment: Lucene as baseline, GBRT with 12 features and GBRT with an additional keyword extraction feature. As results show, the proposed models outperform the baseline in Mean Average Precision. GBRT with 12 features performs worse than the baseline when considering top 3 results considering precision and NDCG. It means that this method is providing related articles in later positions of the returned ranked lists. Using the keyword extraction feature re-ranks the results and puts more related ones on top of the list (table 5.13).

In this test, students were asked to skim the articles and decide based on what they think about relatedness. This kind of experiment is different from the experiment done with the experts. An editor understanding of relatedness is more accurate and more well-defined than an ordinary person. Thus, there is a higher variance for returned values when we analyzed subjective data from students (figure 5.1).

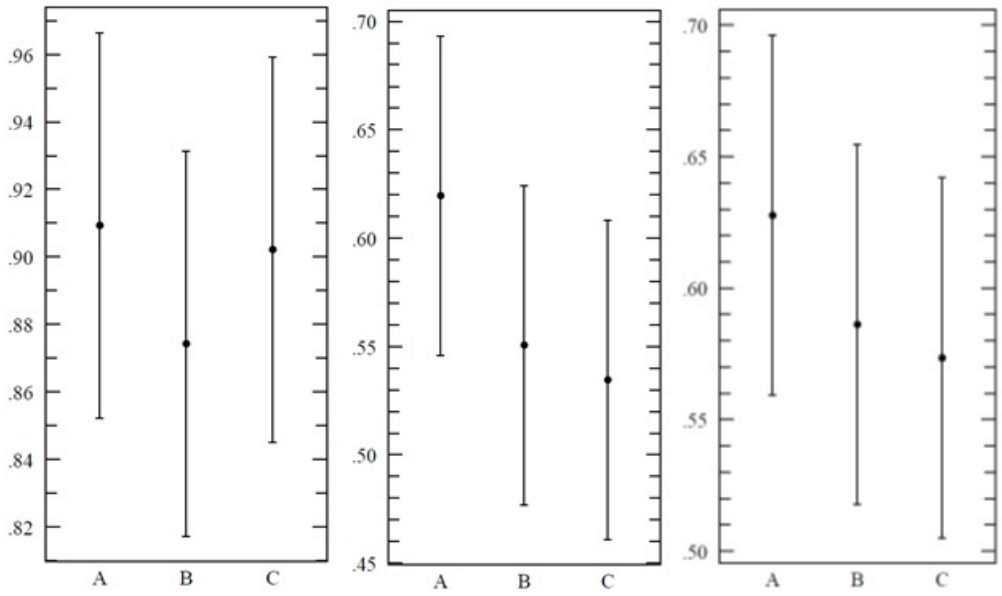

Figure 5.1: ANOVA test results on students' feedback. The test was subjective, which introduces a wide range of values that contributes to the high variance of results. This is very different from experts' feedback result which was more objective.

Chapter 6

Conclusion

In this thesis, we explored an approach based on named entities to address the problem of post-click news recommendation. Like previous work, we combined various distinct similarity measures between two articles to find a relatedness function using a regression model. Our main contribution is the use of named entities and their semantic similarity in addition to traditional lexical similarity measures. We studied the effect of named entities and demonstrated that simple Jaccard index of two sets of named entities is a very important feature. We compared two similarity measures based on two different paradigms: skip-gram vs. Wikipedia categories. The winner was skip-gram as categories represent broader level of relatedness which is not sufficiently precise in news domain. We also showed that using LDA similarity is not suitable to capture semantic similarity of articles in the specific task of news recommendation.

Named entities were extracted from the body of news articles using the award winning system Tulip. To the best of our knowledge, this is the fastest and most accurate system for extracting named entities out of text available. Then we used the skip-gram model, a state-of-the-art similarity measure, to calculate similarity of named entities. This is the first time Tulip and skip-gram model are being used together and the results can be produced very fast.

We introduced a stochastic keyword extraction process which sits in a middle ground between topic modeling and lexical similarity in order to fill the gap between topical similarity and traditional lexical similarity. This process extracts the keywords that represent the core topical keywords of a text. Then a cosine similarity measure was used on those keywords to find similar documents. The cosine similarity calculated using extracted keywords formed a new similarity feature in the regression feature set. Although this proposed feature is stochastic and does not produce exactly the same results each time, we claim that this does not cause a problem for two

reasons: first, other lexical similarities in the regression feature set cover any possible mistakes by this method and second, other output keywords from this method are still able to represent the core topic of the text. Our experiments showed that this feature is doing very well when we are dealing with entire threads of news stories.

