

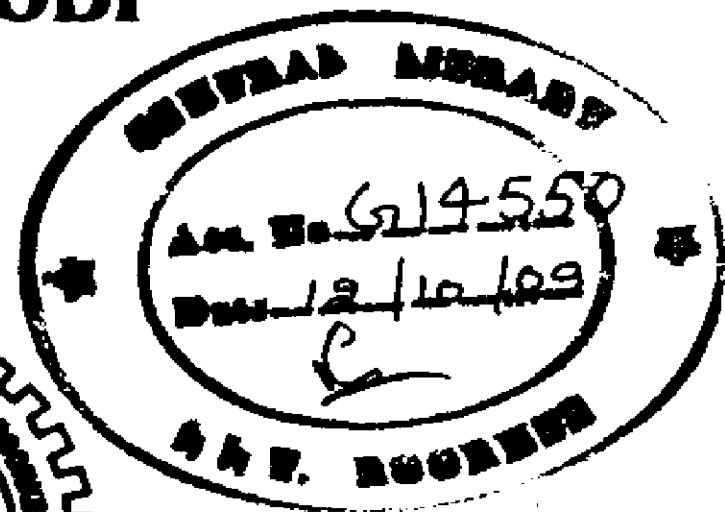
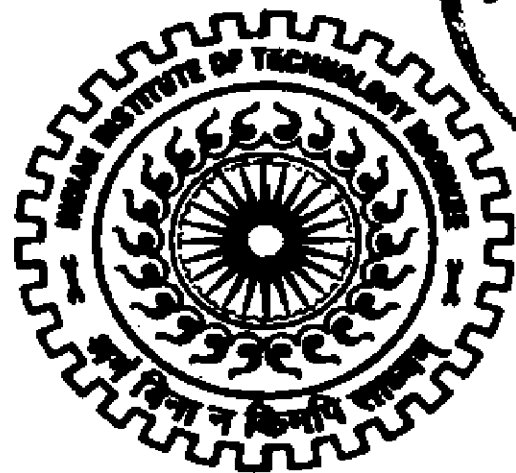
INCREMENTAL APPROACH FOR TEXT CLASSIFICATION

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree
of*
**MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY**

By

SHWETA MODI



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2009**

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled **“INCREMENTAL APPROACH FOR TEXT CLASSIFICATION”** towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology in Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2008 to June 2009, under the guidance of **Dr. Durga Toshniwal, Assistant Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 22/06/09

Place: Roorkee



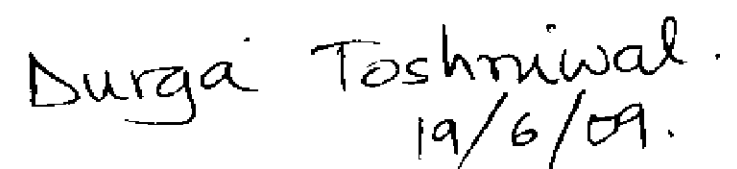
(SHWETA MODI)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 19/06/09

Place: Roorkee



(Dr. Durga Toshniwal)

Assistant Professor

E & CE DEPT.

IIT Roorkee – 247 667

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to my guide **Dr. Durga Toshniwal**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for her able guidance, regular source of encouragement and assistance throughout this dissertation work. It is her vision and insight that inspired me to carry out my dissertation in the upcoming field of 'Text Document Mining'. I would state that the dissertation work would not have been in the present shape without her umpteen guidance and I consider myself fortunate to have done my dissertation under her.

I also extend my sincere thanks to **Dr. S. N. Sinha**, Professor and Head of the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for providing facilities for the work.

I also wish to thank my brother and all my friends for their valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my parents for everything that they have given to me. I thank them for the sacrifices they have made so that I could grow up in a learning environment. My family always stood by me in everything I have done, providing constant support, encouragement and love.

SHWETA MODI

ABSTRACT

Text documents are generated from various businesses, research, government and other organizations as they store data in digital form. Classification is a supervised grouping of data. Text classification is a method of associating one (or more) predefined categories to a particular document. In many applications the data keeps getting generated over time. Under these circumstances, the traditional text classification methods may be incapable to deal with. Therefore, incremental classification techniques are required in such cases.

In this thesis, we propose an algorithm for incremental text classification. The text documents have been preprocessed before applying classification techniques to them. Preprocessing, involves stopwords removal and stemming of words. The porter's stemming analyzer has been used for the purpose of word stemming. After stopwords removal and stemming of words, the documents are converted into vectors on the basis of term frequencies and inverse document frequencies. To obtain the class labels for the classifier, we have applied k-means clustering to the dataset. The clustering also results into extraction of relevant terms for the dictionary. With each increment, new terms get added to the dictionary. The newly added terms are assigned unit weights whereas the weight for the terms in the dictionary is reduced. This process continues as more and more data keeps getting generated. The idea of weights is used to show the incremental evolution of dictionary. The proposed algorithm is applied on real case data collected from Google sports news collected over fixed interval of 1 month.

CONTENTS

CANDIDATE’S DECLARATION	i
ACKNOWLEDGEMENTS	ii
ABSTRACT	iii
LIST OF FIGURES	vi
1. Introduction	1
1.1 Introduction	1
1.2 Motivation for work	2
1.3 Problem Statement	5
1.4 Organization of the Thesis	5
2. Literature Review	6
2.1 Preprocessing	6
2.1.1 Document Representation	6
2.1.2 Feature Selection	8
2.1.3 Dimension Reduction	12
2.2 Text Classification	15
2.3 Research Gaps	19
3. Proposed Work	20
3.1 Overall Architecture for Proposed Algorithm	20
3.2 Preprocessing	21

3.3	Clustering and Term Extraction	24
3.4	Term Dictionary	25
3.5	TF-IDF Vector Generation	26
3.6	Classifier Model Generation	26
3.7	Classifier	27
4.	Implementation Details	28
4.1	Code Platform	28
4.2	Dataset Description	28
4.3	Class Hierarchy	28
4.4	Training Procedure	29
4.5	Testing Procedure	34
5.	Result and Discussion	37
5.1	Results and Discussion	37
5.2	Analysis	41
6.	Conclusion	42
6.1	Conclusion	42
6.2	Suggestions for further work	43
	REFERENCES	44
	APPENDIX A: SOURCE CODE LISTING	47

List Of Figures

Figure No.	Description	Page No.
1.1	Process of Text Mining	2
3.1	The overall architecture for proposed algorithm	20
3.2	Flowchart for Preprocessing Text Documents	21
3.3	Input document to preprocessing module.....	22
3.4	Output Term frequency document of preprocessing module.....	23
3.5	Output Dictionary of preprocessing module.....	23
3.6	Flowchart for Clustering Module.....	24
3.7	Output centroids of clustering module.....	25
3.8	TF-IDF Vector Generation	26
3.9	Classifier Model Generation and Result Validation	27
3.10	Classifier	27
4.1	Class hierarchy of training module.....	29
4.2	Class hierarchy of testing module.....	29
5.1	Entropy weighted sum on timestamp4 dataset	37
5.2	Classification Accuracy with static dictionary	38
5.3	Classification Accuracy with dynamic dictionary.....	39
5.4	Classification Accuracy with weighted terms in dynamic dictionary	39
5.5	Linear Decrement of Term Weights by 5%, 10% and 15%	40

CHAPTER 1

INTRODUCTION

1.1 Introduction

Data mining or knowledge discovery from data is a process of extracting or mining knowledge from large amount of data. It extracts the hidden, unknown but very useful information from the abundant, incomplete, noisy and stochastic data obtained over time [1]. Text mining is analogous to data mining as it extracts useful information from data sources through the identification and exploration of interesting patterns [2].

A substantial portion of the available information is stored in text databases which consist of large collections of electronic documents like news articles, e-mail messages, research papers, e-books, digital libraries and web pages. Nowadays most of the information in research, industry, business, government and other organization are stored as electronic documents.

Text mining consists of four main areas:

- Pre-processing task include all the processes required to prepare data for text mining system's core mining operations.
- Core mining operations are most important in text mining system which includes pattern discovery, trend analysis and incremental knowledge discovery algorithms.
- Presentation layer components include GUI and pattern browsing functionality as well as access to query language. Visualization tools and user-facing query editors and optimizers also come under this category.
- Refinement technique or post-processing include methods that filter redundant information and cluster closely related data in a given text mining system, to

represent a full, comprehensive suite of suppression, ordering, pruning, generalization and clustering approaches aimed at discovering optimization [2].

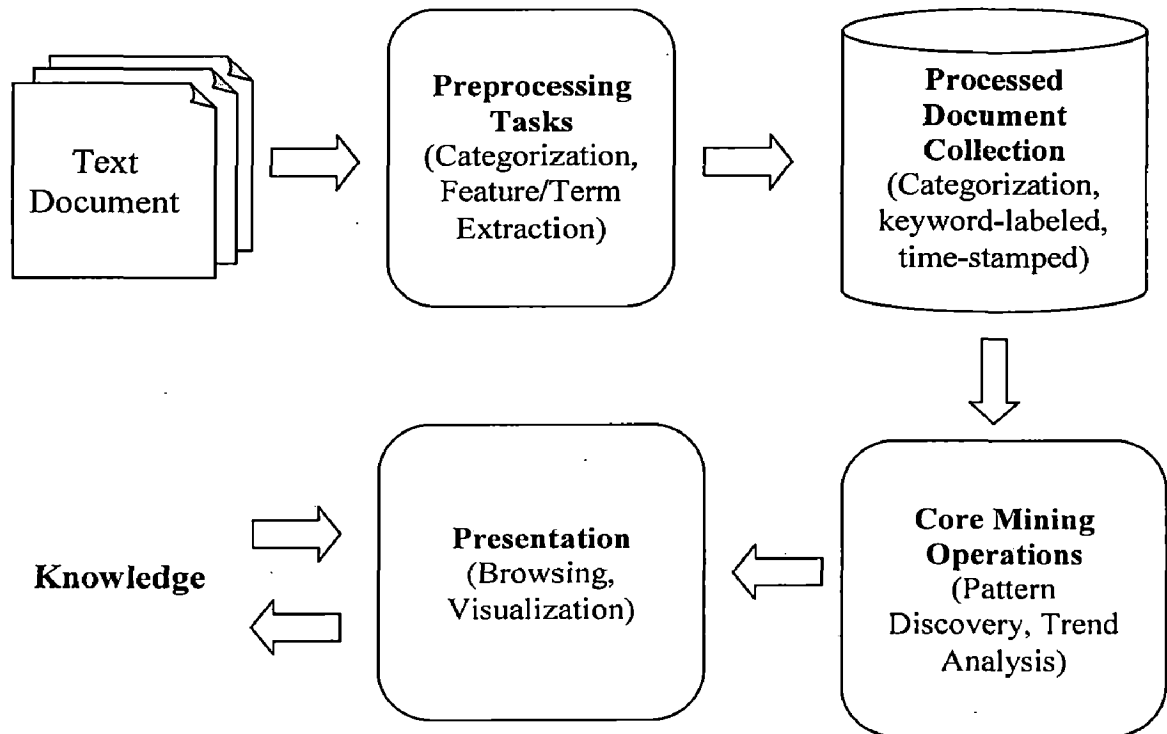


Figure 1.1: Process of Text Mining

1.2 Motivation

Text databases are rapidly growing due to the increasing amount of information available in electronic form, such as electronic publications, various kinds of electronic documents, e-mail, and the World Wide Web (which can also be viewed as a huge, interconnected, dynamic text database) [1]. In data mining it is assumed that the data to be mined is structured whereas the data stored in text databases is mostly semistructured means neither completely unstructured nor completely structured. It is important to organize such a large collection of documents into structured ontology.

The organization of documents facilitates navigation and search and simultaneously provides a framework for continual maintenance as document repository grows in size. Manual construction of structured ontology is one possible solution and has been adopted by Yahoo to organize the internet to structure library content. However

it has the obvious disadvantage of being too labor intensive and is viable only in large corporations. Thus it is desirable to seek automatic methods for organizing document collection [3].

One important area of text mining is the automatic classification of the text documents. The goal of text categorization methods is to associate one (or more) predefined categories to a particular document based on the likelihood suggested by a training set of labelled documents. Automatic text categorization technology can be applied to many application problems including finding answers to similar questions or queries, classifying news by subject or newsgroup, categorizing web pages, organizing e-mail messages, etc. Text document classification presents many challenges like it is difficult to capture high level semantics and abstract concepts of natural languages by simply looking at a few words, since words have semantic ambiguity such as polysemy and synonymy. In various text documents like emails, engineering or medical diagnostic documents, the sentences and phrases do not follow standard grammar rules as the text contain many typos and self invented acronyms. For the machine learning research community, text document classification faces two additional challenges: high dimensions in feature space, large number of text categories, enormous amount of training data, and new training data can emerge at later time. Some of these issues can be overcome by incremental learning approach. Incremental learning refers to the process of accumulating and managing knowledge over time [4].

An important research issue for text classification is the representation of feature space in compact form and the discovery of the complex relationships that exist between features, documents and classes. There are several approaches that try to quantify the notion of information for the basic components of a text classification problem. A characteristic valuation function like information gain or expected cross entropy or the weighted evidence for text, word frequency can also be used for dimensionality reduction. Clustering is one of the approaches used in this context. Thus clustering is an important area of research where it is used to aid text

classification for dimensionality reduction. In clustering, features are clustered into groups based on selected clustering criteria, where it creates new, reduced-size event spaces by joining similar features into groups. A similarity measure between features is defined like cosine similarity and similar features are combined into single cluster that no longer distinguish among their constituent features. Clustering is also used for sample reduction. A number of documents are grouped into clusters and each cluster is assigned a class label.

A typical example in text classification where incremental learning can be of great importance is in the field of sports world. In case of sports news documents, it is not possible that all types of news documents can be covered for training in one timestamp. The types of articles change with time in sports world and the features important for categorization also changes simultaneously. It is possible that a particular category was not in existence at one timestamp can be present in a future timestamp. Therefore there can be increment of categories with time. It can also be seen that some feature which were important previously are not presently important for classification and even some features did not exist before or were of no importance before are presently important. So there can be increase of features or change in the weights of features in a category with time by giving more weights to recent terms.

An incremental learning system updates its hypotheses, when necessary, in response to newly available training data without re-examining the old data. Such a learning strategy is both spatially and temporally economical, because it removes the need to store and reprocess old instances, it is most appropriate for learning tasks in which training data sets become available over a long period of time. The incremental learning learn new knowledge from a new batch of training examples without referring to the previously used training data nor forgetting the knowledge learnt from the previous training data [4].

1.3 Problem Statement

The aim of proposed research work is given as follows: “*To design an incremental algorithm for classifying text documents.*” The following aspects are considered in the designing of the algorithm:

1. The data increments have been considered over equal intervals of time.
2. The proposed work is suitable for applications wherein the frequency at which increments are considered is known. This means that the changes in the dataset are not very abrupt and unexpected in terms of time of occurrence.

1.4 Organization of the Thesis

The report is divided into six chapters including this introductory chapter. The rest of this thesis report is organized as follows:

Chapter 2 contains a brief description of text classification. A literature review on feature selection methods and dimension reduction methods is done.

A detailed description of proposed work is described in **Chapter 3**. Various modules for the work are discussed in detail.

In **Chapter 4**, the details of the code platform used and a brief description of dataset is given. The class hierarchy for training module and testing module built for the work is shown. The training procedure and testing procedure are also described.

Chapter 5 describes the results and discussion on the results. It also provides an analysis on the correctness of the proposed work.

Finally the thesis is concluded in **Chapter 6**. Some suggestions for future work are given.

CHAPTER 2

Literature Review

In this chapter, we briefly discuss about text classification. A literature review on feature extraction, dimension reduction and text classification is done.

2.1 Preprocessing

Preprocessing include all the processes required to prepare data for text mining system's core mining operations.

2.1.1 Document Representation

A classifier cannot directly process the text documents in their original form. Therefore, during a preprocessing step, the documents are converted into a structured representation. Typically, the documents are represented by feature vectors. A document is represented as a vector in this feature space, which is a sequence of features and their weights. The most common vector space model (*VSM*) [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] assumes that the objects are vectors in the high-dimensional feature space. A common example is the bag-of-words model, which simply uses all words in a document as the features, and thus the dimension of the feature space is equal to the number of different words in all of the documents.

The methods of giving weights to the features may vary. The simplest method is the binary, in which the feature weight is either one or zero, if the corresponding word is present in the document than it is weighed as one otherwise zero. Another more complex weighting scheme is to take into account the frequencies of the word in the document, in the category, and in the whole collection. It is known as Term Frequency Inverse Document Frequency (*TF-IDF*) scheme [4, 5, 7, 11], which gives the word w in the document d the *TF-IDFWeight*(w, d) as in Eq. (1).

$$TF - IDFWeight(w, d) = TermFreq(w, d) \cdot \log(N/DocFreq(w)) \quad (1)$$

where $TermFreq(w, d)$ is the frequency of the word in the document, N is the number of all documents, and $DocFreq(w)$ is the number of documents containing the word w [2].

Another popular method is the Normalized Term Frequency Inverse Document Frequency. Let the number of occurrences of word j in document i , is f_{ji} , and the number of documents which contain the word j , is d_j . Using these counts, the i^{th} document as a w -dimensional vector x_i can be represent as follows. For $1 \leq j \leq w$, set the j^{th} component of x_i , to be the product of three terms as given in Eq. (2).

$$x_{ji} = t_{ji} \cdot g_j \cdot s_i \quad (2)$$

where t_{ji} is the term weighting component and depends only on f_{ji} , while g_j is the global weighting component and depends on d_j , and s_i is the normalization component for x_i . Intuitively, t_{ji} captures the relative importance of a word in a document, while g_j captures the overall importance of a word in the entire set of documents. In such weighting schemes the objective is to enhance discrimination between various document vectors for better retrieval effectiveness. There can be various schemes for selecting the term, global, and normalization components, and one popular scheme is the normalized term frequency inverse document frequency.

In this scheme $t_{ji} = f_{ji}$, $g_j = \log(d/d_j)$ and s_i as in Eq. (3).

$$s_i = \left(\sum_{j=1}^w (t_{ji} g_j)^2 \right)^{-1/2} \quad (3)$$

The effect of normalization is only to retain the proportion of words occurring in a document which ensures that documents dealing with the same subject matter, but differing in length lead to similar document vectors [3, 12].

