TYPIYFING WIKIPEDIA ARTICLES

by

QUAZI MAINUL HASAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

DECEMBER 2010

ACKNOWLEDGEMENTS

It is always a pleasure to thank those people without whom it would have been impossible to complete my thesis. First and foremost, I would like to express my deepest gratitude to my thesis supervisor Dr. Chengkai Li for supporting me and guiding me throughout my research work. Without his constant help and advice it wouldn't be possible for me to finish my thesis. I would also like to thank my other committee members Dr. Matthew Wright and Dr. Chris Ding for providing me their valuable ideas and suggestions and taking time to be in my committee.

I owe my deepest gratitude to my parents, Quazi Sulaiman Ahmad and Kamrun Nessa Khanam for their constant inspiration, care and support throughout my whole life. I would also like to thank my loving wife, Nabila Rahman for supporting me in every steps of my life as being a best friend of mine. Moreover, I wish to thank my siblings Quazi Manjurul Hasan, Quazi Mushfiqul Hasan and Quazi Farzana Shahnaz for their inspiration and support.

I would like to express my gratitude to the Department of Student Enrollment Services, University of Texas at Arlington for funding me, providing all time support to me during my study and giving me the opportunity to work with the cutting edge technologies. I also extend my thanks to Dennis O'Hare and Robert Gunnip for their immense assistance.

I greatly appreciate the help of my friends here in Arlington who has always been providing me their helping hands whenever I need it. I would like to thank my friend Late A.S.M Shahidur Rahman and my dearest friends for inspiring me during my school and college life.

On top of all, I would like to say my humble gratitude to Allah for all the blessings I have had so far in my life.

November 24, 2010

ABSTRACT

TYPIFYING WIKIPEDIA ARTICLES

Quazi Mainul Hasan, M.S.

The University of Texas at Arlington, 2010

Supervising Professor: Chengkai Li

In Wikipedia, each article represents an entity. Entity can have different types like person, country, school, science etc. Although Wikipedia encapsulates category information for each page, sometimes it is not sufficient to deduce the type of a page just from its categories. But, incorporating the clear type information in a Wikipedia page is very important for the users, as it will help them to explore the pages in more organized way. Hence, in my thesis, we explore different standard classification techniques, mainly Naïve Bayes and Support Vector Machines and experiment how these techniques can be made more effective for typifying Wikipedia articles by using different feature selection methods. We proposed a method where Wikipedia categories are used as features. Moreover, we combine both words and Wikipedia categories as features in the feature vector, which improves the classification accuracy and out performs the other standard methods as well. To compare our methods we calculate the accuracy of different methods and used well known data mining tool “WEKA”.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Wikipedia is an instance of collective intelligence, which is based on an open editable model and written collaboratively by large number of Internet users. Wikipedia is now being considered as one of the fastest growing collections of articles. The current collection of Wikipedia has around 3,478,144 articles and 22,223,054 pages in total in addition to 13,433,763 registered users [1]. Hence, as the size of Wikipedia knowledge base is growing rapidly; organization of this vast number of articles has become one of the key issues to consider. Anyone who have surfed the Wikipedia knowledge base has happily experienced the feeling of being lost switching from one interesting article to another and discovered information that is totally unknown to them before. However, exploring the Wikipedia articles based on their types would be much easier. It will allow a user to focus on a particular type as well as the pages of that type during exploration.

In Wikipedia, each article represents an entity. Entities can have different types like person, country, school, science etc. Incorporating clear type information (Typifying) can play an important role in exploring Wikipedia articles. Typifying Wikipedia articles will arrange the articles in a more organized manner. Hence in this thesis we tried to use text classification to give Wikipedia articles specific type information based on Wikipedia article text and its category information.

1.1 Motivation

Although Wikipedia encapsulates category information for each page, sometimes it is not enough to deduce the type of a page just from its categories. Some Wikipedia articles have a very small list of categories and some have a very large set of categories. For some Wikipedia articles the categories are irrelevant and are not sufficient to provide clean type for those

articles. Moreover, for lengthy articles in Wikipedia there can be hundreds of categories which are not particularly relevant and this will cause category clutter, also known as over-categorization [2]. However, the type of a Wikipedia article is also dependent on the text of an article. So typing Wikipedia articles based on its text would be more reasonable. On the other hand, adding types manually to each Wikipedia article is very tedious, as the size of the corpus is extremely large. Therefore, an automatic typifying system is necessary because it will save a lot of time and manual labor.

1.2 Challenges

Classifying a Wikipedia article based on this text information and categories imposes several challenges. The first problem is to choose appropriate features from the Wikipedia articles in order to achieve satisfactory classification accuracy. Some approaches consider Wikipedia category information per page as features. But, the whole Wikipedia category hierarchy is not organized and is barely useful [3]. For example, “Elvis Presley” is in the category “Grammy Awards” which is a super category of “Grammy Award winners”, but “Elvis Presley” is a Grammy Award winner not a Grammy Award. Hence it would be more reasonable if “Elvis Presley” has a category “Grammy Award winners” instead of “Grammy Awards” as category. Another approach is to use individual text or word of an article as features for deriving the type information. However, it may be possible to choose a subset of words from the whole collection of words extracted from Wikipedia articles, which are more relevant as the features. But it is still difficult to automatically identify the important or related words per document. Therefore, in our approach, we have applied classification based on the text content of a Wikipedia article and considered subset of the entire word or entity set as our feature vector. We have also used Wikipedia categories up to Level 2 as our features. In both the case, we have incorporated the frequency of occurrences for each word, entity or category as the weight of that feature.

Several methods have been discussed in literature for document or text classification. Among those, the most famous methods are naïve Bayes classifier, decision-trees, nearest neighbor classifier, neural networks and Support Vector Machine (SVM). Hence, in this thesis, we have applied both naïve Bayes and SVM classification methods in order to typify the Wikipedia articles. In our approach, we have used the available type information from DBpedia, which is our training set. Since the accuracy of DBpedia ontology is 95% [4] we used the DBpedia dataset to train our classifier. But, as DBpedia database does not contain Wikipedia page contents, we have extracted the required Wikipedia articles from Wikipedia. Another challenging task is to build a new method, which improves the classification accuracy and outperforms the classification accuracy of other methods that we used in our thesis for typifying Wikipedia articles. We combine both words and Wikipedia categories as features in the feature vector, which improves the classification accuracy and out performs the other methods as well.

1.3 Contribution of Thesis

In this thesis, we have used naïve Bayes and Support Vector Machine classification algorithm to classify Wikipedia articles. To construct the training data for the classifier we used different features:

- Words
- Entities
- Wikipedia categories
- Entity types

Moreover, we have also used different variations of these features. In our thesis, we work with 100 class labels, which is a very complex classification problem. We have got an accuracy of 84.4% where we have considered the local frequent words from each article including their frequencies. We have also got 83.6% of accuracy by using Wikipedia categories as features up to level-2. We have improved the classification accuracy by combining 2 different features, one

is Wikipedia text and the other one is Wikipedia categories, which perform better than the individual base classifiers. By using SVM, we have got an accuracy of 91%.

1.4 Organization of Thesis

The thesis is organized as follows. Chapter 2 gives a detailed background on text/document classification, different classification algorithms, DBpedia ontology and related work. Chapter 3 explains several standard techniques and our proposed methods for classifying Wikipedia articles in purpose of giving it a clear type information. Chapter 4 describes technical details of our implementation and preprocessing of our dataset from collected from different sources. It also describes parameter tuning for polynomial kernel. Chapter 5 explains our simulation results and compares results of using different features and algorithms. Based on the results in Chapter 5 conclusion is drawn and future scopes are discussed in Chapter 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we provide a general overview of text classification, naïve Bayes classifier, and Support Vector Machine with different kernel methods and DBpedia ontology.

2.1 Document Classification

In document classification, we are given a description ***d*** Є **X** of a document, where **X** is the document space; and a fixed set of classes **C** = {c_1, $c_{2, \ldots,}$ c_n}. Classes can be called labels or categories. We are also given a set of training data, **D** of labeled documents <***d***, c>, where <***d***,c> Є (**X** x **C**).

We wish to learn a classifier **α** by using a learning algorithm that maps documents to classes [11]:

$$\alpha: X \rightarrow C$$

Once the **α** is learned this can be applied to the test set to predict the most appropriate class for each document in the test set.

2.2 Naïve Bayes Classifier

Naïve Bayes classifier is a probabilistic classifier. The probability of a document ***d*** being in class **c** is computed as follows [12]:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d}^{n} P(t_k|c)$$

where, $P(t_k|c)$ = conditional probability of term t_k occurring in a document of class **c**.

$P(c)$ = prior probability of a document occurring in class c.

n_d = No. of tokens in the document *d*

$< t_1, t_2, \ldots t_k >$ are the tokens in document *d*

The goal is to find the best class for the document. The best class in Naïve Bayes classification is the most likely or maximum a posterior (MAP) class c_{map}:

$$c_{map} = \arg max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

P is written as $\hat{P}$ since this value is estimated from the training data.

2.3 Support Vector Machine

"Support Vector Machines are a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis" [12] [13]. SVM is a non-probabilistic classification algorithm. For a given set of input data it tries to predict the class of each input data from two possible classes. An SVM algorithm first trains its classification model by using a set of training examples and then predicts the class of a new example. More intuitively, an SVM presents the examples as points in n-dimensional space and mapped in such a way that examples of different class labels can be separated by a maximum clear gap. Similarly new examples are mapped to the same space and then the classification model predicts on which side of the gap or margin it belongs to.

