2016

Automated Identification of Computer Science Research Papers

Tong Zhou
University of Windsor

Automated Identification of Computer Science Research Papers

By

Tong Zhou

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2016

Automated Identification of Computer Science Research Papers

by

Tong Zhou

APPROVED BY:

D. Yang
Department of Mathematics and Statistics

A. Ngom
School of Computer Science

J. Lu, Advisor
School of Computer Science

May 19, 2016

DECLARATION OF ORIGINALITY

ABSTRACT

The fast growing speed of the size of scholarly data have made it necessary to find out efficient machine learning ways to automatically categorize the data. This thesis aims to build a classifier that can automatically categorize Computer Science (CS) papers based on text content. To find out the best method for CS papers, we collect and prepare two large labeled data sets: CiteSeerX and arXiv, and experiment with different classification approaches including Naive Bayes and Logistic Regression, different feature selection schemes, different language models, and different feature weighting schemes. We found that with large size of training set, Bi-gram modeling with normalized feature weight performs the best for all the two data sets. It is surprising that arXiv data set can be classified up to 0.95 F_1 value, while CiteSeerX reaches lower F_1 (0.764). That is probably caused by labeling of CiteSeerX is not as accurate as arXiv data set.

AKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor Dr. Jianguo Lu for his constant guidance and encouragement during my whole Master's period in the University of Windsor. Without his valuable help, this thesis would not have been possible.

I would also like to express my appreciation to my thesis committee members Dr. Alioune Ngom, Dr. Dilian Yang and Dr. Yung Tsin. Thank you all for your valuable guidance and suggestions to this thesis.

Meanwhile, I would like to thank Yi Zhang for all the helps during my research process, including sentence2vec modeling on arXiv data set, duplicate detection on CiteSeerX data set; and to thank Dr. Yan Wang for the CiteSeerX title correction researches.

Last but not least, I want to express my gratitude to my parents and my friends who give me consistent help over the past two years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Scholarly data is very important for researchers to search for related papers to build the basis for their own research ideas. However, the size of the scholarly data has increased dramatically which cause problems for researchers to efficiently search for relevant and high quality papers. Reported by [23], the most popular scholarly search engine Google Scholar has gathered and indexed more than 160 million scholarly documents into their database. A newly developed database: Microsoft Academic Search (MAS) [29] also contains more than 80 million publications. A major advantages of MAS is that it provides the hierarchical research fields that papers belong to (e.g. Computer Science, Math, Physics, Economics), which can be treated as a classification system. The academic paper classification is very important and useful. For example, when building an academic search engine specializing in a certain area or recommending relevant papers, we need to classify whether a document crawled from the Web belongs to this area. There are numerous techniques to address these problems, but the basic building block is the text classification techniques based on supervised machine learning method.

There are three main aspects regarding techniques of text classification: text preprocessing, feature extraction/selection, classifier. There are lots of mature algorithms and systems related to these techniques. Take algorithms to train classifiers for example: Naive Bayes [21], k-NN [35], SVM [13], Logistic Regression [12]. Each document in training data set is processed into a feature set, along with their pre-assigned class labels (categories), inputted into the classifier. The classifier is trained by using the feature sets of all training documents. Then, for newly received test documents, the trained classifier extracts feature sets from test documents, and process the feature sets based on the learned knowledge from training set to automatically predict the class labels for test documents. Although text classification is widely studied, there are only few studies focusing on academic papers [7] [14] [19] [3] [5]. We are not clear how accurate we can classify academic papers, and what are the best methods. In this thesis, we specifically focus on the classification of Computer Science (CS) papers. Our research questions are: 1) Can we tell the difference between a CS paper and a non-CS paper? 2) What is the best method for academic paper classification? 3) What are the best parameters for each method? Each classification method has many parameters. Take Naive Bayes [21] method for example, there are different models [6] (e.g.,Uni-gram, Bi-gram), different feature selection methods [28] [37] (e.g., mutual information (MI), χ^2, pair-wise mutual information (PMI)), different pre-processing [1] (e.g., stop words, stemming), systemic bias correction [27] (e.g., length normalization and weight adjustment). Due to the unique characteristics of academic papers, the choosing of correct parameters need to be investigated. 4) Whether the neural network approach helps in this area? Given the recent success of deep learning in many domains, we need to check whether approaches spawn from word2vec [22] and sentence2vec [16] can improve the performance.

In this thesis, we conducted a series of experiments on various scholarly data sets, including arXiv [32] and CiteSeerX [10] to find answers of the above questions. Compared with most of the previous works, our main improvement is the huge size of the data set used and various methods and parameters tested. In arXiv data set, each document is labeled with an specific research area such as CS, Math, Physics, and the label is considered accurate because it is self-identified by its authors. The most recent arXiv collection contains 84,172 CS papers and 575,043 non-CS papers. We removed duplicates to avoid the noise introduced by repeated papers. Also, we used sampling to make the sizes of positive and negative classes balanced to eliminate the imbalanced data problem [27]. After duplicate removing and balancing, our arXiv data set collection contains 80K CS papers and 80K non-CS papers, the text of each paper contains title and abstract. On the other hand, CiteSeerX provides a much bigger data set. Based on their downloadable data source there are 2.1 million documents provided and the whole paper text that parsed from the original PDF file is provided. However, documents are not labeled and there are even documents not belong to academic papers (e.g. manual, report, slides) because all the documents are crawled from the web. In order to solve the labeling problem, we use DBLP [17] data set which is a manually maintained Computer Science bibliography to find intersections with CiteSeerX papers and label the intersection part as CS papers while the remaining CiteSeerX papers as non-CS papers. As a result, we got 665,483 CS papers and 1,452,639 non-CS papers. We also use sampling to make the data set balanced and our experimental CiteSeerX data set contains 600k CS papers and 600k non-CS papers.

The methods we tested include Multinomial Naive Bayes (MNB) on Uni-gram and Bi-gram models, and MNB and logistic regression on vector representations generated using sentence2vec. We chose these methods due to the scalability issue. We tried other methods, such as the well-known SVM, without success. The experiments are carried on two powerful servers with 256 GB memory and 24 core CPU. Our results show that CS papers can be classified with high accuracy in arXiv data set, while relatively lower accuracy in CiteSeerX data set. As for arXiv data set, most methods can achieve an F_1 value above 0.9. The best method is the Bi-gram model using MNB. The out-of-box sentence2vec is inferior to the Bi-gram model by almost 2 percent. Interestingly, removing stop words helps in all the methods, even in sentence2vec, while stemming has limited impact. Historically, PMI is considered inferior in text classification [34]. We show that when the training data is large, it out-performs MI and χ^2. And for CiteSeerX data set, the highest F_1 value is around 0.764, also achieved by Bi-gram models, with PMI selected features. Due to labeling and text cleanness problems, CiteSeerX classification results are not as good as arXiv. Moreover, through several feature weight normalization approaches, the classification F_1 value can be improved, especially the gap between precision and recall can be lowered which infer a better classification result even F_1 measure values are similar.

CHAPTER 2

Review of The Literature

This chapter reviews the previous researches and publications on academic paper classification approaches and some related text classification techniques.

2.1 Academic Paper Classification Approaches

2.1.1 Classify Biomedical Articles

[7] automatically classify whether a biomedical article has direct relations to the areas of 6 proteins / polypeptides topics. Their classification system is a typical two class scenario (positive class and negative class).

Data Set and Labeling

They gathered a corpus of abstracts from the MEDLINE database that contains large amount of biological article information. They query the names of the six proteins / polypeptides topics and download the top 500 returned articles for each protein and finally obtained a data set with 2,889 abstracts. One of the authors manually labeled each of the 2,889 abstracts. The criterion to label an abstract into positive class is that the abstract must clearly and evidently indicate that protein x is found in location y of this abstract.

Methodology

Before applying the classification algorithms to their data set, they pre-process the abstract text by using stemming technique. They applied their baseline method: sentence co-occurrence predictor and Naive Bayes as classification algorithms to apply to the stemmed data set. Their baseline algorithm is a simple method that treat a document as positive only if a protein and sub-cellular location occur in the same sentence inside the document. As for Naive Bayes, they used class term frequency and add-one smoothing to estimate the conditional probability $P(t|c)$ for a word t being in a class c, which can be also called as Multinomial Naive Bayes. They used cross validation evaluation method to test the class label (positive or negative) for each document (abstract) by using other documents as training set. The measurement they used are *precision* and *recall*, *precision* is the ratio of correct positive predictions out of all positive predictions, and *recall* is the ratio of correct positive predictions out of all real positive documents.

Results

They plotted the precision and recall curves for the baseline algorithm: sentence co-occurrence and Naive Bayes. At the same recall level, Naive Bayes can reach higher precision. They concluded that when recall = 25%, the precision for baseline algorithm is 44% and the precision for Naive Bayes algorithm is 70%.

2.1.2 Categorize Papers based on ACM CCS Categories

[14] proposes an approach to categorize CiteSeer papers into 268 different categories based on the ACM CCS class definition.

Data Set and Labeling

Their data set was obtained from CiteSeer (later known as CiteSeerX). They mentioned that they gathered 1,164,939 academic papers and found out 31,121 of them contain author-assigned ACM tags. The ACM tags belong to the ACM's Computing

Classification System (CCS), which is a three level classification tree that defines the hierarchical set of computer science research categories. There are 369 different ACM CCS tags found among all the 31,121 papers. However, 101 ACM CCS tags own less than 10 papers, which they considered too small. They used the remaining papers within the remaining 268 ACM CCS tags as their labeled training set, each paper is labeled with one unique ACM CCS tag.

Methodology

The authors didn't mention the details of their text pre-processing steps. Assume they used the original un-normalized text as input and separate each distinct word as features. As for their training set used for the classifier, they sampled 10 papers in each of the 268 ACM CCS categories and construct a training set with 2,680 papers. They only used one classification algorithm: k-NN, which is an approximity based algorithm that predict the class label of a test document by comparing the classes distribution of its nearest k neighbors, and choose the most common (or top k common) class among the neighbors to assign to the test document. They choose the top 3 (or 6, 9, 12) common classes assigned to each test document.

They didn't use the cross validation scheme to evaluate the classifier, instead, they used the 2,680 labeled papers to test all the remaining (around 1 million) un-tagged papers in their CiteSeer data set collections. They built a user based system, to let users judge the precision of the ACM CCS tags assigned to the test papers.

Results

Their system users were asked to judge the relevance of the returned ACM CCS tags by three precision levels: very relevant (level 2), relevant (level 1), or irrelevant (level 0). Their average precision value is 1.4.

2.1.3 Classify using Logistic Regression

[19] conducted text classification on multiple data sets by using logistic regression algorithm.

Data Set and Labeling

They experimented with three kinds of data sets. The first data set is Cora, which contains 4,187 papers in the research area of machine learning, each of the paper has pre-defined class label that categorize the paper into one of the seven possible sub-topics within machine learning. The second data set they used is CiteSeer. They gathered around 3,600 papers with pre-defined six categorizes: Agents, Artificial Intelligence, Database, Human Computer Interaction, Machine Learning and Information Retrieval, each CiteSeer paper is assigned with one of the six categorizes. The last data set they used is WebKB, which contains web pages from four computer science departments. The class labels are also pre-defined, which are the topics of the web page. The categorizes include faculty, student, project and course. They gathered 700 web pages, each page can be treated as a text document with one of the four categorizes.

Methodology

They used stemming and stop words removal to pre-process the document text of both the three data sets. They also removed rare words in the text, but they didn't provide the specific definition of rare word. For Cora data set, after pre-processing, the whole data set contains 1400 different words (size of dictionary). For CiteSeer data set, the size of the dictionary is 3000; and the dictionary size of WebKB data set is 2338.

The authors used Logistic Regression as their classification algorithm to calculate the conditional probability distribution of document X under class c by $P(c|OA(X), LD(X))$. $OA(X)$ is the content attributes of document X, which include text information such as title, abstract and full text, and $LD(X)$ is the link features (citation links). Note

	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F_1 Measure
Cora	0.674	0.662	0.626	0.643
CiteSeer	0.607	0.551	0.552	0.551
WebKB	0.862	0.876	0.795	0.832

TABLE 1: Classification Results made by [19]

that this paper used the citation links to improve the classification results, compared with pure text classification using Logistic Regression. Since we only focus on full text classification, we only discuss about their full text classification methodology and results. In the multi-class classification scenario, they trained one-against-others model for each class to transfer it into two-class classification problem. For the testing process, select the class that has the highest posterior probability.

Results

For the three data sets, the authors separated them into three sub sets equally to conduct 3-fold cross validation to test the accuracy of their classification system. Table 1 shows their classification results when only using content $OA(X)$ as the attributes of documents. We can see the accuracy and F_1 measure for WebKB are much higher than Cora and CiteSeer.

2.1.4 Lower Feature Dimensionalities

[3] discussed about lowering feature dimensionality to simplify the complexity of classifiers.

Data Set and Labeling

The authors used the similar data sets with the previous related work. Two data sets are used, one is Cora, the other is CiteSeer. Different from [19], they didn't explore the citation links between papers. Their gathered Cora data set contains 3,191 machine learning papers, each of them has been labeled into one of the seven classes. CiteSeer contains 3,186 labeled papers, in total there are six classes.

Methodology

Compared with [19], this paper dig out more about the effect of feature dimensionality to the classification results. They mainly used three different kinds of methodologies to lower the feature dimensionality:

- Mutual Information algorithm: measure the dependency between a feature and a specific class. If a feature is highly related to only one specific class, the feature should has higher scores.
- Topic Models: transfer the text of the paper (w_l, c_l) into a set of topics $(\theta_1^l, ..., \theta_m^l, c_l)$, θ_i^l is the probability of topic i in paper w_l, c_l is the class label of paper w_l.
- Feature Abstraction: find clusters of similar features to reduce the feature dimensionality. Input the whole data set $\mathcal{D} = (W_l, c_l)_{=1,...,N}$, output an abstraction hierarchy τ over the whole vocabulary space $\mathcal{V}$. Abstraction hierarchy is a rooted tree that internal nodes correspond to abstracted features (clusters of words).

They used two classification algorithms: SVM and Logistic Regression.

Results

They used 5-fold cross validation for all the evaluation experiments. They showed several plots to present the comparison of classification results. Different plots show the feature selection results comparison on different data set (Cora / CiteSeer) and different classification algorithms (SVM / Logistic Regression). From their experiment results, Logistic Regression and SVM only have slight performance differences, and Cora data set can produce higher classification accuracy than CiteSeer data set.

However, the performance of different feature selection algorithms varies greatly. The performance of feature selection algorithms are showed by classification accuracy curves. The authors used two topic models, one is topic distribution, the other is topic words. Topic distribution performs much worse than Mutual Information and abstract features, and topic words performs similar (slightly worse) than feature abstract. In all of their plots, Mutual Information accuracy curves are similar with abstract features accuracy curves, and both of them can reach a better classification accuracy compared with using all features to do classification.

2.1.5 Feature Representation for Academic Papers

[5] proposed a new way of feature representation to improve the classification system especially for academic papers. Their classification goal is to identify whether a crawled document is an academic paper.

Data Set and Labeling

They used two data sets. They crawl 833 documents from the web by themselves to construct their first data set, and the second data set is obtained from CiteSeerX, including 1,409 documents. They manually labeled each document into positive or negative class. Positive corresponds to research articles that including academic papers published in conferences, journals, or book chapters, technical reports. Negative corresponds to other non research article documents, including long documents, books, slides, brochures, even news, agenda, etc.. Based on their manual labeling, for the first crawled data set, they labeled 352 docs into positive and 481 as negative. For the second CiteSeerX data set, they labeled 811 docs into positive, while 598 as negative.

Methodology

They proposed the novel way for feature representation that they called: structural features. Based on the trait of academic papers, the authors summarize 27 different structural features to represent a document:

- 2 File Specific Features: FileSize, PageCount
- 11 Text Specific Features: DocLength, NumWords, NumLines, NumWordsPg, NumLinesPg, RefRatio, SpcRatio, LnRatio, UcaseStart, SymbolStart

- 6 Section Specific Features: Abstract, Introduction, Conclusion, Acknowledge, References, Chapter
- 8 Containment Features: ThisPaper, ThisBook, ThisReport, ThisThesis, ThisManual, ThisStudy, ThisSection, TechRep

Note that Section Specific Features and Containment Features are valued with binary $True$ or $False$. For Section Specific Features, if a document has the corresponding sections, e.g. abstract, they the feature is valued as $True$. For containment features, if the corresponding word occurs in the document, then it is labeled as $True$. Based on their method, the feature dimensionality of each document is lowered into 27 dimensions, which is much smaller than traditional bag of words.

Results

They also compared their method with bag of words feature representation. For the first data set (crawled), there are 7,443 distinct words in total, and 15,248 distinct words in the second data set (CiteSeerX). They experimented with 5 algorithms: SVM, Logistic Regression, Naive Bayes, Decision Tree, Random Forest.

For the results of the crawled data set, using their structural features, SVM achieved the highest F_1 measure value, which is 0.854, and except Naive Bayes, all the other algorithms produced similar classification results. By using bag of words, Naive Bayes reaches the best F_1: 0.749. For the CiteSeerX data set, using structural features, still, except Naive Bayes (0.801), all the other four algorithms performs similar, while Random Forest is the best (0.863). However, if using bag of words model, Naive Bayes (0.772) outperforms SVM (0.680) greatly.

2.2 Researches on Text Classification and Feature Selection Algorithms

In this section, we focus on more comprehensive comparison of different text classification and feature selection algorithms. Most of the previous researchers conducted more thorough text classification experiments on more popular data sets such as Reuters-21578, instead of academic paper data sets. This section aims to summarize their conclusions on most commonly used algorithms.

2.2.1 Comparison of Classification Algorithms

[36] thoroughly compares the performances of 5 different text classification algorithms: SVM, k-NN, LLSF (Linear Least Squares Fit), NNet (Neural Network) and NB (Naive Bayes).

They also simply used two popular feature selection algorithms: χ^2-$statistic$ and $information\ gain$ to produce the feature ranks. However, the major goal of this paper is still the performance comparison of different classification algorithms, the feature selection algorithm is only an auxiliary tools for test the classification algorithms based on different feature sizes. Their experimental settings are shown as follows:

- SVM: use both linear and non-linear kernels provided by SVM_{light}.
- k-NN: set k to 45 which is based on their previous conclusion of parameter optimization.
- LLSF: set singular value to 500 based on their previous conclusion of parameter optimization.
- NNet: set the number of hidden units in the middle layer to 64.
- NB: default parameters.

The data sets they used are Reuters-21578 and Reuters-21450. They used cross validation scheme to evaluate the classification algorithms, the measurement used include micro average precision, micro average recall, micro average F_1 and macro average F_1. Based on the results of experiments on full feature set, for micro average F_1, they concluded the performance ranking for the 5 algorithms as follows:

$$SVM > kNN \gg \{LLSF, NNet\} \gg NB$$

And for the macro average F_1, they concluded the performance ranking for the 5 algorithms as follows:

$$\{SVM, kNN, LLSF\} \gg \{NB, NNet\}$$

Actually for all the 5 algorithms, with the increase of the feature size, the trend of F_1 measure is also increasing. When the feature size is large enough, the F_1 measure of all the 5 algorithms can increase higher than 0.8. But still, the performance curves of SVM, k-NN and LLSF are better than NB and NNet.

2.2.2 Improvement on Naive Bayes

Although [36] concluded that more sophisticated algorithms such as SVM can outperform Naive Bayes, However, the time and space complexity of executing SVM are much higher than Naive Bayes. Especially, in the case of text classification, the number of features can be much larger compared with other kinds of classification scenario. Therefore, Naive Bayes is still very popular in the field of text classification. Actually several previous researchers have proved that even though the independent assumption of Naive Bayes can be unrealistic in most of the text classification scenarios, Naive Bayes can still performs surprisingly well [21].

The authors in [21] illustrated two model for Naive Bayes: Bernoulli Model and Multinomial Model. Bernoulli Model uses the presence or absence of words in a text document as features to represent a document. The words' frequencies in document are not taken into consideration. Multinomial Model in contrary, capture the frequencies of features in each document, and use the features' total class frequencies (total occurrences in all documents of the class) to estimate the probabilities. They used four data sets (Yahoo, Newsgroups, Industry Sector, WebKB) to do experiments and their results showed that Multinomial Model performs better than Bernoulli Model in text classification scenarios.

Another paper [27] described how to improve Naive Bayes algorithm to make it more accord with Multinomial Model. They used three steps processing to normalize feature weight instead of using term frequency (TF) directly:

1. **Use Log Term Frequency**: [27] claimed that power law distribution can better model text. Their experiments showed that usually term frequency probability distribution curve has a heavier tail than power law distribution curve. When the term frequency value increase, the gap between the term frequency of real text and power law keep expanding, which means the term frequency probability of power law drops quicker than real text. [27] also showed that log term frequency can make term frequency probability distribution proportional to power law distribution, which can model the text and feature weight better. Using log calculation, the term frequency can be transformed into:

$$d_{ij} = \log(d_{ij} + 1) \tag{1}$$

2. **inverse Document Frequency (iDF)**: discount the weight of features by their document frequency. Document frequency of feature d_{ij} means how many documents have d_{ij}. The iDF calculation can be expressed as follows:

$$d_{ij} = d_{ij} \log \frac{\sum_k 1}{\sum_k \delta_{ik}} \tag{2}$$

3. **Length Normalization**: discount the term frequencies of long documents to diminish the dominant effect of long documents to the classification results.

$$d_{ij} = \frac{d_{ij}}{\sqrt{\sum_k (d_{kj})^2}} \tag{3}$$

For the smoothing of Naive Bayes parameter estimation, they also experiments lots of smoothing factors. Instead of add-one smoothing which is a commonly used factor, they concluded that factor with value 10^{-4} is better for Multinomial Model of Naive Bayes algorithm. They used three different data sets: Industry Sector, 20 Newsgroups and Reuters to do the experiment, the classification results all show that the normalized feature weight can produce much better classification accuracy.

2.2.3 Comparison of Feature Selection Algorithms

There are two major branches of feature selection schemes, one is filter method, the other is wrapper method. Due to the complexity of wrapper method and the high dimensionality of feature space in text classification scenario, filter method is commonly used. [28] compared and concluded the most efficient filter feature selection algorithms for text classifiers. They experimented with two data sets: Reuters-21578 and small portion of Reuters Corpus Version 1 (RCV1). The filter feature selection algorithms used include Document Frequency (DF), Information Gain (IG), χ^2. They concluded that χ^2-statistic (CHI) consistently outperformed other feature selection criteria for multiple classifiers and the two data set they used (Reuters-21578 and small portion of Reuters Corpus Version 1). They also concluded that cut the low DF features (delete them from feature set) can boost the performance for almost all the classification algorithms (including Naive Bayes, k-NN, SVM.), which also showed that χ^2-statistic (CHI) is unreliable for rare words. In their Micro-F_1 measure for the different combinations of classifiers and feature selection algorithms, the results are highly clustered by classifiers, and SVM outperforms other classifiers which Naive Bayes is the worst. However, they also mentioned that a good feature selection method enables KNN to surpass SVM's performance.

[30] proposed another feature selection approach especially optimized for Naive Bayes. They claimed that unlike the existing filter feature selection algorithms, their approach can maximize the discriminative performance especially for Naive Bayes text classification. Their approach is based on $\mathcal{J}$*-divergence*, which is derivative from KL-divergence. Their calculation equations are shown as follows:

$$\begin{gathered}\mathcal{KL}(P_1, P_2) = \int_x p(x|H_1) \log \frac{p(x|H_1)}{x|H_2} dx = \sum_{i=1}^{M} p_{i1} \log \frac{p_{i1}}{p_{i2}} \\ \mathcal{J}(P_1, P_2) = \mathcal{KL}(P_1, P_2) + \mathcal{KL}(P_2, P_1)\end{gathered} \tag{4}$$

The transformation of the first equation is due to the discrete distribution of features in text classification scenario. M is the vocabulary size, and $p(x|H_i)$ is the conditional probability of feature x being in class c_i. They experimented with 20-

Newsgroups data set, and the classification result curves showed that their proposed feature selection algorithm performs better than DF and χ^2. Especially when the number of features is small, the performance gap between traditional method (such as χ^2) and their proposed $\mathcal{J}$*-divergence* algorithm is bigger.

2.3 Summary

As a conclusion, the most common drawbacks of the previous work is the small size of training data set. None of them used larger than 10,000 documents to train classifier. Especially in the academic paper classification scenario, due to the huge amount of academic research fields and the long document length, the amount of terminologies used can be quite huge so that smaller data set cannot well represent enough features used in academic papers. As for the academic paper classification results achieved by previous researchers, most of them are not good enough. [7] claimed 0.7 precision with Naive Bayes; [19] achieved 0.551 F_1 and 0.6 accuracy with Logistic Regression on CiteSeerX data set, and with almost the same data set, [3] achieved 0.7 accuracy with SVM with reduced feature dimensionality. [5] reached 0.86 (CiteSeerX) with Random Forest on structural features, 0.77 (CiteSeerX) with Naive Bayes on bag of words model.

When the size of training data set largely increase, lots of problems need to be considered. Firstly, the classifier efficiency problem need to be considered, more complex algorithms such as SVM maybe infeasible on bigger training data size. [14] uses k-NN to train a classifier with 268 classes, and each class only has 10 training documents. The training data size is extremely small and the efficiency of k-NN is also very low compared with other algorithms. Secondly, labeling problem can also be a difficulty. [14] [19] [3] described that the labels are provided by the data set itself, while [7] [5] manually labeled the whole training data set. However, when the data set size is too large, it is impossible to manually inspect every document to assign class labels. Thirdly, the related works on academic paper classification didn't well utilized the feature selection algorithms. For example, the best reported algorithm χ^2

haven't been used. [5] proposed the structural features. Although they proved that their classification results are better than using bag of words features, the workload for finding out and valuing each structural features in every document is very huge, which will become an impossible task if the training set size is large.

Apart from the above difficulties, advanced language modeling such as N-gram language modeling haven't been utilized into academic paper classification scenario, the previous researchers merely use single words as features, didn't consider the phrases. Moreover, length variation issues of academic papers haven't been considered yet, which can also cause problems to the classification system. Advanced feature weight representation haven't been considered too, most of them only use the original value of word occurrences in document (or called Term Frequency) to represent the feature weight, which is not strong enough for a more complex text environment.

CHAPTER 3

Data Set

The first step of text classification is to obtain the training data set contain enough amount of both positive (CS) and negative (non-CS) class documents. Compared with previous works, one of the major improvement of this thesis is to largely increase the size of training data set to include enough instances that can represent multiple research fields. In the sections below, we will introduce two big scholarly data set that can be used as our big training data set.

3.1 CiteSeerX

CiteSeerX [10] [15] [18] [33] is a huge academic research paper digital library. For more than ten years, it automatically crawled more than 2 million scientific literature. The data set can be obtained from Amazon S3. For each academic paper, CiteSeerX data set provides the paper full text and the paper metadata. Full text is parsed from the original crawled PDF format documents. Metadata is further parsed from the full text using SVM header parser. Metadata includes paper title, authors (name, affiliation), published venue, published year, abstract, etc. CiteSeerX also uses ParsCit to extract the citation strings from the reference part of documents. The original text of each reference is extracted as raw citation string and metadata in raw citation string is further parsed including cited paper title, authors, year, published venue etc..

