MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

Session 2015-16

Department of Computer Science and Engineering

Major Project on

Document Classification Using Multinomial Naïve Bayes Model

Submitted By:

Anand Namdev	121112118
Amit Birla	121112030
Adish Jain	121112085
Suneel Kumar	121112021

Under the Guidance of

Dr. S.K Saritha

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

DECLARATION

We, hereby, declare that the following report which is being presented in the Major Project Documentation entitled **Document Classification Using Multinomial Naïve Bayes Model** is the partial fulfillment of the requirements of the final year (eight semester) **Major Project** in the field of **Computer Science And Engineering.** It is an authentic documentation of our own original work carried out under the Guidance of **Dr. S. K Saritha** .None of matter contained therein has been copied or extracted from anywhere else. The work has been carried out entirely at **Maulana Azad National Institute of Technology, Bhopal**. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

Anand Namdev	121112118
Amit Birla	121112030
Adish Jain	121112085
Suneel Kumar	121112021

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

CERTIFICATE

Session 2015-16

This is to certify that **Anand Namedev, Amit Birla, Adish Jain, Suneel Kumar** students of B.Tech final year (Computer Science And Engineering) have successfully completed their project "**Document Classification Using Multinomial Naïve Bayes Model**" in partial fulfillment of their major project in Computer Science and Engineering.

Dr. S.K Saritha

(PROJECT GUIDE)

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected and learned guide **Dr. S.K Saritha** , for her valuable help and guidance. We are thankful for the encouragement that she has given us in completing this project successfully. Her rigorous evaluation and constructive criticism was of great assistance. It is imperative for us to mention the fact that this major project could not have been accomplished without the periodic suggestions and advice of our project co- ordinator. We are also grateful to our respected director **Dr.Appu Kuttan K.K.** for permitting us to utilize all the necessary facilities of the college. Needless to mention is the additional help and support extended by our respected HOD, **Dr. R.K. Pateria,** in allowing us to use the departmental laboratories and other services. We also thank our project coordinator **Dr. Manish Pandey** for helping us throughout the project. We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help. Last but certainly not the least, we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

Anand Namdev	121112118
Amit Birla	121112030
Adish Jain	121112085
Suneel Kumar	121112021

Table of Content

List of Figures

List of Tables

List of Charts

Abstract

Classification is the process of dividing the data into number of groups which are either dependent or independent of each other and each group acts as a class. This project represents classification of text which is commonly known as document classification. The main purpose of this paper is to analyze the task document classification and to learn that how can we achieve high classification accuracy in the context of text documents. This task has several applications, including automated indexing of scientific articles according to technical terms, filing patents into patent directories, automated population of hierarchical catalogues of Web resources, spam filtering, identification of document genre, authorship attribution and even automated essay grading. Automated text classification is attractive because it frees organizations from the need of manually organizing the documents, which can be too expensive, or simply not feasible given the time constraints of the application or the number of documents involved. Naive Bayes approach is used to deal with the problem of document classification with term weighting as a additional technique to increase the performance measures like accuracy, precision and recall. The dataset for the evaluation purpose is collected from UCI repository dataset in which some changes have been done from our side. In this paper we compare and evaluate the performance of one of the supervised document classification model called as the naïve bayes technique by using different standard performance measures and a standard corpus.

1. Introduction:

Data Mining is the process of applying machine learning techniques for automatically or semi automatically analyzing and extracting knowledge from stored data. It is defined as non-trivial extraction of implicit, novel and actionable knowledge from large datasets. Data mining can also be defined as technology which enables data analysis, exploration and visualization of very large databases using high level of abstraction. Data mining models can be categorized as predictive models and descriptive models. Classification is one of the important aspects of data mining which is a predictive modeling technique. Classification techniques are used in various real world problems with respect to application domain as well as for various research purposes. It is used to group the data instances into proper class i.e. Classify them. Classification is used to build structures from examples of past decisions that can be used to make decisions for unseen or future cases. Various algorithms which are used for classification are decision tree learning, nearest neighbor, naïve's bayes classification, neural network, support vector machine. Classification techniques are used for web page classification, spam email filtration etc. Text categorization is the task of automated assigning of texts to predefined categories based on their content. Text categorization is the primary requirement of Text Retrieval systems. As the amount of online text increases, the demand for text categorization for the analysis and management of text is increasing. Though the text is cheap, it is expensive to get the information that, to which class a text belongs to. This information can be obtained from automatic categorization of text at low cost, but building the classifier itself is expensive because it require a lot of human effort or it must be trained from texts which have themselves been manually classified. Document Classification is one of the aspect of text categorization is a fundamental learning problem of many information management and retrieval tasks. In the task of document classification when more than two classes exists then it can be termed as multi-class document classification. It is usually performed in two stages: 1) the training phase and 2) the testing phase. During the training phase, sample documents are provided to the document classifier for each predefined category. The classifier uses machine learning algorithms to learn a class prediction model based on these labeled documents. In the testing phase, unlabeled documents are provided to the classifier, which applies its classification model to determine the categories or classes of the unseen documents. This training-testing approach makes the process of document classification a supervised learning task where unlabeled documents are categorized into known categories. There exist so many algorithms based on the Naïve Bayes

Classifier to classify text. For applying Naïve Bayes Classifier, each word position in a document must be treated as an attribute and the value of that attribute to be the word found in that position. Naïve Bayes categorization is given by equation (1):

Pr(Category Word) = (Pr (Word/ Category) * Pr (Category))/ Pr (Word)
Where, Pr=probability

In our approach firstly we have used General Naïve Bayes approach for classifying the documents. The system is given a set of example documents which are used to train the classifier. For preprocessing the text documents stop words are removed. Then collection of frequently occurring words from each document is done. This is done by matching each word of the training document with the words contained in pre-defined vocabulary. The vocabulary is the collection of feature set from the training set documents which can be built by using any feature set extraction method. Then new documents are classified using Naïve Bayes approach but using derived feature sets. As the training dataset is arranged in a linear manner and the classification is performed in linear manner we will call this approach as flat or linear classification.

2. Document Classification

Text classification, also known as text categorization or topic spotting, is a supervised learning task, where pre-defined category labels are assigned to documents based on the likelihood suggested by a training set of labelled documents. Until this machine learning approach to text categorization, the most popular approach was knowledge engineering. In knowledge engineering, expert knowledge is used to define manually a set of rules on how to classify documents under the pre-defined categories. It is evident that the machine learning approach to document classification leads to time and cost savings in terms of expert manpower without loss in accuracy.

In the problem of text classification we have a set D of documents and a set C of pre-defined categories. The aim is to assign a Boolean value to each $<d_i, c_{ji}>$ pair, where $d_i \in D$ and $c_j \in C$. A value of true assigned to $<d_i, c_{ji}>$ stands for the decision of assigning document d_i to category c_j. While value of false assigned to $<d_i, c_{ji}>$ stands for the decision of not assigning document d_i to category c_j. To state more formally, the task is to approximate the unknown target function $f : D \times C \rightarrow \{true, false\}$, that describes the way the documents should actually be classified, that is the number of the true positives and the number of false negative should be maximized by the function.

Text categorization has many interesting application areas such as document organization, text filtering and hierarchical categorization of Web pages. Document organization is the task of structuring documents of a corporate document base into predefined classes. For instance, advertisements incoming to a newspaper office may be classified into categories such as Cars, Real Estate, Computers, and so on before publication. Text filtering is the process of classifying a dynamic collection of text documents into two disjoint categories as relevant and irrelevant. An example is a newsfeed system where news articles incoming to a newspaper from a news agency such as Reuters are filtered. If it is a sports newspaper, delivery of news articles not related to sport are blocked. Similarly, an email filter may be trained to classify incoming messages as spam or not spam, and block the delivery of spam messages.

2.1 The basic Picture

TC may be formalized as the task of approximating the unknown target function $\Phi: D \times C \rightarrow \{T, F\}$ (that describes how documents are to be classified) by means of a function Φ: ˆ $D \times C \rightarrow \{T, F\}$

called the classifier, where C = {c1. . . c|C|} is a predefined set of categories and D is a (possibly infinite) set of documents. If Φ (d_j, c_i) = T, then d_j is called a positive example (or a member) of c_i, while if Φ (d_j, c_i) = F it is called a negative example of c_i. The categories are just symbolic labels that are no additional knowledge of their meaning is usually available, and it is often the case that no metadata (such as e.g. publication date, document type, publication source) is available either. In these cases, classification must be accomplished only on the basis of knowledge extracted from the documents themselves.

TC is a subjective task, when two experts (human or artificial) decide whether or not to classify document d_j under category c_i, they may disagree, and this in fact happens with relatively high frequency. A news article on George W. Bush selling his shares in the Texas Bulls baseball team could be filed under Politics, or under Finance, or under Sport, or under any combination of the three, or even under neither, depending on the subjective judgment of the expert. The kind of learning that Machine Learning techniques engage in is usually called supervised learning, as it is supervised, or facilitated, by the knowledge of the preclassified data. Depending on the application, TC may be either a single-label task (i.e. exactly one $c_i \in C$ must be assigned to each $d_j \in D$), or a multi-label task (i.e. any number $0 \leq n_j \leq |C|$ of categories may be assigned to a document $d_j \in D$).

Text Classification can be roughly distinguished into three different phases, which are independent from each other (i.e. a solution to one problem not being influenced by the solutions given to the other two): document indexing, classifier learning, and classifier evaluation.

2.1.1. Document indexing

Document indexing denotes the activity of mapping a document dj into a compact representation of its content that can be directly interpreted (i) by a classifier building algorithm and (ii) by a classifier, once it has been built. The document indexing methods usually employed in TC are borrowed from IR, where a text d_j is typically represented as a vector of term weights d_j = w1j . . . w|T |j. Where T is the dictionary i.e. the set of terms (also known as features) and $0 \leq wk_j \leq 1$ quantifies the importance of t_k in characterizing the semantics of d_j.
An indexing method is characterized by (i) a definition of what a term is, and (ii) a method to compute term weights. Concerning (i), the most frequent choice is to identify terms either with the

words occurring in the document (with the exception of stop words which are eliminated in a pre-processing phase), or occurring with their stems (i.e. their morphological roots, obtained by applying a stemming algorithm). Concerning (ii), term weights may be binary-valued (i.e. $wkj \in \{0, 1\}$) or real-valued (i.e. $0 \leq wkj \leq 1$), depending on whether the classifier-building algorithm and the classifiers, once they have been built, require binary input or not. When weights are binary, these simply indicate presence/absence of the term in the document. When weights are non-binary, they are computed by either statistical or probabilistic techniques. One popular class of statistical term weighting functions is tf ∗ idf, where two intuitions are at play: (a) the more frequently t_k occurs in d_j, the more important for d_j it is (the term frequency intuition); (b) the more documents t_k occurs in, the less discriminating it is, i.e. the smaller its contribution is in characterizing the semantics of a document in which it occurs (the inverse document frequency intuition). Weights computed by tf ∗ idf techniques are often normalized so as to contrast the tendency of tf ∗ idf to emphasize long documents.

2.1.2. Classifier learning

A text classifier for c_i is automatically generated by a learning process which, by observing the characteristics of a set of documents pre-classified under c_i or c_i, decides the characteristics that a new unseen document should have in order to belong to ci. In order to build classifiers for C, one needs a set Ω of documents such that the value of Φ (d_j, c_i) is known for every $<d_j, c_i> \in \Omega \times C$. In text classification it is customary to partition Ω into two disjoint sets Tr (the training set) and Te (the test set). The training set is the set of documents observing which the learner builds the classifier. The test set is the set on which the effectiveness of the classifier is finally evaluated. In the test phase, “evaluating the effectiveness” means running the classifier on a set of pre-classified documents and checking the degree of correspondence between the output of the classifier and the preassigned classes.