In general, the larger the margin the lower is the generalization error. In figure 2.1, a simple example of linearly separable data points are shown. The distance between the thin lines is called margin. Also the points on the 2 hyper planes that define the margin are the support vectors. SVM tries to find a 1-dimensional hyper plane that separates the input examples base on their class labels. However, there can be infinite number of lines. SVM finds a line that is oriented so that the margin between the support vectors is maximized.

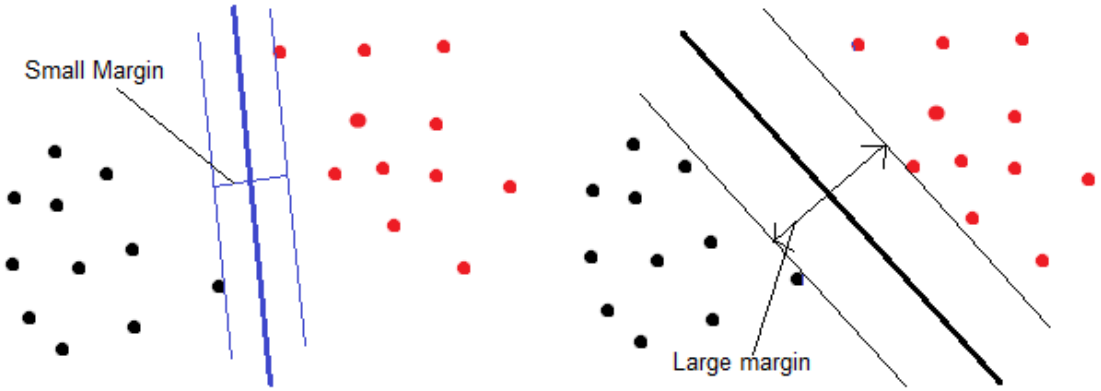


Figure 2.1: Hyper plane in SVM with different margin [25]

Also, there are cases where the data points are not linearly separable. In that case we will need a nonlinear line that separates the data points. Instead of finding a nonlinear line that separates out the data, SVM maps the data points to a high dimensional space by using Kernel functions where a hyper plane can be used to separate the data. A simple example is shown in figure 2.2.

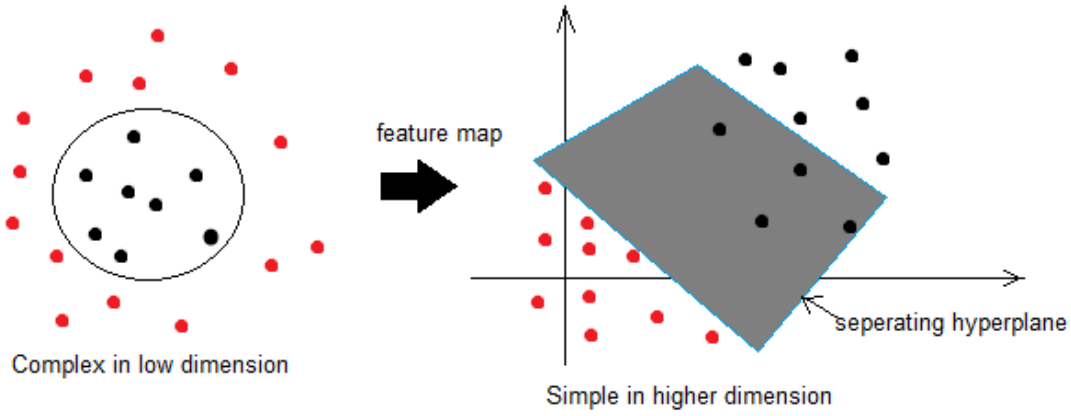


Figure 2.2: Mapping of nonlinearly separable data points using Kernel function [25]

Several Kernel functions are proposed in the field of SVM. Among them the most important kernels are:

- Polynomial Kernel

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$$

- Radial basis function (RBF)

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \gamma > 0$$

- Sigmod

$$K(x_i, x_j) = \tanh\left(\gamma x_i^T x_j + r\right)$$

where, γ, and d are kernel parameters.

The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter r.

2.4 DBpedia Ontology

"The DBpedia Ontology is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. The ontology currently covers over 259 classes which form a subsumption hierarchy and are described by 1,200 different properties" [15]. The DBpedia ontology currently contains about 1,478,000 instances. Table 2.1 below lists the number of instances for several classes within the ontology:

Table 2.1: No. of Instances per Class in DBpedia Ontology

Class	Instances
Place	413,000
Person	312,000
Work	320,000
Species	146,000
Organization	140,000
Building	33,000

A part of the DBpedia Ontology is shown in figure 2.3 for class "Animal", the classes under the "Animal" are the leaves. The root of the DBpedia ontology is "Thing".

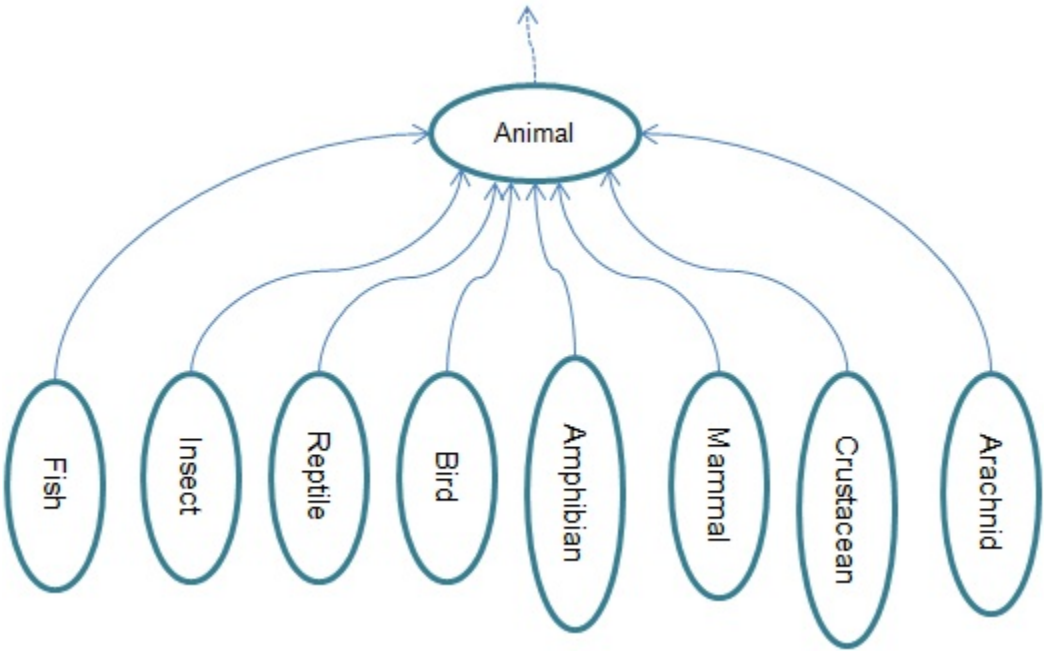


Figure 2.3: Partial DBpedia Ontology for type “Animal”

2.5 Related Work

The increasing numbers of text documents over the World Wide Web are giving rise to several interesting applications to search for information over these pages. In order to make the process of document or information search to be more automatic, document classification techniques based on the text contents are being applied extensively by many applications. As a result, a lot of document classification approaches have already been studied and experimented over the whole decades. In paper [17], the authors have studied four different classification approaches such as naïve Bayes, nearest neighbor, decision trees, subspace etc and applied them to categorize the articles from Yahoo news into seven classes. They have also tried some combinatorial approaches; But finally found, those cannot always supersede the accuracy of the best individual classifier. Besides this, a large number of linear classification methods such as linear least squares fit (LLSF), logistic regression and support vector machine (SVM) are also described in literature [18] in order to perform text-based categorization.

Existing works have also expanded the traditional Bag of Words (BOW) model by including semantic relations from an automatically constructed thesaurus of concepts based on Wikipedia knowledge [19]. Their result for varying data set confirms significant improvements over the baseline algorithms. However, as the text categorization involves a large scale of computational effort and time, centroid based classification approaches are also applied in order to improve the efficiency. Guan et al. [20] describes a faster centroid based classifier named Class-Feature-Centroid (CFC), where a centroid is determined after considering both inter-class and inner class terms distribution. CFC has been applied to Reuters as well as to some newsgroup email collections and the result shows that it outperforms the state-of-the-art SVM classifier in case of sparse dataset. Existing works also prove that class dependant feature scaling with recursive feature elimination approach improves the performance of standard naïve Bayes document classification significantly [21]. Kaptein and Kamps (2008) use link information to improve the classification accuracy [23]. They showed that utilizing link structure between the Wikipedia articles in classification can cause marginal improvement in classification accuracy. Ringland et al (2006) incorporated structural features of Wikipedia and developed a German corpus which classifies Wikipedia articles into Named Entity categories [24]. Saleh et al (2010) have proposed a method to classify Wikipedia articles into named entities using SVM's with threshold adjustment [26]. They have utilized multilingual features that use unstructured and semi-structured portions of Wikipedia articles for classification. Their experimental results also showed that using multilingual features helps to improve classification accuracy.

