Spring 2015

COMPARATIVE ANALYSIS OF PARTICLE SWARM OPTIMIZATION ALGORITHMS FOR TEXT FEATURE SELECTION

Shuang Wu
San Jose State University

COMPARATIVE ANALYSIS OF PARTICLE SWARM OPTIMIZATION ALGORITHMS FOR TEXT FEATURE SELECTION

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Shuang Wu

May 2015

The Designated Project Committee Approves the Project Titled

COMPARATIVE ANALYSIS OF PARTICLE SWARM OPTIMIZATION ALGORITHMS FOR TEXT FEATURE SELECTION

by

Shuang Wu

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2015

Dr. Sami Khuri

Dr. Suneuy Kim

Dr. Teng Moh

ABSTRACT

COMPARATIVE ANALYSIS OF PARTICLE SWARM OPTIMIZATION ALGORITHMS FOR TEXT FEATURE SELECTION

by Shuang Wu

With the rapid growth of Internet, more and more natural language text documents are available in electronic format, making automated text categorization a must in most fields. Due to the high dimensionality of text categorization tasks, feature selection is needed before executing document classification. There are basically two kinds of feature selection approaches: the filter approach and the wrapper approach. For the wrapper approach, a search algorithm for feature subsets and an evaluation algorithm for assessing the fitness of the selected feature subset are required. In this work, I focus on the comparison between two wrapper approaches. These two approaches use Particle Swarm Optimization (PSO) as the search algorithm. The first algorithm is PSO based K-Nearest Neighbors (KNN) algorithm, while the second is PSO based Rocchio algorithm. Three datasets are used in this study. The result shows that BPSO-KNN is slightly better in classification results than BPSO-Rocchio, while BPSO-Rocchio has far shorter computation time than BPSO-KNN.

ACKNOWLEGEMENT

I would like to express my sincere gratitude to Dr. Sami Khuri for his instruction on this project. I would also like to thank Dr. Suneuy Kim and Dr. Teng Moh for being my committee members. Finally, I would like to thank my dear husband for being so supportive in every way. I couldn't have done all these without his support.

TABLE OF

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

With the rapid growth of Internet and mobile communication, we have many more electrical documents available on all kinds of websites and cloud storages than ever before. In order to better understand and organize these documents, automatic text categorization has become a must for industrial and academic purposes. Text categorization consists in classifying a document into one or several pre-defined categories according to their contents. Many applications, such as search engines, take advantage of this technology.

The first step of text categorization is to transform a document into a vector. Each dimension of this vector corresponds to a term present in the whole dataset. If this term ever occurs in this document, its value will be a non-zero double value. Therefore, it brings a critical question: should we use all the terms appeared in the datasets? Or should we just select some representative terms considering the fact that there are so many terms in a dataset while not all of them are necessary for text categorization? This problem is known as feature selection in machine learning because a term is regarded as a feature or a dimension when a document is transformed into a vector in text categorization task. Feature selection is extremely important for text categorization since it is quite normal to have more than ten-thousand terms in a document dataset. Such high dimensionality makes it very difficult to carry out text categorization using machine learning algorithms. By doing feature selection, not only can we decrease the dimension, but also we can eliminate redundant and irrelevant features, so that classification performance can be improved, learning and executing process can be made faster, and the structure of the learning model can be simplified [1].

There are basically two categories of feature selections: filter approaches and wrapper approaches. Filter approach doesn't need a specific learning algorithm. Metrics, such as Information Gain, are used to measure features due to the information it carries. Features are ranked according to the score gained using metrics and then selected out. Wrapper approach, in contrast, needs to have a particular learning algorithm. A subset of features is selected out to be used in this algorithm so that it can be evaluated by the classification accuracy [2]. Generally speaking, the wrapper approach has better performance than the filter approach because it considers the whole subset of features rather than a single feature. However, the filter approach is argued to be more efficient and more general.

In this work, I used both the filter approach and the wrapper approach. The filter approach was used at first to reduce the dimension to the level that the dimension of a dataset can be processed with not so much effort when using the wrapper approach. Here I take advantage of the fast computation of the filter approach. Because if I use the wrapper approach directly, the computation time of feature selection will be quite long due to the complexity of calculation of accuracy. What's more, it is more convenient and efficient to get rid of some noisy and unnecessary features using the filter approach before I compare the two different wrapper approaches in question.

For the filter approach, I used Information Gain to measure the score of each feature. After the dimension reduction by Information Gain, I compared two wrapper approaches. As stated before, the wrapper approach needs a specific machine learning algorithm to calculate the accuracy of classification. There are two classifiers I used in this study. One is the K-Nearest Neighbors, the other is Rocchio. Meanwhile, I also used

a method to search the feature space. Suppose we have a dataset with *n* features, the size of search space would be 2^n. Considering the fact that the dimension of dataset after the filter approach is still quite high, it is impossible to search the whole space exhaustively in most cases [1]. Therefore, in this work, I used Particle Swarm Optimization (PSO) which is a heuristic algorithm that belongs to evolutionary computation technique. This kind of technique, including PSO, Genetic Algorithms (GA), Ant Colony Optimization (ACO) etc., is famous for its global search ability. Compared to other evolutionary search algorithms, PSO has fewer parameters, and is less computationally expensive. It also has the advantage of converging more quickly. Therefore, PSO is considered to be a promising search method for feature selection problems [3].

The rest of this report is organized in the following fashion: 1) In the Literature Review section, I go through the basic concept of Vector Space Model and term frequency-inverse document frequency, explain what the filter approach and the wrapper approach are and go into detail about the filter method -- information gain and the wrapper approaches -- Particle Swarm Optimization based K-Nearest Neighbors and Rocchio which I used in this work. I also briefly introduce Weka and the three classifiers used. 2) In the Experimental Study section, I show the two corpuses used in my experiment and elaborate implementation and experiment procedures. I show the results of text classification and analyze comparatively for Binary PSO-KNN and Binary PSO-Rocchio. At the end, I draw a conclusion and discuss the possible improvement for future work.

2 Literature Review

2.1 Text Pre-processing, VSM and tf-idf

Text categorization (TC) belongs to the task of Natural Language Processing and Data Mining. It takes unstructured texts which are natural languages as input, and uses machine learning algorithms to classify the input into different categories from a pre-defined set [4]. If every text instance in this dataset has only one category, it is a single-label TC task. If every instance has two or more categories, it is a multi-label TC task [5]. In this work, I only consider the case of single-label TC task because multi-label TC task is much more complex and beyond the scope of my study. Compared to other classification problems which are not formed by data of natural languages, the feature space of TC is especially high dimensional and sparse. In TC, many features are noisy and unnecessary. For example, there are words like “a”, “the” and “my” that cannot offer any information in TC problem. Therefore, it is quite necessary to execute dimension reduction before classification procedure.

2.1.1 Text pre-processing

Text-preprocessing procedure usually includes the following steps: 1) conversion to UTF-8 encoding; 2) removing hyphens, punctuation marks, numbers, digits, non-English letters and diacritics; 3) removing stop words (such as “the”, “at”, “I” and “on”); 4) eliminating rare words (words that occur less than five times in the dataset); and 5) executing word stemming [2].

2.1.2 Vector Space Model (VSM)

Before we go deep into dimension reduction, let us first take a look at the concept of Vector Space Model (VSM). VSM is an algebraic model for representing text documents as vectors of terms or phrases [19]. The set of terms are the terms that are present in the whole dataset of TC problems, and the class label of one entry is from categories of a pre-defined set as mentioned before. Usually, if a term occurs in a document, the vector of this term in the entry of this document will not be zero.

So what makes VSM model valid for TC? There is a basic hypothesis in using VSM for classification which is called contiguity hypothesis. It assumes that "Documents in the same class form a contiguous region and regions of different classes do not overlap." [11] How should we interpret this hypothesis? There are many TC tasks that can be classified by word patterns. For example, suppose we have a TC task that has three classes -- China, Kenya and UK. Documents that belong to the China class tend to have high values on features such as Beijing, Chinese and Mao, while documents that belong to UK tend to have high values on dimensions such as London, British and Queen. Documents from different classes therefore have clear and contiguous regions of their own, as shown in Fig. 1, and it should be feasible to draw boundaries among all the classes so that classification can be applied [11].