Due to the nature of the problem, it is challenging to define negative examples to create a dataset for training and evaluating any proposed solution. In previous work, the problem was tackled from an information retrieval point of view and only ranking of article was the issue. In this problem, we introduced similarity features to learn a relatedness model. Hence, it requires negative examples as well as positive examples of related news articles. To the best of our knowledge, there is no dataset containing both classes of examples. We studied various strategies to extract relatedness label of a news pair from a typical news dataset and compared them by comparing their final model. The most effective strategy was a graph-based method which considers the entire thread of news stories as related articles with scores as function of their relative distance.

We conducted experiments using traditional regression methods as well as learning to rank methods to find the best model that captures relatedness of two news articles. We measured the performance of the algorithms to find immediate related articles based on the gold standard and its ability to find related articles within the same story thread. The latter task suggested using more advanced learning to rank algorithms and measuring the models using ranking criteria such as MAP and NDCG. It turned out that a gradient boosted regression tree which is an boosting method with ordinary regression trees outperforms other models.

We reported the importance of each feature used in this study. To the best of our knowledge, there is no similar work in this domain that has used information based similarity, one of the well studied similarity function in the information retrieval. In all of our studies, this feature was one of the most important features. We showed that Jaccard index of named entities play an important role to capture relatedness. Also our keyword extraction algorithm was performing better than most of the lexical similarity functions. For those cases that the importance of features was not obvious from feature selection algorithms, due to getting different ranking by different algorithms, we trained and tested a regression model with different subsets

of features.

One of the directions of future research is testing various named entity recognition and disambiguation methods instead of Tulip that are able to annotate concepts. Concepts are usually general words with general meanings in the knowledge base and are much harder to disambiguate than named entities. In this way, there are going to be more entities (concepts) per article. The other direction is using other similarity measures such as Wikipedia links. Other measures are likely to be much slower, but they may provide better results. The response time of the system is always an issue, so optimizing or distributing the computation can be one of the future challenges.

The other direction of future work is creating a specific news recommender for each group of readers. Reader profiling based on their behavior is a well-studied area that can be directly used in combination with this system. The profiling can help providing better recommendation for each group by filtering their behavior from their first interaction with the web site.

The keyword extraction algorithm introduced in this work has the potential for further extension. The algorithm is stochastic, so the results are not guaranteed to be the same over multiple runs. This problem can be addressed by running the algorithm many times and producing a set of keywords which represent the consensus output of the system. To deal with the increasing cost of computation, it can be implemented in parallel such that its results will be merged since each run of the system is completely independent of the others.

Bibliography

Marko Balabanović and Yoav Shoham. Fab: Content-based, Collaborative Recommendation. *Commun. ACM*, 40(3):66–72, March 1997. ISSN 0001-0782. doi: 10.1145/245108.245124. URL http://doi.acm.org/10.1145/245108.245124.

David M. Blei and Peter I. Frazier. Distance Dependent Chinese Restaurant Processes. *J. Mach. Learn. Res.*, 12:2461–2488, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078184.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944937.

David M. Blei, Thomas L. Griffiths, and Michael I. Jordan. The Nested Chinese Restaurant Process and Bayesian Nonparametric Inference of Topic Hierarchies. *J. ACM*, 57(2):7:1–7:30, February 2010. ISSN 0004-5411. doi: 10.1145/1667053.1667056. URL http://doi.acm.org/10.1145/1667053.1667056.

Toine Bogers and Antal van den Bosch. Comparing and Evaluating Information Retrieval Algorithms for News Recommendation. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 141–144, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-730–8. doi: 10.1145/1297231.1297256. URL http://doi.acm.org/10.1145/1297231.1297256.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

Jethro Borsje, Leonard Levering, and Flavius Frasincar. Hermes: A Semantic Web-based News Decision Support System. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 2415–2420, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-753-7. doi: 10.1145/1363686.1364258. URL http://doi.acm.org/10.1145/1363686.1364258.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

Michel Capelle, Frederik Hogenboom, Alexander Hogenboom, and Flavius Frasincar. Semantic News Recommendation Using Wordnet and Bing Similarities. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 296–302, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480426. URL http://doi.acm.org/10.1145/2480362.2480426.

David Carmel, Ming-Wei Chang, Evgeniy Gabrilovich, Bo-June Paul Hsu, and Kuansan Wang. ERD'14: entity recognition and disambiguation challenge. In *ACM SIGIR Forum*, volume 48, pages 63–77. ACM, 2014.

Stéphane Clinchant and Eric Gaussier. Information-based Models for Ad Hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 234–241, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0153-4. doi: 10.1145/1835449.1835490. URL `http://doi.acm.org/10.1145/1835449.1835490`.

Mostafa Dehghani, Azadeh Shakery, Masoud Asadpour, and Arash Koushkestani. A learning approach for email conversation thread reconstruction. *J. Information Science*, 39(6):846–863, 2013. doi: 10.1177/0165551513494638. URL `http://dx.doi.org/10.1177/0165551513494638`.

Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research*, 4:933–969, 2003. URL `http://www.jmlr.org/papers/v4/freund03a.html`.

Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29:1189–1232, 2000.

Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, volume 7, pages 1606–1611, 2007.

Evgeniy Gabrilovich, Susan Dumais, and Eric Horvitz. Newsjunkie: providing personalized newsfeeds via analysis of information novelty. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 482–490, New York, NY, USA, 2004. ACM Press. ISBN 158113844X. doi: 10.1145/988672.988738. URL `http://dx.doi.org/10.1145/988672.988738`.

Frank Goossen, Wouter IJntema, Flavius Frasincar, Frederik Hogenboom, and Uzay Kaymak. News Personalization Using the CF-IDF Semantic Recommender. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, pages 10:1–10:12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0148-0. doi: 10.1145/1988688.1988701. URL `http://doi.acm.org/10.1145/1988688.1988701`.

Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics, 2011.

Lan Huang, David Milne, Eibe Frank, and Ian H. Witten. Learning a Concept-based Document Similarity Measure. *J. Am. Soc. Inf. Sci. Technol.*, 63(8):1593–1608, August 2012. ISSN 1532-2882. doi: 10.1002/asi.22689. URL http://dx.doi.org/10.1002/asi.22689.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

Claudia Leacock and Martin Chodorow. Combining local context and WordNet similarity for word sense identification. *WordNet: An electronic lexical database*, 49 (2):265–283, 1998.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. DBpedia–A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 2014.

Hang Li. A Short Introduction to Learning to Rank. *IEICE Transactions*, 94-D (10):1854–1862, 2011. URL http://dblp.uni-trier.de/db/journals/ieicet/ieicet94d.html#Li11.

Marek Lipczak, Arash Koushkestani, and Evangelos Milios. Tulip: Lightweight Entity Recognition and Disambiguation Using Wikipedia-based Topic Centroids. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation*, ERD '14, pages 31–36, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3023-7. doi: 10.1145/2633211.2634351. URL http://doi.acm.org/10.1145/2633211.2634351.

Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, 2011. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_3. URL http://dx.doi.org/10.1007/978-0-387-85820-3_3.

Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. Learning to Model Relatedness for News Recommendation. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 57–66, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0632-4. doi: 10.1145/1963405.1963417. URL http://doi.acm.org/10.1145/1963405.1963417.

Richard Maclin and David Opitz. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 1999.

Olena Medelyan, Ian H Witten, and David Milne. Topic indexing with Wikipedia. In *Proceedings of the AAAI WikiAI workshop*, volume 1, pages 19–24, 2008.

Olena Medelyan, Eibe Frank, and Ian H Witten. Human-competitive tagging using automatic keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1318–1327. Association for Computational Linguistics, 2009.

Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

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

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

David Milne and Ian H Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

Marnix Moerland, Frederik Hogenboom, Michel Capelle, and Flavius Frasincar. Semantics-based News Recommendation with SF-IDF+. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, WIMS '13, pages 22:1–22:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1850-1. doi: 10.1145/2479787.2479795. URL `http://doi.acm.org/10.1145/2479787.2479795`.

LBJFR Olshen, Charles J Stone, et al. Classification and regression trees. *Wadsworth International Group*, 93(99):101, 1984.

Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012-015*, 2012.

Michele Trevisiol, Luca Maria Aiello, Rossano Schifanella, and Alejandro Jaimes. Cold-start News Recommendation with Domain-dependent Browse Graph. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 81–88, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2668-1. doi: 10.1145/2645710.2645726. URL `http://doi.acm.org/10.1145/2645710.2645726`.

I Witten and David Milne. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceeding of AAAI Workshop on Wikipedia and*

Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA, pages 25–30, 2008.

Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *LREC*, volume 8, pages 1646–1652, 2008.

Appendix A

Skip-gram Model

The work by Mikolov et al. [2013] grabbed a lot of attention since they published it. They proposed using different neural network techniques and a new shallow learning approach (as opposed to deep learning) to create a language model. One of their ideas was using skip-gram model. In this model, each word is projected into a high dimensional space as a vector. The values of the vectors are determined by considering surrounding words. In this way, we assume that related words appear close to each other in a data set. Since this is a very fast method to train, it can be trained on very large text corpus, in a size about 1 billion news articles from Google News.

They demonstrated that the meaning and relationship between words are encoded spatially. In other words, their model provided a good semantic representation of words: similar words are closer to each other in terms of their distance (Figure A.1) and relationship between words are preserved by same vector direction (Figure A.2). So the cosine similarity of the vectors of two words can measure similarity of them from different dimensions where each dimension represent a hypothetical aspect of similarity between two words. This model provides a high quality and accurate vector representation of words, but there is no efficient and proven way to use it for document pair similarity. This is a very active field of research though. In this work, we used skip-gram model for two purposes: the first was to use it as a similarity function in our semantic similarity functions. The second use case was for keyword extraction tasks as an alternative way to use it for document similarity (see section 3.3).