CHAPTER 3

METHODS FOR TYPIFYING WIKIPEDIA ARTICLES

In this chapter we discuss the different methods we used for typifying Wikipedia articles. First we discuss how the words in Wikipedia articles are used as features and their variations. Then we discuss using Wikipedia entities and entity types as features respectively. We also discuss how the Wikipedia categories can be used as features to improve the accuracy. Finally we combine 2 features, word and Wikipedia categories, to improve the classification accuracy.

3.1 Using Words as Features

To train our classifier we first need to prepare our training data. As our first approach to prepare our training data, we consider the words inside each Wikipedia article. So here each word is a feature. In pattern recognition or machine learning, a feature vector is an n-dimensional vector of numerical features that represent some objects. If we consider a dictionary of words and represent word presence and absence in a Wikipedia article by 1 and 0 respectively, then we can represent a Wikipedia article by a feature vector. Before representing a Wikipedia article as a feature vector we stemmed the article using Porter Stemmer algorithm [7] and then removed the stop words (e.g., a, an, the, this, that, them, he, him, etc.) by using the MySQL Full-Text Stop words list [16]. Stemming algorithm finds the stem, base or root from derived words. In addition, it helps us to reduce the no of unique words. Stop words help us to remove those words that are irrelevant. Once these are performed on the Wikipedia article, we parse each word from the Wikipedia article and represent the Wikipedia article by the words in each article. In Figure 3.1, an example is shown for a partial Wikipedia Article “Mohammad Ali”. From Figure 3.1 it can be easily seen how the total no. of words has been reduced.

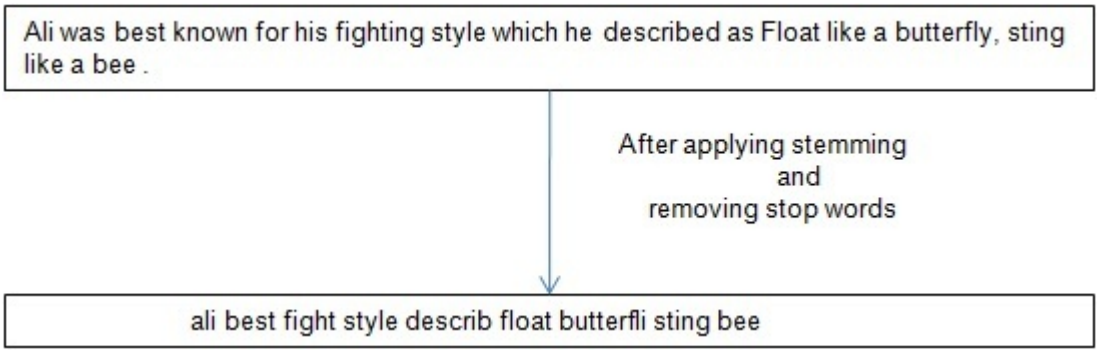


Figure 3.1: The effect of applying Stemming and removing stop words.

3.1.1 k-Local Frequent Words (k-LFW)

In this method we construct the feature vector, by considering the most frequent words from each article. The full process is as follows:

1. After preprocessing is performed on each Wikipedia article, we count the frequency of each word in each Wikipedia article.
2. Then we sort the words by their frequencies in descending order.
3. We choose k (k=5, 10, 25, 50, 75, depending on the experiment) top words from each Wikipedia article.

   For example, consider 2 articles a_1, a_2 and they have the words sorted by frequency as follows (In this chapter we will use these 2 examples several times in different methods):

a_1	$\{(w_1,8), (w_2,6), (w_3,3), (w_4,2), (w_5,2)\}$
a_2	$\{(w_1,7), (w_5,6), (w_6,4), (w_7,2)\}$

   Here, $(w_1, 8)$ means word w_1 appears 8 times in article a_1

   If the number of local frequent words to consider from each article is 3, then from a_1 we will choose $L_1 = \{w_1, w_2, w_3\}$ and from a_2 we will choose $L_2 = \{w_1, w_5, w_6\}$. So the feature vector, **L** will have the following list of features:

$$\mathbf{L} = \{w_1, w_2, w_3, w_5, w_6\}$$

4. Then we by using this feature vector we can represent each of the 2 articles (a_1, a_2). Here we use 2 different approach:

   5.a) LFW Without Frequency (LFW W/O Freq): In this approach, we consider the presence and absence of words of an article in a feature vector. We represent the absence and presence as 0 and 1 respectively. So the 2 articles a_1 and a_2 will be represented as follows:

   $$a_1 = \{1,1,1,1,0\}$$

   $$a_2 = \{1,0,0,1,1\}$$

   5.b) LFW With Frequency (LFW W/Freq): In this approach, we consider the frequency of words of an article in a feature vector. Hence in this approach the 2 articles a_1 and a_2 will be represented as follows:

   $$a_1 = \{8,6,3,2,0\}$$

   $$a_2 = \{7,0,0,6,4\}$$

5. After representing each of the Wikipedia articles as a feature vector, we use them to train our classifier for both Naïve Bayes and SVM and then we classify test data using the trained classifier for LFW W/O Freq and LFW W/Freq separately.

3.1.2 All Unique Words (AUW)

In this method we consider all unique words from each article to build the feature vector. For the 2 articles (a_1, a_2) from the last section the feature vector can be represented as below:

$$L = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$$

Here we also have considered AUW W/O Freq and AUW W/Freq. For AUW W/O Freq, the feature vectors for both articles are as follows:

$$a_1 = \{1,1,1,1,1,0,0\}$$

a_2 = {1,0,0,0,1,1,1}

For AUW W/Freq:

a_1 = {8,6,3,2,2,0,0}

a_2 = {7,0,0,0,6,4,2}

3.1.3 n-Global Frequent Words (n-GFW)

In this method we construct the feature vector as follows:

1. First we calculate the frequency of each word over all the articles. So by considering a_1 and a_2 the frequency of all the words will be as follows:

w_1	w_2	w_3	w_4	w_5	w_6	w_7
15	6	3	2	8	4	2

2. Then we sort the words based on their frequency and took the top n words to construct the feature vector. So after sorting the words, the list will be as follows:

w_1	w_5	w_2	w_6	w_3	w_4	w_7
15	8	6	4	3	2	2

If n=5, then the top 5 words from the above list will form the feature vector. This feature vector will be used to represent each Wikipedia article. Here we also consider both frequency and without frequency approach. For article (a_1, a_2) the feature vector for both GFW W/O Freq and GFW W/Freq are given below:

GFW W/O Freq:

a_1 = {1,1,1,0,1}

a_2 = {1,1,0,1,0}

GFW W/Freq:

$a_1 = \{8,2,6,0,3\}$

$a_2 = \{7,6,0,4,0\}$

3.2 Using Wikipedia Entities as Features

In this method, instead of using words as features we use Wikipedia entities as features. In Wikipedia, each article is linked to several other Wikipedia entities. In Figure 3.2 Wikipedia entities are shown partially for a Wikipedia article. The entities are those words that are hyperlinked to other Wikipedia articles.

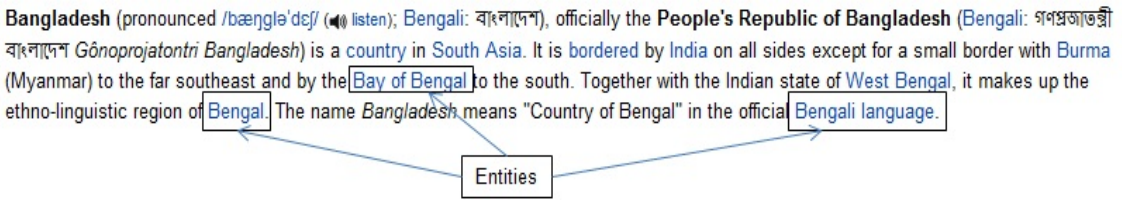


Figure 3.2: Wikipedia entities in an article.

3.2.1 k-Local Frequent Entities (k-LFE)

In this method we construct the feature vector, by considering the most local frequent entities from each article. The full process is as follows:

1. For each Wikipedia article, we count the frequency of each entity.
2. Then we sort the entities by their frequencies in descending order.
3. We choose k (k=5, 10, 25, 50, 75, depending on the experiment) top entities from each Wikipedia article.

   For example, consider 2 articles a_1, a_2 and they have the entities sorted by their frequency as follows (In this section we will use these 2 examples several times):

a_1	$\{(e_1,4), (e_2,3), (e_3,2), (e_4,1), (e_5,1)\}$
a_2	$\{(e_1,3), (e_5,2), (e_6,1), (e_7,1)\}$

Here, (e_1, 4) means the entity e_1 appears 4 times in article a_1

If the number of local frequent entities to consider from each article is 3, then from a_1 we will choose $L_1 = \{e_1, e_2, e_3\}$ and from a_2 we will choose $L_2 = \{e_1, e_5, e_6\}$. So the feature vector, **L** will have the following list of features:

$$\mathbf{L} = \{e_1, e_2, e_3, e_5, e_6\}$$

4. Then we by using this feature vector we can represent each of the 2 articles (a_1, a_2). Here we use 2 different approach similar to LFW:

   5.a) LFE W/O Freq: By following the same procedure as in LFW W/O Freq we can represent the 2 articles a_1 and a_2 as follows:

   $$a_1 = \{1,1,1,1,0\}$$

   $$a_2 = \{1,0,0,1,1\}$$

   5.b) LFE W/Freq: Similarly by following the step as in LFW W/Freq we get the following 2 representations:

   $$a_1 = \{4,3,2,1,0\}$$

   $$a_2 = \{3,0,0,2,1\}$$

5. After representing each of the Wikipedia articles as a feature vector, we use them to train our classifier for both Naïve Bayes and SVM and then we classify test data using the trained classifier for LFE W/O Freq and LFE W/Freq separately.