	DBLP	**CiteSeerX**
#Papers	2,797,143	2,118,122
Paper Areas	Computer Science	All Areas
Data Collection	Manually Collected	Crawled
Metadata Richness	Title Authors Published Venue Pages Year	Title Authors Affiliations Published Venue Pages Year References
Metadata Correctness	High (manually maintained)	Low (Automatically parsed)
Full Text	No	Yes

TABLE 2: Comparison between CiteSeerX and DBLP

3.1.1 Labeling using DBLP

Based on our observation, part of the academic papers in CiteseerX data set don't belong to the field of computer science, even don't belong to academic papers. We have found papers that belongs to the field of chemistry (e.g. Document 10.1.1.157.7467 with title "Educating Chemical Engineers in Product"), Geology (e.g. Document 10.1.1.6.7639 with title "Earthquake nucleation and its relationship to earthquake clustering"), even more unrelated areas such as education (e.g. Document 10.1.1.123.9915 with title "National Report on the Development of Education in Kenya").

On the other hand, DBLP [17] is a computer science bibliography. It has collected computer science research papers that published in all important journals and conferences on computer science. DBLP splits research papers into several categories based on their published sources, the major paper types are conference papers, journal papers, books, Ph.D. thesis. For most of the papers, the metadata contains titles, authors, years, pages, URL of the paper, venue name (conference or journal), crossref (the ID of the venue). However, the DBLP data set didn't contain the abstract, full text and references. Therefore, the DBLP data set itself can't be used to do full text related experiments.

Table 2 lists the major differences between CiteSeerX and DBLP data set. Considering CiteSeerX provided paper full text, also contained papers from both Computer Science (CS) research areas and not Computer Science (non-CS) research areas, it can

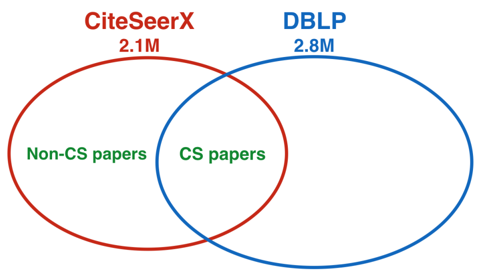


FIGURE 1: Creating big scholarly training data set

be used as a good source of our big scholarly training data set. The only problem is how to separate the CiteSeerX data set into two parts of papers: CS and non-CS. Our solution is to use DBLP data set to identify Computer Science papers in CiteSeerX data set. Since all DBLP papers are belong to Computer Science categories, the intersection set of CiteSeerX and DBLP can be treated as CS papers, the remaining CiteSeerX papers can be treated as non-CS papers.

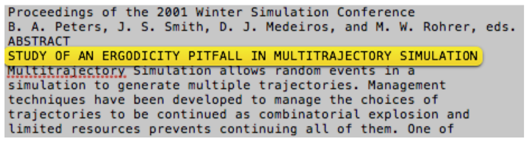
3.1.2 Separate Classes: Merging of CiteSeerX and DBLP

As shown in Fig. 1, CiteSeerX data set can be separated to two subsets using DBLP. The merged subset can be labeled as CS papers, the remaining CiteSeerX subset can be labeled as non-CS papers. These two subsets construct our first big scholarly data set, which contains 2.1 million papers.

Problems and Difficulties

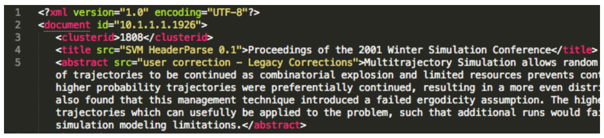
One obvious solution of merging CiteSeerX and DBLP is to utilize the metadata of papers. Since both CiteSeerX and DBLP provide paper metadata (e.g. title), it is easy to just select one kind (or multiple kinds) of metadata and perform string comparison to the metadata in CiteSeerX and DBLP.

However, the problem is the low accuracy of automatically parsed CiteSeerX metadata. Based on our manual inspection and investigation, CiteSeerX data set contains mal-formed and incorrect extracted metadata. Error occurs when automatically extract metadata [4]. Full text of the academic papers has uncertain and disordered



```
Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.
ABSTRACT
STUDY OF AN ERGODICITY PITFALL IN MULTITRAJECTORY SIMULATION
Multitrajectory Simulation allows random events in a
simulation to generate multiple trajectories. Management
techniques have been developed to manage the choices of
trajectories to be continued as combinatorial explosion and
limited resources prevents continuing all of them. One of
```

(a)



```
<?xml version="1.0" encoding="UTF-8"?>
<document id="10.1.1.1.1926">
  <clusterid>1808</clusterid>
  <title src="SVM HeaderParse 0.1">Proceedings of the 2001 Winter Simulation Conference</title>
  <abstract src="user correction - Legacy Corrections">Multitrajectory Simulation allows random
  of trajectories to be continued as combinatorial explosion and limited resources prevents cont
  higher probability trajectories were preferentially continued, resulting in a more even distr
  also found that this management technique introduced a failed ergodicity assumption. The highe
  trajectories which can usefully be applied to the problem, such that additional runs would fa
  simulation modeling limitations.</abstract>
```

(b)

FIGURE 2: Full Text and Extracted Metadata

formats. Lots of paper titles didn't appear in the start position of the full text as expected. Titles and author lists mixed together with the full text. The disordered full text formats lead to the difficulty of parsing metadata. Even though they proposed the metadata cleaning process [24], errors still exist. For example, some section subtitles were identified and parsed as paper title, such as "related work" (e.g. Document 10.1.1.117.7236), "experiment" (e.g. Document 10.1.1.224.908), "acknowledgements" (e.g. Document 10.1.1.101.7199). This may cause by wrongly identifying of the title position in the full text. In some other cases, the venue information or author affiliations were identified as title strings. As the example shown in Fig. 2, the first line of the full text is the name of the published conference of this paper, and the real paper title is located in the 4-th line. However, CiteSeerX parsed metadata treat the first line as paper title. Moreover, [4] proposed an algorithm to purify CiteSeerX by using DBLP, which can be equivalent to the merging of CiteSeerX and DBLP. Instead of using pure string comparison, they use N-gram language modeling to normalize both CiteSeerX and DBLP paper titles. Then they use Jaccard Similarity to compare the two normalized paper titles. Meanwhile they also use author set $((a_c \subseteq a_d$ or $a_d \subseteq a_c))$ and page comparison $(p_d \approx p_c)$ as an auxiliary comparison method. Their result shows that when using 3-gram to normalize titles and only using paper title to compare, the highest evaluation score (F_1 measure: a harmonic mean between precision and recall) can reach 0.77 when set Jaccard Similarity threshold to 0.7. We consider that their merging result is not good, with nearly 30% wrongly identified papers. Moreover, even though their 0.77 result is not convincing too, since lots of the paper title metadata in CiteSeerX are inaccurate.

In order to avoid the inaccurate CiteSeerX metadata, we propose a novel approach to find intersection between CiteSeerX and DBLP. Our algorithm is described in the

	indexed fields (for each paper).
	First 400 words of full text.
	Paper ID.
	Title Metadata.
	Authors Metadata.

TABLE 3: Full Text Index for CiteSeerX

following section.

3.1.3 Merging Algorithm

The idea of our approach is to utilize the paper full text in CiteSeerX data set, since the real metadata can be found in paper full text. For example, as shown in Fig. 2 (a), all the essential metadata can be found in paper full text. Title is found in the 4th line. Published venue information is found in the 1st line, and author list information is found in the 2nd line. Having this basis, the core concept of our approach is to search DBLP metadata in CiteSeerX paper full text. Note that the metadata in DBLP is manually maintained, the accuracy and precision of the metadata can be well ensured.

- **Incrementally search** DBLP metadata in CiteSeerX paper full text.
- Searching priority: $title > authorList > venue$
- Gradually reduce the matched CiteSeerX papers to minimum level to obtain more precise match.

Indexing CiteSeerX Full Text

The total amount of whole CiteSeerX paper full text is extremely large (more than 100 GB). It is impossible to directly search DBLP metadata in 100 GB full text. Thus, we use Lucene [11] to build index for the CiteSeerX full text to increase the searching efficiency.

Different index fields can be created for each document, and all the fields are related. Indexing can significantly increase the efficiency when searching for matched strings. As we can see in Table 3, we created 4 fields in Lucene to index the contents separately. Among these fields, Paper ID is the unique identification of a paper in CiteseerX (DOI). Doc ID can be used to quickly locate the relevant files in the data set. And text field is extracted from the full text files, but we only index the first 400 words of the text instead of the whole text, since the purpose of indexing text is to find the possible titles and the titles usually appear in the first few lines. And moreover, if we include too many text of a paper (especially the tailing part), it can cause some other paper titles included, e.g. cited paper titles listed in references. Paper full text need to be normalized before add into Lucene index. We only do lightweight normalization for full text here by lowering case for all English letters and delete punctuation. As for index searching, we use a highly efficient Lucene function called: $MMapDirectory$ to search the full text field of the index. We also visualize our indexed CiteSeerX data set via web page search engine using Django web framework [8].

Incremental Search

Algorithm 1 shows the whole process of data set merging to create our big scholarly training set. For each of the 2.8 million DBLP papers, use title metadata to search the CiteSeerX full text index I_c. If only one CiteSeerX match returned, directly add the match into the merged data set. However, if multiple CiteSeerX matches returned, more metadata need to be used to further purify the matching result. Considering the case that DBLP title may match some incorrect titles, which means match a part of the full text that is not the CiteSeerX paper real title. Such cases could happen especially when DBLP title is short. Thus, using other DBLP metadata (e.g. authors, venue) can prevent such wrong matches. We call this process as "incremental search".

As can be seen in Algorithm 1, if multiple CiteSeerX papers are matched with one DBLP title searching, incremental search further extracts authors and venue metadata and split the two metadata into words, search each word in every matched CiteSeerX papers' full text. There are two stop criterion for the incremental search iteration process: one is all of the $authorList, venue$ words has been searched and all the remaining CiteSeerX matches will be retained; the other is before all the

Algorithm 1 Data Set Merging Algorithm

```
Require: DBLP $\mathcal{D}$, CiteSeerX full text index $I_c$
  initialize merged set $\mathcal{C}_D = \emptyset$
  for each paper $p_i$ in $\mathcal{D}$ do
    $t_p \leftarrow getTitle(p_i)$
    $\{t_a\} \leftarrow getAuthorTerms(p_i)$
    $\{t_v\} \leftarrow getVenueTerms(p_i)$
    normalize $t_p$, all $t_a \in \{t_a\}$, all $t_v \in \{t_v\}$
    $\{Hit\} \leftarrow searchIndex(I_c, t_p)$
    if only one $Hit$ found then
      add $[p_i, Hit]$ into $\mathcal{C}_D$
    else
      for each $t$ in $\{t_a\} + \{t_v\}$ do
        for each $Hit$ in $\{Hit\}$ do
          if $t$ not in $Hit.full_text$ then
            delete $Hit$ from $\{Hit\}$
          end if
        end for
        if size of $\{Hit\} = 0$ then
          roll back to the $\{Hit\}$ at last iteration
          break
        end if
      end for
      add $[p_i, \{Hit\}]$ into $\mathcal{C}_D$
    end if
  end for
  Return $\mathcal{C}_D$
```

Predicted Class → *Actual Class* ↓	in DBLP	not in DBLP
in DBLP	33 (TP)	2 (FN)
not in DBLP	1 (FP)	64 (TN)

TABLE 4: Confusion Matrix for the Merging Algorithm

$authorList, venue$ words has been searched, CiteSeerX papers has been reduced to a minimum number > 0, then keep these minimum amount of matches.

3.1.4 Evaluation of the Merging Algorithm

From the 2.1M CiteSeerX papers, we randomly select 100 papers and manually label them as positive class (in DBLP) or negative class (not in DBLP). We use the following four steps to manually label each CiteSeerX paper:

1. Check full text to manually obtain the real paper title.
2. Search the real paper title in DBLP title index.
3. Compare authors. Matched paper should have same title and authors.
4. If there are difficulties to judge the paper (e.g. no authors given), search and download the original PDF paper to compare with DBLP metadata.

Finally we labeled 35 out of 100 papers as positive class, and we applied the 100 sampled set to the incremental search algorithm. We use the confusion matrix to record the result. 34 papers are predicted as positive, only 1 of them are False Positive. And False Negative is 2. The *Precision* is 0.971, *Recall* is 0.943 and F_1 measure is 0.957. Table 5 shows the 3 False cases. For the FP case, the DBLP title is the prefix of the matched CiteSeerX paper title, and the author of the two papers are the same. For the two False Negative cases, both of the matches are reduced by incremental search due to lack of meta data in the matched CiteSeerX documents.

Wrong Case	CiteSeerX Paper	DBLP Paper
FP_1	Scalable Load-Distance Balancing in Large Networks	Scalable Load-Distance Balancing
FN_1	Proving and Disproving Termination of Higher Order Functions (Lack of authors, deleted by incremental search)	Proving and Disproving Termination of Higher Order Functions
FN_2	Bitvalue Inference Detecting and Exploiting Narrow Bitwidth Computations (Lack of venue, deleted by incremental search)	Bitvalue Inference Detecting and Exploiting Narrow Bitwidth Computations

TABLE 5: False Positive and False Negative cases

3.1.5 Results and Conclusion

505,352 (18.1%) DBLP records and 665,483 (31.4%) CiteSeerX documents are matched. Note that duplicates exists in CiteSeerX data set, in some cases there are multiple CiteSeerX documents matched one DBLP record. Among the matches, 331,453 DBLP records and CiteSeerX documents has unique one-to-one mapping, while the other 334,030 CiteSeerX documents matched the remaining 173,899 DBLP records (multiple mapping exists). Finally, we separate 665,483 CiteSeerX papers as positive class (CS papers), and all the remaining 1,452,639 papers are treated as negative class (non-CS papers).

3.2 arXiv

arXiv is also a freely accessed highly-automated academic papers library which was started in 1991 and maintained by Cornell University Library. It collected almost millions of articles in physics, mathematics, computer science, statistics, etc. Our arXiv collection contains around 840,218 papers. Different from CiteSeerX, the paper text contains paper title + abstract, which is much shorter of text in CiteSeerX data set. However, we observed that duplicates existed in arXiv data set. Our duplicate detection method is to compare the URL of a paper, if multiple papers have the same URL, we randomly keep one of them. After removing duplicates, 659,215 papers are retained. arXiv provides the label (e.g. CS, Math, Physics, etc.) for each paper.

	Before Remove Duplicates	**After Remove Duplicates**
CS	127,872	84,172
Math	297,094	215,143
Physics	88,896	65,564

TABLE 6: #Papers of CS, Math, Physics in arXiv Data Set

Table 6 shows the amount of papers in the three essential categories: CS, Math, Physics before and after removing duplicates.

arXiv can also be a very good scholarly data set for our experiments for the following three reasons:

1. Shorter and cleaner full text: Each paper text only have title + abstract, the text length is much shorter than full texts in CiteSeerX data set, using such text can perform quick test for our CS classification system. Moreover, the texts are parsed from Latex file, which can obtain a much better and cleaner text compared with CiteSeerX full text which is parsed from the web crawled PDF files.

2. Big training data size: The size of the data set is also big enough, more than 80,000 CS papers included.

3. Better data labeling: The class label of the papers are manually checked, and the research fields of the papers are strictly defined, for those non CS papers, they are in multiple fields, e.g. Physics, Mathematics, Biology, etc..

In this thesis, we conduct our classification experiments based on the balanced data set, and after remove duplicates, there are around 80,000 CS papers remained. Therefore, we make the data set balanced by randomly sampling 80,000 CS and 80,000 non-CS papers, to finally create a training set that includes 160,000 papers. Note that non-CS class represents all the other categories except CS.

CHAPTER 4

Researches on Citation Graphs

As a side research, this chapter mainly dig out the citation network patterns for DBLP and CiteSeerX. Citation graph is a directed graph that each edge represents a citation relation. If node A cited node B, then an edge $A \to B$ should be added into the citation graph. The in degree of a node represents how many citation received of this node, and the out degree represents how many papers this node cited (size of references of this node).

Based on CiteSeerX and DBLP, we define the following three different kinds of citation graphs:

Definition 1 *(Graph $\mathcal{A}$) Graph $G_A = (V_A, E_A)$, where V_A is a subset of papers in DBLP, and E_A is a set of citation links extracted from publishers ACM and IEEE.*

Definition 2 *(Graph $\mathcal{C}$) Graph $G_C = (V_C, E_C)$, where V_C is the set of papers in CiteSeerX, and E_C is the set of citation links in CiteSeerX.*

Definition 3 *(Graph $\mathcal{B}$) Graph $G_B = (V_B, E_B)$, where $V_B \subseteq D$ is s subset of papers in DBLP, and E_B is the set of citation links in CiteSeerX. i.e.,*

$$E_B = \{(a, b) | a, b \in D \land (a, b) \in E_C\} \tag{1}$$

4.1 Graph A: DBLP Citation Graph

For DBLP data set, the citations haven't been provided. However, based on our knowledge, ArnetMiner [31] used ACM and some other minor citation records to

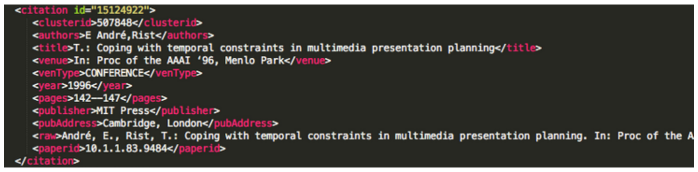
	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$
Number of Edges	4,191,677	1,244,002	4,277,924
Number of Nodes	781,108	352,926	970,586
Average Degree	5.4	3.5	4.4

TABLE 7: Statistics of citation networks $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. All the isolated nodes are excluded.

build the citation graph among DBLP papers. Until now, their newest data set collection contains 2,146,341 DBLP papers, and the corresponding citation graph contains 781,108 DBLP papers and 4,191,677 citation edges (isolated nodes are not included). The average citation degree is 5.366. We use $\mathcal{A} = \{V_A, E_A\}$ to represent the DBLP citation graph. The node set $\{V_A\}$ includes all the connected 781,108 papers, and $\{E_A\}$ is the set of connection among $\{V_A\}$. Table 7 includes the statistics of graph $\mathcal{A}$. The connected nodes are only 36.39% among all of their DBLP paper collections, the remaining nodes don't have any citation relations (either in degree: citation received or out degree: citation made) present in ArnetMiner citation graph. However, the ArnetMiner citation relations are only parsed and obtained from ACM and few other outside sources. Thus, the citation relations maybe incomplete among those DBLP papers.

4.2 Graph C: CiteSeerX Citation Graph

In CiteSeerX data set, citation string metadata is provided in each CiteSeerX paper which can be used to reveal the citation graph among papers in CiteSeerX data set. An example of a citation string metadata is shown in Fig. 3. ”raw” tag contains the original citation string parsed from full text. Other tags contains the metadata furthered parsed from the original citation string, such as cited paper title. Based on our observation, the further parsed metadata are not as accurate as the original citation string. The best way to find out the citation relations is to match paper metadata (e.g. title, authors) with the citation string, in order to relate a paper to its cited paper. For example, assuming that paper A has a citation string c; and there



```xml
<citation id="15124922">
    <clusterid>507848</clusterid>
    <authors>E André,Rist</authors>
    <title>T.: Coping with temporal constraints in multimedia presentation planning</title>
    <venue>In: Proc of the AAAI ‘96, Menlo Park</venue>
    <venType>CONFERENCE</venType>
    <year>1996</year>
    <pages>142--147</pages>
    <publisher>MIT Press</publisher>
    <pubAddress>Cambridge, London</pubAddress>
    <raw>André, E., Rist, T.: Coping with temporal constraints in multimedia presentation planning. In: Proc of the A
    <paperid>10.1.1.83.9484</paperid>
</citation>
```

FIGURE 3: Example of citation metadata in CiteSeerX

is a paper B, we search its title string t_B and all the authors $\{a_B\}$ in c, if both t_B and $\{a_B\}$ can be found in c, then we can say that citation string c refers to paper B, which means paper A cited paper B, an edge $A \rightarrow B$ is found. The algorithm is described in Alg. 2.

Due to the huge amount of papers in CiteSeerX data set, we need to use Lucene to index citation strings in order to increase the searching efficiency. There are two fields related to each indexed document:

- CiteSeerX Document ID (DOI)
- Original Citation String

Assuming that a paper has n citation strings, then this paper should have n indexed documents, each indexed document related to one unique citation string. Having the citation string index, the algorithm start by iterating each matched paper p_i in the merged data set, using DBLP title metadata to search the index, return all the matched indexed documents $\{Hit\}$, then use DBLP authors metadata to filter the matched results and obtain the final result set $\{Hit\}^*$. For each matched result Hit in $\{Hit\}^*$, add the edge $Hit \rightarrow p_i$ into citation graph. (Hit is the DOI related to the matched citation string)

4.2.1 CiteSeerX Title Correction

Another problem of constructing the citation graph is the inaccurate CiteSeerX metadata. Although citation string has relatively high accuracy, the accuracy of title metadata is very low (only 60%). Moreover, title metadata is crucial for matching citation strings; if title metadata is wrong, especially if the title metadata is very

Algorithm 2 Citation Graph for CiteSeerX Data Set

```
Require: citation index $I_c$, CiteSeerX $\mathcal{C}$, title train set $\{t\}$
  train classifier $\{t\} \to ClassifyTitle()$ for title correction
  for each paper $p_i$ in $\mathcal{C}$ do
    $t_p \leftarrow getTitle(p_i)$
    if $ClassifyTitle(t_p) \to false$ then
      jump to next paper
    end if
    $\{t_a\} \leftarrow getAuthor(p_i)$
    $\{Hit\} \leftarrow searchIndex(I_c, t_p)$
    for each $Hit$ in $\{Hit\}$ do
      if all $t_a \in \{t_a\}$ found in $Hit.citString$ then
        record the graph edge: $Hit \to p_i$
      end if
    end for
  end for
```

	Before	After
Number of Edges	10,595,956	4,277,924
Number of Nodes	1,286,659	970,586
Average Degree	8.2	4.4

TABLE 8: Difference after duplicate removal for graph $\mathcal{C}$.

short, it can cause huge amount of wrongly matched citation strings, which will cause this title (this paper) has huge amount of citation received and most of the citation relations can be wrong.

Thus, before the searching, we need to exclude wrong titles and also short titles. As shown in Alg. 2, the first step is to train a classifier for title correction. This is a typical two class classification task. Positive class is real titles, negative class is wrong titles. After train the classifier, for each CiteSeerX papers, we need to use it to first check if the title is predicted as positive class, then search the title in citation string index. The classification algorithm we used is Naive Bayes, and the title training set is provided by our research partner Dr. Yan Wang at the Central University of Finance and Economics in China.

4.2.2 Merging Duplicates

After executing Algorithm 2, we obtain a citation graph that contains 10,595,956 edges and 1,286,659 connected nodes as shown in Table 8.

However, in CiteSeerX data set, an in-neglectable problem is that there may exist multiple documents with different DOI (Document ID) that refer to the same paper, or different versions of the same paper. CiteSeerX data set is constructed by crawling documents from the web, thus it is possible that some papers, especially popular papers, can appears on multiple web locations. For example, paper: *"Graph-Based Algorithms for Boolean Function Manipulation"*, which is a famous Computer Science paper, has been collected into the CiteSeerX data set 5 times with 5 different document ID. This paper has very high in-degree (number of citations received), which is 4030 among the whole data set, and all the 5 duplicates have the same in-degree number (4030). In order to eliminate the effect of duplicate papers to the citation graph statistics, near duplicate detection need to be performed to remove redundant duplicates that only keep one copy of each unique paper. We use the following two step duplicate removal steps to remove redundant duplicate papers:

1. Full text detection: Apply SimHash to the full text of each paper in CiteSeerX data set to find out all duplicate document pairs. By creating the fingerprint for each document, we can estimate the similarity by calculating the hamming distance between each document pair. We also use Jaccard Similarity to further validate the similarity between each document pair. This work is provided by our research partner Yi Zhang.

2. Metadata detection: for papers that have same title and same in-degree count, only keep one paper, remove all the other redundant papers.

As a result of full text detection, we obtained 1,003,774 duplicate pairs among CiteSeerX papers. 221,548 papers are included in these duplicate pairs. For all of these duplicate pairs, we also find out the duplicate groups, for example: if paper A and paper B is a duplicate pair, and paper A and paper C is also a duplicate pair, we say paper $[A, B, C]$ belong to the same duplicate group. We only keep one paper in each duplicate group. In total there are 100,668 duplicate group found, and $221,548 - 100,668 = 120,880$ papers were removed from the citation graph in the first step.

In the second step, 180,087 duplicate groups were found and 221,388 duplicates were removed. Table 8 shows the statistics of CiteSeerX graph before and after removing duplicates. We use $\mathcal{C} = \{V_C, E_C\}$ to represent the CiteSeerX citation graph after removing duplicates. We can see that after removing duplicates, the half of the citation edges are deleted, and the average degree among CiteSeerX papers (nodes) also almost decrease by half. This is because lots of high in-degree papers can have duplicates, removing those duplicates significantly decrease the number of total edges.

4.3 Graph B: Graph for Merged Data Set

Graph $\mathcal{B}$ is the restriction of $\mathcal{C}$ on the nodes from DBLP. At the previous section, we have illustrated our merged data set that represents the intersections of CiteSeerX and DBLP papers, graph $\mathcal{B}$ is actually the citation graph of the merged data set, and the citation relations are extracted from CiteSeerX metadata. Graph $\mathcal{B}$ is also a sub-graph of Graph $\mathcal{C}$. Table 7 lists the statistics of graph $\mathcal{B}$. 1,244,002 edges (29.08%) out of all the 4,277,924 Graph $\mathcal{C}$ edges are kept in graph $\mathcal{B}$. There are 352,926 CiteSeerX papers connected in this sub-graph.