Our keywords extraction idea works based on the properties of skip-gram model. Each word is a vector in the skip-gram space. So there is the possibility to perform vector arithmetic on the word vectors. For example adding vectors of words Man and Woman will produce a vector very similar to the word Human or adding vector of words head, leg, and hand will produce a vector very similar to the word body. In

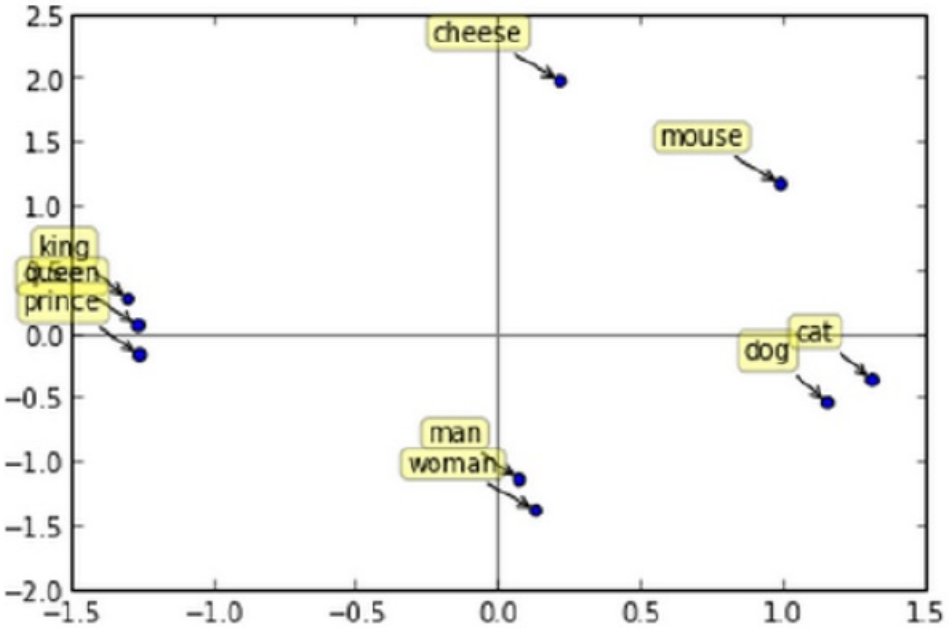

Figure A.1: Similar words are closer to each other in skip-gram (word2vec) space.

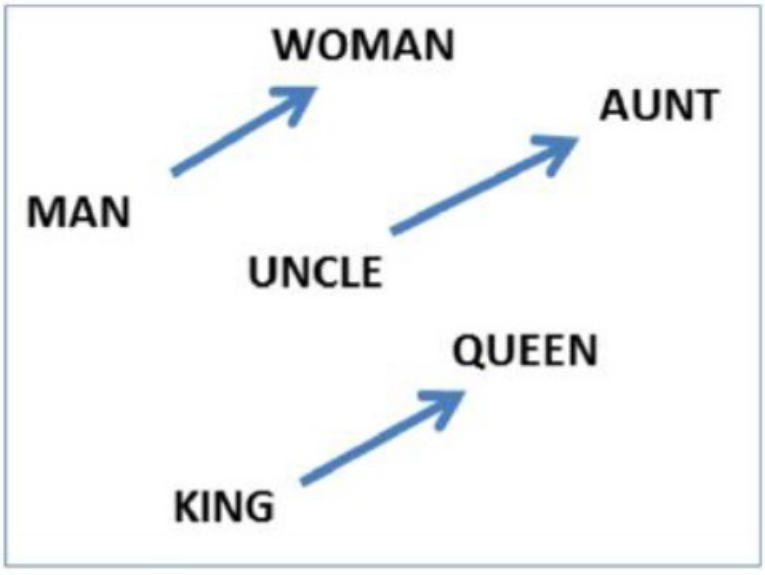

Figure A.2: The vector between two points represents a relationship

our keyword extraction component, we used this property to find a vector representation of a group of words of calculating sum of all vectors. As mentioned earlier, cosine similarity of vectors also produce semantic similarity score of them. So cosine similarity of a vector which represent a group of words with a single vector tells us the similarity of those two.

Skip-gram model can also be trained using mentions of named entities in the text. In this case, the produced vectors will represent named entities instead of words while it preserves all of the properties of ordinary word skip-gram model. We used cosine similarity of vector of named entities to calculate similarity of two named entities (see section 3.4). We preferred using skip-grams instead of other methods, because it was trained specifically on a news dataset and it is well suited for our news recommendation task. One other advantage of skip-grams is the speed of the similarity function which is basically done in constant time.