3.2.2 All Unique Entities (AUE)

In this method we consider all unique entities from each article to build the feature vector. For the 2 articles (a_1, a_2) from the last section the feature vector can be represented as below:

$$L = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$$

In a similar fashion as in AUW, we can achieve the following representation for AUE.

AUE W/O Freq:

$$a_1 = \{1,1,1,1,1,0,0\}$$

a_2 = {1,0,0,0,1,1,1}

For AUE W/Freq:

a_1 = {4,3,2,1,1,0,0}

a_2 = {3,0,0,0,2,1,1}

3.2.3 n-Global Frequent Entities (n-GFE)

In this method we construct the feature vector as follows:

1. First we calculate the frequency of each entity over all the articles. So by considering a_1 and a_2 the frequency of all the entities will be as follows:

e_1	e_2	e_3	e_4	e_5	e_6	e_7
7	3	2	1	3	1	1

2. Then we sort the entities list based on their frequency and took the top n entities to construct the feature vector. So after sorting the entities will be come as follows:

e_1	e_2	e_5	e_3	e_4	e_6	e_7
7	3	3	2	1	1	1

If n=5, then the top 5 entities from the above list will form the feature vector. This feature vector will be used to represent each Wikipedia article. Similarly as in GFW we can represent feature vector for GFE W/O Freq and W/Freq as below:

GFE W/O Freq:

a_1 = {1,1,1,1,1}

a_2 = {1,0,1,0,0}

GFE W/Freq:

$a_1 = \{4,3,1,2,1\}$

$a_2 = \{3,0,2,0,0\}$

3.3 Using Wikipedia Entity Types as Features (EType)

In this method, we used the type of a Wikipedia entity as feature. The type information is taken from the DBpedia ontology. In our first approach, from each Wikipedia article we get the list of entities. Then for each entity we try to get the type of that entity from the DBpedia. In DBpedia there are 200 types. Hence the size of the feature vector here is 200. If the type exists in the DBpedia then add that type in list of features for that article. So for an article a_1 if the types for the entities we get are $\{t_1,t_2,t_3,t_4,t_{87}, t_{100}, t_{102}\}$ then this article can be represented as follows:

	t_1	t_2	t_3	t_4	…	t_{86}	t_{87}	…	t_{100}	t_{102}	…	t_{200}
a_1	1	1	1	1	…	0	1	…	1	1	…	0

The above table shows the feature vector where the frequency is not considered. In another approach, we count the frequency of entity types as well and used that frequency in the feature vector. Here the frequency of an entity type is the total number of times that type appears in that article. For example, we get the frequency count for (t_1, t_2, t_6, t_9) in an article as (3,4,6,1). Then the feature vector will be as below:

	t_1	t_2	t_3	t_4	…	t_6	t_7	…	t_9	t_{102}	…	t_{200}
a_1	3	4	0	0	…	6	0	…	1	0	…	0

3.4 Using Wikipedia Categories as Features

In this method, we use Wikipedia categories as features. In Figure 3.3, a category list is shown for a Wikipedia article. In Wikipedia, there is a hierarchy of categories. Each Wikipedia article can have multiple categories.

Categories: Bangladesh | South Asian countries | Countries of the Indian Ocean | Bengal | Developing 8 Countries member states | Divided regions | Least Developed Countries | Liberal democracies | Members of the Commonwealth of Nations | Organisation of the Islamic Conference members | South Asian Association for Regional Cooperation member states | States and territories established in 1971 | Former British colonies

Figure 3.3: Wikipedia Category List

3.4.1 Level-1 Categories

In this method, only the Level-1 categories of a Wikipedia article are used to construct the feature vector for representing an article. In Figure 3.4, Level-1 categories are shown for Wikipedia article "Muhammad Ali". Here also we propose 2 variations, one is All Unique Categories (AUC L1) and the other one is Global Frequent Categories (n-GFC L1). These are discussed below:

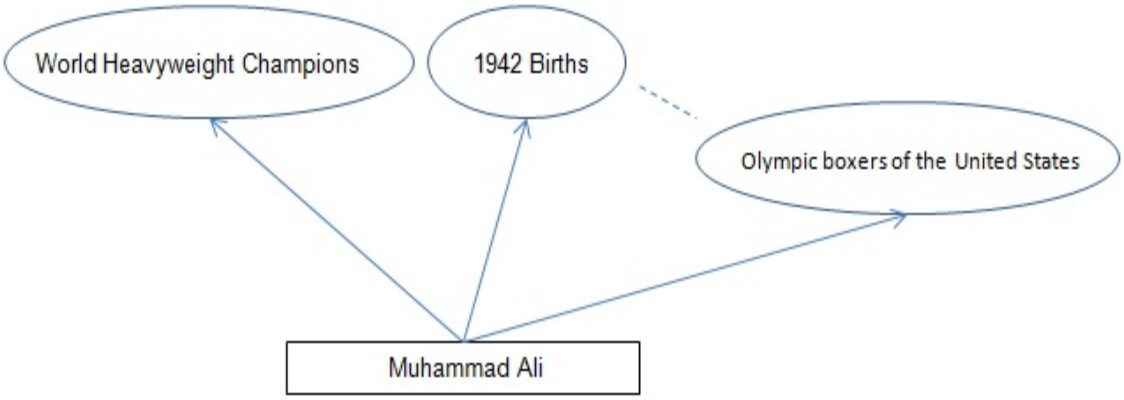


Figure 3.4: Partial Level -1 Categories for Wikipedia Article "Muhammad Ali"

- All Unique Categories (AUC L1):
    1. For each article we get the Level-1 categories and construct a list of unique categories by combining all these categories. For example, article a_1 and a_2 has category list $\{c_1, c_2, c_3, c_4\}$ and $\{c1, c2, c_5, c_6\}$ respectively. Then the unique category list will be $\{c_1, c_2, c_3, c_4, c_5, c_6\}$ which will form the feature vector.

2. Then we represent each article by the feature vector as follows:

   a_1 = {1,1,1,1,0,0}

   a_2 = {1,1,0,0,1,1}

In this method we didn't represent Wikipedia articles by using frequencies of categories. The reason behind this is Level-1 categories cannot appear multiple times in a Wikipedia article.

- Global Frequent Categories (n-GFC L1):

1. In this method, we construct a list of unique categories by combining all the Level-1 categories and also calculate the frequency of each category over all the articles.
2. Then we sort the categories by their frequency in descending order and took top n categories.

   For article a_1 and a_2 from the above example, the sorted category list by their frequency will be {c_1, c_2, c_3, c_4, c_5, c_6}. If we choose top 5 categories as features in the feature vector then the articles can be represented as follows:

   a_1 = {1,1,1,1,0}

   a_2 = {1,1,0,0,1}

3.4.2 Level-2 Categories

In this method, we consider the super categories of an article up to Level-2. In Figure 3.5 both Level-1 and Level-2 categories are shown. For each Wikipedia article, we get the list of categories for both Level-1 and Level-2. Here we proposed two approaches.

- All unique categories with Level-2 categories (AUC L2)
- Global frequent categories with Level-2 categories (n-GFC L2)

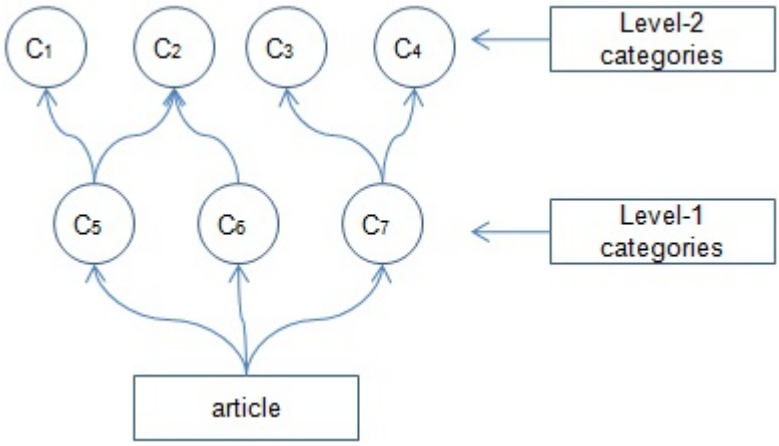


Figure 3.5: Level-2 categories in a Wikipedia article.

AUC L2:

Here we construct a list of unique categories in the same way as we did in AUC L1. Then we use this list of unique categories as features in the feature vector and represent each article by this feature vector. From Figure 3.5, the article has 3 Level-1 categories {c_5, c_6, c_7}. These 3 categories are added in the unique category list. c_5 has to super categories {c_1, c_2}. These 2 categories are also added in the unique category list. c_6 has one super category c_2. But as it is already added in the list it will be ignored. {c_3, c_4} are the super categories for c_7 and will be added in the unique category list. So the unique category list we get here is {c_1, c_2, c_3, c_4, c_5, c_6, c_7}. Same way we keep updating the unique category list by considering the categories of other Wikipedia articles. Once we have the final unique category list, we can use that list to construct our feature vector and represent each article. For representing each article, we consider again AUC L2 W/O Freq and AUC L2 W/Freq.