Another method used in [10] is described as follows. The web page is regarded as a set of lemma form of (T_1, T_2, \dots, T_n) in the *VSM*. Each lemma is endowed with certain authority value W_i . Each web page can be expressed by a web page characteristic vector $(T_1, W_1; T_2, W_2; \dots; T_m, W_n)$ to express. The formula used to calculate each

lemma authority value W_{ik} for all web pages in each web page training set D_k is given in Eq. (4).

$$W_{ik} = (1 + \ln tf_{ik}) \ln(N/N_k) \quad (4)$$

where tf_{ik} means lemma emergence frequency in D_k , N means classification system number, N_k means lemma web page frequency.

2.1.2 Feature Selection

The number of different words is large even in relatively small documents such as short news articles or paper abstracts. The dimension of the bag-of-words feature space for a big collection can reach hundreds of thousands; moreover, the document representation vectors, although sparse, may still have hundreds and thousands of nonzero components. Most of the words irrelevant to the categorization task can be dropped without harming the classifier performance and may even result in improvement owing to noise reduction. The preprocessing step that removes the irrelevant words is called feature selection [1, 3, 5]. Most text classification systems at least remove the non-content bearing stopwords.

Some systems perform stemming of words [5, 11, 14]. Stemming reduces the amount of dimensions and enhances the relevancy between word and document or categories. For example, "development", "developed" and "developing" will be all treated as "develop" after stemming.

Part-of-Speech Tagging (*POS*) tagging [2] is the annotation of words with the appropriate *POS* tags divide words into categories based on the role they play in the sentence in which they appear. *POS* tags provide information about the semantic content of a word. Nouns usually denote "tangible and intangible things," whereas prepositions express relationships between "things." Most *POS* tag sets make use of the same basic categories like Article, Noun, Verb, Adjective, Preposition, Number, and Proper Noun. Some systems contain a much more elaborate set of tags.

To filter the features, a measure of the relevance of each feature needs to be defined. The simplest measure is the document frequency. Experiments have suggested that by using only the top 10 percent of the most frequent words does not reduce the performance of classifiers. This contradicts the law of information retrieval, according to which the terms with low-to-medium document frequency are the most informative. There is no contradiction, however, because the large majority of all words have a very low document frequency, and the top 10 percent do contain all low-to-medium frequency words [1, 6].

Another measure of feature relevance that takes into account the relations between features and the categories is the information gain [1, 6, 9, 11] given in Eq. (5).

$$IG(w) = \sum_{c \in C \cup C'} \sum_{f \in \{w, w'\}} P(f, c) \cdot \log \frac{P(c|f)}{P(c)} \quad (5)$$

measures the number of bits of information obtained for the prediction of categories by knowing the presence or absence in a document of the feature f . The probabilities are computed as ratios of frequencies in the training data. IG is a measure based on entropy. Features that reduce the entropy the most are favoured for this method.

The IG measure can also be used in some other way for feature reduction task. To find a new word set $W' = \{w'_1, w'_2, \dots, w'_r\}$, $r < f$, where f is the number of features before reduction. W and W' should work equally well for all the desired properties with D . After feature reduction, each document d_i is converted to a new representation $d'_i = \langle w'_{i1}, w'_{i2}, \dots, w'_{ir} \rangle$ and the converted document set is $D' = \{d'_1, d'_2, \dots, d'_n\}$. If r is very much smaller than f , computation cost can be drastically reduced. This approach uses IG to select W' from W , and W' is a subset of W . This approach only uses the selected features as inputs for classification tasks. It measures the reduced uncertainty by an information-theoretic measure and gives each word a weight. The bigger the weight of a word is, the larger is the reduced uncertainty by the word. Let $\{c_1, c_2, \dots, c_p\}$ denote the set of classes. The weight of a word w_i is calculated as in Eq. (6).

$$\begin{aligned}
G(w_i) = & -\sum_{i=1}^p P_r(c_i) \log P_r(c_i) \\
& + P_r(w_i) \sum_{i=1}^p P_r(c_i | w_i) \log P_r(c_i | w_i) \\
& + P_r(w'_i) \sum_{i=1}^p P_r(c_i | w'_i) \log P_r(c_i | w'_i)
\end{aligned} \tag{6}$$

The words of top r weights in W are selected as the features in W' . Information gain is applied to compress the complexity of the document set from $O(nf)$ to $O(nr)$. If r is much smaller than f , the computation cost associated with document processing can be drastically reduced [8].

Another good measure is the chi-square[1, 6, 11] given in Eq. (7).

$$\chi^2_{max}(f) = \max_{c \in C} \frac{|T_r| \cdot (P(f, c) \cdot P(f', c') - P(f, c') \cdot P(f', c))^2}{P(f) \cdot P(f') \cdot P(c) \cdot P(c')} \tag{7}$$

which measures the maximal strength of dependence between the feature and the categories. Chi-square ranking favours features that are strongly dependent on relevant or irrelevant classes. One problem with this method is that it may give a high score to a rare feature. For example, a feature may only appear in 5 documents in a collection of 100,000 documents, but if all these 5 documents belong to the relevant class, the feature may still get a high score, which is counter-intuitive.

Likelihood ratio attempts to address the issue of assigning high scores to rare features in ranking. For a large sample size, it tends to behave similarly to chi-square ranking, but it also works well for a small sample size [6].

Mutual Information only measures the dependency between a feature and its relevant class, and as a result, it tends to favour rare terms if they are mostly used for relevant documents [6, 11].

Term Discrimination tries to measure the ability of a feature for distinguishing one document from the others in a collection. A very popular feature often has a negative discrimination value, since it tends to reduce the differences between documents, while a rare feature usually has a close-to-zero value, since it is not significant enough to affect the space density [6].

In ranking features, Count Difference (*CD*), tries to reflect that a feature whose document frequency for one class is higher than that for the other class is desirable since it helps distinguishing between the two classes and if a feature is rare in the training documents, its use will be limited since it only affects a small number of documents. Given a feature, we can partition the set of training documents into four regions in the following contingency Table 1.

	Relevant	Irrelevant
Feature Used	A	B
Feature Not Used	C	D

Table 1. Feature-Class Contingency Table

The relative document frequency, which is the ratio of the document frequency of a feature for one class over the average document frequency for the same class is defined in Eq. (8).

$$\begin{aligned} \text{relative } DF(t, u) &= a_t/a' \quad \text{and} \\ \text{relative } DF(t, \bar{u}) &= b_t/b' \end{aligned} \quad (8)$$

Here, a and b denote the average document frequencies for the relevant and irrelevant classes, which are computed in Eq. (9).

$$\begin{aligned} a' &= \frac{1}{M} \sum_{t=1}^M a_t \quad \text{and} \\ b' &= \frac{1}{M} \sum_{t=1}^M b_t \end{aligned} \quad (9)$$

where M is the number of original features before the selection process. Using the relative document frequencies, count difference score of a feature can be defined as the difference between its two relative document frequencies given by Eq. (10).

$$CD(t) = (a_t/a' - b_t/b')^2 \quad (10)$$

Intuitively, the relative document frequency measures the importance of a feature against the average feature for one class. If a feature is rare, its relative document frequency will be low, whereas if a feature is popular, its relative document frequency

will be high. The count difference tends to favour features whose relative document frequencies for one class are higher than those for the other class. If a feature is popular for both classes, its count difference score will be reduced [6].

2.1.3 Dimension Reduction

Dimensions of document spaces are always too high to deal with directly for many classification algorithms. Many collections of documents only contain a very small vocabulary of words that are really useful for classification. Dimensionality reduction techniques are a successful avenue for solving the problem. Dimensionality reduction techniques can be divided into two kinds: attribute reduction and sample reduction [9]. The feature selection methods discussed above are attribute reduction methods.

Clustering is used as a down-sampling pre-process to classification, in order to reduce the size of the training set resulting in a reduced dimensionality and a smaller, less complex classification problem, easier and quicker to solve. However, it should be noted that dimensionality reduction is not accomplished directly using clustering as a feature reduction technique, but rather in an indirect way through the removal of training examples that are most probably not useful to the classification task and the selection of the most representative redundant training set. In most of the cases this involves the collaboration of both clustering and classification techniques [15, 16]. Clustering can also be used as feature reduction technique by clustering similar features as discussed in [17].

Several different variants of clustering exist. A flat (or partitional) clustering produces a single partition of a set of objects into disjoint groups, whereas a hierarchical clustering results in a nested series of partitions. Each of these can either be a hard clustering or a soft one. In a hard clustering, every object may belong to exactly one cluster. In soft clustering, the membership is fuzzy – objects may belong to several clusters with a fractional degree of membership in each.

Irrespective of the problem variant, the clustering optimization problems are computationally very hard. The brute-force algorithm for a hard, flat clustering of n -element sets into k clusters would need to evaluate $k^n/k!$ possible partitioning. Even enumerating all possible single clusters of size l requires $n!/l!(n-l)!$, which is exponential in both n and l . Thus, there is no hope of solving the general optimization problem exactly, and usually some kind of a greedy approximation algorithm is used.

Agglomerative algorithms begin with each object in a separate cluster and successively merge clusters until a stopping criterion is satisfied. Divisive algorithms begin with a single cluster containing all objects and perform splitting until a stopping criterion is met. “Shuffling” algorithms iteratively redistribute objects in clusters.

The most commonly used algorithms are the k -means (hard, flat, shuffling) [3, 9], the EM -based mixture resolving (soft, flat, probabilistic), and the HAC (hierarchical, agglomerative) [2].

K -Means Clustering is used in [3, 8, 12, 14, 13, 18]. The method is described as: Let $\{P_j\}_{j=1}^k$ be a partition of D where k is a user-specified constant. The goal of the k -means clustering algorithm is to maximize the objective function in Eq. (11).

$$\sum_{j=1}^k \sum_{d_i \in P_j} Sim(d_i, P_j) \quad (11)$$

where $Sim(d_i, P_j)$ is the similarity measure between document d_i and P_j . A popular similarity measure is defined in Eq. (12).

$$Sim(d_i, P_j) = \cos(d_i, m_j) = \frac{d_i \cdot m_j}{\|d_i\| \|m_j\|} \quad (12)$$

where m_j is the centroid of P_j is defined in Eq. (13).

$$m_j = \frac{1}{|P_j|} \sum_{d \in P_j} d \quad (13)$$

Cosine similarity [3, 12] is easy to interpret and simple to compute for sparse vectors [9]. Another similarity measure that can be used is Euclidean Distance [12] which is defined in Eq. (14).

$$D(x_i, x_j) = \sqrt{\sum_k (x_{ik} - x_{jk})^2} \quad (14)$$

which is a particular case with $p = 2$ of Minkowski metric is given in Eq. (15).

$$D_p(x_i, x_j) = \left(\sum_k (x_{ik} - x_{jk})^{2p} \right)^{1/p} \quad (15)$$

There are many other possible similarity measures suitable for their particular purposes.

To evaluate the performance of k-means clustering algorithm, entropy based cluster validity measure can be used. Let k be the number of clusters obtained by clustering approach and L is the number of classes given by the data source. For each cluster i , calculate an entropy e_i of the cluster using Eq. (16).

$$e_i = - \sum_{j=1}^L p_{ij} \log_2 p_{ij} \quad (16)$$

where $p_{ij} = m_{ij}/m_i$ is the probability that a member of cluster i belongs to class j . Note that m_i is the number of objects in cluster i and m_{ij} is the number of objects of class j in cluster i . The entropy weighted sum E is defined as the sum of the entropies of each cluster weighted by the size of each cluster as shown in Eq. (17)

$$E = \sum_{i=1}^K \frac{m_i}{M} e_i \quad (17)$$

where M is the total number of data points. If E is smaller, the performance of a clustering method is better [8].

A Difference Similitude Matrix (*DSM*) based approach in [9] is used to reduce the dimensionality of item-by-document matrix which represents pre-specified collections of document, and generate rules for text classification.

Suppose IS is an information system, and $IS = \langle O, C, D, V, f \rangle$, where O denotes the system object set; C denotes the condition attribute set; D denotes the decision attribute set; $V = U (V_a : a \in (C \cup D))$ denotes the attribute value set; $f : O \times (C \cup D) \rightarrow V$

is the function that specifies the attribute values. As for the objects in information system, we can define a $m \times m$ difference-similitude matrix M_{DS} to represent their attributes and values. The matrix has two types of elements, similarity item m^s_{ij} and difference m^d_{ij} , which is defined in Eq. (18).

$$m_{ij} = \begin{cases} m^s_{ij} = \begin{cases} \{q \in C: f(q, x_i) = f(q, x_j)\}, D(x_i) = D(x_j) \\ \{\emptyset: \forall (f(q, x_i) \neq f(q, x_j))\}, D(x_i) = D(x_j) \end{cases} & i = 1, 2, \dots, m \\ m^d_{ij} = \begin{cases} \{q \in C: f(q, x_i) \neq f(q, x_j)\}, D(x_i) \neq D(x_j) \\ \{\emptyset: \forall (f(q, x_i) = f(q, x_j))\}, D(x_i) \neq D(x_j) \end{cases} & ; j = 1, 2, \dots, m \end{cases} \quad (18)$$

where m is the number of condition attributes and n is the number of instances in dataset. $Sigp(D)$ similarity significance and $Sigq(D)$ difference significance are essential to define similarity significance and difference significance respectively. Though these concepts are used during reduction. The principle that *DSM*-based reduction conforms is to get the following things without losing information of the original system after reduction:

- Minimum number of remained attributes to describe rules;
- Minimum number of classification rules.

2.2 Text Classification

Text classification (*TC* – also known as text categorization, or topic spotting) is the task of automatically sorting a set of documents into categories (or classes, or topics) from a predefined set [19]. The general text classification task can be formally defined as the task of approximating an unknown category assignment function $F: D \times C \rightarrow \{0, 1\}$, where D is the set of all possible documents and C is the set of predefined categories. The value of $F(d, c)$ is 1 if the document d belongs to the category c and 0 otherwise. The approximating function $M: D \times C \rightarrow \{0, 1\}$ is called a classifier, and the task is to build a classifier that produces results as “close” as possible to the true category assignment function F [2].

Rule-based methods in [9] and distance-based methods [7, 10] are the two most popular approaches for text classification. Rule-based methods use small subsets of keywords as condition attributes of decision rules, which means only part of the

keywords need to be examined by rules and the speed of classifying new document is faster than distance-based methods [9].

Text classification is a kind of typical model directive machine learning problem. It is generally divided into training and categorizing two stages. Its concrete algorithm described in [7, 10] is as follows:

Training stage:

- (1) $C = \{c_1, c_2, \dots, c_n\}$ // Define the category set, n is the total no. of categories
- (2) $S = \{s_1, s_2, \dots, s_m\}$ // Give training text set, m is the total no. of documents
 For $i = 1$ to m
 Training text s_i is marked as the sign c_j that is belonged to category
 Endfor
- (3) For $i = 1$ to m
 $X[s_i] \leftarrow$ characteristic vector of s_i
 $X[c_j] \leftarrow$ characteristic vector is representative of each category c_j of corresponding s_i
 Endfor

Categorizing stage:

- (4) $Tree \leftarrow$ information classification tree
 $N \leftarrow$ leaf node total number of the tree
 $X[1\dots n] \leftarrow$ characteristic vector of each leaf node
 $Threshold[1\dots n] \leftarrow$ threshold of information similitude degree for each leaf node
 $Info \leftarrow$ information to wait for classification
- (5) For $i = 1$ to n Do
 $Deg \leftarrow$ information similitude degree between $Info$ and $X[i]$
 If $Deg > Threshold[i]$ Then
 Add $Info$ and Deg to classification form of corresponding $X[i]$ sort
 For each node P from $X[i]$ node to Tree root path Do

```

                Add Info and Deg to classification form of corresponding P
                sort
            Endfor
        Endif
    Endfor

```

The text classification method is based on the concept of representing the training datasets in the form of category characteristic vectors.

As described in [9], text classification means selecting a small number of keywords to present document content and to describe classification rules, and the rules should be as few as possible. Keywords are taken as attributes to denote words or phrases those are important for classification. The Difference Similitude Matrix (*DSM*) based methods take both differences and similarities of objects into account, and can get good reduced attributes and rules without complex mathematical operations. In this method, a *DSM* based approach is applied to reduce the dimensionality of item-by-document matrix which represents pre-specified collections of document, and generate rules for text classification. The generated rule is written in Eq. (19).

$$r_{ik} : (h_1 = a_1) \wedge (h_2 = a_2) \wedge \dots \wedge (h_n = a_n) \Rightarrow d_j \rightarrow C_j \quad (19)$$

where r_{ik} is the k^{th} rule to decide whether d_j belongs to c_i , h_n is the word frequency of remained keyword t_n after *DSM* reduction and a_n is the corresponding value.

When a new document comes to be classified, count and discretize the item frequencies of attributes, then use the discretized results as condition attributes to try to match the rules defined while training the classifier. If any successes, then put the document into the category that rule describes.

K Nearest-Neighbour classifier described in [20] does not require model building. In this approach, all the training examples those are relatively similar to the attributes of the test example are found. These examples are known as nearest-neighbours, are used to determine the class label of the test example. A nearest-neighbour classifier represents each example as a data point in a d -dimensional space, where d is the

number of attributes. In this approach k is the number of nearest-neighbours used to determine the class label of the test example. If k is too small, then the nearest-neighbour classifier may be susceptible to overfitting because of noise in the training data. On the other hand, if k is too large, the nearest-neighbour classifier may misclassify the test instance because its list of neighbours may include data points that are located far away from its neighbourhood.

This classification algorithm computes the distance (or similarity) between each test example $z = (x', y')$ and all the training examples $(x, y) \in D$ to determine its nearest-neighbour list, D_z . The algorithm is described as follows:

1. Let k be the number of nearest-neighbours and D be the set of training examples.
2. for each test example $z = (x', y')$ do
 - i. Compute $d(x', x)$, the distance between z and every example, $(x, y) \in D$.
 - ii. Select $D_z \subseteq D$, the set of k closest training examples to z .
 - iii. $y' = \underset{(x_i, y_i) \in D_z}{\operatorname{argmax}} I(v = y_i)$
3. end for

Such computation can be costly if the number of training examples is large. The k nearest-neighbour classifier is used in [11] for classification.

A confusion matrix in table2 provides the information needed to determine, how well a classification model performs.

	Prediction Class	
	Class = 1	Class = 2
Feature Used	f_{11}	f_{10}
Feature Not Used	f_{01}	f_{00}

Table 2. Confusion matrix for a 2-class problem

The performance metric accuracy defined in [20] is given in Eq. (20).