4.4 Comparison Between Three Graphs

4.4.1 Adding DBLP citations (Graph A vs. Graph B)

Nodes in graph $\mathcal{A}$ and graph $\mathcal{B}$ are all within the range of DBLP data set, however, edges in $\mathcal{B}$ are constructed using CiteSeerX citation string metadata. Based on our experiment, for all the 1,244,002 citation relations (edges) in citation graph $\mathcal{B}$, 325,775 edges can be found in ArnetMiner's DBLP citation graph, and $1,244,002 - 325,775 = 918,227$ edges can be added into the whole ArnetMiner citation network $\mathcal{A}$. Considering that originally ArnetMiner has already obtained around 4 million edges for the DBLP data set, now we can expand the DBLP citation network by adding an extra 918,227 edges. Besides, there are still some nodes in graph $\mathcal{B}$ not in $\mathcal{A}$, these nodes are the papers in DBLP but not included in Aminer data set, we use $V_B - V_A$

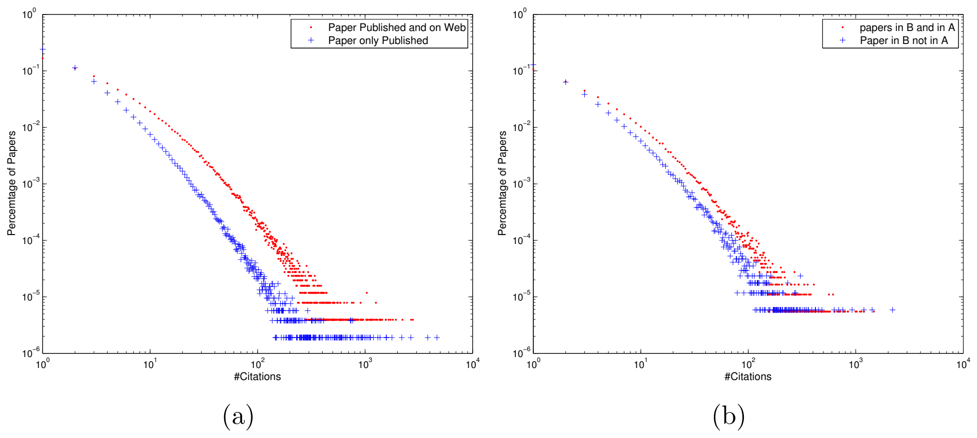


FIGURE 4: (a) Citation Distributions among nodes in $\mathcal{A}_B$ and $\mathcal{A}_{\overline{B}}$; (b) Citation Distributions among nodes in $\mathcal{B}_A$ and $\mathcal{B}_{\overline{A}}$

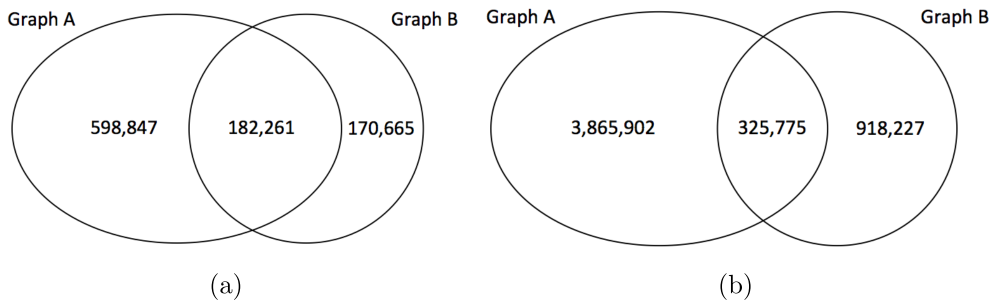


FIGURE 5: (a) Node intersection of $\mathcal{A}$ and $\mathcal{B}$; (b) Edge intersection of $\mathcal{A}$ and $\mathcal{B}$

to represent these nodes, and there are 170,665 such nodes. For the 918,227 new edges, there are 707,126 (77%) edges involved with these 170,665 nodes, the remaining 211,101 (23%) edges are new connections among existed Aminer nodes. Fig. 4 (b) shows the citation differences between the 182,661 nodes (red nodes in Fig. 4 (b)) in $V_A \wedge V_B$ and 170,665 nodes (blue nodes in Fig. 4 (b)) in $V_B - V_A$. We can see that nodes in $V_A \wedge V_B$ has higher average citations. But the largest citation value belongs to one of the nodes in $V_B - V_A$ (the rightmost blue node). Fig. 4 (a) shows the in-degree distributions for $V_A \wedge V_B$ nodes (red nodes) and $V_A - V_B$ nodes. We can see that intersected nodes in Graph $\mathcal{A}$ also have higher average in-degree in graph $\mathcal{A}$ compared with the nodes that not in CiteSeerX data set.

	$\mathcal{A}$	$\mathcal{B}$
Papers in $V_A \wedge V_B$	10.5	4.7
Papers in V_A	5.4	n/a
Papers in $V_A - V_B$	2.9	n/a
Papers in V_B	n/a	3.5
Papers in $V_B - V_A$	n/a	2.3

TABLE 9: Comparisons Between Graph $\mathcal{A}, \mathcal{B}$

4.4.2 Published, on web papers (Graph A vs. Graph C)

Thinking of the two data set in another way: DBLP data set is manually collected through recognized computer science publications, thus the papers included in DBLP can be treated as published papers. On the other hand, CiteSeerX data set is automatically crawled from the web, some of the crawled papers maybe not published, even not academic research papers, thus the papers included in CiteSeerX can be treated as on web papers. The difference between Graph $\mathcal{A}$ and Graph $\mathcal{C}$ can be visualized as the relationship between published papers and on web papers. The major comparison we made is to utilize the in-degree (number of citation received) distribution. We define the concept "published" and "on web" strictly as follows:

- $\mathcal{A}_{\overline{C}}$: (Only published) in DBLP data set but not in CiteSeerX data set.
- $\mathcal{A}_C, \mathcal{C}_A$: (Published & On web) the intersection of DBLP and CiteSeerX papers.
- $\mathcal{C}_{\overline{A}}$: (Only on web) in CiteSeerX data set but not in DBLP data set.

What we want to conclude is that we expect the papers that not only published but also on web can have the highest average in-degree, since we believe that famous published papers must be available on web and even have multiple distributions and versions on different web sources (which is also a major reason of duplicates in CiteSeerX data set). Since we got two citation graphs: $\mathcal{A}$ and $\mathcal{C}$, firstly we separate $\mathcal{A}$ into $\mathcal{A}_C$ and $\mathcal{A}_{\overline{C}}$ to compare the in-degree differences between published & on web papers and only published papers. From Fig. 6 (a) we can clearly see that published & on web papers have higher in-degree, and the average in-degree is 10.459, while

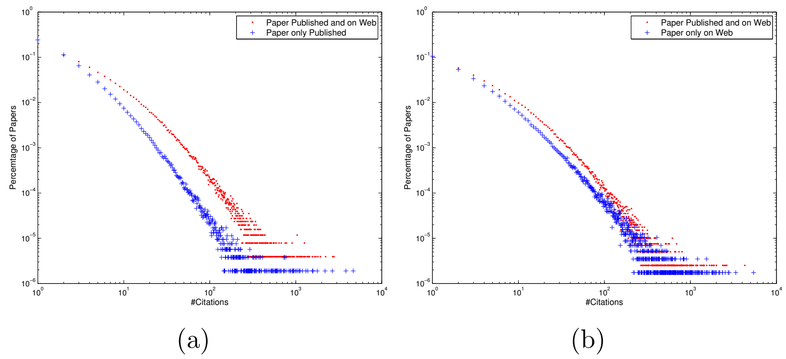


FIGURE 6: (a) In-Degree distribution of published & on web papers and only published papers using ArnetMiner Citation Graph; (b) In-Degree distribution of published & on web papers and only on web papers using CiteSeerX Citation Graph

	$\mathcal{A}$	$\mathcal{C}$
Papers in $V_A \wedge V_B$	10.5	5.8
Papers in V_A	5.4	n/a
Papers in $V_A - V_B$	2.9	n/a
Papers in V_B	n/a	4.4
Papers in $V_B - V_A$	n/a	3.4

TABLE 10: Comparisons Between Graph $\mathcal{A}, \mathcal{C}$

published papers only have 2.901 average in-degree. Secondly, we separate $\mathcal{C}$ into $\mathcal{C}_A$ and $\mathcal{C}_{\overline{A}}$ to compare published & on web papers and only on web papers. From Fig. 6 (b) we can see the difference is smaller than Graph $\mathcal{A}$, but still, published & on web papers have higher average in-degree which is 5.832; while on web papers have 3.426 average in-degree.

4.4.3 Duplicates and Citations

It can be concluded that duplicates are caused by multiple web locations of the same paper, it is not hard to imagine that if a paper is more popular, or has higher quality, it would be posted on the web more often, which means is has more duplicates. An important measurement of the quality of a paper is the citation count: how many citations it received since it was published. Therefore, the number of citations received should be proportion to the number of duplicates. Based on the hypothesis,

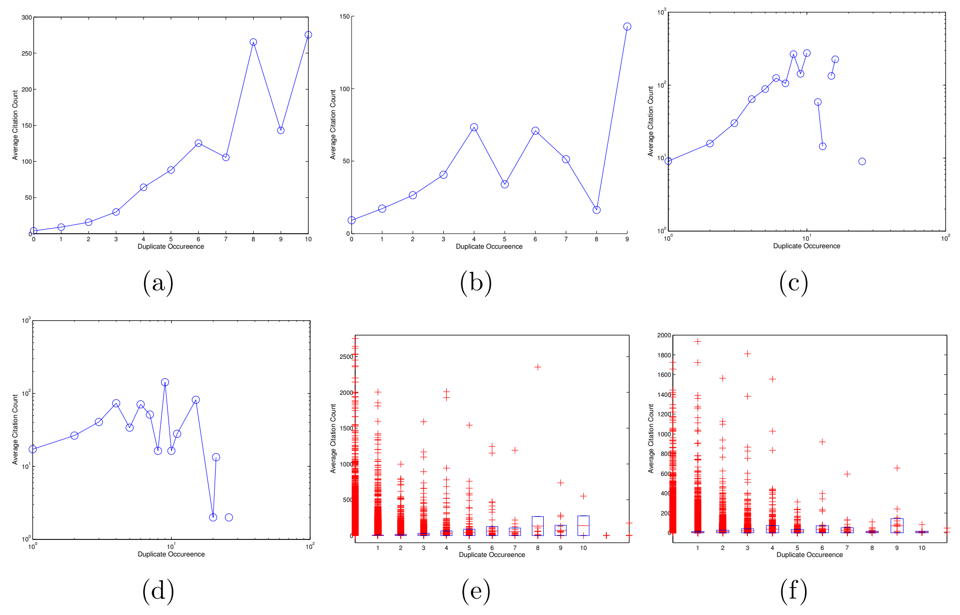


FIGURE 7: Citation count against duplicate occurrence. (a, b) average citation count against duplicate occurrence; (c, d) loglog plot of (a, b); (e, f) box plot; (a, c, e) CiteSeerX Graph; (b, d, f) AMiner Graph

we conducted an experiment that compare the relationship between the number of citations received and the number of duplicates. Fig. 7 shows citation counts as a function of duplicate occurrences. Note that the duplicates count were analyzed based on CiteSeerX full text, first we use the whole CiteSeerX citation graph $\mathcal{C}$ to directly reveal the relationship between citations and duplicates. Fig. 7 (a) (c) (e) shows the relations between CiteSeerX paper duplicate count and CiteSeerX paper citation received (in-degree in citation graph). We can see that the citation counts almost grows linearly with the increase of the duplicate occurrences. Note that we limit the duplicate occurrence within the range of 10 is because larger duplicate occurrences have very few amount of papers which may not reveal the real average citation count level. We can also see the outliers when the duplicate occurrences are large. The bottom box plot also shows the linearly increasing phenomenon.

We also utilize the Aminer citation graph to reveal whether the Aminer citation count can also in proportion to the duplicate occurrence. Although we don't have the duplicate statistics on Aminer (DBLP) data set, based on the merged data set of CiteSeerX and DBLP, we can obtain the duplicate statistics of the merged part of

DBLP data set by transforming from CiteSeerX duplicate statistics (each matched CiteSeerX paper is associated with Aminer paper). Fig. 7 (b) (d) (f) shows the citation counts against duplicate occurrences by using citations in Aminer citation graph. We can see that there is also a linearly increasing phenomenon especially when the duplicate occurrences are small.

CHAPTER 5

Classification System Methodology

This chapter illustrates our methodology for the implementation of classification system in detail.

5.1 Difficulties of Classifying Academic Papers

Most of the previous text classification related researches used Reuters data set, which is the most commonly used data set. Compared with such kinds of data sets, classifying academic papers have the following three difficulties:

- Length of a paper is usually much longer than common text-based documents (e.g. Reuters data set)
- Feature dimensionality could be extremely large (#words)
- Length, format of different papers varies greatly.

Feature is the basic element that input into the classifier to train the classifier. In text classification, the basic concept is that each distinct word is treated as an individual feature. However, when it comes to the text of academic papers, the amount of different words can be quite large. This will lead to very high feature dimensionality. The performance and efficiency of classifier is highly related to the following three aspects:

1. **The number, quality of features**: in order to train the classifier more efficiently, feature dimensionality need to be lowered down and informative features need to be included in the reduced feature set as many as possible.

2. **Classification Algorithm**: also a critical factor of deciding the performance, efficiency and accuracy of the whole classification system. Because of the nature of academic papers and large amount of training data set, more efficient, less complex classification algorithm is needed.

3. **Feature Weights**: Due to the different length, format of papers, more balanced weighting scheme instead of pure Term Frequency should be used to ensure more balanced level of classification precision and recall.

5.1.1 Lower Feature Dimensionality

There are lots of ways to lower the feature dimensionality. Feature selection is one of the most frequently used feature dimensionality lowering method. During the classification process, different features have different capabilities of making the classifier more or less easy to classify a document into one specific class. The goal of feature selection algorithms is to select highly capable and informative features out from all the other features.

Language modeling is also an effective way to not only lower the feature dimensionality, but also normalize text to increase the performance and accuracy of the classifier. Our main language modeling method used in this thesis is N-gram. N-gram represent a contiguous sequence of N items from a given sequence of text. Higher value of N can make features more related to context, since longer phrases are used as features. Other text normalization techniques we used includes case folding, regular expression normalization, stop words removal and stemming.

5.1.2 More Efficient Classifier

There are lots of classification algorithms existed. However, in the field of text classification, Naive Bayes, SVM and Logistic Regression are the most commonly used classification algorithm. Among the three algorithms, Naive Bayes is the most efficient and simplest one, with much faster running speed and much lower memory consumption. Moreover, previous researchers have already proved that although the independent assumption is clearly unreliable in most real-world tasks, Naive Bayes can still performs surprisingly well [21]. Especially in our case, applying much sophisticated algorithms (such as SVM) can be unfeasible due to scalability issue.

5.1.3 Deal with length variations

In most of the previous works, the authors didn't mention feature weighting schemes. In each document, each corresponding feature has a weight (feature value), in default the weight is assigned as Term Frequency (TF): the number of occurrences of the feature in a document. However, when it comes to academic papers, only simply using TF as feature weight is not enough. The most critical problem is the length of papers (especially papers from different journals, conferences) can be quite different. Under such circumstances, features in long papers will have higher TF, which may cause strong effect to the classifier to make the classifier more possible to classify a new document into long paper's class. Thus, instead of using term frequency directly, we need to use more advanced and balanced feature weighting scheme in order to solve the length variations problem.

5.2 The proposed solution

Tackling the above three difficulties, our classification system combine various feature selection algorithms, language models and feature weighting schemes to produce final feature set as a input for the classification algorithms. In order to ensure the efficiency while the size of training set is extremely large, we implement two models of Naive Bayes algorithm as our major classifier. However, as a comparison, we also used the Logistic Regression classifier implemented by Sci-kit learn[25]. As shown in Fig. 8, process full text model correspond to transfer and normalize original text, build feature set corresponds to language modeling and feature weight processing. We implemented two different representations of feature set, one is N-gram models, the other is sentence2vec model. Feature set is the input of the classification system. Inside our classification system, there are two core components: feature selector and

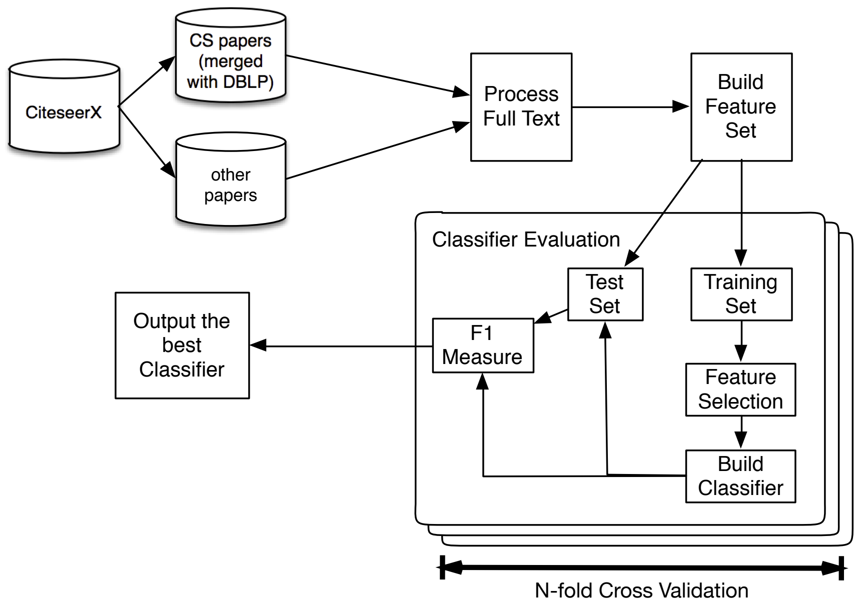


FIGURE 8: Classification System

classifier. We implemented three feature selection algorithms (Mutual Information (MI), χ^2 and Pointwise Mutual Information (PMI)). We separate the feature set into training set and test set in order to apply cross validation scheme to evaluate different classifiers to finally output the best one. In the following sections, we will introduce our classification system in detail.

5.3 Language Modeling

5.3.1 N-gram

1. **Regular expression tokenization**: in order to create a purer text environment for the classifier, we use Regular expression tokenizer provided by NLTK[2] to tokenize word into tokens and each token contains only alphanumeric letters.

2. **Case folding**: transform all upper case English letters into the corresponding lower case letters, in order to make the classifier case insensitive.

3. **Remove stop words**: stop words are the extremely common words which are

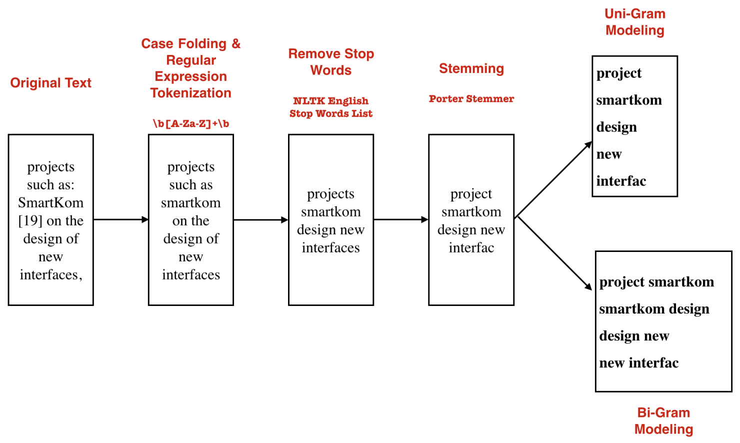

Original Text
projects such as: SmartKom [19] on the design of new interfaces,
Case Folding & Regular Expression Tokenization
\b[A-Za-Z]+\b
projects such as smartkom on the design of new interfaces
Remove Stop Words
NLTK English Stop Words List
projects smartkom design new interfaces
Stemming
Porter Stemmer
project smartkom design new interfac
Uni-Gram Modeling
project
smartkom
design
new
interfac
project smartkom
smartkom design
design new
new interfac
Bi-Gram Modeling

FIGURE 9: Example of Language Modeling

meaningless to the classification decision. Most typical stop words in English can be ”a”, ”an”, ”the”, ”of”, etc. In this thesis, we use NLTK [2] English Stop Words List to filter stop words out of the text.

4. **Stemming**: it is used to integrate different forms of English words. Stemming can be defined as a heuristic process that chops off the ends of words to eliminate the effect of derivation of words. For example: automate, automatic and automation can be all integrate into the same word. In this thesis, we use Porter algorithm [26] as our English language stemmer.

5. **N-gram**: It is used to create a set of features as an input for training the classifier. We use two different N-gram methods: Uni-gram and Bi-gram. Uni-gram means only use 1 word as feature, every distinct word is a unique feature, while Bi-gram means use 2 continuous words as a feature, every distinct 2-word phrase is a unique feature.

Fig. 9 shows a simple example of how we implement the language modeling to transform original text into feature set. First we do tokenization and case folding, assuming we only keep English letters, meaningless characters such as ”[19]” will be removed. The second step is stop words removal, extremely common words ”such”, ”as”, ”on”, ”the” and ”of” are removed. The third step is stemming, the suffix of ”interfaces” is removed. Then the final step is using Uni-gram and Bi-gram to transform normalized original text into feature set. Uni-gram set has 5 features, and Bi-gram set has 4 features.

Algorithm 3 Uni-gram, Bi-gram Language Modeling

```
Require: $Tokenizer$, $StopwordList$, Input String $Str$
  $UniGramSet = BiGramSet = \emptyset$
  $termList = tokenizer(str)$
  for $i = 0$ to $size(termList) - 1$ do
    if $termList[i]$ not in $StopwordList$ then
      $termList[i] = lowercase(termList[i])$
      $termList[i] = potterStemmer(termList[i])$
      $UniGramSet \leftarrow termList[i]$
    end if
    $i = i + 1$
  end for
  $BiGramSet$ = create Bi-gram($UniGramSet$)
  return $UniGramSet$, $BiGramSet$
```

5.3.2 Sentence2vec

Sentence2vec [16] is a deep learning approach to learn the embedding of sentence from the training data set. One typical sentence2vec model is Distributed Memory Model of Paragraph Vectors (PV-DM), which trains a sentence vector along with the word vectors to predict the missing content. In this model, paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph. Sentence2vec have several parameters that can be optimized, the most important parameters are window-size, negative sample size, and the dimension of the vectors.

The dimension of vectors decides the feature dimensionality. Compared with N-gram models that treat each different grams (Uni-gram or Bi-gram) as a unique feature, sentence2vec modeling can significantly lower down the feature dimensionality, which make it applicable to use other classification algorithms such as Logistic Regression.

5.4 Feature Weights

Feature weight is also a critical aspect to affect the accuracy of the classifier. In this thesis, we use the following three different weighting schemes, including the default TF as feature weight, and two advanced normalized feature weights in order to deal with the length variations and text complexity in academic papers.

5.4.1 Term Frequency

The simplest weighting scheme can be Term Frequency (TF), which is defined as the number of occurrences of a feature in one document. Thus, each document is represented as a set of $\{t_i : f_i\}$. t_i is the i-th feature in the document, and f_i is the term frequency of t_i.

5.4.2 Length Normalization

Based on the feature weight normalization process proposed by [27], the classifier can more realistically handle text while not giving up the advantages of Multinomial Naive Bayes. We produce the following equation for the feature weight length normalization:

$$f_i' = \frac{f_i}{\sqrt{\sum_k (f_k)^2}} \tag{1}$$

By using length normalization, we don't change the length of a paper by selecting part of the paper to make every paper has equal length, instead, we want to eliminate the effect of length to the feature weights. For example, f_i appears n times in paper A which length is t. There is another paper B which length is $2t$, and f_i appears $2n$ times in B. In order to eliminate the effect of length, we cannot just directly weight F_{iA} as n and f_{iB} as $2n$, we need to assign the same weights to both of them, since B is 2 times longer than A. [27] showed that longer documents have larger probabilities for larger term frequency values, even worse, when the term frequency value increase, the term frequency probability gap between long documents and short documents keep enlarging.

5.4.3 Full Normalization

Except length problem, there are also other feature weighting question such as should rare words be assigned with higher weight? One step further, we can combine TF-iDF weighting scheme together with the length normalization, which we called as Full Normalization. The equation is shown as follows:

$$\begin{aligned} f_i' &= \log(1+f_i) \cdot \log \frac{N}{\sum_d \delta_{id}} \\ f_i'' &= \frac{f_i'}{\sqrt{\sum_k (f_k')^2}} \end{aligned} \tag{2}$$

The first equation that calculates f_i' is the TF-iDF normalization. Note that the TF here uses logarithm frequency, and we add one before using logarithm to prevent 0 term frequency; also add 1 has the advantages of being an identity transform for zero and one term frequency values. iDF (inversed document frequency) is to give more weight to rare occurred words and diminish the strong effect of common words. The second equation that calculates f_i'' is length normalization, which is calculated after obtain TF-iDF weight f_i'.

5.5 Feature Selection Issues

Feature selection is the process of selecting a subset of relevant features for use in the classification model construction, and based on the assumption that data set contains lots of redundant or irrelevant features. It is very important as it can lower the feature dimensionality and exclude useless features that can improve both the classification efficiency and accuracy.Feature selection process includes evaluation and ranking. There are two models for feature evaluation: filter model and wrapper model. Filter model uses statistical characteristics of the features for evaluation, which is independent from the classifier, and wrapper model calculates the score of a subset of features by inducing a classifier. High time complexity is the major drawback of wrapper model. Due to the high feature dimensionality and text complexity in our case, we only use filter model. We use three filter model algorithms: Mutual

Information (MI), χ^2, Pointwise Mutual Information (PMI). Moreover, we use simple ranker method for feature ranking, which ranks the features based on their scores in an descending order.

5.5.1 Mutual Information

Mutual Information measures the information between classes and features. The value of a feature's mutual information $M_i(w)$ is decided by the level of co-occurrence of class i and the feature w. In other words, if a feature w is highly related to only one specific class, it will has higher mutual information score since having this feature the classifier can more easily make the correct decision. However, if a feature w is independent with any classes, for example, it occurs equal times in both classes' documents, then this feature has no value to the classifier, it should be scored as the lowest ranked feature.

Generally speaking, Mutual Information measures the difference between expected and observed frequency. The difference measurement is conducted by using the following equation:

$$I(U, C) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{N_{e_t e_c}}{N} \log \frac{N_{e_t e_c}}{E_{e_t e_c}} \tag{3}$$

We use U to represent the event of a feature occurring or not, and use C to represent the event of a class occurring or not. Since in our case, we only do two class classification (CS and non-CS), U and C are binary values, 1 means occurring, 0 means not occurring. N has different meanings when using Bernoulli model (number of training documents) and Multinomial model (the total of lengths of all training documents). $N_{e_t e_c}$ is the observed frequency, equals to $N \cdot P(U = e_t, C = e_c)$,which means the actual probability (frequency) of these two events happen together; and $E_{e_t e_c}$ is the expected frequency, equals to $N \cdot P(U = e_t) \cdot P(C = e_c)$, which means the probability (frequency) of these two events happen together given that these two events are independent with each other. As can be seen in the equation, Mutual Information uses division calculation to measure the difference between observed frequency

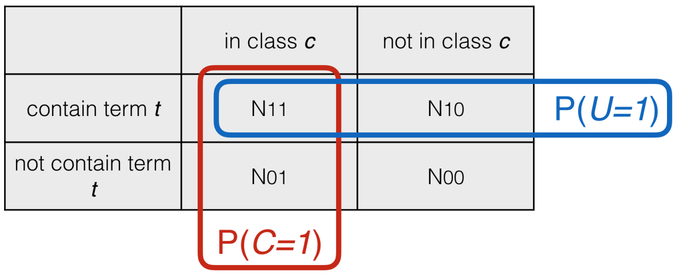


FIGURE 10: Definition of Observed Frequency

and expected frequency. If a feature is really totally independent with a class, which means the observed frequency $N_{e_t e_c}$ equals to the expected frequency $E_{e_t e_c}$, then the mutual information score for this feature $I(U, C)$ will equal to 0, since $\frac{N_{e_t e_c}}{E_{e_t e_c}} = 1$ lead to $\log_2 1 = 0$.

Using the probability of U and C, Mutual Information can be represented as the following equation:

$$I(U, C) = \sum_{e_t \in 0,1} \sum_{e_c \in 0,1} P(U = e_t, C = e_c) \log \frac{P(U = e_t, C = e_c)}{P(U = e_t) P(C = e_c)} \tag{4}$$

In order to obtain the Mutual Information score for each feature (distinct word), we need to calculate the observed frequency and expected frequency for each feature. Thus, we define the observed frequency matrix as shown in Fig. 10:

Since we have two kinds of events: occurring of class and occurring of features. There are $2 \times 2 = 4$ different combinations of cases. We use $N_{e_t e_c}$ to represent these four cases. For example, N_{11} represents both occur of class and feature, which can also be represented as $P(U = 1, C = 1)$. As can be seen that N_{11} is the observed frequency of class and feature occurring together. Accordingly, N_{10} represent the observed frequency of feature occurring without class occurring. Based on $N_{11}, N_{10}, N_{01}, N_{00}$, we can calculate the expected frequency of these four different cases. Taking N_{11} as an example, we have:

- N_{11}: the observed frequency of t and c occurring together.
  $N_{11} = N \cdot P(U = 1, C = 1) = N \cdot \frac{N_{11}}{N} = N_{11}$

	Paper ID	Paper Text	Class Label
Training Set	1	Data Data Computer	CS
	2	Complexity Data Data	CS
	3	Software Data	CS
	4	Network Wireless Software Protocol	CS
	5	Data Climate Business	non-CS
	6	Economic Popular State Business	non-CS

Bernoulli Model			Bernoulli Model		
	CS	non-CS		CS	non-CS
contain *Data*	3	1	contain *Data*	5	1
not contain *Data*	1	1	not contain *Data*	7	6

TABLE 11: Example of Calculating Observed Frequency

- E_{11}: the expected frequency of t and c occurring together.
  $E_{11} = N \cdot P(U=1) \cdot P(C=1) = N \cdot \frac{N_{11}+N_{10}}{N} \cdot \frac{N_{11}+N_{01}}{N} = \frac{N_{1.}N_{.1}}{N}$

E_{11} is the expected frequency of class and feature occurring together. As we described before, expected frequency is calculated under the assumption that two events are independent. Thus, it should be equal to $P(U=1) \cdot P(C=1)$. As can be in Fig. 10, $P(U=1)$ equals to $N_{11}+N_{10}$, which means the frequency of feature occurs, regardless of whether class occur or not. Accordingly, $P(C=1)$ equals to $N_{01}+N_{00}$.

