Automatic Microblog Classification and Summarization

by

Beaux P. Sharifi

B.S., University of Colorado, Colorado Springs, 1996

A thesis submitted to the Graduate Faculty of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2010
This thesis for Masters of Science degree by

Beaux P. Sharifi

has been approved for the

Department of Computer Science

by

Dr. Jugal Kalita, Chair

Dr. Maria Augusteijn

Dr. C.H. Edward Chow

Date
Sharifi, Beaux P. (M.S., Computer Science)

Automatic Microblog Classification and Summarization

Thesis directed by Professor Jugal Kalita

Recently, a new form of blogging has emerged known as microblogging. Microblogging is a simplified form of blogging where entries are restricted in length, typically to around 140 characters or less. Microblog usage has grown dramatically recently thanks in part to Twitter, the leading provider of microblogs, and the integration of microblogging services within several popular social networking websites such as Facebook. Due to their short length, microblogs are updated more often compared to traditional blogs – up to several times an hour. In fact, one recent study mentioned that the number of messages being sent using Twitter alone is over 50 million per day. These messages are sent at random or in response to perceived events or situations. While each message is short, collectively they represent a large body of information that can be used to discovering trends, opinions, news, and other source of real-time information. In this paper, we attempt to address some of the opportunities and challenges of automatically processing microblogging data by considering two specific problems. First, we attempt to automatically classify a single microblog post into a set of high-level categories using a naïve Bayes classifier. While such tasks have been performed before using traditional blogs, no such research exists to our knowledge of applying this technique to microblogging data. Our research indicates that even though an average microblogging post is only 11 words in length, they can be categorized into one of ten categories with an F1-measure up to 78%. The other problem we seek to address is a way to automatically summarize a large number of
microblogging messages that are related to a single topic. We develop two algorithms that are able to extract either highly representative phrases or complete sentences from a large group of messages and display them as summaries. Results from our algorithms are shown to be competitive with human-generated summaries for the same set of topics.
To my precious wife Sarah and our beautiful daughter Hazel

(and to our little one to come...)
Acknowledgements

Anyone who has ever taken on such an effort knows that a thesis is not unlike raising a child. It takes a small village of people to see a project of this magnitude through. This thesis is no exception. While, I have many people to thank for helping through this process, I would like to make special mention of the following individuals.

First, I would like to sincerely thank Professor Jugal Kalita. Jugal, you are an outstanding professor and one of the highlights of my education at UCCS. During the many times that I felt I was at a dead end, your insight and encouragement helped me push through. This thesis would not have been half as successful if it wasn’t for you. Thank you.

To the members of my thesis committee, Dr. Edward Chow and Dr. Maria Augusteijn, thank you for your encouragement, feedback, and support.

To the members of Dr. Kalita’s weekly research group, especially Ryan Freckleton, Justin Gray, Jeremy Witmer, and Suzette Stoutenberg, thank you for the many ideas and encouragement along the way as well. I have enjoyed our weekly pizza night together and hope to continue seeing many of you in weeks to come. I also hope to return many of the favors you have given me.

To Mark-Anthony Hutton, thank you for putting up with a crazy student in the throes of last minute research! You have been a fantastic student and friend to work with. You never hesitated in helping me out and always delivered beyond my expectations. Thank you Mark!

To Mike, Kyle, and Ramin, my best friends, who have always been there for me. To Mike, who always has the level advice I have often needed along this journey and many others. Thank you. To Kyle, thank you for encouraging me to take the higher road in my education. Your encouragement to
pursue a thesis instead of a project was instrumental in helping me accomplish this task. To Ramin, I want to say that Boofer is complete! Your encouragement and friendship thorough this and many other trials have been a joy to receive. Thank you.

To my mom whom none of this would have been possible without the deep sacrifice and constant belief you have shown me since I was a young boy. From that expensive Eagle computer you bought me that had to sit on the floor because we couldn’t even afford a desk to put it on, your sacrifice has paved much of what I am today. I love you.

To my darling Sarah, my beautiful wife who has had to endure many long nights away from me, thank you for supporting me through this difficult process, for taking care of our little Hazelnut on nights I could not, and for always believing in me. To the end, you inspired me and always pushed me to give it my best. I love you.

Finally, to the Lord my God, whom none of this would have been possible without. You have never left me nor forsaken me through this difficult journey. I love you.
Contents

1 Introduction.................................................................................................................................................. 1

1.1 Motivation................................................................................................................................................. 3

1.2 Approach Overview ............................................................................................................................... 4

1.3 Organization of Thesis ............................................................................................................................ 6

2 Naïve Bayes Classifier .................................................................................................................................. 9

2.1 Overview................................................................................................................................................ 9

2.2 Theory.................................................................................................................................................. 9

2.3 Estimating Probabilities ......................................................................................................................... 12

2.4 Text Classification and Naïve Bayes ..................................................................................................... 14

2.4.1 Feature Definition ............................................................................................................................ 14

2.4.2 Probability Estimation ...................................................................................................................... 16

2.4.3 Naïve Bayes Algorithm for Text Classification .............................................................................. 17

2.5 Feature Selection.................................................................................................................................. 18

2.5.1 Frequency-based Feature Selection ................................................................................................ 19

2.5.2 Mutual Information ......................................................................................................................... 20

2.6 Measures of Performance ...................................................................................................................... 22

2.7 Conclusions ........................................................................................................................................ 24

3 Microblog Classification ............................................................................................................................. 25

3.1 Introduction .......................................................................................................................................... 25
3.2 Related Work .................................................................................................................. 25
3.3 Approach ....................................................................................................................... 26
3.4 Classifier Choice .......................................................................................................... 27
3.5 Document Definition ..................................................................................................... 27
3.6 Target Classification Set ............................................................................................... 28
3.7 Training Corpus ........................................................................................................... 28
3.8 Training Corpus Results ............................................................................................... 31
3.9 Optimization ................................................................................................................ 33
3.10 Testing Corpus ............................................................................................................ 38
3.11 Results ........................................................................................................................ 40
3.12 Conclusions ............................................................................................................... 43

4 Microblog Summarization and its Evaluation ................................................................. 45

4.1 Introduction ................................................................................................................... 45
4.2 Related Work ............................................................................................................... 46
4.2.1 A Brief History of Summarization ........................................................................... 46
4.2.2 Approaches in the Literature ................................................................................. 47
4.2.3 Problem Dimensions ............................................................................................. 48
4.2.4 Challenges Described in the Literature ............................................................... 50
4.3 Problem Description .................................................................................................... 51
4.4 Selected Approaches .................................................................................................. 52
5 Preliminary Summarization ................................................................. 65

5.1 Introduction ...................................................................................... 65

5.2 Random Approach ........................................................................... 65

5.3 Length Approach ............................................................................ 65

5.4 Results and Discussion .................................................................... 66

5.5 Conclusions ................................................................................... 69

6 Phrase Reinforcement Summarization .............................................. 71

6.1 Introduction ...................................................................................... 71

6.2 Motivation ....................................................................................... 71

6.3 Phrase Reinforcement Algorithm ..................................................... 73

6.4 Results and Discussion .................................................................... 84

6.5 Conclusions ................................................................................... 92

7 TF-IDF Summarization ........................................................................ 94

7.1 Introduction ...................................................................................... 94

7.2 TF-IDF ............................................................................................ 94

7.3 TF-IDF Algorithm ........................................................................... 96
7.4 Evaluation Results ........................................................................................................ 98
7.5 Normalization Results .................................................................................................. 102
7.6 Conclusions ................................................................................................................ 109

8 Summary and Conclusions .......................................................................................... 110

8.1 Contributions ............................................................................................................. 116
8.2 Future Work ............................................................................................................... 118

References ..................................................................................................................... 124

Appendix A: Manual Microblog Classification Instructions .............................................. 131
List of Tables

Forthcoming...
List of Figures

Forthcoming...
1 Introduction

During recent years, socially-generated content has become pervasive on the World Wide Web. The enormous amount of content generated in blog sites, social-networking sites such as Facebook and MySpace, encyclopedic sites such as Wikipedia, have not only empowered ordinary users of the Web, but also contributed to the vastness as well as the richness of the Web’s content. One relatively new form of socially-generated content are microblogs, which are blogs constrained to very short updates of around a 140 characters or less. Microblogs are extremely limited in the length since they are designed to be transmitted over the Simple Messaging Service (SMS). Using normal text-messaging, users can update a microblog and receive updates all from a mobile phone. Combining the ubiquity of the cell phone with the ease of composing short messages, the practice of microblogging has become extremely popular in a relatively short period of time.

The concept of microblogging was conceived by Jack Dorsey of Odeo, Inc., back in 2006 when he and some co-workers were brainstorming new product ideas for their company (Sagolla, 2009). His idea was to use the Simple Messaging Service (SMS) of the cell phone to communicate one’s status to a group of followers. From this idea, the concept of microblogging was formed and realized through the creation of the microblogging website and service Twitter.com. Since then, Twitter has grown to one of the most popular websites on the Web. Presently, Twitter maintains over 100 million users that generate over 50 million updates (or “Tweets”) a day (Weil, 2010). While the majority of these tweets are pointless babble or simple conversations, approximately 3.6% of them are topics of mainstream news (Pear Analytics, 2009). Furthermore, even within the simple conversations of friends, a large amount of information is being communicated that could serve a variety of data mining applications.
Unfortunately, many of the tools available to users today to search and use this vast quantity of microblogging data are still in their relative infancy. For example, Twitter provides a search engine for searching for posts that contain a set of key words. However, the results are simply a list of posts returned by recency instead of relevancy. Therefore, it is not uncommon to receive a large amount of spam, posts in other languages, rants, and other sources if misinformation. Another service provided by Twitter is a list of currently trending topics. Trending topics are popular words or phrases that are currently occurring within the massive influx of new messages. For each trending topic, Twitter again only provides a list of related posts thus making it difficult to determine why a term is trending. While these types of tools may have novel uses for a casual user, they do not provide any real means for deeper analysis. In order to make practical usage of this large quantity of data, automated techniques that can perform data mining through natural language processing and machine learning are required.

In this paper, we attempt to address some of the opportunities and challenges of automatically processing microblogging data by considering two specific problems. First, we attempt to automatically classify a single microblogging post into a set of high-level categories. While such tasks have been performed for traditional blogs, no research to our best knowledge exists for applying this technique with microblogs. Therefore, we explore this problem and its opportunities by developing a naïve Bayes classifier that we train and test on a large corpus of Twitter microblogs. Second, in order to help users comprehend the massive amounts of new microblogging data, we address one of the major shortcomings of Twitter’s list of trending topics. Instead of providing a list of posts that are related to a trending topic, we determine to automatically generate a concise summary by processing a large number of posts that contain the topic’s phrase. For this task, we develop two novel algorithms based on research in the area of automatic text summarization and apply it to a corpus of popular Twitter topics.
1.1 Motivation

Blogging is not a new phenomenon. Since 1996, the practice of blogging, or creating an online journal to document one’s life, express one’s opinions or emotions, or to form and maintain communities has been a popular practice (Nardi et al., 2004). The amount of socially generated content from this medium is enormous as there are an estimated 126 million blogs currently in existence (BlogPulse, 2010). With the massive amount of socially-generated content, researchers have been using machine learning combined with natural language processing techniques to make use of this traditional blog data for quite some time as well. For example, researchers have attempted to discover blogger’s interests (Teng and Chen, 2006), gender (Yan and Yan, 2006), moods (Mishne, 2005), and public sentiment towards recent events (Durant and Smith, 2006). Others have attempted to separate legitimate blogs from spam blogs (known as “splog”) (Kolari et al., 2006). Finally, other researchers have focused on reducing the amount of information in blogs through summarization (Zhou and Hovy, 2006; Hu et al., 2007) and summarizing the opinions of users within blogs (Ku et al., 2006).

Two primary techniques for performing many of the above applications on traditional blogs are automatic classification and automatic text summarization. While these approaches have been applied to traditional blogs, very little to no published research exists to our best knowledge on the application of these approaches towards microblogs. Such applicability cannot be immediately assumed for the following reasons. First, individual updates to a microblog are considerably shorter than updates to traditional blogs. Our research indicates that a single microblog post averages only 11 words in length. Second, while traditional blogs are less structured than formal text, microblogs are even less so. Microblogs resemble the “structure” of instant messaging more than traditional blogs. Finally, microblogs have been shown to be used for different motivations and purposes than traditional blogs. Java et al. found that microblogs are often used for discussing daily activities and seeking or sharing information (Java et al., 2007).
If automatic classification could be successfully applied towards microblogs, then such usage could provide many of the same opportunities that exist for traditional blogs. Furthermore, microblogs may also provide several new opportunities. For example, microblogs are often updated several times a day as a user comments on their status, activities, and events they perceive or participate in. With a constant source of real-time data from an individual microblogger, this data could be used for understanding individual patterns of behavior such as routines, habits, interests, or hobbies. These, in turn, could be used for targeting advertisements, surveillance, or helping a user become more self-aware. Likewise, collective patterns of behavior such as the ones mentioned for traditional blogs could also be studied on a more fine-grained basis with the availability of frequent updates to microblogs.

Finally, microblog summarization also has a number of potential uses. One use is to help understand the trending topics that Twitter and other microblogging services present to the microblogging community. Trending topics are the most frequent words or phrases that are occurring at the current moment. Rather than trying to manually understand trending topics by reading a set of related posts, one could envision an automatically generated summary that would explain the trends in real-time. An automatic summary would thus save readers time and also help determine which topics were worth further reading. These summaries could also be tailored towards specific events by combining their use with automatic classification. For example, posts could be filtered using automatic classification into a specific topic of interest such as the political situation in Iran. From these, summaries of recent events in Iran could be generated automatically. If this process was repeated over time, one could generate a running timeline of summarized events within specific categories.

1.2 Approach Overview

In this thesis, we attempt to apply the machine learning tasks of automatic classification and automatic summarization to the relatively new medium of microblogs. For each task, we considered the
most successful approaches made available in the literature and used these approaches as the basis for our selected approaches. For automatic classification, we found the naïve Bayes classifier to be the most promising approach based on two comparative studies that compared using naïve Bayes against a Support Vector Machine for classifying traditional blogs. In light of these studies, we chose a naïve Bayes classifier and used it to classify a single microblogging post into one of ten popular interest categories such as Sports, Science, or Technologies. The naïve Bayes classifier is a statistical approach towards classification that uses a set of features and associated probabilities to classify an unknown object. The next two chapters of our thesis are dedicated towards this topic.

The remainder of this thesis is devoted towards automatic microblog summarization. Automatically summarizing a microblog is an instance of the more general problem of automatic text classification since microblogs are primarily composed of text. Automatic text summarization is a diverse field with a wide variety of techniques and approaches. However, these approaches can be generally divided according to one of two types of summaries produced: either extracts or abstracts. An extractive system attempts to find the most salient pieces of texts from the source text and use it as the summary. In contrast, an abstractive system doesn’t use the source text as the summary but rather generates its own paraphrase of the source text. Extractive systems are considerably more popular in the research primarily because of their simplicity and flexibility towards documents of different domains. Abstractive systems are generally more successful at producing more concise summaries since they are not limited to the source text. However, because their usage requires outside knowledge sources such as ontologies, they are typically only used in limited domains.

Since our task is to summarize microblogs and microblogging topics are extremely diverse, we also chose to use extraction. Furthermore, we apply two completely different methods towards this problem. First, we develop a novel graph-based algorithm called the Phrase Reinforcement algorithm
that composes a graph representing the common phrases used around a central topic. From this graph, the algorithm is able to identify the most commonly used phrase that contains a topic phrase. This phrase is then used as a summary. For the second technique, we apply a classical approach towards summarization known as Term Frequency – Inverse Document Frequency (TF-IDF). TF-IDF is nothing more than a method for weighting terms (words) according to their relative frequencies within a set of documents. These weights are then summed across the terms in a sentence for producing an overall sentence weight. The sentence with the most weight is chosen as the summary. However, we develop two novel adaptations to this algorithm that allows its usage within the context of microblogs since microblogs are not a typical type of document.

Finally, for all of our tasks, we use Twitter exclusively as our source of training and testing data. Since Twitter is the leading microblogging service on the web and has over 100 million users, it provides the depth and breadth of data that is needed for the types of machine learning algorithms we chose. Furthermore, because of its flexible programming API, it makes for a logical choice from a development perspective as well.

### 1.3 Organization of Thesis

In Chapter 2, we present a thorough overview of the mathematical foundations of the naïve Bayes classifier. Included in this overview are a derivation of the naïve Bayes equation, a presentation of the maximum likelihood estimate (MLE) formula for estimating required probabilities, and a detailed look at how the classifier is applied towards the specific issue of text classification. Finally, we conclude the chapter with a presentation on how to optimize the classifier as well as evaluate its performance.

Chapter 3 applies the mathematical concepts of Chapter 2 by implementing a naïve Bayes classifier for the specific task of classifying microblogging posts into one of ten interest based categories such as Sports, Science, or Politics. After presenting the related research in traditional blog
classification, we motivate our approach for choosing the naïve Bayes classifier over other automated techniques such as using a Support Vector Machine or Language Model. Finally, we present a detailed look at the steps we used to train, optimize, and evaluate our classifier and then conclude with our results.

In the remaining chapters of our thesis, we change focus and look at another application of machine learning and NLP with microblogs: automatic microblog summarization. In Chapter 4 we give a brief history of automatic text summarization and also describe the many dimensions and challenges associated with this diverse field. Following this, we begin describing our selected approaches for this task by giving an overview of the algorithms we develop in future chapters. The remainder of this chapter is dedicated to describing how we will evaluate our automatic summarization results.

In Chapter 5, we give a brief presentation of some simple preliminary approaches we perform in order to establish an expected range of performance for our two evaluation metrics we chose in Chapter 4.

Chapter 6 presents our first formal algorithm into automatic microblog summarization through the presentation of our novel Phrase Reinforcement algorithm. The motivation for this algorithm as well as its methodology are presented in detail with a running example to illustrate the algorithm’s unique graph-based structure. We conclude the chapter with an evaluation of its results, some sample generated summaries, and a discussion of its performance and limitations.

We conclude our approaches in automated microblog summarization in Chapter 7. In this chapter, we develop another implementation based on the classical term-weighting technique known as TF-IDF. After presenting the algorithm, we evaluate its performance using the evaluation metrics developed in Chapter 4 and present some example summaries. Finally, we devote the remainder of the
chapter to describing how we determined a novel sentence length normalization algorithm which is necessary for preventing the algorithm from biasing towards the longest sentences.

Finally, Chapter 8 summarizes and concludes the totality of our work. We give an overview of the previous chapters as well as highlight any key results, conclusions, and limitations to our work. We conclude the thesis with a description of our key contributions and a list of potential future research. This chapter is followed by a list of our literature references and appendices.
2 Naïve Bayes Classifier

2.1 Overview

Naïve Bayes classifiers (NBC) are one of the most popular supervised learning techniques available for classification. Their popularity stems from the fact that they are efficient to implement, mathematically easy to follow, and often provide competitive results compared to other learning-based techniques such as decision trees or neural networks (Mitchell, 1997). For these reasons, naïve Bayes classifiers have been used in a variety of information retrieval (IR) and natural language processing (NLP) applications such as spam filtering and news article classification (Kim et al., 2006).

Naïve Bayes classifiers help solve the problem of simple classification: given an object of unknown class and a set of predefined classes, find the most appropriate class in which the unknown object belongs. Since the NBC is a supervised learning technique, it must be trained before it can begin classifying. In particular, it requires a supervisor, or trainer, to provide it a set of pre-classified training examples in order to learn how to classify unknown examples. These examples are typically composed of a vector of features in addition to their target classification result. The NBC classifier, in turn, learns to recognize probabilistically how much each feature contributes to an object belonging to one class over another. From this, it is able to learn how to classify an unknown example into one of the available classes.

2.2 Theory

Mathematically, we can formalize our description of a naïve Bayes classifier and its role by synthesizing the descriptions found in (Mitchell, 1997), (Manning, 2008), and (Kalita, 2002). Assume we are given a discrete set of target classes \( C = \{c_1, c_2, \ldots, c_J\} \) and a set of \( m \) training examples of the form:
\[
\{ (x_1, c_1), \ldots, (x_m, c_m) \mid x_i \in X, c_i \in C \} 
\]

where \( X \) is the feature space and \( x_i \) is a feature vector \((a_1, a_2 \ldots a_n)\) composed of \( n \) attribute values such that \( a_i \in A_i \). Each \( c_i \) is the target classification value of \( x_i \) given a target concept \( f : X \rightarrow C \) that maps any instance \( x \in X \) into its target classification value \( c \in C \). The role of the naïve Bayes classifier is to learn the target concept \( f \) after seeing the set of \( m \) training examples.