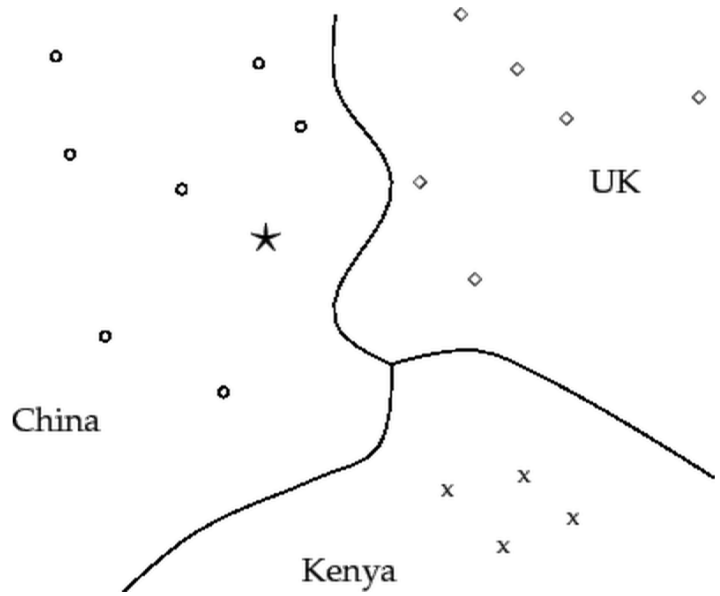

Figure 1: Vector space classification into three classes [11]

Whether the documents belonging to the same class can be mapped into a contiguous region is highly influenced by decisions we made for document representation, such as stop list and weighting type. For instance, suppose we have two classes of documents, one is written by a single person, the other is written by a group of people. We can imagine that for the documents that belong to the single person class, the value on the dimension "I" would be quite high, and it is extremely useful information for classification. However, if we didn't look deep into this factor and just use normal stop-words list, it is highly likely that "I" will be removed. Therefore, when we don't choose document representation wisely, the contiguity hypothesis cannot hold, making vector space classification not successful [11].

I used two vector space classification methods in my work, K-Nearest Neighbors (KNN) and Rocchio. KNN classification does not require training a model, it just sees which class is the majority of the classes of the *k* nearest neighbors of the document in question, and assigns this label to this document. Rocchio classification calculates the

center of mass of all the documents in each class and divides the problem space into regions based on centroids. Both methods will be discussed in the next sections.

2.1.3 Term frequency - inverse document frequency (tf-idf)

The values of terms in VSM can be calculated using several methods. Among them, term frequency-inverse document frequency (tf-idf) is a widely-used statistics method [2]. Tf-idf is used to describe how critical a term is to a document in a corpus. To get to know tf-idf, we will need to know what term frequency (tf) and inverse document frequency (idf) are. Suppose we have a corpus of *k* documents, tf and idf of a term *t* are defined as follows:

- tf(t,d) = the number of times *t* appears in document *d* / total number of terms in the document *d*
- idf(t) = log(k / the number of documents in the corpus in which *t* occurs at least once)

Therefore,

- Tf-idf(t,d) = tf(t, d) * idf (t)

Intuitively, if a term appears in a document very frequently and it does not occur in other documents often, it means that this term offers important information for classifying this document. For example, the term “football” appears in a document very often, but doesn’t appear in other documents, this information reveals that it is quite possible that this document belongs to the “sports” category. However, if we consider the term “report”, although it might also appear very frequently in the same document, it can be found in other documents quite often. Therefore, the term “report” cannot give us much information about this document.

2.2 Feature Selection: Filter and Wrapper Approaches

There are generally two categories of feature selection--the filter approach and the wrapper approach.

The filter approach, as shown in Fig. 2, was proposed before the wrapper approach, and it assesses the merits of features from the dataset without taking a specific machine learning algorithm into consideration. The filter approach always has the disadvantage of missing interactions between features [2], and it also ignores the possible effects on the performance of machine learning algorithm of the selected feature subsets [6].

Figure 2: The feature filter approach [6]

Realizing the disadvantage of the filter approach, Ron Kohavi and George John proposed the wrapper approach in 1997, as shown in Fig. 3. In the wrapper approach, a machine learning algorithm is used as a black box when executing the feature subset search, which means that we don't need to have the knowledge of this algorithm and we only need the interface. By using the interface, we evaluate the merits of feature subsets according to the accuracy of the induced classifier to obtain the optimal feature subset [6].

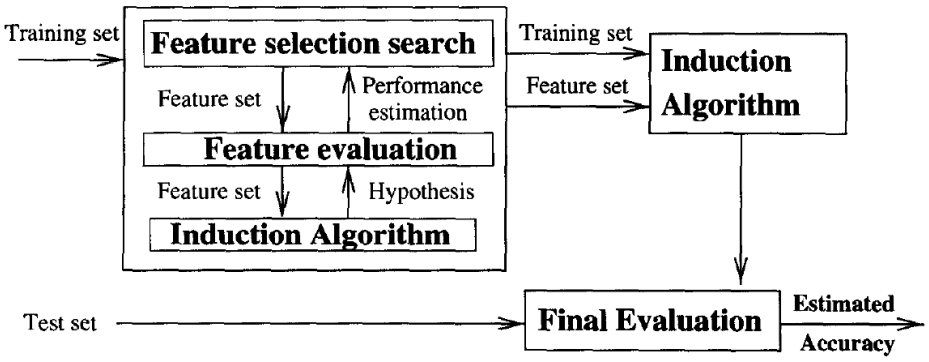

Figure 3: The wrapper approach to feature subset selection [6]

2.3 Filter Approach: Entropy and Information Gain

Information theory was developed by Shannon, and its key concept is entropy. Entropy measures the uncertainty of random variables [3].

Let X be a random variable, its uncertainty can be measured by entropy H(X) which is defined as following:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x) \quad (1)$$

where $p(x) = Pr(X=x)$ is the probability density function of X.

For two variables X and Y, when X is unknown and Y is known, the uncertainty of X given Y is the conditional entropy H(X|Y) defined as following:

$$H(X|Y) = -\sum_{x \in X, y \in Y} p(x, y) \log_2 p(x|y) \quad (2)$$

From this definition, we know that if X completely depends on Y, then H(X|Y) is zero; if X has nothing to do with Y, then H(X|Y) is H(X).

Information gain is defined as follows:

$$IG(X, Y) = H(X) - H(X|Y) = -\sum_{x \in X, y \in Y} p(x, y) \log_2 \frac{p(x,y)}{p(x)p(y)} \quad (3)$$

By definition, it is clear that information gain means the decrease in the uncertainty of X by knowing Y. Therefore, if X completely depends on Y, then IG(X, Y) is H(X); if X has nothing to do with Y, then IG(X, Y) is 0.

For feature selection tasks, a filter approach using information gain evaluates the worth of an attribute by measuring the information gain with respect to the class [10], which can be shown as:

$$IG(Class, Feature) = H(Class) - H(Class|Feature) \quad (4)$$

2.4 Wrapper Approach: Particle Swarm Optimization with KNN and Rocchio

2.4.1 Particle Swarm Optimization

For a TC problem, suppose we have a corpus that has *n* features in total, then the search space will be 2^n. Usually *n* is bigger than a thousand, and can be reached to up to nearly ten thousand. Therefore, it is impossible to use exhaustive search because the search space is so large in most situations. We can see that the search strategy can strongly influence the result of feature selection. A lot of search techniques, such as greedy algorithms, have been used in feature selection problems. However, many suffer from getting stuck in local optima [3].

Evolutionary computation algorithms are well-known for their global search ability. Compared to other evolutionary algorithms such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO) has the advantages of having fewer parameters, less computation and faster convergence. Therefore, PSO has been widely used for feature selection over recent years [3].

PSO was developed by Kennedy and Eberhart in 1995. They were inspired by observing the swarm behavior in flocks of birds, schools of fish, and swarms of bees that are thought to have Swarm Intelligence. PSO is a population-based optimization tool that can be used for a variety of optimization problems [8]. The canonical PSO was originally developed for continuous optimization problems, but it is not very practical because lots of practical engineering problems, such as feature selection, are combinatorial optimization problems. That's why Binary PSO was developed. We study at the canonical PSO first, and then discuss BPSO that was used in my work.

2.4.1.1 Canonical PSO

The canonical PSO model utilizes a population of particles. These particles are scattered in the problem space of the optimization problem in question. So how are these particles represented? They are represented by their positions and their velocities in this problem space. Like a flock of birds, these particles move iteratively through the *d*-dimension problem space based on a given rule, trying to find out the global optimal position. The definition of the global optimal position is the position in the *d*-dimension space where the fitness value is optimal. At the beginning of the search, the position of each particle is randomly initialized. In each iteration, the velocity of each particle is computed according to the PSO rule, and the position of each particle will be updated in accordance with its velocity. When a pre-defined criterion is reached, iterations stop.

For example, suppose we have an optimization problem of three variables and each variable has the domain of real number, then the problem space is a three dimension space. If we try to use PSO to solve this problem, and we plan to use ten particles in this case, we can initialize the initial position of each particle. For example, particle No.1 has initial position (1.22, 4.53, 5.78), particle No.2 has initial position (10.25, 1.36, 15.75). In each iteration, velocity of each particle is computed using the PSO rule. For example, the velocity of particle No.1 could be (-3.23, 1.48, -9.68), then its position will be updated accordingly, and the fitness of each particle will be calculated and recorded.

So what is the rule of calculating the velocity? In PSO, there are two kinds of positions that are quite important. The first is the best position ever occurred for one particle, which is called the best personal position. The best position means that this position gains the best fitness out of all the positions this particle has been to. The second

is the best position ever occurred for the whole swarm, which is called the best global position. The core idea of calculating the velocity of each particle is that the movement of each particle is always a combination of its current velocity, the trend to move to the best personal position and the trend to move to the best global position. After we get the velocity, we can then obtain the next position of this particle. Fig. 4 gives an illustration of this procedure.