A.1 Document Vectors

We tried to propose a naive method to produce document vectors from word vectors to use it as one of our features. However it turned out that this method does not produce good results. After looking at the produced results, we focused on finding keywords from text to and used skip-grams in that way to find document pair similarity, but we thought this naive approach worth mentioning.

The naive solution is using word vectors is to sum up the vectors for all the words. This did not work because of the noise that was created by general words. To deal with noise, we used TF-IDF value of words to assign weighting on them. Formally, we could generate a document-vector matrix by using following formula:

$$D = W.V \tag{A.1}$$

Where V indicates word vectors and W indicates document-word vector with TF-IDF values. In this way, the resulting document vector D is very noisy, because we are considering all the words. The cosine similarity of two D vectors of two documents would indicate similarity of them if D was clean enough. Removing stop words helps a bit, but does not improve the quality of document vector significantly. It was clear that if we remove general words, we get better result and our research for

removing general words led us to a new idea of doing word clustering and selecting most informative words (Section 3.3).

Appendix B

Sunflower

Sunflower is a generalization system developed at Dalhousie University by Marek Lipczak[1] and was later used in other systems. This system works based on category graph extracted from Wikipedia. In Wikipedia, each article belongs to at least one category. Categories are special Wikipedia pages which cover a broader area than their individual articles. For example an article about Barack Obama belongs to the category of Presidents of the United States which contains all the presidents of that country. Categories can have a parent category or a child category. This property of categories leads to a hierarchical view of categories, from a fined-grained topic toward a very general topic like Politics. The relationship between categories can form a graph, in which an edge represents a parent-child relationship and each node represents a category. This graph is often called the *category graph*

Each language version of Wikipedia has its own version of category graph. Sunflower is a graph, which is a unification of category graphs of 120 Wikipedia language versions. The intuition behind using different language versions is that each language acts like a witness for the importance of stored relation. In other words, if more language versions contain a relationship between two categories, it means that the relationship is more important than others. For example the fact that Barack Obama is the president of the United States is cited in more languages than he is a winner of Grammy Awards. This information help us find the main category that an article falls into.

The input of Sunflower is an article in Wikipedia, and the output is the unified, weighted, and directed graph of categories, starting from the given article. In practice, each node of the graph, whether it is starting node, or a category in the category graph, can be expanded into many other nodes. Sunflower provides a width parameter to limit number of branches and considers only edges with highest value. The graph

[1]This work has not been published yet.

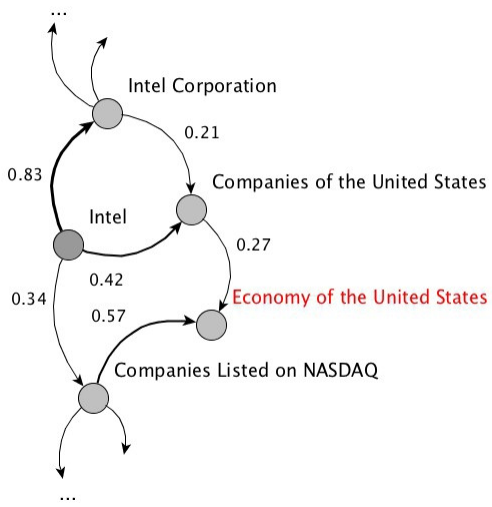

Figure B.1: Example of category graph generated by Sunflower given Intel as input.

can be traversed up to many levels and produce more general topics. Sunflower limits number of levels that the graph can be generalized by providing a height parameters. These two parameters can control how much a concept can drift and how much generalization is needed.

Sunflower graph is directed and weighted. The direction is from sub-category to parent category. The weight shows the fraction of languages which indicate the relationship. Starting from a given article, each path can produce a weight by multiplying the weights on its edges. Each category (i.e. node in the graph) can be scored by summing up the score of all the incoming paths. Figure B.1 shows an example of how the scoring system looks on the graph. In this example, given Intel as the query article, we calculate each category score. For example the score of Economy of the United States is the sum of the scores of two incoming paths:

- Sum of first path: Intel $->$ Companies of the United States $->$ Economy of the United States $\Rightarrow 0.42 \cdot 0.24 = 0.11$
- Sum of the second path: Intel $->$ Companies Listed on NASDAQ $->$ Economy of the United States $\Rightarrow 0.34 \cdot 0.57 = 0.19$
- Importance score of "Economy of the United States" given Intel as input $= 0.11 + 0.19 = 0.3$

The returned graph is a compact and accurate category profile for the given

Wikipedia article. It is a sparse representation of articles in category space.

One of the additional goals of developing Sunflower was to generalize a given concept. The hierarchical structure of categories in Wikipedia is used for this purpose. Generalization of concepts is helpful for determining context of two concepts. For example if two concepts are about politics, then Sunflower can identify politics as a shared topic by generalizing both concepts. This idea can be extended in order to calculate contextual similarity of concepts by using the category profile of two concepts. In other words, for each concept, a vector in category space is generate. The cosine similarity of the vectors of two concepts can generate a similarity score for two concepts.