The Bayesian approach to solving the above problem is to assign the most probable class \( c \in C \) given an instance \( x \in X \). This is known as finding the maximum a posteriori (MAP) classification value. In other words, the Bayesian classifier is interested in finding \( c_{\text{MAP}} \) where:

\[
c_{\text{MAP}} = \arg \max_{c_j \in C} P(c_j | x) 
\]

Therefore, the most likely classification of a new instance \( x \) is the one that maximizes the conditional probability \( P(c_j | x) \). Substituting our feature vector \((a_1, a_2 \ldots a_n)\) for \( x \) we have:

\[
c_{\text{MAP}} = \arg \max_{c_j \in C} P(c_j | a_1, a_2, \ldots, a_n) 
\]

Approximating the probability \( P(c_j | a_1, a_2, \ldots, a_n) \) is very difficult since it would require an extremely large set of training examples. Therefore, we can help simplify this probability by rewriting the expression using Bayes theorem which states:

\[
P(A | B) = \frac{P(B | A) P(A)}{P(B)} 
\]

where \( A \) and \( B \) are any two random variables. Applying Bayes theorem to Equation 3 above, we have:

\[
c_{\text{MAP}} = \arg \max_{c_j \in C} \frac{P(a_1, a_2, \ldots, a_n | c_j) P(c_j)}{P(a_1, a_2, \ldots, a_n)} 
\]
Since for every \( c_j \in C \) the value \( P(a_1, a_2, ..., a_n) \) is a constant, we can drop it without affecting \( c_{\text{MAP}} \) like so:

\[
c_{\text{MAP}} = \underset{c_j \in C}{\text{argmax}} \left[ P(a_1, a_2, ..., a_n|c_j)P(c_j) \right]
\]

Determining the probability \( P(c_j) \) is not difficult since we can approximate it using the training data as we will explain momentarily. However, approximating the value \( P(a_1, a_2, ..., a_n|c_j) \) still requires a large number of training examples. We can further factor this probability by applying the chain rule (Russell & Norvig, 2003) which states:

\[
P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|x_{i-1}, ..., x_1)
\]

Applying the chain rule of probability we have:

\[
c_{\text{MAP}} = \underset{c_j \in C}{\text{argmax}} \left[ P(c_j) \prod_{i=1}^{n} P(a_i|a_{i-1}, ..., a_1, c_j) \right]
\]

Finally, in order to allow approximating \( P(a_i|a_{i-1}, ..., a_1, c) \) with a reasonably small number of training examples, we make one large simplifying (or naïve) assumption: namely that the attributes \( a_i \) are conditional independent from one another given a target classification \( c_j \). In other words, we assume that:

\[
P(a_1, a_2, ..., a_n|c_j) = \prod_{i=1}^{n} P(a_i|c_j)
\]

By making this final assumption, we can approximate the conditional probability \( P(a_i|c_j) \) with relatively few training examples. This last simplification gives the canonical form of the naïve Bayes classifier, namely:
\[
c_{NB} = \arg\max_{c_j \in C} \left[ P(c_j) \prod_{i=1}^{n} P(a_i|c_j) \right]
\] (10)

where \(c_{NB}\) is the predicted target classification value of the naïve Bayes classifier.

Finally, multiplying together a large number of small probabilities can cause floating point underflow errors. Therefore, what is commonly performed is adding logarithms of the probabilities instead of multiplying them like so:

\[
c_{NB} = \arg\max_{c_j \in C} \left[ \log P(c_j) + \sum_{i=1}^{n} \log P(a_i|c_j) \right]
\] (11)

Note that the maximization \(\arg\max\) still holds since \(\log(xy) = \log(x) + \log(y)\) and the logarithm function is monotonically increasing (Manning, 2008).

Intuitively, one can describe this last form of the naïve Bayes classifier as a sum of weights where each term contributes to the evidence supporting a particular target classification (Manning, 2008). The first term \(\log P(c_j)\) give more weight to more likely classes since they stand a better chance of being correct. The remaining terms \(\log P(a_i|c_j)\) weigh the amount of supporting evidence for a target classification given the occurrence of a particular attribute value.

### 2.3 Estimating Probabilities

In order to solve the canonical naïve Bayes formula for our MAP target classification, we must know the following probabilities: the prior probabilities of a class \(P(c_j)\) for all \(c_j \in C\) and the conditional probabilities \(P(a_i|c_j)\) for all \(c_j \in C\) and \(\{a_i \in A_i\}_{i=1}^n\). We can approximate these probabilities from the training data by using their observed relative frequency as the probability estimate. Doing so is called using the maximum likelihood estimate (MLE) since it “makes the observed data maximally likely.” (Manning, 2008) We can define the MLE of a probability as the following:
\[ p = \frac{q}{r} \]  \hspace{1cm} (12)

where \( \hat{p} \) is the MLE estimate of the true probability \( P \), \( q \) is the observed number of times an event occurs, and \( r \) is the number of opportunities in which the event could occur. Using the MLE, we can approximate the probability \( P(c_j) \) as follows:

\[ P(c_j) = \frac{N(C = c_j)}{m} \]  \hspace{1cm} (13)

where \( N(C = c_j) \) is the number of training examples with a target classification of \( c_j \) and \( m \) is the total number of training examples as before. Similarly, we can approximate the probability \( P(a_i|c_j) \) as:

\[ P(a_i|c_j) = \frac{N(C = c_j, A_i = a_i)}{N(C = c_j)} \]  \hspace{1cm} (14)

where \( N(C = c_j, A_i = a_i) \) is the number of training examples with a target classification of \( c_j \) and attribute \( A_i \) equal to \( a_i \), and \( N(C = c_j) \) is the total number of training examples with a target classification of \( c_j \).

One problem that can occur with the approximation \( \hat{P}(a_i|c_j) \) above is the equation can equal zero when there are not enough training examples. Since \( \log 0 \) is undefined, Equation 11 cannot tolerate zero value terms. Neither can Equation 10 since multiplying the remaining terms by zero causes the entire probability to collapse to zero. One solution to this problem is to use the \( m \)-estimate of probability:

\[ \hat{p}_m = \frac{q + mp}{r + m} \]  \hspace{1cm} (15)

where \( q \) and \( r \) are the same from our MLE formula, \( m \) is a constant, and \( p \) is our prior estimate of the probability we are trying to estimate (Mitchell, 1997). In effect, the \( m \)-estimate of probability adds an additional \( m \) virtual samples distributed according to the probability \( p \). One method for
choosing \( p \) if its estimate is unknown is to assume uniform priors such that if an event can have \( k \) different outcomes, then \( p = \frac{1}{k} \) (Mitchell, 1997).

### 2.4 Text Classification and Naïve Bayes

The preceding section described the general theory of the naïve Bayes classifier without making any assumptions on the types of objects being classified nor their specific types of features. However, given the fact that our task is to classify microblogs, a text-based document, we can make some additional assumptions in order to help relate the theory of the naïve Bayes classifier to our specific task of classifying documents. Document classification, or more generally, text classification, can be performed using the naïve Bayes classifier by applying two fundamental design steps. First, we must define how to represent a document in terms of a number of features (or attributes). Second, we must estimate the various probabilities required by the naïve Bayes classifier. The following two sections describe these steps in more detail.

#### 2.4.1 Feature Definition

Before we can execute the naïve Bayes equation, we must define an object, or more specifically, a document, in terms of features. The reason this step is necessary is because the naïve Bayes equation is not defined in terms of documents, but rather features. There are many choices of features for a document. One can choose to use words, phrases, or any other type of discriminating feature within a document (Kalita, 2002). While larger feature spaces, such as phrases, may potentially increase the accuracy of the classifier, it is often difficult to use them unless a large amount of training data is available. Because the naïve Bayes classifier requires conditional probabilities for each of the features, one must have enough training data in order to generate reasonable probabilities. For this reason, most implementations choose to use individual words as the feature space.
Creating a feature space for a document using individual words is generally performed using one of two methods. The first method, known as the multivariate Bernoulli model, represents a document as a vector of binary word occurrences. In this model, a vector \( \langle w_1, w_2 \ldots w_n \rangle \) is defined over a vocabulary of \( n \) words where each \( w_i \in \{0,1\} \) is a word and indicates the presence of absence of the word in the document being classified (Manning, 2008). If the word \( w_i \) is present anywhere in the document, it is given the value of one (1) in the vector. Otherwise, it is given the value of zero (0). Estimating the probabilities for each of the terms in the vector amounts to estimating the percentage of training documents that contain each of the words. Note that these estimates are irrespective of the number of times each word occurs within a particular document since the vector is binary: either a word occurs within a document or it does not. The Bernoulli model has been shown to perform well when there is a very small vocabulary size on the order of a few hundred words or less (McCallum & Nigam, 1998). However, its performance drastically decreases when either the vocabulary size grows or when performing classification on longer types of documents since it ignores the frequency of word occurrence (Manning, 2008). In these types of situations, it is preferable to use the second method for representing single word feature spaces, the multinomial model.

The multinomial model represents a document as an ordered set of word positions where each position is a random variable that can assume the value of one of the words within a predefined vocabulary. In other words, it defines a document as an ordered vector of words \( \langle w_1, w_2 \ldots w_n \rangle \) where \( w_i \in V \) is the word in \( i \)th word position in the document being classified, \( n \) is the number of words in the document and \( V \) is a predefined set of vocabulary words. Using this definition of a document, the naïve Bayes equation can be restated as follows:

\[
c_{NB} = \arg \max_{c_j \in C} \left[ \log P(c_j) + \sum_{i=1}^{n} \log P(a_i = w_k | c_j) \right]
\]  

(16)
where \( n \) is the number of words in the document and \( w_k \in V \) is the word found in the \( i \)'th word position of the document. Estimating probabilities for each of the \( a_i \) terms requires computing the probability of a particular word occurring in a particular position in a document. Note that this estimate, unlike the Bernoulli model, does take into account the frequency of words since it considers every word position within a document.

Unfortunately, estimating conditional probabilities for every word and word position requires a very large amount of training data since we must have estimates for every combination of word, position, and class. In order to reduce the amount of training data required, the multinomial model makes one additional simplifying assumption \textit{in addition} to the naïve Bayes assumption of conditional independence between attributes. The multinomial model also assumes that the probability of a word occurring in a specific position is the same as the probability of that same word occurring in any other position (Mitchell, 1997). In other words, the \( P(w_i = v_k|c_j) = P(w_m = v_k|c_j) \) for all \( i, j, k, m \) where \( v_k \in V \). Given this additional assumption, the naïve Bayes classifier assumes that words are not only conditionally independent from one another, but also identically distributed throughout the document given the target classification. By making these two assumptions, we are now able to estimate probabilities for these terms with only a reasonable amount of training data.

\textbf{2.4.2 Probability Estimation}

Now that we have defined the features of our classifier, we must also estimate the set of probabilities required by the naïve Bayes equation. For estimating the prior class probabilities, \( P(c_j) \), we can use the maximum likelihood estimate (MLE) exactly as we defined before by defining a training example as a document. In other words, we estimate the prior probability of a class as the percentage of training documents with a target classification equal to that class:
\[ \hat{P}(c_j) = \frac{N(C = c_j)}{m} \]  

(17)

where \( N(C = c_j) \) is the number of training documents with a target classification of \( c_j \) and \( m \) is the total number of training documents.

For estimating the conditional probabilities of attributes, \( P(a_i|c_j) \), we will also use the MLE estimate but also use the \( m \)-estimate of probability with uniform priors. In this case, assuming our simplifying assumptions of words being both independent from one another and identically distributed given the target classification, we can estimate the conditional probability of a word as follows. We estimate the probability as the number of times the word occurs within documents of the target classification divided by the total number of words within the same set of documents. In other words,

\[ P(a_i = w_k|c_j) = \frac{n_{kj} + 1}{n_j + |V|} \]  

(18)

where \( n_{kj} \) is the number of times word \( w_k \) occurs within training documents of target classification \( c_j \) and \( n_j \) is the total number of words (including duplicates) within the same set of documents of class \( c_j \).

2.4.3 Naïve Bayes Algorithm for Text Classification

We have now defined all the necessary pieces for implementing the naïve Bayes classifier for the purposes of performing text classification for microblogs. In summary, we present our classifier in algorithmic form in Figure 1 below in order to help clarify our final design used to classify microblogs.
TrainMultinomialNaiveBayesClassifier( Classes C, Documents D )
{
    Vocabulary V = ExtractUniqueWords( D )
    Number N = CountDocuments( D )

    for each Class c_j \in C do
    {
        Documents d_j = ReturnDocumentsOfClass(c_j, D)
        \[ p(c_j) = \frac{\text{CountDocuments}(d_j)}{\text{CountDocuments}(D)} \]
        Text_j = ConcatenateTextOfAllDocuments( d_j )
        \[ n_j = \text{CountWordPositions}(\text{Text}_j) \]

        for each Word w_k \in V do
        {
            \[ n_{kj} = \text{CountOccurrencesOfWord}(w_k, \text{Text}_j) \]
            \[ p(a_i|c_j) = \frac{n_{kj} + 1}{n_j} \]
        }
    }
    ClassifyDocument( Document d )
    {
        TBD...
    }
}

Figure 1. The multinomial naïve Bayes algorithm.

2.5 Feature Selection

In the preceding sections, we assumed that the features of our classifier were the set of all words found within the training documents. While training the classifier in this way is surely possible, doing so can lead to the problem of overfitting. Overfitting is a problem where the classifier performs better on the training data compared to its general performance over non-training data. It can occur when either the training data is not a good statistical representation of more general sets of data or when it contains a large number of noise features (Mitchell, 1997). Noise features are features that add incorrect evidence of a target classification through statistical anomalies in the training data. For example, suppose a rare word that has no information about one of the target classes happens by
chance to only occur in training documents of a single class. Since the naïve Bayes classifier is based on probabilities of words from the training data, the word will give incorrect evidence towards the single class when found in future documents.

Since the goal of the classifier is to optimize its general performance over all instances of data, we can reduce the effects of overfitting by attempting to remove the noise features from the feature set. Choosing a subset of features from the training data is known as feature selection and is performed using a variety of techniques. In each of these techniques, the goal is to find the set of features that provide the most discriminating power over choosing the correct target classification while eliminating the noise features.

2.5.1 Frequency-based Feature Selection

The first method for choosing features is based upon a consequence of Zipf’s Law which states that the naturally occurring frequencies of words in a natural language such as English are inversely-proportional to their frequency rank. In particular, a word of frequency rank k occurs approximately 1/k times as often as the most frequent word in the same set of documents (Joachims, 2001). A consequence of this relationship is that a small subset of words within the English language will tend to dominate the set of words in documents while the vast majority of other words will be highly infrequent. From this knowledge, we can draw a couple of conclusions for choosing the set of words from our training documents as features.

First, since most words are highly infrequent, when they do occur, they are likely to become noise features since we will likely not have enough training data in order to determine a good statistical approximation of their conditional probabilities. Therefore, we will likely want to remove most of the infrequently occurring words that don’t occur more than some threshold such as n times or more. The value n should be chosen experimentally based on the data available. Second, using a similar argument,
a small subset of words will occur very frequently. These words, known as stop words, are words such as “and”, “the”, and “it” that occur frequently for any type of document. For this reason, they don’t provide much discriminating power and can also be removed in order to prevent them from becoming noise features. One method for removing stop words is to sort the available set of training words by frequency and then to manually remove the most frequent \( m \) words that don’t appear to contain information about the available set of target classifications (Kalita, 2002). Manually creating the stop word list is advisable since the stop word list will be short (as per Zipf’s Law) and because it prevents accidently removing a frequent word that may have information about the target classification.

### 2.5.2 Mutual Information

Another method for feature selection is a more sophisticated statistical test that measures the amount of mutual or shared information between the target classification and the available features. Mutual information quantifies the amount of information (in an information theory sense) the presence or absence of a word contributes to knowing the target classification. For example, knowing that a document contains the word “football” increases our certainty that the document is sports related.

In general, mutual information is an attempt to measure the amount of mutual dependence there is between two random variables. Specifically, it quantifies the amount of information added to one random variable from knowledge of another (Schnieder, 2005). For two discrete random variables, it is defined as:

\[
M I(X; Y) = \sum_{x \in X} \sum_{y \in Y} P(X = x, Y = y) \log_{2} \frac{P(X = x, Y = y)}{P(X = x)P(Y = y)}
\]  

(19)

In order to apply the concept of mutual information towards feature selection, we must define the random variables of the mutual information equation in such a way that it allows us to rank our features in terms of their mutual information. A common method for this is to assign one of the
random variables as containing the set of target classification values \((C)\) and the other as a binary random variable \(W_t\) for each word \(w_k\) in the vocabulary \(V\). The binary variable \(W_t\) indicates whether a random word position in a document contains the word \((W_t = 1)\) or does not \((W_t = 0)\) (Schneider, 2005). By defining the second random variable over each word, or feature, we can compute the amount of information each feature contributes to knowing target classification.

While Equation 19 gives us a usable form for comparing features, we still need to estimate the probabilities defined by the formula. Estimating the probabilities can be made easier by rewriting the equation using the product rule of probability which states:

\[
P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)
\]

Using the product rule, we substitute \(P(X = x, Y = y)\) with \(P(X = x|Y = y)P(Y = y)\) which reduces the equation to:

\[
MI(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(X = x|Y = y)P(Y = y) \log_2 \frac{P(X = x|Y = y)}{P(X = x)}
\]  

(21)

Substituting our new definitions of the random variables within the equation, we have:

\[
MI(W_t;C) = \sum_{x \in \{0,1\}} \sum_{c_j \in C} P(W_t = x|C = c_j)P(C = c_j) \log_2 \frac{P(W_t = x|C = c_j)}{P(W_t = x)}
\]  

(22)

Given this new form of the equation, we can estimate the class probabilities \(P(C = c_j)\) using the same MLE we used in Equation 17 for the naïve Bayes equation. For the feature probabilities \(P(W_t = x)\), we also use the MLE but add the \(m\)-estimate of probability with uniform priors to prevent zero probabilities in the denominator of Equation 22 like so:

\[
\hat{P}(W_t = 1) = \frac{n_k + 1}{n + |V|}
\]  

(23)

\[
\hat{P}(W_t = 0) = 1 - \hat{P}(W_t = 1)
\]  

(24)
In these equations, \( n_k \) is the number of times word \( w_k \) occurs within all the training documents and where \( n \) is the total number of words within all the training documents. Finally, to estimate the conditional probabilities \( P(W_i = x|C = c_j) \), we use essentially the same estimate but constraining our document set to only those within class \( c_j \):

\[
\hat{P}(W_i = 1|C = c_j) = \frac{n_{kj} + 1}{n_j + |V|} \tag{25}
\]

\[
\hat{P}(W_i = 0|C = c_j) = 1 - \hat{P}(W_i = 1|C = c_j) \tag{26}
\]

where \( n_{kj} \) is the number of times word \( w_k \) occurs within training documents of class \( c_j \) and \( n_j \) is the total number of words in training documents of class \( c_j \).

Given that we have now defined a method for measuring the amount of mutual information for each feature or word, we can now rank a set of features from highest to lowest mutual information. Once this is performed, we can choose to keep the top \( N \) features with the most mutual information as one of our feature selection method. In practice, \( N \) is chosen experimentally based on resulting performance over a set of training and testing data (Kalita, 2002). This process will be described more fully when we apply mutual information towards reducing the number of features in our microblog classifier.

### 2.6 Measures of Performance

Now that we have methods for implementing our naïve Bayes classifier for performing text classification and methods for choosing features in a way that minimizes overfitting, we need some methods for measuring and comparing our classifier’s performance. For this, we will use two common metrics used within the domain of information retrieval: precision and recall.

Precision and recall are most easily understood in the context of retrieving documents as the result of a query (Manning, 2008). Suppose we perform a hypothetical query on a database that
returns a set of documents. Also suppose that some of the documents are relevant to our query and some are not. If we divide the number of returned documents that were relevant by the total number of documents returned, we get an idea of the purity of the results which is precision:

\[
\text{Precision} = \frac{\# \text{relevant and retrieved}}{\# \text{retrieved}}
\]  
(27)

Likewise, if we divide the number of returned documents that were relevant to our query by the actual number of relevant documents in the database, we get an idea of the completeness of the results which is recall:

\[
\text{Recall} = \frac{\# \text{relevant and retrieved}}{\# \text{relevant}}
\]  
(28)

While having two different metrics is insightful, they often only make sense in the context of one another. To understand this, consider the fact that the query could achieve perfect precision by returning only a single relevant document. Alternatively, it can achieve perfect recall by returning every document in the database. Instead, one typically desires some balance between precision and recall. The most common means of balancing these two metrics is through a weighted harmonic mean of the two values, \textit{F-measure}:

\[
F_\beta - \text{measure} = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}
\]  
(29)

where \(\beta\) is a weighting factor that either gives more weight to precision \((\beta < 1)\) or recall \((\beta > 1)\) (Manning, 2008). Typically, \(\beta\) is set to 1 so that it gives equal weight to precision and recall. In this case, the metric is usually written in its shorthand form: \(F_1\).

Finally, we can relate the information retrieval metrics precision and recall to the domain of document classification as follows. Given a “query” that is trying to find all the training documents with a target classification of \(c_j \in C\), we call all the training documents that actually belong to \(c_j\) as the set of relevant documents. Next, assuming we classify every training document using our
classifier, we call all the documents that are classified as belonging to \( c_j \) (either correctly or incorrectly) as the set of retrieved documents. Finally, we call all the retrieved documents that also belong to \( c_j \) as the set of retrieved and relevant documents.