2.1.3. Classifier Evaluation

Training efficiency (average time required to build a classifier from a given corpus Ω), classification efficiency (average time required to classify a document) and effectiveness (average correctness of testing documents in the testing phase) are all legitimate measures of success for a learner. In Text Classification, effectiveness is usually considered the most important criterion, since it is the most reliable one when it comes to comparing different TC methodologies on the other hand efficiency depends on too volatile parameters like software/hardware platforms. In TC application all three parameters are important and one must carefully look for a tradeoff among them. In single-label TC, effectiveness is usually measured by accuracy that is the percentage of correct classification decisions. Here error is the converse of accuracy which is E = 1 − A). However in multi-label TC, accuracy is not an adequate measure. The reason for this is that in binary (multi-label) TC applications the two categories c_i and c_i are usually unbalanced that is one contains far more members than the other. In this case, building a classifier that has high accuracy is difficult that is the classifier that assigns all documents to the most heavily populated category and there are no applications in which one is interested in such a classifier5. As a result, in binary TC it is often the case that effectiveness with respect to category c_i is measured by a combination of precision wrt c_i (π_i), the percentage of documents that belong to c_i that in fact belong to it, and recall wrt c_i (ρi), the percentage of documents belonging to c_i that are in fact to belong to it. The more detailed description about classification evaluation is given in performance measures in the later part of the project.

3. Document Preprocessing and representations

In order to classify text documents by applying machine learning techniques, documents should first be preprocessed. In the preprocessing step, the documents should be transformed into a representation suitable for applying the learning algorithms. The most widely used method for document representation is the vector space model. In this model, each document is represented as a vector **d**. Each dimension in the vector **d** stands for a distinct term in the term space of the document collection. A term in the document collection can stand for a distinct single-word, a stemmed word or a phrase. Phrases consist of multiple words such as “data mining” or “mobile phone” and constitute a different context than when used separately. Phrases can be extracted by using statistical or Natural Language Processing (NLP) techniques. Phrases can also be extracted by manually defining the phrases for a particular domain such as done to filter spam mail in. However, this does not fulfill our requirement to organize documents in generic domains such as the Web. In vector space presentation, defining terms as distinct single words is referred to as “bag of words” representation. As “bag of words” representation is the most frequently used method for defining terms and it is computationally more efficient than the phrase representation, we have chosen to adapt this method to define terms of the feature space. One challenge emerging when terms are defined as single words is that the feature space becomes very high dimensional. In addition, words which are in the same context such as biology and biologist are defined as different terms. So, in order to define words that are in the same context with the same term and consequently to reduce dimensionality we have decided to define the terms as stemmed words. To stem the words, we have chosen to use Porter's Stemming Algorithm, which is the most commonly used algorithm for word stemming in English. Preprocessing and document representation phase, which is implemented in Python 2.7 IDLE shell, consists of the following steps:

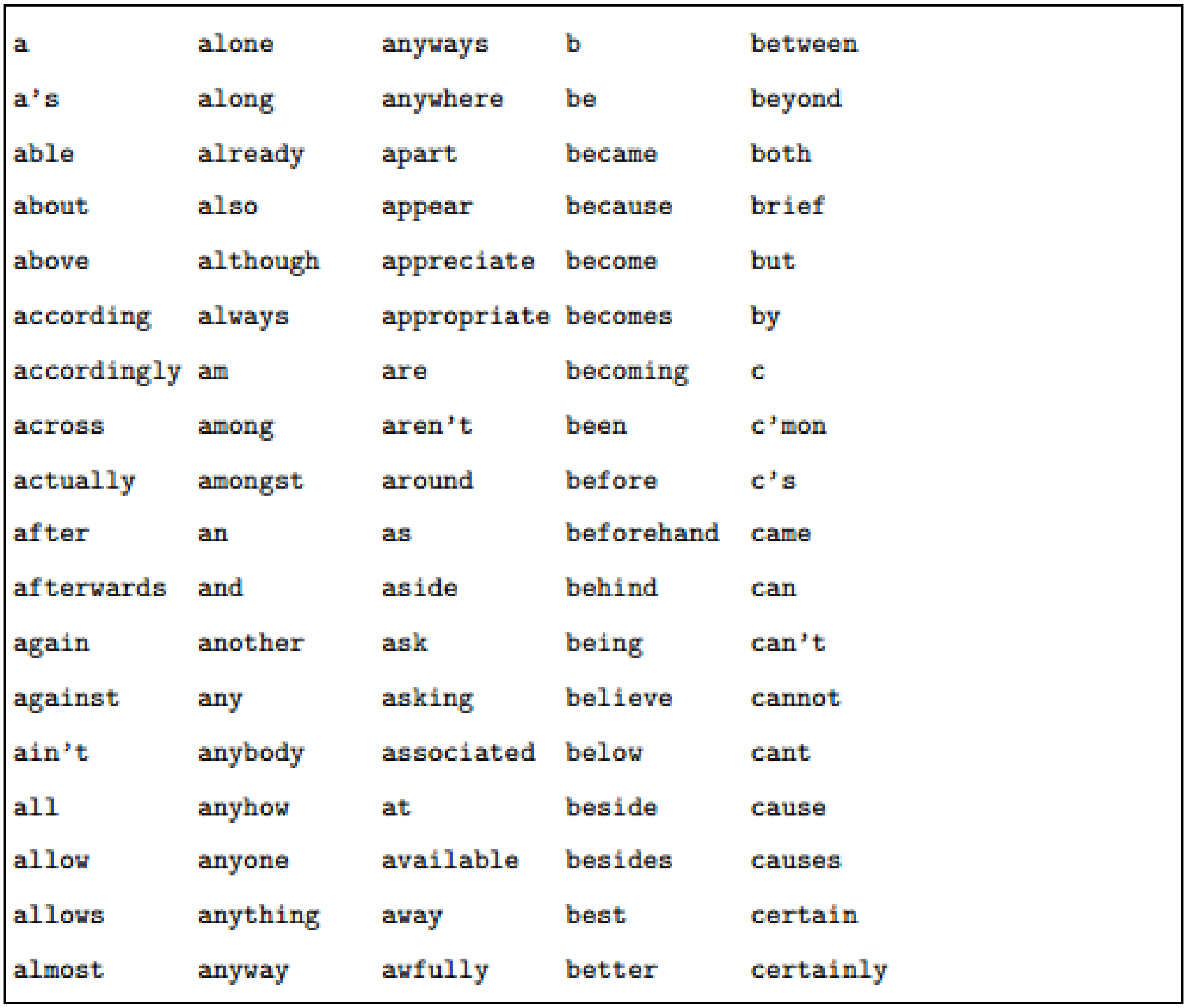
- Parsing the documents and case-folding
- Removing stopwords
- Stemming
- Term weighting

3.1. Parsing the Documents and Case-folding

In this step, all the HTML or SGML mark-up tags and non-alpha characters are removed from the documents in the document corpora. Parsing the documents also involves cleaning the documents with non-alpha characters, which do not contain any information and thus can be easily removed. Case-folding which stands for converting all the characters in a document into the same case, is performed by converting all the characters into lower-case. Tokens consisting of alpha characters are extracted.

3.2. Removing Stopwords

There are words in English such as pronouns, prepositions and conjunctions that are used to provide structure in the language rather than content. These words, which are encountered very frequently and carry no useful information about the content and thus the category of documents, are called stopwords. Removing stopwords from the documents is very common in information retrieval. We have decided to eliminate the stopwords from the documents, which will lead to a drastic reduction in the dimensionality of the feature space. Figure 1 shows a portion of the stopword list.

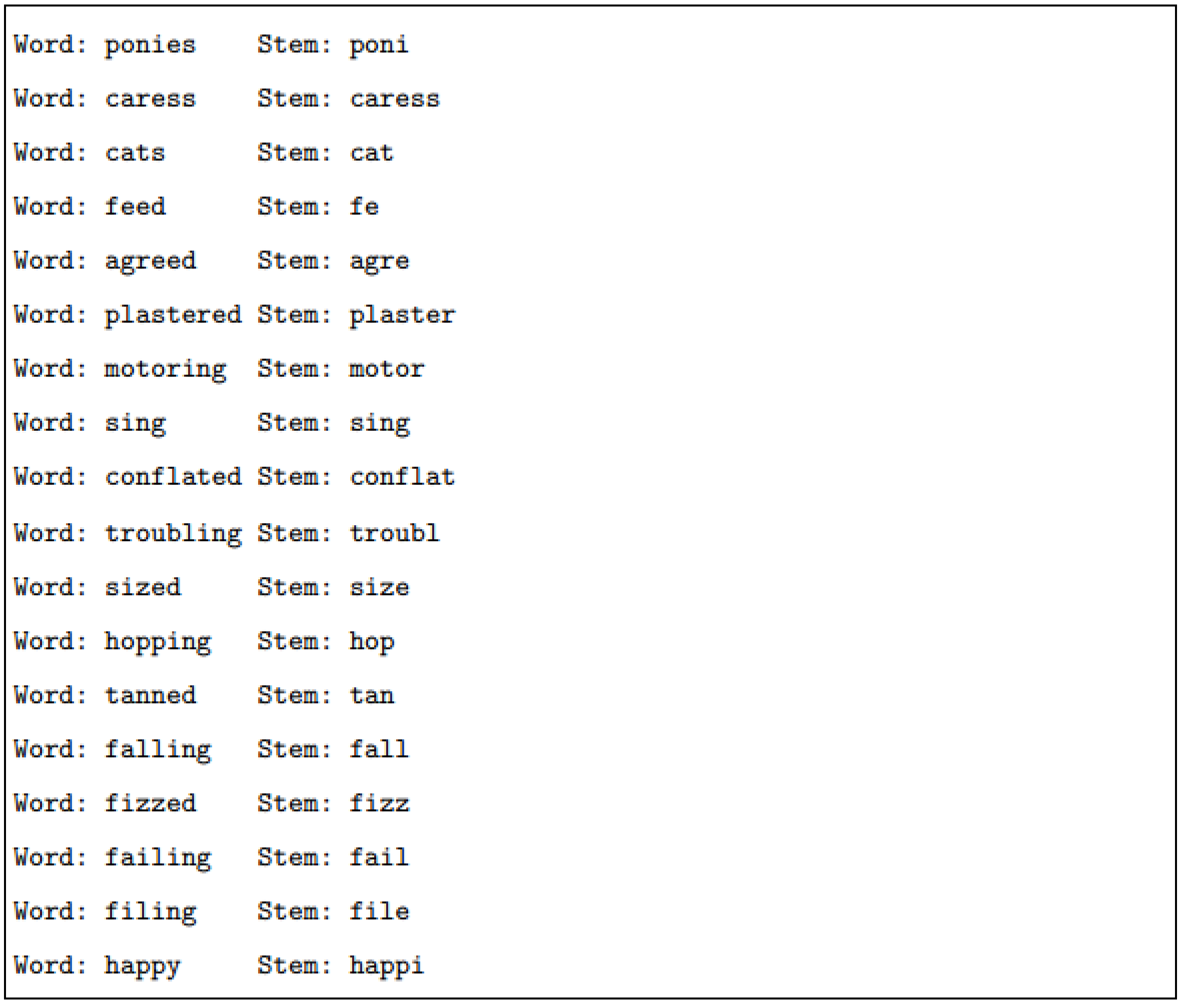


a	alone	anyways	b	between
a's	along	anywhere	be	beyond
able	already	apart	became	both
about	also	appear	because	brief
above	although	appreciate	become	but
according	always	appropriate	becomes	by
accordingly	am	are	becoming	c
across	among	aren't	been	c'mon
actually	amongst	around	before	c's
after	an	as	beforehand	came
afterwards	and	aside	behind	can
again	another	ask	being	can't
against	any	asking	believe	cannot
ain't	anybody	associated	below	cant
all	anyhow	at	beside	cause
allow	anyone	available	besides	causes
allows	anything	away	best	certain
almost	anyway	awfully	better	certainly

Figure 1 Stopwords list (English)

3.3. Stemming

In order to define words that are in the same context with the same term and consequently to reduce dimensionality, we have decided to define the terms as stemmed words. To stem the words, we have chosen to use Porter's Stemming Algorithm, which is the most commonly used algorithm for word stemming in English. In this way for instance, we reduce the similar terms "computer", "computers", and "computing" to the word stem "comput". For stemming the words of the dictionary we have used the inbuilt porter stemmer. This stemmer is embedded in the preprocessing



```
Word: ponies    Stem: poni
Word: caress    Stem: caress
Word: cats      Stem: cat
Word: feed      Stem: fe
Word: agreed    Stem: agre
Word: plastered Stem: plaster
Word: motoring  Stem: motor
Word: sing      Stem: sing
Word: conflated Stem: conflat
Word: troubling Stem: troubl
Word: sized     Stem: size
Word: hopping   Stem: hop
Word: tanned    Stem: tan
Word: falling   Stem: fall
Word: fizzed    Stem: fizz
Word: failing   Stem: fail
Word: filing    Stem: file
Word: happy     Stem: happi
```

Figure 2 Sample of words and their corresponding stems found by Porter's

system. Figure 3 displays a sample of words and the stems produced by Porter's Stemming Algorithm. After stemming, terms that are shorter than three characters are also removed as they do not carry much information about the content of a document.