AUC L2 W/O Freq:

If the unique category list has {c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9} then in this method an article with category list {c_1, c_2, c_3, c_4, c_5, c_6, c_7} will be represented as {1,1,1,1,1,1,1,0,0} where 1 indicates the presence of that category and 0 means absence of that category in that article.

AUC L2 W/Freq:

In this approach, instead of 1/0, we use frequency of each category in an article. We count the number of times a category appears for up to lavel-2 after we get the unique category list. From Figure 3.5 the frequency of c_5, c_6, and c_7 is 1. But, the frequency of c_2 is 2 since it appears as a super category for both c_5 and c_6. So the article with category list {c_1, c_2, c_3, c_4, c_5, c_6, c_7} will be represented as {1,2,1,1,1,1,1,0,0} by using the unique category list shown in the example for AUC L2 W/O Freq.

n-GFC L2:

The steps for this method are as follows:

1. The frequency of each category (categories up to Level-2) is calculated over all the articles. The frequency is calculated the same way as we did for the method AUC L2 W/Freq
2. The category list is sorted in descending order based on their frequency.
3. We choose the top n categories from the sorted list to construct the feature vector.

For example, suppose there are 2 articles a_1 and a_2 having the following frequencies for their category list.

a1 = {(c1,1) (c2,1) (c3, 2) (c4, 1)}

a2 = {(c3,1) (c4,1) (c6, 2)}

The frequency of each category will be {(c1,1) (c2,1) (c3,3) (c4,2) (c6,2)} over the 2 articles. If we sort them in descending order and take the top n=3 categories then the list will be {c3, c4, c6} which is the feature vector. Again we consider both frequency and without frequency to represent each article.

GFC L2 W/O Freq:

a1 = {1,1,0}

a2 = {1,1,1}

GFC L2 W/Freq:

a1 = {2,1,0}

a2 = {1,1,2}

3.5 Improving Classification Accuracy

To improve the classification accuracy we combine 2 different types of features from each article to form a new feature vector:

1. Words in a Wikipedia article
2. Categories in the article

To select the feature "words" we followed the same process as we did in 25-LFW W/Freq and to select the feature category we follow the steps in AUC L2 W/Freq.

3.5.1 Combining features

For combining the 2 kinds of features, we just concatenate them together to form a new combined feature list. This can be better explained by a simple example. Suppose by using 25-LFW W/Freq we get the feature list as {w1,w2,w3,w4,w5,w6} and by using AUC L2 W/Freq we get the feature list as {c1,c2,c3,c4,c5,c6}. Then the combined feature list will be:

{w1,w2,w3,w4,w5,w6,c1,c2,c3,c4,c5,c6} … (1)

Now, if an article a1 has words with corresponding frequencies as {(w1,6)(w3,5)(w4,1)} and the category list with their corresponding frequencies as {(c1,4)(c3,2),(c5,1)}. Then the article a_1 will be represented by using the feature vector in (1) as follows:

{6,0,5,1,0,0,4,0,2,0,1,0}

So we represent each article with our combined feature vector which will form our training data and train our classifier based these training data.

CHAPTER 4

IMPLEMENTATION

In this section, detailed explanation of preprocessing DBpedia dataset and Wikipedia dataset is given. We performed the experiments using Weka [5] api for java. The technical details are discussed in this chapter. How the Wikipedia articles are stemmed and how the stop words are removed are also described briefly. Finally parameter tuning for SVM is explained.

4.1 Preprocessing of DBpedia Datasets

In our implementation we have considered “Ontology Infobox types” dataset (*.csv) from the list of DBpedia core datasets [6]. The whole dump is exported to a MySQL database by using MS Access as an export tool. Ontology Infobox type dataset contains triples of the form:

<< $object, rdf:type, $class >>

Here $object contains the name of the Wikipedia article, $class contains the type of a particular Wikipedia article and rdf:type indicates the relationship. One example of such an instance in the dataset is given below:

- $object: <http://dbpedia.org/resource/Alain_Connes>
- rdf:type: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
- $class: <http://dbpedia.org/ontology/Scientist>

In DBpedia, the $object is in URL encoded format. In order to get the name of an article, apply URL decode first and then insert it into our database table “instance_type_mapped” inside the “typing_wikipedia” database. For example, after applying URL decode to ‘%21Hero’, the article name will become ‘!Hero’ which is the real article name in Wikipedia. Also, we extracted only the type name from the $class object by removing the extra characters that was introduced by DBpedia framework. For example, http://dbpedia.org/ontology/Scientist is extracted as Scientist,

which is the type name for ‘Alain_Connes’ in the above example. We extracted 200 unique types from DBpedia dataset by applying our extraction method. These individual types are in the ‘dbpedia_types’ table of ‘typing_wikipedia’ database. We extracted 847,210 articles with their corresponding types. These article names are mapped with Wikipedia page titles to get their corresponding “page_id” from the Wikipedia database, since the page id in DBpedia is different from page id in table “article” under the Wikipedia database “wikidata20091001”. We store the Wikipedia articles’ text content in “article” table where each Wikipedia article can be identified by their unique page id.

We work with specific types instead of more general types. In order to find the general and specific types, we first calculate the number of Wikipedia articles in each DBpedia types. Then we sort the DBpedia types in ascending order based on their count for each type. The general types will have larger count and specific types will have smaller count since specifics types are the children of the general types. So we manually look into the count for each type and choose the top 100 specific types. There are more than 100 specific types in DBpedia ontology. Also, this selection of specific types is verified manually by looking into the DBpedia ontology which can be found at [22]. We store all the specific types in ‘specifictype’ datatable of ‘typing_wikipedia’ database. To improve the performance of our system and to query the large amount of dataset, we have applied BTree index on the data value in each table. For example, we have created ‘lookupinstancename’ index on the ‘instance_name’ attribute of the ‘instance_type_mapped’ data table in ‘typing_wikipedia’ database.

4.2 Preprocessing of Wikipedia Articles

We have used Wikipedia xml dump which is extracted from Wikipedia on the 1st of October 2009. The xml dump is stored in the database for the ease of operation. In our experiment we have worked with 5000 training Wikipedia articles and 1000 test Wikipedia

articles. We have separated the training data and test data explicitly so that there is no overlap between the training data and test data.

4.3 Stemming Algorithm

Stemming algorithm is applied to the Wikipedia articles which helps us to reduce the total number of unique words. This also helps us to reduce the complexity of the data in our dataset. We use the Porter Stemming algorithm [7] to stem the Wikipedia articles.

4.4 Parameter Tuning Non-Linear SVM with Polynomial Kernel

For SVM, a good accuracy can be achieved by choosing the optimal parameter values and the right kernels. As a baseline approach we first tried to use the linear SVM for LFW with 5000 training data, considering frequency and without frequency. But, we got a very low accuracy of 26% which tells us that our dataset is nor linearly separable. So we decided to use the polynomial kernel. In polynomial kernel there are 2 parameters involved. One is the polynomial exponent (PE) and the other one is the complexity parameter (C). We started with a value of {C, PE} = {1, 1} and measure the accuracy. Then we increase the C parameter by making the PE constant with a step size of 0.1 and do the vice-versa and measure the accuracy. The same we decrease the C and PE value with step size of 0.1. We also measure the accuracy by increasing the C value and decreasing the PE value simultaneously and vice versa. Then we choose the parameter values for the maximum accuracy. We further tune the parameter by performing the same process describes above for a step size of 0.01. We have performed the same process for different features to choose the best parameter values for C and PE in SVM using polynomial kernel.

CHAPTER 5

EXPERIMENTS AND EVALUATION

In this chapter we present the experiments we have performed for different setups. We use these results to evaluate the performance of our methods. We also compare our methods based on their accuracy. In the following sections we describe the experimental setups, results and comparison of those results.

5.1 Setup

For experiments we develop our program in Java. We use java API and Weka API for java. We vary our training dataset from 2000~5000 Wikipedia articles and as a test set we used 1000 test Wikipedia articles. There are 100 class labels which are the types from DBpedia ontology. We perform our experiments for all the methods by using both naïve Bayes and SVM with polynomial kernels.

Table 5.1: Summary of Basic Setup for Classification

	Dataset size			
Training Data	2000	3000	4000	5000
Test Data	1000			
No. of Class labels (types)	100			

We run the experiments for each of the method we discussed in chapter 3. Then we compare the methods based on accuracy. Accuracy is defined as follows:

$$\text{Accuracy, (\%)} = \frac{\text{No. of correctly classified instances}}{\text{Total } \textit{no. of instances in the test set}} \cdot 100$$

5.2 Results and Comparison

Figure 5.1 and Figure 5.2 show the accuracy for 25 LFW W/Freq and W/O Freq by using naïve Bayes classifier and SVM respectively. From the results it can be said that LFW W/Freq

performs better than LFW W/O Freq in both Naïve Bayes and SVM. Figure 5.3 compares the accuracy between Naïve Bayes and SVM for 25 LFW W/Freq. In Table 5.2 different parameter values for SVM are listed for different methods we used. For each method we try to get the parameter values in such a way that gives us the best accuracy. We tried different parameter values for complexity parameter and polynomial exponent. In Table 5.2 the parameter values listed are those values for which we get the best accuracy.