### 2.7 Conclusions

In this chapter, we formally defined the mathematics of the naïve Bayes classifier and considered in detail its application towards the domain of text classification. In the next chapter, we apply the naïve Bayes classifier to the domain of microblog classification in order to understand its applicability to this new document type.
3 Microblog Classification

3.1 Introduction

In Chapter 2, we formally introduced the naïve Bayes classifier, a supervised learning technique for performing automatic classification. In particular, we described its theory while also addressed the practical issues of its application towards text classification. In this chapter, we apply this knowledge to a large collection of microblogs in order to understand its performance limitations on this relatively new domain. Rather than attempt to classify entire microblogs, we instead choose to classify a single microblogging post. Microblog posts are extremely limited in length – typically only 140 characters in length. For this reason, it is unknown how well the naïve Bayes classifier will perform on such a short document. In order to understand this research question better, we compose a large training corpus of microblogs from the popular microblogging service Twitter. Next, we train a naïve Bayes classifier to automatically classify a single microblogging post into one of ten popular interest categories such as Sports, Entertainment, or Science. Following this, we optimize our classifier and then evaluate its performance on a separate manually-scored testing corpus. Finally, we conclude the chapter with an analysis of the classifier’s performance.

3.2 Related Work

To the author’s best knowledge, automatic classification has never been applied directly to microblogs. However, a significant amount of research has been performed on applying automatic classification to traditional blogs.

Teng and Chen use a Support Vector Machine (SVM) to classify whether bloggers are interested in five different topics: computers, healthcare, politics, movies, and soccer (Teng and Chen, 2006). Durant and Smith use both SVM’s and naïve Bayes comparatively in order to classify the sentiment of bloggers in regards to President Bush’s management of the Iraq war (Durant and Smith, 2006). Gilad and Rijke
don’t classify users into specific categories. Rather, they use language models in order to match bloggers with advertisements for the purposes of displaying more relevant ads (Gilad and Rijke, 2006). Other researchers attempt to classify more personal attributes of a blogger such as their gender (Yan and Yan, 2006), their mood (Mishne, 2005), or even their personality type (Oberlander and Nowson, 2006). Finally, a number of studies exist that attempt to classify the blog itself rather than the blogger. For example, (Kolari et al., 2006) use an SVM and (Mishne, Carmel, and Lempel, 2005) use language models in order to determine whether a blog is legitimate or spam (known as “splog”).

A number of potential applications of classifying blogs were mentioned in the previous studies. Teng and Chen describe that classification can be used for monitoring public opinions, grouping bloggers into communities, and discovering customer opinions or concerns towards products (Teng and Chen, 2006). More general applications also are mentioned such as trend analysis, social psychology, and political science (Kolari et al., 2006).

3.3 Approach

Our approach into automatic microblog classification is organized as follows. In Section 3.3, we describe why we chose a naïve Bayes classifier as opposed to another classifier. In Section 3.4, we describe how we defined a microblog document while in Section 3.5, we describe our target classification set. Next, in Section 3.6 we describe how we formed our training corpus and used it to train our classifier. Our initial performance results on the training data are presented in Section 3.7. Section 3.8 follows with a discussion on three techniques we used to optimize our classifier. In Section 3.9, we describe how we formed a new corpus for testing our classifier with a presentation of the testing results in Section 3.10. Finally, in Section 3.11 we conclude the chapter with a discussion of our classifier’s performance on the testing corpus and comment on our results.
3.4 Classifier Choice

Based on our research, three primary methods were used for performing automatic classification of a traditional blog: Naïve Bayes (NB), Support Vector Machines (SVM), and Language Models (LM). Typically, only a single technique was used within a given study. However, (Durant and Smith, 2006) and (Oberlander and Nowson, 2006) both use an SVM and a naïve Bayes classifier in order to compare their relative performance. Durant and Smith report that their naïve Bayes classifier predicted the correct political sentiment 78.06% of the time compared to 75.47% for the SVM. Oberlander and Nowson also report better success with naïve Bayes. They report that naïve Bayes outperformed an SVM on 41 of their 60 classification tasks with 14 wins and 5 ties. None of the studies made any comparisons against a language model classifier. Because the naïve Bayes classifier outperformed the SVM in both of these comparisons, we chose to use a naïve Bayes classifier for our research as well. In particular, we chose an implementation very similar to the one described by Mitchell (1997). For a detailed description of our particular implementation, please refer to Chapter 2.

3.5 Document Definition

The document for a naïve Bayes classifier is the smallest unit of classification. It determines the type of input used by the classifier for determining the corresponding output classification. While a microblog is a collection of posts written by a single user, we chose to define our document as a single microblog post rather than an entire microblog. We chose this particular document definition since (1) we were interested in the level of performance attainable with such a short document type and (2) we reasoned that single post results could be aggregated for classifying an entire microblog.
3.6 Target Classification Set

For the target classification set, we chose ten categories from the top-level categories of Google Directory\(^1\). Google Directory is a hierarchical search engine provided by Google. Instead of searching the web using traditional query terms, Google Directory allows one to search using a hierarchical set of categorized hyperlinks. At the top of this hierarchy are 15 broad categories that divide a large part of the web. Since these categories are general enough to divide the web, we reasoned that they would serve as a general set of categories for dividing our microblog posts as well.

We chose to use only ten of the high-level Google Directory categories for two reasons. First, we decided ten categories were enough to demonstrate the efficacy of the classifier. Second, for some categories it was difficult to find sufficient training data. For example, it was easy to find microblogs related to Sports but rather difficult for categories such as Religion or Home. In all, we ended up with training data for the following ten categories: Arts & Entertainment, Business, Food & Drink, Computer Games, Health, Politics, Science, Sports, Technology, and World. For a more detailed description of these categories, please refer to Appendix 1.

3.7 Training Corpus

The naïve Bayes classifier is a supervised learning technique. Therefore, it requires training using a set of pre-classified documents before it is able to classify a document of unknown class. For our training corpus, we chose to use microblogs exclusively from the domain Twitter.com since it is the leading microblogging service on the web and provides a wide range of users. We began forming our training corpus by using a combination of personal and commercial Twitter accounts. All of the accounts were public and were found using Twitter Search\(^2\), a Google search engine dedicated to the domain Twitter.com. The personal accounts were maintained by a single user but dedicated to a single

---

\(^1\) [http://directory.google.com/](http://directory.google.com/)

subject. The commercial accounts, on the other hand, were maintained by a corporation or business and provided large numbers of highly relevant posts.

In order to acquire the posts, we developed an automated crawler dedicated to Twitter.com. The crawler was fed a list of named Twitter accounts as input as well as the corresponding target classification categories. Each microblog account was hand-selected such that the containing posts were only related to a single category. Our input list was tailored to provide approximately 20K posts per category. Given the list of accounts to crawl, the Twitter crawler would download the posts from each account using Twitter’s programmatic API and then write the results to a file. Furthermore, each file was annotated with an XML comment that would describe the target classification value associated with the file’s contents. Since crawling required several hours of running time, the crawler logged its progress to a file in case the crawler encountered a problem and needed to resume from a previous stopping point. Developing an automated microblog crawler for Twitter was a necessity to our work since the classifier’s training data became stale over time. This problem is known as *concept drift* and occurs because the concepts underlying a class gradually change over time (Manning, 2008). By automating the crawling of our training data, we allowed our training data to be refreshed as often as needed by simply running a program. Given our automated crawler and lists of accounts to crawl, we formed a training corpus of approximately 20K microblog posts per category for a total of 206K posts for all ten categories.

Once the training corpus was acquired, we next wrote a custom parser to extract the individual posts from the downloaded files and also to perform various forms of filtering on the posts themselves. Filtering was required since many posts contained several types of noisy features such as embedded hyperlinks or HTML code. These types of features needed to be removed before using the posts for classification since we desired only the set of English words in a post as the set of features for training
and classification. A number of filtering steps were required. First, we converted any HTML-encoded characters (e.g. “&lt”) or Unicode characters (e.g. “ሤ”) into their equivalent ASCII characters or removed them if such equivalents did not exist. Next, we removed any embedded hyperlinks (e.g. “http://...”), HTML code (e.g. “<a../a>”), headings (e.g. “NEWS.”), references (e.g. “[...]”), and tags (e.g. “<...>”). Finally, we tokenized our remaining post into a simple list of words by breaking the post along any non alpha-numeric boundaries while also removing these types of characters since they were also noisy features. Once complete, each post was reduced to a simple list of English words. Table 1 below summarizes the various types of filtering performed on the posts before they were used as input into the naïve Bayes classifier.

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing steps performed prior to classification (described below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Convert any HTML-encoded characters into ASCII.</td>
</tr>
<tr>
<td>2</td>
<td>Convert any Unicode characters (e.g. “\u24ff”) into their ASCII equivalents or remove.</td>
</tr>
<tr>
<td>3</td>
<td>Filter out any embedded URL’s (e.g. “http://”), HTML (e.g. “&lt;a../a&gt;”), headings (e.g. “NEWS.”), references (e.g. “[...]”) and tags (e.g. “&lt;...&gt;”).</td>
</tr>
<tr>
<td>4</td>
<td>Break the post into words by tokenizing it along any non alpha-numeric character boundaries.</td>
</tr>
</tbody>
</table>

After filtering and tokenizing the posts into words, the amount of training data averaged approximately 227K words per category with a standard deviation of approximately 33K words. The average number of words for a single post was only 11 words. The total number of remaining words for each target category within our training corpus is presented below in Figure 1.
Figure 1. Remaining number of words within the training corpus after preprocessing.

As seen in Figure 1, most categories contained between 200K – 250K words for training. The largest exception was for the category Politics which contained only 143K words. Differences in the number of words available for each category were due to variations in the average post length and the amount of filtering necessary. Unfortunately, Durant and Smith (2006) indicate that imbalanced like these can bias the classifier towards the more represented classes. However, rather than attempt balancing the training data to an equal number of words per category, we focused our efforts on optimizing the classifier in order to remove its sensitivity towards these types of issues.

3.8 Training Corpus Results

Before optimizing our classifier, we first determined our initial classifier’s performance on our training corpus using a standard validation technique known as stratified 10-fold cross-validation (Durant and Smith, 2006). This validation technique works as follows. First, the corpus is randomly divided into ten relatively equal partitions in both size and class distribution. Next, nine of the partitions
are used for training and the remaining partition is used for testing. This process is then repeated for each of the partitions that are held out as a testing partition. Between each iteration, the results of the classifier are measured against the training partition in terms of recall, precision, and F₁-measure as defined in Chapter 2. Finally, the results across all ten iterations are then macro-averaged together for an overall measurement of performance.

The performance results for the training corpus are shown below in Figure 2.

![Training Corpus Performance](image)

**Figure 2.** Training corpus performance using stratified 10-fold cross-validation
As seen in Figure 2, the average\(^3\) performance across all ten categories was 0.79 precision, 0.78 recall, and 0.78 F\(_1\)-measure. These results were higher than expected given that we are classifying individual posts with an average length of only 11 words. The best performing category was Sports which had precision, recall, and F\(_1\)-measure scores all equal to 0.88. The worst performing category was World which had performance scores of 0.57 precision, 0.74 recall, and 0.65 F\(_1\)-measure. This category also had the largest difference between precision and recall scores while the majority of the remaining categories exhibited a balance between these two opposing metrics. Finally, the standard deviation was 0.08 F\(_1\)-measure across all of the ten categories.

### 3.9 Optimization

Following the training of our classifier, we proceeded to optimize its features. Feature optimization is the process of finding the minimal set of features that best generalize across multiple datasets. It is important for a couple of reasons. First, Durant and Smith indicate that imbalances in the training data can bias a NB classifier towards the categories that are more heavily represented within the training data (Durant and Smith, 2006). Kalita demonstrates that by removing stop words and relatively infrequent words, a NB classifier can be made less sensitive to these types of imbalances (Kalita, 2002). Second, Manning et al. describe that NB classifiers are prone to overfitting (Manning, 2008). Overfitting is the inclusion of features that don’t generalize well across multiple datasets. Manning et al. also demonstrate that by reducing the feature set through the computation of mutual information, a NB classifier’s overall F-measure performance can be improved on unseen datasets.

We employed all three of the above feature optimizations: stop word removal, infrequent word removal, and the computation of mutual information. Stop words and infrequent words were identified

---

\(^3\) We averaged the performance values by performing a weighted average of each category. The weight of a particular category was equal to the ratio of category posts to the total number of posts in the corpus. All performance values shown in this chapter that are averages across all of the categories are weighted-averages.
by sorting our classifier’s dictionary by frequency and then inspecting the most and least frequent words, respectively. We removed the most frequent 200 stop words by hand. Removing these features had a slight positive effect on the training corpus’ performance, as seen below in Figure 3.

![Training Corpus Performance vs. Removal of Stop Words](image)

**Figure 3. Training corpus performance after the removal of stop words**

In order to remove infrequent words, we next plotted the number of remaining words within our training corpus against a minimum word frequency as shown below in Figure 4. Note the list of 200 stop words were removed prior to generating the results seen in Figure 4.
Figure 4: Number of words versus minimum word frequency for the training corpus

In Figure 4, we see the effects of Zipf’s Law as described in Chapter 2. In particular, we see that the vast majority of words occur very infrequently. For example, there are almost 80K words that occur one or more times in our training corpus but only 25K words that occur five or more times. While Figure 4 indicates a decaying exponential relationship for the frequency of words, Figure 5 indicates that performance is only affected linearly for the same set of minimum word frequencies.
A preferable method to the method we used here is to use another tuning corpus that is separate from the training and testing corpora as described by Mitchell (1997). Using a tuning corpus, one would typically remove features from the training data until the performance on the tuning corpus began to degrade. We leave this optimization as future work and describe it in Chapter 8.

Figure 5: Average performance versus minimum word frequency for the training corpus

For selecting our minimum word frequency, we chose to use the middle of the curve in Figure 4 by keeping all words that occurred three or more times. This reduced the number of words in our classifier’s dictionary from 79K words to approximately 34K words. While we reduced the number of words by more than half, it only affected the classifier’s average $F_1$-measure performance by less than one-half of a percentile on the training corpus. We reasoned that this conservative approach⁴ removed a majority of the noisy features without being overly aggressive.

The last optimization technique we applied was the computation of mutual information. Mutual information, as described in Chapter 2, quantifies the amount of information the presence or absence of a word contributes towards knowing the target classification. We applied this optimization technique as

---

⁴ A preferable method to the method we used here is to use another tuning corpus that is separate from the training and testing corpora as described by Mitchell (1997). Using a tuning corpus, one would typically remove features from the training data until the performance on the tuning corpus began to degrade. We leave this optimization as future work and describe it in Chapter 8.
follows. First, we computed the amount of mutual information between a word and the target classification set for each word in our dictionary. Next, we sorted the words by their mutual information value and then chose the top \( n \) words with the most mutual information. In order to choose a value of \( n \), we plotted the classifier’s performance on the training corpus for different values of \( n \). These results are presented below in Figure 6. Again, please note that the application of mutual information was performed after removing stop words and infrequent words.

![Performance vs. Top N Mutual Information Words](image)

**Figure 6: Average performance versus minimum word frequency for the training corpus**

As seen in Figure 6, the classifier’s performance can be negatively affected if feature selection is too aggressive. We again decided to be conservative and chose the top 15K words with the most mutual information. This resulted in reducing the number of remaining words in the dictionary from \(~34K\) to 15K while only reducing the average F1-measure performance by less than a percentile on our training corpus.
At the completion of optimizing the classifier using all three optimization techniques, we reduced the classifier’s dictionary size from its original size of ~80K words to a final size of 13K words. This large reduction in word count only reduced our average F$_1$-measure performance on our training corpus by 0.98% for a final F$_1$-measure of 0.78. While our optimizations lowered the average performance on our training corpus, the intent was to improve the classifier’s performance on an unseen corpus by reducing the amount of overfitting. In the next section, we evaluate the classifier’s performance on such a dataset.

3.10 Testing Corpus

Following the training and optimizing of our NB classifier, we composed an additional testing corpus in order to understand how the classifier would perform on a dataset that was independent from training. Furthermore, rather than hand-classifying this testing corpus ourselves, we used three groups of volunteers. The microblogs for the testing corpus were found by using the Twitter Public Timeline\(^5\) which is a running list of the most recently made public posts within Twitter. We selected the first 50 non-commercial microblog accounts sequentially and blindly from the public timeline that contained at least 20 posts and were in English. For each of the 50 microblogs, we chose only the first 20 posts for a total of 1000 microblog posts within our testing corpus.

The list of 1000 posts was given to the three groups of volunteers along with some instructions. The instructions (shown in Appendix A) indicate the available categories for categorizing the posts as well as some examples. If a post applied to one or more of the ten target categories, it was marked with each corresponding category. If a post applied to some other category, it was marked with “NA”. Finally, if a post was not comprehensible, it was marked with a question mark. Once the testing corpus was independently scored three times, the posts that had at least two-way agreement between the groups

\(^5\) [http://twitter.com/public_timeline](http://twitter.com/public_timeline)
of volunteers (and had exactly one of the ten target classification categories) were chosen for comparison against the automatic NB classifier. Any remaining posts were unused. In total, we had 432 usable posts with at least two-way agreement and 287 usable posts that had three-way agreement from our original list of 1000 testing posts. Figure 7 below displays the breakdown of usable posts with at least two or three-way agreement.

![Number of Usable Documents: 2-Way Agreement](image1)

![Number of Usable Documents: 3-Way Agreement](image2)

**Figure 7: Distribution of usable testing posts with at least two or three-way agreement**

In Figure 7, we observe that the categories Arts, Food, and Technology compose the bulk of the available testing posts while some categories, particularly Computer Games, Politics, and World, were largely under-represented. Since many of the categories did not contain a sufficient distribution of posts, we analyzed the testing corpus in order to understand which categories were missing. In particular, we analyzed the set of “NA” posts with at least two-way agreement since these posts were marked as belonging to external categories. In order to categorize these “NA” posts, we manually clustered them using an open set of categories. Figure 8 displays the distribution of these manually
classified posts in addition to the posts from our original set of ten target classes. The new categories are shown in blue while our existing categories are shown in yellow.

**Figure 8. Open distribution of categories for the testing corpus (target classes shown in yellow)**

In Figure 8, we display 18 new categories in addition to our original ten target categories and our unknown category ("?"). Recall that we originally created our set of ten target classes based on the high-level categories found within Google Directory. As seen in Figure 8, four of these categories (Food & Drink, Technology, Arts & Entertainment, and Health) are very popular Twitter categories. However, several are not (World, Politics, and Computer Games). Most notably, our target classes do not include the most popular Twitter topic, Social, which represents the casual conversation and event planning between friends.

### 3.11 Results

Following the optimization of our classifier and the composition and manual scoring of our testing corpus, we next proceeded to evaluate the classifier’s performance on the testing corpus. The performance of our naïve Bayes classifier on our testing corpus was measured twice: once for the posts
with at least two-way agreement and again with the posts with three-way agreement. We tested each way in order to understand how much ambiguity affected the classifier. For the testing corpus with at least two-way agreement, our average $F_1$-measure weighted by the number of posts in each category was 0.66 as seen below in Figure 9.

![Testing Corpus Performance: 2-Way Agreement](image)

**Figure 9. Classifier performance on testing corpus with at least two-way agreement**

The category Food scored the highest $F_1$ measure of 0.80. The top four most represented categories (Arts & Entertainment, Food & Drink, Health, and Technology) also had the highest $F_1$ measures of 0.61, 0.80, 0.64, and 0.63, respectively. Many of the lowest scores were from the categories with the fewest numbers of usable posts. For this reason, it is difficult to speculate on their accuracy.

For the testing corpus with three-way agreement, our weighted-average $F_1$ measure improved to 0.74, as seen below in Figure 10. Most categories improved or maintained their same level of performance with the exceptions of Business, Science and World. However, when requiring three-way agreement, the number of Business posts decreased from 10 to 4 and the number of Science and World posts decreased to zero.
Figure 10. Classifier performance on testing corpus with three-way agreement

The improvement in performance with the three-way testing corpus reflects the amount of uncertainty in classification within the two-way agreement testing corpus. Furthermore, it also demonstrates that the NB classifier is not strongly sensitive to uncertainty since its performance only worsened by 8% from the three-way agreement corpus to the two-way agreement corpus.

Finally, since Figures 9 and 10 indicate the NB classifier’s performance after optimization, we could not determine whether or not our three optimization techniques improved the classifier’s performance on the testing corpus. In order to make this assessment, we repeated our evaluation on the testing corpus without optimizing the classifier. Table 2 summarizes our results.

<table>
<thead>
<tr>
<th>Post Agreement Level</th>
<th>Classifier Optimized?</th>
<th>Weighted-Average F$_1$-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-Way or Three-Way</td>
<td>No</td>
<td>0.62</td>
</tr>
<tr>
<td>Two-Way or Three-Way</td>
<td>Yes</td>
<td>0.66</td>
</tr>
<tr>
<td>Three-Way</td>
<td>No</td>
<td>0.71</td>
</tr>
<tr>
<td>Three-Way</td>
<td>Yes</td>
<td>0.74</td>
</tr>
</tbody>
</table>

Table 2. Results of classification optimization on testing corpus.
In Table 2, we indicate our weighted-average F$_1$-measure performance for our testing corpus for all four combinations of agreement level and optimization. As the results indicate, our optimization of the classifier by removing stop words, infrequent words, and selecting words with high mutual information improved the results on our testing corpus – though only by a modest amount. In particular, our average F$_1$-measure performance increased by 4% for the set of posts with at least two-way agreement and improved by 3% for the posts with three-way agreement. While our improvements in performance were less than expected, our overall results are encouraging and indicate a trained and optimized naïve Bayes classifier can correctly recognize our target classes 74% of the time on posts with three-way agreement of target class. Furthermore, these results are only 4% worse than our training corpus performance which had an average F$_1$-measure performance of 0.78. Therefore, it is unlikely we could optimize the classifier’s performance much further.