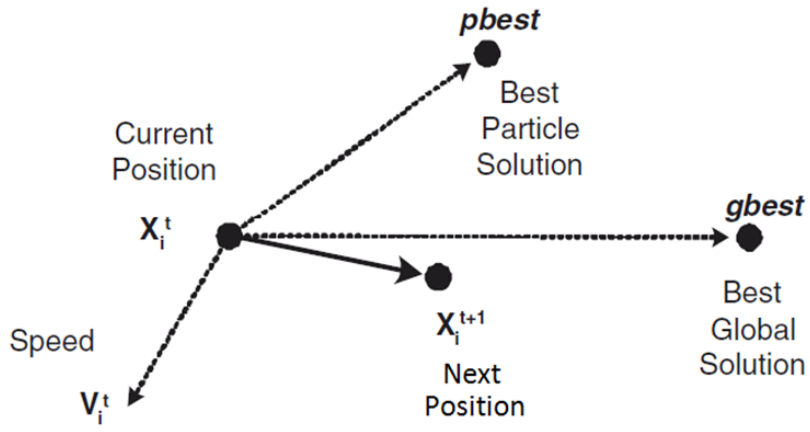

Figure 4: Illustration of how the position of a particle is updated in PSO [12]

Let us see the formula for the PSO rule. For each particle, the position is represented by a position-vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots)$($i$ is the index of the particle, j is the index of dimension); the velocity is represented by a velocity-vector $V_i = (v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots)$; the best personal position is represented by a position-vector $P_i = (p_{i1}, p_{i2}, \ldots, p_{ij}, \ldots)$; and the global best position is represented by a position-vector $G = (g_1, g_2, \ldots, g_j, \ldots)$. During iteration t, the new velocity and the new position are updated as follows:

$$V_i(t+1) = \omega * V_i(t) + c_1 * rand * (P_i - X_i(t)) + c_2 * rand * (G - X_i(t)) \quad (5)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (6)$$

In equation (5), ω is called the inertia factor, which is used to control the impact of current velocity to the next iteration's velocity. It regulates the balance between the local

and global search abilities of the particle swarm. *rand* is the random number distributed in [0,1] and is used to keep the randomness and diversity of the positions of the population. Positive constants c_1 and c_2 represent the weighting coefficients that pull the particle towards the personal best position or the global best position [8]. After we get the velocity using equation (5), we can use equation (6) to obtain the next iteration's position of each particle.

The global best position is improved round by round in PSO procedure. The end criteria of PSO are usually to reach the maximum number of iterations or to stop the iterations after several rounds of no improvement.

2.4.1.2 Binary PSO

In order to broaden the usage of PSO model, in 1997, Kennedy and Eberhart developed Binary PSO (BPSO) for discrete problems [1]. The difference between BPSO and the canonical PSO is that in each dimension, there are only two possible values -- 0 and 1, and the velocity of particle represents the probability of this particle taking 1 as its position in this dimension. Therefore, equation (5) is still applicable in BPSO only that position-vector X_i, personal best position P_i and global best position G are now vectors with only of 0 or 1. Furthermore, a sigmoid function is applied to transform V_i to the range of (0, 1) [1]. The position update equation of BPSO is defined as follows:

$$S(V_i\,(t+1)) = \frac{1}{1+e^{-Vi\,(t+1)}} \tag{7}$$

$$X_i\,(t+1) = \begin{cases} 1, \text{if } rand < S(Vi\,(t+1)) \\ 0, \text{otherwise} \end{cases} \tag{8}$$

where $S(V_i\,(t+1))$ is the sigmoid function, and *rand* is a random number distributed in [0,1].

So how do we apply BPSO to feature selection? Feature selection is in essence an optimization problem. The representation of a particle's position is an *n*-bit binary vector, where *n* is the number of features. When the bit is 1, it means that the corresponding feature is selected; when this bit is 0, it means otherwise.

2.4.2 Evaluators: KNN and Rocchio

When using BPSO as a search method to search for feature subsets in problem space, I used K-Nearest Neighbors (KNN) and Rocchio as the evaluation algorithms to calculate the accuracy of classification on each selected subset. The best global position is updated after each iteration.

The reason for selecting KNN and Rocchio is that most research of wrapper approach used KNN as learning algorithm, while Rocchio was seldom mentioned. KNN and Rocchio are both similarity based. However, the running time of Rocchio is much shorter than KNN. It is worth to compare KNN and Rocchio as learning algorithms in wrapper approach for TC tasks.

2.4.2.1 Similarity Measurements: Euclidean Distance V.S Cosine Similarity

Before I discuss KNN and Rocchio, we need to be familiar with two measurements for distance -- Euclidean distance and Cosine Similarity. Both of them are used quite often in Vector Space Model.

The Euclidean distance is the straight line distance between two vectors in vector space [20].

The Cosine similarity measures the cosine of the angle between two vectors in vector space. Compared to Euclidean distance, it is a measure of similarity of orientation rather than magnitude. Therefore, if two vectors are of the same orientation, their cosine

similarity is 1 because the cosine of 0 ° is 1; while for two vectors at 90 °, their cosine similarity is 0 [21].

When using cosine similarity, a vector is often normalized by dividing the value of each dimension by the length of the vector, thus making all the length of all vectors into 1. Therefore, for normalized vectors, the cosine is simply the dot product of two vectors because denominators have all become 1.

For sparse dimension space, cosine similarity is a better choice than Euclidean distance [11]. Therefore, in my work, cosine similarity was used in the KNN implementation, and normalization was applied to datasets before calculating cosine similarity to reduce the computation time.

2.4.2.2 K-Nearest Neighbors (KNN)

Compared to most other machine learning methods, which can be called “eager” learning methods, KNN is a kind of method that belongs to “lazy” learning or “instance - based” learning. For “eager” machine learning algorithms, such as Support Vector Machine and decision tree, prior assumptions are made about model class and the learning process is about tuning the model class parameters to the training dataset. After training, prediction will be made using the obtained model. So for “eager” methods, most of the effort is spent on the model learning phase rather than the prediction phase. While for “lazy” machine learning algorithms, such as KNN, there is no explicit assumption about the structure of a prediction function or optimization criterion. No learning effort is involved except for storing all the training instances into memory. The main endeavor is put on the prediction phase to predict labels according to the similarity between the instance in question and the other instances in the training dataset [13].

Given the definition of distance which could be Euclidean distance or cosine similarity, for 1NN, we just assign the document in question to the class of its nearest neighbor. For KNN ($k > 1$), we assign the majority class of it k nearest neighbors to the document in question. The class of the nearest neighbor can be used to break a tie if needed. The basic logic of KNN is that according to the contiguity hypothesis mentioned earlier, we assume that the document in question has the same class as the training documents surrounding its local area. Fig. 5 is a visualization of KNN with 2 classes.

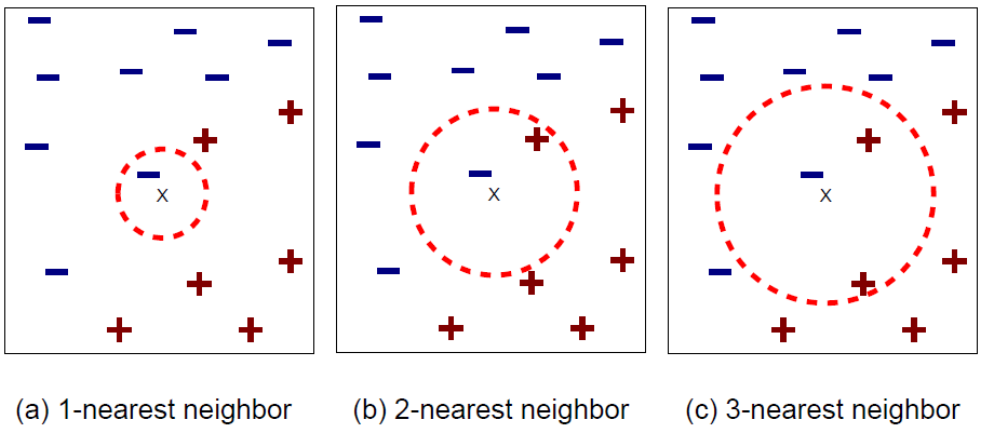

Figure 5: Visualizing k-Nearest Neighbor Classification [13]

Fig. 6 shows the decision boundaries in 1NN which are concatenated segments of the *Voronoi tessellation*. According to Wikipedia [22], "A tessellation of a flat surface is the tiling of a plane using one or more geometric shapes, called tiles, with no overlaps and no gaps." Voronoi tessellation consists of Voronoi cells, and all vectors that are closer to the object vector than other object vectors are within the scope of its corresponding Voronoi cell. For TC task, each object vector is a document vector. As we can see from Fig. 6, all the Voronoi cells are convex polygons containing its corresponding object vector. For KNN ($k > 1$), it is the same situation. The space is also divided into convex polygons [11].