The observed frequency $N_{e_t e_c}$ has different definitions under Multinomial Model and Bernoulli Model. In multinomial Model, $N_{e_t e_c}$ value is the feature's document frequency, for example, N_{1c} means how many class c documents contain feature t, and N_{0c} means how many class c documents not contain t. In Bernoulli Model, $N_{e_t e_c}$ value is the feature's class document frequency, for example, N_{11} means feature t occur how many times in all class c documents, and N_{01} means the total of all Term Frequencies expect feature t in all class c documents.

Table 11 shows an example of how to calculate observed frequency. In the example, there are 6 documents in the training set, 4 documents belong to CS class, and 2 documents belong to non-CS class. We take the feature *Data* for example, as can be seen in Table 11, *Data* occurs in 3 CS documents and 1 non-CS document, while 1 remaining CS document and 1 non-CS document don't contain *Data*. On the other hand, if using Multinomial model, N_{11} is equal to 5 since *Data* occurs 5 times in all the 4 CS documents. And the total length of the 4 CS documents is 12 (word count: $3 + 3 + 2 + 4 = 12$). Thus, $N_{01} = 12 - 5 = 7$. Accordingly, $N_{10} = 1$ (*Data* occurs only 1 time in non-CS documents), and $N_{00} = 7 - 1 = 6$.

Based on the four $N_{e_t e_c}$ values, we can calculate the Mutual Information score for each feature. Transformed from equation 12, the final Mutual Information calculation is based on the following equation:

$$I(U,C) = \frac{N_{11}}{N}\log\frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N}\log\frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N}\log\frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N}\log\frac{NN_{00}}{N_{0.}N_{.0}} \quad (5)$$

5.5.2 χ^2-statistic

Similar with Mutual Information, χ^2 also computes the level of independence between feature t and a class c. High χ^2 value indicates the high dependency of feature and class, also indicates high difference between observed and expected frequency.

However, the difference between Mutual Information and χ^2 is the difference measure between observed and expected frequency. Compared with Mutual Information which uses division to measure the difference, χ^2 uses subtraction. The measure equation can be represented as follows:

$$X^2(U,C) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \quad (6)$$

$N_{e_t e_c}$ is the observed frequency and $E_{e_t e_c}$ is the expected frequency just as the same as definitions in Mutual Information. Still, we can see that if $N_{e_t e_c} = E_{e_t e_c}$, the χ^2 score will become 0. The χ^2 calculation process is the same with Mutual Information, first we need to obtain the four $N_{e_t e_c}$ values, and then calculate the final score $X^2(U,C)$. The final calculation equation can also be transformed into calculating $N_{e_t e_c}$ values, which is shown as follows:

$$X^2(U,C) = \frac{(N_{11}+N_{10}+N_{01}+N_{00}) \times (N_{11}N_{00} - N_{10}N_{01})^2}{(N_{11}+N_{01}) \times (N_{11}+N_{10}) \times (N_{10}+N_{00}) \times (N_{01}+N_{00})} \tag{7}$$

5.5.3 Pointwise Mutual Information

$$PMI(U,C) = \sum_{e_c \in \{0,1\}} \left| \log \frac{N_{1e_c}}{E_{1e_c}} \right| \tag{8}$$

As can be seen in the above equation, PMI also use division to measure the differences between observed and expected frequencies. However, PMI don't consider the probability of event $e_t = 0$, only the probability of occurring feature t is considered. This character makes PMI tend to score more exclusive (in only one specific class) but unpopular and strange words to the top. Using the $N_{e_t e_c}$ representation, the PMI calculation equation can be rewritten as follows:

$$PMI(U,C) = \left| \log \frac{NN_{11}}{N_{1.}N_{.1}} \right| + \left| \log \frac{NN_{10}}{N_{1.}N_{.0}} \right| \tag{9}$$

N_{1c} is the occurrences of feature t in class c. We can see that different from MI, N_{0c} is not considered. $N_{.c}$ is the total length of all documents under class c, and $N_{1.}$ is the occurrences of feature t in all classes. Therefore, a closer of $N_{1.}$ and N_{1c} values infer a more exclusive feature. If $N_{1.} = N_{1c}$, then it represents the feature t totally only occur in one class. The main reason that PMI can highly score very exclusive feature is the absolute value it used. A feature which is more exclusive in c will make smaller value of $N_{1e_{\overline{c}}}$, that make $\frac{N_{1e_{\overline{c}}}}{E_{1e_{\overline{c}}}} \to 0$, and in consequence make $\log \frac{N_{1e_{\overline{c}}}}{E_{1e_{\overline{c}}}} \to -\infty$. After using absolute value, this feature will be assigned with a very high score.

5.6 Classification Algorithms

5.6.1 Naive Bayes Classifier

Naive Bayes Classifier has been widely utilized for many years. It belongs to the category of probabilistic classifier. It is based on the simple assumption that the each feature is independent given the specific class. Moreover, it is bag of words model that assume the order or position of a word is irrelevant. Naive Bayes Classifier computes the posterior probability of a class, based on the distribution of the words in the document, either uses boolean model to represent if a word appear in a document, or calculates the time of occurrence of a word (Term Frequency) in a document. Although such independent assumption is unrealistic, the classification result can still be optimal, especially under zero-one loss, and the attribute dependence is not necessarily to be considered to improve classification results [9]. In other words, Naive Bayes classifier can perform surprisingly well even though the independent assumption is presented.

There are two models of Naive Bayes related to text classification: Multinomial Model and Bernoulli Model. In the following two sections we will introduce these two different models in detail.

5.6.2 Multinomial Naive Bayes

The major feature of Multinomial Naive Bayes is that it captures the frequencies of words in a document [20]. Firstly, we start from the Bayes rule. The equation below shows the classical Bayes rule in the field of probability and mathematical statistics:

$$P(c|d) \propto \frac{\widehat{P}(c)\widehat{P}(d|c)}{\widehat{P}(d)} \rightarrow \widehat{P}(c)\widehat{P}(d|c) \tag{10}$$

We use c to represent a class, and use d to represent a document. Therefore, $P(c|d)$ represents the probability of a document d being in a class c. Actually calculating $P(c|d)$ is the goal of our classification task. We have two classes, use c_1 (CS) and c_2 (non-CS) to represent them, now for each new test document d_i, we need to obtain $P(c_1|d_i)$ and $P(c_2|d_i)$, compare which value is larger. Assume $P(c_1|d_i) > P(c_2|d_i)$, then we can conclude that d_i should belong to c_1. Note that for each probability, we add a superscript, that's because all of these parameters are estimated.

Therefore, the problem is how to obtain $P(c|d)$. We use Bayes rule to transform the conditional probability, transfer $P(c|d)$ into calculating $P(d|c)$, which is easier to obtain. Note that in the Bayes rule equation, we neglect the probability $P(d)$, since $P(d)$ is a constant that we don't need to care about, what we really need to calculate is $P(c)$ and $P(d|c)$. The meaning of $P(d|c)$ is given class c has already happened, what is the probability of document d occurs. In order to calculate $P(d|c)$, base on the bag of words model, we can split document d into a set of its words. The equation is shown below:

$$P(c|d) \propto \widehat{P}(c)\widehat{P}(d|c) \rightarrow \widehat{P}(c)\widehat{P}(t_1 \cdot t_2 ...... t_{n_d}|c) \tag{11}$$

The probability of document d occurs can be equalized to the probability of all the words in document d occur together. Assume document d has n_d words (document length is n_d), then we separate each word, don't need to consider the order or position of different words, finally the probability $P(d|c)$ can be equalized to $P(t_1 \cdot t_2 ...... t_{n_d}|c)$. This is called the bag of words model probability transformation.

The final step of the probability calculation is using the independent assumption. In the field of probability and mathematical statistics, if two incidents A and B are independent, we can say that $P(AB) = P(A) \cdot P(B)$. Based on the independent assumption in Naive Bayes, we can separate the conditional probability for each words in a document. The equation is shown as follows:

$$P(c|d) \propto \widehat{P}(c)\widehat{P}(t_1 \cdot t_2 ...... t_{n_d}|c) \rightarrow \widehat{P}(c) \prod_{1 \leqslant k \leqslant n_d} \widehat{P}(t_k|c) \tag{12}$$

As we reach this final step, we transformed our goal $P(c|d)$ into estimating $P(c)$ and $P(t_k|c)$. $P(c)$ can be called as the prior probability of class c, it is calculated using the following equation:

$$P(c) = \frac{N_c}{N} \tag{13}$$

N_c is the number of class c documents in the training data set, and N is the number of all documents in the training data set. On the other hand, $P(t_k|c)$ is the conditional probability of word t_k occurring in all class c training documents. $P(t_k|c)$

is equalized as the proportion of t_k occurring in all class c training documents.

$$\widehat{P}(t_k|c) = \frac{T_{ct_k} + \min(1,\tau)}{\sum_{\acute{t}\in V}(T_{c\acute{t}} + \min(1,\tau))} = \frac{T_{ct_k} + \min(1,\tau)}{\sum_{\acute{t}\in V} T_{c\acute{t}} + B \cdot \min(1,\tau)} \qquad (14)$$
$$\tau = min\{T_{c\acute{t}}|\acute{t} \in V\}$$

We use T_{ct_k} to represent the class term frequency of word t_k. However, under feature weight normalized environment, T_{ct_k} maybe normalized by length normalization and TF-iDF. V is the size of vocabulary (number of features) in the whole training set. If a word t appears in any of class c documents, add its class term frequency / or normalized weight; otherwise add 0 if the word doesn't appear in any of class c documents. However, there is a special case that we need to take into consideration. If $T_{ct_k} = 0$, we need to prevent $P(t_k|c) = 0$, which will cause the whole probability $P(c|d)$ becomes zero. Considering this case, a new test document d_{new} has word t_k, now we need to calculate the probability $P(c|d_{new})$, however, among all class c training documents, t_k haven't occurred; t_k only occurred in the documents of another class; thus $T_{ct_k} = 0$. In order to prevent such cases, the typical method is add-one smoothing. Add one to the class term frequency of any words. As for the denominator, we also need to add some values to ensure the balance. We follow the typical smoothing method that add the number of total different features (the size of feature set), which is represented by B. However, if feature weights are normalized, add-one smoothing maybe no longer appropriate, since the feature weight are not represented by CTF (class term frequency), instead, they are normalized to a much smaller value (most of them smaller than 1). Therefore, in the normalized environments, we need to replace 1 with the smallest feature weight value τ (note that $\tau > 0$), as shown in equation 14.

There is one more special case that a word in test document never occur in any of the training documents, then this word will be discarded since we don't have this word's estimated conditional probability.

As conclusion, Multinomial Naive Bayes has two steps:

1. Training: estimate parameters for all classes c_i, $\widehat{P}(c_i)$ and $\widehat{P}(t_k|c_i)$ for all features

appeared in training data set.

2. Test: for a new document d, calculate $P(c_i|d)$ based on the estimated parameters.

Algorithm 4 and algorithm 5 further illustrate the process of train and test Multinomial Naive Bayes in detail.

Algorithm 4 Train Multinomial NB

```
Require: Doc set $\mathbb{D}$, classList $\mathbb{C}$
  $V \leftarrow getVocabulary(\mathbb{D})$
  $N \leftarrow countDocs(\mathbb{D})$
  for each class $c \in \mathbb{C}$ do
    $N_c \leftarrow countDocsInClass(\mathbb{D}, c)$
    $prior[c] \leftarrow \frac{N_c}{N}$
    $T_c \leftarrow WordsInClass(\mathbb{D}, c)$
    for each each feature $t \in V$ do
      $T_{ct} \leftarrow countTFinClass(\mathbb{D}, c, t)$
      $condProb[t][c] \leftarrow \frac{(T_{ct} + \min(1, \tau))}{(T_c + len(V) \cdot \min(1, \tau))}$
    end for
  end for
  Return $prior, condProb$
```

Algorithm 5 Test Multinomial NB

```
Require: Test doc $d$, ClassList $\mathbb{C}$, $prior$, $condProb$, $V$
  $W_d \leftarrow getTokens(d)$
  for each class $c \in \mathbb{C}$ do
    $score[c] \leftarrow \log prior[c]$
    for each $t$ in $W_d$ do
      if $t \in V$ then
        $score[c] += \log condProb[t][c]$
      end if
    end for
  end for
  Return $\arg\max_{c \in \mathbb{C}} score[c]$
```

We use this simple example in Table 12 to illustrate how Multinomial Naive Bayes works. Assume we have 4 papers in our training set, 3 papers belong to CS class, and 1 paper belongs to non-CS class. Each of the paper full texts are shown in the

	Paper ID	Paper Text	Class Label
Training Set	1	Data Data Computer	CS
	2	Complexity Data Data	CS
	3	Software Data	CS
	4	Data Climate Business	non-CS
Test Set	5	Climate Business Data Data Data	?

TABLE 12: Example of using Naive Bayes

table. Now we need to predict the class for a new test paper (d_5): "Climate Business Data Data Data". First we estimate the probability of d_5 under class CS. Based on the Multinomial Naive Bayes rules, we can calculate $P(CS|d_5)$ using the following equation:

$$P(CS|d_5) \propto \widehat{P}(CS) \cdot \widehat{P}(Climate|CS) \cdot \widehat{P}(Business|CS) \cdot \widehat{P}(Data|CS) \cdot \widehat{P}(Data|CS) \cdot \widehat{P}(Data|CS)$$

First calculate the prior probability $\widehat{P}(CS) = \frac{3}{4}$ as the fraction of CS papers in the training data set. Then, for each word in d_5, estimate the word conditional probability under CS class. Firstly, for the first word $Climate$, it appears 0 times in all the 3 CS class training papers, so we estimate $\widehat{P}(Climate|CS)$ as $\frac{0+1}{8+6} = \frac{1}{14}$. The total of the 3 CS papers' length is 8 (3+3+2), and among all 4 training papers there are 6 different words (6 features); thus, the denominator is 8+6. And the estimation process is the same for the other four words: $Business, Data, Data, Data$, $\widehat{P}(Business|CS) = \frac{0+1}{8+6} = \frac{1}{14}$ $\widehat{P}(Data|CS) = \frac{5+1}{8+6} = \frac{3}{7}$. Finally the probability of d_5 under class CS is estimated as $\frac{3}{4} \cdot \frac{1}{14} \cdot \frac{1}{14} \cdot (\frac{3}{7})^3 \approx 0.0003$.

Accordingly, the probability of d_5 under class $nonCS$ is estimated as $\frac{1}{4} \cdot \frac{2}{9} \cdot \frac{2}{9} \cdot (\frac{2}{9})^3 \approx 0.0001$. As a result, d_5 has been classified as a CS paper.

5.6.3 Bernoulli Naive Bayes

Different from Multinomial model, Bernoulli Naive Bayes only uses the presence or absence of words in a text document as features to represent a document [20]. Considering a feature's class term frequency T_{ct_k}, if a class c document d_i in training set has this feature, and it occurs t_i times in d_i, for Multinomial model, T_{ct_k} will be updated as $T_{ct_k} \leftarrow T_{ct_k} + t_i$.

However, in Bernoulli model, we don't care about how many times a feature occurs in a document, what only matters is whether this feature occur or not, so in this case, the feature t_k occurs in document d_i, so we add binary value 1 to update the T_{ct_k}, otherwise add binary value 0 to update the T_{ct_k}. Therefore, T_{ct_k} is not longer represent the feature's class term frequency, instead, it means the class document frequency. More generally speaking, Bernoulli model estimate the feature's conditional probability as the fraction of papers that contain this feature. The $P(c|d)$ estimation equation is shown as follows:

$$P(c|d) \propto \widehat{P}(c) \prod_{k=1}^{V} [b_k \widehat{P}(t_k|c) + (1 - b_k)(1 - \widehat{P}(t_k|c))] \tag{15}$$

Since Bernoulli Model uses presence or absence or words to estimate the probability, the estimation is different from Multinomial Model. Instead of multiplying by the conditional probability of each word in the test document, in Bernoulli Model, we need to multiply each feature's probability (all features that within the training data set): if a feature t_k appear in the test document, multiply by $\widehat{P}(t_k|c)$; if a feature t_k not appear in the test document, multiply by $[1 - \widehat{P}(t_k|c)]$, since $[1 - \widehat{P}(t_k|c)]$ represent the probability of this document belong to class c when t_k not appear in it. Thus, in the above equation, b_k is a binary value, if word t_k occur in test document d, $b_k = 1$, we multiply $\widehat{P}(t_k|c)$; otherwise $b_k = 0$, we multiply $[1 - \widehat{P}(t_k|c)]$. Note that when calculating $\widehat{P}(t_k|c)$, each distinct t_k only calculate once, regardless of their occurrences in a document.

Another major difference compared with Multinomial Model is the estimation of feature conditional probability $\widehat{P}(t_k|c)$. As can be seen in algorithm 6, the probability is estimated by the fraction of class document frequency, instead of class term frequency. Moreover, add one smoothing only need to add 2 into the denominator, since there are only two cases to consider: occurrence of t_k or non-occurrence of t_k.

We use the same example in Table 12 to illustrate the training and testing process of Bernoulli Naive Bayes. First calculate the prior probability as the same with Multinomial Model: $\widehat{P}(CS) = \frac{3}{4}$. Then, for each word in d_5, estimate the word conditional probability under CS class. Firstly, for the first word *Climate*, it appears in 0 documents among the 3 CS documents, so we estimate $\widehat{P}(Climate|CS)$ as $\frac{0+1}{3+2} = \frac{1}{5}$. $\widehat{P}(Business|CS)$ also equals to $\frac{1}{5}$. For the word *Data*, it appears in all the 3 CS training documents, thus estimate $\widehat{P}(Data|CS)$ as $\frac{3+1}{3+2} = \frac{4}{5}$. Accordingly, $\widehat{P}(Computer|CS) = \widehat{P}(Complexity|CS) =$

Algorithm 6 Train Bernoulli NB

```
Require: Doc set $\mathbb{D}$, classList $\mathbb{C}$
  $V \leftarrow getVocabulary(\mathbb{D})$
  $N \leftarrow countDocs(\mathbb{D})$
  for each class $c \in \mathbb{C}$ do
    $N_c \leftarrow countDocsInClass(\mathbb{D}, c)$
    $prior[c] \leftarrow \frac{N_c}{N}$
    for each each feature $t \in V$ do
      $N_{ct} \leftarrow countDFinClass(\mathbb{D}, c, t)$
      $condProb[t][c] \leftarrow \frac{(N_{ct} + 1)}{(N_c + 2)}$
    end for
  end for
  Return $prior, condProb$
```

Algorithm 7 Test Bernoulli NB

```
Require: Test doc $d$, ClassList $\mathbb{C}$, $prior$, $condProb$, $V$
  $V_d \leftarrow getVocabulary(d)$
  for each class $c \in \mathbb{C}$ do
    $score[c] \leftarrow \log prior[c]$
    for each each feature $t \in V$ do
      if $t \in V_d$ then
        $score[c] += \log condProb[t][c]$
      else
        $score[c] += \log(1 - condProb[t][c])$
      end if
    end for
  end for
  Return $\arg\max_{c \in \mathbb{C}} score[c]$
```

$\widehat{P}(Software|CS) = \frac{1+1}{3+2} = \frac{2}{5}$.

Finally the probability of d_5 under class CS is estimated as $\frac{3}{4} \cdot (\frac{1}{5})^2 \cdot (1 - \frac{2}{5})^3 \cdot \frac{4}{5} \approx 0.0052$. Note that three features: $Computer, Complexity, Software$ are absent from the test document, and both of their probability is $\frac{2}{5}$, thus we multiply $(1-\frac{2}{5})^3$ into $P(CS|d_5)$. Accordingly, the probability of d_5 under class $nonCS$ is estimated as $\frac{1}{4} \cdot (\frac{2}{3})^2 \cdot (1-\frac{1}{3})^3 \cdot \frac{2}{3} \approx 0.0219$. As a result, d_5 has been classified as a $nonCS$ paper.

We can see that the classification results for d_5 are different when using Multinomial Model and Bernoulli Model. When using Bernoulli Model, the high occurrence of *data* haven't been considered, and two words *Climate* and *Business* provide strong evidence of *nonCS* class, thus it lead Bernoulli Model to predict d_5 into *nonCS* class. In the following chapter, we will use specific experiment to illustrate which model is better for academic papers.

5.6.4 Logistic Regression

As a comparison, we also applied the Logistic Regression (LR) implemented by Scikit-learn [25]. Compared with SVM with non-linear kernel, LR provides linear classification solution and has much faster running speed. Compared with MNB, LR is a discriminative classifier. Instead of transforming the probability of calculating $P(c|d)$ into $P(d|c)$ as MNB, LR takes a more straightforward approach that combing features linearly and applying solving functions to directly calculate the $P(c|d)$. Due to the scalability and efficiency problem, we only apply LR in sentence2vec modeling as a comparison with MNB since sentence2vec models have much lower feature dimensionality.

Moreover, we also utilized the SVM with RBF kernel and Decision Tree algorithm implemented by [25]. However, based on our initial experimental test on smaller arXiv data set, both SVM and decision tree cannot compete with Logistic Regression, and the running times for SVM and Decision Tree are much longer than Logistic Regression. Hence, for the experimental part, we only use Logistic Regression as a comparison with Naive Bayes.

CHAPTER 6

Evaluations

6.1 N Fold Cross Validation and F_1 Measure

In order to evaluate the classifiers, we need to test each document in our training data set to compare the predicted class label and their actual true class label. A common evaluation method is called: cross validation, which means separate the training data set into two parts, one part is used to train the classifier, while the other part is used to test the classifier. By saying test, we mean that the trained classifier is used to predict the class label for another part of the separated data set.

N fold means the data set splitting scheme. We split $\frac{N-1}{N}$ of the data set to be the training set to train the classifier, use the remaining $\frac{1}{N}$ of the data set to be the test set to obtain the predicted class label by using the classifier trained by the $\frac{N-1}{N}$ of the data set. As we can see, the training process is a iteration process, in order to obtain the predicted class label for the whole data set, we need to test each $\frac{1}{N}$ part of the data set, in other words, N classifiers will be trained in order to test different $\frac{1}{N}$ fractions of the data set.

Most popular value of N adopted by most of the researchers is 10, which is called: 10 fold cross validation. First, we use $[0, \frac{1}{10}]$ part of the data to be the test set, and use $[\frac{1}{10}, \frac{10}{10}]$ part of the data to train the classifier. And in the second iteration, use $[\frac{1}{10}, \frac{2}{10}]$ part of the data to be the test set, and use $[0, \frac{1}{10}] \bigcup [\frac{2}{10}, \frac{10}{10}]$ part of the data to train the classifier. Iterate 10 times until all parts of the data have been tested and assigned a predicted class label. In our real evaluation experiments, we use 10-fold for arXiv data set and 3-fold for CiteSeerX data set. Based on the cross validation result, TP, TN, FP, FN can be obtained. TP, True Positive, means the amount of documents that acutual class is True class and the predicted result is also True class, FN is the amount of documents that actual class is

True but predicted to be False class, and FP is the amount of documents that actual class is False but predicted to be True class. F_1 measure defines a weighting between recall and precision. Therefore, higher value of F-measure can ensure the efficiency of the classifier. If we use the harmonic mean of sensitivity and precision, the formula can be written as follows. Note that this formula is used for calculate the F_1 score for each class (for two-class classification case, it calculates the True class, which is CS class in our case).

$$\begin{aligned} F_1 &= \frac{2 \times Precision \times Recall}{Precision + Recall} \\ Precision &= \frac{TP}{TP + FN} \\ Recall &= \frac{TP}{TP + FP} \end{aligned} \tag{1}$$

6.2 Preliminary Study: Classifying conference papers

In order to launch the initial test of our classification system. We firstly use small amount of data to test the classifier. Based on the DBLP published venue metadata, we can find out papers that belong to specific conferences. Based on our observation, there are three very popular conferences: VLDB, SIGMOD, ICSE which include similar amount of papers. The details of the three conferences are listed below:

- **VLDB**: the International Conference on very Large Databases. (2,728 papers identified based on DBLP venue metadata.)
- **SIGMOD**: the International Conference on Management of Data. (2,113 papers identified.)
- **ICSE**: the International Conference on Software Engineering. (2,245 papers identified.)

We can also see from Fig. 11 that the paper length distributions among the three conferences are similar with each other. Both three conferences have top amount of papers within length with 3000 - 4000 words. Note that the word count is based on the normalized text (after stop words removal, remove non-English words, etc...). We first classify VLDB and SIGMOD. Since they are both belong to the topic of database, we consider it is difficult

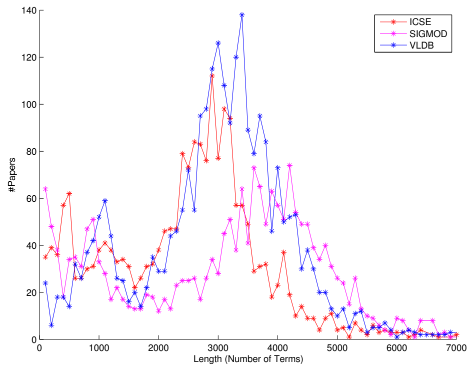


FIGURE 11: Length Distribution of VLDB, SIGMOD and ICSE Papers

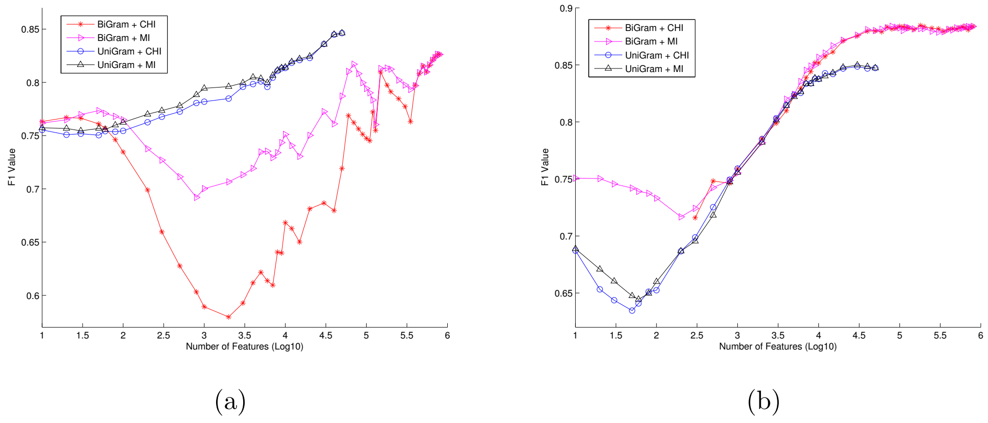


FIGURE 12: VLDB and SIGMOD Classification Results in Bernoulli (a) and Multinomial (b) Naive Bayes

for the classifier to distinguish papers among these two conferences. By classifying these two conferences, we can see the ability of the classifier to separate papers with similar topics. Then, we apply our classifier on VLDB and ICSE. These two conferences are totally different topics: database and software engineering. The classifier is expected to have much better classification result compared with classifying VLDB and SIGMOD. Before the classification experiment, conference specific data in full text need to be removed. We take a simple approach, cut the head and tailing part of the paper, only keep the middle $\frac{1}{3}$ part of the full text.