3.12 Conclusions

In this chapter, we applied the naïve Bayes classifier described in Chapter 2 to the relatively new domain of microblogs. In particular, we trained and optimized a naïve Bayes classifier to recognize ten different interest categories using a microblogging corpus collected from Twitter. Our results indicate that though a single microblog post contains only 11 words, on average, it can be classified into ten different interest categories with relative certainty. In particular, we find that microblog posts with a well-agreed upon target class can be automatically classified into one of ten categories with an average F$_1$-measure performance of 74%.

Finally, our results indicate that popular microblogging topics on Twitter only partially align with many of the high-level categories found within Google Directory. Out of the ten target classes we chose, only four were very popular within our testing corpus: Food & Drink, Technology, Arts & Entertainment, and Health. Three of the other classes (Sports, Science, and Business) were moderately popular, and the
remaining classes (World, Politics, and Computer Games) were largely unpopular. Last, we found that the category Social – the casual conversation between friends – accounts for the majority of tweets filling Twitter’s pages.
4 Microblog Summarization and its Evaluation

4.1 Introduction

In order to help users sort through the vast number of tweets that occur each day, Twitter has added a number of tools that help users find the most important topics and their associated tweets. Twitter’s homepage\(^6\) displays these important topics – which are just short phrases such as “New Moon”, “Glee”, or “Janet Jackson” – for three different ranges of time in order to see what topics are popular at the moment, during the present day, or over the last week. For most topics, users are forced to read through the related posts (by clicking on the topic) in order to try and understand why a topic is trending. This process is tedious and error prone as returned posts are prioritized only by recency. Therefore, for a given topic, users are likely to encounter spam, posts in other languages, rants, and other sources of misinformation. To help users further, Twitter has partnered with the third-party website WhatTheTrend\(^7\) in order to provide definitions of trending topics. WhatTheTrend allows users to manually enter descriptions of why a topic is trending. While the idea is good in theory, in practice WhatTheTrend also suffers with spam and rants as users are free to enter whatever definition they prefer for a trending topic. A quick look at the history of definitions for a few topics on WhatTheTrend often shows definitions oscillating between users trying to spam the website and other users trying to provide accurate information. The biggest drawback to the site is that definitions are entered by hand and not automated. Therefore, there is often some lag time before a new trending topic is defined by a user. While WhatTheTrend is a step in the right direction, a better approach is an automated technique that summarizes trending topics in real-time.

In this chapter, we begin our approaches in automatic microblog summarization by laying the groundwork for our development. In particular, we discuss relevant prior work in the field of automatic

\(^6\) [http://www.twitter.com](http://www.twitter.com)

\(^7\) [http://www.whatthetrend.com](http://www.whatthetrend.com)
text summarization while also discussing its associated challenges. Following this, we give a brief introduction into the various approaches we applied towards this problem which are presented in detail in Chapters 5, 6, and 7. The remainder of this chapter is dedicated to describing how we will evaluate our approaches in summarization. We present two metrics used during the Document Understanding Conferences, a conference dedicated to furthering the state of the art in summarization, and then discuss how we will apply them to our problem. Finally, this chapter concludes with the formation of a set of manual summaries which are used to evaluate the performance of our automated summaries.

4.2 Related Work

Automatically summarizing microblog topics is a new area of research and to the author’s best knowledge, approaches have not been made available in the published literature. However, summarizing microblogs can be viewed as an instance of the more general problem of automated text summarization. Automated text summarization can be defined as the problem of automatically generating a condensed version of the most important content from one or more documents for a particular set of users or tasks (Lin, 2009). Since microblogs are composed primarily of text, it is important to understand what progress has been made in the area of text summarization in order to understand its potential relevance for summarizing microblogs.

4.2.1 A Brief History of Summarization

Fortunately, the field of automatic text summarization has been around for quite some time. As early as the 1950’s, IBM scientist Hans Peter Luhn was experimenting with methods for automatically generating extracts of technical articles. Luhn was interested in reducing the amount of manual effort and expertise involved in this task as well as improving their “consistency and objectivity” (Luhn, 1958). Following Luhn’s work, Edmundson developed additional techniques for summarizing a more diverse corpora of documents for the purpose of helping users “screen” or evaluate documents for further
reading. Early research in text summarization focused primarily on simple statistical techniques that relied upon lexical features such as word frequencies (e.g. Luhn, 1958) or formatting clues such as titles and headings (e.g. Edmundson, 1969). Later work integrated more sophisticated approaches such as machine learning (e.g. Kupiec et al., 1995), natural language processing (e.g. Barzilay and Elhadad, 1997), and hybrid approaches (Neto et al., 2002). In most cases, text summarization is performed for the purposes of saving users time by reducing the amount of content having to be read (e.g. Luhn, 1959; Edmundson, 1969). However, text summarization has also been performed for other purposes such as reducing the number of features required for classifying (e.g. Kolcz et al., 2001) or clustering (e.g. Ganti et al., 1999) documents. While interest in automated text summarization began early, it did not become an active area of research until the 1990’s with the introduction of the World Wide Web (Lin, 2009). With the growth of the Web, interest grew to improve summarization while also summarize new forms of documents such as web pages (e.g. Mahesh, 1997) and blogs (e.g. Zhou and Hovy, 2006; Hu et al., 2007). Most recently, interest has shifted from single document summarization to multiple document summarization thanks in part to annual conferences such as the Text Analysis Conference (TAC) that aim to further the state of the art in summarization by providing large test collections and common evaluation of summarizing systems (TAC, 2010).

4.2.2 Approaches in the Literature

In general automatic text summarization systems can be divided into one of two categories: extractive or abstractive. Extractive systems are designed to summarize a document (or set of documents) by selecting the most salient sentences from the document(s) and then concatenating these sentences together. Sentence selection is usually performed by weighing the sentences in some manner and then choosing the set of sentences with the most weight. Sentence weighting schemes in the literature can be broadly classified into three types: feature-based (e.g. Luhn, 1958; Edmundson, 1969), lexical chain-based (e.g. Barzilay and Elhadad, 1997) and graph-based (e.g. Mihalcea and Tarau, 2004).
In feature-based weighting, the sentences are scored relative to a set of features such as word frequencies, the position of the sentence in the overall document (e.g. in the first or last paragraph), or the inclusion of certain cue words (e.g. “In conclusion”). In lexical-chained based weighting, sentences are scored based upon the strength of lexical chains in which they belong where a lexical chain is a sequence of words that are related such as through synonymy or hyponymy. Finally, in graph-based approaches, a graph is constructed that represents the various text units as vertices and the semantic relations between these vertices as edges. After the graph is constructed, a graph-ranking algorithm is applied to determine the scores of each vertex. Finally, the vertices are sorted by their scores and the text units associated with the top scoring vertices are used as a resulting summary.

Finally, abstractive systems take an entirely different approach towards summarization. Instead of selecting salient sentences for use as the summary, an abstractive approach generates its own summary text through a detailed linguistic analysis and transformation of the source text. These types of systems attempt to represent either the structure or concepts of a source text using parse trees or text knowledge bases, respectively. From these internal representations, an abstractive system would attempt to condense these representations and then transform them into a fluent summary using natural language generation or pre-populated templates (Hahn and Mani, 2000). Because abstractive systems often require domain-specific knowledge, their application is more restrictive than extractive systems. This limitation in addition to their increased complexity has resulted in few abstraction-based summarization systems in the literature in the past decade or so (Jones, 2007).

4.2.3 Problem Dimensions

Before considering the most recent approaches to automated text summarization, it is important to first understand the various dimensions to the problem. Jones divides these dimensions into three
broad categories or “factors”: input, purpose, and output (Jones, 2007). The following is a list of many of the factors she describes.

- **Input factors** describe the source of the information to be summarized and cover dimensions such as the language, genre, register (i.e. linguistic style), length, structure (e.g. location, headings, citations, hyperlinks, etc.) and number of sources (single document vs. multiple documents).

- **Purpose factors** describe the intended purpose of the summary and cover elements such as its audience (e.g. general or specific) and intended use. There can be many different uses for a summary which Hahn and Mani categorize into three primary uses: indicative, informative, or critical (Hahn and Mani, 2000). **Indicative** summaries are intended to help readers decide whether or not a document is relevant for further study (e.g. abstracts). **Informative** summaries are intended to be whole replacements of a document such that a reader would only be required to read the summary in lieu of the document. These types of summaries may contain only the relevant facts contained in the document that relate to a specific query. Finally, **Evaluative** summaries are intended to provide some sort of commentary or opinion on the document in addition to summarizing the content (e.g. a book review).

- **Output factors** describe the generated summary output and cover dimensions such as style, reduction/coverage (i.e. length of the summary relative to the source), coherence (i.e. grammatical correctness), and derivation. Derivation is an important concept and relates to whether the summary consists of literal excerpts from the original source document(s) or paraphrases. These types of derivation are generally known as either extractive or abstractive summaries, respectively, and have a great deal of influence on the overall design of the summarizing system (Lin, 2009).
4.2.4 Challenges Described in the Literature

While Jones has described several of the factors to consider when designing an automated summarizing system, there are also many difficulties associated with these factors mentioned in the literature. One challenge of particular consequence is the output factor of coherence. Coherence is a measure of the fluency of a summary in terms of it obeying the rules of grammar, logic, and discourse. Coherence can be difficult to achieve for both extractive and abstractive systems, though for more so for extractive systems. For extractive systems, coherence can be lost if key concepts are spread across multiple sentences but only a subset of those sentences are chosen for extraction (Lin, 2009). Lin also mentions similar issues with anaphoric and temporal references. Anaphora are the usage of a pronoun or similar word instead of repeating a word used earlier. If an extractive system chooses a sentence that contains anaphora without also choosing the prior sentence that defines the anaphora, the anaphoric reference will be lost. This can result in either an incoherence summary where the term is never defined or a worse situation where the term is defined in a new and incorrect context (Lin, 2009). For abstractive systems, the main challenges with coherence relate to the system’s ability to generate fluent output without relying upon the sentence structure of the source documents. For these types of systems, they must integrate outside knowledge such as the rules of natural language in order to translate their internal representation of the summary into a fluent and coherent summary (Hahn and Mani, 2000).

Other challenges mentioned in the literature are centered around several of the input factors described by Jones. Barzilay and Elhadad warn that systems that rely too heavily upon the availability of certain cue words or structural elements such as location may have varying levels of performance based on the genre of the documents being summaries (Barzilay and Elhadad, 1997). For example, Edmundson’s “Location Method” relies upon the occurrence of predefined heading words such as “Introduction”, “Purpose”, and “Conclusions” that may or may not be present within all types of
documents. Other difficulties are mentioned by Lin who describes some of the problems of summarizing both single and multiple document sources. For single documents, Lin mentions that it can be difficult to improve upon simple techniques such as methods that use abstracts, executive summaries, or particular locations of paragraphs as a summary (Lin, 2009). For example, Lin and Hovy have empirically identified that many genres of documents have the most relevant information in predictable locations (Lin and Hovy, 1997). News articles, for instance, are written such that the most important information is typically placed in the beginning (Zhou and Hovy, 2006). For multiple document sources, both (Hahn and Mani, 2000) and (Lin, 2009) describe problems with issues of contradiction and redundancy. In these cases, summarizing systems must develop methods for identifying both similarities and differences between the source documents and resolve them into the output summary (Lin, 2009).

4.3 Problem Description

Before describing the approaches used to summarize microblogs, we first present a description of the problem. As alluded to earlier, many microblogging services such as Twitter provide a list of trending topics for different periods of time such as during the current day or week. For any of these trending topics, a user can perform a search and retrieve a list of posts that all contain the trending topic phrase. Likewise, a user can also perform the same type of search on any other phrase if interested in seeing what users are blogging about a particular topic or phrase. The difficulty in interpreting the results, however, is the returned posts are only sorted by recency, not relevancy. Therefore, the user is forced to manually read through the posts in order to understand what users are primarily saying about a particular topic. The motivation of the summarizer is to automate this process and generate a more representative summary in less time and effort.
Most search engines built into microblogging services only return a limited number of results when querying for a particular topic or phrase. This is performed in order to reduce the demand on the service. For example, Twitter only returns a maximum of 1500 posts for a single search phrase. Given the fact that we are using Twitter as the microblogging service for evaluating our approaches and that we want to summarize a set of posts for a particular topic, we can now formalize our problem. Our problem description is as follows:

*Problem Description: Given a set of posts that are all related by containing a common search phrase (e.g. a topic), generate a summary that best describes the primary “gist” of what users are saying about that search phrase.*

### 4.4 Selected Approaches

Given the approaches found in the literature and their associated challenges, the first decision in developing a microblog summarization algorithm was to choose whether to use an abstractive or extractive approach. While both of these approaches have their strengths and weaknesses, an extractive approach was chosen since its methodologies more closely related to the structure and diversity of microblogs. For example, abstractive approaches are most beneficial in situations where high rates of compression are required – such as summarizing multiple long documents for small devices such as mobile phones. However, microblogs are the antithesis to long documents. Since they are so short, microblog posts are already highly condensed leading to the greater potential of finding an extract to serve as the summary. Furthermore, abstractive systems usually perform best in limited domains since they require outside knowledge sources such as grammars, parsers, and ontologies. These approaches might not work so well with microblogs since they are unstructured and diverse in subject matter. Extractive techniques, on the other hand, are known to better scale with more diverse domains (Hahn and Mani, 2000).
In the end, we implement several different types of extractive algorithms. First, in order to help establish a baseline of performance using our chosen evaluation metrics (discussed later in this chapter), we implemented two preliminary algorithms based on some very simple techniques. First, we chose a purely random approach towards summarization that just selects a random sentence or post as the selected summaries. Second, we also used another slightly less naïve approach that used both the longest and shortest instances of either a post or sentence as the selected summaries. While these approaches are all very naïve, we needed to establish some baseline performance values for our chosen metrics that we could use to judge the relative effectiveness of our primary algorithms. Without such baselines, it would have been impossible to decipher their results.

Finally, once we had established our performance baselines, we also developed and implemented two primary algorithms. First, we created the novel Phrase Reinforcement algorithm that uses a graph to represent the overlapping phrases between a set of related microblog sentences. This graph allows the generation of one or more summaries at different rates of compression and is discussed in detail in chapter 6. Finally, we developed another primary algorithm based on a well established statistical methodology known as TF-IDF. This approach was chosen in order to determine whether or not we could improve upon our Phrase Reinforcement algorithm using a more traditional approach towards summarization. The TF-IDF algorithm is discussed at length in chapter 7.

Last, while the various summarization algorithms are discussed in depth in future chapters, we describe their evaluation and their evaluation setup in this chapter since it is common to all of our approaches. In each of the later chapters, we use the described evaluation metrics as means to rate the performance of the various techniques and then summarize and contrast their results collectively in chapter 8.
4.5 Evaluation

Evaluating the results of an automated summarization system is not a straightforward task. Since there may be many valid interpretations of the “ideal” summary, there is not a definitive standard against which one can compare the results from an automated system. Because of this limitation, summary evaluation is generally performed using one of two methods: intrinsically, or extrinsically. In an intrinsic evaluation, the quality of the summary is judged based on direct analysis using a number of predefined metrics such as grammaticality, fluency, or content (Lin, 2009). In contrast, extrinsic evaluations take a more purpose driven approach to evaluating summaries. Instead of directly evaluating a summary for its quality, extrinsic evaluations measure how well a summary enables a user to perform some form of task. For example, an extrinsic evaluation may measure how well a user can determine whether or not a document is relevant to a topic or answer key questions after reading the summary (Hahn and Mani, 2000). Since extrinsic evaluations require some form of scenario building, they are often more difficult to conduct and not as prevalent in the literature compared to intrinsic methods.

In order for an intrinsic evaluation to measure the quality of an automated summary, a common approach is to create one or more manual summaries and then to compare the automated summaries against the manual summaries. For example, during the Document Understanding Conference of 2002, the set of manual summaries were compared to a set of automated summaries, one pair at a time. For each pair, evaluators marked which sentences of the manual summaries overlapped in content (i.e. meaning) with sentences from the automated summaries on a five-level scale. The manual summaries were then scored relative to their overall coverage of content with the automated summaries. In addition to measuring for content, the DUC also scored the automated summaries for grammaticality, cohesion, and coherence in order to further measure their quality (Lin and Hovy, 2003).
In general, manual evaluations are often used because they can deal with problems such as different wordings between the manual and automated summaries. However, one problem with manual evaluation is the amount of human error introduced. For example, Lin and Hovy note that during the DUC 2001, 18% of the data for the single document evaluation contained multiple judgments (Lin and Hovy, 2002). Multiple judgments are where a different performance score is given to the same pair of sentences being compared by the same human judge. In order to help mitigate this problem, there have been attempts to automate the evaluation process in addition to the generation of summaries.

One popular automatic evaluation metric that has been adopted by the Document Understanding Conference since 2004 is ROUGE. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation (Lin, 2004) and is a suite of metrics that can be used to automatically measure the similarity between an automated summary and a set of manual summaries. ROUGE is composed of four fundamental metrics (ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S) which measure different overlapping units such as n-grams, word sequences, and word pairs (Lin, 2004). One of the simplest ROUGE metrics is the ROUGE-N metric:

$$ROUGE - N = \frac{\sum_{\text{Summaries}} \sum_{n-grams} \text{Match}(n - gram)}{\sum_{\text{Summaries}} \sum_{n-grams} \text{Count}(n - gram)}$$

(30)

In this definition, $n$ represents the length of the n-grams, $\text{Count}(n - gram)$ represents the number of n-grams in the manual summary, and $\text{Match}(n - gram)$ represents the number of co-occurring n-grams between the manual and automated summaries.

ROUGE-N was inspired from a similar metric named BLEU which is used to automatically evaluate machine translations (see Papineni et al., 2002 for a description of BLEU). While BLEU is fundamentally a precision-based metric, ROUGE-N is a recall-oriented metric since it compares the
number of matching n-grams with the total number of n-grams within the manual summaries. However, ROUGE-N can easily be converted into a precision-based metric by redefining \( \text{Count}(n - n\text{gram}) \) to be the number of n-grams within the automated summaries. This way, ROUGE-N can be used for computing both recall and precision between a pair of summaries.

For each of the different forms of ROUGE, Lin in (Lin, 2004) performed a number of evaluations in order to understand how well ROUGE’s results correlate with the human judgments made during the DUC of 2001, 2002, and 2003. One result of particular consequence for our work is his comparison of ROUGE with the very short summary task of DUC 2003. In this task, systems were required to create a headline-like summary of around 10 words for a single document. Lin found ROUGE-1, ROUGE-L, ROUGE-W, and ROUGE-SU4 and 9 to correlate highly with human judgments for this task. He also found that higher-orders of ROUGE (e.g. \( N > 1 \)), word stemming and stop word removal decreased correlation (Lin, 2004). Since this task is very similar to our task of creating short microblog summaries, we decided to implement ROUGE-1 as one of the metrics for evaluating our results.

Finally, since we wanted some certainty that ROUGE-1 was correlating with a human-based evaluation of our automated summaries, we decided to also implement one of the manual metrics used during the DUC of 2002. This way, we could compare our manual and automated evaluations in order to judge for ourselves whether or not they correlate. For this metric, we chose to use the Content metric from DUC which asks a human judge to measure how completely an automated summary expresses the meaning of a human generated summary. The measure of completeness is ranked according to five different levels and asks whether the automated summary expresses the meaning of none, hardly any, some, most, or all of the manual summary (Lin and Hovy, 2003). We adopted the same scale for our comparisons as well.
4.6 Evaluation Setup

Given our decision to use both ROUGE-1 and the Content metric from the Document Understanding Conference, we next proceeded to define the data we would use for our testing. For collecting our testing data, we used the following approach. For five consecutive days, we collected the top ten currently trending topics from Twitter’s home page at roughly the same time every evening. For each of these trending topics, we downloaded the maximum number of available posts that contained the trending topic phrase from Twitter using its Search API (see Chapter 1 for the different API’s of Twitter). Twitter, in turn, returned approximately 1500 posts for each of these topics. Therefore, at the end of the five days, we had collected 50 trending topics with a set of 1500 posts for each topic.

Next, since many of the posts collected were either spam or contained various noise features, we performed several forms of preprocessing in order to filter the posts into a usable form. These steps were performed consistently before the application of any of the above summarization algorithms and are summarized below.

5. Convert any HTML-encoded characters into ASCII.
6. Convert any Unicode characters (e.g. “\u24ff”) into their ASCII equivalents or remove.
7. Filter out any embedded URL’s (e.g. “http://”), HTML (e.g. “<a.../a>”), headings (e.g. “NEWS:”), references (e.g. “[...”), tags (e.g. “<...>”), and retweet phrases (e.g. “RT” and “@AccountName”).
8. Discard the post if it is spam.
9. Discard the post if it is not in English.
10. Discard the post if another post by the same user has already been acquired.
11. Reduce the remaining number of posts by choosing the first 100 posts.
12. Break the post into sentences.
13. Break the sentences into unigrams.
14. Detect the longest sentence that contains the topic phrase.