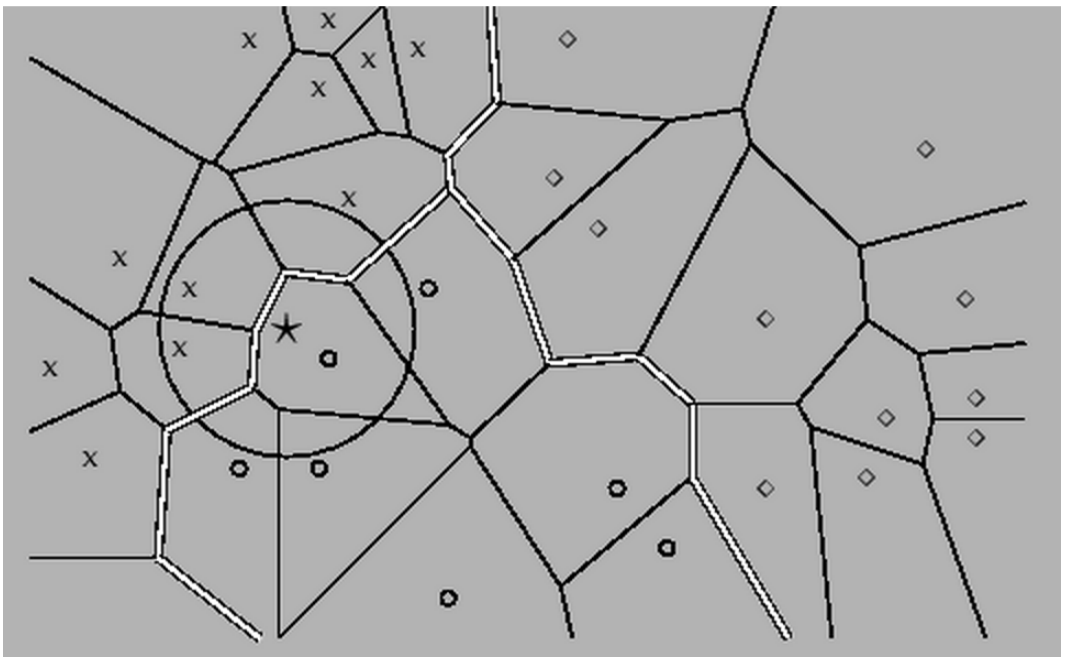

Figure 6: Voronoi tessellation and decision boundaries (double lines) in 1NN classification. The three classes are X, circle and diamond. [11]

1NN is not very robust and very sensitive to noise. The classification totally depends on the class of a single training data, so the result will be distorted if this single data is mislabeled or atypical. KNN ($k > 1$) is more robust in this sense. But we also need to note that if k is too large, the neighborhood may include instances from other classes. Usually, the parameter k is decided based on the experience about the classification problem in question. Quite often k is chosen to be an odd number to lessen the situation of ties. k =3 and k = 5 are frequently used; however, numbers between 50 and 100 are also possible options in some situations [11].

2.4.2.3 Rocchio

Rocchio classification uses centroids to define the decision boundaries. The assumption of contiguity hypothesis of Vector Space Model makes the calculation of centroid valid and applicable for Rocchio classification. The centroid of a class is defined

as the center of mass of its instance vectors. The set of points with equal distance to the centroids of two classes form the decision boundary of these two classes. For a 2-dimension classification problem, the boundary is always a line. For a space whose number of dimension is more than 2, the decision boundary is a hyperplane. Fig. 7 shows an illustration of Rocchio classification of the China, UK and Kenya problem we mentioned earlier.

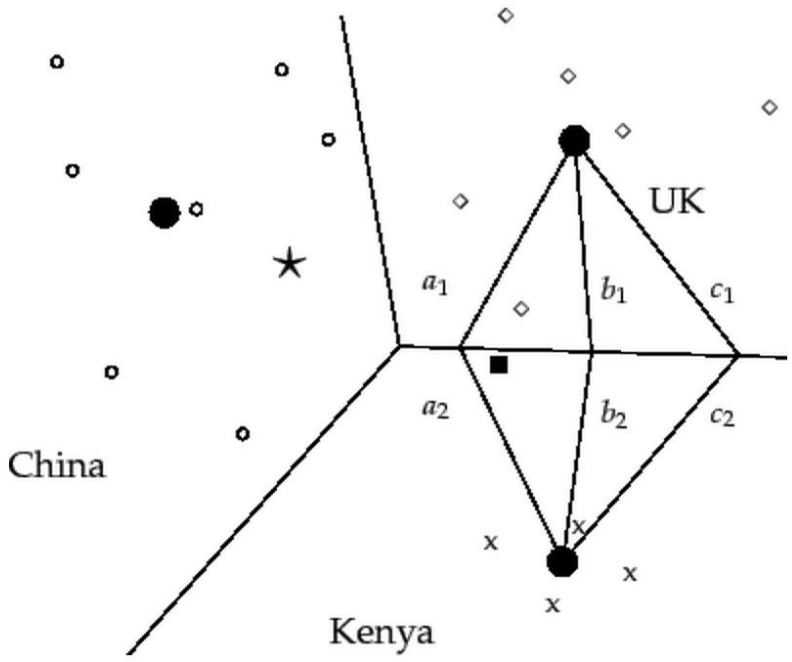

Figure 7: Rocchio classification [11]

2.5 Weka and Classifiers

2.5.1 Weka

Weka (Waikato Environment for Knowledge Analysis) is a machine learning and data mining open source software written in Java and is issued under the GNU General Public License [10]. It was developed by the University of Waikato, New Zealand and got its name from an endemic bird of New Zealand which is also called weka.

Weka supports several standard machine learning and data mining tasks such as data preprocessing, classification, clustering, feature selection and visualization. In my work, I used its classification and feature selection functions.

2.5.2 Classifiers

A classifier is an algorithm used to implement classification task. I used three classifiers in Weka, to execute classification tasks in my work:

- J48 (Weka implementation of C4.5 decision tree) -- an algorithm used to build a decision tree from a set of training data using the concept of information entropy [23]
- Na ïve Bayes -- a simple probabilistic classifier based on applying Bayes' theorem with strong (na ïve) independence assumptions between the features [24]
- Support Vector Machine (SVM) -- an algorithm that performs classification by constructing a hyperplane or set of hyperplanes in a multi-dimensional space [25]

2.5.3 K-Fold Cross Validation

The usual practice of validation is to use a holdout method, which reserves a certain amount of instances for testing and leaves the rest for training. The common holdout percentage is one-third for testing [10].

However, it is quite possible that the holdout set is unrepresentative, especially when the size of the dataset is small, which means that the amount of instances for training and testing is limited. K-fold cross validation is extremely useful in this situation. In cross validation, data is divided into a fixed number of folds, or partitions. Take 4-fold cross validation as an example. Data is split into 4 approximately equal partitions; each in turn is used for testing and the rest is used for training, i.e., one-fourth is used for testing

and three-fourth is used for training. We repeat this procedure 4 times so that every instance is used exactly once for testing in the end [10]. Fig. 8 shows the procedure of 4-fold cross validation [4].

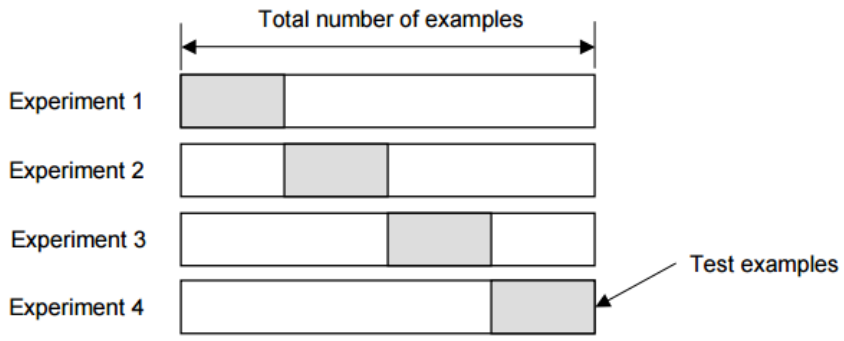

Figure 8: 4 Fold Cross Validation [17]

2.6 Measures for classification

There are several measures for assessing classification performance, and I used accuracy, precision, recall and f-measure in this work. All of them are based on confusion matrix.

2.6.1 Confusion Matrix

A confusion matrix contains information about actual and predicted classification done by a classification system [16]. Fig. 9 shows the confusion matrix for a two-class (Yes and No) classifier [18]. If an instance is actually a Yes class, and is predicted as Yes, then it is a True Positive case; if an instance is actually a No class, and is predicted as Yes, then it is a False Positive (FP) case. Similarly, if an instance is actually a Yes class, and is predicted as No, then it is a False Negative (FN) case; if an instance is actually a No class, and is predicted as No, then it is a True Positive (TP) case. Confusion matrix shows the numbers of instances which belongs TP, FP, FN and TN. Two-class confusion matrix can be easily generalized to multiclass case.