3.4. Term Weighting

In IR documents are usually given as vector representation in which the dimensions (features) of the vector representing a document are the terms occurring in the document. The approach to term representation that the IR community has almost universally adopted is known as the bag-of-words approach: a document d_j is represented as a vector of term weights $d_j = <\omega_{1j}, ..., \omega r_{ji}>$ where r is the size of the dictionary and $0 \leq \omega_{kj} \leq 1$ represents the contribution of term t_k to the specification of the semantics of d_j.

3.4.1. Why term Weighting

In the vector space model, we represent documents as vectors. The success or failure of the vector space method is based on term weighting. Term weighting is an important aspect of modern text retrieval systems. Terms are words, phrases, or any other indexing units used to identify the contents of a text. Since different terms have different importance in a text, an important indicator - he term weight is associated with every term. A term weighting scheme plays an important role for the similarity measure. There are three components in a weighting scheme:

$$a_{ij} = g_i * t_{ij} * d_j$$

Where gi is the global weight of the ith term, t_{ij} is the local weight of the ith term in the jth document, d_j is the normalization factor for the jth document. Usually the three main components that affect the importance of a term in a text are the term frequency factor (tf), the inverse document frequency factor (idf), and document length normalization.

3.4.2. Local Term Weighting

These formula depend only on the frequencies within the document and they do not depend on the inter document frequencies.

Description	**Formula**
Binary	$\begin{cases} 1, & \text{if } t, d > 0 \\ 0, & \text{otherwise} \end{cases}$
Natural	$\mathrm{tf}_{t,d}$
Logarithmic	$1 + \log(\mathrm{tf}_{t,d})$
Augmented	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$

Table 1: term frequency formulae

3.4.2.1 Binary

Binary formula gives every word that appears in a document equal relevance. This can be useful when the number of times a word appears is not considered important.

$$\aleph(\mathrm{t}) = \begin{cases} 1, & \text{if t, d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Terms are either present or absent, so the use of binary weights is often too limiting, because it doesn't provide consideration for partial matches.

3.4.2.2 Term frequency

This formula counts how many times the term occurs in a document. The more times a term t occurs in document d the more likely it is that t is relevant to the document. If this method is used alone it favors common words and long documents. This formula gives more credit to words that appears more frequently, but often too much credit. For instance, a word that appears ten times in a document is not usually ten times more important than a word that only appears one. Binary and term frequency weights are typically used for query weighting, where terms appear only once or twice. For document weighting, these weights are generally not best because binary does not differentiate between terms that appear frequently and terms that appear only once and because term frequency gives too much weight to terms that appear frequently. The logarithm formulas thus give a better solution.

3.4.2.4 Logarithmic term frequency

Logarithms are a way to de-emphasize the effect of frequency. Logarithms are used to adjust within document frequency because a term that appears ten times in a document is not necessarily ten times as important as a term that appears once in that document. Logarithms formulas decrease the effects of large differences in term frequencies.

3.4.2.3 Augmented normalized term frequency

This formula try to give credit to any word that appears and then give some additional credit to words that appear frequently. The formula gives a value of K = 0.5 for appearing in the document plus a bonus (no more than 0.5) that depends on the frequency. Also K must be set to something low (0.3) for large documents and to higher values (0.5) for shorter documents. With this formula, the output value varies only between 0.5 and 1 for terms that appear in the document. By restricting the tf factors to a maximum value of 1.0, this technique compensates the problem of the presence of higher term frequencies for normalization. So this technique turns out to be a weak form of normalization and favors the retrieval of long documents if used alone without another one normalization formula.

3.4.3 Global Term Weighting

These formulas are used to place emphasis on terms that are discriminating and they are based on the dispersion of a particular term throughout the documents. Global weighting tries to give a discrimination value to each term. Many schemes are based on the idea that the less frequently a term appears in the whole collection, the more discriminating it is. Global weighting is in general very successful. The use of global weighting can eliminate the need for stop word removal since stop words should have very small global weights. In practice, however, it is easier to remove the stop words in the preprocessing phase so that there are fewer terms to handle.

Description	**Formula**
Natural	1
Inverse Document Frequency	$\log\left(\frac{N}{d_f}\right)$
Probabilistic Inverse Document Frequency	$\log\left(\frac{N-d_f}{d_f}\right)$
Squared Inverse Document Frequency	$\log\left(\frac{N}{d_f}\right)^2$

Table 2: Inverse document frequency formulae

3.4.3.1 Natural

Sometimes is useful to consider only term frequency terms, when term frequencies are very small or when we are interested to emphasize the term frequencies in a document.

3.4.3.2 Inverse document frequency (IDF)

Inverse Document Frequency (IDF) is a popular measure of a word's importance. It's defined as the logarithm of the ratio of number of documents in a collection to the number of documents containing the given word. This means rare words have high IDF and common words have low IDF. For example we obtain an output value eqaul to 0 if the given term appears in every document. The weight increases as the number of documents in wich the term appears decreases. High value indicates that the word occurs more often in this document than average. It's the most used global term weighting formula.

3.4.3.3 Probabilistic Inverse Document Frequency

Another IDF weight. It assigns weights ranging from $-\infty$ for a term that appears in every document to log $(n - 1)$ for a term that appears in only one document. It differs from IDF because probabilistic inverse actually awards negative weight for terms appearing in more than half of the documents in the collection, and the lowest weight gives, is one.

3.4.3.4 Squared Inverse Document Frequency:

It is similar to the inverse document frequency with a simple modification of the formula. Here the ratio (N/df) is squared first before the logarithm is applied to.

3.4.4 Normalization

The third component of the weighting scheme is the normalization factor, which is used to normalize the document lengths. It is useful to normalize the document vectors so that documents are retrieved independent of their lengths. It's important. If we do not, short documents may not be recognized as relevant. Automatic information retrieval systems have to deal with documents of varying lenghts in text collection. Document length normalization is used to fairly retrieve documents of all lenghts and it's used to remove the advantage that the long documents have in retrieval over the short documents. Two main reasons that necessitate the use of normalization in term weights are:

- **Higher term frequencies:** long documents usually use the same terms repeatedly. As a result, the term frequency factors may be large for long documents.
- **Number of terms:** long documents also have different numerous terms. This increases the number of matches between a query and a long document, increasing the chances of retrieval of long documents in preference over shorter documents.

Description	**Formula**
Squared Sum of Weight	$\sum_{k=1}^{n}(g_k \times t_{kj})^2$
Sum of Weight(Linear)	$\sum_{k=1}^{n}(g_k \times t_{kj})$
Cosine Normalization	$\sqrt{\sum_{k=1}^{n}(g_k \times t_{kj})^2}$

Table 3: Normalization formulae

3.4.4.1 Squared Sum of Weight Normalization

It is also another variant of the cosine normalization where gives more weight to the shorter documents as for each long documents the number of terms in the dictionary will be more and for each term the summation of the weight increase numerously resulting in the larger factor for each longer document then the shorter documents.

3.4.4.2 Sum of Weight Normalization

It is another variant of the cosine normalization where the normalization factor is not squared. This normalization factor gives more preferences to the shorter length documents. Since each term of the dictionary is divided by a more weight, each term has less weight giving more weight to the shorter documents then the longer documents.

3.4.4.3 Cosine Normalization

Cosine Normalization resolves both the reasons for normalization (Higher term frequencies, number of terms) in one step. Higher individual term frequencies increase individual wi values, increasing the penalty on the term weights. Also, if a document has more terms, the number of individual weights in the cosine factor increases, yielding a higher normalization factor. So longer documents have smaller individual term weights and smaller documents are favored over longer ones in retrieval. It’s the most used and popular normalization.

3.4.4.5 Normalization problems

- **Document Length Normalization Problems:** Long documents have an unfair advantage:
  • they use a lot of terms so they get more matches than short documents.
  • They use the same words repeatedly so they have much higher term frequencies.
- **Disadvantages:** One of the problems is solved that: now shorter documents receive a higher normalized score than longer documents with the same matching terms. On the other hand, we have got a new problem: now shorter documents are generally preferred over longer ones.

4. Supervised Technique for Document Classification

Supervised algorithms assume that the category structure or hierarchy of a text database is already known. They require a training set of labelled documents and return a function that maps documents to the pre-defined class labels. For supervised document classification we have used on the best technique commonly known as the Naïve Bayes method.

4.2 Naïve Bayes Model

The naive Bayes (NB) classifier is a probabilistic model that uses the joint probabilities of terms and categories to estimate the probabilities of categories given a test document. The naive part of the classifier comes from the simplifying assumption that all terms are conditionally independent of each other given a category. Because of this independence assumption, the parameters for each term can be learned separately and this simplifies and speeds the computation operations compared to non-naive Bayes classifiers. There are two common event models for NB text classification, multinomial model and multivariate Bernoulli model. We have primarily used the multinomial model of the NB classifier.

$$p(c_j/d_i) = \frac{p(c_j) \times p(d_i/c_j)}{p(d_i)}$$

Where **d_i** is a test document and **c_j** is a category. The posterior probability of each category **c_j** given the test document **d_i**, i.e. P (**c_j|d_i**), is calculated and the category with the highest probability is assigned to **d_i**. In order to calculate P (**c_j|d_i**), P (**c_j**) and P (**di|cj**) have to be estimated from the training set of documents. Note that P (**d_i**) is same for each category so we can eliminate it from the computation. The category prior probability, P (cj), can be estimated as follows:

$$p(c_j) = \frac{\sum_{i=1}^{n} y(d_i/c_j)}{N}$$

Where, N is number of training documents and y (**d_i**, **c_j**) is defined as follows:

$$p(d_i, c_j) = \begin{cases} 1, & \text{if } d_i \in c_j \\ 0, & \text{otherwise} \end{cases}$$

So, prior probability of category **c_j** is estimated by the fraction of documents in the training set belonging to **c_j**. P (**di|c_j**) parameters are estimated according to the multinomial model.

4.2.1 Multinomial Model

In the multinomial model a document **d_i** is an ordered sequence of term events, drawn from the term space T. The naive Bayes assumption is that the probability of each term event is independent of term's context, position in the document, and length of the document. So, each document **d_i** is drawn from a multinomial distribution of terms with number of independent trials equal to the length of **d_i**. The probability of a document **d_i** given its category **c_j** can be approximated as:

$$p(d_i/c_j) = \prod_{i=1}^{|d_i|} p(t_i/c_j)$$

Where |**d_i**| is the number of terms in document **d_i** and **t_i** is the i^th term occurring in document **d_i**. Thus the estimation of P (**d_i**|**c_j**) is reduced to estimating each P (**t_i**|**c_j**) independently. The following Bayesian estimate is used for P (**t_i**|**c_j**):

$$p(t_i/c_j) = \frac{1+\mathrm{TF}(t_i,c_j)}{\sum_{t' \in V}\{\mathrm{TF}(t_i,c_j)+1\}}$$

That is

$$p(t_i/c_j) = \frac{1+\mathrm{TF}(t_i,c_j)}{|T|+\sum_{tk \in T}\mathrm{TF}(t_i,c_j)}$$

Here TF (**t_i**, **c_j**) is the total number of times term ti occurs in the training set documents belonging to category **c_j**. And |**T**| is the total number of terms in the dictionary.