<table>
<thead>
<tr>
<th>Table 1. Preprocessing steps performed prior to summarization (described below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>As seen in the table above, a fair amount of preprocessing is required prior to summarization. These steps warrant some explanation. Steps 1 and 2 replaced any HTML-encoded characters (e.g. “&amp;lt”) or Unicode characters (e.g. “\u1224”) with their ASCII equivalents or removed them if such...</td>
</tr>
</tbody>
</table>
equivalents didn’t exist. Step 3 was performed in order to remove any extraneous information not relevant to the summarization such as hyperlinks to outside sources of information. Following this step, posts were composed of only ASCII text. Next, steps 4 and 5 were performed to filter out any posts that weren’t relevant to the search topic. To remove spam, we trained a custom naïve Bayes classifier from our earlier work to recognize spam. We also applied a second spam-detecting heuristic based on an observation we made: Twitter spam often contains multiple trending topic phrases in order to encourage the spam to be returned during a query on these phrases. Therefore, for any post that contained more than three trending topic phrases, we simply removed it. To remove non-English posts, we performed a second heuristic. Using an English dictionary containing approximately one-quarter of a million words, we removed any posts that didn’t contain at least 40% of its words in the dictionary. Finally, step 6 was performed to prevent a single user from flooding and attempting to dominate a topic and step 7 was performed in order to reduce the remaining number of posts into a manageable size for manual summarization.

Steps 8-10 were performed to transform the remaining set of posts into forms more easily processed by our summarization algorithms. Step 8 was performed using two techniques. First, we used an open source sentence disambiguation tool named SharpNLP for detecting sentence boundaries. Second, since this tool didn’t distinguish sentence boundaries from newline characters (e.g. “\n”), we wrote some custom logic to perform this ourselves. Next, step 9 was performed by breaking the sentences along any non-alpha-numeric boundaries as well as removing these types of characters since they were generally too noisy. Last, step 10 was performed in order to help accommodate some of our evaluative summarization algorithms that only processed a single sentence.

8 http://www.codeplex.com/sharpnlp
4.7 Manual Summary Generation

Following the application of the above preprocessing steps above for our testing data, we next proceeded with generating manual summaries for each of our 50 testing topics. The manual summaries were required since our two chosen evaluation metrics, namely ROUGE-1 and Content, both require manual summaries in order to score or evaluate the automated summaries.

For generating the manual summaries, we used the following approach. Using the help of two volunteers, we asked each volunteer to generate a complete set of 50 manual summaries for all of the topics. The instructions to the volunteers were intentionally left open-ended and subjective. The volunteers were instructed to generate the “best” summary possible in 140 characters or less while using only the information contained within the posts (i.e. no outside information was permitted). The length restriction was imposed since the ROUGE-1 metric is most effective when comparing summaries at the same rates of compression. Since our automated summaries were limited to 140 characters, we imposed the same length restriction on the manual summaries. Finally, Table 2 below displays an example of the first ten manual summaries generated by our two volunteers.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Manual Summary 1</th>
<th>Manual Summary 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#MM</td>
<td>#mm: Users post about their favorite song, band, or line. Same as #MusicMonday</td>
<td>It's Music Monday on Twitter!</td>
</tr>
<tr>
<td>#BeatCancer</td>
<td>Every retweet of #BeatCancer will result in 1 cent being donated towards Cancer Research.</td>
<td>Tweet #beatcancer to help fund cancer research</td>
</tr>
<tr>
<td>Halloween</td>
<td>Everyone rushes to decide what to dress up as for Halloween.</td>
<td>People are excited about Halloween costumes and parties</td>
</tr>
<tr>
<td>Balloon Boy</td>
<td>“Balloon Boy” is a hoax; he was never in a balloon. He should be named &quot;Attic Boy&quot;.</td>
<td>People are tired of hearing about the Balloon boy hoax. Called Attic boy instead.</td>
</tr>
<tr>
<td>#musicmonday</td>
<td>Everyone is posting their current favorite song, band, and lyrics for #MusicMonday.</td>
<td>Users post songs for Music Monday</td>
</tr>
<tr>
<td>Gossip Girl</td>
<td>Gossip Girl's fans hope that Chuck and Blair make up their relationship by the next episode.</td>
<td>Chuck and Blair's relationship in question after Gossip girl episode</td>
</tr>
<tr>
<td>Kelly Clarkson</td>
<td>Between Taylor Swift and Kelly Clarkson, which one do you prefer.....?</td>
<td>Taylor Swift v. Kelly Clarkson</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><strong>Broncos</strong></td>
<td>Denver Broncos versus the San Diego Chargers. 3 returns for touch downs.</td>
<td>3 kickoff returns for touchdowns in first half of the Broncos v. Chargers game, Broncos wear throwbacks</td>
</tr>
<tr>
<td><strong>Eddie Royal</strong></td>
<td>Eddie Royal scores a touch down after a 71 yard return.</td>
<td>Eddie Royal and Darren Sproles return kickoffs and punt returns for touchdowns in Denver v. Chargers game</td>
</tr>
<tr>
<td><strong>Dodgers</strong></td>
<td>Dodgers are leading 4-2 against the Phillies in Baseball.</td>
<td>The Dodgers have a 4-2 lead over the Phillies late in the game. LA fans cheer on the Dodgers and Chargers</td>
</tr>
</tbody>
</table>

**Table 2. Example manual summaries for our first ten testing topics**

### 4.8 Manual Summary Evaluation

In Table 2 above, we see that the manual summaries generated by our two volunteers are semantically very similar to one another but in many cases have different lengths and word choices. Recall that the ROUGE-1 metric compares an automated summary with one or more manual summaries by comparing the amount of overlapping unigrams between the various summaries. Because of the way in which ROUGE-1 is computed, it will be difficult for an automated summary to achieve a high amount of overlap with both manual summaries at the same time. Thus, our ROUGE-1 performance will be somewhat limited by the amount of overlap between the two manual summaries. In order to characterize this amount of overlap, we use the ROUGE-1 metric for comparing the manual summaries against one another. Furthermore, we do the same using the Content metric in order to understand how semantically similar the two summaries are as well. By evaluating our two manual summaries against one another, we help establish some practical upper-limits of performance for our automated summaries since it is unlikely they will be able to perform much better. These results in addition to the results of the preliminary algorithms collectively establish a range of expected performance for our primary summarization algorithms.

In order to measure the ROUGE-1 performance of the two sets of manual summaries, we compared the summaries against one another bi-directionally by assuming either set was the set of
automated summaries. Since ROUGE-1 compares co-occurring unigrams between summaries, we needed to first tokenize our summaries into unigrams. For this, we decided to break the summaries along any non-alpha-numeric boundaries as well as remove these types of characters since they were noisy features. Furthermore, no attempt to remove stop words or stem words was performed since Lin indicated that these attempts made correlations worse in his evaluation (Lin, 2004). Next, we applied both a precision and recall-based version of the ROUGE-1 formula in order to generate a pair of scores for each testing topic. Next, in order to combine the scores, we averaged the precision and recall scores independently across the different topics and then combined the average precision and average recall scores together using F-measure. These steps resulted in a single set of average scores for both sets of manual summaries.

Next, in order to generate the Content performance, we performed a similar experiment by asking a volunteer to manually compare the summaries against one another for their content. In particular, we asked the volunteer to compare each set of manual summaries for how well one summary expressed the meaning of the corresponding manual summary. The volunteer marked each topic with a score from 1 to 5 which corresponded to the content choices that ranged from “none” to “all”, respective of order. These scores where then averaged for all of the fifty topics producing a single average content score for both sets of manual summaries.

Finally, the average results for computing the ROUGE-1 and Content metrics on the manual summaries are shown in Figures 1 and 2 below.
In Figure 1, we can see that the Manual 1 summaries overlap with the Manual 2 summaries with a recall of 0.37 and a precision of 0.31. However, the Manual 2 summaries have just the opposite recall and precision. Since the two manual summaries are each scored relative to another, their recall and
precision scores are symmetric. Furthermore, since the F-measure is the harmonic mean of the two scores, the F-measures are equal for both manual scores at 0.34. In order to produce a single set of performance scores from which we can compare our automated summaries, we average the two manual summary scores to produce the results seen in the columns named “Manual Average”. Our average manual ROUGE-1 scores were 0.34 for F-measure, recall, and precision.

Next, Figure 2 displays the average content overlap between the two sets of manual summaries as judged by our human volunteer and an average of these two results. It appears that our Manual 1 summaries contain slightly more content than our Manual 2 summaries since it scored higher (4.38 as opposed to 4.06). However, both scores are close to their average of 4.22 indicating that the manual summaries express most of the meaning of one another as expected.

Finally, it is interesting to note the average summary length of the two sets of manual summaries since our automated summaries will be compared – token for token – with these summaries. In Figure 3 we see the average word length of the manual summaries as well as their combined average length. The Manual 1 summaries had an average length of 11 words while the Manual 2 summaries had an average length of 9 words. Collectively, they had an average length of 10 words.
Figure 3. Average length of the manual summaries

Given the results of our manual summaries, in the next few chapters we describe our automated approaches towards summary generation. In each chapter we describe the particular algorithms, evaluate them according the evaluation metrics we have defined here, and then compare their results to the manual evaluation results shown above.
5 Preliminary Summarization

5.1 Introduction

In order to lay the groundwork for evaluating our primary microblog summarization algorithms, we first present two preliminary algorithms that we use for generating microblog summaries for each of our 50 testing topics. While these two approaches are simplistic, they serve a critical role towards allowing us to evaluate the results of our primary algorithms since no prior results yet exist in the literature. After presenting our preliminary approaches, we evaluate their performance using the metrics we defined in the Chapter 4. Finally, we discuss their implications for our primary algorithms discussed in future chapters.

5.2 Random Approach

Our first preliminary approach towards summarizing microblogs is a completely naïve approach. Given a filtered collection of posts\(^9\) and topic sentences\(^10\) that are each related to a single microblogging topic, we generate a summary by simply choosing at random either a post or sentence from the set of inputs. To compare these two random approaches, we generate an independent set of 50 summaries, one for each topic, using each approach.

5.3 Length Approach

Our second preliminary approach is also a simple technique that serves as an indicator of how easy or difficult it is to improve upon the random approach to summarization. It works as follows. Given a filtered collection of posts and topic sentences that are each related to a single microblogging topic, we generate four independent summaries. For two of the summaries, we choose both the shortest and longest post from the collection. For the remaining two, we choose both the shortest and

---

\(^9\) The particular means of filtering was described previously in the Evaluation Setup section of Chapter 4.

\(^10\) We defined a topic sentence as the longest sentence in a post that contained the topic phrase.
longest topic sentence from the collection. Finally, in order to evaluate these approaches separately, we generate an independent set of 50 summaries for each of our testing topics using each of the four different approaches.

5.4 Results and Discussion

Using our evaluation setup we discussed in Chapter 4 and the preliminary approaches discussed above, we generated independent sets of summaries for all 50 testing topics. Next, we evaluated these summaries using both the ROUGE-1 and Content metrics in order to understand how these summaries compared with the two sets of manual summaries generated earlier. The results of these metrics are shown below in Figure 1.

![Preliminary ROUGE-1 Performance](image)

**Figure 1: ROUGE-1 performance for the preliminary summaries**

Figure 1 above displays the average ROUGE-1 performance for our preliminary summarization algorithms in addition to the manual summarization results produced in Chapter 4. The first set of preliminary results in Figure 1 is for the random-based approach that generated summaries by either
choosing a random sentence or a random post. The generation of random sentences produced an average recall of 0.23, an average precision of 0.22, and an F-measure of 0.23. These results were higher than we originally had anticipated given our average manual F-measure is only 0.34. However, after more careful examination, we can explain these results by considering two factors. First, the ROUGE-1 metric compares all unigrams between the manual and automated summaries, including stop words. Therefore, some of the overlap we see is explained by the fact that ROUGE-1 is measuring these common words. Second, while we have described our first preliminary approach as “random”, we have introduced a considerable amount of bias into this approach by our preprocessing steps performed earlier. Recall that we have already removed many of the irrelevant posts (such as spam or posts in other languages) and have also isolated the longest set of sentences that contain the topic phrase. Therefore, our random approach is choosing a random sentence from more relevant set of input sentences than the original raw set of data.

Finally, in order to understand how relevant our random sentences are compared to our manually generated summaries, we present its Content performance below in Figure 2.

![Preliminary Content Performance](image)

**Figure 2: Content performance for the preliminary summaries**
Figure 2 indicates that our random sentence approach generated a Content score of 3.0. Given the Content categories and corresponding weights of *none* (1), *hardly any* (2), *some* (3), *most* (4), and *all* (5), our random sentence Content score corresponds exactly to “some” of the meaning of the manual summaries. While this value was higher than our expectations, it indicates that our preprocessing steps have produced a reasonable density of relative content. Therefore, our random approach, on average, captures some of the meaning of the manual summaries.

In addition to choosing random sentences, we also chose random posts. This appeared to slightly improve the recall scores over the random sentence approach (0.24 vs. 0.23), but greatly worsen the precision (0.17 vs. 0.22). These results are not surprising when considering the average summary lengths for the two approaches as shown below in Figure 3.

![Preliminary Average Length](image)

**Figure 3. Average length of the preliminary summaries**

In Figure 3, we see that the random post approach produced an average length of 15 words while the random sentence averaged only 12 words. Since the random sentence approach was closer to the average manual summary length, it scored with higher precision. Overall, the random sentence
approach produced a summary that was more balanced in terms of recall and precision and a higher F-measure as well (0.23 vs. 0.20).

Finally, the results of the second preliminary approach, namely our length-based approach, were unsurprisingly disappointing. Furthermore, they demonstrate why one must use both recall and precision when measuring performance. The shortest sentence and shortest post approaches generated far too short of summaries, averaging only two or three words in length, as seen in Figure 3. Because of their short length, these two approaches generated very high precision but failed horribly at recall, scoring less than either of the random approaches. Choosing the longest sentence and longest post had the exact opposite problem. These two approaches generated fairly good recall (approximately the average manual recall), but very low precision. Therefore, any viable summarization algorithm must be judicious in balancing both precision and recall when generating summaries.

Last, these results demonstrate one of the limitations of the ROUGE-1 metric discussed in Chapter 4: in order to achieve a high combined ROUGE-1 score (i.e. F-measure), the number of tokens between the automated and manual summaries must be relatively similar to one another since ROUGE-1 is simply comparing unigrams. If our automated summaries happen to express the same amount of meaning in a shorter summary than the manual summaries, the ROUGE-1 metric would penalize the automated summary’s recall. Therefore, when using the ROUGE-1 metric, one must assure that the manual and automated summaries have the same amount of compression. For this reason, we required our manual and automated summaries both to be 140 characters or less.

5.5 Conclusions

Given the results of our manual summaries and the preliminary summaries produced here, we now have our expected performance ranges for our two evaluation metrics. For the ROUGE-1 metric, our expected range of performance is between 0.23 and 0.34 F-measure as produced by our random
sentence and manual approaches, respectively. In other words, we expect our primary summarization approaches to be no worse than our random approach and likely no better than our manual approach.

For the Content metric, our expected range of performance is between 3.0 and 4.2 which roughly corresponds to between “some” and “most” of the meaning of the manual summaries, respectively.

Given we now have our expected ranges of performance, we next present our first primary summarization algorithm, the Phrase Reinforcement algorithm, in Chapter 6.
6 Phrase Reinforcement Summarization

6.1 Introduction

Now that we have laid the groundwork for evaluating our various summarization algorithms, we are ready to present our first primary algorithm: the novel Phrase Reinforcement algorithm. The Phrase Reinforcement (PR) algorithm generates one or more summaries by looking for the most commonly occurring phrases for a central topic. By representing these common phrases as a weighted and directed acyclic graph, the PR algorithm is able to generate summaries simply by searching for the most weighted set of paths through the graph.

In this chapter, we describe our motivation for the PR algorithm as well as a detailed presentation of its methodology. Following this, we evaluate the algorithm using our testing procedure described in Chapter 4 and then conclude the chapter with an evaluation of its results.

6.2 Motivation

The Phrase Reinforcement algorithm was inspired by two observations we made while analyzing a set of microblogging posts that are all related to a single topic. The first observation was that users tend to use similar words when describing a particular topic, especially immediately adjacent to the topic phrase. For example, consider the following set of posts that were collected on the day of the comedian Soupy Sales’ death:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIP: Soupy Sales, 83, slapstick legend for generation.</td>
</tr>
<tr>
<td>2</td>
<td>Soupy Sales pass away no way.</td>
</tr>
<tr>
<td>3</td>
<td>A Pie in the Face of Comedy: So Long, Soupy Sales</td>
</tr>
<tr>
<td>4</td>
<td>Our first Soupy Sales RIP cartoon</td>
</tr>
<tr>
<td>5</td>
<td>Goodbye Soupy Sales.</td>
</tr>
<tr>
<td>6</td>
<td>Aw, Soupy Sales died.</td>
</tr>
<tr>
<td>7</td>
<td>Just read that my favorite comedian Soupy Sales died.</td>
</tr>
<tr>
<td>8</td>
<td>Soupy Sales meant pie, dogs, silliness and laughs</td>
</tr>
<tr>
<td>9</td>
<td>Soupy Sales -- RIP -- I always watched his show when I was a kid - classic!</td>
</tr>
</tbody>
</table>
In the above posts, we underlined all of the words that immediately follow the topic phrase “Soupy Sales”. Notice that posts 2, 4, 6, 7, 9 and 10 all contain words immediately after the phrase “Soupy Sales” that in some way refer to his death. Furthermore, posts 4 and 9 share the word “RIP” and posts 6 and 7 share the word “died”. Therefore, for the above set of posts, there exists some overlap in word usage adjacent to the phrase “Soupy Sales”. This overlap occurs because there exists only so many ways to express the main idea of Soupy Sales’ death.

The second observation we made was that microbloggers will often repeat the most relevant posts for a trending topic by quoting other microbloggers. Quoting has its own special convention in Twitter and uses the following form:

RT [@TwitterAccountName]: “Quoted Message”

RT in this case refers to “Re-Tweet” and indicates one is copying a post from the indicated Twitter account to their own. For example, the following are some quoted posts that occurred on the same day as Soupy Sales’ death:

<table>
<thead>
<tr>
<th></th>
<th>RT @dcagle: Our first Soupy Sales RIP cartoon</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>RT @RadioPages: Soupy Sales has died at 83.</td>
</tr>
<tr>
<td>3</td>
<td>RT: @LouYoungNY Soupy Sales Died Today.</td>
</tr>
<tr>
<td>4</td>
<td>RT @burlesquebabes @msdowantiques Soupy Sales -- RIP -- I always watched his show when I was a kid - classic!</td>
</tr>
<tr>
<td>5</td>
<td>RT @sarah_wallace: RIP Soupy Sales.</td>
</tr>
</tbody>
</table>

Retweeting, or quoting, significantly reinforces the overlap of word usage around a topic phrase. While users writing their own microblog post will occasionally use the same or similar words, retweeting can
cause entire sentences to perfectly overlap with one another. This, in turn, greatly increases the average length of an overlapping phrase for a given topic.

Since users tend to use common words around the topic phrase and also repeat relevant content, the Phrase Reinforcement (PR) algorithm attempts to capitalize on these behaviors. The main idea of the PR algorithm is to determine the most heavily overlapping phrase centered about the topic phrase. This phrase is then used as the topic’s summary. The justification for this approach is that repeated information is often a good indicator of its relative importance (Luhn, 1958). In order to determine the most overlapping phrase, the PR algorithm builds a graph that represented all the repeated phrases for a given set of input posts. The graph is then searched for the most overlapping phrase.

6.3 Phrase Reinforcement Algorithm

The Phrase Reinforcement algorithm begins with a starting phrase which is the topic for which one desires to generate a summary. These phrases are typically a trending topic, but can be other non-trending topics as well. For this discussion, assume our starting phrase is the trending topic “Soupy Sales”. Next, the PR algorithm collects a series of posts that each contains the starting phrase. Any number of posts can be used, though for our development we used both 100 (our chosen testing size) and 1500 posts (the maximum returned by Twitter for a given search). Given this set of posts, the next step is to filter out any irrelevant posts (e.g. spam) that may not apply to the given topic. Filtering is an important step since otherwise the algorithm may summarize the irrelevant content rather than the desired content. See the Evaluation Setup section in Chapter 4 for a complete list of the filtering performed for this and the other summarization algorithms. Finally, since we desire to generate at most a single summarizing sentence, we last isolate the longest sentence within each remaining post that contains the trending topic phrase. This way, we limit the hypothesis space to the most relevant
sentences. Continuing with our example, assume we have isolated the following six topical sentences for the trending topic “Soupy Sales”:

1. Aw, Comedian Soupy Sales died.
2. RIP Comedian Soupy Sales dies at age 83.
3. My favorite comedian Soupy Sales died.
4. RT @NY: RIP Comedian Soupy Sales dies at age 83.
5. RIP: Soupy Sales Died Today.
6. Soupy Sales meant silliness and laughs.