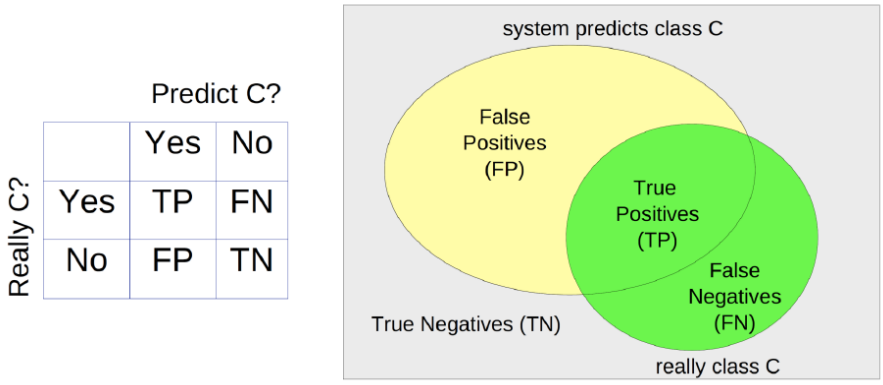

Figure 9: Confusion Matrix [18]

2.6.2 Accuracy and Error

Accuracy and error are two basic criteria for classification [18]. They are defined as follows:

$$\text{accuracy} = \frac{TP+TN}{TP+FN+FP+TN} \tag{9}$$

$$\text{error} = 1 - \text{accuracy} = \frac{FP+FN}{TP+FN+FP+TN} \tag{10}$$

However, when it comes to unbalanced dataset, accuracy and error become meaningless. Consider the case of prediction of occurrence of earthquake. Earthquake is a rare event. If a classifier always predict no, it can gain very high accuracy. However, the accuracy doesn't mean that this classifier is a good one; it only means that this dataset is quite unbalanced. In this case, FN should have higher weight. That is the reason for which we need better measurements than accuracy and error in this kind of situation [18].

2.6.3 Precision, recall and f-measure

Precision, Recall and F-measure are other three widely used measures in classification. They are defined as below:

$$\text{precision} = \frac{TP}{TP+FP} \tag{11}$$

$$\text{recall} = \frac{TP}{TP+FN} \tag{12}$$

$$\text{f- measure} = \frac{2*\text{precision}*\text{recall}}{\text{precision}+\text{recall}} \tag{13}$$

Precision means the percentage of instances classified as positive which are really positive. Recall means the ability of predicting the positive instance as positive. F-measure is the harmonic mean of precision and recall.

To get a flavor of these three measures, assume the case of a search engine. The search engine needs to classify the documents in a dataset into two classes -- relevant or irrelevant, and to return the relevant documents. Therefore, precision here means how many of the returned documents are really relevant; while recall means how many of the relevant documents are returned as relevant. Then how about f-measure? Imagine two extreme situations: 1) the search engine only returned one document and this document is indeed relevant, so the precision is 100%. But the recall is quite low because there are a lot of relevant documents were classified as irrelevant and not returned. 2) The search engine returned all the documents which mean it classified all the documents as relevant. The recall in this situation is 100%, but the precision is quite low. These two extreme situations explain why f-measure is needed -- f-measures of these two extreme situations are both low, because f-measure shows the balance between precision and recall.

In the next section, we study the implementation and experiments of this project. The implementation contains three parts: 1) generating feature subsets using raw training dataset, 2) generating classification dataset using raw testing dataset based on the

obtained feature subsets, and 3) executing Weka classification. We also discuss the classification results and come to a conclusion.

3 Experimental Study

3.1 Datasets

The datasets that I used in this work are two corpuses which are widely used in TC tasks. Ana Cardoso-Cachopo preprocessed these two corpuses and shared them online, making it much easier for us to focus on the TC task itself [15].

3.1.1 Reuters 21578 (R8 and R52)

The first corpus is Reuters 21578, which is currently the most widely used corpus for TC research. The data was originally collected and classified by Carnegie Group, Inc. and Reuters, Ltd. in 1987 [14]. In the original version of Reuters 21578, many of the documents are labeled as having no topics or have more than one topic. What's more, because some classes only have very few documents, the class distribution for the original version is much skewed. Therefore, two sub-collections are usually used for TC tasks, which are called R8 and R52. Both of them are formed by single-labeled documents. R8 is the set of 8 classes with the highest number of positive training examples, while R52 is the set of 52 classes with the highest number of positive training examples and with at least one testing examples [15]. Fig. 10 and Fig. 11 show the distributions of documents per class of R8 and R52.

R8			
Class	**# train docs**	**# test docs**	**Total # docs**
acq	1596	696	2292
crude	253	121	374
earn	2840	1083	3923
grain	41	10	51
interest	190	81	271
money-fx	206	87	293
ship	108	36	144
trade	251	75	326
Total	**5485**	**2189**	**7674**

Figure 10: Reuters 21578—R8 [15]

R52			
Class	**# train docs**	**# test docs**	**Total # docs**
acq	1596	696	2292
alum	31	19	50
bop	22	9	31
carcass	6	5	11
cocoa	46	15	61
coffee	90	22	112
copper	31	13	44
cotton	15	9	24
cpi	54	17	71
cpu	3	1	4
crude	253	121	374
dlr	3	3	6
earn	2840	1083	3923
fuel	4	7	11
gas	10	8	18
gnp	58	15	73
gold	70	20	90
grain	41	10	51
heat	6	4	10
housing	15	2	17
income	7	4	11
instal-debt	5	1	6
interest	190	81	271
ipi	33	11	44
iron-steel	26	12	38
jet	2	1	3
jobs	37	12	49
lead	4	4	8
lei	11	3	14
livestock	13	5	18
lumber	7	4	11
meal-feed	6	1	7
money-fx	206	87	293
money-supply	123	28	151
nat-gas	24	12	36
nickel	3	1	4
orange	13	9	22
pet-chem	13	6	19
platinum	1	2	3
potato	2	3	5
reserves	37	12	49
retail	19	1	20
rubber	31	9	40
ship	108	36	144
strategic-metal	9	6	15
sugar	97	25	122
tea	2	3	5
tin	17	10	27
trade	251	75	326
veg-oil	19	11	30
wpi	14	9	23
zinc	8	5	13
Total	**6532**	**2568**	**9100**

Figure 11: Reuters 21578—R52 [15]

3.1.2 WebKB

The second corpus is WebKB, which are webpages collected by the World Wide Knowledge Base project of the CMU text learning group [15]. These webpages were collected from the computer science department of several well-known universities in 1997 and were labeled into seven classes: Student, Faculty, Staff, Department, Course, Project and Other. However, Ana [15] chose to discard the classes Department and Staff because there were only a few pages from each university. She also got rid of the class Other because pages were very different among this class [15]. Fig. 12 shows the distribution of classes of WebKB.

WebKB			
Class	**# train docs**	**# test docs**	**Total # docs**
project	336	168	504
course	620	310	930
faculty	750	374	1124
student	1097	544	1641
Total	**2803**	**1396**	**4199**

Figure 12: WebKB [15]

3.1.3 File Description

All of the dataset files are text files. Each line of a text file represents one document. Each document consists of its class and its terms -- the first word of each line represents this document's class, then a TAB character, then a sequence of "words" delimited by spaces which represent the terms contained in this document [15]. A snapshot of the text file is shown in Fig. 13.

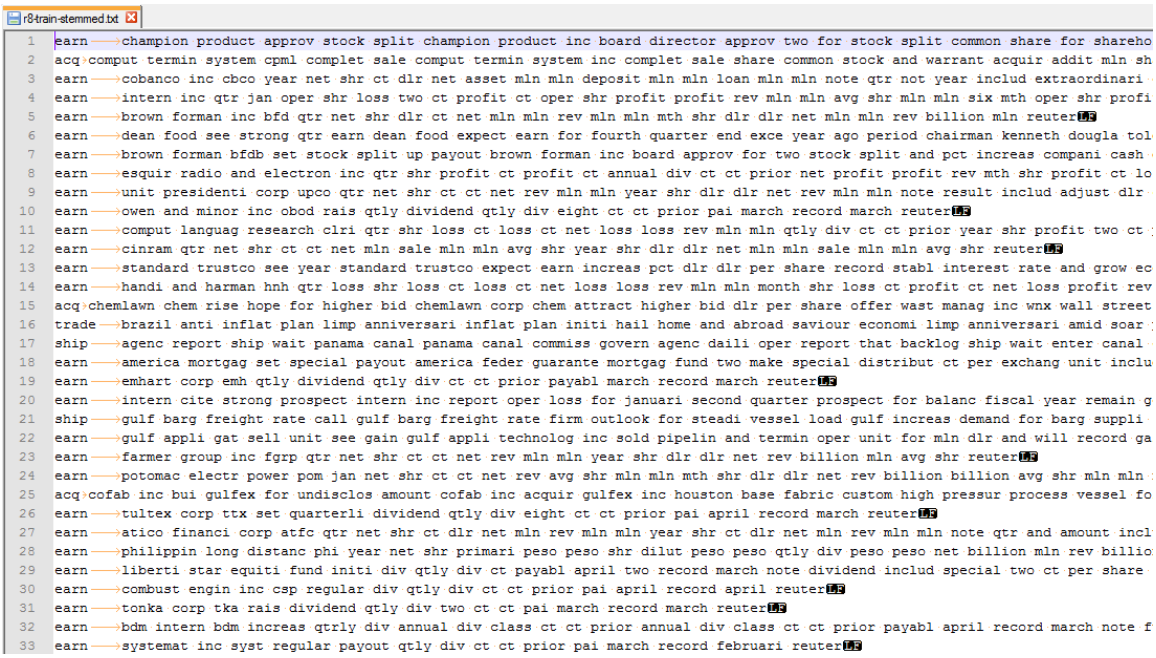

Figure 13: Pre-processed dataset of Reuters 21578—R8

3.1.4 Pre-processing

As mentioned earlier, the datasets I used are already pre-processed. According to Ana Cardoso-Cachopo, the following pre-processing has been done [15]:

1. Substitute TAB, NEWLINE and RETURN characters by SPACE.
2. Keep only letters (that is, turn punctuation, numbers, etc. into SPACES).
3. Turn all letters to lowercase.
4. Substitute multiple SPACES by a single SPACE.
5. Add the title/subject of each document in the beginning of the document's text.
6. Remove words that are less than 3 characters long. For example, removing "he" but keeping "him".
7. Remove the 524 SMART stopwords. Some of them had already been removed, because they were shorter than 3 characters.