5. Algorithm of Multinomial Naïve Bayes Model

Naïve Bayes is one of the supervised text classification technique where the classifier is made to learn with the help of the training documents and then the test documents are classified based on the classifier. The complete algorithm of the Multinomial Naïve Bayes Model is given as follows:

5. Algorithm

```
TRAINMULTINOMIALNB(C, D)
1   $V \leftarrow$ EXTRACTVOCABULARY(D)
2   $N \leftarrow$ COUNTDOCS(D)
3   for each $c \in C$
4   do $Nc \leftarrow$ COUNTDOCSINCLASS(D, $c$)
5           $prior[c] \leftarrow Nc/N$
6           $textc \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS(D, $c$)
7           for each $t \in V$
8           do $Tct \leftarrow$ COUNTTOKENSOFTERM($textc$, $t$)
9           for each $t \in V$
10          do $condprob[t][c] \leftarrow \frac{1+\mathrm{TF}(t_i,c_j)}{|T|+\sum_{tk \in T}\mathrm{TF}(t_i,c_j)}$
11   return $V$, $prior$, $condprob$
```

After the preprocessing step that is after the cleaning, case-folding, stemming and lemmatizing of the documents, the classifier is trained with the help of the training data. By training it means the code is executed for each training dataset and the results obtained during this process will be used to test the documents in the testing phase.

In the training algorithm above, first the dictionary is created. Dictionary is the bag of words model where for each training document the words of it are added to the dictionary leaving behind the duplicates. Now for each category, a text is created which is the concatenation of all documents belonging to that category. This is done to find the tem frequency of each word in the vocabulary. As soon as the text for each class is created, a vector called prior is created which contains the relative value of the number of documents present in that category/class. And it equal to the number of documents in the class and total number of documents in the whole dataset. Now for each word of the dictionary, the tem frequency of that word in the text created before is calculated. And for each word a relative value that is term frequency of that word divided by sum of term frequency of all the words of the dictionary is stored in the 2-d vector condprob. This is basically the score of each term and in each class. Now in our project various term weighting schemes are also applied, which basically multiplies this ratio with inverse document frequency and then normalizing the ratio to negate the effect of longer documents. Now after the training phase the vectors prior and condprob will be used in the testing phase to test the documents.

```
APPLYMULTINOMIALNB(C, V, prior, condprob, d)

1   W ← EXTRACTTOKENSFROMDOC(V, d)
2   for each c ∈ C
3   do score[c] ← log prior[c]
4       for each t ∈ W
5       do score[c] += log condprob[t][c]
6   return arg maxc ∈ C score[c]
```

After the training of the documents, testing is done. That is for each documents the classifier is executed and the performance measures like precision, recall & f-measure values are calculated.

In the testing phase, for each document a collection of token called W is created which simply stores all the token of that testing document. Now for each class c_i and each term of W, a vector called score is maintained. Score simply updates the value from the score vector for each term and for each class. Now a class with maximum score will be the desired class of the testing document. This task is done for each testing document and the performance measure is gathered.

5.1 Flow Chart

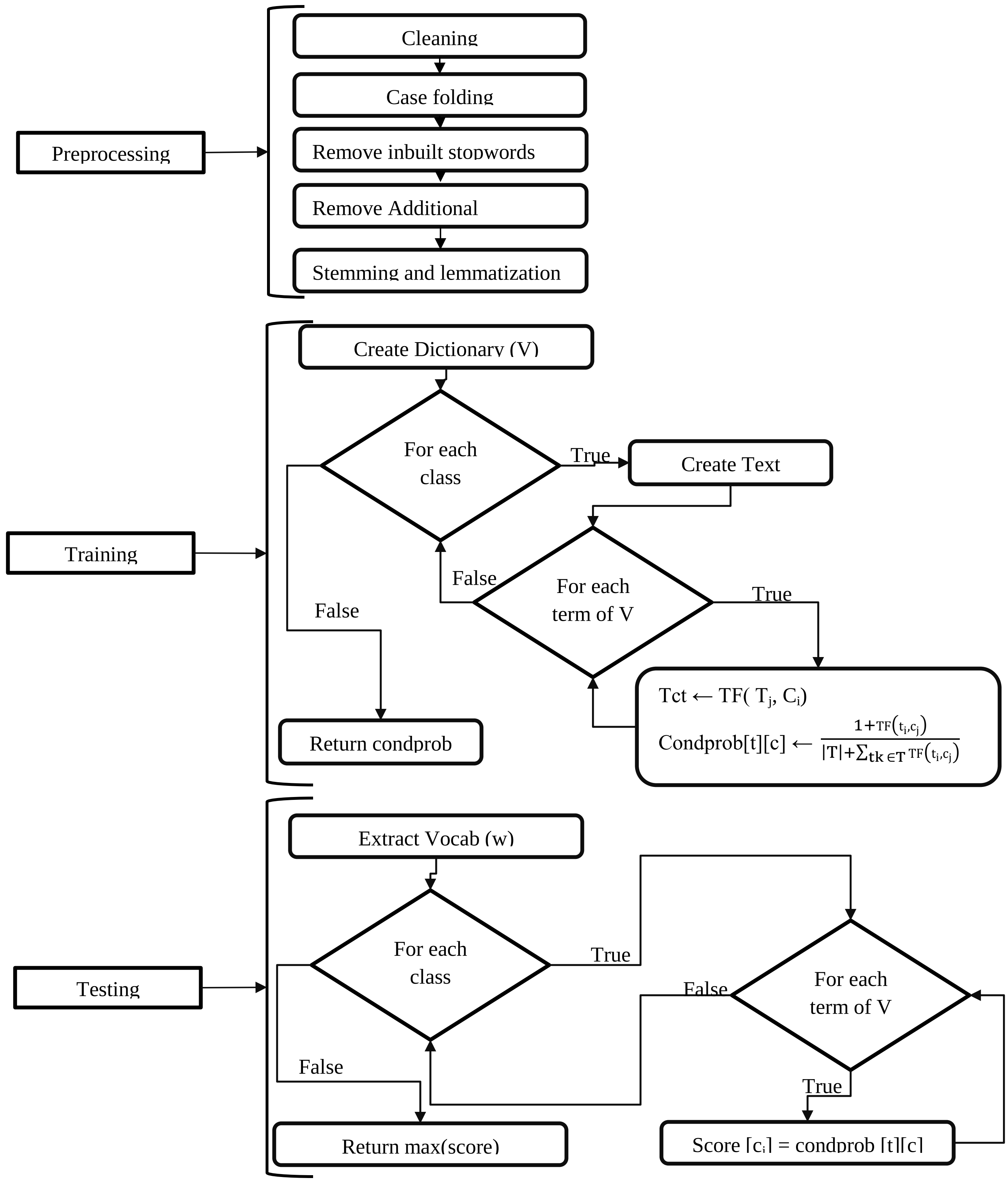


Figure 3 Flow Chart of the Algorithm

6 Software and Hardware Requirement

This project is developed using the following framework tools

Hardware Requirement

- PROCESSOR

  CPU – INTEL DUALCORE AND HIGHER

- HARD DISK – 20GB SPACE REQUIRED MINIMUM
- RAM – 512MB MINIMUM

Software Requirement

- Python 2.7 IDLE Shell
- OPERATING SYSTEM : WINDOWS XP/VISTA/7/8/8.1/10, LINUX
- IDE Python 2.7
- Supporting Python Modules

I. nltk.corpus

II. nltk.tokenize

III. nltk.wordnet

IV. nltk.PorterStemmer()

V. nltk.WordNetLemmatizer()

VI. import Counter

VII. import math

VIII. import os

IX. import nltk

7 Source Code

The source code of the entire project is divided into 3 sections namely – the preprocessing section, the training section and the testing section.

7.1 Preprocessing

Preprocessing consists of codes which performs the preprocessing of the text documents. Preprocessing involves cleaning the data, case folding of the words, removing the stopwords etc.

7.1.1 loop over files.py

This code iterates through all classes and through each document to clean the data. By cleaning the data we mean, removing any non-alpha characters like -,!,=,<,>,[,],?,___ and many others.

```
import os
folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\Training')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\Training'

for dir in folder_list:
    folder_path=os.path.join(root,dir)
    for root1,dirs,filenames in os.walk(folder_path):
        for f in filenames:
            log=open(os.path.join(root1,f),'r+')
            data=""
            lines=log.readlines()
            for i in lines:
                str=i
                str=str.replace('>',' ')
                str=str.replace('<',' ')
                str=str.replace('\n',' ')
                str=str.replace(':',' ')
                str=str.replace('-',' ')
                str=str.replace('^',' ')
                str=str.replace('/',' ')
                str=str.replace('_',' ')
                str=str.replace('+',' ')
                str=str.replace('=',' ')
                str=str.replace('|',' ')
                #str=str.replace('\\',' ')
                str=str.replace('\\',' ')
                data=data+str
            log.seek(0)
            log.write(data)
            log.close()
```

7.1.2 lowercase.py

This code iterates through each classes and each document to convert each word(token) into small case. This is mainly done to bring uniformity in the documents which help in applying further preprocessing stems like removal of stopwords, stemming etc.

```
import os
indir=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\windows'
for root,dirs,filenames in os.walk(indir):
    for f in filenames:
        log=open(os.path.join(root,f),'r+')
        data=""
        lines=log.readlines()
        for i in lines:
            str=i.strip()
            for j in str:
                data=data+j.lower()
        log.seek(0)
        log.write(data)
        log.close()
```

7.1.3 remove_InbuiltStopwords.py

This code removes the inbuilt stopwords of the English language like don’t, cant, do, a, an, the etc. This stop words are available in the nltk stopwords list. And can be easily removed by simply importing the module and removing from the documents.

```
from nltk.corpus import stopwords
import os
stops=stopwords.words('english')
stops=set(stops)
#indir=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\windows'
indir=r'C:\Users\Anand Namdev\Downloads\mini_newsgroup_clean\windows'
for root,dirs,filenames in os.walk(indir):
    for f in filenames:
        log=open(os.path.join(root,f),'r+')
        data=""
        lines=log.readlines()
        for i in lines:
            str=i.split()
        for j in str:
            if j not in stops:
                data=data+j+" "
        log.seek(0)
        log.write(data)
        log.truncate()
        log.close()
```

7.1.4 remove_additionalStopwords.py

Not all stopwords can be removed easily by executing the above code. Some stop words can only be removed after properly studying the dataset and observing words if it occurs too frequently or if its information coefficient is almost zero. This list of additional stopwords varies from each dataset. This code finally also removes any words which is of length less than 3. This is done because words of length less than 3 usually doesn’t contain any information and removal of which greatly removes the vector space.

```
# Removes from additional stopwords from all files over a directory
# Removes all those tokens which are of length less then 3
from nltk.tokenize import wordpunct_tokenize
from nltk.corpus import stopwords
import os
stops=stopwords.words('english')
```

```
stops=set(stops)
new_list=['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{',
'}','*','^','`','0','1','2','3','4','5','6','7','8','9','$','%','&','#','subject','+','|',chr(244),chr(253),chr(228)]
stops.update(new_list)

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\Training')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\Training'

for dir in folder_list:
    folder_path=os.path.join(root,dir)
    for root1,dirs,filenames in os.walk(folder_path):
        for f in filenames:
            log=open(os.path.join(root1,f),'r+')
            data=""
            lines=log.readlines()
            str="".join(lines)
            doc=wordpunct_tokenize(str)
            for j in doc:
                if j not in stops:
                    if len(j)>=3:
                        flag=0
                        for k in new_list:
                            if k in j:
                                flag=1
                                break
                        if flag==0:
                            data=data+j+" "
            log.seek(0)
            log.write(data)
            log.truncate()
            log.close()
```

7.1.5 stemLemma.py

This code stems and lemmatizes each word of each document. This is done to convert each word in its morphological root and remove duplicate words.

```
import nltk
def stemLemma(v): #this v is a type list
    wnl=nltk.WordNetLemmatizer()
    porter=nltk.PorterStemmer()
    ls=[]
    for i in xrange(0,len(v),1):
            word1=wnl.lemmatize(v[i])
            word2=porter.stem(word1)
            ls.append(word2)
    ls=list(ls)
    return ls  #return list
```

7.2 Training

It consists of code which trains the classifier with the help of training documents.