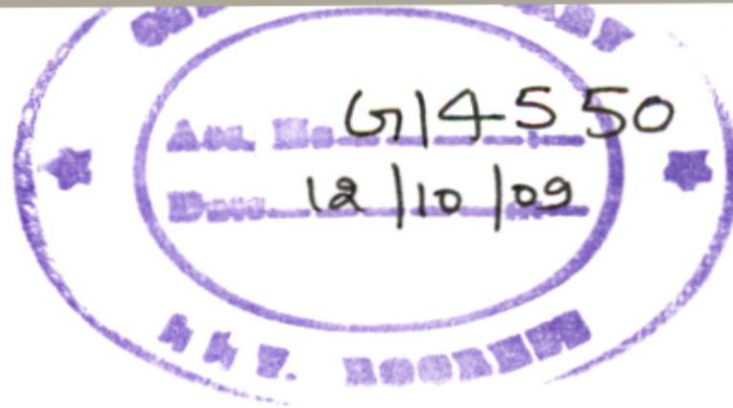
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total Number of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \quad (20)$$

2.3 Research Gap

In many applications it is not possible to build the dictionary beforehand because of lack of domain knowledge. Most of the works uses a predefined dictionary which is also a time consuming and tedious task.

The traditional text classification methods are incapable for applications, where the data keeps getting generated over time. Therefore, incremental classification techniques are required under such circumstances.

Most of the work treats all the terms in the dictionary as equal. As the terms evolve over time, some old terms become irrelevant and therefore terms should be given unequal weights in the dictionary.



CHAPTER 3

Proposed Work

3.1 Overall Architecture for Proposed Algorithm

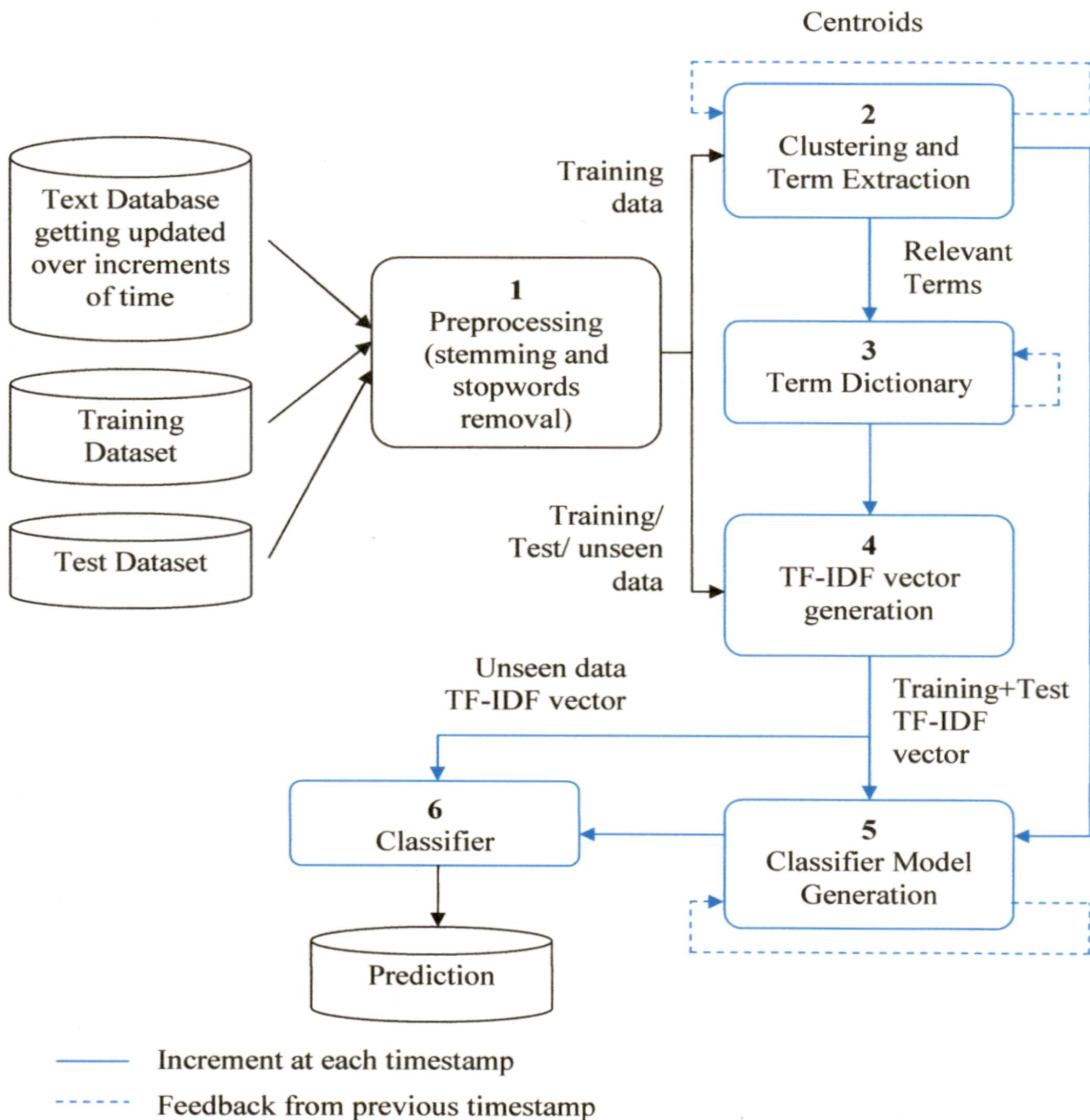


Figure 3.1: The overall architecture for proposed algorithm

The overall architecture for proposed algorithm is shown in Figure 3.1. It consists of six phases. The input data keeps getting updated with time and it is fed into the preprocessing phase represented as block 1. Then the training data is fed for clustering and term extraction in block 2. This block is part of the incremental approach and it is processed at each timestamp. The centroids obtained at a timestamp are sent to the next timestamp clustering process for initializing the centroids. The relevant terms are sent to the term dictionary in block 3. In block 4 the preprocessed data and term dictionary are used to generate *TF-IDF* vectors for training, test, and unseen data. The training and test data are used to generate the classifier in block 5. Block 6 shows the classifier which is used for prediction. The unseen data *TF-IDF* vectors are sent from block 4 to block 6 for prediction. The details of each block are discussed in following sections:

3.2 Preprocessing

Block 1 is for preprocessing documents. The flowchart for the preprocessing every document in the training dataset, test dataset and unseen data is drawn in Figure 3.2. The algorithm is described just after the flowchart.

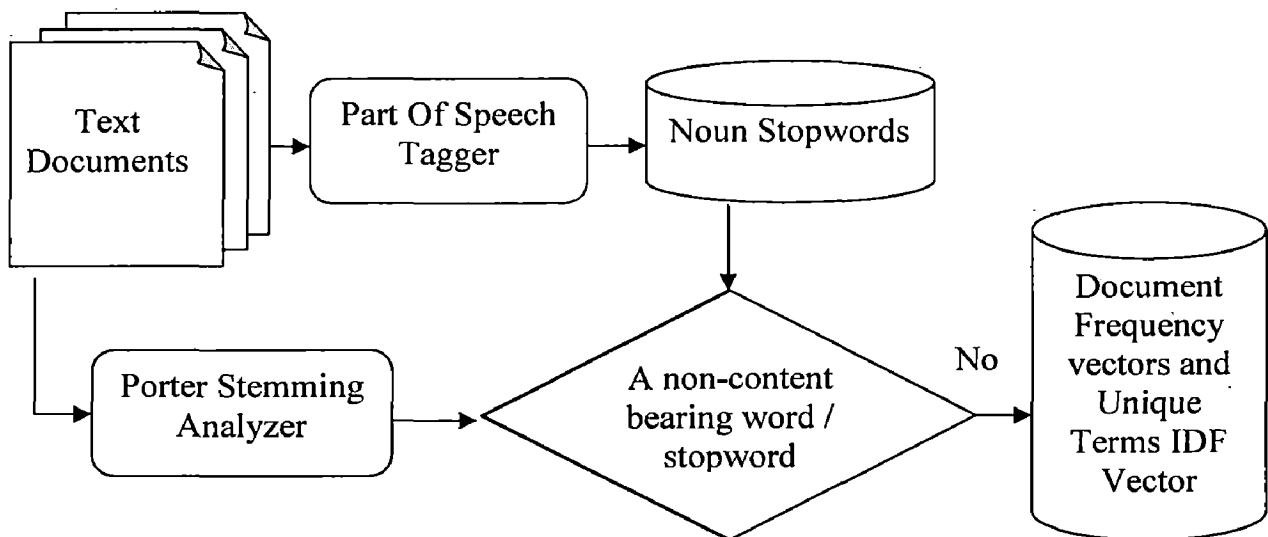


Figure 3.2: Flowchart for Preprocessing Text Documents

Preprocessing Algorithm:

1. Use POS tagger to get the information about the semantic content of a word and create a noun stopwords file.
2. Ignore cases, stem words and extract all unique words from all documents.
3. Eliminate non-content bearing stopwords and noun stopwords.
4. For each document, count the word frequency.
5. Create a global dictionary of all unique words and assign them a unique Id.
6. Store every document's unique words Ids with their frequencies in different files.
7. Store global dictionary word Ids along with words and word *IDF* in different files for each training subset.

A document as input for the preprocessing module is shown in Figure 3.3.

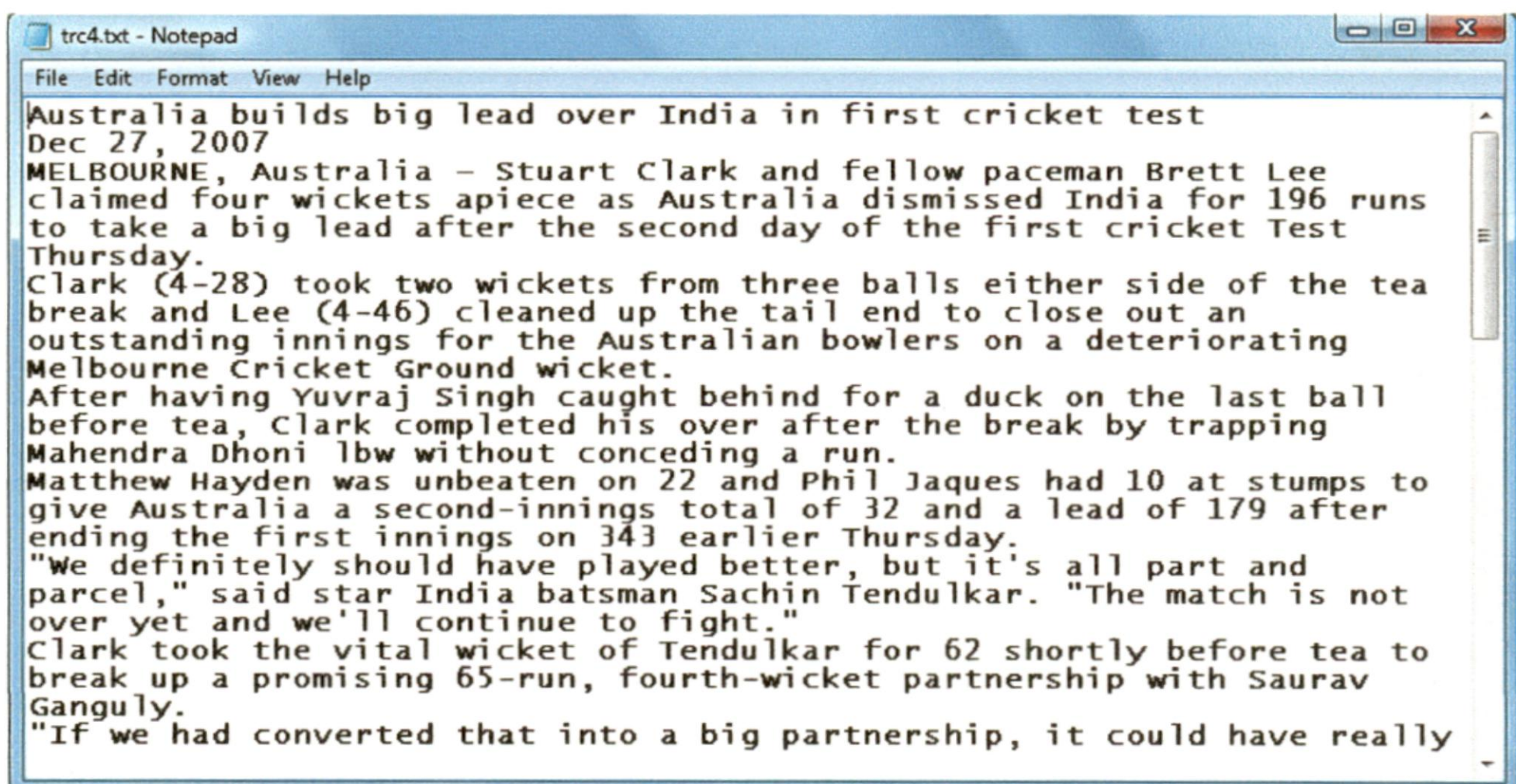


Figure 3.3: Input document to preprocessing module

The Term Frequency Vector of the input document is stored in a document shown in Figure 3.4 and a Global Dictionary containing all the unique terms in a training document subset are shown in Figure 3.5. These documents are the output of preprocessing module. The Term Frequency Vector document consists of the term

IDs and the their frequencies. The Global Dictionary document consists of the term IDs, terms and their *IDF* value.

```

File Edit Format View Help
6 1 15 9 19 1 25 2 27 1 42 2 44 1 47 1 50 2 56 1 68 3 79 3 97 2 101 10
105 8 127 1 128 1 132 1 133 1 143 1 144 3 147 1 153 1 161 1 168 1 175
1 176 1 181 1 183 1 185 1 187 2 203 5 207 1 213 1 214 2 225 3 234 1
254 3 261 1 269 1 282 1 283 1 334 1 344 1 358 1 364 1 395 1 407 1 433
1 445 1 467 2 477 3 478 1 498 3 586 1 593 1 614 1 616 1 621 1 643 4
681 1 726 1 745 1 783 1 818 2 848 1 887 1 926 2 1006 1 1016 1 1035 1
1052 1 1069 1 1103 1 1146 2 1150 1 1165 1 1176 1 1243 1 1333 1 1601 1
1603 1 1615 2 1663 1 1743 1 1751 2 1753 2 1766 1 1807 3 1922 1 1951 1
1962 2 1973 1 1974 1 1975 12 1976 1 1977 3 1978 1 1979 1 1980 1 1981 1
1982 1 1983 2 1984 1 1985 1 1986 1 1987 1 1988 1 1989 4 1990 1 1991 1
1992 1 1993 1 1994 1 1995 1 1996 1 1997 1 1998 2 1999 1 2000 1 2001 1
2002 1 2003 1 2004 1 2005 1 2006 1 2007 1 2008 1 2009 1

```

Figure 3.4: Output Term frequency document of preprocessing module

```

File Edit Format View Help
0 packer 2.531426665422893 1 stai 1.9123874570166697 4 green
2.531426665422893 5 mind 2.531426665422893 6 short 1.8382794848629478
7 postseason 2.6855773452501515 8 experi 2.3978952727983707 9 coach
0.9509762898620452 10 youth 3.378724525810097 11 team
0.43428554664365626 12 lost 1.2584609896100056 13 composur
3.378724525810097 14 down 1.1451323043030026 15 over
0.4168938039317871 16 nfc 2.8678989020441064 17 playoff
1.9123874570166697 18 game 0.33420208808667373 19 against
0.488352767913932 20 resist 2.8678989020441064 23 turnov
2.8678989020441064 24 histori 1.4328143767547836 25 side
1.586965056582042 26 victori 1.1814999484738773 27 realli
1.1814999484738773 28 worri 3.378724525810097 29 control
1.7047480922384253 30 gui 1.8382794848629478 31 weren
3.378724525810097 32 grow 2.6855773452501515 34 dure
1.1814999484738773 35 cours 2.0794415416798357 36 week
0.7637647477738987 37 address 2.8678989020441064 38 footbal
0.9808292530117262 39 reason 1.992430164690206 40 good
0.9509762898620452 41 improv 2.3978952727983707 42 think
0.8938178760220965 43 best 0.81377516834856 44 foot 2.3978952727983707
45 forward 1.8382794848629478 46 certainli 2.6855773452501515 47
better 1.6441234704219905 48 field 1.4816045409242156 49 turn
1.3862943611198906 50 out 0.35020242943311497 51 brutal
3.378724525810097 53 fell 2.3978952727983707 54 wild

```

Figure 3.5: Output Dictionary of preprocessing module

3.3 Clustering and Term Extraction

For clustering each training time-stamped subset, use k-means clustering. The terms are extracted from the centroids, as all the relevant term for categories are in the centroids. The flowchart for the clustering module is drawn in Figure 3.6. The algorithm is described just after the flowchart.

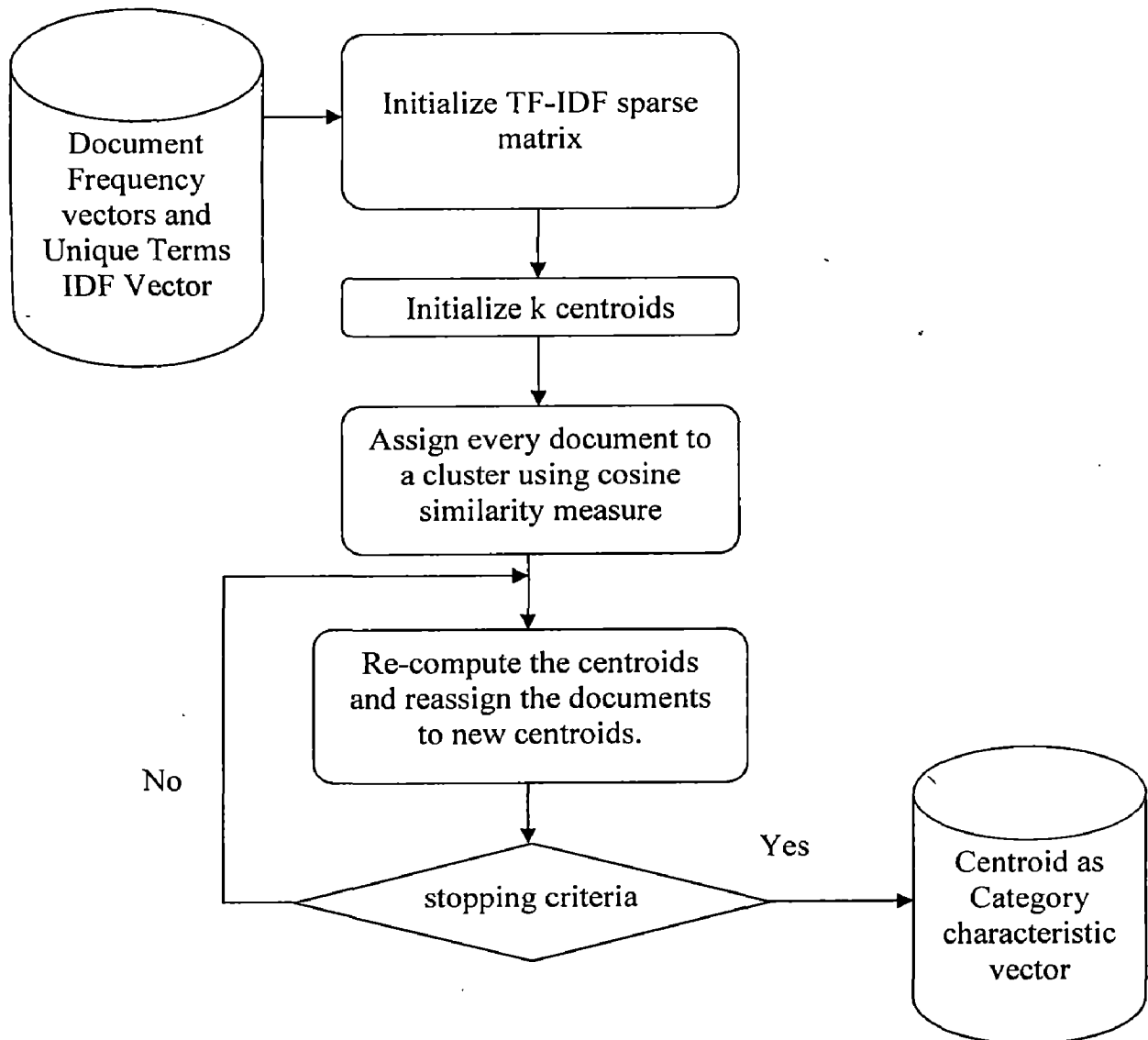


Figure 3.6: Flowchart for Clustering Module

K-means Clustering Algorithm:

1. Initialize the TF-IDF document-word matrix as sparse matrix.
2. Initialize k centroids where k is the no. of centroids is a user specified constant.