Once we have collected the set of input sentences, the Phrase Reinforcement algorithm formally begins. The algorithm begins by initializing the graph with a root node that contains the starting or root phrase:

![soupy sales]

**Figure 1. Phrase Reinforcement root node for topic “Soupy Sales”**

Since we are looking for phrases that overlap with the topic or root phrase, the graph is built in two halves centered about the root node. The left-hand side of the graph represents the overlapping phrases occurring prior to the root phrase while the right-hand side represents the overlapping phrases occurring after the root phrase. The graph is constructed this way since the root phrase is the only phrase guaranteed to be common across all of the input sentences.

Given the graph has two halves, the construction of the graph is built in the same manner: first the left-hand side of the graph is constructed, followed by the right-hand side. To construct the left-hand side, the algorithm first initializes the current node as the root node. Next, it reduces the set of input sentences to the set of sentences that contain the current node’s phrase. The phrase of a node is represented by the sequence of words generated by following the path from the current node to the
root node. Since the current node and the root node are initially the same, the current node’s phrase is just “soupy sales”. Furthermore, since every input sentence is guaranteed to contain the root phrase, our list of sentences does not change initially. Subsequently, the algorithm isolates the set of words that occur immediately before the current node’s phrase. From this set, duplicate words are combined and assigned a count that represents how many instances of those words were detected. Using our example once again, there are two unique words that occur before the phrase “soupy sales”: “comedian” (sentences 1, 2, 3, and 4) and “RIP” (sentences 5). For both of these unique words, the PR algorithm adds them to the graph as nodes with their associated counts to the left of the current node like so:

![PR graph diagram]

**Figure 2. PR graph after adjacent words to the root phrase are added**

Notice in Figure 2 that the nodes are all in lower-case and stripped of any non-alpha-numeric characters (see node “rip” for example). Words are tokenized in this manner in order to increase the amount of overlap between words\(^\text{11}\). Furthermore, notice that each node has an associated count. The count represents the fact that the associated node’s phrase has exactly count number of occurrences within the set of input sentences at the same position and word sequence relative to the root node. For example, the phrase for node “comedian” is “comedian soupy sales”. Since this node has a count of four, we know there are exactly four instances of this phrase within the set of input sentences (sentences 1, 2, 3, and 4). Finally, nodes with a count less than two (such as the node “rip”) are not actually added to the graph since the PR algorithm is looking for overlapping phrases. However, in order

\(^{11}\) While word stemming would increase this overlap, we are not currently stemming words at this time. Please see the Future Work section in Chapter 8 for this and further enhancements to the PR algorithm.
to help illustrate our graph without a volume of example data, we temporarily add these nodes to the graph. We will later remove these nodes once the graph is fully constructed.

Following the addition of each leaf node, the graph continues recursively in a depth-first search ordering to add the remaining ancestor nodes. Therefore, after the addition of the node “comedian” in Figure 2, the algorithm would next set this node as the current node. Following this, the algorithm would isolate the set of input sentences that contain the partial phrase represented by the current node. Recall that a node’s phrase is represented by the word sequence constructed by following the path from the node to the root node. For the node “comedian”, this partial phrase would be “comedian soupy sales”. Isolating our set of input sentences to the set containing this partial phrase, we have:

1. Aw, Comedian Soupy Sales died.
2. RIP Comedian Soupy Sales dies at age 83.
3. My favorite comedian Soupy Sales died.
4. RT @NY: RIP Comedian Soupy Sales dies at age 83.

In this case, there are three unique words that occur prior to the phrase “comedian soupy sales”: “aw”, “rip”, and “favorite”. Adding these to our graph, we have:

![Diagram of PR graph with adjacent words]

**Figure 3. PR graph after adjacent words to the phrase are added**

Finally, the PR algorithm would continue this process recursively for each node added to the graph until all the potential words have been added to the left-hand side of the graph. Once complete, the PR algorithm would then repeat this algorithm symmetrically for the right-hand side of the graph. At
the completion of the graph-building process, the graph would look like the following for our example assuming we allow non-overlapping words:

![Graph Diagram]

**Figure 4. Fully constructed PR graph (allowing non-overlapping words/phrases)**

Given the fully constructed graph in Figure 4, notice the following. First, every path beginning from any node and ending at the root node represents an actual partial phrase within at least N of our input sentences where N is the number within the starting node. For example, the partial phrase represented by the node “laughs” is “soup sales meant silliness and laughs”. Since the node “laughs” has a count of 1, we know that there exists exactly one instance of this partial phrase within the set of input sentences (sentence 6). Next, notice that paths that cross the root node do not necessarily represent an actual input sentence phrase. For example, the phrase “rip soup sales died today” is an actual phrase (sentence 5) but the phrase “rip soup sales meant silliness and laughs” is not.

Next, since we originally allowed non-overlapping phrases, we will now reinstate that restriction by pruning the graph of any sub-trees containing nodes with a count less than 2. This reduces the graph to the following:
Following the construction of the graph, the algorithm next begins preparing for the generation of summaries by weighting the individual nodes. Node weighting is performed in order to account for the fact that some words have more informational content than others. For example, the root node “soupy sales” really contains no information since it is common to every input sentence. For this reason, we give it a weight of zero. Likewise, common stop words are noisy features that do not help discriminate between phrases. In fact, they can often be misleading. For example, consider the following graph:

Figure 6. An example of why stop words can be overly represented

Figure 6 illustrates one of the problems with stop words: they can be overly represented because of their natural frequency. For this reason, we give them a weight of zero as well. Finally, for the remaining words, we first initialize their weights to the same values as their counts. Then, to
account for the fact that some phrases are naturally longer than others, we penalize nodes that occur farther from the root node by an amount that is proportional to their distance:

\[
\text{Weight}(\text{Node}) = \text{Count}(\text{Node}) - \text{Distance}(\text{Node}) * \log_b \text{Count}(\text{Node})
\]

(31)

In the above equation, the logarithm base \( b \) is a parameter to the equation and can be used to tune the algorithm towards longer or shorter summaries. For aggressive summarization (higher precision), the base can be set to small values (e.g. 2 or the natural logarithm \( e \)). While for longer summaries (higher recall), the base can be set to larger values (e.g. 100). We will demonstrate how the algorithm's performance changes for different values of \( b \) in the evaluation section later in this chapter.

Finally, weighting our example graph from before gives us the following graph:

![Figure 7. PR graph demonstrating count / weight if using \( b = 10 \)](image)

As seen in the Figure 7, the nodes have been annotated to display their count / weight. The root node as well as any stop words (e.g. “at”) are given no weight for reasons described earlier. For the remaining nodes, their weights are penalized the further they become from the root node according to Equation 1. In this case, we assumed the logarithm base \( b \) was set to 10 for helping generate longer summaries.

Once the graph has been constructed and weighted, the PR algorithm is ready to begin generating summaries. In order to generate a summary, the PR algorithm looks for the most overlapping phrase within the graph. Since the node’s weights are proportional to their overlap, the algorithm simply has to search for the phrase with the most weight. However, in order to restrict
ourselves to legitimate phrases that occur within the input sentences, we do not consider phrases that 
span the root node. Phrases that span the root node may not reflect actual phrases found within the 
input sentences and therefore may have poor coherence. Instead, we search for all paths (using a 
depth-first search algorithm) that begin at the root node and end at any other node. The path with the 
most weight is then chosen as the best partial summary by the PR algorithm. It is considered only a 
partial summary since it only represents the most common phrase that occurs either before or after the 
root phrase. Consider the following figure:

Figure 8. PR graph demonstrating best partial summary (in green)

In Figure 8, we have highlighted the path with the most weight (a total of 5.1) in green. This 
path represents the most heavily weighted path in the graph between any node and the root node. 
Since this path is limited to either the left-hand side of the graph or the right-hand side, it represents 
only a partial or half summary for the initial topic phrase.

In order to generate the remainder of the summary, we next repeat the PR algorithm for only 
the opposite side of the graph from which we found the partial summary. In this case, since we 
generated a partial summary for the left-hand side of the phrase, we only have to repeat the PR 
algorithm for the right-hand side. Furthermore, since we only want to generate phrases that are 
actually found within the input sentences, we initialize our root node to the partial phrase generated 
earlier. This way, when we filter our input sentences to those containing this phrase, we can only
generate phrases found within the input sentences. Figure 9 below displays the new root node with its value initialized to the most weighted partial phrase generated earlier.

![Figure 9. Root node for generating remaining half of summary](image)

Isolating our sentences to those containing the root node phrase, we have:

2. **RIP Comedian Soupy Sales** dies at 83.
3. RT @NY: **RIP Comedian Soupy Sales** dies at age 83.

Since we are only building the right-hand side of the graph, we next consider the words that follow the phrase “rip comedian soupy sales”. In both cases, we have the same word “dies”. Adding this node and the remaining nodes with their corresponding counts and weights, we have:

![Figure 10. Fully constructed PR graph for second half of summary](image)

Searching this graph for the most weighted path gives us in this case the complete graph:

![Figure 11. PR graph demonstrating best complete summary (in green)](image)

Therefore, the full summary generated by the Phrase Reinforcement algorithm for our example is: “rip comedian soupy sales dies at age 83”. Note that this summary has lost its case-sensitivity and formatting since these features were removed by the PR algorithm in order to increase the amount of
overlap between phrases. In order to recover these features, we perform a simple “best-fit” algorithm between the summary and the set of formatted input sentences to find a matching phrase that contains the summary. We know such a matching phrase must exist within at least two of the input sentences since the PR algorithm only generates summaries from common phrases. To find such a phrase, we first divide our summary into an ordered list of tokens. Next, we search for the occurrence of each token, in order, from within each of the input sentences. However, since we don’t know exactly how many formatting characters (e.g. punctuation, whitespace, etc.) are between each token, we permit up to a threshold number\(^\text{12}\) of intervening characters between each summary token. Once, we find the first matching phrase, we use this phrase as our final formatted summary. Using our example, our unformatted summary first matches sentence 2 which produces the following final summary:

“RIP Comedian Soupy Sales dies at age 83.”

Finally, in order to abbreviate our description, we present pseudo code of the Phrase Reinforcement algorithm below in Figure 12.

\(^{12}\) Empirically, we have found an eight character threshold to be sufficient.
Node CreateNode( String phrase );
String FindMostWeightedSummary( Direction direction, Node node );

GeneratePhraseReinforcementSummary( String topicPhrase, List<String> inputSentences )
begin

Node rootNode, partialSummaryNode;
Summary leftSummary, rightSummary, finalSummary;
double leftWeight, rightWeight;

/* Initialize */
rootNode = CreateNode( topicPhrase );

/* Create both halves of the graph */
GenerateGraph( Direction.Left, rootNode, inputSentences );
GenerateGraph( Direction.Right, rootNode, inputSentences );

/* Find the most weighted summary from each graph half */
leftSummary = FindMostWeightedSummary( Direction.Left, rootNode );
rightSummary = FindMostWeightedSummary( Direction.Right, rootNode );

/* Generate the other half of the partial summary */
if (Weight(leftSummary) < Weight(rightSummary))
    partialSummaryRootNode = CreateNode( rightSummary );
    GenerateGraph( Direction.Left, partialSummaryRootNode, inputSentences );
    finalSummary = FindMostWeightedSummary( Direction.Left, partialSummaryRootNode );
else
    partialSummaryRootNode = CreateNode( leftSummary );
    GenerateGraph( Direction.Right, partialSummaryRootNode, inputSentences );
    finalSummary = FindMostWeightedSummary( Direction.Right, partialSummaryRootNode );
endif;

/* Return the final summary */
return finalSummary;
end;

GenerateGraph( direction, currentNode, inputSentences )
begin

currentNodePhrase = FindNodePhrase( currentNode );
filteredSentences = FindSentencesWithPhrase( inputSentences, currentNodePhrase );
adjacentWords = FindAdjacentOverlappingWords( filteredSentences, currentNodePhrase, direction );

foreach word in adjacentWords do
    adjacentNode = CreateNode( word );
    adjacentNode.Count = word.Count;
    adjacentNode.Weight = Weight( adjacentNode );

    if (direction = Left)
        adjacentNode.RightNodes.Add( currentNode );
    currentNode.LeftNodes.Add( adjacentNode );
    else /* (direction = Right) */
        currentNode.RightNodes.Add( adjacentNode );
    adjacentNode.LeftNodes.Add( currentNode );
endif;

adjacentNode.Weight = WeightNode( adjacentNode );
GenerateGraph( direction, adjacentNode, filteredSentences );
endfor;
end;
6.4 Results and Discussion

Now that the Phrase Reinforcement algorithm has been described, we present its results on the set of 50 testing topics we composed earlier in Chapter 4. The PR algorithm is able to generate summaries using complete microblogging posts as input since it is just building a connected graph of words around the central topic. In other words, it does not matter if the words span sentence boundaries since the PR algorithm ignores punctuation and just treats the posts as a sequence of words. However, in order to increase the compactness of the generated summaries and to improve their coherence, we limited our input to the set of topic sentences\textsuperscript{13}. This way, the resulting summaries would never span a single sentence. Given this approach, we repeated this process for each of our 50 testing topics and generated a complete set of summaries. Following this, we compared our automated summaries with the two sets of manual summaries generated earlier using the two metrics we defined in Chapter 4: ROUGE-1 and Content. The following Figures display our results.

\textbf{Figure 12. The Phrase Reinforcement Algorithm}

\textbf{Figure 13: ROUGE-1 performance for the Phrase Reinforcement summaries}

\textsuperscript{13} Recall that the topic sentence is the longest sentence in a microblogging post that also contains the topic phrase.
Figure 13 displays the ROUGE-1 performance for the Phrase Reinforcement algorithm while using a logarithm base of 100 for the weight of a node as described in Equation 1 (we will describe momentarily why we chose that particular weighting). In addition, the figure displays the ROUGE-1 results we computed earlier for the manual and randomly generated summaries which represented our expected range of results. As seen in the figure, the PR algorithm produced an average recall of 0.30, an average precision of 0.31, and a combined F<sub>1</sub>-measure of 0.30. This is a significant improvement over the random sentence approach which had an F<sub>1</sub>-measure of 0.23. However, it still leaves some room for improvement since the manual summaries had an F<sub>1</sub>-measure of 0.34. Comparing PR summaries using our Content metric, we generated the results seen in Figure 14 below.

![Phrase Reinforcement Content Performance](image)

**Figure 14: Content performance for the Phrase Reinforcement summaries**

For the Content metric, the PR algorithm generated an average score of 3.66. Given the Content categories and corresponding weights of *none* (1), *hardly any* (2), *some* (3), *most* (4), and *all* (5), this score corresponds to expressing between the “some” and “most” of the meaning of the manual summaries which again is an improvement over the random summaries. Finally, Figure 15 below
indicates that the PR summaries were on average only two words longer in length than the average manual summary and equal to the length of the average random sentence.

![Phrase Reinforcement Average Length](image)

**Figure 15: Average length of the Phrase Reinforcement summaries**

While the PR algorithm generates significantly better summaries than the random based approach, it suffers from a couple of problems. First, since the PR algorithm generates phrases instead of sentences for summaries, the summaries are not always coherent. For example, consider the four summaries in Table 3 which were produced by the PR algorithm from our testing set.

<table>
<thead>
<tr>
<th>Topic</th>
<th>PR Phrase (log 100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#musicmonday</td>
<td>#musicMonday.. is</td>
</tr>
<tr>
<td>Eddie Royal</td>
<td>Eddie Royal and Darren Sproles on</td>
</tr>
<tr>
<td>#clubrules</td>
<td>#clubrules Ladies, If ur</td>
</tr>
<tr>
<td>Saw VI</td>
<td>watch Saw VI with</td>
</tr>
</tbody>
</table>

**Table 3. Incoherent summaries produced by the Phrase Reinforcement algorithm**

In Table 3, we see four topics for which the PR summary is completely incoherent. While the PR algorithm was able to find some common phrases around the topic phrase, the common phrases were not long enough in order to be coherent. For example, for the trending topic “Eddie Royal”, the PR algorithm was able to find the common phrase “Eddie Royal and Darren Sproles on” which is missing
some of the key details about these two individuals. In order to help mitigate against this problem, we tried a couple of experiments of replacing the generated PR phrase with corresponding sentences and posts. In one experiment, we replaced the PR phrase with either the shortest topic sentence or post that contained the PR phrase in hopes of completing the missing relevant information. In the other experiment, we tried replacing the PR phrase with either the longest topic sentence or post. The results of these efforts are shown below in Figure 16.

![Phrase Reinforcement ROUGE-1 Performance](image)

**Figure 16: ROUGE-1 results of extending PR phrases into sentences and posts.**

In Figure 16, we see that extending the original PR phrase into either a sentence or post by using either the shortest or longest forms did not improve our results. While the recall slightly improved, the precision dropped too significantly. The reason is seen in Figure 17 which indicates that the length of the resulting summaries were simply too long relative to the manual summaries. These additionally added words reduced the ROUGE-1’s precision.
Figure 17: Average length results of extending PR phrases into sentences and posts.

Another issue that is more fundamental than generating phrases versus sentences is due to the very design of the PR algorithm: in order to generate coherent summaries, the PR algorithm must be able to find a coherent and common phrase around the initial topic phrase from within the set of input sentences. Unfortunately, these types of phrases do not always occur. This can be especially true for some types of topics that do not naturally lend themselves to making a coherent sentence. For instance, consider the trending topic “#clubrules”. This trending topic was defined by one of our manual summaries as “funny suggestions of behavior when going clubbing [dancing]”. The following table shows the first five topic sentences that contains this trending topic phrase.

| #clubrules                                                                                           |
|-----------------------------------------------------------------------------------------------|---|
| #clubrules If u cant dance 2 step yo ass off!!! dont try no crazy shit stay in ya lane ladies and gents |   |
| #clubrules: Ladies don't carry big purses 2 the club & get mad cause there's no room for u2 dance. Nobody told u to pack an overnight bag smh. |   |
| #clubrules ladies once u need 2 spray change shirts and re apply make up its time 4 u 2 go home   |   |
| #ClubRules Gosh darnit! Tip the waitresses and bartenders! Without that alcohol your night would be lame! |   |
RT @mrsloveylabels: #clubrules ladies; stop pickin up those dollar bills off the floor. Birds.

Table 4. Example topic sentences for the trending topic “#clubrules”

For the topic “#clubrules”, the topic phrase does not naturally make up a larger coherent phrase because the topic is more of a title rather than a phrase. Therefore, instead of using the topic phrase as part of a sentence, users tend to place the topic phrase either at the beginning or end of the post. For these types of topics, the PR algorithm will be unable to generate a coherent summary.

Fortunately, the problems mentioned above do not seem to be a significant problem overall for the majority of the topics. Out of the 50 testing topics we chose, only the four topics seen in Table 3 were incoherent. The remainder were coherent phrases or sentences. A sample of these coherent summaries is presented below in Table 5.

<table>
<thead>
<tr>
<th>Topic</th>
<th>PR Phrase (log 100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#BeatCancer</td>
<td>For every tweet tagged with #BeatCancer, eBay &amp; Miller/Coors will donate $.01 to cancer.</td>
</tr>
<tr>
<td>#iusuallylieabout</td>
<td>#iusuallylieabout my age</td>
</tr>
<tr>
<td>A-Rod</td>
<td>A-Rod homers in third straight game</td>
</tr>
<tr>
<td>National League</td>
<td>Phillies defeat Dodgers to take the National League Championship Series.</td>
</tr>
<tr>
<td>Angels</td>
<td>#Dodgers lost 11-0, #Angels lost 10-1.</td>
</tr>
<tr>
<td>Russian Roulette</td>
<td>listening to rihanna's new song russian roulette</td>
</tr>
<tr>
<td>Dodgers</td>
<td>Phillies dump Dodgers for World Series return</td>
</tr>
<tr>
<td>Glee</td>
<td>Thong song on Glee!</td>
</tr>
<tr>
<td>Apple Fires Back</td>
<td>Apple Fires Back at Windows 7 in New Ads - &quot;Apple's &quot;I'm a PC&quot; and &quot;I'm a Mac&quot; dynamic ad duo are at it again i...</td>
</tr>
<tr>
<td>Balloon Boy</td>
<td>Balloon Boy Mom Admits to Hoax : According to court records made public today, Mayumi Heene, Ball..</td>
</tr>
</tbody>
</table>

Table 5. Coherent summaries produced by the Phrase Reinforcement algorithm

Last, we originally described that the PR algorithm is able to tailor the length of its summaries by adjusting the weights of nodes as defined by Equation 1. For example, by assigning less weight to nodes farther from the root phrase, the PR algorithm will more naturally prefer more common shorter phrases over less common longer phrases. In order to demonstrate and help understand the effects of Equation 1, we varied the equation’s logarithm base for a variety of values while measuring its effect on the
algorithm’s performance and average summary length. These results are presented in the following three figures.

Figure 18. ROUGE-1 performance for different weightings of the Phrase Reinforcement algorithm
Figure 19. Average summary lengths for different weightings of the Phrase Reinforcement algorithm
Figure 20. Content performance for different weightings of the Phrase Reinforcement algorithm

In figures 18 and 19, we can see the effect of varying the weighting parameter $b$ within Equation 1. There appears to be a threshold (in our case of when $b ≈ 100$) for which smaller values of $b$ begin reducing the average summary length. As $b$ decreases, the PR algorithm begins trading ROUGE-1 recall performance for precision performance and reducing the Content performance as well. Above this threshold, the average summary length does not increase while the ROUGE-1 and Content performance begins to degrade slightly. While the particular threshold may vary for different sizes of testing data, we can generalize that the weighting parameter $b$ affects the resulting length and performance of the PR algorithm. However, beyond a certain threshold, the PR algorithm’s performance is relatively stable. For our results, it appears that when $b = 100$, our results are maximized with a fairly balanced ROUGE-1 performance a maximized Content performance. Therefore, we use the results of this threshold for all of our remaining work and comparisons with the other algorithms. Finally, note that the results in Figures 18-20 labeled with the title “PR Phrase (NULL)” indicate the absence of the weighting parameter all together. In these cases, we simply use a node’s count as its weight.