6.2.1 Classifying VLDB and SIGMOD papers

First we apply feature selection (Mutual Information and χ^2) onto VLDB and SIGMOD papers, and rank the features based on their feature selection scores. Then select different

Top 20 Features by Mutual Information			Top 20 Features by χ^2		
Feature	CDF in VLDB	CDF in SIGMOD	Feature	CDF in VLDB	CDF in SIGMOD
mining techniques	4	50	mining techniques	4	50
note used	0	20	tuples input	3	27
tuples input	3	27	note used	0	20
node system	0	19	node system	0	19
since similar	0	17	data store	1	19
map reduce	0	17	since similar	0	17
root operator	0	17	map reduce	0	17
data store	1	19	root operator	0	17
candidate query	3	22	candidate query	3	22
ii one	0	15	section present	219	226
arbitrarily nested	0	15	small fraction	25	50
source queries	0	15	nodes network	2	19
way computing	1	17	way computing	1	17
second one	77	17	ii one	0	15
nodes network	2	19	arbitrarily nested	0	15
section present	219	226	source queries	0	15
small fraction	25	50	equivalence classes	20	43
sorted order	97	26	current value	22	45
clustering results	0	14	clustering algorithm	22	45
hierarchical clustering	0	14	second one	77	17

TABLE 13: Top 20 VLDB and SIGMOD Bi-gram Features Selected in Bernoulli Naive Bayes (without stemming)

amounts of top ranked features to compare the classification results. As can be seen in Fig. 12 (a), there are four different curves shown. Under Bernoulli Naive Bayes model, we experimented with four different combinations of N-gram (Uni-gram and Bi-gram) and feature selection algorithms (Mutual Information and χ^2) to train four different classifiers to compare their performances. X-axis represents how many features are used. We can see that both of the four classifiers reach their best classification results when using all features. And Uni-gram modeling with all features performs the best, the F_1 measure value is 0.846. Table 13 lists the top 20 ranked Bi-gram features by Mutual Information algorithm and χ^2 algorithm. The features we listed are words before stemming, since it is more human readable. Actually in the actual classification system, all the features are used after stemming. In the table we can see that although the two conferences are in the same topic Database, there are still some words that tend to be used more often in only one specific conference. Papers in SIGMOD have more exclusive and unique features that can be classified out from VLDB papers, such as ”map reduce”, ”root operator”, ”node system”, etc...

Fig. 12 (b) shows the classification results when using Multinomial Naive Bayes model. This time, features are represented by their class term frequencies, instead of class document frequencies. In Multinomial Model, Bi-gram outperforms Uni-gram modeling throughout all subsets of ranked features, and χ^2 feature selection algorithm performs the best. When selecting 90,000 top ranked features, the F_1 measure value reaches the highest, which is 0.8847. The over fitting problem can also be seen in the figure, when the number of features

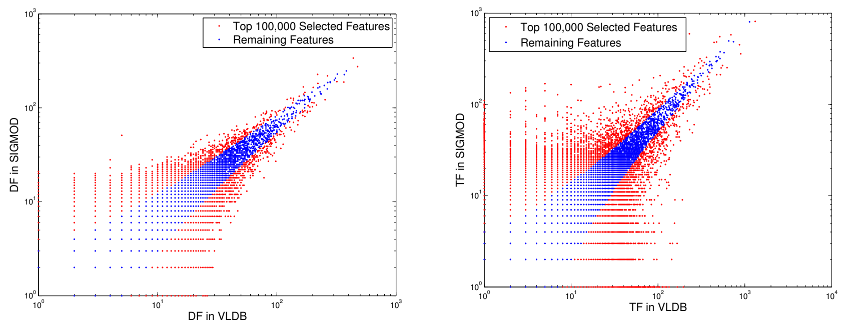
Top 20 Features by Mutual Information			Top 20 Features by χ^2		
Feature	CTF in VLDB	CTF in SIGMOD	Feature	CTF in VLDB	CTF in SIGMOD
gamma gamma	149	612	gamma gamma	149	612
data mining	229	593	data mining	229	593
aa aa	4	168	aa aa	4	168
edge cover	2	151	edge cover	2	151
world set	2	142	base view	9	164
base view	9	164	world set	2	142
ss tree	2	139	ss tree	2	139
ripple join	1	133	reduced tree	4	140
reduced tree	4	140	ripple join	1	133
dr bones	195	0	bounding rectangles	21	156
mining techniques	6	127	mining techniques	6	127
heavy hitters	2	109	base views	20	154
base views	20	154	isolation levels	27	165
bounding rectangles	21	156	heavy hitters	2	109
pig latin	0	100	pig latin	0	100
support lattice	0	99	support lattice	0	99
isolation levels	27	165	strategy ce	0	96
strategy ce	0	96	vdag strategy	0	95
safe plan	166	0	dr bones	195	0
vdag strategy	0	95	flash memory	2	94

TABLE 14: Top 20 VLDB and SIGMOD Bi-gram Features Selected in Multinomial Naive Bayes (without stemming)

are too big (more than around 10^4.5), the classification results stop increasing, instead, if the number of features used keep growing, the classification results (F_1 measure) start to drop slightly. From Table 14 we can see that the top ranked features in Multinomial Model is different from Bernoulli Model.

From Fig. 13 we can more easily see the difference between high ranked features and low ranked features. In the figure, red nodes are the top 100,000 selected features by χ^2 algorithm, and the blue nodes are the remaining features. The left figure shows the ranked features in Bernoulli model, X-axis represents the features' document frequencies in VLDB class, and Y-axis represents the features' document frequencies in SIGMOD class. We can see that all the red nodes are deviated from $y = x$. Red nodes have greater differences of two classes' document frequencies, which means they are more dependent to only one specific class instead of commonly appear in both of the classes. Accordingly, in the right figure which is the ranked features in Multinomial model, X-axis and Y-axis represent the features' class term frequencies in VLDB and SIGMOD respectively.We can see that the parts of the red nodes are even more deviated from $y = x$, which shows that by using class term frequency to select features, top ranked features can be separated more evidently.

We also compared the effect of stop words and stemming to classification results. From Fig. 14 we can see that after performing stemming and stop words removal, the classification result can indeed reach the highest F_1 measure value. Removing stop words can significantly give a performance boost to the classifier (red and pink curves compared with black and blue curves). Stemming can further provide a slight performance boost (compared with red



(a) Bernoulli Model (b) Multinomial Model

FIGURE 13: Comparison of high score features and low score features

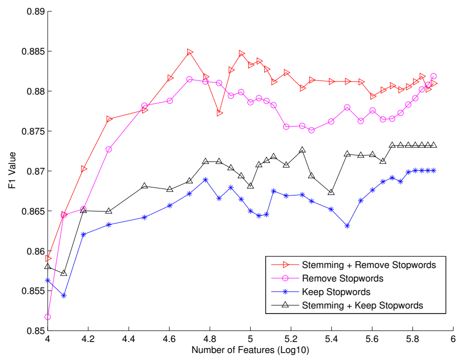


FIGURE 14: The effect of stop words and stemming to classification results

and pink curves).

6.2.2 Classifying VLDB and ICSE papers

We have already proven that Multinomial Naive Bayes and perform better than Bernoulli Naive Bayes. Now we change the experiment data set into VLDB (positive class) and ICSE (negative class) papers and see the capability of the classifier to classify papers from different topics. Fig. 15 shows the VLDB and ICSE classification results. We can see that Bi-gram modeling + χ^2 algorithm (the red curve) still performs the best. When selecting 600,000 features, the classifier reaches the highest F_1 measure value: 0.98534. We can see that both of the four curves can all maintain at a very high F_1 measure value range (all higher than 0.9), this is because the two different topics (database and software engineering) can be easily separated. We can also see from Table 15 that the top ranked features are more dependent with only one specific class. For example, software engineering is clearly tend to

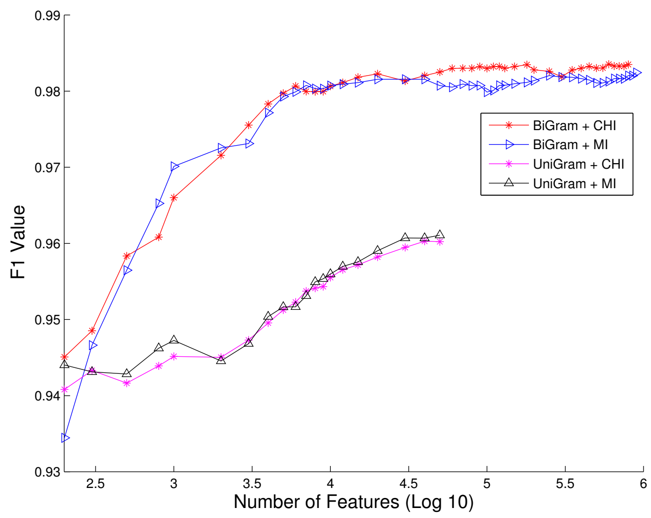


FIGURE 15: VLDB and ICSE Classification Results in Multinomial Naive Bayes

Feature	CTF in VLDB (N_{11})	CTF in ICSE (N_{10})
test cases	4	1314
source code	30	1062
software engineering	9	868
test suite	2	709
test case	10	715
software development	4	538
test suites	1	467
control flow	29	494
gamma gamma	131	679
main memory	775	17
join point	0	387
case study	34	396
query processing	555	3
leaf node	564	21
user interface	82	424
lines code	27	310
process model	19	288
development process	5	256
delta delta	469	927
match pcd	0	230

TABLE 15: Top 20 VLDB and ICSE Bi-gram Features Selected in Multinomial Naive Bayes (without stemming)

belong to ICSE class, and query processing is clearly tend to belong to VLDB class.

6.3 Classifying arXiv Papers

We separate the arXiv papers into 2 parts, papers labeled with CS are added into the positive class set, and all the other papers are added into the negative class set.

6.3.1 Experimental Settings

Our experiments mainly separate to two different kinds of text modeling techniques: one is to extract the Uni-gram and Bi-gram features out from the paper text (title + abstract) and build the feature set to apply our Naive Bayes classifier. The other experiment is to use sentence2vec technique to lower down the feature dimensionality. After series tests, we find the combination of sentence2vec PV-DM model with vector dimension = 100, window size = 10, negative sample = 5 gives the best embeddings for classification. Thus, we keep this setup in the following experiments. Compared with N-gram modeling, the feature dimensionality of sentence2vec is much lower, which is only 100 that make it possible to apply more complex algorithms such as Logistic Regression.

Our arXiv training set contains 160,000 documents with 80,000 CS and 80,000 non-CS. Each document contains paper title and abstract. Before the classification experiments, we tokenize the text of each paper where each token contains only alphanumeric letters, and each token is also case folded. Furthermore, in order to test the effect of stemming and removing stop words to the classification results, we conduct our experiments based on the following four text models:

1. Remove Stopwords (SW)
2. Stemming (ST)
3. Remove Stopwords + Stemming (SW + ST)
4. Original Text (OT)

The four models are the combinations of two control parameters: Remove Stopwords, Stemming. All of the models can be used to build Uni-gram and Bi-gram feature set, also

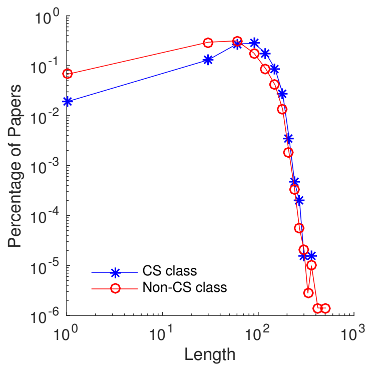


FIGURE 16: Paper length distribution of arXiv data set

used to build sentence2vec feature set. For the Uni-gram and Bi-gram models, three different feature weighting schemes are tested and compared: un-normalized, length normalized, fully normalized (TF-iDF + length normalization). Based on the concept, we mainly summarize the following three questions that need to be answered:

- Experiment 1: What is the best text models (Impact of stop words and stemming)?
- Experiment 2: What will the size of the training data set affect the classification results?
- Experiment 3: What will the number of features affect the classification results?

For the first experiment, we use the full data set (160,000 documents) applied with N-gram and sentence2vec to find out the best text model out from the four different models SW, ST, SW+ST, OT. For the second experiment, we gradually increase the size of the training data set while using the whole features. The training data set size increased from 1,000 to 120,000 (9 round experiments in total with 9 different sizes of data set). The last experiment is changing the number of features. We use χ^2, MI and PMI feature selection algorithms to obtain the score for each feature and rank them based on the score in descending order. By selecting different top K ranked features, we can compare the differences of the classification results. Note that feature selection is only for Uni-gram and Bi-gram features, sentence2vec is not applicable.

Fig. 16 shows the length distribution of arXiv data set, the blue curve is for documents in CS class, and the red curve is for documents in non-CS class. The value of length is the number of words in a document after stop words removal. The document count is

		SW	ST	SW + ST	OT
MNB	Avg. Accuracy	0.9079	0.9034	0.9049	0.9018
	Avg. Specificity	0.9007	0.8990	0.8953	0.8987
	Avg. Precision	0.9021	0.8999	0.8972	0.8992
	Avg. Recall	0.9150	0.9077	0.9146	0.9049
	Avg. F_1	**0.9085**	0.9038	0.9058	0.9021
	Time Consumed	0:01:38	0:01:37	0:01:37	0:01:38
LR	Avg. Accuracy	0.9292	0.9259	0.9266	0.9260
	Avg. Specificity	0.9265	0.9222	0.9237	0.9231
	Avg. Precision	0.9269	0.9228	0.9241	0.9035
	Avg. Recall	0.9318	0.9295	0.9295	0.9290
	Avg. F_1	**0.9293**	0.9261	0.9268	0.9262
	Time Consumed	0:03:49	0:03:51	0:03:47	0:03:48

TABLE 16: Classification Results Comparison by using sentence2vec 100D features on 160,000 balanced arXiv data set

based on length range 30, which means the lengths between $[30n, 30(n+1)]$ are group and count together. Lengths are normalized into $\lfloor\frac{L}{30}\rfloor \times 30$. We can see from Fig 16 that the length distribution of CS and non-CS documents are almost the same. For CS class, most documents are centralized into length range of 60-90, while non-CS class centralized the most in 90-120.

6.3.2 Impact of stop words and stemming

All the experiments in Table 16 on the four different text models are based on 100 dimension features sentence2vec. Two algorithms are used: Multinomial Naive Bayes and Logistic Regression. Table 16 shows the classification results of different text models, the precision, recall and F_1 measure are evaluated by using 10 fold cross validation scheme, and the average score is calculated by 10 sub-scores created by 10 fold. We can see that for all the experiments, Logistic Regression outperforms Multinomial Naive Bayes with more than 2% F_1 measure value. If measured by Logistic Regression, the rank of the four text models should be: $SW > SW + ST > ST > OT$. The two models that retain the stop words perform the worst, and stemming lower down the performance a little bit. Since sentence2vec rely on the context to analyze the text, stemming may damage part of the context in a document. As a conclusion, removing stop words improves the classification accuracy in all the methods, even in sentence2vec, while stemming has limited impact to the improvement. Moreover, we can also see from Table 16 that precision and recall are all

		SW	ST	SW + ST	OT
sentence2vec		**0.9085**	0.9038	0.9058	0.9021
Uni-gram	Un-NL	**0.9326**	0.9293	0.9289	0.9288
	Len NL	**0.9324**	0.9289	0.9286	0.9280
	Full NL	**0.9354**	0.9317	0.9314	0.9312
Bi-gram	Un-NL	**0.9460**	0.9425	0.9440	0.9424
	Len NL	**0.9469**	0.9435	0.9447	0.9433
	Full NL	**0.9467**	0.9449	0.9448	0.9444

TABLE 17: Multinomial Naive Bayes F_1 measure comparison on 160,000 balanced arXiv data set

in a high level, which shows that FP (False Positive) is as low as FN (False Negative).

Table 17 shows the F_1 measure comparison between sentence2vec method and N-gram methods under Multinomial Naive Bayes. We can see that for all the four models, both Uni-gram and Bi-gram surpass sentence2vec method, and SW model always performs the best. When using SW model and Bi-gram with length normalized feature weight, the classification performance is the best where the F_1 measure reaches 0.9469. However, we can see that except the Bi-gram SW model, all the other models, no matter Uni-gram or Bi-gram, full normalized feature weight has the best performance, even though in Bi-Gram SW, Len NL only has very tiny gap with Full NL (0.0002) which can be ignored. In general, for Uni-gram, the feature weight performance ranking is $FullNL > Un - NL > LenNL$, and for Bi-gram, the feature weight performance ranking is $FullNL > LenNL > Un - NL$. Length normalization didn't bring too much improvement which is because that the length of abstract + title in papers of arXiv data set has similar length, not like CiteSeerX data set. However, full normalization can improve the classification results by adding TF-iDF weighting.

Another important measurement is the classification program run time. We know that Naive Bayes is the fastest and simplest classification algorithm compared with other more advanced algorithm such as Logistic Regression. Based on the experimental results, we can see that indeed Naive Bayes has the highest efficiency, for 10 fold cross validation and 160,000 papers, it only took around 1 and half minutes to finish the classification task, while Logistic Regression took around 3 - 4 minutes.

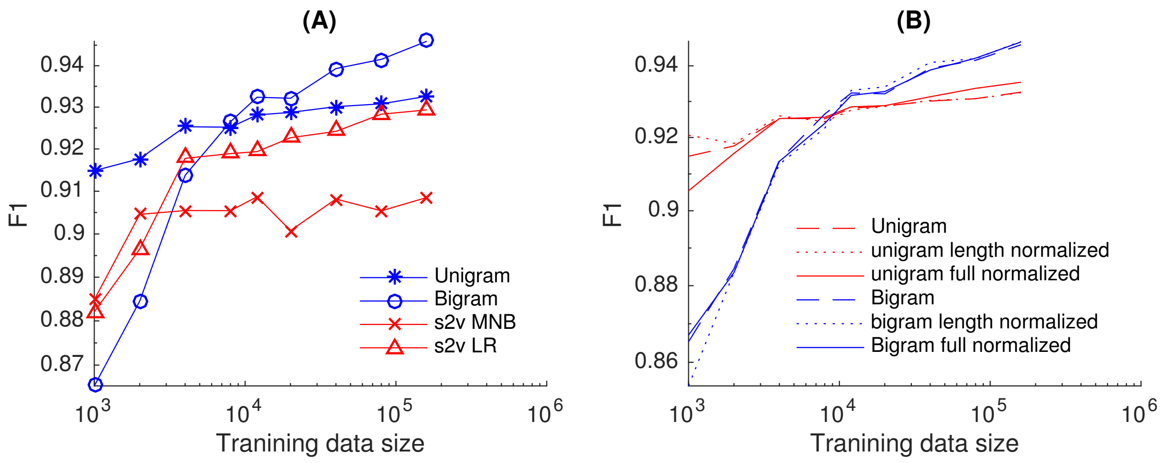


FIGURE 17: Impact of training data size on F_1. (A) Both Uni-gram and Bi-gram model out-perform sentence2vec when training data is large; (B) Normalization plays a minor role in this data.

6.3.3 Impact of Training Set Size

We randomly sampled 9 different training sets with different sizes. Size varies from 1,000 to 160,000. Within each sampled training set, the number of CS documents and non-CS documents are equal to make the data set balanced. In each round, there are 6 sub-experiments with 8 different classification systems: (1, 2) sentence2vec + Multinomial Naive Bayes / Logistic Regression, (3, 4, 5) Uni-gram (un-normalized / length normalized / fully normalized) + Multinomial Naive Bayes, (6, 7, 8) Bi-gram (un-normalized / length normalized / fully normalized) + Multinomial Naive Bayes. For Uni-gram and Bi-gram, un-normalized feature weight use the features' TF (term frequency) as feature values to do the classification.

Fig. 17 shows the comprehensive classification result curves for the 8 different classification systems. The 9 nodes along the curves is the F_1 measure value with 9 different training set sizes. From Fig. 17 (A) we can clearly see that from the beginning the classification results increase fiercely until the size reaches around 2×10^4. Ignore some outliers, we can see that even if the training set size keep increasing, the classification results still slowly become better. Almost for all of the 6 systems, the classification results reach the best when the training set size reaches the largest value 160,000. We can see that at the beginning when training set size is 1,000, the Bi-gram model performs the worst, while the Uni-gram model performs the best. An interesting phenomenon is that when training set size increase, Bi-gram model surpass all the other systems quickly and become the best performer. As a comparison, Uni-gram model and the two sentence2vec models fluctuate

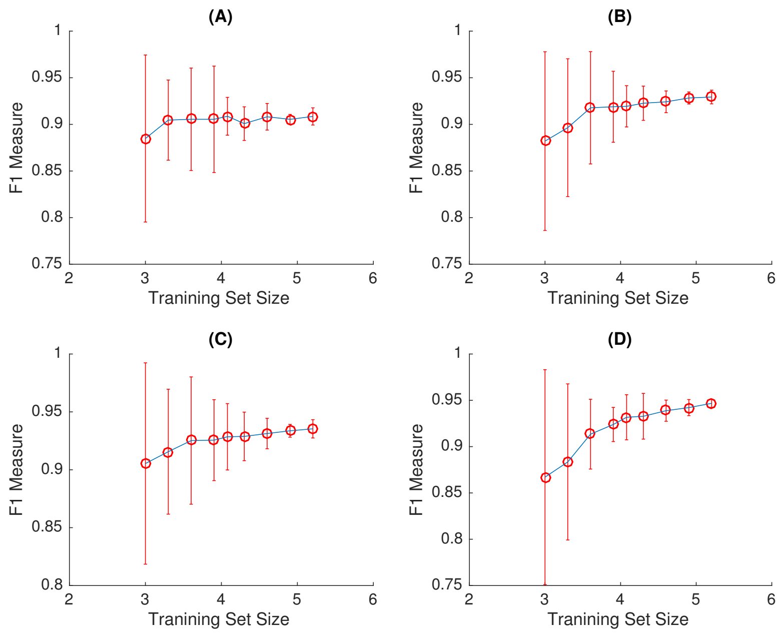


FIGURE 18: Error bar plot of (A) sentence2vec Multinomial Naive Bayes; (B) sentence2vec Logistic Regression; (C) Uni-gram full normalized; (D)Bi-gram full normalized

less than Bi-gram model, and sentence2vec models perform always slightly worse than Uni-gram model. Moreover, for Logistic Regression and Multinomial Naive Bayes algorithms applied in sentence2vec model, at first Multinomial Naive Bayes performs better than Logistic Regression, however, Logistic Regression quickly surpass Multinomial Naive Bayes and the performance gap become increasingly bigger.

17 (B) shows more details of 6 different N-gram models, mainly shows the effect to the classification results of the two feature weight normalization methods. We can see that for both the Uni-gram and Bi-gram, when the training set is large enough, fully normalized feature weight can have slightly better performances. For Uni-gram models, fully normalized first performs the worst, but with the increase of the training set size, it turn out to be the best performer within all Uni-gram models.

Fig. 18 shows the differences of minimum F_1 value and maximum F_1 value with the increase of the training set size, we can see that for both the four models showed in the four plots, the variance range become smaller when training set size increase. Compared with (A) (B), Multinomial Naive Bayes has smaller gap between max F_1 and min F_1.

Table 18 also lists the statistics of average F_1 measure values corresponding to Fig. 17.

		Training Size = 1,000		Training Size = 160,000	
		MNB	LR	MNB	LR
sentence2vec		0.8848	0.8821	0.9085	0.9293
Uni-gram	Un-NL	0.9148	-	0.9326	-
	Len NL	0.9206	-	0.9324	-
	Full NL	0.9054	-	0.9354	-
Bi-gram	Un-NL	0.8653	-	0.9460	-
	Len NL	0.8985	-	0.9469	-
	Full NL	0.8671	-	0.9467	-

TABLE 18: Statistics of Classification Results with the increase of training set size

	Uni-gram	Bi-gram
MI & χ^2	0.97984	0.97011
χ^2 & PMI	0.62230	0.74242
MI & PMI	0.61009	0.71542

TABLE 19: Spearman Correlation between top 1000 MI, PMI and χ^2 selected features

6.3.4 Impact of feature size

The whole training data set (160,000 papers) contains 149,731 different Uni-grams and 5,200,488 different Bi-grams. Although the N-gram feature dimensionality is much larger than sentence2vec, the N-gram feature set is usually a sparse matrix, which means each paper only has very tiny portion of the whole feature set (tiny portion of words appear in the same paper), which make Multinomial Naive Bayes possible to deal with such cases.

We used three different feature selection algorithms: MI, PMI and χ^2 to compare the differences. From Table 19 we can see that the feature ranks based on scores in an descending order under MI and χ^2 are very similar, while PMI is different from MI and χ^2. Both MI and χ^2 tend to select popular words with higher scores, and PMI tend to select very exclusive words. Since during the preliminary experiment of VLDB, SIGMOD and ICSE classification, we proved that χ^2 performs slightly better than MI (although they have almost the same performances), in the sections below, we will analyze the differences between χ^2 and PMI.