Sunflower was used as a module inside Tulip system. The goal of Tulip is to recognize and disambiguate named entities. So Sunflower used in a limited capacity only for named entities in Wikipedia and not all Wikipedia articles. Beside that, used Sunflower similarity as a similarity function for two given entities and compared it with other methods.

Appendix C

Tulip

Tulip (Lipczak et al. [2014]) is an entity recognition and disambiguation system (ERD) developed at Dalhousie University and was the winning system at Named Entity Recognition and Disambiguation Challenge 2014 as a part of SIGIR 2014 organized by Microsoft and Google.[1] The challenge was to annotate words of a given text file and link them to entities of an external knowledge base which in was Freebase[2]. FreeBase is a knowledge base providing structural information for Wikipedia articles and there is a one to one correspondence between Wikipedia articles and FreeBase entities. All the participating systems had to return annotated text for a given query in less than 60 seconds. So response time was an important factor in designing such a system. The accuracy of participating systems was measured in terms of precision and recall based on a gold standard created by judges from the organizers. Named entity recognition and disambiguation is an old field in NLP. But Tulip is not using any NLP technique. Instead it uses a dictionary approach to match mentions of named entities to an external knowledge base.

NERD systems play an important role in text mining. They can link words or phrases to external knowledge bases that contain a meaningful structure. The structure behind each entity is used for different purposes. For example Sunflower uses that structure to generalize a given concept. In this work, NERD systems are used to extract entities and apply semantic similarity functions on them to find related documents. To the best of our knowledge, this is the first time named entities and semantic similarity among them is being used in this domain. Tulip was developer in response to the demand for a very fast and accurate NERD systems. Our previous experience with existing systems were not satisfactory; lack of precision in disambiguation task was always a problem. As NERD systems are placed in lower stack in text mining applications, any produced error will be propagated and make a negative

[1]Most of this section was appeared in main Tulip paper and during presentation at SIGIR 2014
[2]http://www.freebase.com/

impact on the performance of the entire system.

C.1 Data Sources

The effectiveness and efficiency of Tulip would not be possible without a number of openly available datasets. In this section we provide an overview of five datasets used in the project focusing on their applicability to the Entity Recognition and Disambiguation problem.

Wikipedia is being widely used in text annotation since it is densely structured by hundreds of millions of links among millions of articles Witten and Milne [2008]. Internal links in Wikipedia (*wiki-links*) and the anchor text associated with each one create a rich dataset with valuable statistical information. Given the number of links Wikipedia has a very high coverage of entity surface forms that can be extracted from links' anchor text. The frequency of reference to an article by a given anchor text is often used in disambiguating mentions which refer to more than one article. We refer to it as the *commonness score*.

Google's Wikilinks corpus Singh et al. [2012] can be considered as an extension of Wikipedia's wiki-links data to the Web. It contains 40 million mentions of over three million entities. These mentions are gathered based on finding hyperlinks to Wikipedia from a web crawl of over 10 million pages.

Freebase Bollacker et al. [2008] is a collaboratively created knowledge base. It contains data harvested from Wikipedia and other data sources. Each entity described in Freebase is manually assigned to one or more types. The type assignment was used to select the subset entities of entities for the ERD'14 Challenge Carmel et al. [2014]. In the challenge we worked on a subset of over two million entities extracted from Freebase. We refer to them as *Freebase sample*. All entities in the Freebase sample contained a link to corresponding Wikipedia article. The Freebase types can be also used to select entities for type specific approaches. For example, Tulip uses a special preprocessing technique for all entities of type person.

DBpedia Lehmann et al. [2014] is a knowledge base of relations automatically and manually extracted from Wikipedia. Just like Freebase it contains its own type hierarchy. One of the distinctive features of DBpedia is its effort in unifying information from various language versions of Wikipedia. All relations extracted from

Wikipedia articles in languages other than English are mapped to the English counterpart using Wikipedia's language links. We use this feature while computing term vectors representing entities.

Wiktionary is a free, collaboratively created dictionary. It can be considered as a data source complementary to Wikipedia Zesch et al. [2008], with a stronger emphasis on commonly used words and phrases. Wiktionary has a rich representation of common nouns and other parts of speech which we use as an evidence that a spotted entity mention can be in fact a commonly used phrase.

C.2 Modules

There are two sub-tasks in an ERD system. At first, words that are candidate to be annotated must be recognized. Each word is associated with a set of entities in the knowledge base which are called word senses. The word itself is called a surface form. The output of this step is a mapping between candidate surface forms and their set of possible meanings. This output is fed into the next component which is called disambiguator. At this stage, one of the senses for a given surface form must be chosen according to the general meaning of the text. There are many approaches to implement these two components. The mentioned two tasks are usually referred as spotting and disambiguation respectively.