6.5 Conclusions

In this chapter we have presented our Phrase Reinforcement algorithm and demonstrated its relative performance to our preliminary methods. The PR algorithm generates significantly improved summaries with an average ROUGE-1 performance of 0.30 F1-Measure and an overall Content performance of 3.66. While these results are much better than the random sentence approach (0.23 F1-Measure and 3.0 Content), they still leave some room for improvement as our manual summaries still had higher scores (0.34 F1-Measure and 4.22 Content). Some of the limitations of the PR algorithm are due to its requirement of finding common coherent phrases around the topic phrase from within the set of input sentences. As described earlier, these common phrases do not always exist in a coherent form.
for certain types of topics. Finally, we concluded this chapter with some example summaries produced by the PR algorithm as well as the effects of its weighting parameter presented earlier in Equation 1. Given these results, we next perform one final attempt at summarization using an adaptation of a well-known statistical technique known as TF-IDF. This last attempt concludes our approaches in microblog summarization and is presented next in Chapter 7.
7 TF-IDF Summarization

7.1 Introduction

In Chapter 6, we presented the Phrase Reinforcement algorithm which summarizes microblogs by searching for the most common phrase around a topic phrase and then using this phrase as the summary. While this approach significantly improved upon our earlier results, it still left room for improvement as its performance only halved the difference between the random and manual summarization methods. In this chapter, we attempt another approach towards microblog summarization that is based upon a classical technique dating back to early summarization work performed by Luhn (1958). This approach is named TF-IDF, or Term Frequency – Inverse Document Frequency. After presenting this technique and our application of it towards microblog summarization, we evaluate it results using the metrics defined in Chapter 4. We then compare its results to the results of our previous work. Finally, we conclude this chapter with a discussion of our combined results.

7.2 TF-IDF

TF-IDF or Term Frequency – Inverse Document Frequency, is a statistical weighting technique that has been applied to many types of information retrieval problems. For example, it has been used for automatic indexing (Salton, 1989), query matching of documents (Manning, 2008), and automated summarization (Seki, 2002). While generally TF-IDF is not known as one of the leading algorithms in automated summarization (see the related work section in Chapter 4 for more recent algorithms), it is a classical technique that has been applied to many types of problems. For this reason, we attempted its use towards microblog summarization as well.

For automated summarization, the application of TF-IDF is fairly simplistic. The general idea is to assign each sentence within a document a weight that reflects the sentence’s saliency within the document. Once each sentence has been weighted, the sentences can be ordered by their weights from
which the top \( m \) sentences with the most weight can then be chosen as the document’s summary. Determining the weight of a sentence is also straightforward. In particular, the weight of a sentence is just the summation of the individual term weights within the sentence. Terms can be words, phrases, or any other type of lexical feature (Singhal et al., 1996). To determine the weight of a term, we come to the TF-IDF formula which states:

\[
TF - IDF = \text{Term Frequency} \times \log_2 \text{InverseDocumentFrequency}
\]

\[= tf_{ij} \times \log_2 \frac{N}{df_j} \tag{32}\]

where \( tf_{ij} \) is the frequency of the term \( T_j \) within the document \( D_i \), \( N \) is the total number of documents, and \( df_j \) is the number of documents within the collection that contain the term \( T_j \) (Salton, 1989).

As seen in the above equation, TF-IDF is composed of two primary parts: a term frequency and an inverse document frequency. The term frequency component (TF) assigns more weight to words that occur frequently within a document using the rationale that important words are often repeated as discovered by Luhn (1958). On the other hand, the inverse document frequency component (IDF) compensates for the fact that some words such as common stop words (e.g. “the”, “it”, “and”, etc.) are always frequent within any type document. Since these words do not help discriminate between choosing one sentence or document over another, these words are penalized proportionally to their inverse document frequency (the logarithm is taken to balance the effect of the IDF component in the formula). Therefore, to summarize, TF-IDF gives the most weight to words that occur most frequently within a small number of documents and the least weight to terms that occur infrequently or occur within the majority of the documents.

One of the noted problems with TF-IDF is the fact that the formula is relatively sensitive to the document’s length. For example, Singhal et al. note that longer documents have higher term
frequencies since they often repeat terms while also having a larger number of terms (Singhal et al., 1996). Note that this doesn’t have any ill effects on single document summarization since all the sentences use the same weights for terms. However, when generating a summary from multiple documents this becomes an issue because the terms within the longer documents have more weight. In order to compensate for this problem, numerous normalization methods have been proposed that attempt to normalize the weights of the terms according to the document’s length. Some of the most common means are Cosine Normalization, Maximum tf Normalization and Byte Length Normalization which are described in (Singhal et al., 1996).

7.3 TF-IDF Algorithm

Given the above description of TF-IDF, we first define our goal as finding the most salient sentence within a set of input microblogging posts and then using it as the summary. As described earlier, we can define the most salient sentence as the sentence with the most combined term weights where each term weight is defined using Equation 1. However, Equation 1 defines the weight of a term in the context of a document. However, recall that we don’t have a traditional document. Instead, we have a set of microblogging posts that are each related to a topic. Therefore, one fundamental question we must first answer before applying TF-IDF is how we define a document. One option is to define a single document that encompasses all the posts together. In this case, the TF component’s definition is straightforward since we can easily compute the frequencies of the terms across all the posts. However, doing so causes us to lose the IDF component since we only have a single document. On the other extreme, we could define each post as a document. In this case, the IDF component’s definition is clear since each term will only exists within a fraction of the available posts. However, the TF component now has a problem: because each post contains only a handful of words, most of the term frequencies will be a small constant for a given post.
In order to handle the above situation, we decided to redefine TF-IDF in terms of a hybrid document type. In particular, we primarily define a document as a single sentence. However, when computing the term frequencies, we assume the document is the entire collection of posts. Therefore, the TF component of the TF-IDF formula uses the entire collection of posts while the IDF component treats each sentence as a separate document. This way, we have differentiable terms frequencies but also don’t lose the IDF component to the equation. Finally, we define a term as being a single word in a sentence.

Given that we have defined the terms of Equation 1, we next choose a normalization method since otherwise the TF-IDF algorithm will always bias towards longer sentences. The reason for this bias is due to the fact that the weight of a sentence is simply the weight of each of the individual terms. Therefore, longer sentences will always have more weight than shorter sentences with the same terms. For this, we normalize weight of a sentence by dividing it by a normalization factor (described later). Last, since common stop words (e.g. “the”, “it”, etc.) do not help discriminate the saliency of sentences, we give each of these types of a words a weight of zero by comparing them with a prebuilt list. Given this, our definition of the TF-IDF summarization algorithm is now complete for microblogs. We summarize this algorithm below in Equations 2-6 and present it as pseudo code in Figure 1.

\[
Weight(Sentence) = \sum_{i=0}^{R} \frac{Weight(Word_i)}{NormalizationFactor(Sentence)}
\]  

(33)

\[
Weight(Word) = TermFrequency(Word) \times \log_2[InverseDocumentFrequency(Word)]
\]  

(34)

\[
TermFrequency(Word) = \frac{#OccurrencesOfWordInAllPosts}{#WordsInAllPosts}
\]  

(35)

\[
InverseDocumentFrequency(Word) = \frac{#SentencesInAllPosts}{#SentencesInWhichWordOccurs}
\]  

(36)

\[
NormalizationFactor(Sentence) = \max[MinimumThreshold, #WordsInSentence]
\]  

(37)
/* Initialize */
listOfAllSentences = DividePostsIntoSentences( listOfAllPosts );
numSentences       = Count( listOfSentences );
listOfAllWords     = DividePostsIntoWords( allPosts );
listOfUniqueWords  = FindUniqueWords( listOfWords );
listOfStopWords    = LoadStopWordList();
maxSentenceWeight  = 0;
summarySentence    = "";
threshold          = 11;

/* Compute term weights */
for each word w in listOfUniqueWords
    termFrequency = CountWordOccurrences( w, listOfAllPosts ) / Count( listOfAllWords );
    inverseDocFrequency = numSentences / CountSentenceOccurrences( w, listOfAllSentences );
    wordWeight = termFrequency * log2( inverseDocumentFrequency );
    AssignWeightToWord( w, wordWeight );
end for

/* Find sentence with most weight */
for each sentence s in listOfAllSentences do
    /* Compute the weight of the sentence s */
    sentenceWeight = 0;
    for each word w in s do
        if (IsStopWord( w ) != TRUE)
            sentenceWeight = sentenceWeight + RetrieveWeightFromWord( w );
        end if
    end for
    /* Normalize the sentence weight */
    sentenceWeight = sentenceWeight / max( CountWords( s ), threshold );
    /* See if we found a sentence with more weight */
    if ( sentenceWeight > maxSentenceWeight )
        maxSentenceWeight = sentenceWeight;
        summarySentence = s;
    end if
end for

/* Print summary */
print( summarySentence );

---

**Figure 1**: TF-IDF Algorithm for summarizing microblogs

### 7.4 Evaluation Results

Given the above description of the TF-IDF algorithm, we next present its results on the set of 50 testing topics and evaluation metrics defined in Chapter 4. In particular, we generated a summary for each individual topic and then compared the summary to both of the manual summaries using the ROUGE-1 and Content metrics. The results of the evaluation for the ROUGE-1 metric are shown below in Figure 2.
In Figure 2, we present the results of the TF-IDF algorithm for the ROUGE-1 metric in relation to all of our previous and best results thus far. The TF-IDF results are denoted in Figure 2 as “TF-IDF Sentence (11)” to distinguish the fact that the TF-IDF algorithm produces sentences instead of phrases for summaries and that we are using a threshold of 11 words as our normalization factor (we will describe momentarily why we are using this particular threshold value). The TF-IDF algorithm produced an average recall of 0.31, an average precision of 0.34, and a combined $F_1$-Measure of 0.33. Note that these values are very close to the performance levels of our manual summaries of 0.34. Furthermore, they are also better than our Phrase Reinforcement results which had an average recall of 0.30, an average precision of 0.31, and a combined $F_1$-Measure of 0.30.

We next evaluated the automated summaries against the manual summaries using our Content metric in order to understand whether or not we were truly achieving human-comparable summaries. The results of the Content evaluation are shown below in Figure 3.
Remarkably, the TF-IDF algorithm’s Content performance was also very similar to the manual summaries with an average Content score of 4.1 compared to 4.2 for the manual summaries. Given the Content categories and corresponding weights of none (1), hardly any (2), some (3), most (4), and all (5), this score corresponds to expressing slightly more than “most” of the meaning of the manual summaries. This score was also higher than the average Content score of the Phrase Reinforcement algorithm which was 3.66.

Finally, Figure 4 below displays the average length of the TF-IDF summaries.
Interestingly, the TF-IDF summaries were one word shorter, on average, than the manual summaries with an average length of 9 words. In fact, this was the exact average length of the second set of manual summaries generated by our second volunteer. Finally, to illustrate some of the summaries generated by the TF-IDF algorithm, we present a selection of summaries from our testing topics below in Table 1.

<table>
<thead>
<tr>
<th>Topic</th>
<th>TF-IDF Sentence (11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#musicmonday</td>
<td>#musicmonday damn have you guys heard the new gaga song?</td>
</tr>
<tr>
<td>#BeatCancer</td>
<td>Every tweet that includes #beatcancer raises money for cancer research.</td>
</tr>
<tr>
<td>Eddie Royal</td>
<td>Darren Sproles just did his Eddie Royal impersonation, returns punt.</td>
</tr>
<tr>
<td>DWTS</td>
<td>the michael jackson tribute on #dwts tonight was AWESOME!</td>
</tr>
<tr>
<td>Paranormal Activity</td>
<td>Paranormal Activity iz a scary movie!</td>
</tr>
<tr>
<td>#clubrules</td>
<td>#clubrules ladies don't wear smudgy make-up.</td>
</tr>
<tr>
<td>Puck</td>
<td>David Booth grabbed the puck and threw it down the ice.</td>
</tr>
<tr>
<td>Balloon Boy</td>
<td>Balloon Boy's Mom Admitted Hoax to the Cops!</td>
</tr>
<tr>
<td>Barnes &amp; Noble</td>
<td>Barnes &amp; Noble Nook e-Book Reader Official, $259</td>
</tr>
<tr>
<td>Dodgers</td>
<td>Phillies dump Dodgers for World Series return -</td>
</tr>
</tbody>
</table>

Table 1. Example summaries produced by the TF-IDF algorithm

One of the limitations noted about the Phrase Reinforcement (PR) algorithm in Chapter 6 was that it was not able to generate coherent summaries for topics whose phrases that did not naturally
coincide within a sentence. For example, the PR algorithm was not able to generate a coherent summary for the testing topic “#clubrules” because a coherent and common phrase did not exist within the input around this topic phrase. Notice in Table 1 that the TF-IDF algorithm does not suffer from this limitation as the same topic contains a coherent phrase. The TF-IDF algorithm is better able to handle these types of topics because it is a pure “bag-of-words” approach and is not dependent upon word order. Where the PR algorithm is dependent upon finding a common sequence of words, the TF-IDF algorithm is simply considering the frequencies and inverse-document frequencies of each word independently from every other word within a sentence.

Finally, the other limitation noted about the Phrase Reinforcement algorithm was that the summaries produced were phrases rather than sentences. While the TF-IDF algorithm produces sentences instead of phrases, in the majority of cases these sentences appear similar in structure as the phrases produced by the PR algorithm. One of the reasons for this similarity is the largely unstructured nature of microblogs. Since users tend to write more in phrases rather than in complete sentences, often the two sets of summaries are similar in grammatical structure. In fact, in some cases the summaries are completely identical.

7.5 Normalization Results

Earlier in this chapter we noted that the TF-IDF equation (Equation 1) is particularly sensitive to a document’s length. As noted by Singhal et al. (1996), longer documents will often repeat terms more often than shorter documents and therefore cause their term weights to be higher. This, in turn, will cause a bias towards selecting sentences from longer documents. However, a more significant problem is the fact that when using TF-IDF for summarization, a sentence’s saliency is measured by summing the TF-IDF weights of its individual terms. Because of this, longer sentences will always have more weight over shorter sentences that have the same terms. In order to deal with these problems, most
applications attempt to normalize the term weights in order to compensate for a document’s length. The most common method for term normalization is the Cosine Normalization technique as described in (Singhal et al., 1996).

For our work, we concentrated on normalizing the length of a sentence rather than normalizing the term weights for two reasons: (1) the sentence length had a much greater influence than the over-estimation of term weights and (2) we used a hybrid definition of a document which may or may not have been relevant to the standard normalization techniques. Prior to normalizing the length of a sentence, our average sentence length of a summary using TF-IDF was 20 words – much greater than our average manual sentence length of 10 words. Therefore, as predicted, the classical TF-IDF approach towards summarization naturally biased towards the longest sentences.

Our initial attempt to normalize the sentence length was to divide the sentence weight by the number of words in the sentence. However, this effort produced the exact opposite problem. Instead of biasing towards the longest sentences, the TF-IDF algorithm biased towards the shortest sentences with an average length of only three words. Furthermore, these sentences almost always contained just the topic phrase. The problem with this initial approach was that we were in fact computing the average term weight since we were dividing the sentence weight (the sum of term weights) by the number of terms. Therefore, the sentence with the highest average term weight is the smallest collection of the most heavily weighted terms. Hence, we now had two opposite approaches. Given these two extremes, we sought after a technique that would compromise between these two formulas.

The results of our efforts produced the normalization equation seen above in Equation 6 and restated here:

\[
\text{NormalizationFactor}(\text{Sentence}) = \max [\text{MinimumThreshold}, \#\text{WordsInSentence}]
\]
Our normalization factor seen above is simply the maximum of a minimum threshold and the number of words in a sentence. The minimum threshold is simply an integral number of words and approximately represents the desired average summary length. Given this definition, the normalization factor is then used as a divisor into the sentence weight.

By choosing the maximum of either a threshold or the sentence length, we are able to control the average summary length. In fact, we can produce a spectrum of average summary lengths simply by using a range of minimum thresholds. For example, consider the following figure:

![TF-IDF Average Length vs. Weight](image)

**Figure 5: Average length of the TF-IDF summaries for different normalization thresholds**

In Figure 5, we see a range of minimum threshold values ranging from 4-20. Notice that the average summary length is approximately equal to the minimum threshold. In addition, this figure displays the results of our original normalization method of dividing the sentence weight by the number of terms (TF-IDF Sentence (length)) and using no normalization method whatsoever (TF-IDF Sentence (NULL)).
The normalization factor is able to control the average summary length for the following reasons. If a sentence is longer than the threshold, then its sentence weight will be divided by its sentence length. This will reduce its weight to its average term weight as described earlier and produce a downward pressure on the average summary length. On the other hand, if a sentence is shorter than the threshold, then it will be divided by the threshold instead of the number of terms in the sentence. This, in turn, will produce upward pressure on the average summary length by essentially adding \((MinimumThreshold - \#WordsInSentence)\) zero-weight terms. The results of these two forces produce an average summary length approximately equal to the minimum threshold value as seen in Figure 5.

Given our normalization technique, we next computed the average ROUGE-1 and Content performance levels for different minimum thresholds in order to determine an optimal threshold for our testing data. Since ROUGE-1 measures unigram overlap between the manual and automated summaries, our initial guess of an optimal threshold was one that produces an average summary length equal to the average manual summary length of 10 words. According to Figure 5 above, this is a threshold of 12 words. As seen in Figure 6, this is not a bad approximation:
Figure 6: ROUGE-1 performance for different TF-IDF normalization thresholds

As seen in Figures 5 and 6, by varying the normalization threshold, we are able to control the average summary length and resulting ROUGE-1 precision and recall. Furthermore, there appears to be an inflection point where the precision and recall performance levels cross at the threshold value of 12 words. As seen in Figure 5, this inflection point is where the average TF-IDF summary length is equal to the average manual summary length. Therefore, we can use the threshold to control whether we desire better precision or recall or a balance of these two values for the produced TF-IDF summaries. More interestingly, we don’t necessarily need to compromise on Content performance while varying the summary length as Figure 7 attests:
Figure 7: Content performance for different TF-IDF normalization thresholds

Figure 7 demonstrates relatively high Content scores that are no lower than 3.78 for thresholds greater than or equal to 8 words.

Finally, in order to find the optimal threshold for our testing set, we used a linear range of thresholds that centered close to the inflection point in Figure 6. Figure 8 demonstrates the ROUGE-1 performance for thresholds that range from 8 through 13. Figure 9 demonstrates the Content performance for thresholds that range from 10 through 12. From these results, we decided that a threshold of 11 produced an optimal result for comparison against our other techniques.
Figure 8: ROUGE-1 performance for TF-IDF normalization thresholds between 8 and 13

Figure 9: Content performance for TF-IDF normalization thresholds between 10 and 12
7.6 Conclusions

In this chapter we have presented our adaption of the TF-IDF algorithm for summarizing microblogs. While the classical TF-IDF formula produces rather poor results due to its natural bias towards longer sentences, we developed a novel normalization formula that allows us to tailor the average summary length to almost any desired length. By tuning the summary length to a length that approximates the average manual summary length, we have achieved results that are comparable to the results of our manual summaries. Furthermore, these results outperform all of our previous efforts at microblog summarization – including the Phrase Reinforcement algorithm.
8 Summary and Conclusions

In this thesis, we have explored two different applications of machine learning and natural language processing to the relatively new domain of microblogging. In the first phase, we applied a naïve Bayes classifier to the task of automatically classifying microblog posts into one of ten categories. In the second phase, we performed a variety of techniques for automatically producing single sentence summaries for a given microblogging topic. In this chapter, we present a summary of our thesis work that highlights our methodologies, key results, limitations, and final conclusions. Following this, we describe our primary contributions to the state of the art in this field and then conclude the chapter with a look at potential future work.

In the first phase of our thesis, we introduced the naïve Bayes classifier, a supervised learning technique for performing automatic classification. Before applying this classifier towards our specific task of classifying microblogs, we first described the mathematical foundations of this classifier by deriving its canonical form. Following this, we described how to estimate the classifier’s required probabilities from the training data using the maximum likelihood estimate, and then described how to apply the classifier towards the specific task of automatic text classification. Next, we provided a detailed presentation of three classical optimization techniques for helping removing any bias within the classifier towards its training data. These techniques were the removal of common words (stop words), the removal of frequent words, and the computation of mutual information. Finally, we concluded our introduction with a description of our evaluation metrics which were to use precision, recall, and F-measure.

After describing the foundational mathematics, we next presented an implementation of the naïve Bayes classifier that we developed specifically for classifying a single microblog post into one of ten interest categories. Our classifier was trained using a training corpus composed of approximately
200,000 microblogging posts that were acquired through the development of a custom crawler dedicated to Twitter. Following its training, we performed an initial evaluation of the classifier’s performance by using stratified 10-fold cross validation and then optimized its performance using the techniques mentioned earlier. Finally, we evaluated the classifier’s performance a second time on a testing corpus consisting of 1000 microblog posts from 50 different microblogs that were manually classified using a group of volunteers.