Analysis of top ranked features by χ^2

In Table 20, we list the top scored Uni-gram and Bi-gram features using χ^2 feature selection algorithm. For the features that have higher CTF in CS, we bold and use red color to

Uni-gram					Bi-gram				
Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS	Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS
quantum	2856	24576	871	9933	magnetic field	28	2924	13	1738
algorithm	41238	4843	19204	2669	**state art**	4422	279	4005	268
field	4523	20706	3309	11714	field theory	46	2204	37	1575
network	31142	4014	13213	1809	two dimensional	679	3150	525	2433
performance	23800	2081	14981	1613	**log n**	3705	262	1858	181
algorithms	23178	2032	12952	1275	**polynomial time**	3364	156	2409	119
based	47442	12647	29280	10206	x ray	65	1781	45	809
spin	248	10336	134	4011	**paper propose**	3348	234	3344	234
0	5897	19455	3643	8987	ground state	10	1515	8	1104
equation	1193	11352	790	6531	**o n**	3631	441	1933	292
information	28191	4953	15016	3224	black hole	68	1492	23	768
networks	25829	4076	12044	1607	boundary conditions	94	1483	70	1046
learning	16895	965	7206	467	**real world**	2849	244	2470	218
problem	40677	11110	21870	7272	su 2	2	1223	2	769
magnetic	247	8610	178	3925	phase transition	247	1733	158	1161
theory	9177	22542	6141	13581	cross section	25	1247	21	854
user	13107	242	6828	166	one dimensional	404	1993	327	1606
mass	511	8547	369	4777	**paper presents**	3072	366	3040	364
channel	17799	1890	6802	1124	yang mills	0	1151	0	686
equations	2026	11147	1289	6497	**sensor networks**	1922	14	1160	11
data	42199	13137	18081	7917	**real time**	2798	303	1747	228
paper	45105	14926	39094	13182	perturbation theory	13	1121	12	855
users	11392	202	5956	131	standard model	66	1232	59	886
x	5721	16323	1812	5437	phase space	37	1165	28	794
proposed	24313	5188	16596	4409	quantum mechanics	86	1264	72	875
temperature	605	7687	309	4470	**wireless networks**	1768	12	1238	6
complexity	15063	1529	8902	934	low energy	91	1238	76	980
codes	12805	779	3794	345	**results show**	3790	777	3737	766
2	19117	32629	10078	15936	**network coding**	1721	7	750	4
gauge	45	6069	37	3025	**wireless sensor**	1647	5	943	4
phase	3265	12005	1715	6261	cross sections	15	1005	11	685
electron	111	6124	66	3183	**ad hoc**	1936	96	1154	89
states	2775	11167	1931	6264	e e	106	1202	80	716
dimensional	4830	14118	3123	9573	**neural networks**	2139	176	1383	116
wireless	9540	83	4863	48	gauge theory	3	948	2	686
wave	599	6904	342	3937	**paper present**	2894	464	2888	464
particle	659	6536	370	3916	**machine learning**	2061	154	1563	116
graph	16001	2774	6798	1403	**neural network**	1921	109	1274	83
lattice	1581	8111	734	4415	monte carlo	840	2351	651	1755
surface	1137	7246	645	4033	three dimensional	375	1631	286	1259
1	22542	33914	10719	16236	lie algebra	16	935	6	641
algebras	547	6046	313	2709	**simulation results**	2318	272	2266	258
group	4162	11975	2199	6610	mean field	231	1368	128	959
particles	246	5309	147	3152	bose einstein	10	909	5	583
scattering	282	5303	147	3002	electric field	14	913	11	617
design	13082	1904	8389	1333	time dependent	192	1256	141	945
dynamics	2843	9737	1754	6082	1 2	1745	3442	1231	2423
distributed	10819	1096	5953	843	**np hard**	1648	69	1389	61
communication	10083	867	5953	561	density functional	4	851	3	655
quark	5	4360	3	2202	black holes	7	856	6	478
coupling	612	5640	405	3628	moduli space	3	845	3	537
symmetry	615	5641	345	3632	**multiple access**	1433	10	962	9
coding	8151	320	3580	186	angular momentum	3	841	2	542
density	2822	9389	1864	5689	lattice qcd	13	857	8	592
software	8466	440	3977	301	**worst case**	1662	92	1202	73
nodes	9921	936	5002	543	phys rev	9	828	8	734
image	10622	1221	4119	829	**proposed algorithm**	1649	100	1327	85
optimal	15816	3350	9162	2042	u 1	19	833	14	540
algebra	1559	7243	942	3724	**time algorithm**	1505	57	1200	50
transition	1524	7044	1021	4155	lie algebras	7	800	2	471
access	7687	306	4589	265	mathbb r	465	1637	293	893
language	8240	522	4459	373	good agreement	40	871	40	851
logic	8315	567	3560	299	long range	151	1096	101	776
qcd	20	3916	10	2159	**n log**	1583	90	954	72
capacity	8606	690	3876	409	**channel state**	1302	8	1082	8
web	7142	246	2910	149	correlation functions	4	770	4	581
efficient	11996	2050	9259	1723	**computational complexity**	1645	122	1427	106
propose	15135	3433	13411	3244	renormalization group	5	761	2	564
limit	1668	6913	1345	5012	**o log**	1525	83	950	62
boundary	1222	6116	715	3465	experimental data	176	1117	160	1012

TABLE 20: Top 70 Uni-gram and Bi-gram arXiv Features Selected by χ^2 (features with higher CTF in CS are highlighted with red color)

highlight the text to make them more clearly to be seen. It can be seen that the top selected features are clearly related to CS or non-CS subjects, such as network, algorithm, codes, machine learning should more related to CS areas, while quantum, magnetic, spin, black hole should more related to non-CS areas. We can see that although the top scored features appear lots of time in both classes, their class term frequency can still differs greatly. For example, "algorithm", as the second ranked Uni-gram feature, appears lots of times in both classes, but still have predominant term frequency in CS class which can still be a good proof of CS class if a document has word "algorithm". Accordingly, the first ranked Uni-gram feature "quantum" is obviously a Physics word. Although it also occur 2,856 times in CS documents, but it occur 10 times more in non-CS documents. As for the Bi-gram ranked feature list, the top ranked features are more deviated to one certain class compared with Uni-gram features. For example, "magnetic field" as the first ranked feature only occur 28 times in CS documents, while "multiple access" occurs 1433 times in CS documents and only 10 times in non-CS documents which can be a strong evidence of being in CS class. We can also see from Table 20 that the ratio of CS features and non-CS features tend to be equal, which proves that both classes have adequate amount of popular but still distinguishable features. In addition, for another feature selection algorithm Mutual Information, the scored result is very similar with χ^2, thus we don't list the features selected by Mutual Information here.

Analysis of top ranked features by PMI

However, another feature selection algorithm: Pointwise Mutual Information (PMI), tend to score smaller (unpopular but more exclusive) features to the top; in other words, under PMI, a top ranked word could only appear in one class but hardly appear in the other class. Table 21 shows the top 70 scored Uni-gram and Bi-gram features using PMI. We can clearly see the differences of the top scored features using the two algorithms, almost all of the top features totally biased to only one class. Compared with χ^2 algorithm, some of the top scored features maybe not so popularly used, but are exclusively used by only one class of papers. Overall, top features selected by χ^2 can be more meaningful words, while PMI may highly scored some strange words. Since under PMI, most of the popular words are ranked as lower position, select too few amount of features may seriously lower the classification accuracy. We can see from Table 21 that the top selected features are

Uni-gram					Bi-gram				
Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS	Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS
supersymmetric	0	1828	0	1121	yang mills	0	1151	0	686
mev	0	1586	0	925	gauge theories	0	714	0	486
chiral	1	3074	1	1525	su 3	0	680	0	419
pion	0	1161	0	658	non perturbative	0	636	0	497
supersymmetry	0	1035	0	623	spin orbit	0	621	0	345
branes	0	1003	0	465	**sum rate**	946	0	494	0
mesons	0	948	0	603	quantum gravity	0	554	0	347
hadronic	0	921	0	661	heavy quark	0	547	0	341
phonon	0	883	0	428	quantum hall	0	518	0	288
baryon	0	879	0	503	mills theory	0	506	0	347
beamforming	1439	0	616	0	k ahler	0	503	0	265
nucleon	1	1683	1	835	chern simons	0	486	0	274
fermions	1	1602	1	1052	chiral symmetry	0	478	0	330
quark	5	4360	3	2202	de sitter	0	448	0	252
electroweak	0	703	0	442	cp violation	0	439	0	256
supergravity	0	694	0	390	rev lett	0	437	0	406
lepton	0	655	0	451	energy momentum	0	433	0	274
gluon	1	1301	1	748	cosmological constant	0	426	0	276
fermionic	0	607	0	441	moduli spaces	0	421	0	291
quarks	1	1182	1	840	au collisions	0	420	0	232
parton	0	583	0	372	c algebra	0	416	0	258
superfluid	0	574	0	298	su 2	2	1223	2	769
precoding	949	0	388	0	**mimo systems**	680	0	417	0
condensate	1	1134	1	654	**outage probability**	679	0	430	0
ionization	0	564	0	317	gev c	0	403	0	264
ahler	0	552	0	291	transverse momentum	0	402	0	301
singlet	0	545	0	362	coupling constant	0	402	0	345
multicast	891	0	370	0	**access control**	665	0	413	0
meson	2	1563	1	929	**logic programs**	665	0	349	0
gev	4	2544	3	1399	collisions sqrt	0	386	0	257
cnn	842	0	359	0	gauge invariant	0	378	0	289
kev	0	488	0	285	hall effect	0	363	0	236
colliders	0	467	0	311	quark masses	0	359	0	279
p2p	764	0	306	0	sigma model	0	353	0	250
fermion	2	1293	2	814	**ieee 802**	590	0	342	0
chern	1	860	1	504	**interference alignment**	580	0	284	0
qos	1431	1	643	1	chiral perturbation	0	346	0	248
orbifold	0	426	0	259	conformal field	0	342	0	254
csit	685	0	198	0	200 gev	0	337	0	218
yukawa	0	393	0	243	gauge field	0	336	0	266
quenched	1	777	1	549	**massive mimo**	561	0	254	0
massless	1	768	1	583	beta decay	0	332	0	177
downlink	1255	1	723	1	**full duplex**	533	0	229	0
exciton	0	373	0	182	deep inelastic	0	318	0	237
protons	0	371	0	267	top quark	0	316	0	186
ultracold	0	362	0	221	**relay channel**	517	0	287	0
galactic	1	720	1	425	dirac operator	0	305	0	209
hyperfine	0	350	0	202	**power allocation**	999	1	496	1
nonperturbative	0	349	0	281	**mobile ad**	498	0	328	0
isospin	0	344	0	224	magnetic moment	0	292	0	201
pseudoscalar	0	330	0	243	quark mass	1	584	1	414
helicity	0	329	0	201	dimensional electron	0	291	0	218
dilaton	0	328	0	187	non relativistic	0	291	0	244
baryons	0	325	0	209	**decode forward**	486	0	344	0
brst	0	321	0	158	inelastic scattering	0	289	0	232
tev	1	628	1	393	**information csi**	484	0	481	0
majorana	0	308	0	169	**interference channel**	966	1	521	1
hubbard	0	308	0	196	excited states	0	286	0	244
mssm	0	306	0	174	sqrt s	0	279	0	162
polyakov	0	301	0	178	gauge fields	0	276	0	218
hubble	0	301	0	201	al phys	0	272	0	245
pions	0	300	0	222	transport properties	0	271	0	231
photoproduction	0	295	0	171	pb pb	0	265	0	143
magnetoresistance	0	292	0	177	j psi	1	525	1	208
deuteron	0	289	0	159	effective potential	0	261	0	187
condensates	1	579	1	366	pb collisions	0	259	0	138
mills	3	1159	1	692	electric dipole	0	257	0	168
rhic	1	568	1	341	integrable systems	0	254	0	192
floer	0	281	0	128	**802 11**	421	0	225	0
uplink	913	1	499	1	spin polarization	0	246	0	149

TABLE 21: Top 70 Uni-gram and Bi-gram arXiv Features Selected by PMI (features with higher CTF in CS are highlighted with red color)

very exclusive, for example: ”p2p” occurs 764 times in CS documents and 0 times in non-CS documents, and ”mimo systems” occurs 680 times in CS documents and also 0 times in non-CS documents. Such words maybe quite popular within a small range of research groups; however, it is inappropriate to represent the common words of whole CS research areas. Therefore, by using PMI, enough amount of features must be included to ensure the classification accuracy. Moreover, the top ranked features under PMI are almost all belong to non-CS class, while top CS features ranked much lower than non-CS features. This is because non-CS papers have more highly occurred exclusive words.

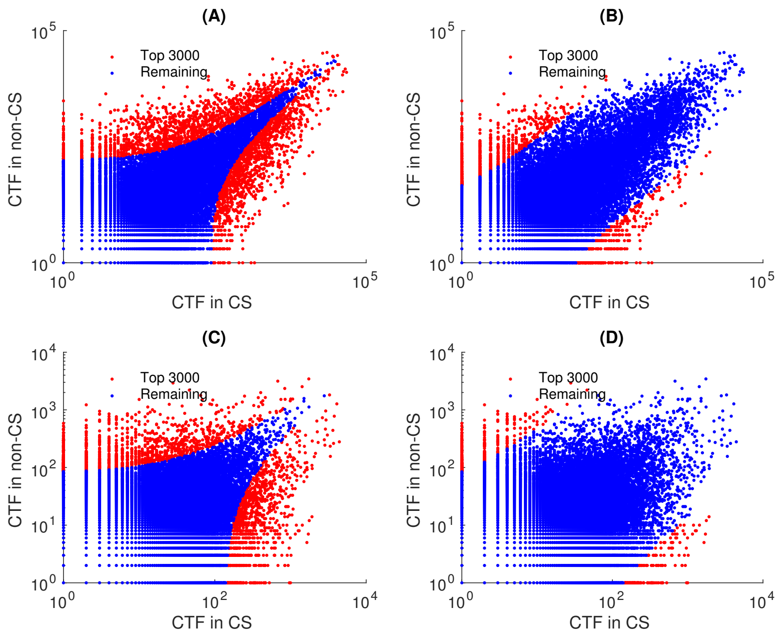


FIGURE 19: Top features selected by different methods. Panel (A) and (C): χ^2; (B) and (D): PMI. (A) and (B) are Uni-gram models; (C) and (D) are Bi-gram models.

Feature Distribution

In Fig. 19 we can clearly see the plotted top ranked few features are all bias in only one specific class, either CS or non-CS. X-axis represents the CTF (Class Term Frequency) in CS class, Y-axis represents the CTF in non-CS class. In Fig. 19, red nodes represent the top 3,000 ranked Uni-gram features, and the blue nodes are all the remaining features. We can see that all of the red nodes are deviated from $y = x$, which means the CTF in one class are significantly larger than the other class. Form Fig. 19 we can also see that χ^2 tend to select big/popular words have high total occurrences but still have significant different

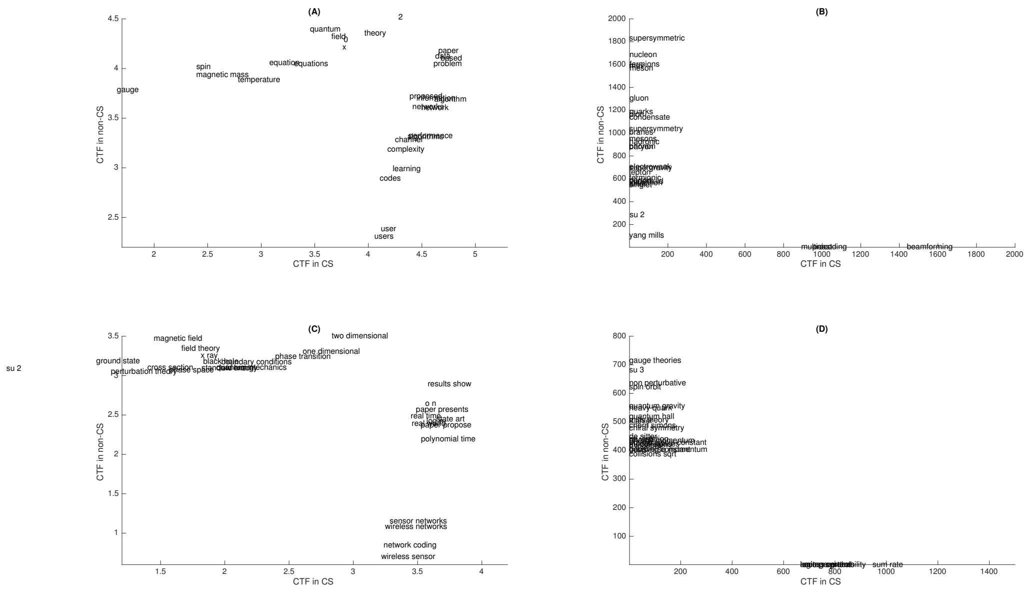


FIGURE 20: The name of the top 30 features selected by different methods. Panel (A) and (C): χ^2; (B) and (D): PMI. (A) and (B) are Uni-gram models; (C) and (D) are Bi-gram models.

occurrences in the two classes while PMI tend to select words that CTF in other class is almost 0. Fig. 20 shows the CTF distribution of the name of the top 30 features. For panel (A) and (C) we log the axis in order to see the text more clearly. We can see that for PMI algorithm, all the top features are attached to the X-axis or Y-axis that shows 0 CTF in other class.

Experiment Results

Fig. 21 shows the classification results when using three different kinds of feature selection algorithms: PMI, MI and χ^2. For each algorithms, we use Bi-gram, Uni-gram respectively, and also use the three different feature weighting schemes: length / fully normalized feature weight and original Class Term Frequency as feature weight (six models in total). We can see that when using less amount of features, the classification results of using PMI drop significantly compared with χ^2 and MI. However, for un-normalized Uni-gram features, from Fig. 21 (A) we can see that when using more than top 30,000 Uni-gram features, actually the classification results are better than using all features, and also more features can let PMI performs better than χ^2 for all the six models; and using original class term

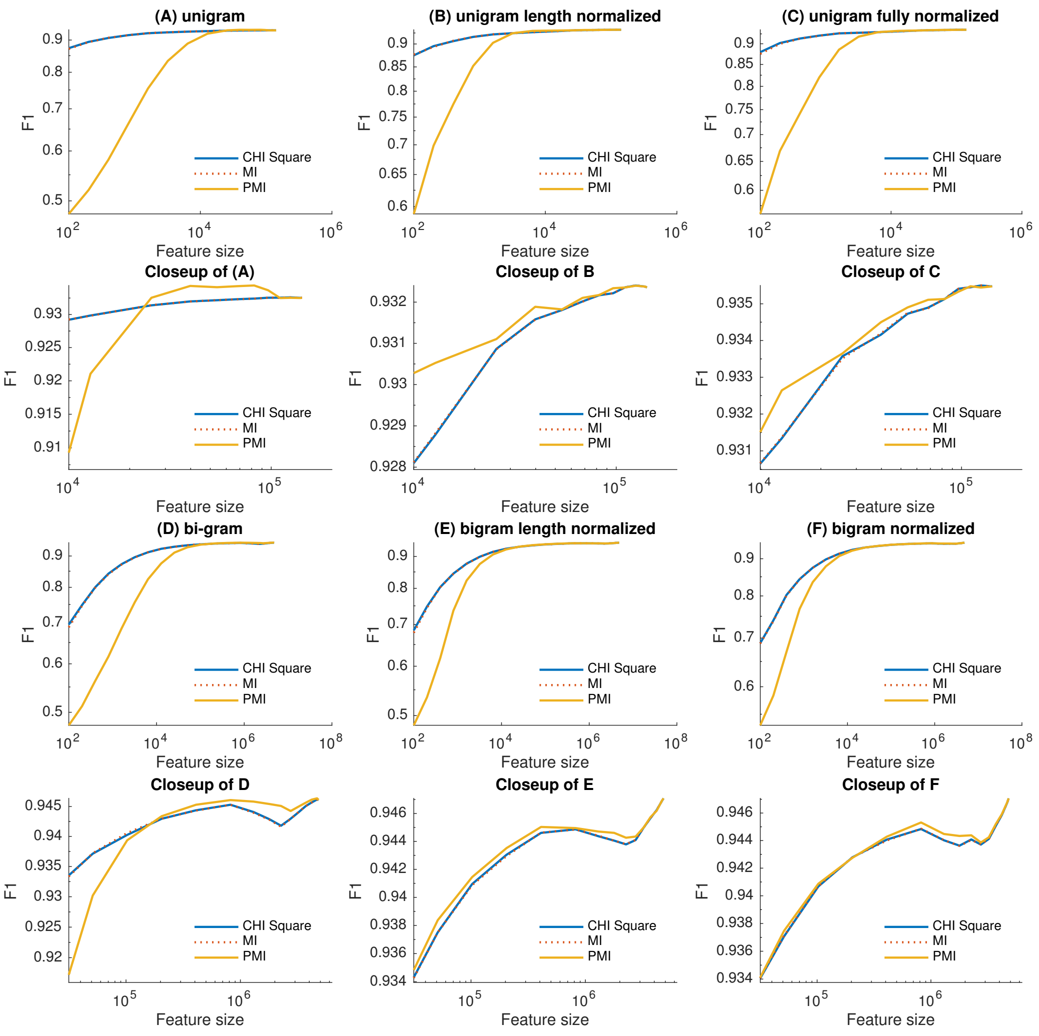


FIGURE 21: Impact of feature size for Uni-gram and Bi-gram models, with combination of text normalization. (A) Uni-gram; (B) Uni-gram length normalized; (C) Uni-gram fully normalized; (D) Bi-gram; (E) Bi-gram length normalized; (F) Bi-gram fully normalized.

frequency as feature weight perform worse than normalized feature weight. Compared with (B) Length normalization and (C) full normalization, we can see that after normalization, PMI increase faster, and before the feature size reaches 10^4, PMI has already surpass MI and χ^2.

Fig. 21 (D) (E) (F) shows the Bi-gram classification results. both PMI, MI and χ^2 reaches the best classification results when using all features. When gradually decreasing features, firstly χ^2 and MI drops lower than PMI, but then PMI drops below χ^2 and MI. We can see from all the eight plots that MI and χ^2 can always perform at almost the totally same F_1 measure levels, while PMI differs greatly with MI and χ^2. In addition, for Bi-gram features, an interesting phenomenon is that from feature size 10^6 to all features (5.2 millions), both the F_1 measure of un-normalized and normalized models first drop then increase greatly to the highest values with all features. The final increasing trend is caused by the increasing of TN (True Negative) value that pull the classification precision higher.

6.3.5 Observation to the Naive Bayes Classification Result

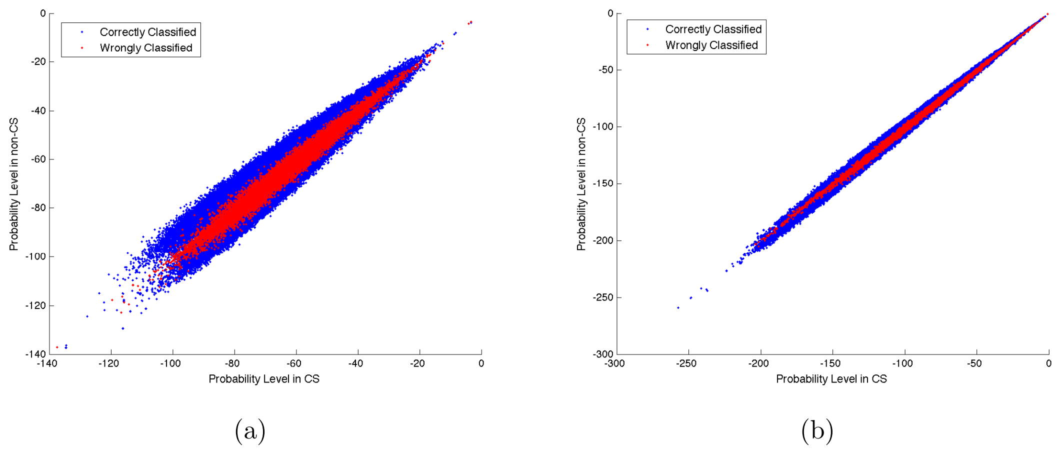


FIGURE 22: Naive Bayes Classification Probability Distributions: Uni-gram (a) and Bi-gram (b)

Fig. 22 shows the Multinomial Naive Bayes classification probability distributions. In Fig. 22 (a) (b), each node represent a paper, the X-axis is the probability score of class CS and the Y-axis is the probability score of class non-CS. If the value in X-axis is larger than the value in Y-axis, the paper will be classified as CS class. The blue nodes in these two plots represent the correctly classified papers, and red nodes represent the wrongly classified papers. We can see that all of the wrongly classified papers are more centralized to the line $y = x$, which means the probability of being in the two classes are quite similar. Accordingly, if a paper is more deviated from $y = x$, it will has smaller chance to be wrongly classified.

6.4 Classifying CiteSeerX papers

In this section, we will use the paper full text in CiteSeerX data set to further test our classifier. The total CiteSeerX data set contains 2.1 million papers, there are around 600,000 CS papers based on the matched result with DBLP. We also make the training data set balanced by sampling 600,000 CS papers and 600,000 non-CS papers. The size of the training set is almost 8 times larger than arXiv data set, and the text content in each CiteSeerX paper is the whole original text, which is also much longer than document length in arXiv (title + abstract) data set. Due to the huge size of CiteSeerX data set, we only use the best proved models to conduct the following experiments. So far our experiments have proved that Multinomial Naive Bayes performs better than Bernoulli Naive Bayes for academic papers, and N-gram (Uni-gram, Bi-gram) models performs better than sentence2vec models when the training set size is large. Therefore, for CiteSeerX, we only use N-gram models and Multinomial Naive Bayes to test the classification accuracy. In all CiteSeerX experiments, we use 3-fold cross validation to evaluate classifiers.

6.4.1 Experimental Settings

The text complexity of papers in CiteSeerX data set is much higher than papers in arXiv data set, first is because CiteSeerX papers contain the every part of the full original text which can include huge amounts of different words / strange words into the vocabulary; second is the cleaning and accuracy level of the full text in CiteSeerX is lower than arXiv because the full text in CiteSeerX data set is automatically parsed from the original PDF files, and due to the complexity and variations of PDF files the parsing may not reliable. Based on the fact, we use the NLTK regular expression tokenizer to tokenize and lower case the text in each paper in CiteSeerX data set by only keeping English letters. The regular expression we used is $\backslash b[A-Za-z]+\backslash b$. As for stop words and stemming, in the arXiv classification experiments, we have already proved that keep stop words will lower

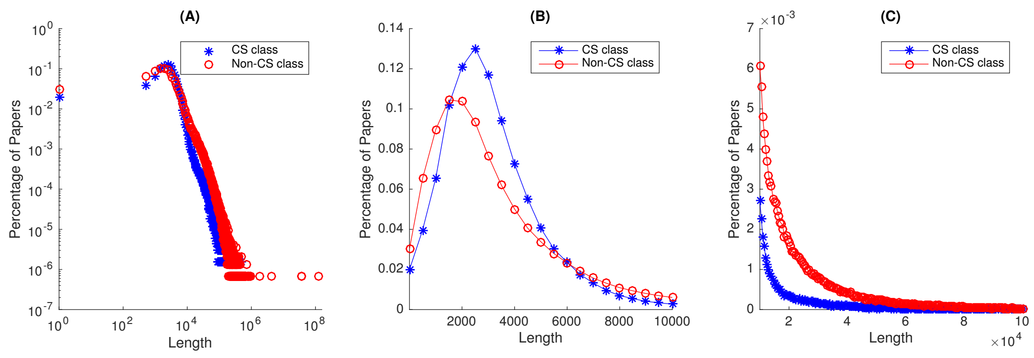


FIGURE 23: Paper length distribution of CiteSeerX data set: (A) total distribution in loglog plot; (B) distribution in lower range; (C) distribution in higher range

the classification accuracy, and stemming didn't cause too much effect to the classification accuracy. Therefore, in this section, we only test with the text model that after stemming and stop words removal, since stemming can lower the vocabulary size that decrease the classification time and space complexity. After the text pre-processing, our CiteSeerX classification experiment will mainly separate into the following two parts:

- Impact of training size: Gradually increase the training set size to compare the classification results (use full features).
- Impact of feature size: Applied different feature selection algorithms on the full data set and select different highly ranked amount of features to compare the classification results.

Each experiment is tested by the Multinomial Naive Bayes classification algorithm with Uni-gram and Bi-gram model. Also, both un-normalized and normalized feature weight are tested. For feature selection algorithms, as we already showed that χ^2 performs quite similar with Mutual Information and overall χ^2 can achieve better performance than Mutual Information. Thus we don't use Mutual Information in this section. However, as a comparison, we still use pointwise mutual information (PMI) to test the impact of feature sizes to the classification results. We use two powerful servers with 24 core CPU and 256GB memory to run the CiteSeerX classification experiments.

Fig 23 shows the length distribution of CiteSeerX data set, the blue curve is for papers in CS class, and the red curve is for papers in non-CS class. The value of length is the number of words in a papers after stop words removal. The count is based on length range

500, which means the lengths between $[500n, 500(n+1)]$ are group and count together. Compared with arXiv data set, CiteSeerX average document length are much larger. There are even papers longer than 10^6, and lengths of different papers varies greatly. We can also see from Fig 16 that non-CS papers tend to be longer than CS papers. Fig 16 (B) shows the length distribution on smaller value ranges. In most of the ranges, the number of CS papers are more than non-CS papers, however, in larger length ranges, non-CS papers are more than CS papers.

6.4.2 Impact of Training Set Size

In this section, we aim to find out the effect of the training set size to the classification results. In the previous arXiv experiments, we conclude that with the increasing of the training set size, the classification results (measured by F_1) indeed increase, but the increasing trend gradually become slower, similar with $y = \log x$, at first the classification results increased very fast, and then gradually slow down. We need to find out whether this increasing trend can also be applied to the CiteSeerX data set.