7.2.1 vocab.py

This code creates the dictionary of tokens by iterating over all documents and adding each word into the dictionary.

```
import os
from nltk.tokenize import wordpunct_tokenize

def create_vocab(folder_list,root):
    data=""
    for dir in folder_list:
        folder_path=os.path.join(root,dir)
        for root1,dirs,filenames in os.walk(folder_path):
            for f in filenames:
                log=open(os.path.join(root1,f),'r')
                lines=log.readlines()
                str="".join(lines)
                data=data+str

    doc=wordpunct_tokenize(data)
    return list(doc)
```

7.2.2 countAllDocs.py

This code simply counts the total number of documents spanned over all classes

```
import os
def countAllDocs(folder_list,root):    # counts all docs in all classes
    sum=0
    for dir in folder_list:
        folder_path=os.path.join(root,dir)
        for root1,dirs,filenames in os.walk(folder_path):
            sum=sum+len(filenames)
    return sum.
```

7.2.3 text.py

This code concatenates the text of all documents for a particular class $c_{i.}$ This is mainly done to create a text for each class c_i and then calculate the term frequency TF(t,d) for each word of the dictionary.

```
import os
def concatenateAllDocsofC(folder_path):
    data=""
    for root1,dirs,filenames in os.walk(folder_path):
        for f in filenames:
            log=open(os.path.join(root1,f),'r')
            lines=log.readlines()
            str="".join(lines)
            data=data+str
        return data,len(filenames)
```

7.2.4 main3.py (2-1-2)

In our project we have used 3 different term weighting techniques, in which for local term weighting, global term weighting and normalization we have used a different formula and tried to analyze the result.

Main3.py is the first attempt to write the training function in which following formulae has been used for term frequency(TF) , inverse document frequency(IDF) and normalization.

TF – (2)	IDF – (1)	Normalization – (2)
$tf_{t,d}$	1	$\sum_{k=1}^{n}(g_k \times t_{kj})$

```
from __future__ import division
import os
from nltk import *
from math import *
from nltk.tokenize import wordpunct_tokenize
from stemLemma import *  # Lemmatizing and stemming the vocabulary
from vocab import *  # creates vocabulary
from countAllDocs import * # counts all documents
from text import * #concatenate All Docs of Class C
from collections import Counter

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training2')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training2'

vocab=create_vocab(folder_list,root)
vocab=list(vocab)
n=countAllDocs(folder_list,root)
vocabStemmed=stemLemma(vocab)

words=Counter(vocabStemmed)
keyValuePair=words.most_common() # list of tuple(key,frequency) in reverse prder
mostFreqK=[]  #list of most frequent k-words with k=5000,10000,len(words)
classDict={} # it is a dictionary where each element is a (term,list)
prior=[] #prior probability of each class: /n
for i in xrange(0,len(words),1):
   mostFreqK.append(keyValuePair[i][0])
   classDict[keyValuePair[i][0]]=[]

l=0
for dir in folder_list: # that is for each class
   folder_path=os.path.join(root,dir)
   text_c,nc=concatenateAllDocsofC(folder_path)  # type(text_c)=string
   doc=wordpunct_tokenize(text_c) #converts the string into tokens
   text_c_stemmed=stemLemma(doc)  #type(text_c_stemmed)=list
```

```
        #print dir," length=" , len(set(text_c_stemmed))
        text_c=" ".join(text_c_stemmed)  # list---->string conversion

        prior.append(nc/n)
        words_c=text_c.split()
        wordCount=Counter(words_c).most_common()
        mydict=dict(wordCount)

        biggerList=[]
        biggerDict={}
        countAllFreq=0

        for word in mostFreqK:
            value=0
            if mydict.has_key(word):
                value=mydict.get(word)
            countAllFreq=countAllFreq+pow(value,1)
            tuple_variable=(word,value)
            biggerList.append(tuple_variable)

        for word in mostFreqK:
            value=0
            if mydict.has_key(word):
                value=mydict.get(word)
            ratio=(value+1)/(countAllFreq+len(mostFreqK))  #smoothing is done here

            classDict[word].append(ratio)

        biggerDict=dict(biggerList)
        l=l+len(text_c)
        print dir,len(text_c),len(words_c)
    print l
    def UseData():
        return classDict,prior
```

7.2.5 main2.py (3-1-1)

Here following formulae of TF, IDF, Normalization is used.

TF – (3)	IDF – (1)	Normalization – (1)
$1 + \log(tf_{t,d})$	1	$\sum_{k=1}^{n}(g_k \times t_{kj})^2$

```
from __future__ import division
import os
from nltk import *
from math import *
from nltk.tokenize import wordpunct_tokenize
from stemLemma import *  # Lemmatizing and stemming the vocabulary
from vocab import *  # creates vocabulary
from countAllDocs import * # counts all documents
from text import * #concatenate All Docs of Class C
from collections import Counter

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training2')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training2'

vocab=create_vocab(folder_list,root)
vocab=list(vocab)
n=countAllDocs(folder_list,root)
# stemming and lemmatizing each word of the vocabulary
vocabStemmed=stemLemma(vocab)

words=Counter(vocabStemmed)
keyValuePair=words.most_common() # list of tuple(key,frequency) in reverse prder
mostFreqK=[]  #list of most frequent k-words with k=5000,10000,len(words)
classDict={} # it is a dictionary where each element is a (term,list)
prior=[] #prior probability of each class: /n
for i in xrange(0,len(words),1):
    mostFreqK.append(keyValuePair[i][0])
    classDict[keyValuePair[i][0]]=[]

l=0
for dir in folder_list: # that is for each class
    folder_path=os.path.join(root,dir)
    text_c,nc=concatenateAllDocsofC(folder_path)  # type(text_c)=string
    doc=wordpunct_tokenize(text_c) #converts the string into tokens
    text_c_stemmed=stemLemma(doc)  #type(text_c_stemmed)=list
    text_c=" ".join(text_c_stemmed)  # list---->string conversion

    prior.append(nc/n)
    words_c=text_c.split()
    wordCount=Counter(words_c).most_common()
    mydict=dict(wordCount)

    biggerList=[]
    biggerDict={}
    countAllFreq=0

    for word in mostFreqK:
        value=0
        if mydict.has_key(word):
            value=log(mydict.get(word),10)+1
        countAllFreq=countAllFreq+pow(value,2)
        tuple_variable=(word,value)
        biggerList.append(tuple_variable)

    for word in mostFreqK:
        value=0
        if mydict.has_key(word):
            value=log(mydict.get(word),10)+1
        ratio=(value+1)/(countAllFreq+len(mostFreqK))  #smoothing is done here

        classDict[word].append(ratio)

    biggerDict=dict(biggerList)
    l=l+len(text_c)
    print dir,len(text_c),len(words_c)
print l

def UseData():
    return classDict,prior
```

7.2.6 feature2.py

This code is used as a input of main5 to find the inverse document frequency of each term. This code for each class calculates the number of documents which contain each term, which is commonly called as document frequency.

```
import os
from extractVocab import *
from stemLemma import *
from calculate import *

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\11111\Training')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\11111\Training'

def getKey(item):
    return item[1]

classDict={}
i=0
for dir in folder_list :
    folder_path=os.path.join(root,dir)
    for root1,dirs,filenames in os.walk(folder_path):
        for f in filenames :
            text_doc=extractVocab(root1,f)  # type(text_c)=list
            text_doc_stemmed=stemLemma(text_doc)  #type(text_doc_stemmed)=list
            text_doc_stemmed=set(text_doc_stemmed)
            text_doc_stemmed=list(text_doc_stemmed)

            for word in text_doc_stemmed:
                if len(classDict)==0 or classDict.has_key(word)==False :
                    classDict[word]=[0]*20
                classDict[word][i]=classDict[word][i]+1
                classDict[word][19]=classDict[word][19]+1
    i=i+1
    print i,len(classDict)
```

7.2.7 main5.py

This code trains the classifier with new term weighting scheme given as follows.

TF – (4)	IDF – (4)	Normalization – (3)
$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	$\log\left(\frac{N}{d_f}\right)^2$	$\sqrt{\sum_{k=1}^{n}(g_k \times t_{kj})^2}$

```
from __future__ import division
import os
from nltk import *
from math import *
from nltk.tokenize import wordpunct_tokenize
from collections import Counter
from vocab import *  # creates vocabulary
from countAllDocs import *
from stemLemma import *
from text import *
import feature2

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training10')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Training10'

vocab=create_vocab(folder_list,root)
n=countAllDocs(folder_list,root)
vocabStemmed=stemLemma(vocab)
words=Counter(vocabStemmed)

keyValuePair=words.most_common() # list of tuple(key,frequency) in reverse prder
mostFreqK=[]  #list of most frequent k-words with k=5000,10000,len(words)
classDict={} # it is a dictionary where each element is a (term,list)
prior=[] #prior probability of each class: /n
```

```
for i in xrange(0,len(words),1):
    mostFreqK.append(keyValuePair[i][0])
    classDict[keyValuePair[i][0]]=[]
featureDict=feature2.classDict

l=0
var=0
for dir in folder_list :
    folder_path=os.path.join(root,dir)
    text_c,nc=concatenateAllDocsofC(folder_path)  # type(text_c)=string
    doc=wordpunct_tokenize(text_c) #converts the string into tokens
    text_c_stemmed=stemLemma(doc)  #type(text_c_stemmed)=list
    text_c=" ".join(text_c_stemmed)  # list---->string conversion

    prior.append(nc/n)
    words_c=text_c.split()
    wordCount=Counter(words_c).most_common()
    mydict=dict(wordCount)

    biggerList=[]
    biggerDict={}
    countAllFreq=0
    mx=-1
    for word in mostFreqK:
        if mydict.has_key(word):
            tf_word=mydict.get(word)
            mx=max(mx,tf_word)

    for word in mostFreqK :
        tf_word=0
        df_word=1
        idf_word=1
        if mydict.has_key(word):
            tf_word=mydict.get(word)
        if featureDict.has_key(word):
            df_word=featureDict[word][var]
        if df_word!=0:
            idf_word=log(pow(n/df_word,2),10)
        value=(0.5+((0.5*tf_word)/mx))*idf_word
        countAllFreq=countAllFreq+pow(value,2)
        tuple_variable=(word,value)
        biggerList.append(tuple_variable)

    for word in mostFreqK:
        tf_word=0
        df_word=1
        idf_word=1
        if mydict.has_key(word):
            tf_word=mydict.get(word)
        if featureDict.has_key(word):
            df_word=featureDict[word][var]
        if df_word!=0:
            idf_word=log(pow(n/df_word,2),10)
        value=(0.5+((0.5*tf_word)/mx))*idf_word
        ratio=(value+1)/(sqrt(countAllFreq)+len(mostFreqK))
        classDict[word].append(ratio)

    biggerDict=dict(biggerList)
    l=l+len(text_c)
    print dir,len(text_c),len(words_c)
    var=var+1

print l


def UseData():
    return classDict,prior
```

7.3 Testing

Testing codes include testing of the documents on the results stored by training documents.