8. Apply Porter's Stemmer to the remaining words.

3.2 Implementation

I determined to use 5 as *k* in this work after trying values 3 and 5 because k=5 got better results; therefore BPSO-KNN will be called BPSO-5NN in the following paragraphs.

Four software components were implemented in Java: VSM dataset generator, BPSO-5NN algorithms, BPSO-Rocchio algorithms, and Feature filter.

The whole data pipeline of my implementation is illustrated in Fig. 14. The implementation of this project consists of three steps:

1. Generate Feature Selection (FS) subsets using raw training dataset as shown in Fig. 14. The output of this step are three feature subsets -- Information Gain FS subset, BPSO-5NN FS subset and BPSO-Rocchio FS subset, which correspond to {3}, {4} and {5} in Fig. 14:
    a. Data transformation: implement the Vector Space Model (VSM) dataset generator using Java programming language to transform the pre-processed Raw Training datasets {1} ({1} in Fig. 14) into VSM Training dataset {2} with the tf-idf weighting.
    b. Filter approach: use software tool Weka to execute the first round of feature selection using Information Gain filter method on {2} to get Information Gain FS subset {3}.
    c. Wrapper approaches: implement BPSO-KNN and BPSO-Rocchio algorithm using Java programming language, and execute the second round of feature selection using BPSO-KNN and BPSO-Rocchio

relatively to get BPSO-5NN FS subset {4} and BPSO-Rocchio FS subset {5}.

2. Generating four types of dataset used in classification using raw testing dataset as shown in Fig. 14. In this work, four types of datasets were used in the final classification tasks. They are Original VSM testing dataset {7}, Information Gain VSM testing dataset {8}, BPSO-5NN VSM testing dataset {9}, BPSO-Rocchio VSM testing dataset {10}:
    a. Type 1: Use VSM dataset generator to transform Raw Testing dataset {6} to obtain {7}
    b. Type 2: Use feature filter to filter {6} based on {3}, then use VSM dataset generator to obtain {8}
    c. Type 3: Use feature filter to filter {6} based on {4}, then use VSM dataset generator to obtain {9}
    d. Type 4: Use feature filter to filter {6} based on {5}, then use VSM dataset generator to obtain {10}
3. Executing Weka Classification. Use three classifiers from Weka (J48, Naïve Bayes, LibSVM) to run classification on four types of dataset, which are {7}, {8}, {9} and {10}. The classification tasks were executed using 10-fold cross validation.

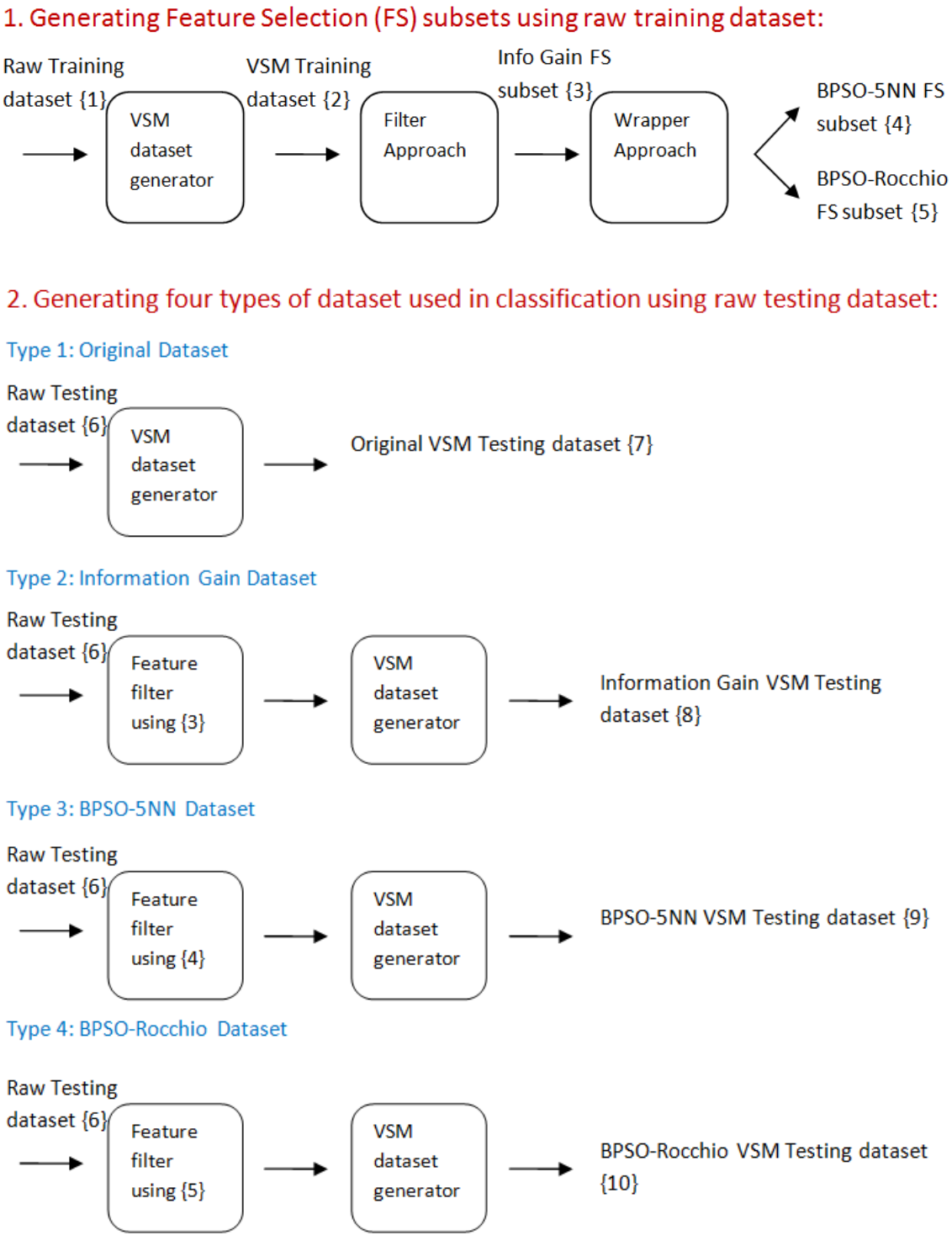

Figure 14: Data pipeline in experimental study

3.2.1 VSM dataset generator

In order to transform the pre-processed Raw Training dataset and Filtered Testing dataset into SVM dataset, a VSM dataset generator was implemented in Java. The main task of this implementation is to calculate tf-idf of each term in each document. First, I count the total number of documents and calculate the number of documents that a specific term has occurred for each term to get idf. Second, I go through each document to calculate the number of appearance of a specific term and divide it by total number of terms in the document to get tf. By multiplying idf and tf, I get tf-idf weighting of each term in each document and outputted them into a new text file to make the VSM dataset. A comma was used to separate class and terms. See Fig. 15 for the format of the obtained VSM datasets.

Figure 15: VSM dataset of Reuters 21578—R8

3.2.2 Filter Approach

In this step, I used Weka to execute the first round of feature selection using Information Gain filter approach.

3.2.2.1 Information Gain Attribute Evaluator and Ranker

Weka provides both filter and wrapper approaches in its feature selection library. I used the Information Gain Attribute Evaluator which is a filter method to carry out the first round feature selection. For a filter method such as Information Gain, its evaluation is not based on a feature subset but a single feature. Therefore, the search method can only be a rank method rather than the feature subset search method. I chose the top *n* attributes whose values are not zero as the selected feature subset in this step. Fig. 16 shows the snapshot of Weka Information Gain Attribute Evaluator.