3. Assign n documents in D to clusters using cosine similarity measure between document d_i and centroid c_j , where $1 \leq i \leq n$ and $1 \leq j \leq k$.
4. Re-compute the centroids and reassign the documents to the centroids.
5. Repeat step 4 until there is no change in the 2 consecutive re-computed centroids.

The input of the clustering documents is shown in Figure 3.4 and Figure 3.5. The output of the clustering module is the centroids of clusters and they are stored in a document as shown in Figure 3.7. The Centroid of Clusters document consists of the features in the centroid and their *TF-IDF* values.

```

packer 1.8184824186332698 stai 2.7136021011998737 green
2.3539293971054818 mind 2.0 short 2.6457513110645907 postseason
2.309401076758503 coach 3.928571428571429 youth 1.4142135623730951
team 4.629019992067058 lost 3.1235807588017885 down 3.086974532565159
nfc 1.975658322294524 playoff 2.8323527714997336 game
4.354938733534976 turnover 1.3130643285972257 histori 3.127716210856122
realli 3.8013155617496435 worri 1.4142135623730951 gui
2.6261286571944513 cours 2.3333333333333334 week 3.54474503897027
footbal 4.873672965232997 reason 2.3333333333333333 good
3.8497419160916238 think 3.3113308926626095 best 3.6829475375170038
forward 2.5298221281347035 certainli 1.7320508075688772 better
2.7136021011998723 field 3.0532901344551733 turn 3.1529631254723287
out 4.923659639173307 brutal 1.4142135623730951 fell
2.3333333333333326 wild 2.23606797749979 card 1.4142135623730951 left
2.8284271247461907 right 3.0508510792387606 tackl 2.309401076758503
situat 2.449489742783178 aren 1.4142135623730951 bounc
1.7320508075688772 kind 2.840187787218772 tight 1.5075567228888183
nobodi 1.7320508075688774 young 2.334868926348074 respond
1.4142135623730951 score 2.941742027072761 drive 1.6666666666666667
certain 2.0 threw 2.138089935299395 touchdown 2.5999999999999996 pass
2.612789058968723 ran 1.7320508075688774 gain 1.6666666666666667 yard
2.398852020855824 offens 2.9999999999999996 fire 1.8898223650461363
season 4.451145741570189 group 2.1213203435596424 train 2.0 often

```

Figure 3.7: Output centroids of clustering module

3.4 Term Dictionary

In block 3 the dictionary of terms relevant to categories is build. With each increment the terms in the dictionary are updated and weights are allotted to the terms based on the timestamps. The newly added terms are assigned unit weights whereas the weight for the terms in the dictionary is reduced. This process continues as more and more data keeps getting generated. As time passes, very old irrelevant terms get removed from the dictionary.

3.5 TF-IDF Vector Generation

In block 4 the *TF-IDF* vectors for all training, test and unseen documents is generated. The terms generated by block 3 are stored in the terms dictionary and only those terms are considered for generating vectors. The *IDF* for the terms is obtained from the unique terms *IDF* vector obtained after preprocessing documents. The *TF-IDF* vectors are stored in files for the block 5 and block 6. The *TF-IDF* vector generation is shown in Figure 3.8.

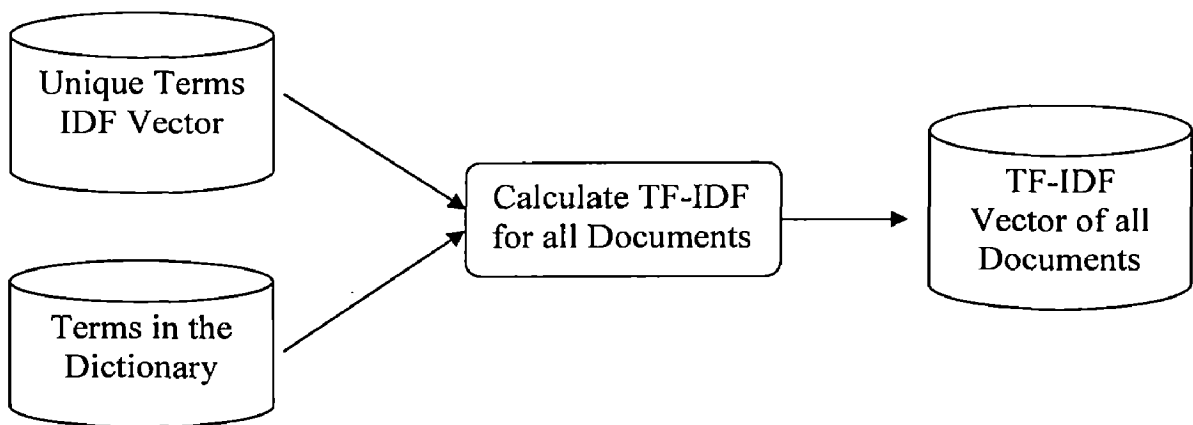


Figure 3.8: TF-IDF Vector Generation

3.6 Classifier Model Generation

Block 5 is for classifier model generation. The *TF-IDF* vectors of test documents and the category characteristic vectors are fed as input to calculate the cosine similarity between each of them. The maximum cosine similarity of a document with all the category characteristics vector is calculated and if it is above threshold then the category with maximum cosine similarity is assigned to the document. The results obtained from the classifier are validated with the original results and the classifier model is validated. The Process of classifier model generation is shown in Figure 3.9.

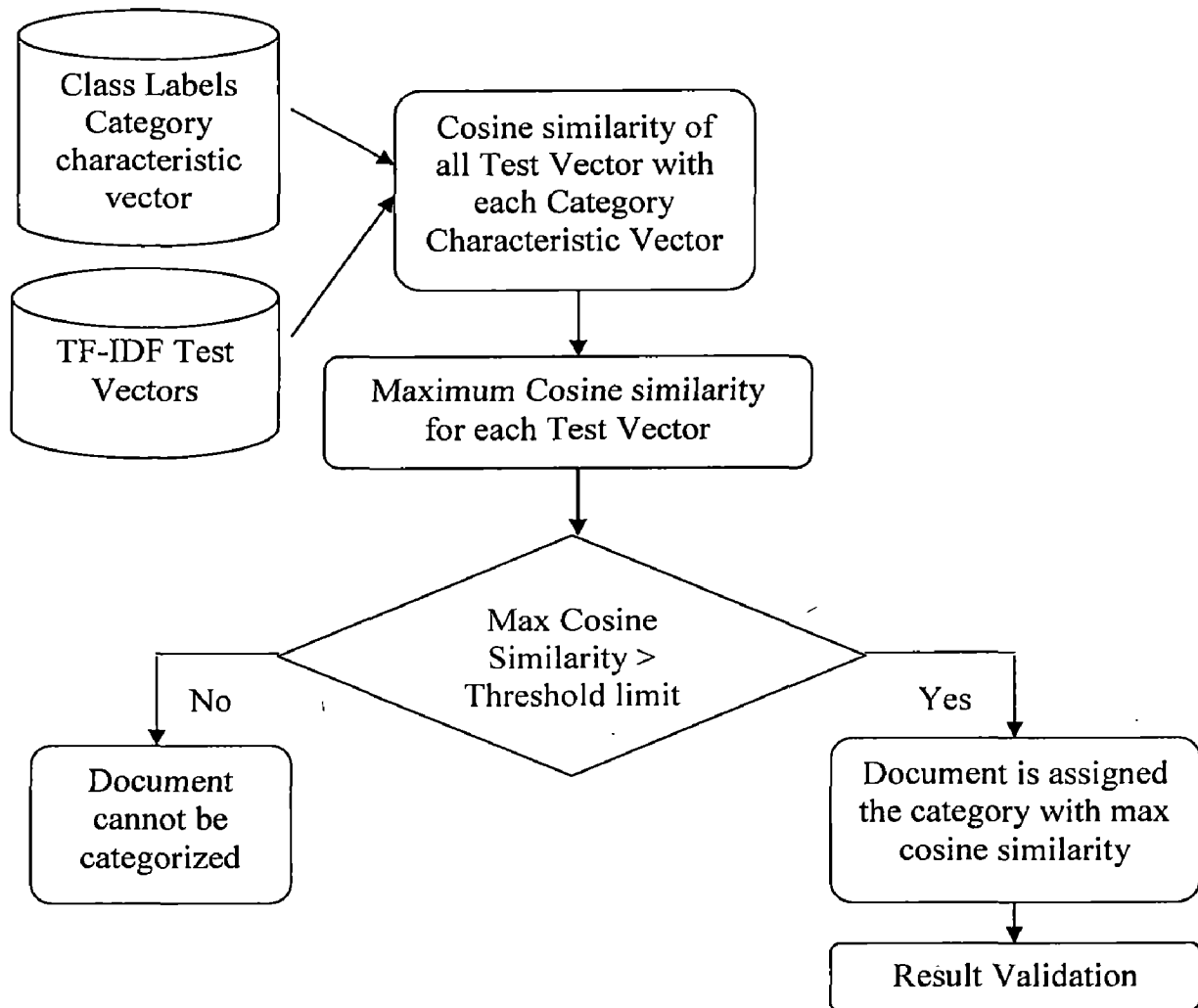


Figure 3.9: Classifier Model Generation and Result Validation

3.7 Classifier

The classifier generated in block 5 is used for prediction of class labels of unseen documents. The *TF-IDF* vectors for unseen data are obtained from block 4. The classifier is used for prediction is shown in Figure 3.10.

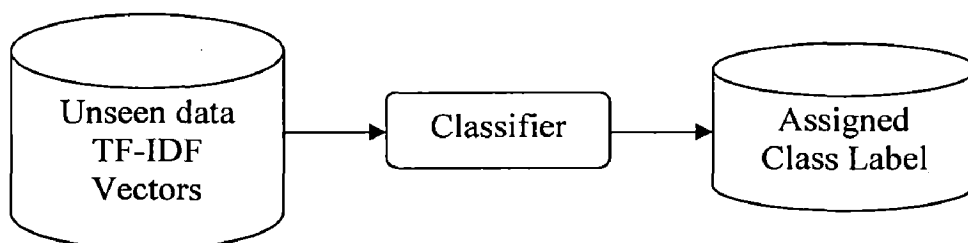


Figure 3.10: Classifier

CHAPTER 4

IMPLEMENTATION DETAILS

The implementation details of the proposed algorithm are listed as follows:

4.1 Code Platform

The programming language Java has been chosen and it is platform independent. The environment used is Eclipse 3.3 IDE which is a open source IDE for Java/J2ee Development applications. I have developed the code using JDK 1.5. The Part Of Speech Tagger JTextPro is available as a Java utility in [22] and it can be easily added in the library for use as java is used for coding. The code for Porter Stemming Analyzer is also available in java in [23].

4.2 Dataset Description

The text documents data has been taken from [21]. It comprises of a collection of sports documents from various sports categories, like cricket, football, tennis, boxing, swimming and chess. The data has been collected on a monthly basis. The number of text documents collected in each month is variable. For our purpose, we have assumed each increment to be made up of text documents pertaining to 3 months period. 3 months is chosen because it represents business quarter. However any suitable interval period can be chosen.

The training documents are chosen randomly from all the documents in a time-stamp and rest of the documents are taken for testing.

4.3 Class Hierarchy

The class hierarchy used to implement the proposed work is shown as follows:

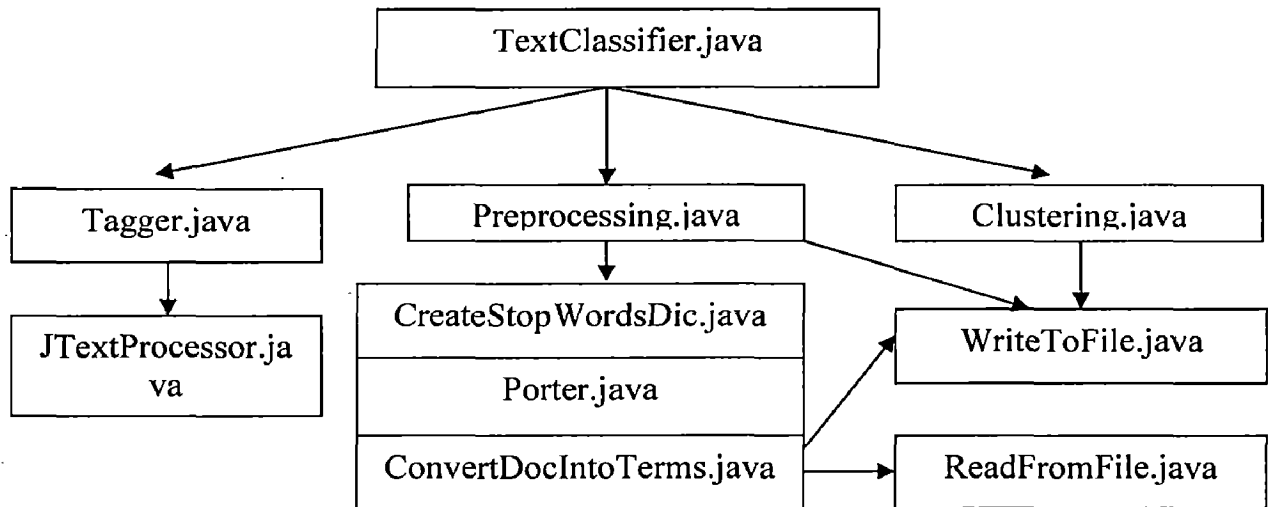


Figure 4.1: Class hierarchy of training module

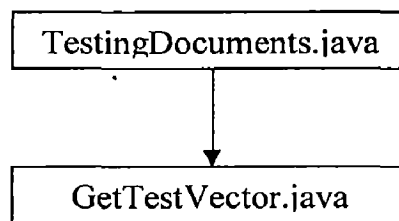


Figure 4.2: Class hierarchy of testing module

4.4 Training Procedure

TextClassifier.java

This class is used to train the model. This class initializes the global variable used for building the classifier like the file name of text files containing training documents path and the file name of the text files containing documents path for storing the word frequency vector. File name storing dictionary at each increment and storing centroids obtained at each increment is also initialized.

```

String TrainDocFile1 = "TrainingDocs1.txt";
String TermFreqFile1 = "TermFrequencyDocs1.txt";
String GlobalFile1 = "GlobalTerms1.txt";
  
```

```
String CentroidOfClusters1 = "CentroidOfClusters1.txt";
```

The main() function of class Tagger.java is called to tag each word in the documents. The function ConvertIntoUniqueTerms() of class PreprocessData.java is called here for preprocessing training documents. The main function of class Clustering.java is called for clustering the documents.

```
Tagger tg = new Tagger();  
tg.main();
```

```
PreprocessData ppd = new PreprocessData();  
ppd.ConvertIntoUniqueTerms(docList1,TermFreqDocList1,GlobalFile1,docList2,TermFreqDocList2,GlobalFile2,docList3,TermFreqDocList3,GlobalFile3,docList4,TermFreqDocList4,GlobalFile4,docList5,TermFreqDocList5,GlobalFile5);
```

```
Clustering cl = new Clustering();  
cl.main(TermFreqDocList1,GlobalFile1,centroid11,centroid12,CentroidOfClusters1);
```

PreprocessData.java

This class is used to convert documents into unique terms using the ConvertIntoUniqueTerms() function.

```
public static void ConvertIntoUniqueTerms(String[] DocList1,String[] StemDocList1,String GlobalFile1,String[] DocList2,String[] StemDocList2,String GlobalFile2,String[] DocList3,String[] StemDocList3,String GlobalFile3,String[] DocList4,String[] StemDocList4,String GlobalFile4,String[] DocList5,String[] StemDocList5,String GlobalFile5)
```

First it create the noun stopwords CreateStopWordsDic.java class using and then stem the words using Porter.java class and store them in a text file using WriteToFile.java class.

```
CreateStopWordsDic cswd1 = new CreateStopWordsDic();
```

```
cswd1.main(TagdocList5,StopWordsFile5);
```

```
Porter p = new Porter();
```

```
String StemmedStopWords1 = p.StemmedDoc(StopWordsFile1);
```

```
WriteToFile wtf = new WriteToFile();
```

```
wtf.WriteToFile(StopWordsFileNew3, StemmedStopWords3);
```

Stem each training document using Porter.java class and convert each document into word frequency vector using ConvertDocIntoTerms.java class

```
StemmedDoc1[i] = p.StemmedDoc(DocList1[i]);
```

```
ConvertDocIntoTerms cdt1 = new ConvertDocIntoTerms();
```

```
cdt1.main(StemedDoc1, StemDocList1, GlobalFile1, StopWordsFileNew1);
```

CreateStopWordsDic.java

The stopwords dictionary is initialized using CreateStopWordsDic() constructor. For each training document the function getTerms() is called to get the nouns in the documents. Print and write the unique stopwords in all documents in a common stopwords dictionary using PrintAndWriteDictionary() function.

```
public CreateStopWordsDic()
```

```
{
```

```
    zero = "0";
```

```
    for(int i = 0; i<DicSize; i++)
```

```
        dictionary1[i] = "0";
```

```
}
```

```
public void getTerms(String FileName){}
```

```
public void PrintAndWriteDictionary(String StopWordsFile) {}
```

Porter.java

This class transforms a word into its root form. The function stem() calls many function step1(), step2(), step3(), step4(), step5() and step6() to stem the words.