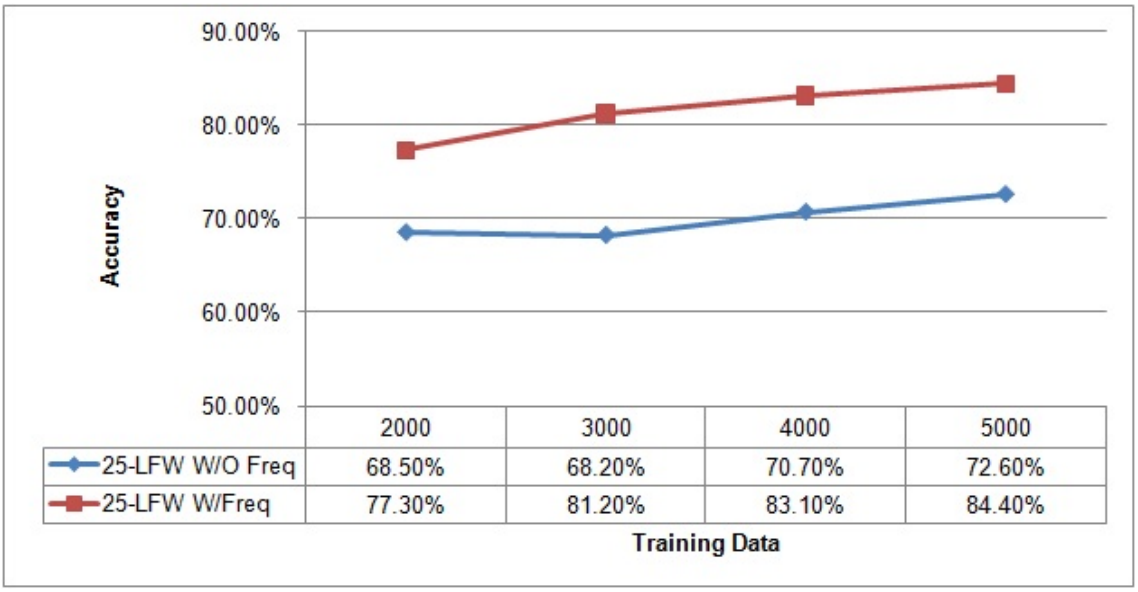


Figure 5.1: Comparison between 25-LFW W/Freq and W/O Freq for Naïve Bayes.

Table 5.2: Parameter Values for Polynomial Kernel in SVM

Methods	Complexity Parameter (C)	Polynomial Exponent (PE)
k-LFW, n-GFW, AUW W/O Freq	0.2	1
k-LFW, n-GFW, AUW W/Freq	0.2	0.5
k-LFE, n-GFE, AUE W/O Freq	3	0.1
k-LFE, n-GFE, AUE W/Freq	1.8	0.18
AUC L1, GFC L1 W/O Freq	0.85	0.6
AUC L1, GFC L2 W/Freq	0.85	0.6
Combined features	0.8	0.24

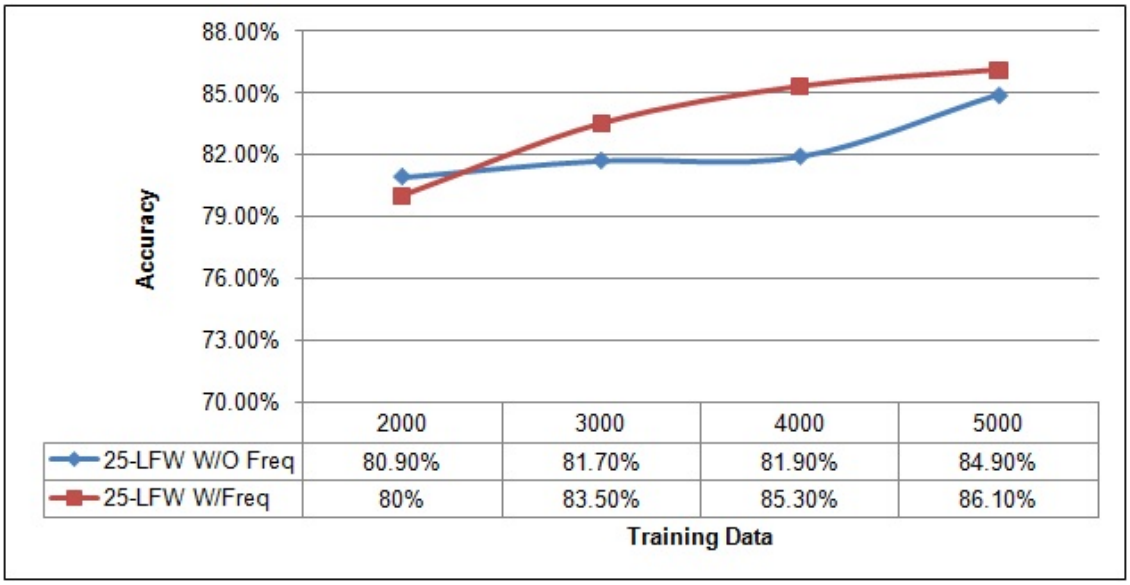


Figure 5.2: Comparison between 25-LFW W/Freq and W/O Freq for SVM.

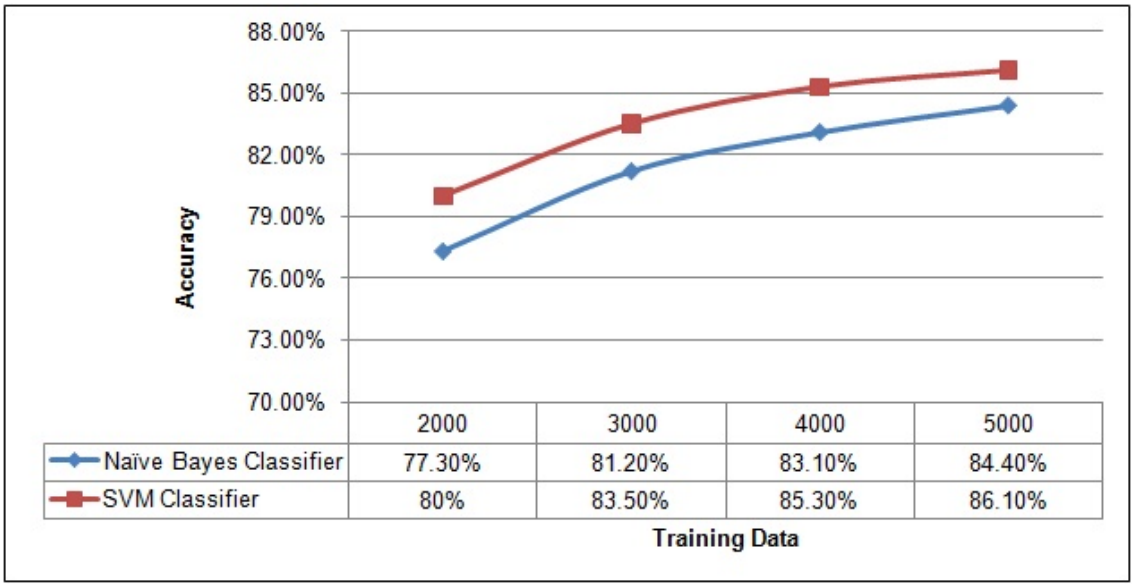


Figure 5.3: Comparison between Naïve Bayes and SVM classifier for 25-LFW W/Freq

Next, In Figure 5.4 we compare the results for 25-LFW W/Freq, AUW W/Freq and 20000-GFW W/Freq using naïve Bayes classifier. We choose 25-LFW W/Freq, AUW W/Freq and 20000-GFW W/Freq since from Table 5.3 and Table 5.4 it can be seen that we get the best accuracy for them. We achieve a maximum of 84.4% accuracy for 25-LFW W/Freq. The feature vector

size in this case is 18K. From the figure 5.4 AUW W/Freq perform worse than 25-LFW W/Freq and 20000-GFW W/Freq. The feature vector size for AUW W/Freq is 69K in this case. One reason is that it was over fitting because of the large size of the feature vector.