In Tulip, internal components were selected in a way to achieve very fast performance. The first component, Solr Text Tagger[3], was used for recognizing surface forms in text. In the second component, Sunflower, is used for recognition and eliminate false positive annotations. Both of the components are very fast with very low memory footprint comparing to other ERD systems.

Solr Text Tagger uses a lexicon (dictionary) approach to find candidates. The lexicon of surface form text - entity pairs contains the vocabulary of words and phrases that can potentially indicate the mention of the entity in the text. To build a comprehensive lexicon we used information from three data sources: (1) Freebase sample extracted by the challenge organizers (2) Wikipedia, (3) Google's Wikilinks corpus (see Subsection C.1 for details). First, we extracted all the entity names from the provided Freebase sample. Next we processed all internal Wikipedia links from a

[3]https://github.com/OpenSextant/SolrTextTagger

Wikipedia dump retrieved in February 2014 storing the anchor text and the link target. We combined this dataset with Google's Wikilinks corpus creating a repository of anchor text - entity pairs together with the commonness score. The dataset was later merged with entity names, treating all names as a single link to their entity. The objective of the challenge was to recognize only the entities provided in the sample. However, some of these entities share the name or the anchor text with other entities in Wikipedia. Therefore we filtered the repository keeping only the entities from the Freebase sample or Wikipedia articles that share at least one anchor text with any of the Freebase entities.

Tulip's lexicon contains over 4 million surface forms linked to over 2 million entities. High coverage of entities comes with the cost of a large number of common phrases that are mistakenly taken as surface forms. For example, a word *details* is used as a surface form for over 5000 entities in our dataset. We prune surface forms that are likely to be incorrect, which are words with a high number of linked entities or strings with a majority of non-letter characters. Another problem are ambiguous entity names which are also used as common words (e.g., "It" – Stephen King's novel). As these surface forms can be a cause of a large number of falsely recognized entities, we decided to mark them as potentially suspicious. If these surface forms are spotted in the text as potential entity mentions, the Spotter will assign a *suspicious* flag to the mention. The current version of the system uses a composition of three soft filters: (1) **stop-word filter** marks all stop-words or phrases composed of stop-words (e.g., *This is*); (2) **Wiktionary filter** marks all common nouns, verbs, adjectives, etc. found in Wiktionary; (3) **lower-case filter** marks all lower-case words or phrases. It is important to notice that the first two filters are case insensitive. For all filters, mentions with strong evidence for the connection with the default sense are not filtered. By strong evidence we consider a large number of links with a given mention as an anchor text leading to the same entity. For example, the word "Apple" very often leads to Apple Inc. company.

An alternative approach for recognition would be using trained named entity recognizer like Stanford NER[4] package. In this approach, the text is passed into the NER component and named entities are extracted. Then for each named entity list

[4]http://nlp.stanford.edu/software/CRF-NER.shtml

of possible senses must be extracted from the dictionary. The advantage of this approach is that it does not require a clean dictionary which contains only valid named entities and it enables us to use a broad dictionary. During the challenge, there was not enough time to implement this approach so Solr Text Tagger remained as our spotter.

Given a text as the input, the text is passed through the Solr Text Tagger at the first stage. The output of this stage is candidate words and their possible entities for mapping. As mentioned before, the default sense of each word is also marked. At this point, the candidates and only their *default sense* are marked with a suspicious and non-suspicious flag. All of the candidate forms that are in lower case, common words, or stop words are marked as suspicious and others are marked as non-suspicious. The entities of non-suspicious forms (*entity core*) are passed to Sunflower and the returned category profile for each entity are agglomerated together to form a topic centroid. In other words, the topic centroid of the text is the representation of all non-suspicious entities in the category space. The mentioned representation comes from importance scores generated by Sunflower.

For all non-suspicious entities, the relatedness of its category profile and the topic centroid is calculated. An entity would be removed from output set if it is not very similar to the centroid. This comparison let us remove those non-suspicious candidates that are not related to the core topic of the text. In this way all the false positive examples are removed, and only those mentions with correct default sense is accepted. The similarity measure used at this stage is the cosine similarity of vectors of centroid and entities. In the end, all suspicious candidates are compared with the centroid and if they are similar to the centroid, they are accepted.