In step1() the word gets rid of plurals, -ed, -ing. For example

bowled -> bowl

players -> player

agreed -> agree

step2() turns terminal y into i when there is another vowel in the stem.

step3() maps double suffices to single ones. so -ization (= -ize plus -ation) maps to -ize etc.

step4() deals with -ic-, -full, -ness etc. similar strategy to step3.

step5() takes off -ant, -ence etc.

step6() removes a final -e if m() > 1.

m() measures the number of consonant sequences between 0 and j.

```
private final void step1(){}  
private final void step1(){}  
private final void step3() {}  
private final void step4() {}  
private final void step5() {}  
private final void step6() {}
```

WriteToFile.java

This class has a function WriteToFl() which writes the data in the text file FileName.

```
public void WriteToFl(String FileName, String data) {}
```

ConvertDocIntoTerms.java

This class takes as input the stemmed words of all documents in a training subset and converts them into word frequency vectors and also create a Globalvector containing all the unique words in this dataset and write this vector in a file named GlobalTerms.

In main(), first the noun stopwords file is read and stored in an array.

```
String[] StopWords2 = Readf[0].split(" ", -1);
```

A string array GlobalVector is created to store all the unique words and initialized to zero. For every stemmed document's data getTermNFreq() function is called. In this function noun stopwords and non-content bearing words are eliminated. And all the unique words are identified and their frequency (number of documents the word occurs) is calculated.

```
String TermNFreq = c.getTermNFreq(StemmedDoc[l], StopWords2);
```

All the words are read and if not in the global vector then they are added to it and if already in the global vector then their occurrence (number of times the word is present in all) is incremented.

Sort the words on the basis of their unique Ids in global vector. For every document, store the word's Ids along with their frequencies in a file. Store the Global dictionary word Ids along with the words and their IDF's (Inverse Document Frequency).

ReadFromFile.java

This class has a function ReadFromFl() which reads from the text file FileName and returns a string array.

```
public String[] ReadFromFl(String Filename, int StringLength)throws IOException{}
```


Tagger.java

In this class the JTextProcessor.java main() function is called. This class is a part of JTextPro which is a utility that helps in tagging.

```
String[] args1 = {"models", "DEC07_FEB08_FOOTBALL/trfb1.txt"};  
JTextProcessor.main(args1);
```

Clustering.java

This class is used to cluster a dataset into k clusters, where k is constant for clustering. A hash map is used as sparse matrix is created to store the centroids and TFIDF vectors of training documents. Another hash map is created to store the term Ids along with their IDFs. Another hash map is created to store the term Ids along with the terms.

```
HashMap<String, Double> map = new HashMap<String, Double>();  
HashMap<Integer, Double> global_map = new HashMap<Integer, Double>();  
HashMap<Integer, String> global_Terms_map = new HashMap<Integer, String>();
```

A function CreateClusters() is called to create the cluster of the dataset and the cluster result is printed in the form of, which document fall in which cluster and the centroids are stored as category characteristics vectors.

```
public void CreateClusters(int[] TermIDs, String CentroidOfClusters){}
```

4.5 Testing Procedure

TestingDocument.java

This class is used to test the model. This class initializes the global variable used for building the classifier like the file name storing dictionary of each increment and storing centroids obtained at each increment.

```
String GlobalFile1 = "GlobalTerms1.txt";  
String CentroidOfClusters1 = "CentroidOfClusters1.txt";
```

The function `GetIDF()` is called to get the Terms and their IDF's stored in the `GlobalTerms` File. Then the function `GetDictionary()` is called for creating the Dynamic dictionary and get category characteristic vectors with their TFIDF. The dictionary is created with words having different weights giving more weight to more recent words.

```
String[][] TermNIDFs1 = td.GetIDF(GlobalFile1);  
  
td.GetDictionary(CentroidOfClusters1,1,TermNIDFs1);
```

The file storing the names of the test documents is initialised.

```
String TestDocFileNames = "TestDocument.txt";
```

For each test document, the function `getTermFrequency()` of class `GetTestVector.java` is called to get the word frequency vector of the document.

```
GetTestVector gtv = new GetTestVector();  
double[] WtTfIdf = td.GetWtTfIdfVector(TermFreq);
```

Then the weighted TF-IDF Vector for each test document is created using the `GetTfIdfVector()` function.

```
double[] WtTfIdf = td.GetWtTfIdfVector(TermFreq);
```

This `WtTfIdf` Vector is matched with all the category characteristic vectors using `CosineSimilarity()` function.

```
double CosSim1 = td.CosineSimilarity(WtTfIdf,1);
```

The maximum cosine similarity is found and the CosSim having maximum value is checked with the threshold limit, if it is above threshold then it is classified under that category else the document cannot be classified.

GetTestVector.java

This class reads the test document and returns the word frequencies of the words in the dictionary.

```
public int[] getTermFrequency(String FileName, String[] Dictionary){}
```

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Results and Discussion

The model is trained with 522 training documents. Initially for the first quarter, the number of documents in training dataset was 87. In the next quarter 129 documents were taken, and in the third quarter the number of documents in the training dataset was 111. In fourth and fifth quarters the datasets were 119 and 76 documents respectively. And testing of classifier is done with 372 documents. The test dataset is build from 5 timestamps and 6 categories that evolve over time while training model is build. The number of test documents for first quarter was 53. In the next quarters, the number of test documents were 66, 99, 86, 68. The training and testing documents are selected randomly from all the 3 months in a quarter.

To evaluate the performance of k-means clustering algorithm used in our work, entropy based cluster validity measure is adopted.

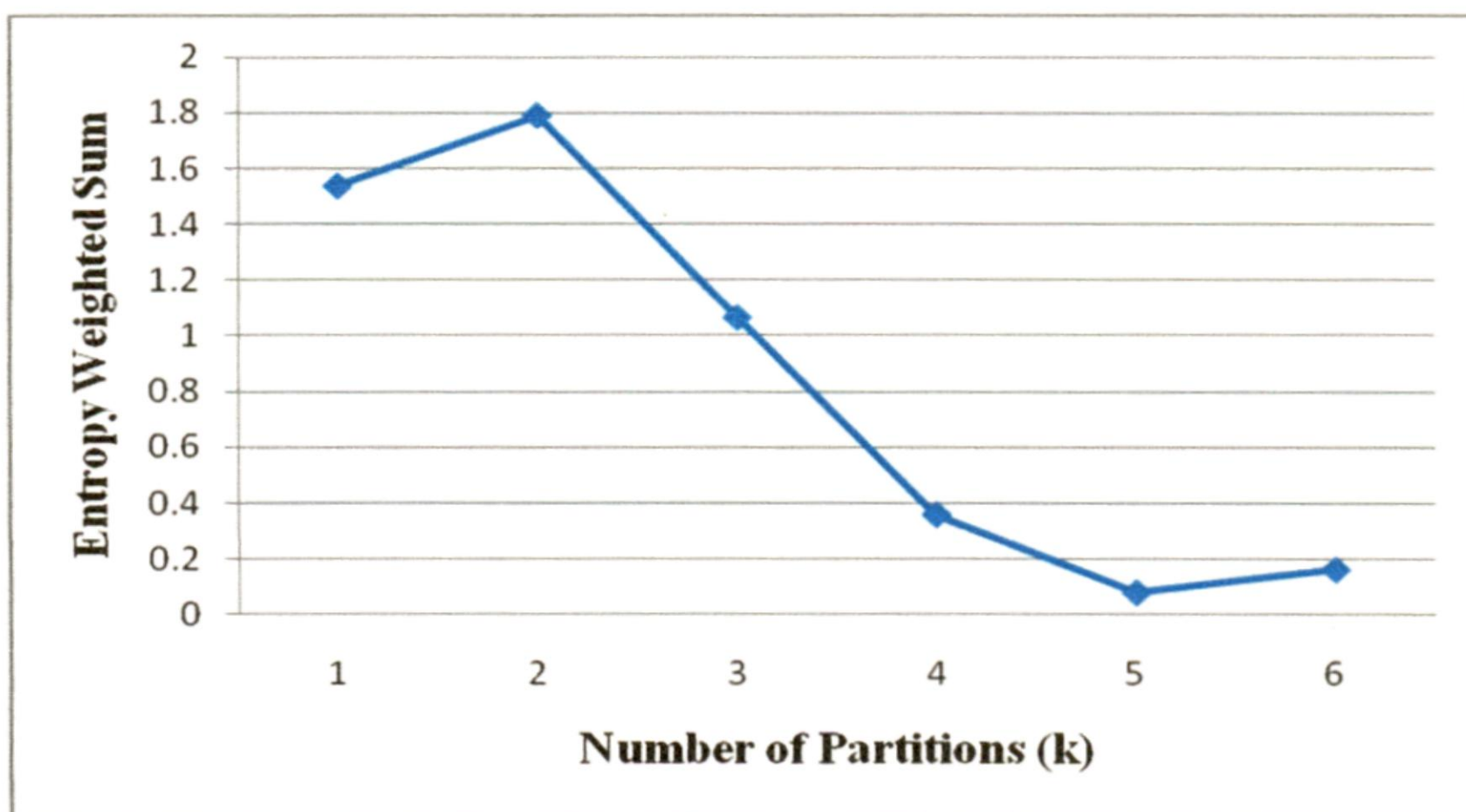


Figure 5.1: Entropy weighted sum on timestamp4 dataset

The entropy is calculated using Eq. (16). The entropy weighted sum E is defined in Eq. (17). The entropy weighted sum of the k-means clustering algorithm is shown in Figure 5.1.

The classifier is validated by testing the test dataset at all the timestamps. The accuracy is calculated using Eq. (20).

A static dictionary does not change with time as there is no incremental approach. The dictionary was created using dataset of first timestamp. The test documents taken for first timestamps consists of documents from first timestamp. And then in second timestamp, the test documents consist of documents from second timestamp. And same is done for third, fourth and fifth timestamps. The accuracy of classifier is shown in Figure 5.2. It can be seen that the accuracy is highest for the first timestamp, as the classifier was build using first quarter training dataset.

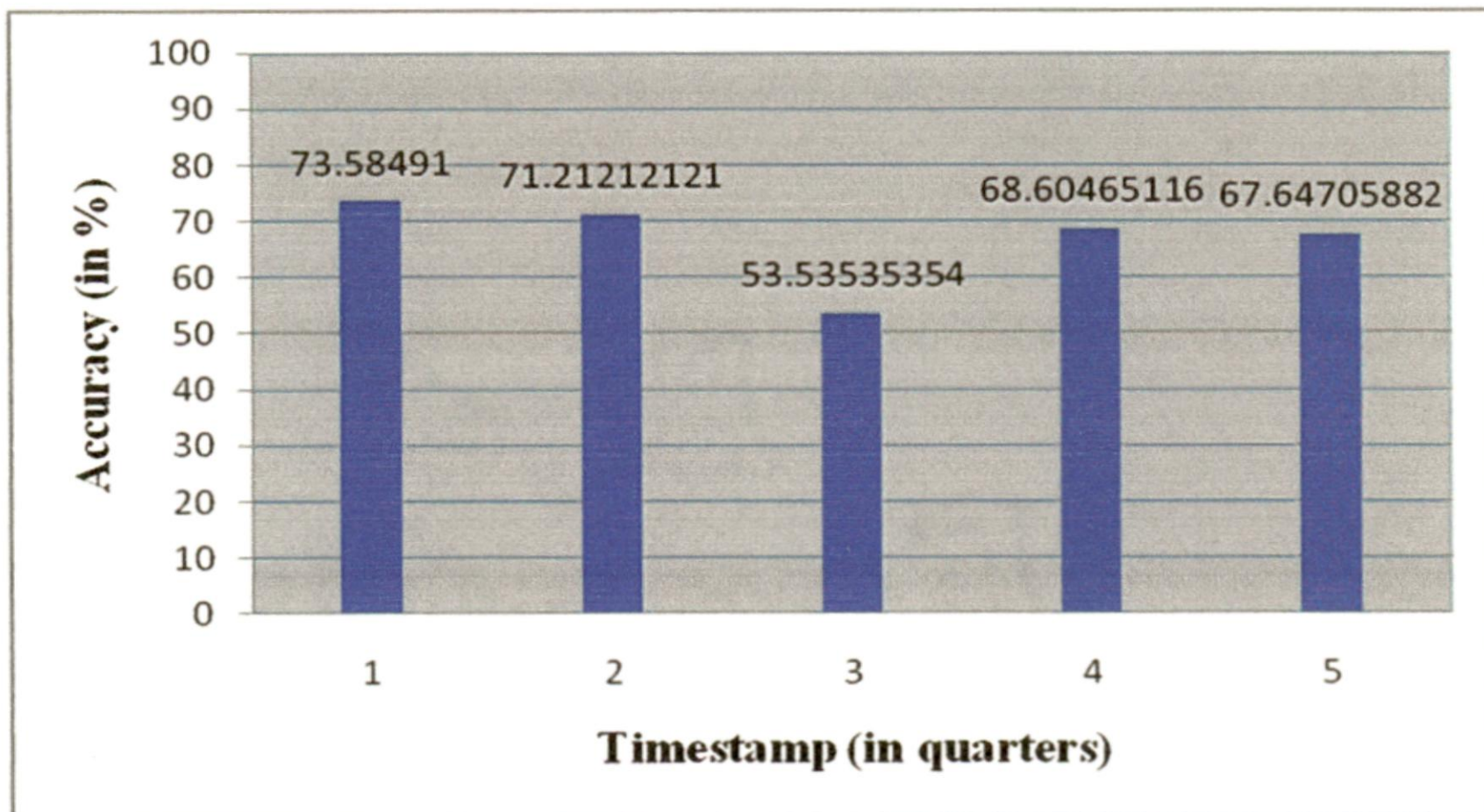


Figure 5.2: Classification Accuracy with static dictionary

In the incremental approach the training model gains new information over time. The test documents are taken in the same way as taken in case of static dictionary. The model is tested at each timestamp and the accuracy of classifier is calculated. The

terms in the dictionary are of equal weights. The accuracy of classifier is shown in Figure 5.3. It can be seen that the accuracy increases in incremental approach.

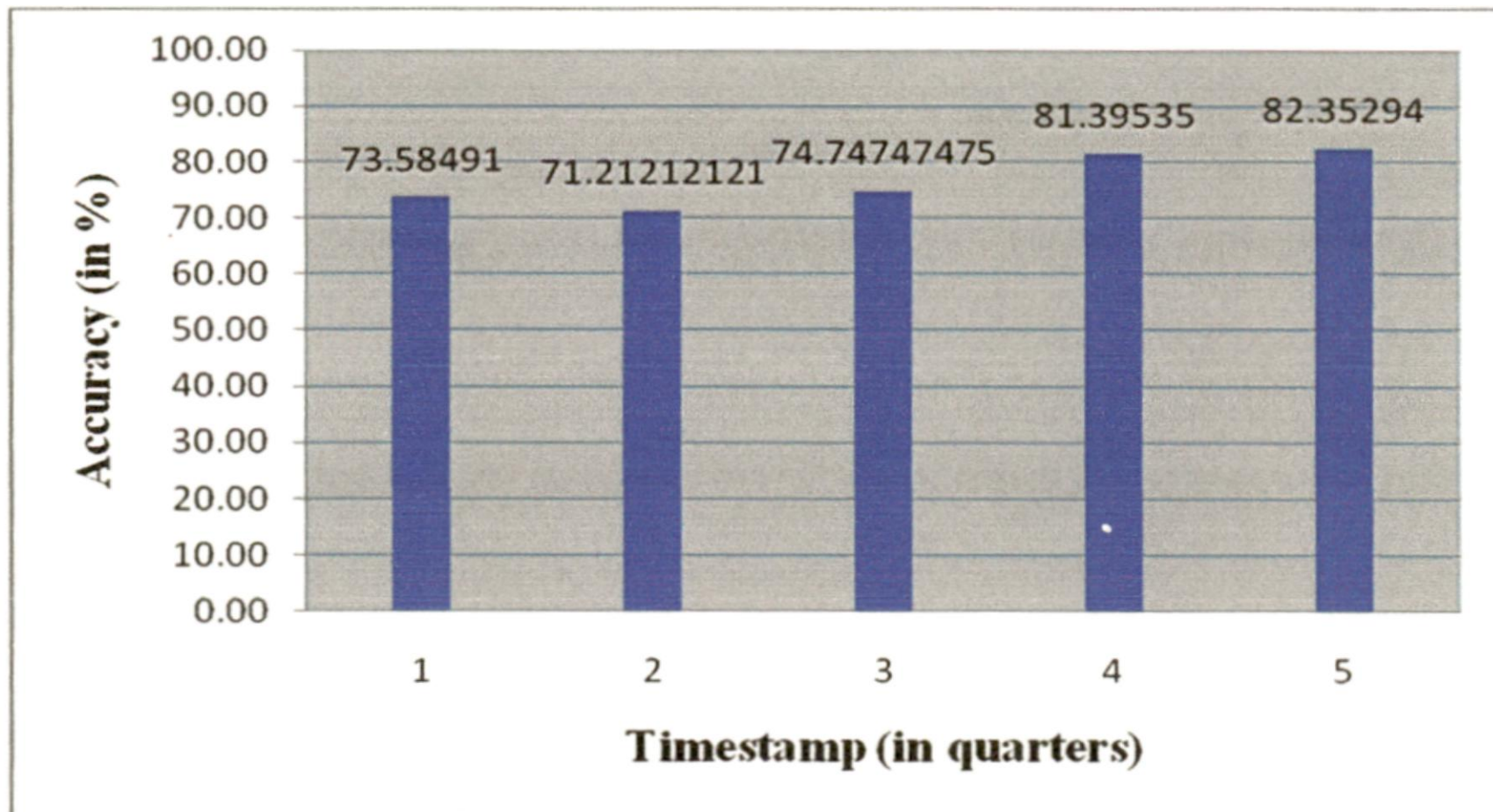


Figure 5.3: Classification Accuracy with dynamic dictionary

Another result can be shown with classification accuracy for classifier having dynamic dictionary and terms with different weights in dictionary, more recent terms are given more weights. The classification accuracy is shown in Figure 5.4.