Table 5.3: Accuracy of k-LFW W/Freq, where k=5, 10, 25, 50, 75 using Naïve Bayes

No. of Training data	5-LFW W/Freq	10-LFW W/Freq	25-LFW W/Freq	50-LFW W/Freq	75-LFW W/Freq
2000	61.80%	73.60%	77.30%	80.20%	79.70%
3000	64.50%	76.10%	81.20%	81.70%	80.40%
4000	68%	78.00%	83.10%	83.60%	82.50%
5000	69%	79.30%	84.40%	84.40%	83.40%

Table 5.4: Accuracy of AUW W/Freq and n-GFW W/Freq, where n=20000, 30000, 40000 using Naïve Bayes

No. of Training data	AUW W/Freq	20000-GFW W/Freq	30000-GFW W/Freq	40000-GFW W/Freq
2000	68.50%	72.80%	70.40%	68.50%
3000	67.20%	75.50%	72.00%	69%
4000	68.90%	79.10%	76.60%	73.60%
5000	71.80%	81.60%	80%	77%

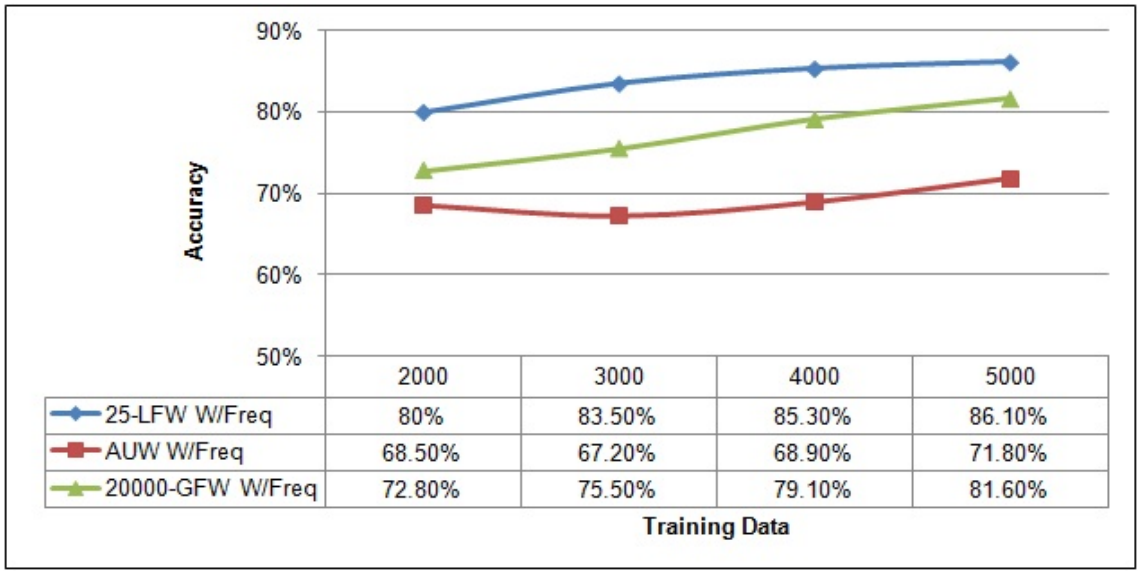


Figure 5.4: Comparisons between 25-LFW, AUW, and 20000-GFW W/Freq for Naïve Bayes

In Figure 5.5 we compare 25-LFE W/Freq, AUE W/Freq and 20000-GFE W/Freq methods. From Table 5.5 it can be seen that we get the best accuracy for 25-LFE W/Freq for 500 train data. From Table 5.6 AUW W/Freq for 5000 train data has the best accuracy. We choose 20000-GFE W/Freq since 20000-GFE W/Freq and 30000-GFE W/Freq has almost the same accuracy for 5000 train data. The lower accuracy for these three methods is for the reason that we consider only the entities from each Wikipedia article and the number of entities for each Wikipedia article can be small in number.

Table 5.5: Accuracy of k-LFE W/Freq, where k=5, 10, 25, 50, 75 using Naïve Bayes

No. of Training data	5-LFE W/Freq	10-LFE W/Freq	25-LFE W/Freq	50-LFE W/Freq	75-LFE W/Freq
2000	49.10%	56.10%	59.2	58.60%	58.20%
3000	51.40%	57.20%	61%	60.60%	60.40%
4000	56.50%	62.50%	65%	64.66%	64.20%
5000	58.30%	63.60%	68%	67%	66.20%

Table 5.6: Accuracy of AUE W/Freq and n-GFE W/Freq, where n=20000, 30000 using Naïve Bayes

No. of Training data	AUE W/Freq	20000-GFE W/Freq	30000-GFE W/Freq
2000	58%	57%	57.6
3000	61%	58.00%	59.60%
4000	64%	61.00%	62.30%
5000	67%	65.00%	65.6%

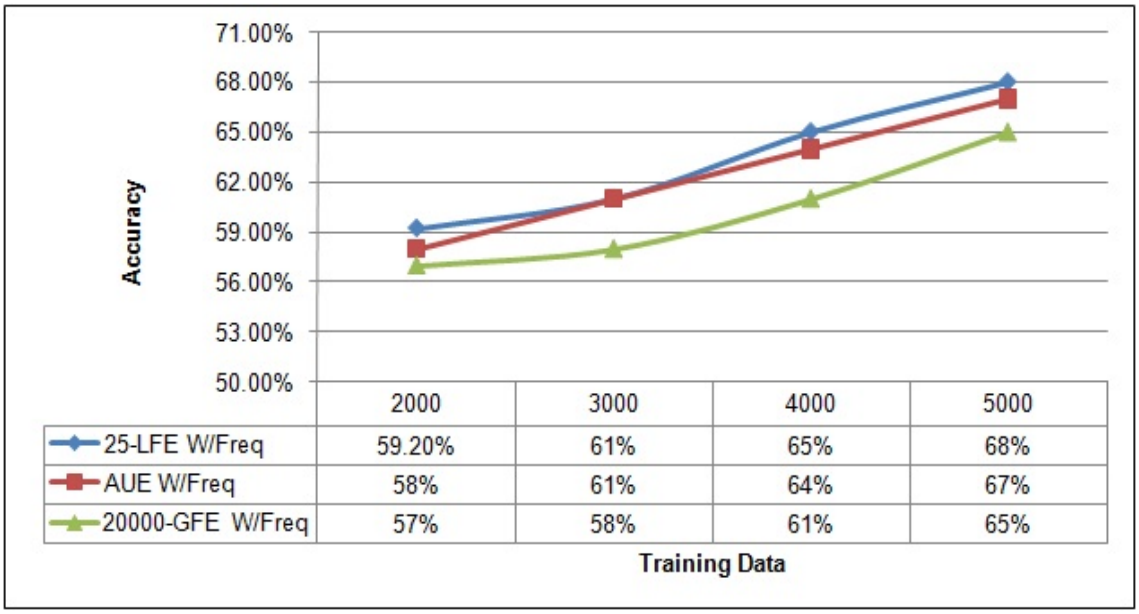


Figure 5.5: Comparisons between 25-LFE, AUE, and 2000-GFE W/Freq for Naïve Bayes

In Figure 5.6 we compare the results for AUC L1, AUC L2 W/O Freq and AUC L2 W/Freq. The accuracy for AUC L1 is low compared to other 2 methods which are reasonable since AUC L1 has very small number of categories in the feature list. For AUC L1 it is not possible to use the frequency since categories in an article cannot appear multiple times.

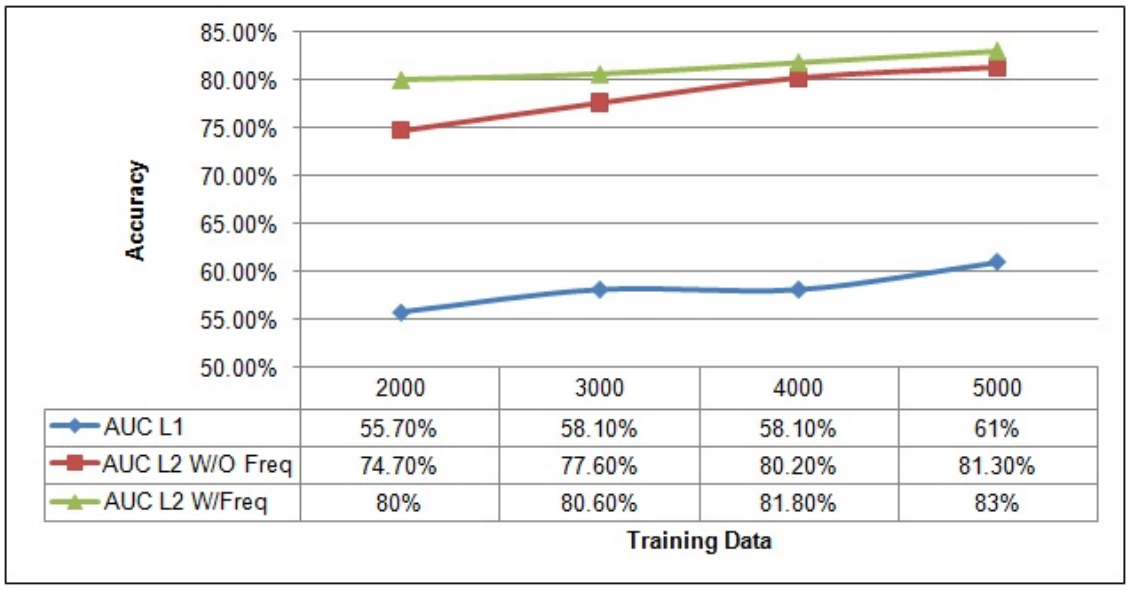


Figure 5.6: Comparisons between AUC L1, AUC L2 W/O Freq, and AUC L2 W/Freq for Naïve Bayes

On the other hand, for AUC L2 W/Freq and W/O Freq the accuracy is high because there are more categories in the feature list which increase the chance of predicting the type of an article. Next, we compare the methods with combined features for both naïve Bayes and SVM. From Figure 5.7, we achieve 91% accuracy by using the method of combined feature for SVM. The parameter values for this experiment are given in Table 5.2. For naïve Bayes we also get a good accuracy of 88.1%.