7.3.1 extractVocab.py

This code extracts the vocabulary of each testing document and preprocesses it.

```
import os
from nltk.tokenize import wordpunct_tokenize
def extractVocab(root,f):
    log=open(os.path.join(root,f),'r')
    lines=log.readlines()
    str="".join(lines)
    doc=wordpunct_tokenize(str)
    return list(set(doc))
```

7.3.2 testing2.py

This code tests the testing documents.

```
import os
from extractVocab import *
from main5 import *
#from main2 import *
#from main3 import *
from stemLemma import *
from answer import *

folder_list=os.listdir(r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Testing2')
root=r'C:\Users\Anand Namdev\Downloads\Compressed\datasets machine
learning\mini_newsgroup_clean\BackUp\10 fold\Testing2'
folder_path=root
classDict,prior=UseData()

def findmax(score):
    max=-1
    index=-1
    for i in xrange(0,19,1):
        if score[i]>max:
            max=score[i]
            index=i
    return index

classCount={}
checkAnswer={}
className=["autos","baseball","crypt","electronics","graphics","hockey","ibm_hardware","
mac_hardware","med","misc","miscforsale","motorcycles","politics_guns","politics_mideas
t","politics_misc","religion_christian","religion_misc","space","windows"]

for i in className:
    checkAnswer[i]=0
    classCount[i]=0

score=[None]*19 # to calculate the score of each class and then find max of all

for dir in folder_list:
    folder_path=os.path.join(root,dir)
    for root1,dirs,filenames in os.walk(folder_path):
        for f in filenames:
            w=extractVocab(root1,f)  # w is already list
            wStemmed=stemLemma(w)
            for i in xrange(0,19,1):
                score[i]=prior[i]
                for term in w:
                    value=0
                    if classDict.has_key(term):
                        value=classDict[term][i]
                    score[i]=score[i]+value
            number=findmax(score)



            if className[number]==answer[f]:
                checkAnswer[className[number]]=checkAnswer[className[number]]+1
            if classCount.has_key(className[number]):
                classCount[className[number]]=classCount[className[number]]+1
sum=0
for class_var in checkAnswer:
    sum=sum+checkAnswer[class_var]
print checkAnswer
print
print classCount
print sum
```

8 Experiment Results

8.1 Document Data Sets

In our project experiment we have used a standard document corpora widely used in automatic text classifications. The description of the standard document corpora is as follows:

Name of the corpora – Newsgroup data

Number of classes – 19

Number of documents – 1900

Autos	Baseball	Cryptography	Electronics
Graphics	Hockey	Ibm_Hardware	Mac_hardware
medicine	Miscellaneous	Misc. for sale	Motorcycles
Politics guns	Politics Mideast	Politics miscellaneous	Religion Christian
Religion miscellaneous	Space	Windows	

Table 4 Class Name of News Group dataset

Serial no.	**Class Name**	**No. of Documents**	**No. of terms**
1	Autos	100	2411
2	Baseball	100	2678
3	Cryptography	100	4177
4	Electronics	100	2464
5	Graphics	100	2259
6	Hockey	100	3026
7	Ibm_hardware	100	2265
8	Mac_hardware	100	2010
9	Medicine	100	2448
10	Miscellaneous	100	2019
11	Misc. for sale	100	2266
12	Motorcycles	100	3056
13	Politics guns	100	3472
14	Politics Mideast	100	4816
15	Politics miscellaneous	100	3925
16	Religion Christian	100	3351

17	Religion miscellaneous	100	3767
18	Space	100	5061
19	windows	100	4907

Table 5 Description of Classes

8.2 Performance Measure

In order to measure the performance of our project, we have used several performance measures.

8.2.1 Accuracy

Accuracy is used to describe the closeness of a measurement with respect to the true value. In simple words accuracy is the ratio of total number of correct predictions by the classifier and total number of predictions by the classifier. A prediction is said to be correct if it a true positive that is if the machine described a document to be in class c and the actual result was also the document to be in class c. A prediction is also said to be correct if it a false negative that is if the machine declared a document to be in not in class c and the actual results also stated that the document didn’t belong in class c. Hence in simple words accuracy is the ratio of sum of true positive and false negative with total number of predictions.

$$\text{accuracy}=\frac{\text{number of true positives} + \text{number oftrue negatives}}{\text{number of true positives+false positives+false negatives+true negatives}}$$

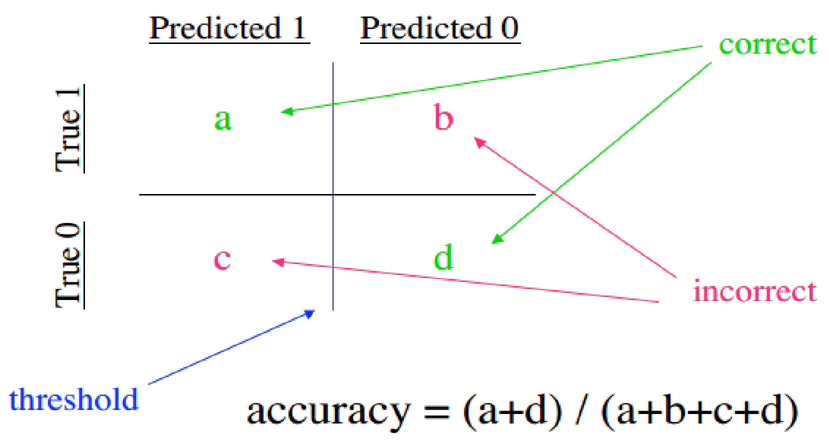


Figure 4 Accuracy Matrix

8.2.2 Precision

Precision is the ratio of true positive and sum of true positive and false positive. True positive is the number of documents which are correctly labelled as a class c. That is when a document is classified to be in class c and the document actually belongs to class c. And false positive is the number of documents which the machine declared to be not in class c but actually belonged to c. Hence precision measures the total number of correct predictions to a class with respect to the total number of prediction of class c.

$$\text{precision}=\frac{\text{number of true positives}}{\text{number of true positives+false positives}}$$

Or

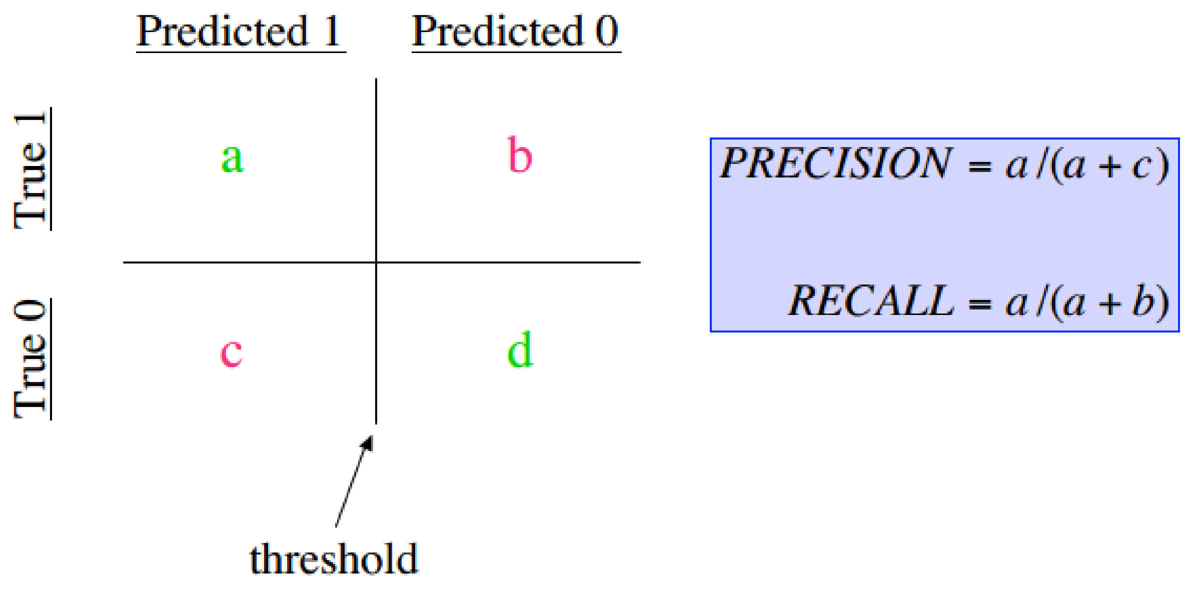


Figure 5 Precision & Recall matrix

$$\pi_i = \frac{TP_i}{TP_i + FP_i}$$

8.2.3 Recall

Recall is the ratio of true positive and sum of true positive and false negative. That is recall measures how many documents are relevant to the query. False negative is the total number of documents which the machine declared not to be in class c and were actually didn't belong to class c.

$$\text{recall} = \frac{\text{number of true positive}}{\text{number of true positive+false negative}}$$

8.2.4 F-measure

F-measure is the combined measure of the precision and recall. It is the harmonic mean of the precision and recall. And it is equal to

$$\text{F} - \text{measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision+recall}}$$

Or it can be also written as

$$\text{F} = \frac{2\pi\rho}{\pi + \rho}$$

The overall F1 measure score of the entire classification problem can be computed by two different types of averages, micro-average and macro-average.

8.2.4.1 Micro Averaged F-measure

In micro-averaging, F1-measure is computed globally over all category decisions. ρ and π are obtained by summing over all individual decisions.

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FP_i)}$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FN_i)}$$

Where M is the number of categories. Micro-averaged F1-measure is then computed as:

$$F(\text{micro} - \text{averaged}) = \frac{2\pi\rho}{\pi + \rho}$$

The micro-averaged F1-measure gives equal weight to each document. It is therefore considered as an average over all the document/category pairs.

8.2.4.2. Macro Averaged F-measure

In macro-averaging, *F*1-measure is computed locally over each category first and then the average over all categories is taken. π and ρ are computed for each category as given from above equations. Then *F*1-measure for each category *i* is computed as:

$$F_i = \frac{2\pi_i\rho_i}{\pi_i + \rho_i}$$

The macro-averaged *F*1-measure is obtained by taking the average of *F*1-measure values for each category.

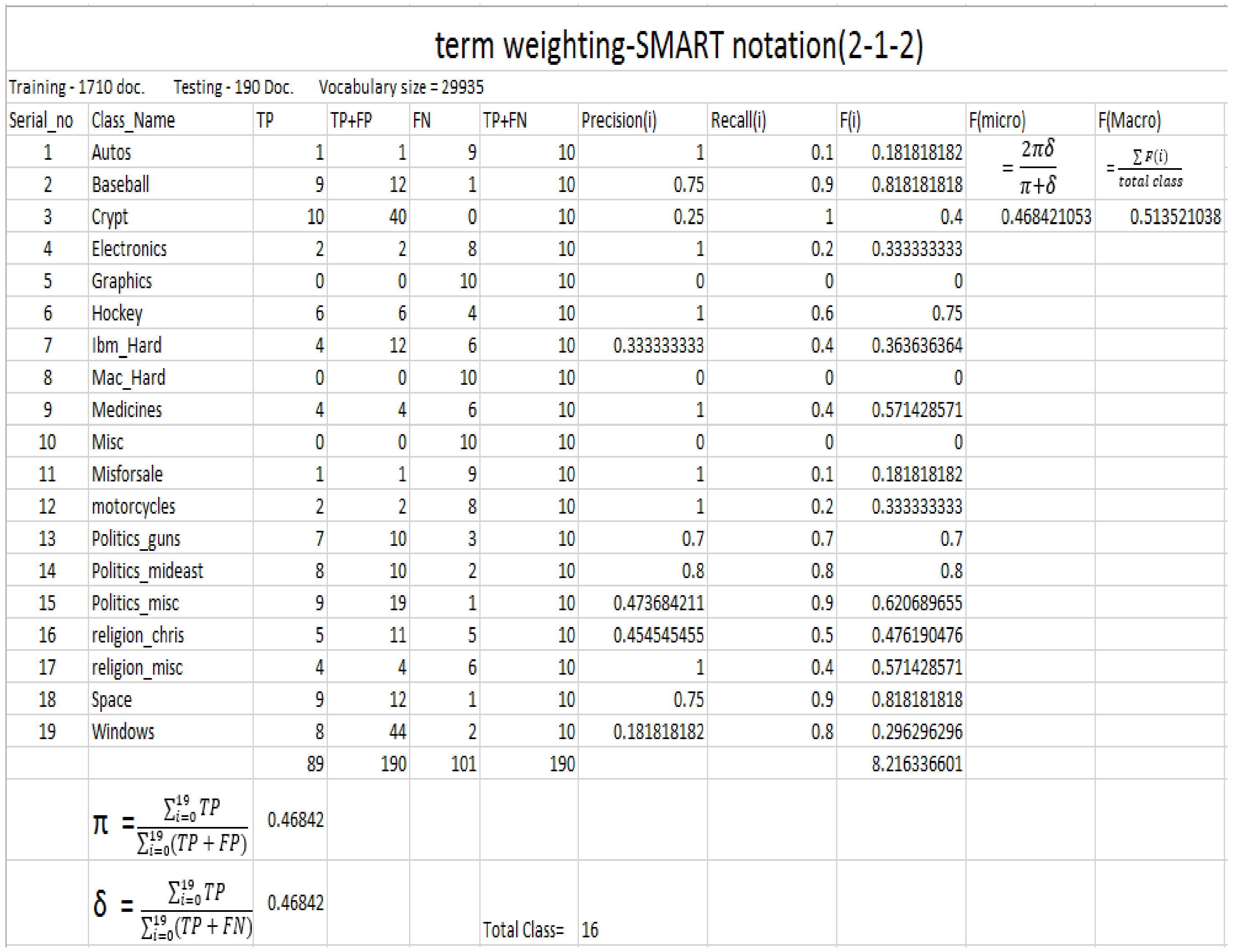
$$F(\text{macro} - \text{averaged}) = \frac{\sum_{i=1}^{m} F_i}{M}$$

Where *M* is total number of categories. Macro-averaged *F*1-measure gives equal weight to each category, regardless of its frequency.