The system looks very simple compared to other available systems with very sophisticated disambiguation component, because it operates mostly on default sense and yet, it is very effective. To find out the importance of default sense in ERD systems, a study on 50 document of the challenge gold standard was conducted and it turned out that using only default sense of candidates can disambiguate 85% of mentions. Also, 5% of the mentions could be disambiguated using other mentions in the text(e.g., *E72* and *Nokia E72*). Although proper disambiguation can be useful for the remaining 10% of cases, it is more likely that by considering other senses the

rank	team name	F1	prec./recall	latency
1	Microsoft Research	**0.76**	0.83/0.70	1.49
2	*Tulip*	0.74	0.76/**0.71**	**0.29**
3	Seznam Research	0.72	0.79/0.66	2.33
4	National Taiwan University	0.71	0.76/0.67	7.66
5	Heidelberg Inst. for Theoretical Studies	0.70	0.77/0.65	4.97
6	Neofonie GmbH	0.70	0.76/0.65	0.53
7	Northwestern University	0.69	0.72/0.65	0.70
8	A^3 Lab (University of Pisa)	0.67	**0.87**/0.54	0.86
9	University of Amsterdam	0.63	0.74/0.55	0.71
10	University of the Basque Country	0.63	0.74/0.55	37.29

Table C.1: Challenge results for the first ten systems in long document track.

system would incorrectly discard the default sense. We confirmed this hypothesis in preliminary experiments in which all entities were scored in this step.

The most important aspect that makes Tulip superior to existing system, beside its response time, is its ability to reject some candidates according to the main topic of the text. As described earlier, we only focus on default sense of surface forms. So there is no sophisticated disambiguation component. Instead, we called the component that rejects bad candidate as recognizer because of its ability to recognize the main topic of the text.

C.3 Tulip Performance

Tulip response time is very fast comparing to other comparable systems. It produces a response for a 500 word query document in about 0.29 seconds. The F1 measure of the system in the challenge was 0.74. Tulip was used as a crucial component of the proposed system for extracting named entities from news articles. According to the challenge results(Table C.1) It is currently the state-of-the-art in named entity recognition and disambiguation task.

It's important to remember Tulip is a named entity recognizer and disambiguator system and named entities are different from concepts. For example the word guitar is not a named entity, but it is a concept describing all models of guitar. Instead, the word Fender Telecaster is a named entity since there is a type of guitar existing with this name. Other system that works based on dictionary usually suffer from this narrow difference of concepts and named entities. We solved this problem using

Wiktionary by tagging general words which usually refer to concepts. Words like flower, building, wall, and light are general concept. If the word wall refers to Pink Floyd's album The Wall then that would be considered in the recognition part by comparing to the topic centroid of the text. Otherwise, the system will never annotate such words.

In the news domain, recognizing named entities can help us identify people, places, parties, sport teams, and many other type of information accurately. In contrast, existing systems return many false positive example which results in an inaccurate entity profile of an article. Such errors propagate through a pipeline system like ours and decreases performance of the following components.

Appendix D

Raw Article for Clustering Example

Springtime is typically oilpatch bonus season. As the grass turns green, car dealerships and upscale stores tend to get busier. This year is different. Bonuses if they exist at all are expected to be small. Wages have been frozen. Oilpatch workers are happy to have a job at all. And that's the private sector. Public sector workers, like teachers and nurses, are on high alert as the provincial government makes noises about their pay. Alberta's premier, however, has said he does not plan to reopen contracts. Economists are saying that Albertans have a problem: they make too much money. And not only is it unaffordable, it's hurting the province's competitive position in the global marketplace. According to Statistics Canada numbers out last week, the average weekly wage in Alberta was 1,163 last year, 23 per cent higher than the national average. The median household income in the province is close to 100,000. Money not there anymore You're not going to fix it overnight, says Glen Hodgson, chief economist at the Conference Board of Canada. It's happened over a decade, with having nominal wage increases or bonuses that are one or two There's pressure right now for firms and governments to get their compensation under control because the revenues aren't there anymore. Hodgson was presenting the Conference Board's economic outlook for Western Canada, one in which he predicts Alberta will slip into recession this year and eke out 1.2 per cent growth in 2016. Until this year, Alberta's economy was growing at a fast clip: between 4.5 per cent and five per cent a year. That's what pushed wages up in the first place, said Gil McGowan, president of the Alberta Federation of Labour. Wages are the only way that ordinary people get their share of the pie, he said. If people believe in markets and I would point out that labour markets are markets they shouldn't be just accepting the fact that we have higher wages, but lauding the fact that we have higher wages, because it means that the market is working. Private vs. public sector Alberta's provincial government is preparing its budget for the next fiscal year, a budget that will have a 7-billion hole

because of lower energy prices. Alberta Premier Jim Prentice has been sounding the drum on wages, saying that the public sector earns too much. Public sector unions counter that argument by saying that wages need to be competitive with the private sector. But those wages are also expected to come down. We're seeing corporations this year with no wage increases, you're seeing layoffs, which brings down wages, said Charlie Fischer, former chief executive of energy giant Nexen. There are lots of ways to make sure you make your employees feel that you're working with them trying to sustain their jobs. People would rather have a job with a little less pay than no job, he added.