Deal with huge sizes

We randomly sampled 13 different training sets with different sizes. The training set varies from 1,000 papers up to the full set 1,200,000 papers. The increasing of the training set size accord with exponential growth trend. Each sampled training sets keep the data set balanced. For Uni-gram features, we use all features to do the classification experiments for each of the 13 training sets. However, for Bi-gram features, the size of the vocabulary on the full training set can be so huge that even our servers could not handle such a huge feature set. The average Bi-gram feature set length for a document is averagely 15 times bigger than Uni-gram feature set, which means 15 times more memory will be consumed to process the Bi-gram feature set than Uni-gram feature set. Based on our observation, using Python programming language to deal with the Uni-gram feature set on the full training set (1,200,000 papers) consumes around 100GB memory, which means the corresponding Bi-gram feature set will consume 1500GB memory, which is unfeasible. Therefore, for the Bi-gram all feature set, using all features, we only do classification experiments on the first 10 different training sets that sizes varies from 1,000 to 200,000.

However, alternatively, if not using all features, only use part of the features to construct the Bi-gram feature set, the total vocabulary and the memory consumption could be lowered down. Since in the previous section we proved that χ^2 is very efficient to highly rank popular words and give rare / small words relatively less scores, while PMI can perform even better if feature size is large enough. Hence, we first use χ^2 or PMI to rank all Bi-gram features and only limit the top k ranked features to include into feature set to do classification. By using this method, we can still run Bi-gram features on all the 13 different training sets up to 1,200,000 papers. We use two different k values ($k = 500,000$ and $k = 50,000$) to test the differences of classification results. We use these four models (Uni-gram, Bi-gram, Bi-gram limiting 500,000 features and Bi-gram limiting 50,000 features) to test the impact of training set size to the classification results. Each model is tested with three different feature weighting schemes: un-normalized (original CTF), length normalized, fully normalized.

Experiment Results

Fig. 24 shows the thorough comparisons of all the classification results of all models. Panel (A) shows the comparison of the three different feature weighting schemes for both Uni-gram and Bi-gram models with unlimited features ($k = +\infty$). All the 6 performance curves have the increasing trend, similar with arXiv data set, at first the F_1 value increasing speed is faster, when the training set size gain to a large number, the increasing speed drops. Finally, when reaching the full training set size, the F_1 value also reaches the best for all the models. The three red curves represent the Uni-gram classification results and blue curves show the Bi-gram results. We can see that with the same training set size, Bi-gram performs better than Uni-gram at around 1% F_1 value. Even at last when training set size gain to 1,200,000, the best Uni-gram model didn't out perform the best Bi-gram model at size 200,000. Take a closer look at the three red Uni-gram curves, we can see that at first the full normalized model performs the worst, while at full training set size it outperform all the other two feature weighting models. However, as for Bi-gram, un-normalized Bi-gram model performs the best among all the three Bi-gram models while full normalized model performs the worst, which shows a contrary result compared with Uni-gram models.

Fig. 24 (B) shows the three feature weighting models under Bi-gram with limit of 500K features ($k = 500,000$), and (C) with limit of 50K features ($k = 50,000$). Different with using all Bi-gram features, both the two plots show that length normalization models

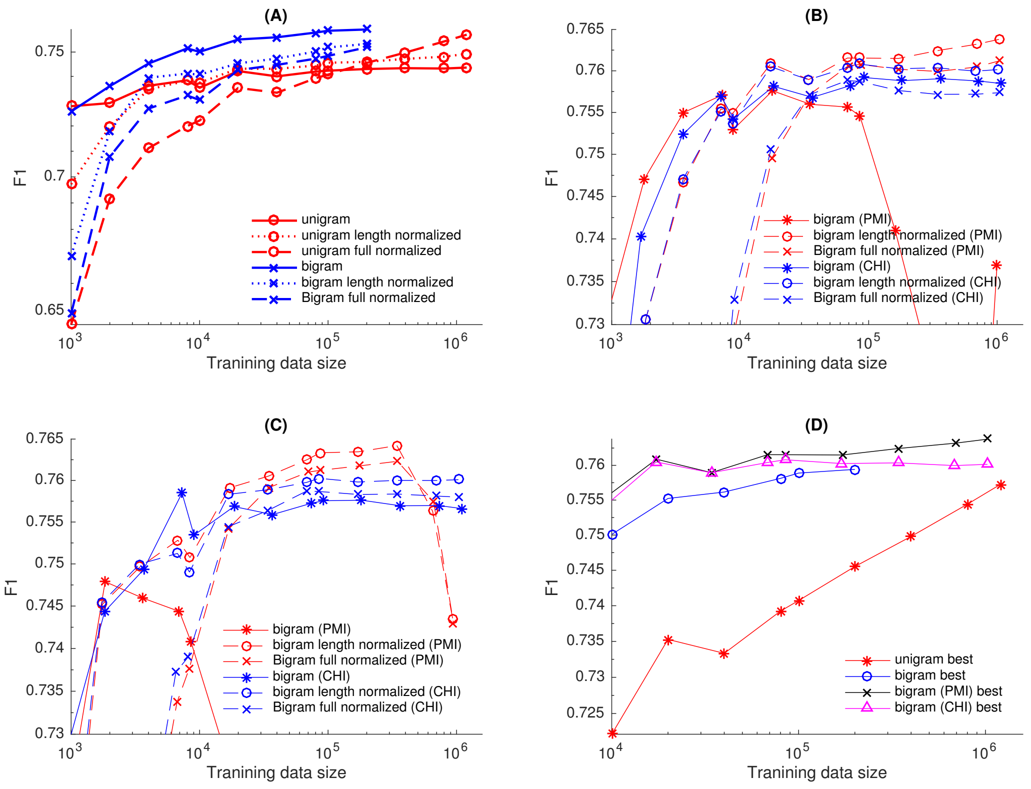


FIGURE 24: Classification Results when increasing training set size. X-axis: training set size, Y-axis: F_1 value. (A) 3 feature weighting schemes under Uni-gram and Bi-gram (Unlimited features); (B) 3 feature weighting schemes under Bi-gram (limit 500K features); (C) 3 feature weighting schemes under Bi-gram (limit 50K features); (D) Overall comparison of Uni-gram and Bi-gram

performs the best, while un-normalized models perform the worst. 500K features model perform better than 50K features model, however all of them still performs better than Bi-gram all features models in (A). The interesting phenomenon is that by limiting the number of features, the length normalization model perform better than full normalization model, this is because rare words in CiteSeerX data set may not reliable due to the text cleanness problem, and TF-iDF give more weights to rare words. We can also see from Fig. 24 (B) (C) that χ^2 performs more stable than PMI, but overall PMI can reach the best F_1 result. Especially for PMI, un-normalized models perform very poor, and the performances of models with $k = 50,000$ drops at last (training size $> 400K$) due to the feature size is not large enough to keep the F_1 values. At last, (D) shows the comparison between all the best Uni-gram and Bi-gram models, we can see that Uni-gram first performs much lower than Bi-gram models, but at last the increasing speed is faster than Bi-gram models. As a conclusion, we summarize the following points:

- Cut lower ranked Bi-gram features can increase the performance, since CiteSeerX data set is not as clean as arXiv, there are mal-formed text and lowered ranked features may contain lots of noisy / mal-formed words.

- After cut lower ranked Bi-gram features, length normalization without TF-iDF significantly performs the best. Due to the length variation of CiteSeerX papers, length normalization may be more powerful applied on CiteSeerX data set.

Impact of Normalization to Precision and Recall

Fig. 25 shows the comparison of Precision, Recall and F_1 value curves with the increasing of training set size and Table 22 shows the statistics of Precision, Recall and F_1 value of using full training set size in detail. (A) plots the Uni-gram models, and we use Full normalized model to compare since it performs the best, and (B) plots the Bi-gram models and we use Length normalized model to compare. We can see from both plots that after normalization, F_1 value becomes slightly better, and the gap between precision and recall greatly reduced. Before normalization, more non-CS papers are classified as CS papers which lowered the precision value, while most of the CS papers has been classified correctly so that recall value is high. After normalization, due to the length balance, less non-CS papers are wrongly classified that pull up the precision value.

		Accuracy	Specificity	Precision	Recall	F_1
Uni-gram ($+\infty$)	Un-NL	0.6983	0.5097	0.6432	0.8818	0.7436
	Len NL	0.7179	0.5837	0.6706	0.8492	0.7493
	Full NL	**0.7377**	**0.6481**	**0.7014**	**0.8235**	**0.7572**
Bi-gram ($+\infty$)	Un-NL	**0.7192**	**0.5425**	**0.6613**	**0.8921**	**0.7594**
	Len NL	0.7352	0.6515	0.7006	0.8152	0.7534
	Full NL	0.7342	0.6530	0.7007	0.8117	0.7520
Bi-gram (500K PMI)	Un-NL	0.7032	0.5750	0.6617	0.8313	0.7369
	Len NL	**0.7342**	**0.6088**	**0.6872**	**0.8596**	**0.7638**
	Full NL	0.7290	0.5944	0.6804	0.8637	0.7612
Bi-gram (50K CHI)	Un-NL	0.7072	0.5040	0.6473	0.9103	0.7566
	Len NL	**0.7280**	**0.5941**	**0.6798**	**0.8618**	**0.7601**
	Full NL	0.7213	0.5696	0.6698	0.8730	0.7580

TABLE 22: Classification Results Comparison of Uni-gram and Bi-gram with size of 1,200,000 papers (the size of Bi-gram with unlimited features is 200,000)

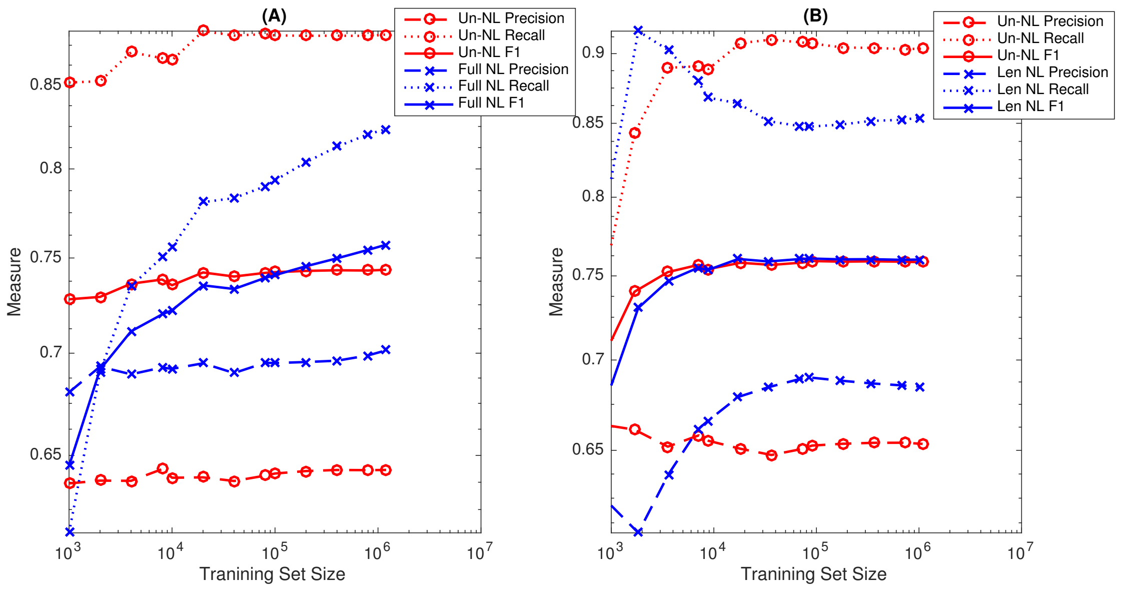


FIGURE 25: Comparison of precison, recall and F_1 before and after normalization; (A) Uni-gram unlimited features; (B) Bi-gram 500K features (Selected by χ^2)

6.4.3 Time Complexity

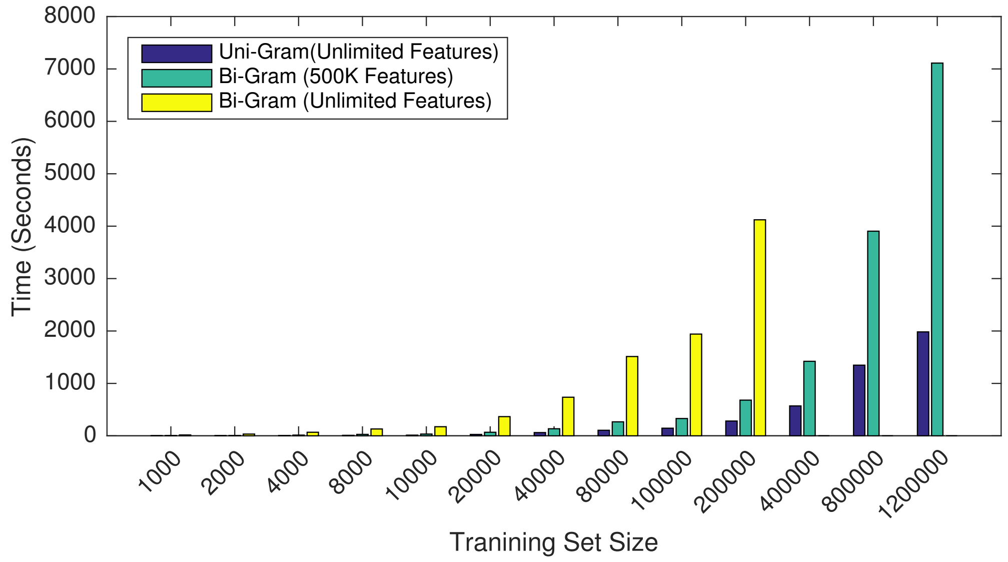


FIGURE 26: Comparison of time consuming of running Multinomial Naive Bayes

We use 3-fold cross validation to do each of the classification evaluation experiment. The classifier running time grows very fast with the increasing of the size of training data set. By using the sparse matrix, the running time complexity can be reduced to $O(nl_{avg})$ instead of $O(nV)$, where n is the training set size, l_{avg} is the average number of features in a document, and V is the size of vocabulary ($l_{avg} \ll V$). From Table 22 we can see that when the training set size grows to 1,200,000, Uni-gram with unlimited features cost 33 minutes to finish a 3-fold cross validation, while Bi-gram with 1 million features limitation takes 1 hour 42 minutes. Since the whole Bi-gram set contains 337 million features, we can't run the Bi-gram with unlimited features with the full set. Retreat back to a smaller data set that contains 200,000 papers, while Bi-gram with unlimited features takes 1 hour 12 minutes to run a 3-fold cross validation, however, Uni-gram with unlimited features only takes 4 minutes and Bi-gram with 1 million features only takes 23 minutes. Fig. 26 also shows the trend of time consumed of running Multinomial Naive Bayes with the increasing of training set size. The X-axis correspond to the 13 different sampled training sets. 10 correspond to size 200,000, after that Bi-gram with unlimited features is not tested, but the running time should greatly exceed 8000 seconds if keep increasing the size for this model.

6.4.4 Impact of feature size

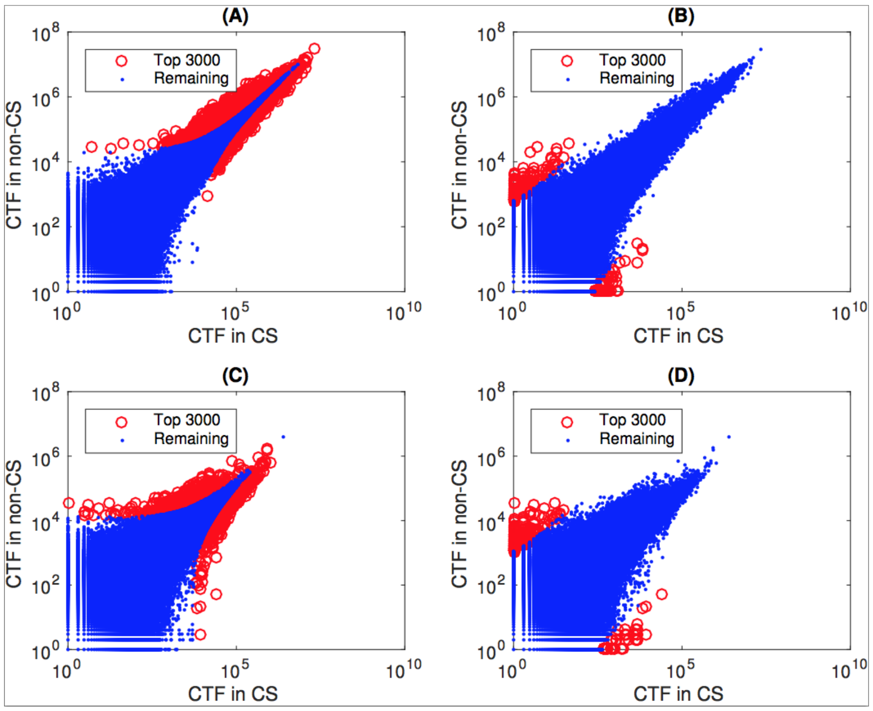


FIGURE 27: Top features selected by different methods. Panel (A) and (C): χ^2; (B) and (D): PMI. (A) and (B) are Uni-gram models; (C) and (D) are Bi-gram models.

After tokenization, stemming and stop words removing, the full CiteSeerX training data set (1,200,000 papers) contains 24,299,156 Uni-gram features and 337,227,855 Bi-gram features. We applied two feature selection algorithms (χ^2 and PMI) onto the Uni-gram and Bi-gram features. Fig. 27 shows the distribution of the 3,000 top ranked features among all the features. The rank is based on the feature selection scores in an descending order. Compared with arXiv data set, the differences of top ranked features between χ^2 and PMI algorithms can be more clearly seen. Most of the top χ^2 ranked features are concentrated at higher CTF ranges, while still deviated from $y = x$. Such words are very popular words that maybe used in many major disciplines. Even so, they still have occurrence differences between CS and non-CS class. On the other hand, PMI still shows the same result compared with arXiv data set, it tend to rank very exclusive features to the top, all the top ranked features under PMI are concentrated surround the axis areas, which means the CTF in other class is almost 0. We can see that the CTF range of CiteSeerX features are much larger than arXiv features. In arXiv data set, the CTF of top occurred Uni-gram features are within the range of 10^5, and Bi-gram features within 10^4. However, as we can see in Fig. 27, top occurred Uni-gram features nearly reach 10^8 CTF value, while top occurred Bi-gram features also exceed 10^6 CTF value. Note that in CiteSeerX, we have used stemming to process the text, and the feature distribution showed in arXiv data set didn't apply with stemming algorithm. Even if stemming is used in CiteSeerX data set, the size of vocabulary and feature CTF are still much larger than arXiv data set. This also shows that much more different words are used in the whole paper full text than in abstracts and titles (arXiv data set). However, we also need to consider the potential mal-formed text in CiteSeerX data set since the data set cleanness and accuracy is lower than arXiv so that lots of mal-formed text can form lots of new features.

Analysis of top ranked features by χ^2

Table 23 shows the top 70 ranked Uni-gram and Bi-gram features by using χ^2 algorithm. All the features shown are after stemming processing. For the features that have higher CTF in CS, we bold and use red color to highlight the text to make them more clearly to be seen. We can see that among the top 70 Uni-gram features, the ratio of CS features and non-CS features tend to be equal, while ratio of Bi-gram CS features tend to be higher than non-CS Bi-gram features. The top ranked CS features are indeed very popular words appears in all categories of papers. For example: "algorithm" as the first ranked CS features, appears 9 million times in all 600,000 CS papers and 5.1 million times in 600,000 non-CS papers. Accordingly, all the top ranked features are popular words that both have high CTF and CDF (class document frequency) in both classes, but the two CTF values still have great differences (deviated from $y = x$). In the last section, we showed that the top ranked non-CS features in arXiv data set are mainly in the categories of Physics and Math. Different from arXiv data set, we can see from Table 23 that top ranked non-CS features are more related to non-science and engineering areas (e.g. country, market, water, school, health care, climate change), even words shouldn't belong to academic research articles. This may caused by the crawling problem of CiteSeerX data set, we observed that some of the papers that labeled as non-CS class are not academic articles, there are all kinds of documents included, e.g. slides, reports, manuals, instructions, etc.. Hence, lots of words can appear in non-CS class.

Uni-gram					Bi-gram				
Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS	Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS
algorithm	9094260	5120571	444082	277696	**comput scienc**	1043131	604085	328879	162906
comput	10761115	7984629	572006	460679	unit state	76886	697975	26070	94561
node	6317054	3754342	230732	139734	**intern confer**	610577	347959	210417	117368
queri	3476848	1549513	144713	82773	**polynomi time**	260751	75995	48536	15944
graph	3752044	2011206	241260	154355	**lower bound**	433741	219407	85053	51332
year	722689	4822896	192839	286256	**run time**	486805	278407	112860	58126
set	11521264	10814739	562450	519026	**logic program**	287874	107359	31336	13738
countri	198580	2429368	32683	112131	**springer verlag**	435485	256937	169434	108063
health	244015	2576215	31924	100276	**ieee transact**	352799	182596	129614	66797
market	505263	3236997	62280	137859	**upper bound**	376180	214822	103592	60997
edg	2666065	1574932	182389	141160	interest rate	15024	285959	2855	23943
tree	3172221	2078762	201431	134222	**relat work**	284483	138690	167827	66825
econom	312848	2626677	64176	152262	**vol pp**	634408	494969	120212	89890
water	253606	2411417	32013	96186	**acm transact**	187461	61345	64989	28405
let	4554051	3670377	361142	277586	per cent	16224	273292	2571	15722
ieee	2316380	1330554	377645	201088	**delta delta**	722427	606267	30721	24508
govern	297315	2451278	79073	165046	**ieee tran**	289045	154412	92532	48264
acm	1603841	708818	316417	133191	**artifici intellig**	308956	174703	98880	53463
bound	2846704	1938533	287175	223065	**worst case**	274909	146195	84054	45030
educ	439476	2718445	67795	133029	exchang rate	9336	221697	1908	16796
proof	3118084	2271453	221100	161838	**nostringv nostringv**	478565	358206	187	164
network	5901436	5464640	302688	254255	**data structur**	406475	282105	98567	60369
nation	513496	2814287	179298	242420	long term	122326	533232	48444	108603
problem	6739211	6625710	518606	463929	**sensor network**	197667	85992	18867	8495
optim	3433226	2734436	340720	265975	**execut time**	256689	142403	55071	28824
path	2872938	2122258	228699	193819	develop countri	15487	222811	3155	27254
percent	203265	1781019	45697	103547	climat chang	10078	203941	1476	13323
chapter	675094	3044936	74237	131440	**machin learn**	238877	127428	65165	31524
school	337192	2155961	112936	161635	**depart comput**	220985	111962	142549	64553
risk	411550	2332124	68195	144514	**experiment result**	238555	130923	110274	53345
semant	2041385	1324548	178485	102372	health care	33261	269695	7327	27925
proc	1529203	829372	257986	164441	**model check**	172883	73588	22163	9869
constraint	2806391	2139714	271022	224386	**program languag**	331271	223460	87785	54624
report	1743627	5338885	376946	382146	**log log**	234130	130715	36046	25632
proceed	2317200	1672717	393273	308621	**proc ieee**	175522	78580	71915	32010
industri	408616	2217651	109427	173154	**proc acm**	124566	39348	49994	16347
sector	142567	1441750	20750	82986	**real time**	626305	560779	116383	96248
firm	176256	1535412	18428	69968	**comput vision**	186945	90966	46571	22918
age	254241	1763229	55347	125959	**proceed intern**	225439	126717	101470	56138
fund	188599	1543674	80679	134906	**note comput**	224998	127694	83501	36461
logic	2402476	1808226	209195	172986	**futur work**	271632	174457	161496	85986
financi	158593	1438733	41819	112859	**ieee comput**	185408	92491	94185	42247
imag	3812753	3446310	182869	182480	world bank	6804	164618	1219	18649
time	11389566	13205693	541248	527490	**page springer**	137411	52841	50500	20645
denot	2187391	1624808	332253	240339	**acm press**	155509	68699	61786	25729
bank	268931	1708424	43571	93597	**shortest path**	163616	76481	29077	14121
lemma	2011014	1444799	124355	77458	**section present**	232154	141492	146898	85101
perform	6375121	6674183	492557	439485	public health	13934	181430	3140	25567
price	596383	2541327	70560	124921	**data mine**	193814	105635	39690	20706
student	778193	2975985	74801	112092	**ad hoc**	273620	185981	61587	49516
public	1053752	3606894	190533	262306	**time algorithm**	114236	40348	45881	16892
approach	4648731	4518732	505074	443954	cross section	39126	248333	11648	48415
product	1748469	5096109	263431	333420	**approxim algorithm**	106433	34981	22722	8297
growth	234214	1585085	64333	140281	**proceed acm**	138574	60718	63949	26878
tax	64890	1045270	10303	49612	**figur show**	720660	709154	230072	183258
fig	2626714	2153889	222303	155208	**lectur note**	273481	191896	101770	61186
cluster	2064673	1540745	134136	114003	work paper	33054	227066	19923	54864
shall	385936	1951996	87851	110921	**softwar engin**	259765	178127	64774	38686
capit	113127	1177587	29315	88122	land use	13128	165124	1905	16385
invest	147319	1281139	33643	98447	per capita	5233	135826	1150	18394
search	2513654	2049858	269084	218926	privat sector	9106	149555	2495	26029
base	7718279	8522483	558590	524728	**algorithm use**	232655	152704	124815	73055
assess	499842	2229179	116637	196081	**databas system**	148638	73278	38150	19412
environment	167309	1329008	40507	99046	**public key**	199293	121194	21106	11968
theorem	2656552	2236051	194712	133941	monetari polici	4191	128448	412	9805
random	2110140	1629056	242804	205268	**int conf**	126470	54959	48432	22190
agenc	139626	1216703	41540	102120	**np hard**	101940	35260	32057	11297
model	11311364	13428799	496710	458774	**hash function**	114478	45406	17952	7774
protocol	2310378	1865816	147846	124408	**object orient**	322480	251511	61772	43683
rule	3589617	3370316	260525	256981	**technic report**	338169	269133	167957	105247

TABLE 23: Top 70 Uni-gram and Bi-gram CiteSeerX Features Selected by χ^2 (features with higher CTF in CS are highlighted with red color)