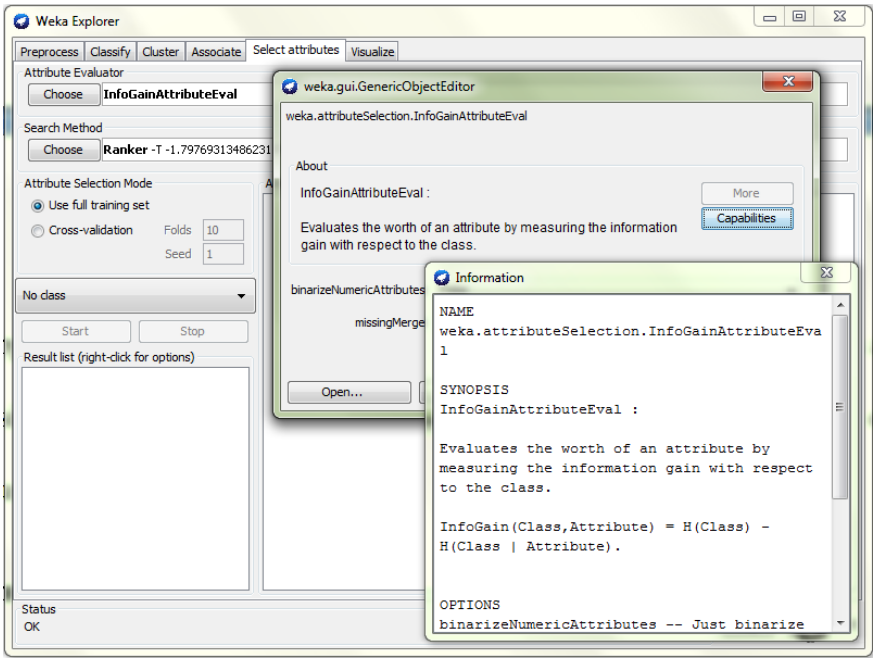

Figure 16: Weka: Information Gain Attribute Evaluator

3.2.3 Wrapper Approach

Two wrapper approaches were used in this work -- BPSO-5NN and BPSO-Rocchio. Both used BPSO as search method. I implemented this part in Java as well. The algorithm of BPSO was first implemented, leaving an interface for the fitness evaluator. 5NN and Rocchio were then implemented separately fitting the BPSO interface so that they can be used to calculate the classification accuracy when a specific feature subset needs to be evaluated.

3.2.3.1 BPSO procedure

The step by step view of the whole BPSO implementation procedure is explained in what follows:

1. A population of particles was created in an N-dimension feature space. Each particle has three vectors -- its current position vector, its current velocity vector and its personal best position vector. There is another vector for global best position. At the beginning, the current position vector and the current velocity vector of each particle were initialized randomly, and the personal best position was initialized with a value equal to its corresponding current position vector. The global best position was initialized by selecting the position with the best fitness value within all the particles' current positions.
2. Every iteration was carried out as follows:
    a. Calculate the fitness of each particle using either 5NN or Rocchio
    b. Update the personal best position of each particle
    c. Update the global best position of the whole swarm

d. Update the velocity of each particle using BPSO formula

e. Update the position of each particle using the updated velocity

3. Terminate iterations when the number of iteration was reached.

3.2.3.2 Classification Accuracy

The classification accuracy of a specific position that represents a specific feature subset was calculated using the procedure mention in [2] as follows:

1. Assume a variable C=0. For each document vector in the dataset:
    a. Calculate the distance of this document vector to all the other document vectors. For 5NN, cosine similarity is used. For Rocchio, Euclidean distance is used.
    b. Classify this document vector. For 5NN, label it with the majority class of its 5 nearest neighbors. For Rocchio, label it with the class whose centroid is the nearest to it.
2. If the prediction class is the same as the known class of this document vector, increase variable C by 1.

After all the document vectors were processed, the classification accuracy of this specific position was calculated as C divided by the number of documents in the dataset.

3.2.3.3 Fitness function

The fitness value of each position is not just the classification accuracy. Because it is a feature selection task, the number of feature selected is also a critical factor to be considered. The fewer the features, the better the fitness value. Therefore, by taking both classification accuracy and number of feature selected into account, as suggested in [2], the fitness function is defined as follows:

$$\text{Fitness} = \alpha * \text{Classification Accuracy} + \beta * (N-T)/N \quad (14)$$

where

1. N is the total number of features. T is the number of feature selected. Therefore, the smaller T is, the larger (N-T)/N will be.
2. α and β are used to define the importance of classification Accuracy and feature subset size. The sum of α and β is 1.

3.2.3.4 BPSO Parameters

After trying several different combinations of values, BPSO parameters were determined and set as follows:

1. Inertia weight ω = 1.2, *c1* = 1.49 and *c2* = 1.49
2. Termination criterion is a maximum of 50 iterations
3. Swarm size is 16 particles
4. α and β in fitness function was set to be 0.85 and 0.15, respectively

3.3 Result and Discussion

Table 1, 2 and 3 show the classification results of the classifiers of J48, Naïve Bayes and LibVSM using Weka. The classification results including four metrics: classification accuracy, precision, recall and f-measure. The results show the comparison among four types of dataset as mentioned before. The 2nd to 5th Column of Table 1, 2 and 3 correspond to type 1 to 4 in Fig. 14, respectively.

3.3.1 Reuters 21578--R8

Table 1 shows the results of Reuters 21578--R8 dataset. There are three things that are noticeable in this result: 1) after two rounds of feature selection, the number of

features was reduced from 8576 to around 800. Even though it has been cut down to one tenth, the classification performance keeps basically the same. For Naïve Bayes classification, BPSO-5NN is even 1% better than the original dataset. 2) The number of features narrowed down from 1370 to around 800 for second round, which is a significant dimension reduction, and the classification performance also keeps basically the same. 3) The result of each metrics shows that BPSO-5NN and BPSO-Rocchio have almost the same performance. For J48 classification, the BPSO-Rocchio has even slightly better performance than BPSO-5NN.

Table 1: Classification Result of Reuters 21578--R8

Classifier & Metrics		**Original Dataset** Instances: 2189 Attributes: 8576	**Information Gain** Instances: 2189 Attributes: 1370	**BPSO-5NN** Instances: 2189 Attributes: 791	**BPSO-Rocchio** Instances: 2189 Attributes: 855
J48	Accuracy	0.901325	0.902695	0.893102	0.901782
	Precision	0.899	0.9	0.889	0.897
	Recall	0.901	0.903	0.893	0.902
	F-Measure	0.9	0.901	0.891	0.899
Naïve Bayes	Accuracy	0.875286	0.885793	0.889447	0.871631
	Precision	0.913	0.917	0.913	0.911
	Recall	0.875	0.886	0.889	0.872
	F-Measure	0.892	0.899	0.9	0.889
Lib SVM	Accuracy	0.942896	0.946094	0.940155	0.920512
	Precision	0.942	0.945	0.939	0.917
	Recall	0.943	0.946	0.94	0.921
	F-Measure	0.941	0.945	0.939	0.918

3.3.2 Reuters 21578--R52

Table 2 shows the results of Reuters 21578--R52 dataset. 1) Compared to R8 dataset, the dimension reduction of this R52 dataset was even more noticeable -- from 9731 to 300, while the classification performance keeps basically the same. 2) For this dataset, the two PSO based algorithms gained nearly the same amount of features, and BPSO-5NN has a little bit better metric results than BPSO-Rocchio by 1% for each classifier.

Table 2: Classification Result of Reuters 21578—R52

Classifier & Metrics		**Original Dataset** Instances: 2568 Attributes: 9731	**Information Gain** Instances: 2568 Attributes: 431	**BPSO-5NN** Instances: 2568 Attributes: 300	**BPSO-Rocchio** Instances: 2568 Attributes: 301
J48	Accuracy	0.78271	0.779984	0.778816	0.765187
	Precision	0.74	0.731	0.71	0.707
	Recall	0.783	0.78	0.779	0.765
	F-Measure	0.757	0.75	0.739	0.732
Naïve Bayes	Accuracy	0.765576	0.818925	0.807243	0.780374
	Precision	0.827	0.847	0.835	0.825
	Recall	0.766	0.819	0.807	0.78
	F-Measure	0.787	0.828	0.817	0.798
Lib SVM	Accuracy	0.872274	0.896807	0.879673	0.867601
	Precision	0.871	0.887	0.868	0.848
	Recall	0.872	0.897	0.88	0.868
	F-Measure	0.862	0.888	0.865	0.851

3.3.3 WebKB

Table 3 shows the results of Reuters WebKB dataset. 1) For this dataset, two rounds of feature selection reduced the number of features from 4799 to around 700. The classification performance keeps basically the same, while the two PSO based algorithms show better result for LibSVM classifier. 2) Two PSO based algorithm performs basically the same. BPSO-Rocchio has 100 features fewer features, and achieved 1% better results for LibSVM compared to its competitor.

Table 3: Classification Result of WebKB

Classifier & Metrics		**Original Dataset** Instances: 1396 Attributes: 4799	**Information Gain** Instances: 1396 Attributes: 1115	**BPSO-5NN** Instances: 1396 Attributes: 755	**BPSO-Rocchio** Instances: 1396 Attributes: 610
J48	Accuracy	0.777937	0.77149	0.768625	0.765759
	Precision	0.777	0.768	0.766	0.764
	Recall	0.778	0.771	0.769	0.766
	F-Measure	0.776	0.768	0.764	0.761
Naïve Bayes	Accuracy	0.765759	0.776504	0.757163	0.742837
	Precision	0.77	0.788	0.771	0.759
	Recall	0.766	0.777	0.757	0.743
	F-Measure	0.766	0.779	0.759	0.747
Lib SVM	Accuracy	0.793696	0.82235	0.81447	0.823066
	Precision	0.794	0.822	0.812	0.822
	Recall	0.794	0.822	0.814	0.823
	F-Measure	0.79	0.82	0.812	0.822

3.3.4 Analysis of running times

Table 4 shows the average computation time of 50 rounds of BPSO iteration for every dataset for BPSO-5NN and BPSO-Rocchio. The ratio between the two average iteration times is also computed to highlight the difference in computation time between the two approaches.