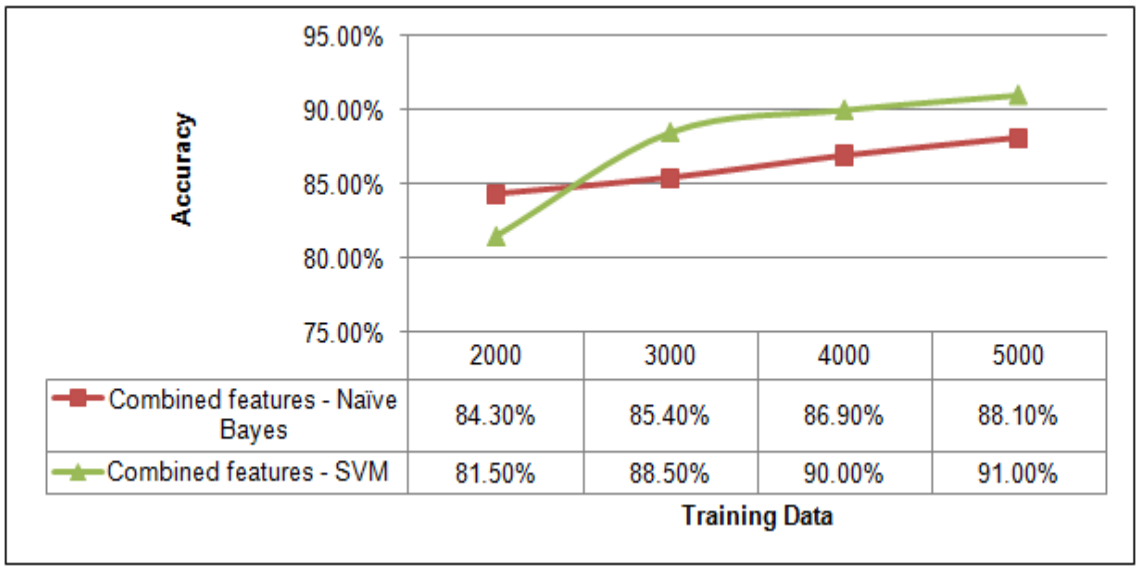
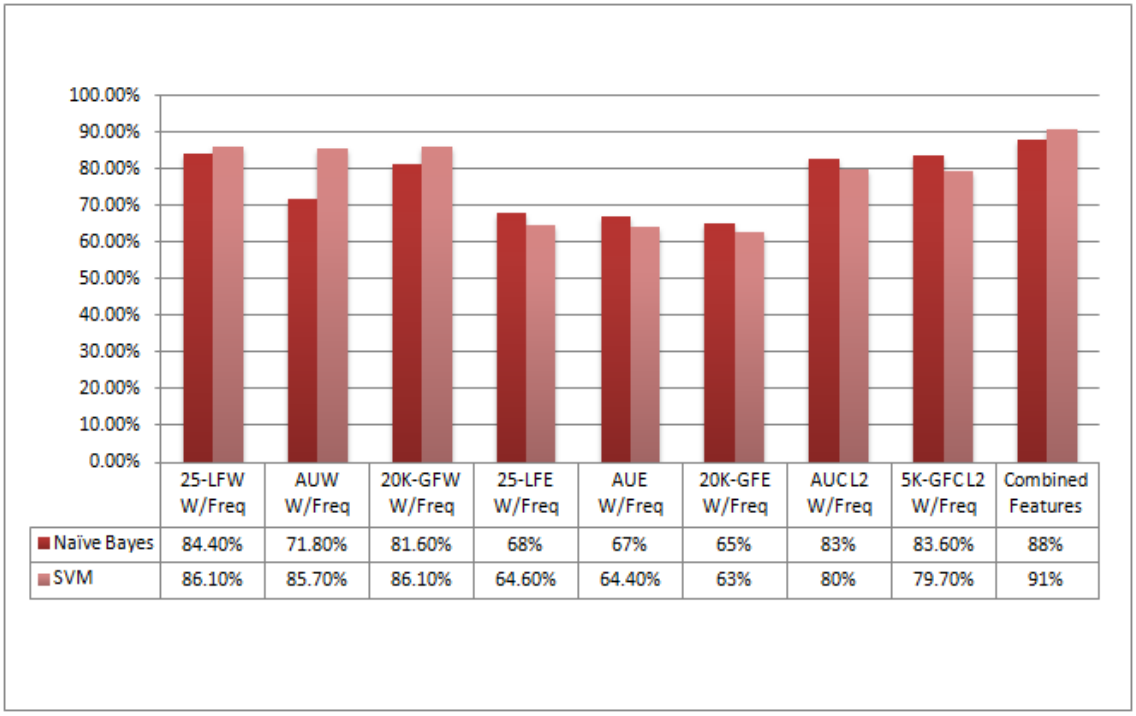


Figure 5.7: Comparison between combined features using naïve Bayes and SVM classifier

Finally we compare all the methods for both SVM and naïve Bayes in Figure 5.8. From this figure we see that our new method AUC L2 W/Freq performs as good as in 25 LFW W/Freq. Also, the other new method where we combined the features outperformed all the other methods.



	25-LFW W/Freq	AUW W/Freq	20K-GFW W/Freq	25-LFE W/Freq	AUE W/Freq	20K-GFE W/Freq	AUCL2 W/Freq	5K-GFCL2 W/Freq	Combined Features
Naïve Bayes	84.40%	71.80%	81.60%	68%	67%	65%	83%	83.60%	88%
SVM	86.10%	85.70%	86.10%	64.60%	64.40%	63%	80%	79.70%	91%

Figure 5.8: Comparison of all the methods for Naïve Bayes and SVM classifier

CHAPTER 6

CONCLUSION AND FUTURE WORKS

Typifying Wikipedia articles correctly is very important because of the large, continuously growing Wikipedia articles. Also, better organization of Wikipedia articles by grouping the articles according to their types will help people retrieve article. Moreover, people will be able to explore or filter documents of the same type together. In this thesis, we applied different features to typify Wikipedia articles. We propose a new method of typifying Wikipedia articles based on their categories and shows that it can achieve almost the same accuracy compared to the standard methods. In addition, we further improved our accuracy by combining the features. Our results for combined features demonstrate that it outperforms other standard methods and even perform better than our new method for AUC L2. Also, our results show that SVM outperforms Naïve Bayes classifier for the combined features if the parameter for SVM can be chosen carefully.

We developed our proposed our methods using Wikipedia category hierarchy up to Level-2. In future, we will be working on replacing this Wikipedia category hierarchy by YAGO [3] and compare its accuracy with our current method.

REFERENCES

[1] http://en.wikipedia.org/wiki/Wikipedia:About.

[2] http://en.wikipedia.org/wiki/Wikipedia:Overcategorization

[3] Suchanek, Fabian M., Kasneci, Gjergji and Weikum, Gerhard, “Yago - A Large Ontology from Wikipedia and WordNet”, 2008.

[4] http://blog.dbpedia.org/category/dataset-releases/

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

[6] http://wiki.dbpedia.org/Downloads351#h115-3

[7] Porter, M.F. (1980), An Algorithm for Suffix Stripping, Program, 14(3): 130–137

[8] http://tartarus.org/~martin/PorterStemmer/

[9] http://www.cs.waikato.ac.nz/~ml/weka/arff.html

[10] http://nlp.stanford.edu/IR-book/html/htmledition/the-text-classification-problem-1.html

[11] http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html

[12] http://en.wikipedia.org/wiki/Support_vector_machine

[13] Cortes, Corinna and Vapnik, V., "Support-Vector Networks", Machine Learning, 20, 1995

[14] http://www.dtreg.com/svm.htm

[15] http://wiki.dbpedia.org/Ontology

[16] http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html

[17] Y. H. Li and A. K. Jain, Classification of Text Documents. The Computer Journal, 41(8), 1998, 537-546.

[18] Zhang, T., and Oles, F. J., Text categorization based on regularized linear classification methods. Information Retrieval, 4, 2001, 5–31.

[19] Pu Wang, Jian Hu, Hua-Jun Zeng and Zheng Chen, Using Wikipedia knowledge to improve text classification. Knowledge and Information Systems Journal, 19, 2008, 265-281.

[20] Hu Guan, Jingyu Zhou and Minyi Guo, A class-feature-centroid classifier for text categorization. Proceedings of the 18th international conference on World Wide Web, 2009, 201-210.

[21] E. Youn, M. K. Jeong, Class dependent feature scaling method using naive Bayes classifier for text datamining, Pattern Recognition Letters, 2009.

[22] http://www4.wiwiss.fu-berlin.de/dbpedia/dev/ontology.htm

[23] Kaptein, Rianne and Kamps, Jaap, Using Links to ClassifyWikipedia Pages, INEX 2008, LNCS 5631, pp. 432–435, 2009.

[24] Ringland, Nicky, Nothman, Joel, Murphy, Tara and R. Curran, James. Classifying articles in English and German Wikipedia (2006)

[25] http://www.dtreg.com/svm.htm

[26] Saleh, Imam, Darwish, Kareem, and Fahmy, Aly. Classifying Wikipedia Articles into NE's using SVM's with Threshold Adjustment, Proceedings of the 2010 Named Entities Workshop, ACL 2010, pages 85–92.

BIOGRAPHICAL INFORMATION

Quazi Mainul Hasan completed his Bachelors in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in June, 2007. As a Software Engineer he worked in several software outsourcing companies and has almost 4 years of professional experience. He started his Masters in Computer Science at The University of Texas at Arlington in Fall 2008 and joined The Innovative Databases and Information Systems Research (IDIR) Lab at UT Arlington in Summer 2009. He also worked as a Graduate Assistant in Student Enrollment Service department at UT Arlington from Fall 2008. His research involves data mining, web data management, and information retrieval.