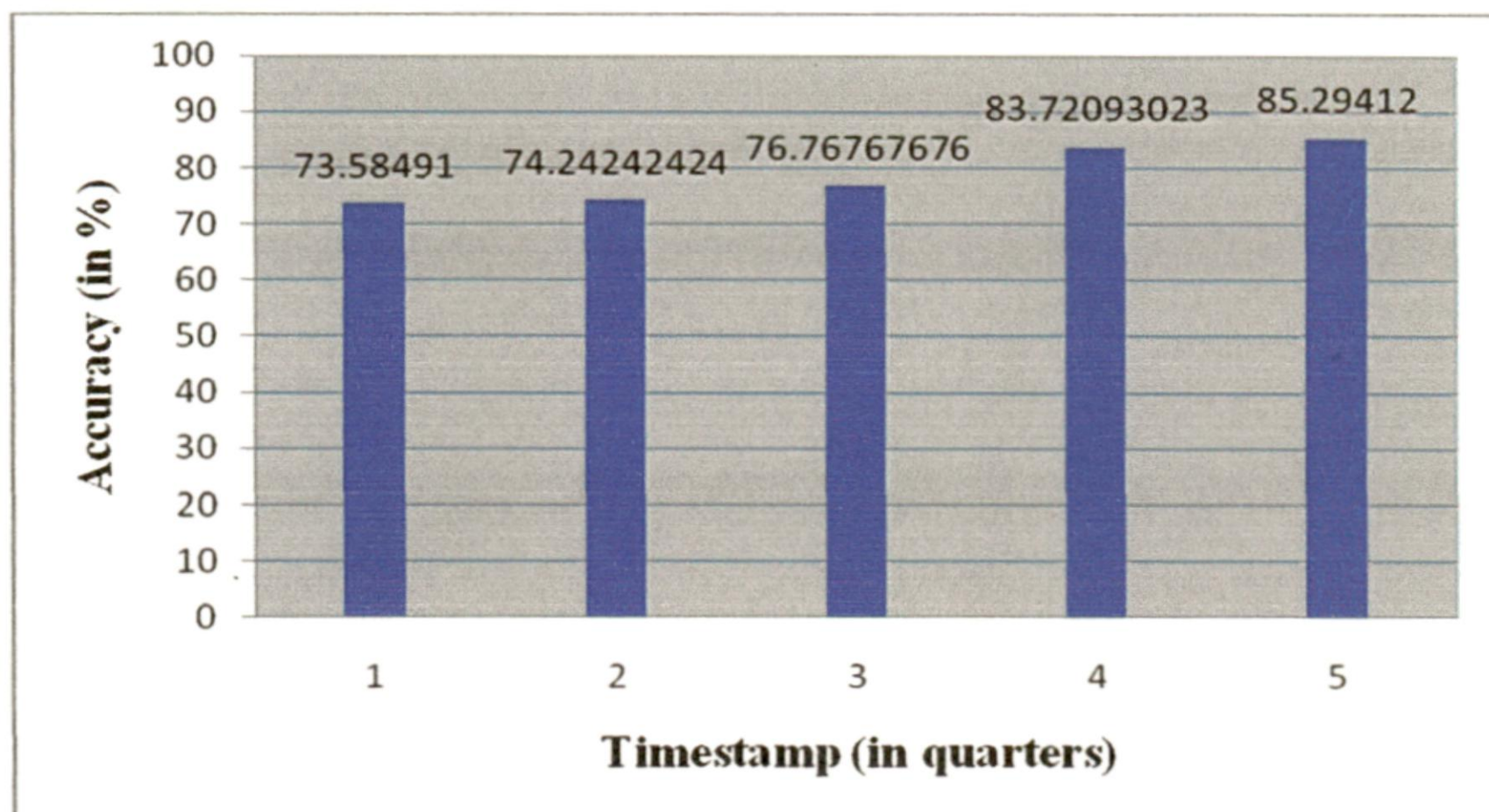


Figure 5.4: Classification Accuracy with weighted terms in dynamic dictionary

It can be seen that the classification accuracy increases with each timestamp. For each timestamp, the classification accuracy is lowest in case of static dictionary and is higher in case of dynamic dictionary.

In case with dictionary having different weights for terms the classification accuracy is higher in comparison to dictionary with equal weighted terms for each timestamp.

The term weights are chosen using a linear function. The most recent terms have unit weight and other term's weights decreases linearly using Eq. (21).

$$Wt_n = ((100-nD)/100)Wt_r, \quad n=1,2,3\dots \quad (21)$$

where Wt_r is the weight of most recent quarter terms, and Wt_n is weight of terms in previous quarter. The just previous quarter has $n=1$, and n increases with the quarter having older timestamp. D is the percentage decreasing rate like 5, 10, 20, 25, etc.

Figure 5.5 shows the classification accuracy at 5 timestamps. The 3 results are for the linear decrease of 5%, 10% and 15% in the term weights represented by blue, red and green colors respectively. As $D = 10$ has the highest accuracy, so it is chosen for the function shown in Eq. (21).

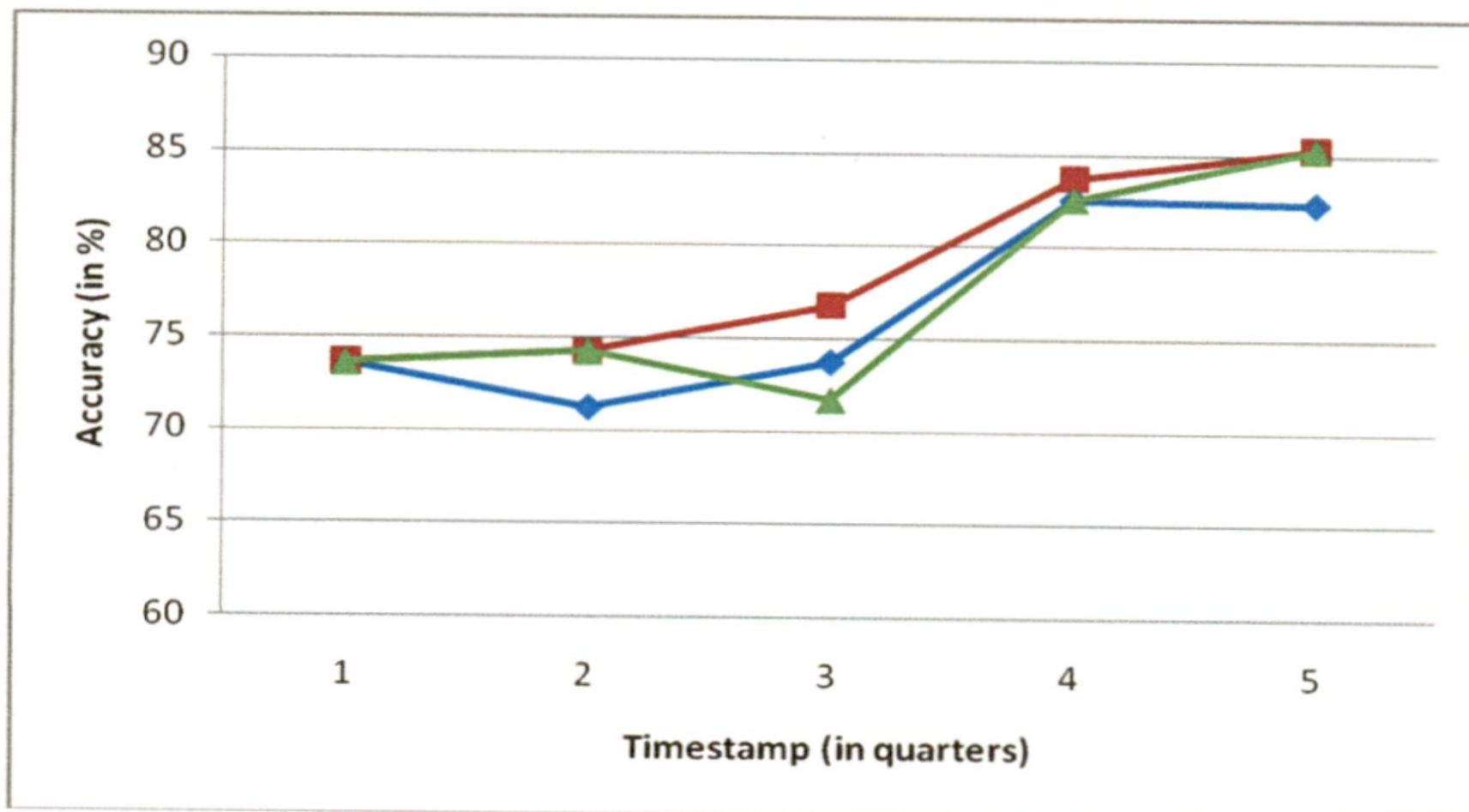


Figure 5.5: Linear Decrement of Term Weights by 5%, 10% and 15%

5.2 Analysis

The incremental approach text classification gives better results in comparison to text classification algorithm without increments. As time passes the information gained from classifier gives better classification accuracy. The cost of building the incremental text classifier will be more in comparison to classifier without increments but the greater accuracy is achieved in incremental approach. In case of classifier build on all the timestamp documents, the accuracy would be highest but the cost of training such a model will also be highest due to high dimensionality.

It is also seen that more weights to more recent terms in the dictionary gives better results than the dictionary with equal weights to all terms. Irrelevant terms get less weight in the dictionary of different weights and so the accuracy is higher in such case.

CHAPTER 6

CONCLUSION

6.1 Conclusion

Text documents are obtained from various sources like businesses, research, government and other organizations, as they store data in digital form. Classification is a supervised approach used to assign class labels to unseen data. When classification techniques are applied to text documents, then this process is known as text classification. Many applications use the data that keeps getting generated over time. For such applications, the traditional text classification methods may be incapable to deal with. Therefore, incremental approaches for text classification are required in these circumstances. One such application is the sports news documents which keep getting generated over time.

We focus our attention on incremental approach for text classification. First the text documents are preprocessed before applying classification techniques to them. While preprocessing, stopwords are removed and stemming of words are done. The porter's stemming analyzer has been used for the purpose of stemming words. After preprocessing, the documents are converted into vectors on the basis of term frequencies and inverse document frequencies. K-means clustering is applied to the training dataset, to obtain the class labels. The clustering also results into extraction of relevant terms for the dictionary. With each increment, new terms get added to the dictionary. The newly added terms are assigned unit weights whereas the weight for the terms in the dictionary is reduced linearly. This process continues as more and more data keeps getting generated. The idea of weights is used to show the incremental evolution of dictionary. As time passes, very old terms are removed from the dictionary as they get zero weights. We have applied our algorithm on real case data collected from Google sports news collected over fixed interval of 1 month and the increments are done quarterly. The proposed work has been verified on the

dataset as shown in the results and this work can also be used in other applications like e-mails.

6.2 Suggestions for further work

In our proposed work, we had used words as features. The semantic relation of words is not considered and we will use them in extension of the work.

Feature Clustering for dimension reduction can be applied along with the documents clustering for dimension reduction.

We can apply this approach on different applications and check whether it is useful for improving the classification accuracy in those applications.

This approach can be applied to applications in which abrupt changes occur. For handling abrupt and sudden changes, some enhancements in the algorithm needs to be done.

REFERENCES

- [1] J. Han and M. Kamber, "Data mining: concepts and techniques," Second Edition, Elsevier Inc., 2006, pp. 614-628.
- [2] R. Feldman and J. Sanger, "Text mining hand book, advanced approaches in analyzing unstructured data," Cambridge University Press, 2007, pp. 13-15.
- [3] I. S. Dhillon, J. Fan and Y. Guan, "Efficient clustering of very large document collections," In V. Kumar and C. Kamath and R. Grossman, *Data mining for scientific and engineering applications*, Dordrecht; Boston, 2001, pp. 1-25.
- [4] Z. Chen, L. Huang and Y. L. Murphey, "Incremental learning for text document classification," *Proc. of International Joint Conference on Neural Networks (IJCNN 2007)*, Orlando, Florida, USA, August 12-17, 2007, pp. 2592-2597.
- [5] Z. Yun-tao, G. Ling and W. Yong-cheng, "An improved TF-IDF approach for text classification," *Journal of Zhejiang University Science*, Vol. 6A, No. 1, August 2005, pp. 49-55.
- [6] J. Cai and F. Song, "Maximum entropy modeling with feature selection for text categorization," *4th Asia Information Retrieval Symposium (AIRS 2008)*, Harbin, China, Vol. 4993, January 15-18, 2008, pp. 549-554.
- [7] S. Yin, Y. Qiu and J. Ge, "Research and realization of text mining algorithm on web," *Proc. of International Conference on Computational Intelligence and Security Workshops 2007 (CISW 2007)*, 15-19 Dec. 2007, pp. 413-416.
- [8] J. Y. Jiang, J. W. Chen and S. J. Lee, "A clustering scheme for large high-dimensional document datasets," *Proc. of Advances in Computation and Intelligence, Second International Symposium (ISICA 2007)*, Wuhan, China, Vol. 4683, September 21-23, 2007, pp. 511-519.

- [9] X. Huang, M. Wu, D. Xia and P. Yan, "Difference similitude matrix in text classification," *Proc. International conference on Fuzzy systems and knowledge discovery*, Changsha, CHINA, Vol. 3614, 2005, pp. 21-30.
- [10] S. Yin, G. Wang, Y. Qiu and W. Zhang, "Research and implement of classification algorithm on web text mining," *Proc. Third International Conference on Semantics, Knowledge and Grid*, 29-31 Oct. 2007, pp. 446-449.
- [11] L. W. Lee and S. M. Chen, "New methods for text categorization based on a new feature selection method and a new similarity measure between documents," *Proc. of the 19th International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems*, Annecy, France, Vol. 4031, June 2006, pp. 432-441.
- [12] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine Learning*, Vol. V42, No. 1, 1 January 2001, pp. 143-175.
- [13] I. Dhillon, J. Kogan, C. Nicholas, "Feature selection and document clustering," *Survey of Text Mining 2004*, pp. 73-100.
- [14] M. F. Porter, "An algorithm for suffix stripping," *Program*, 14(3), 1980, pp. 130-137.
- [15] A. Kyriakopoulou, "Text classification aided by clustering: a literature review," *I-Tech Education and Publishing KG*, Vienna, Austria, 2008, pp. 233-252.
- [16] A. Kyriakopoulou, T. Kalamoukis, "Text classification using clustering," *Proc. of ECML-PKDD Discovery Challenge Workshop (2006)*, Berlin, Germany, September 22, 2006,

- [17] N. Slonim, N. Tishby, "The power of word clusters for text classification," *Proc. of 23rd European Colloquium on Information Retrieval Research, Darmstadt (ECIR-01)*, 2001, pp. 1-12.
- [18] Z. Minier, L. Csato, "Kernel PCA based clustering for inducing features in text categorization," *Proc. of European Symposium on Artificial Neural Networks, Bruges, Belgium, April 25-27, 2007*, pp. 349-354.
- [19] F. Sebastiani, "Text categorization," WIT Press, Southampton, UK, 2005, pp. 109-129.
- [20] P.N. Tan, M. Steinbach, V. Kumar, "Introduction to data mining," 2007, pp.50-55.
- [21] <http://www.news.google.com> [accessed on 30 May, 2009].
- [22] <http://sourceforge.net/projects/jtextpro/> [accessed on 1 May, 2009].
- [23] <http://tartarus.org/~martin/PorterStemmer/java.txt> [accessed on 1 May 2009].

APPENDIX: SOURCE CODE LISTING

TextClassifier.java

```
import java.io.IOException;
/*
 * The Text classifier class is to build the classification module
 */
public class TextClassifier {

    public static void main(String[] args) throws IOException {

        //Original Input Training Document list
        String TrainDocFile1 = "TrainingDocs1.txt";

        //Training Document list consisting TFIDF
        String TermFreqFile1 = "TermFrequencyDocs1.txt";

        //Document with all the unique terms with their IDs and IDF
        String GlobalFile1 = "GlobalTerms1.txt";

        //File containing all the centroids of a cluster
        String CentroidOfClusters1 = "CentroidOfClusters1.txt";

        //File containing category characteristics vectors
        String CatCharVect1 = "CategoryCharVector1.txt";

        //Total number of documents to be trained in each increment
        int NumTrainingDocs1 = 76;

        String[] docList1 = new String[NumTrainingDocs1];
        String[] TermFreqDocList1 = new String[NumTrainingDocs1];

        ReadFromFile rf = new ReadFromFile();
        try {
            docList1 = rf.ReadFromFile(TrainDocFile1, NumTrainingDocs1);
            TermFreqDocList1 = rf.ReadFromFile(TermFreqFile1, NumTrainingDocs1);
        } catch (IOException e) {
            System.out.println("Cannot read File");
        }

        //POS Tagging is done to convert the documents in tagged form
        Tagger tg = new Tagger();
        tg.main();

        //Preprocess the documents before clustering
        PreprocessData ppd = new PreprocessData();
```

```
ppd.ConvertIntoUniqueTerms(docList1,TermFreqDocList1,GlobalFile1);
```

```
int[][] centroid11 = { //football
                      {1,3},
                      {215,3},
                      {300,3},
                      {613,3},
                      };
int[][] centroid12 = { //cricket
                      {1979,3},
                      {1989,3},
                      {1996,3},
                      {2014,3},
                      };
int[][] centroid10 = { //blank centroid
                      {4,0},
                      {4,0},
                      {4,0},
                      {4,0},
                      };
```

```
//Cluster the documents on the basis of their Term's TFIDF to find the terms for
//categorization
Clustering cl = new Clustering();
cl.main(TermFreqDocList1,GlobalFile1,centroid11,centroid12,centroid10,centro
id10,centroid10,centroid10,CentroidOfClusers1,CatCharVect1,6);
}
}
```