Table 4: Comparison of Computation Time between BPSO-5NN and BPSO-Rocchio

	Reuters 21578--R8 (8 classes)	Reuters 21578--R52 (52 classes)	WebKB (4 classes)
BPSO-5NN	13.24 min	8.956 min	2.054 min
BPSO-Rocchio	37.12 sec	1.962 min	5.046 sec
Ratio	21.4	4.6	24.4

Suppose the dataset has *m* classes and *n* instances.

For BPSO-5NN, the computation time of each iteration mainly consists of two parts: 1) Calculating cosine similarity between each instance, whose time complexity is $O(n^2)$. 2) Calculating the top 5 nearest neighbor which takes O(n). Therefore the time complexity of BPSO-5NN is $O(n^2)$.

For BPSO-Rocchio, the computation time of each iteration mainly consists of two parts as well: 1) Calculating the Euclidean centroids of all the instances of *m* classes which takes O(n). 2) Calculating the Euclidean distance of each instance to these *m* centroids and find out the nearest one which takes O(mn). Therefore the time complexity of BPSO-Rocchio is O(n).

From the table, we can see that for WebKB and Reuters 21578--R8, average iteration time of BPSO-5NN is over 20 times to that of BPSO-Rocchio, but this ratio is only 4.6 when it comes to Reuters 21578--R52. The 20 times difference fits the difference of computation time complexity between these two algorithms. For Reuters 21578--R52, it has 52 classes, which means that for each instance, 52 Euclidean distances need to be counted in BPSO-Rocchio. Therefore, compared to the other two dataset which have much fewer classes, the time difference is not so significant.

3.3.5 Analysis of classification results and running times

Table 5 shows the average classification accuracy of three classifiers for each dataset.

Table 5: Average of Classification Accuracy Summary

	Original Dataset	**Information Gain**	**BPSO-5NN**	**BPSO-Rocchio**
Reuters 21578--R8	0.909091 (8576 features)	0.911527 (1370 features)	0.907568 (791 features)	0.897975 (855 features)
Reuters 21578--R52	0.806853 (9731 features)	0.831905 (431 features)	0.821911 (300 features)	0.804387 (301 features)
WebKB	0.779131 (4799 features)	0.790115 (1115 features)	0.780086 (755 features)	0.777221 (610 features)

First of all, for all the three datasets, after two rounds of feature selection, the feature numbers were all greatly reduced. The classification accuracy of both PSO based algorithms are about the same compared to the accuracy using original dataset. BPSO-5NN has better results than the original dataset for Reuters 21578--R52 and WebKB. This shows that this integrated feature selection method combining both the filter and wrapper method is successful -- greatly reduced dimension and good classification results were achieved in this work.

Second, it is noticeable that the major dimension reduction happens in the first round of feature selection which is the filter approach phase, and the classification accuracies of the Information Gain phase are the highest for all three corpuses. However, because the major purpose of this work is to compare the feature selection performance

between BPSO-5NN and BPSO-Rocchio, and as previously mentioned, by performing the filter approach feature selection first, it is much easier to run the two PSO based wrapper algorithms. Therefore, more emphasize should be put on the results obtained from BPSO-5NN and BPSO-Rocchio dataset.

Third, as we can see from Table 5, the overall classification performance of BPSO-5NN is about 1% to 2% better than BPSO-Rocchio, but the computation time difference between this two is quite huge -- for datasets which have less than 10 classes, the ratio of computation time between the two PSO based wrapper algorithms can be up to 20 times. Therefore, for most situations, BPSO-Rocchio is a better choice especially when computation time is a factor.

3.3.6 Conclusion and Future Work

In this work, I implemented a two-round feature selection for text categorization combining both the filter and wrapper approaches. The filter approach is Information gain method. The two wrapper approaches are BPSO based 5NN and Rocchio. The comparison of the classification results was done in two levels: 1) Comparison between the original dataset and the two-round feature selected dataset; 2) Comparison between the two wrapper approaches -- BPSO-5NN and BPSO-Rocchio. The result shows: 1) this two-round feature selection implementation is successful -- the number of features was substantially reduced and the classification performance was slightly improved; 2) BPSO-Rocchio has much shorter computation time and comparable classification accuracy compared to BPSO-5NN.

For future work, more corpuses can be tried using this implementation in distributed systems. Corpuses like 20 Newsgroups and Cade [15] were not used in this

study due to the extremely large size of their dataset. They should be tried in a distributed system in the two-round feature selections and used in the final classification task to verify the results reported in this work.

References

[1] B. Xue, M. Zhang and W. N. Browne, "Single feature ranking and binary particle swarm optimisation based feature subset ranking for feature selection," in *Proceedings of the Thirty-Fifth Australasian Computer Science Conference* - Volume 122, Melbourne, Australia, 2012, pp. 27-36.

[2] H. K. Chantar and D. W. Corne, "Feature subset selection for Arabic document categorization using BPSO-KNN," in *Nature and Biologically Inspired Computing* (NaBIC), 2011 Third World Congress on, 2011, pp. 546-551.

[3] L. Cervante, Bing Xue, Mengjie Zhang and Lin Shang, "Binary particle swarm optimisation for feature selection: A filter based approach," in *Evolutionary Computation* (CEC), 2012 IEEE Congress on, 2012, pp. 1-8.

[4] Yaohong Jin, Wen Xiong and Cong Wang, "Feature selection for chinese text categorization based on improved particle swarm optimization," in *Natural Language Processing and Knowledge Engineering* (NLP-KE), 2010 International Conference on, 2010, pp. 1-6.

[5] A. Cardoso-Cachopo, "Improving Methods for Single-label Text Categorization," PhD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[6] R. Kohavi and G. H. John, "Wrappers for Feature Subset Selection," *Artif. Intell.*, vol. 97, pp. 273-324, dec, 1997.

[7] Xing Liu and Lin Shang, "A fast wrapper feature subset selection method based on binary particle swarm optimization," in *Evolutionary Computation* (CEC), 2013 IEEE Congress on, 2013, pp. 3347-3353.

[8] A. Abraham, H. Guo and H. Liu, "Swarm Intelligence: Foundations, Perspectives and Applications," vol. 26, pp. 3-25, 2006.

[9] Y. Bengio and Y. Grandvalet, "No Unbiased Estimator of the Variance of K-Fold Cross-Validation," J.Mach.Learn.Res., vol. 5, pp. 1089-1105, dec, 2004.

[10] I. Witten, E. Frank and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed, Amsterdam: Morgan Kaufmann Publishers, 2011.

[11] C. Manning, P. Raghavan & H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008. Retrieved April 2, 2015 from http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

[12] F. Hembecker, H. Lopes and W. Godoy Jr., "Particle Swarm Optimization for the Multidimensional Knapsack Problem," vol. 4431, pp. 358-365, 2007.

[13] T. Thanh, “Big Data Mining: Lazy/Instance-based Learning”, Retrieved April 2, 2015 from https://www.dropbox.com/s/nqysrn8s89rt18n/lazy%20learning.pdf

[14] D. Lewis, “Reuters-21578”, Retrieved April 2, 2015 from http://www.daviddlewis.com/resources/testcollections/reuters21578/

[15] A. Cardoso-Cachopo, “Datasets for single-label text categorization”, Retrieved April 3, 2015 from http://web.ist.utl.pt/acardoso/datasets/

[16] H. Hamilton, “Confusion Matrix”, Retrieved April 3, 2015 from http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html

[17] R. Gutierrez-Osuna, “Lecture 13: Validation”, Retrieved April 3, 2015 from http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf

[18] T. Thanh, “Big Data Mining: Data Mining Process: A Quick Run-Through”, Retrieved April 2, 2015 from https://www.dropbox.com/s/kwn9fvo3qpcwtab/process.pdf

[19] Wikipedia, “Vector Space Model” entry, Retrieved April 2, 2015 from http://en.wikipedia.org/wiki/Vector_space_model

[20] Wikipedia, “Euclidean distance” entry, Retrieved April 4, 2015 from http://en.wikipedia.org/wiki/Euclidean_distance

[21] Wikipedia, “Cosine similarity” entry, Retrieved April 4, 2015 from http://en.wikipedia.org/wiki/Cosine_similarity

[22] Wikipedia, “Tessellation” entry, Retrieved April 6, 2015 from http://en.wikipedia.org/wiki/Tessellation

[23] Wikipedia, “C4.5 algorithm” entry, Retrieved April 5, 2015 from http://en.wikipedia.org/wiki/C4.5_algorithm

[24] Wikipedia, “Naive Bayes classifier” entry, Retrieved April 5, 2015 from http://en.wikipedia.org/wiki/Naive_Bayes_classifier

[25] Wikipedia, “Support vector machine” entry, Retrieved April 5, 2015 from http://en.wikipedia.org/wiki/Support_vector_machine

Appendix

The source code of this writing project can be found in the following link in Github: https://github.com/shwu2012/CS298