Uni-gram					Bi-gram				
Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS	Name	CTF in CS	CTF in non-CS	CDF in CS	CDF in non-CS
tikzpictur	0	34282	0	28	clin timeout	0	21740	0	2
pgfpoint	0	8524	0	20	datamonitor plc	1	33820	1	15
moferror	0	7884	0	7	end tikzpictur	0	16585	0	27
zibeta	3071	0	1	0	begin tikzpictur	0	16564	0	27
xmmreg	0	7123	0	21	**omlp mutex**	6545	0	3	0
smjmap	0	7115	0	8	**mpcp suspens**	6530	0	3	0
jmapaq	0	7113	0	5	**mpcp virtual**	6529	0	2	0
sjnaam	0	7082	0	8	**base omlp**	6525	0	1	0
laapaw	0	6558	0	9	**spin mpcp**	6525	0	1	0
xscf	0	5503	0	21	hdb tiff	0	14026	0	9
osfxsr	0	5400	0	49	senso stretto	0	13155	0	16
pgfsi	0	5396	0	19	analysi datamonitor	0	12329	0	6
siread	0	5133	0	19	sptheo timeout	0	11878	0	2
naesb	5	30296	3	108	reflect moferror	0	7675	0	7
ecorefer	0	4993	0	21	campu camperdown	0	7452	0	4
springerni	3	19176	3	52	**na gnd**	2965	0	5	0
svori	0	4408	0	13	milano lombardia	0	7053	0	1
sjcodc	0	4338	0	5	**function preq**	2852	0	1	0
pgfkey	0	4209	0	22	submiss fv	0	6941	0	1
adobeconv	1671	0	1	0	bargain id	0	6766	0	35
pgfusepath	0	4071	0	20	cate vac	0	6765	0	34
emisfact	0	3782	0	2	affirm empl	0	6764	0	33
mwarray	0	3507	0	6	gori acrl	0	6764	0	33
drawingml	0	3362	0	7	sandburg carl	0	6678	0	9
pgfpictur	0	3316	0	24	wsdot bridg	0	6467	0	49
sortedcontinu	0	3177	0	1	tlr level	0	6419	0	36
tcscdi	0	3153	0	17	**brass grundlagen**	2607	0	24	0
iteratei	1252	0	6	0	smjmap issn	0	6286	0	7
infochunk	1230	0	4	0	iac ir	0	6218	0	4
nzerdc	0	2964	0	3	ust tia	0	6180	0	10
smjcat	0	2935	0	7	**datenbanken ii**	2518	0	23	0
sacpa	0	2863	0	14	tangut ideograph	0	6079	0	1
pgfmathresult	0	2861	0	19	plc avail	4	30315	3	36
eurosistema	0	2771	0	35	camperdown darlington	1	11720	1	28
reliabilityfirst	0	2710	0	80	jmapaq issn	0	5831	0	5
objectlinklist	0	2664	0	2	**edf fm**	2358	0	13	0
sjmael	0	2519	0	6	coden smjmap	0	5640	0	6
usetikzlibrari	0	2480	0	25	icap mg	0	5619	0	7
twpe	0	2477	0	13	coden jmapaq	0	5489	0	5
dbenviron	0	2465	0	8	sjnaam issn	0	5476	0	8
myisamchk	0	2437	0	21	fv june	0	5407	0	1
sdchmm	995	0	9	0	review lra	0	5394	0	29
batchact	988	0	3	0	laapaw issn	0	5307	0	8
klmirqd	986	0	2	0	darlington mode	1	10475	1	3
mexcpt	0	2406	0	16	evx sp	0	5205	0	4
osxm	0	2406	0	16	**np idl**	8255	3	7	3
orderedcontinu	0	2403	0	1	chamber thursday	0	4986	0	69
pgfpathlineto	0	2358	0	20	**uniformli wss**	2019	0	3	0
feederindex	0	2346	0	1	coden sjnaam	0	4848	0	8
vnsnap	954	0	7	0	tel coop	0	4848	0	9
sdsrv	0	2328	0	34	springerni com	3	19175	3	51
hrungsgebiet	0	2261	0	11	link springerni	3	19171	3	48
itdseo	0	2239	0	14	spd server	0	4791	0	19
colhdg	0	2238	0	33	arria ii	0	4700	0	36
pjmsettlement	0	2232	0	7	emilia romagn	0	4606	0	2
timequest	1	4445	1	42	juli altera	0	4565	0	22
ldapux	0	2190	0	13	may qnx	0	4544	0	4
algoej	0	2173	0	9	oracl clusterwar	1	8845	1	76
cmisexampl	0	2162	0	5	cumb sem	0	4421	0	1
jacoah	0	2140	0	23	cr prereq	0	4398	0	6
ellrd	0	2122	0	3	pp gpo	0	4389	0	12
tangut	2	6318	2	16	journal hdb	2	13042	2	13
osxmmexcpt	0	2103	0	50	nof api	0	4343	0	8
nccsdo	0	2088	0	23	coden laapaw	0	4273	0	9
rmajett	0	2039	0	7	chapter nof	0	4263	0	5
txlisp	0	2022	0	1	discount nottest	0	4242	0	1
apstag	0	2017	0	5	nottest unit	0	4232	0	1
jpdcer	0	2016	0	17	adv hon	0	4146	0	7
lgia	1	3968	1	31	statu bsi	0	4112	0	43
ombalt	0	1963	0	1	**jx rpc**	1678	0	3	0

TABLE 24: Top 70 Uni-gram and Bi-gram CiteSeerX Features Selected by PMI (features with higher CTF in CS are highlighted with red color)

Analysis of top ranked features by PMI

Now let's turn to the top 70 selected Uni-gram and Bi-gram features by PMI algorithm. Very different from χ^2 algorithm, PMI ranked very exclusive unpopular words into the top position. As shown in Table 24, most of the top ranked PMI selected features are even malformed words due to the PDF parsing problem. And most of the features have very low CDF (class document frequency) which shows that these features only centralized in very few document. For example: the 3rd ranked feature "moferror", only appear in 7 non-CS documents but in total have 3,071 occurrences. It is highly possible that the 7 documents have PDF parsing problem or the documents not belong to academic articles, maybe they are just some web pages or even source codes.

Moreover, there are much more non-CS features than CS features that ranked as top features. This phenomenon illustrates that non-CS documents have more features exclusively and highly occurred. This is reasonable since non-CS class can include any kinds of documents. As a contrary, our CS class is obtained from DBLP data set which only focus on published CS articles so that the exclusively word concentration is not as large as those non-CS documents.

Another observation to the top PMI ranked feature list is that CS features with lower CTF can be ranked into a similar position with a non-CS feature that have higher CTF value. For example: "moferror" has CTF value 7,884 in non-CS class and ranked as 3rd, while "zibeta" has CTF value 3,071 in CS class and ranked as 4th. This is because the total document length in non-CS is larger than total length in CS. Considering the equation to calculate the PMI score, it is calculated by $\left|log_2 \frac{NN_{11}}{N_{1.}N_{.1}}\right| + \left|log_2 \frac{NN_{10}}{N_{1.}N_{.0}}\right|$, where N_{1i} represents the feature's CTF in class i, $N_{.i}$ represents the total document length in class i, and $N_{1.}$ represents the total occurrences of the feature in all documents among both classes. Because $N_{.CS} < N_{.nonCS}$, which means total document length of CS is smaller than non-CS, thus higher N_{1nonCS} ranked into a similar positive with lower N_{1CS}. Actually if we use length normalized feature weight to calculate the feature selection scores, the unbalanced phenomenon will become more severe since all CS features will be ranked to a lower position.

As a conclusion, we can see that PMI feature selection algorithm is not so reliable to apply to CiteSeerX data set, the top ranked exclusive features can contain lots of mal-formed words. Much larger feature size need to be used in order to ensure the classification accuracy. However, we will still examine the performance of PMI algorithm at CiteSeerX data set to find out if PMI can still outperform χ^2 if the amount of features used is large enough.

Uni-gram Experiment Results

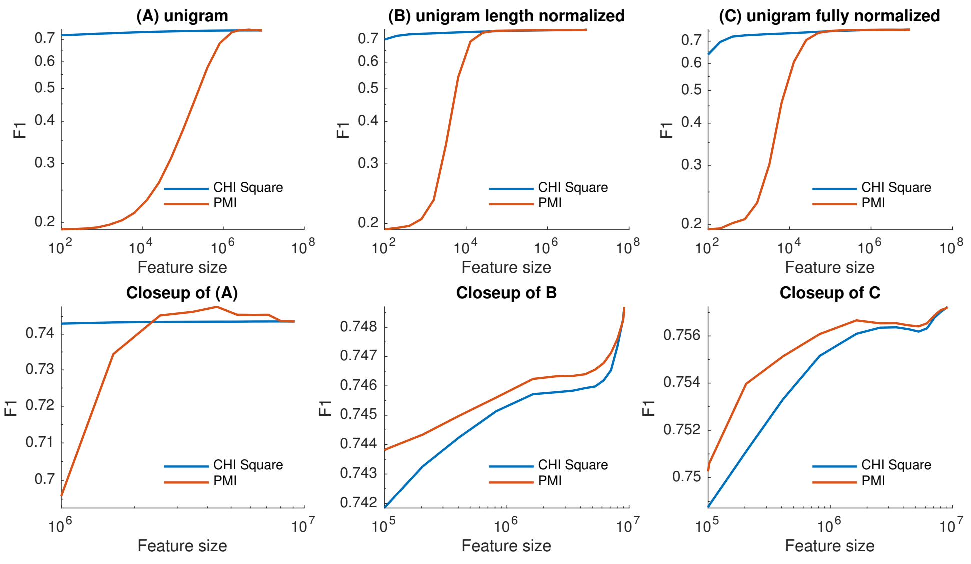


FIGURE 28: Impact of feature size for Uni-gram model, with combination of text normalization. (A) Uni-gram un-normalized; (B) Uni-gram length normalized; (C) Uni-gram fully normalized

Fig. 28 shows the classification results of Uni-gram model when using the three different feature weighting schemes. Each plot shows the comparison of χ^2 and PMI algorithms under the same feature weighting scheme. Similar with the results we obtained from arXiv data set, using un-normalized feature, when the number of selected features is large enough, PMI performs even better than using all features (Fig. 28 (A)). Under un-normalized feature model, the highest F_1 measure value reached by PMI is 0.7477 when using around 4.4 million Uni-gram features, compared with using all features, the F_1 result is only 0.7436 (see from Table 22). However, the same as arXiv data set, after feature weight normalization, PMI can no longer perform better than using all features, but normalized feature weight can let PMI increases faster, with feature size larger than 10^4, PMI can surpass χ^2 for both length normalization and full normalization environment. Moreover, if too few features are used, PMI performs very poor, F_1 measure value even lower than 0.2. The reason is because the unreliable exclusive words appear in the parsed full text. From Table 24 we can see that the top ranked features are more likely to be mal-formed words. Another reason is the much longer document length compared with arXiv data set, more PMI selected features need to be included in order to give the classifier sufficient information to correctly classify documents.

As for the comparison of three feature weighting models, un-normalized feature weight perform the worst, while full normalized feature weight perform the best, even at feature size 50,000 under χ^2 algorithm, the classification F_1 value has already surpass un-normalized feature weight model with full feature size. From all the three plots in Fig. 28 we can see that χ^2 has very stable performance. For un-normalized models, it only fluctuate from 0.72 to 0.743. However, fully normalized model is not as stable as un-normalized model, at first (smallest feature size) it performs the worst, but with the increase of feature size, it quickly surpass all the other two feature weighting models.

Bi-gram Experiment Results

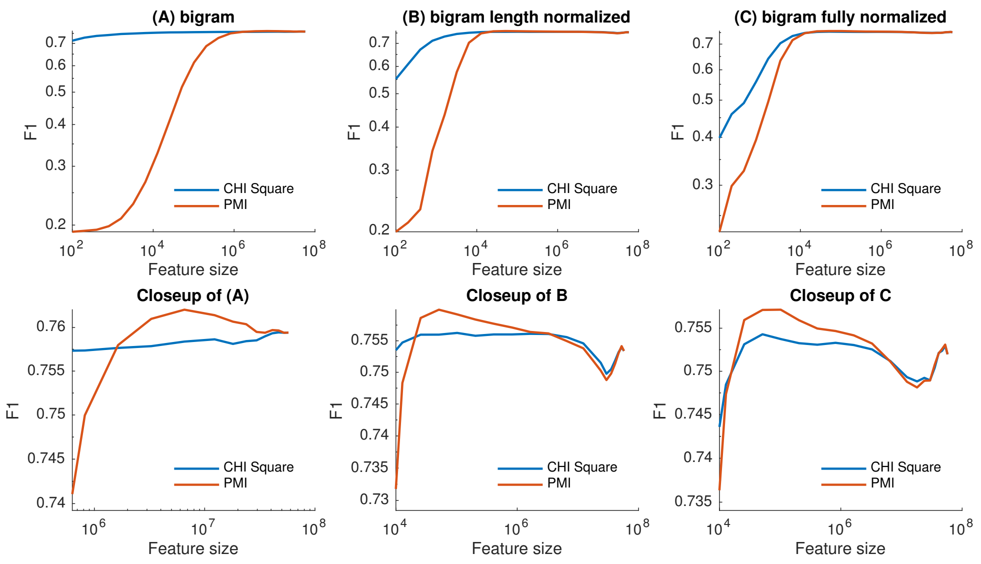


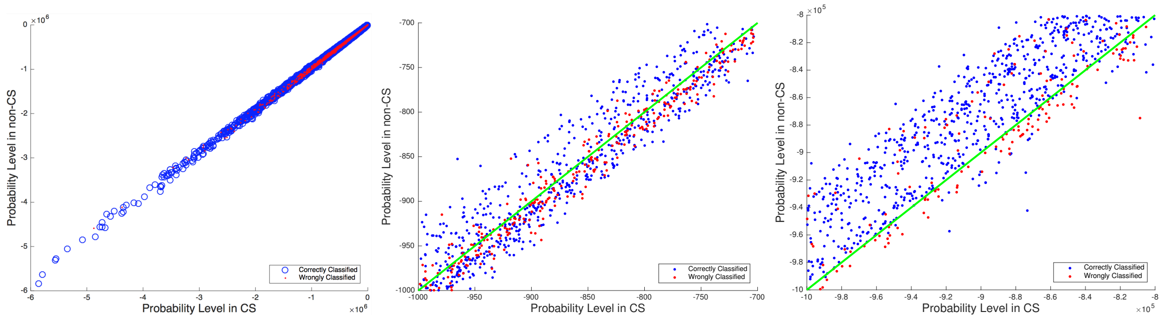
FIGURE 29: Impact of feature size for Bi-gram model, with combination of text normalization. (A) Bi-gram un-normalized; (B) Bi-gram length normalized; (C) Bi-gram fully normalized

Fig. 29 shows the comparisons of classification results of Uni-gram model when using the three different feature weighting schemes. The size of the training set of this experiment is 200,000. Note that Uni-gram feature size experiment we use the full training size: 1,200,000 papers. 29 (A) shows the comparison of χ^2 and PMI under un-normalized model. The same as arXiv data set and Uni-gram CiteSeerX data set, F_1 for χ^2 keep increasing with the increase of feature size, while using all features (around 60 million) the F_1 results reaches the highest value 0.7594. However, PMI reach its best F_1 value when only use 6.5 million top ranked features, the F_1 is 0.7621 even higher than using all features. Also, at first PMI perform very poor, while the number of features is larger than 10^6, it start to perform better than χ^2.

In 29 (B) (C), we can see that the Bi-gram shows a different trend of the changes of F_1 values. In both normalized models, when using more than 50,000 features, the classification F_1 value start to drop, which shows the over fitting phenomenon. Although at last when using all features, both length normalization model and full normalization cannot out perform un-normalized model, but when using around 50,000 features, the F_1 values are much higher than using all features, which can compete with un-normalized model. Note that 1. normalized model can reduce the gap between precision and recall which indicates a better classification result even though the F_1 values are similar; 2. normalized model reaches 0.76 F_1 (by using PMI) with only 50,000 features for which time and space cost for train the classifier is much smaller than using all features. Based on the above two aspects, it can be concluded that normalized models are better than un-normalized model.

Moreover, when the number of features exceeds 10^4, PMI start to perform better than χ^2, which illustrates that normalized model can let PMI have better performance using less features than un-normalized model. Among the two normalized Bi-gram models, the over fitting problem exists. This can illustrated that low text accuracy and data set cleanness problem in CiteSeerX data set can cause Bi-gram model produce much more noisy features (constructed by mal-formed words). Thus we can see that both the two feature selection algorithm performance drops when feature size is too large. It is surprising that PMI can reach the best perform and greatly surpass χ^2 by only using top 50,000 (0.1%) features. As for the comparison between two normalized models, length normalization model perform slightly better than full normalization model, this is because that rare words (especially Bi-gram) is highly possible to be mal-formed words, full normalization that using TF-iDF give such rare words more weights that may lower the classification results.

6.4.5 Observation to the Naive Bayes Classification Result



(a) (b) (c)

FIGURE 30: Naive Bayes Classification Probability Distributions: Uni-gram (a) and closer look on smaller values(b) and bigger values (c)

Fig. 30 shows the Multinomial Naive Bayes classification probability distributions. Each node represent a paper, the X-axis is the probability score of class CS and the Y-axis is the probability score of class non-CS. If a node locate in the southeast side of $y = x$, it will be classified as CS, otherwise non-CS. Blue nodes in the plot represent the correctly classified papers, and red nodes represent the wrongly classified papers. Compared with arXiv classification result shown in Fig. 22, the conclusion remains the same: wrongly classified papers are more centralized around $y = x$, which means they have more similar probability scores between the two classes. However, compare with arXiv data set in Fig. 22 again, we can see from Fig. 30 (a) that all the nodes are more attached to $y = x$, and the probability value range is much larger than arXiv range. This is mainly because every document in CiteSeerX data set contains the whole paper text, the length is much longer than abstract and titles included in arXiv data set, adding the probability of all the words appear in whole paper text cause the much larger probability range.Because the range is large and due to the scaling of the plot, the nodes look more attached to $y = x$. If we take a closer look at Fig. 30 (b) (c), we can see that nodes are still deviated from $y = x$ and correctly classified papers tend to be more deviated. Fig. 30 (b) shows the closer look at smaller probability values, we can see that there are more CS papers appeared than non-CS papers; Fig. 30 (c) shows the larger probability value ranges, more non-CS papers appear. This phenomenon further illustrates that more non-CS papers have long document length than CS papers.

Moreover, from Table 25 we can see that top mis-classified non-CS papers are highly

	Paper Title
1	Static RAM with Multiple Sensing Schemes For Mentor 1997 Student VLSI Design Contest In Digital Design Category and Novice Class
2	Document for the Real-Time Message Passing Interface
3	Handbook of Constraint Programming
4	Handbook of Applied Cryptography
5	Interim Reports on work of the International Institute for Applied Systems Analysis
6	PPFS: A High Performance Portable Parallel File System
7	Lecture Notes in Computer Science 3503 Commenced Publication in 1973
8	Final report of European project: number IST-1999-12324
9	Advanced Topics in Computer Systems CS262
10	Space-time coding techniques with bit-interleaved coded modulations
11	Concepts, Techniques, and Models of Computer Programming
12	Fundamentals of Computer Design
13	Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles
14	Planning Algorithms
15	On Protocol Security in the Cryptographic Model

TABLE 25: Example of top mis-classified non-CS papers

similar with CS papers. Actually lots of them are not academic papers, those non-CS instances include report, review, books, even CS course materials. Due to the labeling problem, part of the labeled non-CS papers can actually be CS papers. We treat intersection of CiteSeerX and DBLP to be the CS papers. However, DBLP only includes CS papers published on high quality CS publications, it is possible that some other CS papers in CiteSeerX data set haven't been included in the interestion set. This is the reason why more labeled non-CS papers were mis-classified than labeled CS papers (False Positive higher than False Negative which cause the precision lower than recall). Refer to arXiv data set, for which labels have already been provided, such problems don't exist. Precision and recall are both in similar high values.

CHAPTER 7

Discussions and Conclusions

In this thesis, we used big scholarly data set with size more than millions documents to answer the question whether Computer Science academic papers can be classified out from papers in all the other research fields. Two data sets are used: CiteSeerX and arXiv. We use DBLP data set to merge with CiteSeerX in order to label CiteSeerX set into CS and non-CS. Based on the difficulties of classifying academic papers, we implemented a comprehensive classification system to conduct lots of experiments for comparison and evaluation:

1. train and test classifier with two algorithms: Naive Bayes (Bernoulli, Multinomial), Logistic Regression.
2. train classifiers with lots of different sizes of training sets, from 1,000 docs up to millions of docs.
3. use three models: Uni-gram, Bi-gram, sentence2vec.
4. use three feature selection algorithms to lower down feature dimensionality: χ^2, PMI, MI.
5. Use two feature weight normalization schemes to deal with length variations: length normalization and full normalization (Len Normalization + TF-iDF).
6. experiment the impact of stop words and stemming to the classification results.

When experimenting with arXiv data set, we find that most classification methods can reach an F_1 value as high as 0.9. The best method is Multinomial Naive Bayes on Bi-gram language model, which obtained an F_1 value close to 0.95. However, on sentence2vec representation, neither logistic regression nor Naive Bayes can compete with Bi-gram model. Other classification methods, including SVM, are also tested on smaller data sets because of the scalability issue of these algorithms. SVM performs similar with Logistic Regression on sentence2vec representation. For multinomial Naive Bayes text classification, it was long believed that PMI is not a good candidate for feature selection. On the contrary to this believe, we show that PMI is better than χ^2 and MI. This is probably because of the size of our training data is bigger– in Fig. 21 we can see that PMI is inferior until the feature size exceeds 10^4. We also shows that stop word removing improves the performance for all the methods, including bag of words model, Bi-gram model, and various classification methods on distributional vector representation of documents. On the other hand, stemming has limited impact on the performance. As for feature weight normalization, full normalization improve the classification accuracy a little bit compared with other schemes, but overall, feature weight normalization can't bring too much improvement, this is because the lengths of abstract + title are very similar (can be seen from Fig. 16).

However, classification results on CiteSeerX data set are not as good as arXiv data set, the best F_1 value we obtained is around 0.76 when using Bi-gram with reduced normalized features. This is partly because of the data set labeling problem and the data set text cleanness problem. Firstly, for the labeling problem, papers not within CiteSeerX and DBLP interestion may still be CS papers. In Table 25 we also show that based on the MNB classification results of labeled non-CS papers, those papers which CS probability highly exceed non-CS probability could actually be CS papers. Thus, it explains the phenomenon that more labeled non-CS papers have been classified as CS papers that lower the precision value. However, we surprised to find out that by using feature weight normalization process, the gap between precision and recall can be lower down and the F_1 also improved. Secondly, for the text cleanness problem, we can see from Table 24 that data set contains lots of mal-formed words due to the PDF parsing problem, which also caused the incredibly high feature dimensionality, especially for Bi-gram (more than 330 million features). Thus, under Bi-gram model, in Fig. 24 (A) we can see that using all features the Bi-gram classification results are not the best, and feature normalization cannot improve the results too. But if we select the top k ranked Bi-gram features, the F_1 can be improved and length normalization can even further boost the F_1 value higher (Fig. 24 (B) (C)). The phenomenon of length normalization performs better than full normalization also infer that TF-iDF weighting may not appropriate for CiteSeerX data set since rare words can be mal-formed words that confused the classifier. If rare words are given too much weights, the classification results could be affected.

In addition, we also tried to classify papers in narrow areas, such as papers in conferences VLDB, SIGMOD, and ICSE, each class trained on two thousand of papers. We also observed high accuracy in these experiments. Among VLDB and ICSE, the F_1 is above 0.98 because these two conferences focus on very different topics, one in database, the other in software engineering. What is surprising is that among VLDB and SIGMOD, which are both database conferences, the F_1 value is also above 0.88.

Compared with most of the previous works, it is surprising to see that Computer Science academic papers can be classified with high accuracy (nearly 95%) based on content only. With such high accuracy, we can envision numerous applications in the pipeline. We are building an academic search engine in the area of computer science. When crawling the data from the Web and online social networks, we can judge whether a document is a computer science paper; when conducting author disambiguation, we can determine whether a paper is written by a certain person or a group of researchers or a community of academics; when recommending papers, we can classify the paper according to a researcher's profile.

REFERENCES

[1] Aggarwal, C. C. and Zhai, C. (2012). A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer.

[2] Bird, S. (2006). Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics.

[3] Caragea, C., Silvescu, A., Kataria, S., Caragea, D., and Mitra, P. (2011). Classifying scientific publications using abstract features. American Association for Artificial Intelligence.

[4] Caragea, C., Wu, J., Ciobanu, A., Williams, K., Fernández-Ramírez, J., Chen, H.-H., Wu, Z., and Giles, L. (2014a). Citeseer x: A scholarly big dataset. In *Advances in Information Retrieval*, pages 311–322. Springer.

[5] Caragea, C., Wu, J., Williams, K., Gollapalli, S. D., Khabsa, M., Teregowda, P., and Giles, C. L. (2014b). Automatic identification of research articles from crawled documents. In *WSDM 2014 Workshop on Web-scale Classification: Classifying Big Data from the Web*.

[6] Cavnar, W. B., Trenkle, J. M., et al. (1994). N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175.

[7] Craven, M., Kumlien, J., et al. (1999). Constructing biological knowledge bases by extracting information from text sources. In *ISMB*, volume 1999, pages 77–86.

[8] Django (2013). Django software foundation. In *V1.5*. Lawrence, Kansas.

[9] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130.

[10] Giles, C. L., Bollacker, K. D., and Lawrence, S. (1998). Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98. ACM.

[11] Hatcher, E., Gospodnetic, O., and McCandless, M. (2004). Lucene in action.

[12] Hosmer, D. W., Jovanovic, B., and Lemeshow, S. (1989). Best subsets logistic regression. *Biometrics*, pages 1265–1270.

[13] Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features.* Springer.

[14] Kodakateri Pudhiyaveetil, A., Gauch, S., Luong, H., and Eno, J. (2009). Conceptual recommender system for citeseerx. In *Proceedings of the third ACM conference on Recommender systems*, pages 241–244. ACM.

[15] Lawrence, S., Giles, L. C., and Bollacker, K. (1999). Digital libraries and autonomous citation indexing. *Computer*, 32(6):67–71.

[16] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*.

[17] Ley, M. (2009). Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500.

[18] Li, H., Councill, I. G., Bolelli, L., Zhou, D., Song, Y., Lee, W.-C., Sivasubramaniam, A., and Giles, C. L. (2006). Citeseer χ: a scalable autonomous scientific digital library. In *Proceedings of the 1st international conference on Scalable information systems*, page 18. ACM.

[19] Lu, Q. and Getoor, L. (2003). Link-based classification. In *ICML*, volume 3, pages 496–503.

[20] Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

[21] McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

[22] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[23] Orduña-Malea, E., Ayllón, J. M., Martín-Martín, A., and López-Cózar, E. D. (2014). About the size of google scholar: playing the numbers. *arXiv preprint arXiv:1407.6239*.

[24] Ororbia II, A. G., Wu, J., Khabsa, M., WIlliams, K., and Giles, C. L. (2015). Big scholarly data in citeseerx: Information extraction from the web. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 597–602. International World Wide Web Conferences Steering Committee.

[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830.

[26] Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

[27] Rennie, J. D., Shih, L., Teevan, J., Karger, D. R., et al. (2003). Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, volume 3, pages 616–623. Washington DC).

[28] Rogati, M. and Yang, Y. (2002). High-performing feature selection for text classification. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 659–661. ACM.

[29] Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-j. P., and Wang, K. (2015). An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 243–246. International World Wide Web Conferences Steering Committee.

[30] Tang, B., Kay, S., and He, H. (2016). Toward optimal feature selection in naive bayes for text categorization. *arXiv preprint arXiv:1602.02850*.

[31] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM.

[32] Warner, S. (2005). The arxiv: Fourteen years of open access scientific communication. In *Free Culture and the Digital Library Symposium Proceedings 2005*, page 56.

[33] Wu, J., Williams, K., Chen, H.-H., Khabsa, M., Caragea, C., Ororbia, A., Jordan, D., and Giles, C. L. (2014). Citeseerx: Ai in a digital library search engine. In *The Twenty-Sixth Annual Conference on Innovative Applications of Artificial Intelligence, IAAI*, volume 14.

[34] Xu, Y., Jones, G. J., Li, J., Wang, B., and Sun, C. (2007). A study on mutual information-based feature selection for text categorization. *Journal of Computational Information Systems*, 3(3):1007–1012.

[35] Yang, Y. and Chute, C. G. (1994). An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems (TOIS)*, 12(3):252–277.

[36] Yang, Y. and Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM.

[37] Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *ICML*, volume 97, pages 412–420.

VITA AUCTORIS

NAME:	Tong Zhou
PLACE OF BIRTH:	Zhengzhou, Henan province, China
YEAR OF BIRTH:	1990
EDUCATION:	Beijing University of Posts and Telecommunications, B.Eng., Computer Science and Technology, Beijing, China, 2011
	Beijing University of Posts and Telecommunications, M.Sc., Computer Science and Technology, Beijing, China, 2014
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2016