PreprocessData.java

```
import java.io.File;
import java.io.IOException;
/*
 * This class remove stopwords, stem words and convert documents into words
 */
public class PreprocessData {

    public PreprocessData(){}

    public static void ConvertIntoUniqueTerms(String[] DocList1,String[]
        StemDocList1,String GlobalFile1)
    {
        CreateStopWordsDic cswd1 = new CreateStopWordsDic();
        Porter p = new Porter();
        WriteToFile wtf = new WriteToFile();

        String StopWordsFile1 = new String("StopWordsDic1.txt");
        String StopWordsFileNew1 = new String("NEWStopWordsDic1.txt");
```

```

String[] TagdocList1 = new String[StemDocList1.length];

//Read documents to be tagged
ReadFromFile rf = new ReadFromFile();
try {
    TagdocList1 = rf.ReadFromFl("FromTagger1.txt", StemDocList1.length);
} catch (IOException e) {
    System.out.println("Cannot read File");
}

//creating dictionary of stop words
cswd1.main(TagdocList1,StopWordsFile1);

//creating new stemmed dictionary of stop words(MANNUAL CHANGES ARE
//REQUIRED)
String StemmedStopWords1 = p.StemmedDoc(StopWordsFile1);
String[] StopWordsstr1 = StemmedStopWords1.split(" ", -1);
String[] dictionary1 = new String[StopWordsstr1.length];
StringBuffer str1 = new StringBuffer("");

for(int j=0; j<StopWordsstr1.length;j++)
    dictionary1[j] = "0";
for(int j=0;j<StopWordsstr1.length;j++)
    for(int i=0;i<StopWordsstr1.length;i++)
    {
        if(StopWordsstr1[j].length()<3)
            break;
        if(StopWordsstr1[j].equalsIgnoreCase(dictionary1[i]))
            break;
        if(dictionary1[i].equals("0"))
        {
            dictionary1[i]=StopWordsstr1[j];
            str1.append(dictionary1[i]);
            str1.append(" ");
            break;
        }
    }
StemmedStopWords1 = str1.toString();
File fl1 = new File(StopWordsFileNew1);
boolean exists1 = fl1.exists();
if(exists1 == true)
    fl1.delete();
wtf.WriteToFl(StopWordsFileNew1, StemmedStopWords1);

String[] StemedDoc1 = new String[StemDocList1.length];

int i=0;
int NumTrainingDocs1 = DocList1.length;

```



```

        for(i=0;i<NumTrainingDocs1;i++)
        {
            //System.out.print(" IPdoc "+ DocList1[i]+" OPdoc " +StemDocList1[i]);
            StemedDoc1[i] = p.StemmedDoc(DocList1[i]);
        }
        ConvertDocIntoTerms cdt1 = new ConvertDocIntoTerms();
        cdt1.main(StemedDoc1,StemDocList1,GlobalFile1,StopWordsFileNew1);
    }
}

```

CreateStopWordsDic.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
/*
 * The class is used to convert documents into words
 */
public class CreateStopWordsDic {

    int DicSize=3000;
    String zero;
    String[] dictionary1 = new String[DicSize];

    public CreateStopWordsDic()
    {
        zero = "0";
        for(int i = 0; i<DicSize; i++)
            dictionary1[i] = "0";
    }

    public void PrintAndWriteDictionary(String StopWordsFile)
    {
        System.out.println("Dictionary");
        for(int i=0;i<DicSize;i++)
        {
            if(dictionary1[i].equals(zero))
            {
                System.out.println("TotalTerms = " + i);
                break;
            }
            else
                System.out.println("    " + dictionary1[i]);
        }

        //writing to a file
        StringBuffer sbuf = new StringBuffer("");
        String str = "";
    }
}

```

```

for(int l=0; l<DicSize; l++)
{
    if(dictionary1[l].equals(zero))
        break;
    sbuf.append(dictionary1[l]);
    sbuf.append(" ");
}
str = sbuf.toString();
WriteToFile wf = new WriteToFile();
File fl1 = new File(StopWordsFile);
boolean exists1 = fl1.exists();
if(exists1 == true)
    fl1.delete();

wf.WriteToFl(StopWordsFile, str);
System.out.println("Dictionay of stopWords created");
}

```

```

public void getTerms(String FileName)
{
    String readf = "";
    String SplitWords = " ";
    String splitWords2 = "[/]";
    File file1 = new File(FileName);
    FileReader frd = null;

    try
    {
        frd = new FileReader(file1);
        BufferedReader bfrd = new BufferedReader(frd);
        try
        {
            while ((readf = bfrd.readLine())!= null)
            {
                String[] strr = readf.split(SplitWords, -1);

                for(int i=0;i<strr.length;i++)
                {
                    String temp = "";
                    if(strr[0].isEmpty())
                        break;
                    if(strr[i].contains("NNP"))
                    {
                        String[] st = strr[i].split(splitWords2, -1);
                        temp = st[0];
                    }
                    else
                        continue;
                    for(int j=0;j<DicSize;j++)
                    {

```

```

                if(temp.equalsIgnoreCase(dictionary1[j]))
                    break;
                if(dictionary1[j].equals(zero))
                {
                    dictionary1[j]=temp;
                    break;
                }
            }
        }
    }
}
catch (IOException e)
{
    System.out.println("\nSorry, an IOException occurred.
        Returning intermediate matrix.");
    e.printStackTrace();
}
catch (FileNotFoundException e)
{
    System.out.println("\nSorry, file not found. Returning intermediate
        matrix.");
    e.printStackTrace();
}
}

public static void main(String[] ListOfDocs,String StopWordsFile)
{
    CreateStopWordsDic c = new CreateStopWordsDic();
    for(int i=0;i<ListOfDocs.length;i++)
        c.getTerms(ListOfDocs[i]);

    c.PrintAndWriteDictionary(StopWordsFile);
}
}

```

ConvertDocIntoTerms.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ConvertDocIntoTerms {

    int VectorSize=1000;
    String zero;
    String[] LocalVector = new String[VectorSize];
    int[] TermsFreq = new int[VectorSize];

```

```

public void Initialize()
{
    zero = "0";
    for(int i = 0; i<VectorSize; i++)
    {
        LocalVector[i] = zero;
        TermsFreq[i] = 0;
    }
}

public void printDictionary()
{
    System.out.println("Term Frequency");
    for(int i=0;i<VectorSize;i++)
    {
        if(LocalVector[i].equals(zero))
        {
            System.out.println("TotalTerms = " + i);
            break;
        }
        else
            System.out.println(LocalVector[i] + " " + TermsFreq[i]);
    }
}

public String getTermNFreq(String StemmedFileData, String[] StopWords2)
{
    Initialize();
    String SplitWords = "[!]*[! ]*[,]*[, ]*[.]*.[ ]*[-]*[- ]*[*]*[* ]*[*]; ]*[*]*[* ]*[*]: ]*[/]*[/ ]*[(]*[( ]*D ]*[ ]*[ ]";
    String[] StopWords = {
        "almost", "also", "among", "awai", "and", "any",
        "although", "are", "anoth", "ani", "anyth", "alwai",
        "becaus", "but", "been", "becom", "becam",
        "sundai", "mondai", "tuesdai", "wednesdai",
        "januari", "februari", "jun", "juli", "mai", "sept",
    };

    Boolean AStopWord = false;
    int i=0,j=0;
    String[] str = StemmedFileData.split(SplitWords, -1);
    for( i=0;i<str.length;i++)
    {
        if(str[i].isEmpty())
            break;
        AStopWord = false;
        for(j=0;j<StopWords.length;j++)
        {
            if(str[i].equalsIgnoreCase(StopWords[j]))
                AStopWord = true;
        }
    }
}

```

```

    }

    for(j=0;j<StopWords2.length;j++)
    {
        if(strr[i].equalsIgnoreCase(StopWords2[j]))
            AStopWord = true;
    }

    if(strr[i].length()<3||strr[i].length()>20)
        AStopWord =true;

    for(j=0;j<VectorSize;j++)
    {
        if(AStopWord.equals(true))
            break;

        if(strr[i].equalsIgnoreCase(LocalVector[j]))
        {
            TermsFreq[j]++;
            break;
        }

        if(LocalVector[j].equals(zero))
        {
            LocalVector[j]=strr[i];
            TermsFreq[j]++;
            break;
        }
    }
}

StringBuffer sbuf = new StringBuffer("");
String str = "";
for(int l=0; l<VectorSize; l++)
{
    if(LocalVector[l].equals(zero))
        break;
    sbuf.append(LocalVector[l]);
    sbuf.append(" ");
    sbuf.append(TermsFreq[l]);
    sbuf.append(" ");
}
str = sbuf.toString();
return str;
}

public static void main(String[] StemmedDoc, String[] TermFreqDocs, String
GlobalTermsFile, String StopWordsFile)
{
    WriteToFile wf = new WriteToFile();

```

```

ReadFromFile rfl1 = new ReadFromFile();
String[] Readf = new String[1];
try {
    Readf = rfl1.ReadFromFl(StopWordsFile, 1);
} catch (IOException e) {
    System.out.println("Cannot Read File" + StopWordsFile);
}
String[] StopWords2 = Readf[0].split(" ", -1);

int i=0;
//creating a global vector
String zero = "0";
int GlobalVectorSize = 10000;
String[] GlobalVector = new String[GlobalVectorSize];
//The presence of term in all documents
int[] TermOccurs = new int[GlobalVectorSize];

for(i=0; i<GlobalVectorSize; i++)
{
    GlobalVector[i] = zero;
    TermOccurs[i] = 0;
}
//converting documents into unique terms with their respective frequencies
ConvertDocIntoTerms c = new ConvertDocIntoTerms();
String splitpattern = " ";
int TotalDocs = StemmedDoc.length;
double[] IDF = new double[GlobalVectorSize];

for(int l=0;l<StemmedDoc.length;l++)
{
    String TermNFreq = c.getTermNFreq(StemmedDoc[l], StopWords2);
    String[] Terms = TermNFreq.split(splitpattern);

    for(i=0;i<Terms.length;i=i+2)
    {
        for(int j=0;j<GlobalVectorSize;j++)
        {
            if(GlobalVector[j].equals(zero))
            {
                GlobalVector[j]=Terms[i];
                TermOccurs[j]++;
                Terms[i] = Integer.toString(j);
                break;
            }
            if(Terms[i].equals(GlobalVector[j]))
            {
                TermOccurs[j]++;
                Terms[i] = Integer.toString(j);
                break;
            }
        }
    }
}

```

```

    }
}

//Sorting the terms on the basis of their Ids in global vector
String temp = "";
for(i=0;i<Terms.length;i=i+2)
    for(int j=i+2;j<Terms.length;j=j+2)
    {
        if(Integer.parseInt(Terms[i])>Integer.parseInt(Terms[j]))
        {
            temp = Terms[i];
            Terms[i] = Terms[j];
            Terms[j]=temp;
            temp = Terms[i+1];
            Terms[i+1] = Terms[j+1];
            Terms[j+1] = temp;
        }
    }

StringBuffer sb = new StringBuffer("");
String sss = "";
System.out.println("no. of terms in doc " + l + " is: " + Terms.length);

for(i=0;i<Terms.length;i=i+2)
{
    sb.append(Terms[i]);
    sb.append(" ");
    sb.append(Terms[i+1]);
    sb.append(" ");
}
sss = sb.toString();

File fl1 = new File(TermFreqDocs[l]);
boolean exists1 = fl1.exists();
if(exists1 == true)
    fl1.delete();

wf.WriteToFl(TermFreqDocs[l], sss);

//System.out.println("l is = " + l +"OPDoc" + TermFreqDocs[l]);
}

StringBuffer sbuff = new StringBuffer("");
String globalstr = "";
int j=0;
for(i=0;i<GlobalVectorSize;i++)
{
    if(GlobalVector[i].equals(zero))
    {

```

```

        System.out.println("total unique terms are " + (i-1));
        break;
    }

    if(TermOccurs[i]==0)
        IDF[i]=0.0;
    else
        IDF[i] = Math.log((1+TotalDocs)/(double)TermOccurs[i]);

    if(TermOccurs[i]>Math.round(TotalDocs*.02))
    {
        System.out.println(j++ +" Term "+ GlobalVector[i]+ " Freq " +
            TermOccurs[i] +" IDF " + IDF[i]);

        sbuff.append(i);
        sbuff.append(" ");
        sbuff.append(GlobalVector[i]);
        sbuff.append(" ");
        sbuff.append(IDF[i]);
        sbuff.append(" ");
    }
}
globalstr = sbuff.toString();
File fl2 = new File(GlobalTermsFile);
boolean exists2 = fl2.exists();
if(exists2 == true)
    fl2.delete();

wf.WriteToFile(GlobalTermsFile, globalstr);
}
}

```

WriteToFile.java

```

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class WriteToFile {
    public void WriteToFile(String FileName, String data)
    {
        File file1 = new File(FileName);

        try
        {
            boolean DoesExist = file1.exists();
            if(DoesExist == false)
            {
                boolean success = file1.createNewFile();
            }
        }
    }
}

```



```

        //if(success)
            //System.out.println("\nCreating new file \"" + FileName +
            // "\" .....Done.");
    }
} catch (IOException e)
{
    System.out.println("\nSorry, an IOException occurred.
    Returning.");
    e.printStackTrace();
    return;
}

FileWriter fwr1 = null;
try
{
    fwr1 = new FileWriter(file1, true);
    BufferedWriter BufWr = new BufferedWriter(fwr1);
    BufWr.write(data);
    BufWr.close();
} catch (IOException e)
{
    System.err.println("\nSorry, an IOException occurred.
    Returning.");
    e.printStackTrace();
    return;
}
}
}

```

ReadFromFile.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ReadFromFile {
    public String[] ReadFromFl(String Filename, int StringLength) throws IOException
    {
        String readf = "";
        File file1 = new File(Filename);
        FileReader frd = new FileReader(file1);
        BufferedReader bfrd = new BufferedReader(frd);

        String[] ss = new String[StringLength];
        int i=0;
        while ((readf = bfrd.readLine()) != null)
        {
            ss[i] = readf.toString();
        }
    }
}

```

```

        i++;
    }
    return ss;
}
}

```

Tagger.java

```

import jtextpro.*;
/*
 * The Tagger is used to tag all the documents like seperating nouns, verbs,..etc.
 */
public class Tagger {

    public static void main() {

        String[] args1 = {"models", "JUN08_AUG08_BOXING/tr1.txt"};
        JTextProcessor.main(args1);
    }
}

```

Clustering.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;

public class Clustering {
    //It contains centroid and TFIDF of all documents
    HashMap<String, Double> map = new HashMap<String, Double>();
    //It contains the Term IDs and IDF
    HashMap<Integer, Double> global_map = new HashMap<Integer, Double>();
    //It contains the Term IDs and Terms
    HashMap<Integer, String> global_Terms_map = new HashMap<Integer, String>();
    HashMap<String, Double> Char_Vector_map = new HashMap<String, Double>();
    int TotalVectors = 0;

    public int[] ReadGlobalFile(String GlobalTermsFile)throws IOException
    {
        int i=0,j=0;
        String readf = "";
        String SplitPattern = " ";
        File file1 = new File(GlobalTermsFile);
        FileReader frd = new FileReader(file1);
        BufferedReader bfrd = new BufferedReader(frd);
        readf = bfrd.readLine();
        String[] ss = readf.split(SplitPattern, -1);
        int TotalTerms=ss.length/3;
    }
}

```

```

int[] TermIDs = new int[TotalTerms];

j=0;
for(i=0;i<ss.length-2;i=i+3)
{
    Integer val1 = Integer.parseInt(ss[i]);
    String val2 = ss[i+1];
    Double val3 = Double.parseDouble(ss[i+2]);
    TermIDs[j] =val1;
    global_map.put(new Integer(val1), new Double(val3));
    global_Terms_map.put(new Integer(val1), new String(val2));
    j++;
}
return TermIDs;
}

public int[] ReadTFFile(String TFFileName)throws IOException
{
    String readf = "";
    String SplitPattern = " ";
    File file1 = new File(TFFileName);
    FileReader frd = new FileReader(file1);
    BufferedReader bfrd = new BufferedReader(frd);
    readf = bfrd.readLine();
    String[] ss = readf.split(SplitPattern, -1);
    int[] sss = new int[ss.length];
    for(int i=0;i<ss.length-1;i++)
        sss[i] = Integer.parseInt(ss[i]);
    return sss;
}

public void CalculateTFIDF(String[] TFDocs, int[] TermIDs, int[][] centroid1, int[][]
    centroid2, int[][] centroid3, int[][] centroid4, int[][] centroid5, int[][]
    centroid6, int NoOfClusters)throws IOException
{
    int i=0,j=0,row=0;
    double TfIdf = 0.0;

    System.out.println("centroid1");
    for(j=0;j<4;j++)
        for(i=0;i<TermIDs.length;i++)
            if(centroid1[j][0]==TermIDs[i])
            {
                Double k = global_map.get(TermIDs[i]);
                TfIdf = centroid1[j][1]*k.doubleValue();
                map.put(row+"_"+TermIDs[i], new Double(TfIdf));
                System.out.println("map = "+ row + "_" + TermIDs[i] + ": "
                    + map.get(row+"_"+TermIDs[i]));
                break;
            }
}

```

```

map.put(row+"_cat", new Double("1"));
row++;

int[][] TFvector = new int[TFDocs.length][];
for(i=0;i<TFDocs.length;i++) //each document in the training set
{
    TFvector[i] = ReadTFFile(TFDocs[i]);
    row++;

    for(j=0;j<TFvector[i].length-1;j=j+2) //each term in the document
    {
        //each term present in the global vector
        for(int l=0;l<TermIDs.length;l++)
        {
            if(TFvector[i][j]==TermIDs[l])
            {
                Double k = global_map.get(TFvector[i][j]);
                Tfldf = TFvector[i][j+1]*k.doubleValue();
                map.put(row+"_ "+TFvector[i][j], new
                    Double(Tfldf));
                break;
            }
        }
    }
    map.put(row+"_cat", new Double("0"));
}
TotalVectors = row;
}

public void CreateClusters(int[] TermIDs, String CentroidOfClusters, int
                            NoOfClusters)
{
    WriteToFile wfl = new WriteToFile();

    boolean ClusterAgain = true; //checking whether next iteration needed or not
    double numerator1=0, numerator2=0, numerator3=0, numerator4=0,
        numerator5=0, numerator6=0;
    double denominator1=0, denominator2=0, denominator3=0, denominator4=0,
        denominator5=0, denominator6=0, denominator7=0;
    double CosSim1=0, CosSim2=0, CosSim3=0, CosSim4=0, CosSim5=0,
        CosSim6=0;

    double Cat = 0;
    int loopcounter=1;

    while(ClusterAgain == true)
    {
        System.out.println("counter = "+ loopcounter);
        ClusterAgain = false;
        for(int row=NoOfClusters;row<TotalVectors+1;row++)
        {

```

```

numerator1 =0 ;numerator2=0; numerator3 =0; numerator4=0;
numerator5=0; numerator6=0;
denominator1=0;denominator2=0;denominator3=0;denominator4=
0;denominator5=0;denominator6=0;denominator7=0;

```

```

System.out.println("The row is " + (row-NoOfClusters) + " cat = "
+ map.get(row+"_cat"));