Our results of automatic classification of microblogs were as follows. First, rather than classifying an entire microblog, we chose to classify individual posts instead since we were interested in the potential accuracy of such a short document type. We found that an average microblogging post from Twitter consists of only 11 words and thus provides a very small number of features for accurate classification. Despite this, our naïve Bayes classifier was able to classify a single microblogging post with considerable accuracy. For our training data, the classifier’s stratified 10-fold cross-validation results indicate a weighted-average F1-measure performance of 0.78 with a standard deviation of 0.08 across each of the ten target categories. More importantly, our results on our testing corpus were not much worse. After optimization, our classifier had a weighted-average F1-measure performance of 0.74 which was only 4% worse than the performance of our training data. One limitation of our results was the fact that our testing corpus only provided enough testing data for four of our ten target categories. Because our target categories only represented a subset of the most popular categories for Twitter, six of the categories did not have enough testing data in order to provide an accurate measure of their performance. However, results from our testing corpus indicate the naïve Bayes classifier can perform well for all of the ten target categories. Additional testing data would have complemented our training results however.
Following the results of our automatic classification work, we next proceeded with the second phase of our research: automatic microblog summarization. For this effort, we first presented an overview of the research in the related field of automatic text summarization. In particular, we gave a brief history, described the dimensions to the problem and their associated challenges, and then presented an overview of our approaches based on this research. Following our introduction, we also presented an overview of our evaluation methodology which was to choose two popular metrics used by the Document Understanding Conference, namely Content and ROUGE-1. Since both of these metrics require a set of manually generated summaries in which to compare the automatically generated summaries, we next described our method for composing a manual pair of summaries for 50 trending Twitter topics.

After presenting an overview of text summarization and our evaluation metrics, we next began describing our preliminary approaches towards summarization. Since previous results in this field did not exist for our domain, we first derived an expected range of performance for our evaluation metrics. For the expected lower-end of performance, we composed a naïve random-based approach to summarization that simply chose sentences at random from the input. Similarly, for the expected upper-end of performance, we compared our manually generated summaries against one another. Our preliminary approaches indicated that we expected a Content score (a measure of the amount of overlap in meaning between a manual and automatic summary) between 3.0 and 4.2 which correspond to the automated summary representing between “some” and “most” of the meaning of the manual summaries. For the ROUGE-1 metric (a measure of the co-occurring unigrams between a manual and automatic summary), our expected range of performance was between 0.23 and 0.34 F-measure.

Our first primary approach in automated summarization was the development of the novel Phrase Reinforcement (PR) algorithm. This algorithm was inspired from the simple observation that commonly
occurring phrases around a topic phrase often serve as a good summary. In order to discover these phrases, the algorithm constructs a directed acyclic graph of weighted nodes where each node represents a common phrase in the input and the weights reflect their corresponding frequencies. Once the graph is constructed, the PR algorithm searches all the various paths in the graph that begin or end with the root node. The path with the most weight is then used as a partial summary since this path only represents the most common phrase that either occurs before or after the topic phrase. Finally, the PR algorithm constructs one final graph that represents the most common phrases that contains the previously constructed partial summary. From this graph, the PR algorithm is able to generate a complete summary representing the most commonly used phrase centered about the initial topic phrase.

Using the above algorithm, the PR algorithm produced an average ROUGE-1 score of 0.30 F₁-measure and an average Content score of 3.66. The Content score indicated that the automatic summaries represented between some and most of the meaning of the manual summaries. Overall, our PR algorithm’s results indicate a significant improvement over our random-based approach towards summarization which generated scores of 0.23 F₁-measure for ROUGE-1 and 3.0 for Content. One perhaps unexpected result of the algorithm was the fact that the algorithm produced a coherent summary for 46 out of the 50 topics. These results may be surprising considering the algorithm is looking for overlapping phrases instead of complete sentences. However, because of the common “retweeting” or quoting behavior of Twitter users, there are often complete sentences of common phrases within the input. Furthermore, these retweeted posts often contain highly relevant content as users most often retweet the most salient posts for a given topic.

While common phrases often exist for a topic, occasionally they do not. For some topics (such as hashtag topics like “#clubrules”), the topic phrases do not naturally coincide within a larger phrase or
sentence. As a result, the PR algorithm is not able to generate a coherent summary. Fortunately, these types of topics were relatively few in our experience. One limitation of our analysis of this algorithm was our relatively few input posts for a given topic. Because we required generated manual summaries for each of our topics, we had to limit our input to 100 posts for a given topic. Otherwise, the time to generate manual summaries would have been prohibitive. Due to this limitation, it is unclear whether or not a larger collection of posts would have produced better resulting summaries since it would have provided additional opportunities for identifying common phrases. We leave this analysis as future work as well as additional enhancements to our algorithm.

After the composition and evaluation of the Phrase Reinforcement algorithm, we next applied a more traditional approach towards automated summarization known as Term Frequency – Inverse Document Frequency or simply TF-IDF. This approach is a classical technique with roots back in early summarization performed by Luhn (Luhn, 1958). In brief, the TF-IDF algorithm assigns a weight to every term based on a combination of the term’s frequency and inverse document frequency. Given a weighting of every term, a sentence weight can be derived by simply summing the weights of its terms. The sentence with the most weight is then chosen as a resulting summary.

While simple in concept, the TF-IDF approach towards summarization suffers a fatal problem: it is heavily biased towards the longest sentences since a sentence’s weight is simply the sum of the weights of its terms (or words). For this reason, our initial results had a very low ROUGE-1 precision of 0.18 which was lower than our random sentence summaries of 0.22. However, it did contain fairly high recall and Content scores (0.35 Recall and 3.78 Content). In order to compensate for this problem, we developed a novel sentence length normalization technique that allowed biasing the algorithm towards a summary with a predefined length. Using a bias that matches the average lengths of our manual summaries, we greatly improved the TF-IDF algorithm’s performance. Our best performing
normalization produced an average ROUGE-1 score of 0.33 F₁-measure and an average Content score of 4.1. These results were a significant improvement over our PR algorithm (0.30 F₁-measure and 3.66 Content) and nearly rivaled the results of our manual summaries (0.34 F₁-measure and 4.22 Content). Furthermore, our normalization method had one other benefit as well. Since we could control the resulting summary lengths, different ratios of precision and recall could be achieved depending on the needs of the application. For example, a small mobile device requiring even shorter summaries could sacrifice some recall for precision if needed.

Last, through the course of our work, we had a few general observations as well. First, it should be noted that we used the automated evaluation metric ROUGE-1 in addition to the manual evaluation metric Content since we were unsure of whether we could trust the ROUGE-1 metric to correlate with a manual evaluation. In other words, our Content metric was used as a confidence-check of the ROUGE-1 metric. Fortunately, our results corroborates with Lin’s which found that for very short document types, the ROUGE-1 metric correlates with human-based judgements (Lin, 2004). Therefore, future evaluations of short summaries can save themselves great amounts of effort by not having to manually evaluate the resulting summaries using the Content-based metric. While the Content metric gave some confidence to our results, it consumed considerable amounts of time and effort in order to ensure the metric was applied evenly and fairly across all of the generated summaries for the numerous approaches we tried.

Another observation we had was that microblogs represent a fairly noisy domain with significantly less structure than traditional blogs. Because of their short length, many microblog posts embed external information such as links to other web pages and use brief phrases with little to no formatting. In order to perform any natural language processing on these posts, we had to perform a considerable amount of preprocessing before the microblog posts could be usable for either classification or
summarization. Such efforts should not be underestimated as we had to provide many stages of preprocessing before we could recover a microblog’s natural language.

Finally, in order to effectively perform any form of machine processing on a source of microblogging data, one must develop an effective strategy for dealing with spam. When we first began using Twitter as a source of microblogging data, spam was practically non-existent. However, over the course of the last two years, spam has grown to be enough of a problem that it corrupts our results if it is not removed. For this reason, we trained another naïve Bayes classifier for recognizing spam as well as applied a series of heuristics. However, even with such efforts, one must realize that spam is a moving target. In order to effectively deal with its prevalence, we had to periodically retrain our spam classifier using a new set of training data. Fortunately, our data could be acquired within a fixed time period and then reused. Thus, our spam retraining efforts were mainly limited to when we collected a new set of training or testing data. However, for a real-time application of our methods, one would need to frequently retrain a spam classifier as spam continuously evolves and changes its revealing signatures.

8.1 Contributions

Our work on microblogging classification and summarization has made the following contributions to furthering the state of the art in natural language processing and machine learning:

1. A novel graph-based summarization algorithm named the Phrase Reinforcement (PR) Algorithm. The PR algorithm is able to generate one or more summaries by creating a weighted and directed acyclic graph representing the common phrases used around a starting topic phrase. By searching the graph for the path with the most total weight, the algorithm is able to find the most commonly used phrase and use it as the resulting summary. Our results demonstrate exceptional summaries for a variety of topics.
2. Two publications and presentations of the Phrase Reinforcement algorithm. The first publication, entitled *Automatic Summarization of Twitter Topics* (Sharifi, 2010) was presented at the National Workshop on Design and Analysis of Algorithms (NWDAA10, 2010) and published within their proceedings. The second publication, entitled *Summarizing Microblogs Automatically* (Sharifi, 2010) will be presented at the North American Chapter of the Association for Computational Linguistics (NAACLHLT, 2010) in Los Angeles, California on June 3rd, 2010.

3. A unique adaptation of the Term Frequency – Inverse Document Frequency (TF-IDF) algorithm for summarizing microblogs. Our approach allows compensating for the algorithm’s natural bias towards the longest sentences while also providing the flexibility to choose between different ratios of precision or recall. This approach has been demonstrated to produce summaries that approximate human-generated summaries using two commonly used evaluation metrics: Content and ROUGE-1. Finally, our results also significantly improve upon our published Phrase Reinforcement algorithm.

4. Baseline performance values for both microblog classification and summarization. Prior to our work, no published results were available for either of these topics (to the best of our knowledge). Without prior work, it required additional time and effort to generate expected ranges of performance for these two tasks in order to be able to evaluate our work. Given our use of industry-accepted evaluation metrics such as ROUGE-1, we have provided a set of baseline performance levels against which future efforts in microblog classification and summarization can be evaluated. This will hopefully motivate and simplify new efforts in this field.

5. Demonstration of the efficacy of the naïve Bayes classifier on a very short document type. Our results indicate that, on average, a single microblog post consists of only 11 words. Even with such a short document, we have demonstrated average F1-measure classification results of
approximately 0.74 for a set of 10 categories. These results demonstrate the capacity of the naïve Bayes classifier for classifying very short documents and the potential for using microblogs for many of the same applications as classifying traditional blogs.

6. Collaboration and active involvement of a UCCS undergraduate student through the Research Experiences for Undergraduates\(^\text{1}\) (REU) program sponsored by the National Science Foundation.

7. Availability of our developed software and data for future UCCS students interested in research in natural language processing and machine learning.

### 8.2 Future Work

While much progress has been made, the research presented in this thesis can be extended in a number of potential ways. For automatic classification of microblogs, there are a number of opportunities for enhancement. First, given our developed crawler that can automatically download copious amounts of microblogging data from Twitter, our training corpus could easily be extended to a much broader and deeper amount of training data. For example, one could extend our initial set of ten target classes to a broader set of classes based on our evaluation of popular Twitter topics. One could also generate additional training data for our existing categories in order to improve the classifier’s performance. Yan and Yan report that a naïve Bayes classifier’s performance improved monotonically with the amount of training data when classifying traditional blogs (Yan and Yan, 2006). In their work, they graphed the F-measure performance of a naïve Bayes classifier against their training data size. By doing so, they were able to extrapolate the expected performance of the Bayesian classifier based on the size of the training data. Since our work simply used a fixed size training corpus, one research opportunity would be to plot the average F-measure performance against different training corpus sizes.

---

This effort could be performed with our existing training data today without acquiring additional data—
though more training data might be insightful.

Other research directions in automated classification could include using a different classifier type
altogether. While we used a naïve Bayes classifier, many other researchers have had success at applying
other types of automated classifiers such as Support Vector Machines (e.g. Teng and Chan, 2006) or
Language Models (Gilad and Rijke, 2006). Implementing another classifier and comparing its
performance to our work could help distinguish between the different approaches and help understand
their relevancy towards microblogs. While additional classifiers could be used, one could also extend
our classification efforts to other sources of microblogging data. Many popular microblogging services
exists that could be used as alternative or supplemental data sources. For example, Facebook,
MySpace, and Google all provide microblogging services.

Instead of using a different classifier, one could further our efforts at optimizing the naïve Bayes
classifier we have chosen. One simple optimization was mentioned earlier in Chapter 3. When we
removed infrequent words and selected words with high mutual information, our thresholds for feature
selection were based on our observations of their effects on the training corpus. This is not an optimal
method as one is really interested in observing the effects of optimization on an unseen corpus, not the
training corpus. Mitchell describes a better approach to this problem. Instead of just using a training
and testing corpus, introduce a third tuning corpus that is used solely for optimization (Mitchell
describes this as a “validation set”). When determining the threshold for feature selection such as how
many infrequent words to remove from the training data, one could plot the classifier’s performance
against the tuning corpus as the threshold is varied instead of the training data (Mitchell, 1997).
Initially, as features are removed from the training data, the performance of the classifier on the tuning
corpus should improve as the amount of overfitting of the classifier is reduced. Eventually, however, the
reduction in features will begin to negatively affect even the turning corpus’s performance as too many features are removed from the classifier. At this inflection point, one can stop removing features from the training data. Mitchell indicates that this technique can more accurately predict optimal feature selection thresholds when optimizing a classifier. It is likely that this approach could improve upon our optimization method which only improved our results by 3-4%.

Finally, microblog classification can also be extended by integrating additional sources of information that are coexistent with the blog. For example, every microblog post is tagged with a time stamp that could be used in combination of classification in order to perform various forms of trend analysis. For example, one could understand what times of the days or days of the week a particular subject is popular. The results of this work could be used for target advertising to help display the most relevant ads for certain times of day. Other applications of classification and temporal analysis could be used for surveillance such as trying to automatically determine a person’s routine such as when a user discusses matters that relate to home or work. Last, automatically classifying a user’s sentiment or changes in sentiment towards a particular subject or product could have numerous business applications.

A number of opportunities exist for extending our work in microblog summarization as well. In Chapter 6, we presented our novel Phrase Reinforcement (PR) algorithm which automatically generated summaries by building a directed acyclic graph of the common phrases found within a set of training posts. A number of research opportunities exist for extending the application of this algorithm. First, our initial results indicate that we are able to generate many human-comparable summaries with as few as 100 input posts. However, it is unclear whether or not our results could be improved with a larger number of inputs. One research area is to provide a much larger set of input posts (e.g. 1000’s of posts) and evaluate whether or not a larger graph structure improves the resulting summaries. Instead of
simply providing more data, one could also attempt to improve the algorithm. One significant limitation of the algorithm is it does not provide any form of word stemming when attempting to determine the set of overlapping phrases. If words were stemmed, one could achieve a much greater amount of overlap and help target the algorithm towards a more representative common phrase. Such efforts may greatly improve the algorithm’s performance while also reducing the amount of data it requires.

Another algorithmic improvement could be developing a better weighting scheme. For example, we weight each node proportionally to their corresponding word frequency. However, another approach would be to weight the nodes based on a different metric. For example, one could use the TF-IDF term weighting formula developed in Chapter 7 as a substitute node weight. Perhaps by combining the two algorithms, better results could be achieved.

Finally, one could use our novel Phrase Reinforcement graph as a visualization tool. There are many visualizers on the web today that attempt to visualize the vast amount of microblogging data in order to help users derive additional meaning. For example, Twitter StreamGraphs\textsuperscript{15} displays a stacked set of “streams” (or rivers) indicating the most popular words associated with a query term. The relative widths of the streams reflect their popularity. Unfortunately, most of these visualizers available today, such as this one, are limited to single word associations. Fortunately, our Phrase Reinforcement algorithm already internally generates a weighted graph that represents the most popular phrases associated with a given topic. One could easily illustrate this graph structure using an open source graphing tool such as GraphViz\textsuperscript{16}. Such a graph could provide an insightful and useful form of summarization.

Besides extending the PR algorithm, there are a number of ways one could enhance the summaries themselves. Our work was aimed at providing very short 140-character-or-less summaries

\textsuperscript{15} http://www.neoformix.com/Projects/TwitterStreamGraphs/view.php
\textsuperscript{16} http://www.graphviz.org/
that reflected the primary meaning of a topic. However, such brief summaries are not always desired. One may also want to see second-order concepts associated with a topic in order to understand related issues. For example, our PR algorithm generated the summary “#usuallylieabout my age” for the trending topic “#usuallylieabout”. While users may be interested in knowing what people most lie about, they may also be interested in the top 10 things people lie about it. Fortunately, both of our summarization algorithms can be extended to provide a broader type of summary. For the PR algorithm, one could search the graph for the set of paths with the most weight rather than just the single most weighted path. A similar approach could be employed with the TF-IDF algorithm. Instead of finding the sentence with the most weight, one could find the set of sentences with the most weight.

The only problem with the above approaches is most likely several of the top sentences produced by the algorithm will have similar or duplicate information. In order to counter against such redundant information, one could measure the edit distance between the various resulting summaries and discard any summaries that are similar within some threshold. A better approach would be to first cluster the input posts to find group posts around a similar concept. Once the clusters were formed, they could independently be summarized using either the PR or TF-IDF algorithms without modification. The resulting summaries could then be grouped together into a longer coherent summary.

Another method for enhancing the summaries is to integrate temporal information similar to the described enhancement for classification. For example, one could summarize a given topic over a range of time in order to summarize how a topic develops. This approach was performed by Chieu who extracted summaries from a set of documents and then placed the summaries along a timeline spanning several months (Chieu, 2004). Given the immediate nature of microblogs, one could summarize a topic for much smaller ranges of time to illustrate how topics develop in real-time or over longer courses of time similar to Chieu’s work.
Finally, broader research opportunities exist in combining our work in automated microblog classification and summarization. For example, one such opportunity is attempting to use the techniques developed in this thesis as a source of real-time news generation. In order to create such an application, one could envision the following scenario. First, one could continuously collect microblogging data using a streaming source of data such as Twitter’s Streaming API\textsuperscript{17}. Next, one could dynamically classify the streaming data into a set of newsworthy categories using automatic classification similar to what we have done. Given a collection of related microblogs for a given category, one could next generate a set of topical trending topics, similar to Twitter’s trending topics, but dedicated to a single category. Finally, one could use the trending topics and related posts as input into one of the summarization algorithms developed within our work. Such efforts could produce real-time headlines for any desired numbers of topics. A source of real-time microblogging news could provide a crucial service such as disseminating critical information on a recent disaster or emergency.

\textsuperscript{17} http://apiwiki.twitter.com/Streaming-API-Documentation
References


North American Chapter of the Association for Computational Linguistics, Los Angeles, California, June 2010.


Appendix A: Manual Microblog Classification Instructions

In this appendix, we present the instructions presented to our three groups of volunteers for manually classifying the 1000 posts from our testing corpus.

**General Rules:**
1. Mark each post in the left hand margin in front of the post with the appropriate abbreviation.
2. Try to use the abbreviations below when marking each post. If it is easier, you may use your own abbreviations like Art, Tech, etc.
3. Mark as many categories as apply to each post. In other words, a single post can have multiple categories.
4. If no categories apply, mark the post N/A. Please don't leave it blank.
5. Be careful not to miss the very first post on each blog. It does not have a bold heading like the remaining posts. There should be 20 posts/blog.

<table>
<thead>
<tr>
<th>Category</th>
<th>Abbreviation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arts &amp; Entertainment</td>
<td>A</td>
<td>movies, tv, theater, ballet, music, etc</td>
</tr>
<tr>
<td>Business</td>
<td>B</td>
<td>stock market, finance, investing</td>
</tr>
<tr>
<td>Food &amp; Drink</td>
<td>F</td>
<td>cooking, recipes, wine, bartending, etc</td>
</tr>
<tr>
<td>Computer Games</td>
<td>G</td>
<td>computer games (note: be careful not to use Technology for this subject)</td>
</tr>
<tr>
<td>Health</td>
<td>H</td>
<td>fitness, dieting, working out, health issues, etc</td>
</tr>
<tr>
<td>Politics</td>
<td>P</td>
<td>election, candidates, political issues, etc</td>
</tr>
<tr>
<td>Science</td>
<td>Sc</td>
<td>general science, space, weather, etc</td>
</tr>
<tr>
<td>Sports</td>
<td>Sp</td>
<td>football, basketball, baseball, etc</td>
</tr>
<tr>
<td>Technology</td>
<td>T</td>
<td>google, apple, tech news, tech gadgets, etc</td>
</tr>
<tr>
<td>World</td>
<td>W</td>
<td>foreign affairs, foreign news, world travel, etc</td>
</tr>
<tr>
<td>None of the Above</td>
<td>N/A</td>
<td>posts that are comprehensible but don't apply to any of the above categories</td>
</tr>
<tr>
<td>Unsure</td>
<td>?</td>
<td>posts that are incomprehensible</td>
</tr>
</tbody>
</table>