	Microaveraging	**Macroaveraging**
Precision (π)	$\pi = \frac{\sum_{i=1}^{\lvert C \rvert} TP_i}{\sum_{i=1}^{\lvert C \rvert} TP_i + FP_i}$	$\pi = \frac{\sum_{i=1}^{\lvert C \rvert} \pi_i}{\lvert C \rvert} = \frac{\sum_{i=1}^{\lvert C \rvert} \frac{TP_i}{TP_i + FP_i}}{\lvert C \rvert}$
Recall (ρ)	$\rho = \frac{\sum_{i=1}^{\lvert C \rvert} TP_i}{\sum_{i=1}^{\lvert C \rvert} TP_i + FN_i}$	$\rho = \frac{\sum_{i=1}^{\lvert C \rvert} \rho_i}{\lvert C \rvert} = \frac{\sum_{i=1}^{\lvert C \rvert} \frac{TP_i}{TP_i + FN_i}}{\lvert C \rvert}$

Table 6 Micro & Macro F-Measure

8.3 Results and Observation

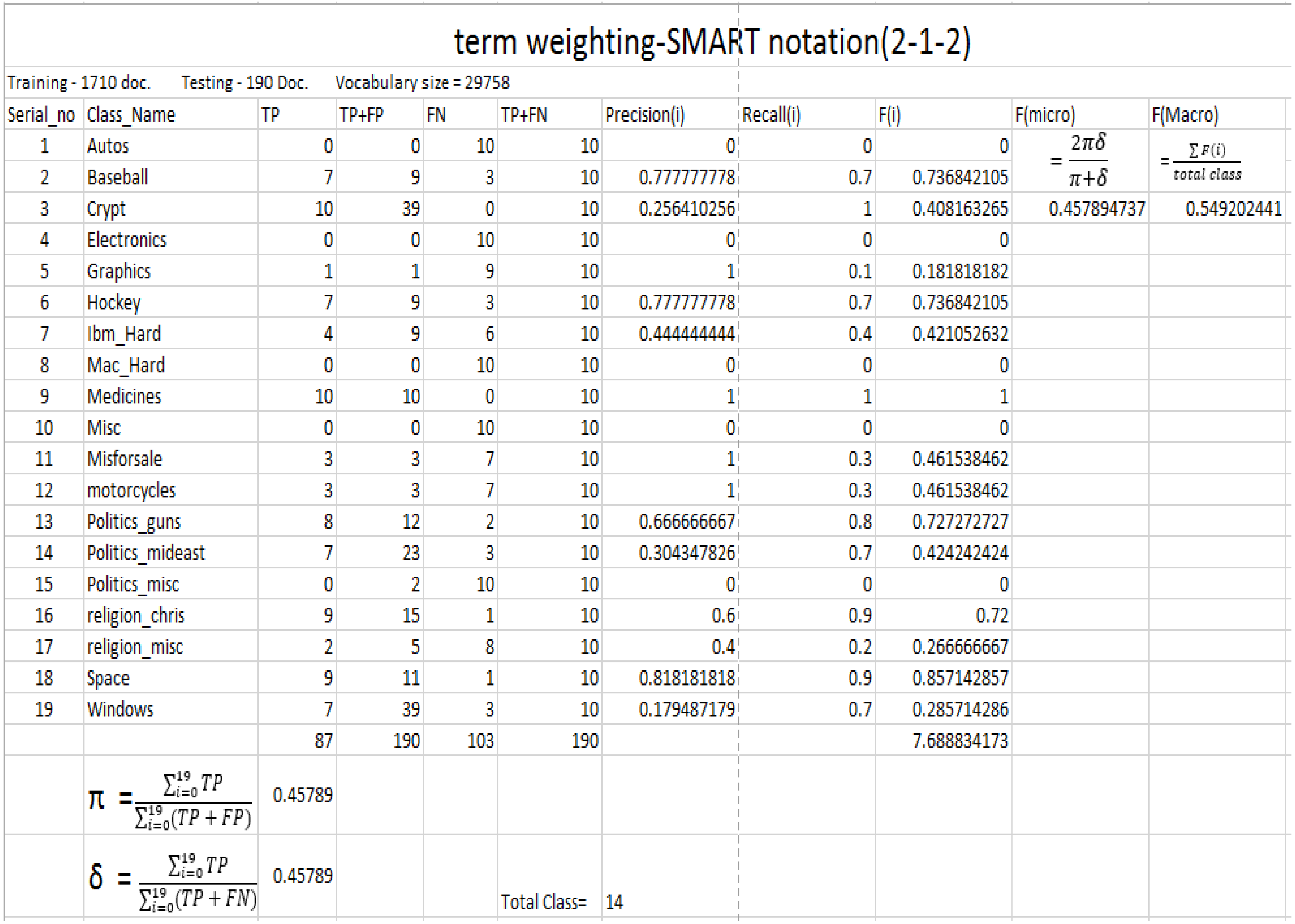


term weighting-SMART notation(2-1-2)

Training - 1710 doc. Testing - 190 Doc. Vocabulary size = 29935

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	1	1	9	10	1	0.1	0.181818182	$= \frac{2\pi\delta}{\pi+\delta}$	$= \frac{\sum F(i)}{total\ class}$
2	Baseball	9	12	1	10	0.75	0.9	0.818181818		
3	Crypt	10	40	0	10	0.25	1	0.4	0.468421053	0.513521038
4	Electronics	2	2	8	10	1	0.2	0.333333333		
5	Graphics	0	0	10	10	0	0	0		
6	Hockey	6	6	4	10	1	0.6	0.75		
7	Ibm_Hard	4	12	6	10	0.333333333	0.4	0.363636364		
8	Mac_Hard	0	0	10	10	0	0	0		
9	Medicines	4	4	6	10	1	0.4	0.571428571		
10	Misc	0	0	10	10	0	0	0		
11	Misforsale	1	1	9	10	1	0.1	0.181818182		
12	motorcycles	2	2	8	10	1	0.2	0.333333333		
13	Politics_guns	7	10	3	10	0.7	0.7	0.7		
14	Politics_mideast	8	10	2	10	0.8	0.8	0.8		
15	Politics_misc	9	19	1	10	0.473684211	0.9	0.620689655		
16	religion_chris	5	11	5	10	0.454545455	0.5	0.476190476		
17	religion_misc	4	4	6	10	1	0.4	0.571428571		
18	Space	9	12	1	10	0.75	0.9	0.818181818		
19	Windows	8	44	2	10	0.181818182	0.8	0.296296296		
		89	190	101	190			8.216336601		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.46842								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.46842			Total Class=	16				

Table 7 Training 5 (2-1-2 SMART Notation)

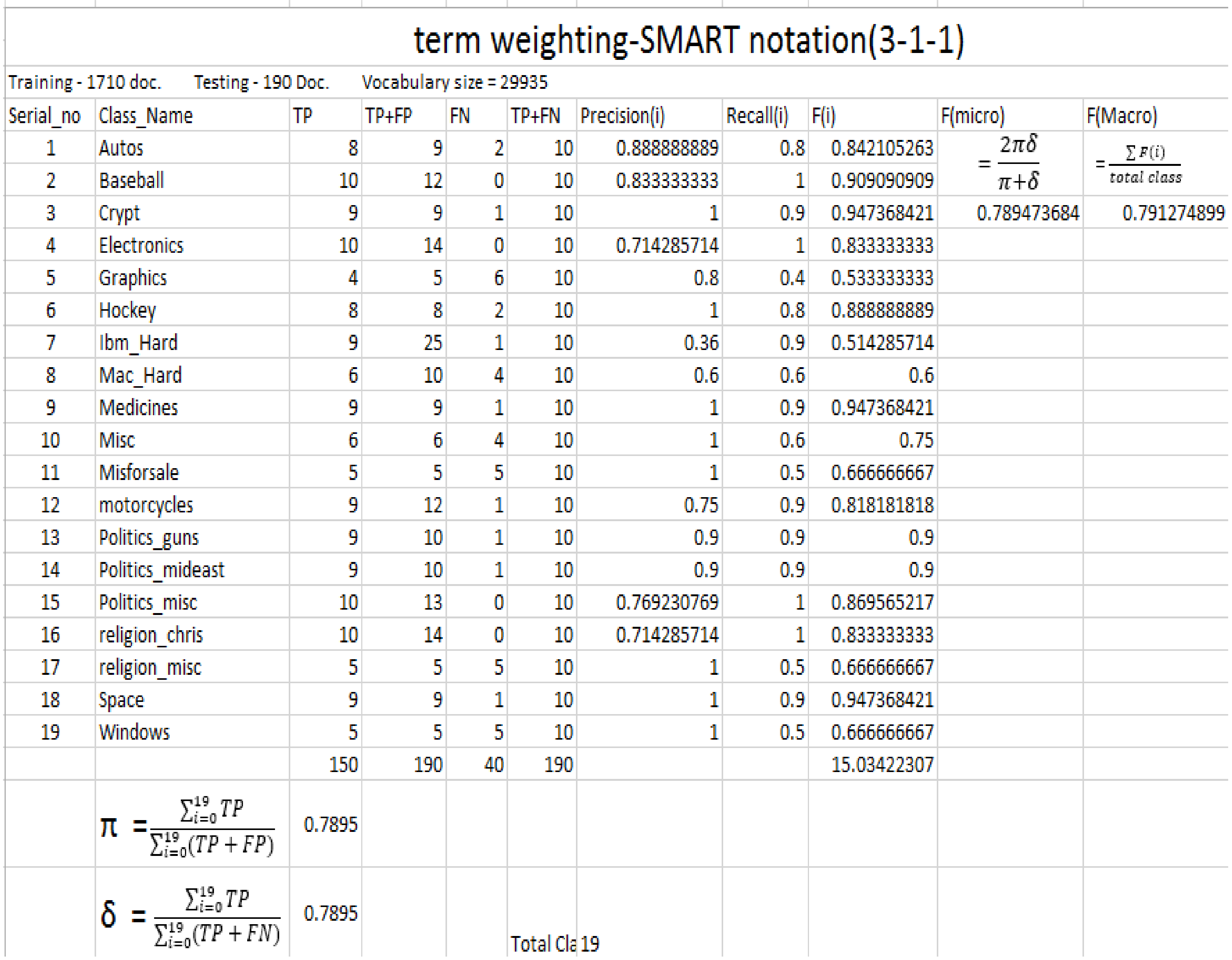


term weighting-SMART notation(2-1-2)