```

```

for(int col=0; col<TermIDs.length;col++)
{
    Double d1 = map.get(row+"_"+TermIDs[col]);
    Double cat1 = map.get(0+"_"+TermIDs[col]);
    if(d1!=null)
        denominator1 = denominator1 +
            (d1.doubleValue()*d1.doubleValue());
    if(cat1!=null)
        denominator2 =denominator2 +
            (cat1.doubleValue()*cat1.doubleValue());
    if(d1==null)
        continue;
    if(cat1 ==null)
        continue;
    //System.out.println("TermIDs = "+ TermIDs[col]);
    numerator1 = numerator1 +
        d1.doubleValue()*cat1.doubleValue();
}
for(int col=0; col<TermIDs.length;col++)
{
    Double d2 = map.get(row+"_"+TermIDs[col]);
    Double cat2 = map.get(1+"_"+TermIDs[col]);
    if(cat2!=null)
        denominator3 =denominator3 +
            (cat2.doubleValue()*cat2.doubleValue());
    if(d2==null)
        continue;
    if(cat2 ==null)
        continue;
    //System.out.println("TermIDs = "+ TermIDs[col]);
    numerator2 = numerator2 +
        d2.doubleValue()*cat2.doubleValue();
}
for(int col=0; col<TermIDs.length;col++)
{
    Double d3 = map.get(row+"_"+TermIDs[col]);
    Double cat3 = map.get(2+"_"+TermIDs[col]);
    if(cat3!=null)
        denominator4 =denominator4 +
            (cat3.doubleValue()*cat3.doubleValue());
    if(d3==null)
        continue;
    if(cat3 ==null)

```

```

        continue;
//System.out.println("TermIDs = "+ TermIDs[col]);
        numerator3 = numerator3 +
                d3.doubleValue()*cat3.doubleValue();
    }
    for(int col=0; col<TermIDs.length;col++)
    {
        Double d4 = map.get(row+"_"+TermIDs[col]);
        Double cat4 = map.get(3+"_"+TermIDs[col]);
        if(cat4!=null)
            denominator5 =denominator5 +
                (cat4.doubleValue()*cat4.doubleValue());
        if(d4==null)
            continue;
        if(cat4 ==null)
            continue;
//System.out.println("TermIDs = "+ TermIDs[col]);
        numerator4 = numerator4 +
                d4.doubleValue()*cat4.doubleValue();
    }
    for(int col=0; col<TermIDs.length;col++)
    {
        Double d5 = map.get(row+"_"+TermIDs[col]);
        Double cat5 = map.get(4+"_"+TermIDs[col]);
        if(cat5!=null)
            denominator6 =denominator6 +
                (cat5.doubleValue()*cat5.doubleValue());
        if(d5==null)
            continue;
        if(cat5 ==null)
            continue;
//System.out.println("TermIDs = "+ TermIDs[col]);
        numerator5 = numerator5 +
                d5.doubleValue()*cat5.doubleValue();
    }
    for(int col=0; col<TermIDs.length;col++)
    {
        Double d6 = map.get(row+"_"+TermIDs[col]);
        Double cat6 = map.get(5+"_"+TermIDs[col]);
        if(cat6!=null)
            denominator7 =denominator7 +
                (cat6.doubleValue()*cat6.doubleValue());
        if(d6==null)
            continue;
        if(cat6 ==null)
            continue;
        numerator6 = numerator6 +
                d6.doubleValue()*cat6.doubleValue();
    }
    CosSim1 = numerator1/ (Math.sqrt(denominator1))*

```

```

        Math.sqrt(denominator2));
    double MaxCosSim = 0.0;
    double ThreshLimit = 0.0;
    Cat = 0;
    if(CosSim1 > MaxCosSim && CosSim1 > ThreshLimit)
    {
        MaxCosSim = CosSim1;
        Cat = 1;
    }

    Double category = map.get(row + "_cat");
    if(category.doubleValue() != Cat)
    {
        ClusterAgain = true;
        map.remove(row + "_cat");
        map.put(row + "_cat", new Double(Cat));
    }
}
//calculate centroids
double centroid1 = 0;
double num1 = 0, den1 = 0;

for(int col = 0; col < TermIDs.length; col++)
{
    num1 = 0; den1 = 0;
    for(int row = NoOfClusters; row < TotalVectors + 1; row++)
    {
        Double category = map.get(row + "_cat");
        Double d1 = map.get(row + "_" + TermIDs[col]);

        if(category.doubleValue() == 1)
        {
            if(d1 == null)
                continue;
            num1 = num1 + d1.doubleValue();
            den1 = den1 + (d1.doubleValue() * d1.doubleValue());
        }
    }
    if(den1 == 0)
        centroid1 = 0;
    else
        centroid1 = num1 / (Math.sqrt(den1));

    map.remove("0_" + TermIDs[col]);
    map.remove("1_" + TermIDs[col]);
    map.remove("2_" + TermIDs[col]);
    map.remove("3_" + TermIDs[col]);
    map.remove("4_" + TermIDs[col]);
    map.remove("5_" + TermIDs[col]);
}

```

```

if(centroid1!=0.0||centroid2!=0.0||centroid3!=0.0||centroid4!=0.0||ce
ntroid5!=0.0||centroid6!=0.0)
{
    if(centroid1>centroid2&&centroid1>centroid3&&centroid1
>centroid4&&centroid1>centroid5&&centroid1>centroid6)
        map.put("0_" + TermIDs[col], new
            Double(centroid1));
    elseif(centroid2>centroid3&&centroid2>centroid4&&centro
id2>centroid5&&centroid2>centroid6)
        map.put("1_" + TermIDs[col], new
            Double(centroid2));
    elseif(centroid3>centroid4&&centroid3>centroid5&&centro
id3>centroid6)
        map.put("2_" + TermIDs[col], new
            Double(centroid3));
    else if(centroid4>centroid5&&centroid4>centroid6)
        map.put("3_" + TermIDs[col], new
            Double(centroid4));
    else if(centroid5>centroid6)
        map.put("4_" + TermIDs[col], new
            Double(centroid5));
    else
        map.put("5_" + TermIDs[col], new
            Double(centroid6));
}
}
loopcounter++;
}

boolean[] CentroidsExist = new boolean[NoOfClusters];

for(int row =0; row<NoOfClusters;row++)
{
    CentroidsExist[row] = false;
    for(int col=0;col<TermIDs.length;col++)
    {
        Double d1 = map.get(row+"_" +TermIDs[col]);
        if(d1!= null)
        {
            CentroidsExist[row] = true;
            break;
        }
    }
}

StringBuffer sbuf1 = new StringBuffer("");
String ss = "";

for(int row =0; row<NoOfClusters; row++)

```



```

    {
        if(CentroidsExist[row] == false)
            continue;
        for(int col=0;col<TermIDs.length;col++)
        {
            Double dd1 = map.get(row+"_"+TermIDs[col]);
            String ss1 = global_Terms_map.get(TermIDs[col]);

            if(dd1 == null)
                continue;
            else
            {
                sbuf1.append(ss1);
                sbuf1.append(" ");
                sbuf1.append(dd1);
                sbuf1.append(" ");
            }
        }
        sbuf1.append("\n");
    }
    File fl1 = new File(CentroidOfClusters);
    boolean exists1 = fl1.exists();
    if(exists1 == true)
        fl1.delete();

    ss = sbuf1.toString();
    wfl.WriteToFile(CentroidOfClusters, ss);
}

public static void main(String[] TFDocs, String GlobalTermsDoc, int[][] centroid1,
                        int[][] centroid2, int[][] centroid3, int[][] centroid4, int[][]
                        centroid5, int[][] centroid6, String CentroidOfClusters, String
                        CatCharVect, int NoOfClusters) throws IOException
{
    Clustering cl = new Clustering();
    int[] TermIDs = cl.ReadGlobalFile(GlobalTermsDoc);

    cl.CalculateTFIDF(TFDocs, TermIDs, centroid1, centroid2, centroid3, centroid4, centroid
                    5, centroid6, NoOfClusters);
    cl.CreateClusters(TermIDs, CentroidOfClusters, NoOfClusters);
}
}

```

TestingDocuments.java

```

package ReportCode;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

```

```

import java.io.IOException;

public class TestingDocument {

    int TotalTermsInDic =5000;
    int CatVectorSize = 550;
    String[][] catv1 = new String[CatVectorSize][2];
    String[][] catv2 = new String[CatVectorSize][2];
    String[][] catv3 = new String[CatVectorSize][2];
    String[][] CommonDictionary = new String[TotalTermsInDic][3]; //Terms IDFWts

    public TestingDocument()
    {
        for(int i=0;i<TotalTermsInDic;i++)
            for(int j=0;j<3;j++)
                CommonDictionary[i][j] = "0";
        for(int i=0;i<CatVectorSize;i++)
            for(int j=0;j<2;j++)
            {
                catv1[i][j] = "0";
                catv2[i][j] = "0";
                catv3[i][j] = "0";
            }
    }

    //This function creates dictionary and get category characteristic vectors eith their
    //TFIDF not weighted
    public void GetDictionary(String CentroidOfClustFile,int TrainingDataSet, String[][]
        TermNIDFs)
    {
        int i=0,j=0;
        ReadFromFile rfl = new ReadFromFile();
        String[] SS1 = new String[6]; //reading file content in SS1
        String[] vect1 = new String[1100];
        String[] vect2 = new String[1100];
        String[] vect3 = new String[1100];

        String[][] TempCatv1 = new String[CatVectorSize][2]; //Training set vector
        String[][] TempCatv2 = new String[CatVectorSize][2]; //Training set vector
        String[][] TempCatv3 = new String[CatVectorSize][2]; //Training set vector

        for(i=0;i<CatVectorSize;i++)
            for(j=0;j<2;j++)
            {
                TempCatv1[i][j] = "0";
                TempCatv2[i][j] = "0";
                TempCatv3[i][j] = "0";
            }

        for(i=0;i<1100;i++)

```

```

{
    vect1[i] = "0";
    vect2[i] = "0";
    vect3[i] = "0";
}

int NoOFCategories = 0;
String Wt = "1.0";

if(TrainingDataSet == 5)
{
    try {
        SS1 = rf1.ReadFromF1(CentroidOfClustFile, 3);
    } catch (IOException e) {
        System.out.println("File cannot be read");
        e.printStackTrace();
    }

    NoOFCategories=3;
    vect1 = SS1[0].split(" ", -1);
    vect2 = SS1[1].split(" ", -1);
    vect3 = SS1[2].split(" ", -1);

    System.out.println("vect1 size is " + vect1.length);
    System.out.println("vect2 size is " + vect2.length);
    System.out.println("vect3 size is " + vect3.length);

    Wt = "0.6";
}

for(i=0;i<vect1.length-1;i=i+2)
{
    TempCatv1[i/2][0] = vect1[i];
    TempCatv1[i/2][1] = vect1[i+1];
    for(int k = 0; k<TotalTermsInDic;k++)
    {
        if(vect1[i].equalsIgnoreCase(CommonDictionary[k][0]))
        {
            CommonDictionary[k][2] = Wt;
            for(int counter=0;counter<TermNIDFs.length;counter++)
            {
                if(vect1[i].equalsIgnoreCase(TermNIDFs[counter][0]))
                {
                    CommonDictionary[k][1] =
                        TermNIDFs[counter][1];
                    break;
                }
            }
            else
                continue;
        }
    }
}

```

```

    }
    break;
}
if(CommonDictionary[k][0].equals("0"))
{
    CommonDictionary[k][0] = vect1[i];
    CommonDictionary[k][2] = Wt;
    for(int counter=0;counter<TermNIDFs.length;counter++)
    {
        if(vect1[i].equalsIgnoreCase(TermNIDFs[counter][0]))
        {
            CommonDictionary[k][1] =
                TermNIDFs[counter][1];
            break;
        }
        else
            continue;
    }
    break;
}
}
}
}

```

```

if(TrainingDataSet == 5)
{
    for(i=0;i<CatVectorSize;i++)
        for(j=0;j<2;j++)
        {
            if(TempCatv1.equals("0"))
                break;
            else
                catv1[i][j]=TempCatv1[i][j];
        }
    for(i=0;i<CatVectorSize;i++)
        for(j=0;j<2;j++)
        {
            if(TempCatv2.equals("0"))
                break;
            else
                catv2[i][j]=TempCatv2[i][j];
        }
    for(i=0;i<CatVectorSize;i++)
        for(j=0;j<2;j++)
        {
            if(TempCatv3.equals("0"))
                break;
            else
                catv3[i][j]=TempCatv3[i][j];
        }
}
}
}
}

```

```

    }
}

public void PrintDic()
{
    int i=0;
    System.out.println("dictionary is");
    for(i=0;i<TotalTermsInDic;i++)
    {
        if(CommonDictionary[i][0].equals("0"))
            break;
    }
    TotalTermsInDic = i;
}

public String[] GetDicTerms()
{
    String[] DicTerms = new String[TotalTermsInDic];
    System.out.println("total terms " + TotalTermsInDic);
    for(int i=0;i<TotalTermsInDic;i++)
    {
        DicTerms[i] = CommonDictionary[i][0];
    }
    return DicTerms;
}

public String[][] GetIDF(String GlobalTermsFile)
{
    int i=0,j=0;
    String readf = "";
    File file1 = new File(GlobalTermsFile);
    FileReader frd;
    try {
        frd = new FileReader(file1);
        BufferedReader bfrd = new BufferedReader(frd);
        readf = bfrd.readLine();
    } catch (FileNotFoundException e) {
        System.out.println("File not found " + GlobalTermsFile);
    } catch (IOException e) {
        System.out.println("IOException");
    }

    String[] ss = readf.split(" ", -1);
    int TotalTerms=ss.length/3;
    String[][] TermWithIDFs = new String[TotalTerms][2];

    j=0;
    for(i=0;i<ss.length-2;i=i+3)
    {
        TermWithIDFs[j][0] = ss[i+1];

```

```

        TermWithIDFs[j][1] = ss[i+2];
        j++;
    }
    return TermWithIDFs;
}

public double[] GetWtTfIdfVector(int[] TermFreq)
{
    double[] TfIdf = new double[TotalTermsInDic];
    for(int i=0;i<TotalTermsInDic;i++)
    {
        TfIdf[i] = TermFreq[i]*Double.parseDouble(CommonDictionary[i][1])
                    *Double.parseDouble(CommonDictionary[i][2]);
    }
    return TfIdf;
}

public double CosineSimilarity(double[] WtTfIdf1, int CatVectorNo)
{
    int VectorSize = 0;
    double numerator1 =0 ;
    double denominator1=0,denominator2=0;
    double CosSim = 0;

    String[][] CategoryVector = new String[CatVectorSize][2];
    double[] TestDoc = new double[CatVectorSize];
    double[] Weight = new double[CatVectorSize];

    for(int i=0;i<CatVectorSize;i++)
        for(int j=0;j<2;j++)
        {
            if(CatVectorNo == 1)
                CategoryVector[i][j]= catv1[i][j];
            else if(CatVectorNo == 2)
                CategoryVector[i][j]= catv2[i][j];
        }

    for(int i=0;i<CatVectorSize;i++)
    {
        if(CategoryVector[i][0].equals("0"))
        {
            VectorSize = i;
            break;
        }
        for(int j=0;j<TotalTermsInDic;j++)
        {
            if(CategoryVector[i][0].equalsIgnoreCase(CommonDictionary[j][0]))
            {

```

```

        TestDoc[i] = WtTfIdf1[j];
        Weight[i] = Double.parseDouble(CommonDictionary[j][2]);
        break;
    }
}

for(int i=0;i<VectorSize;i++)
{
    double CatVectValue = Double.parseDouble(CategoryVector[i][1])
        *Weight[i];
    denominator2 = denominator2 + Math.pow(CatVectValue,2);

    if(TestDoc[i]==0.0)
        continue;
    else
    {
        numerator1 = numerator1 + TestDoc[i]*CatVectValue;
        denominator1 = denominator1 + TestDoc[i]*TestDoc[i];
    }
}
CosSim = numerator1/(Math.sqrt(denominator1)*Math.sqrt(denominator2));
return CosSim;
}

```

```

public static void main(String[] args) {

    TestingDocument td = new TestingDocument();
    GetTestVector gtv = new GetTestVector();

    //File containing Terms IDs, Terms, IDFs
    String GlobalFile1 = "GlobalTerms1.txt";

    //File containing all the centroids of a cluster
    String CentroidOfClusters1 = "CentroidOfClusters1.txt";

    //Timestamp1
    System.out.println("File = " + CentroidOfClusters1);
    String[][] TermNIDFs1 = td.GetIDF(GlobalFile1);
    td.GetDictionary(CentroidOfClusters1,1,TermNIDFs1);

    td.PrintDic();

    String TestDocFileNames = "TestDocument2.txt";
    int TotalTestDoc = 86;

    String[] DicTerms = td.GetDicTerms();

    String[] Testdoc = new String[TotalTestDoc];
}

```

```

ReadFromFile rf = new ReadFromFile();
try {
    Testdoc = rf.ReadFromFile(TestDocFileNames, TotalTestDoc);
} catch (IOException e) {
    System.out.println("Cannot read File" + TestDocFileNames);
}

double MaxCosSim = 0.0;
double Thresholdlimit = 0.05;

int CorrectCategories = 0;
for(int i=0;i<TotalTestDoc;i++)
{
    System.out.println("Document is " + Testdoc[i]);
    //only term freq, the terms are identified as index of the array
    int[] TermFreq = gtv.getTermFrequency(Testdoc[i], DicTerms);
    double[] WtTfIdf = td.GetWtTfIdfVector(TermFreq);

    String Category = "null";
    double CosSim1 = td.CosineSimilarity(WtTfIdf,1);

    MaxCosSim =0.0;
    if(CosSim1>MaxCosSim)
    {
        MaxCosSim = CosSim1;
        if(MaxCosSim>Thresholdlimit)
            Category = "Football";
    }

    if(MaxCosSim<Thresholdlimit)
        System.out.println("Test Document is not classified in any
                                category.");

    else
        System.out.println("Test Document Belongs to " + Category + "
                                category And " + "MaxCosSim= " + MaxCosSim);
}
}
}

```

GetTestVector.java

```

package ReportCode;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class GetTestVector {
    public int[] getTermFrequency(String FileName, String[] Dictionary)

```



```

{
String SplitWords = "[!]*[! ]*[,]*[ , ]*[.]*[. ]*[-]*[- ]*[\"']*[\"' ]*[:]*[: ]*[/]*[/ ]*[()]*[( )]*D]*D ]*[ ]*[ ]";
ReadFromFile rf = new ReadFromFile();
Porter p = new Porter();
String StemmedData = p.StemmedDoc(FileName);
int i=0,j=0;
String[] strr = StemmedData.split(SplitWords, -1);

int DicSize = Dictionary.length;
int[] TermFrequency = new int[DicSize];
for( i=0;i<strr.length;i++)
{
    if(strr[i].isEmpty())
        break;

    for(j=0;j<DicSize;j++)
    {
        if(strr[i].equalsIgnoreCase(Dictionary[j]))
        {
            TermFrequency[j]++;
            break;
        }
    }
}
return TermFrequency;
}
}
}

```