Training - 1710 doc. Testing - 190 Doc. Vocabulary size = 29758

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	0	0	10	10	0	0	0	$= \frac{2\pi\delta}{\pi+\delta}$	$= \frac{\sum F(i)}{total\ class}$
2	Baseball	7	9	3	10	0.777777778	0.7	0.736842105		
3	Crypt	10	39	0	10	0.256410256	1	0.408163265	0.457894737	0.549202441
4	Electronics	0	0	10	10	0	0	0		
5	Graphics	1	1	9	10	1	0.1	0.181818182		
6	Hockey	7	9	3	10	0.777777778	0.7	0.736842105		
7	Ibm_Hard	4	9	6	10	0.444444444	0.4	0.421052632		
8	Mac_Hard	0	0	10	10	0	0	0		
9	Medicines	10	10	0	10	1	1	1		
10	Misc	0	0	10	10	0	0	0		
11	Misforsale	3	3	7	10	1	0.3	0.461538462		
12	motorcycles	3	3	7	10	1	0.3	0.461538462		
13	Politics_guns	8	12	2	10	0.666666667	0.8	0.727272727		
14	Politics_mideast	7	23	3	10	0.304347826	0.7	0.424242424		
15	Politics_misc	0	2	10	10	0	0	0		
16	religion_chris	9	15	1	10	0.6	0.9	0.72		
17	religion_misc	2	5	8	10	0.4	0.2	0.266666667		
18	Space	9	11	1	10	0.818181818	0.9	0.857142857		
19	Windows	7	39	3	10	0.179487179	0.7	0.285714286		
		87	190	103	190			7.688834173		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.45789								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.45789			Total Class=	14				

Table 8Training 8 (2-1-2 SMART Notation)

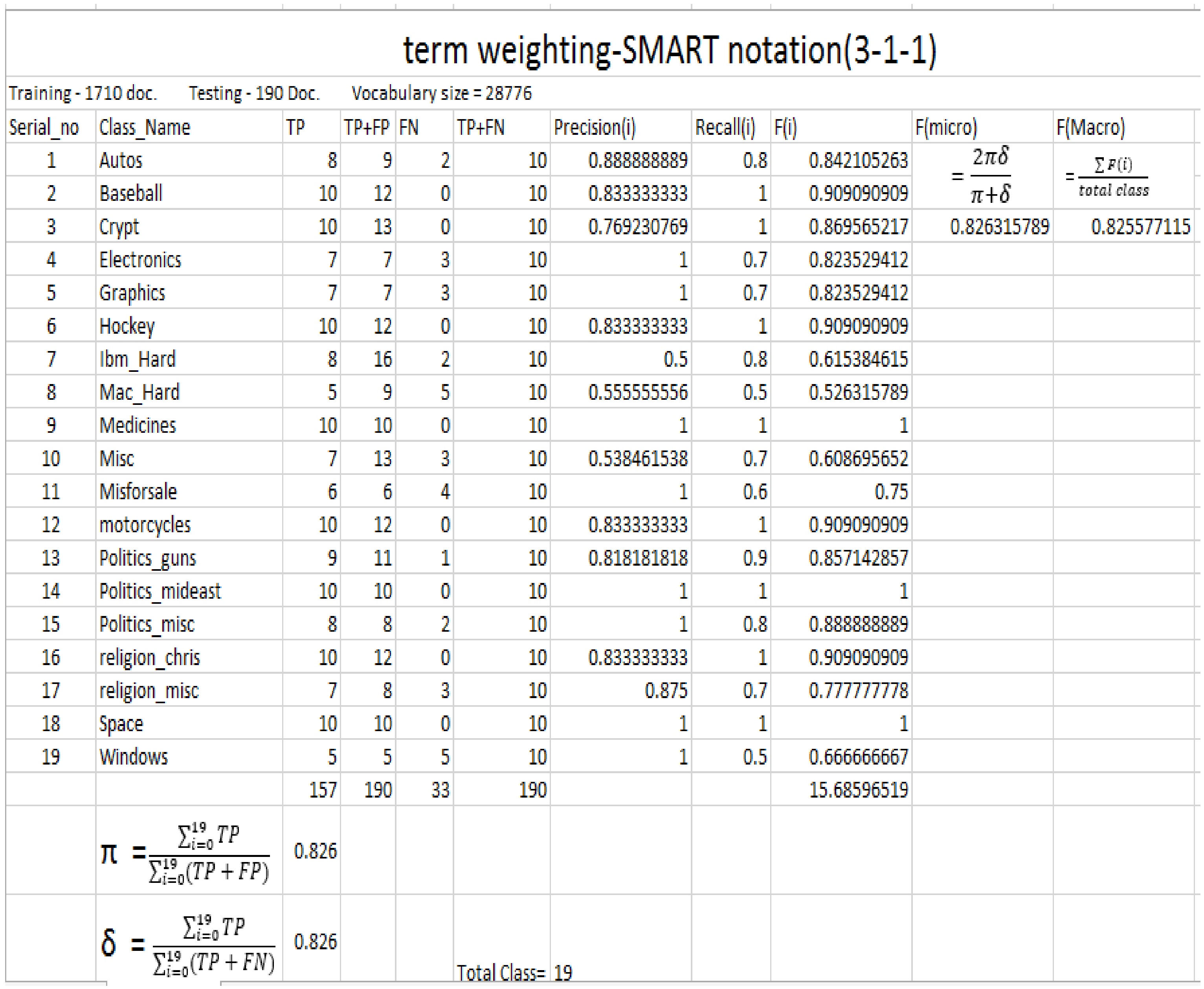


term weighting-SMART notation(3-1-1)

Training - 1710 doc. Testing - 190 Doc. Vocabulary size = 29935

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	8	9	2	10	0.888888889	0.8	0.842105263	$= \frac{2\pi\delta}{\pi+\delta}$	$= \frac{\sum F(i)}{total\ class}$
2	Baseball	10	12	0	10	0.833333333	1	0.909090909		
3	Crypt	9	9	1	10	1	0.9	0.947368421	0.789473684	0.791274899
4	Electronics	10	14	0	10	0.714285714	1	0.833333333		
5	Graphics	4	5	6	10	0.8	0.4	0.533333333		
6	Hockey	8	8	2	10	1	0.8	0.888888889		
7	Ibm_Hard	9	25	1	10	0.36	0.9	0.514285714		
8	Mac_Hard	6	10	4	10	0.6	0.6	0.6		
9	Medicines	9	9	1	10	1	0.9	0.947368421		
10	Misc	6	6	4	10	1	0.6	0.75		
11	Misforsale	5	5	5	10	1	0.5	0.666666667		
12	motorcycles	9	12	1	10	0.75	0.9	0.818181818		
13	Politics_guns	9	10	1	10	0.9	0.9	0.9		
14	Politics_mideast	9	10	1	10	0.9	0.9	0.9		
15	Politics_misc	10	13	0	10	0.769230769	1	0.869565217		
16	religion_chris	10	14	0	10	0.714285714	1	0.833333333		
17	religion_misc	5	5	5	10	1	0.5	0.666666667		
18	Space	9	9	1	10	1	0.9	0.947368421		
19	Windows	5	5	5	10	1	0.5	0.666666667		
		150	190	40	190			15.03422307		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.7895								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.7895			Total Cla	19				

Table 9 Training 9 (3-1-1 SMART Notation)

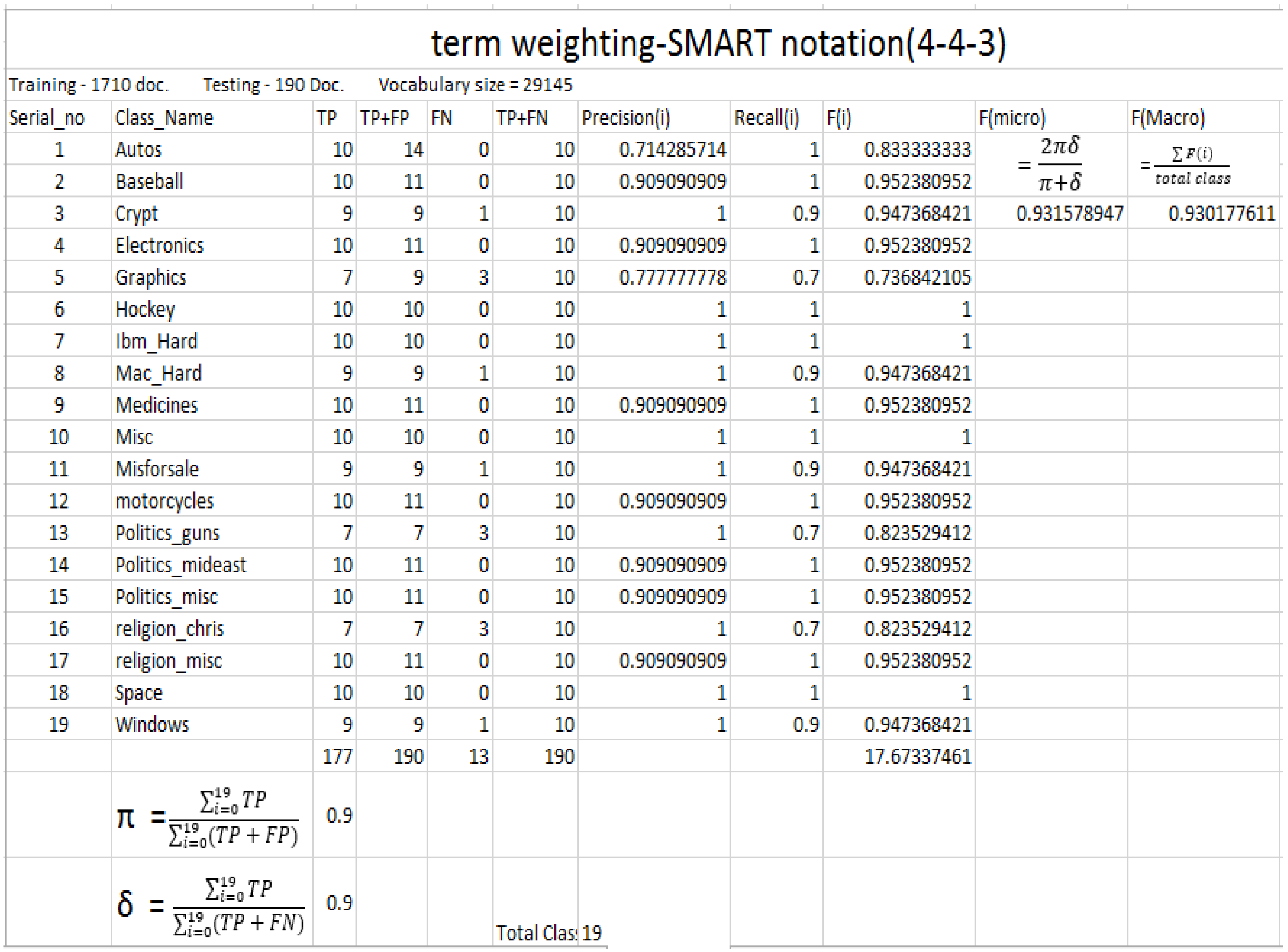


term weighting-SMART notation(3-1-1)

Training - 1710 doc.　Testing - 190 Doc.　Vocabulary size = 28776

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	8	9	2	10	0.888888889	0.8	0.842105263	$= \frac{2\pi\delta}{\pi+\delta}$	$= \frac{\Sigma F(i)}{total\ class}$
2	Baseball	10	12	0	10	0.833333333	1	0.909090909		
3	Crypt	10	13	0	10	0.769230769	1	0.869565217	0.826315789	0.825577115
4	Electronics	7	7	3	10	1	0.7	0.823529412		
5	Graphics	7	7	3	10	1	0.7	0.823529412		
6	Hockey	10	12	0	10	0.833333333	1	0.909090909		
7	Ibm_Hard	8	16	2	10	0.5	0.8	0.615384615		
8	Mac_Hard	5	9	5	10	0.555555556	0.5	0.526315789		
9	Medicines	10	10	0	10	1	1	1		
10	Misc	7	13	3	10	0.538461538	0.7	0.608695652		
11	Misforsale	6	6	4	10	1	0.6	0.75		
12	motorcycles	10	12	0	10	0.833333333	1	0.909090909		
13	Politics_guns	9	11	1	10	0.818181818	0.9	0.857142857		
14	Politics_mideast	10	10	0	10	1	1	1		
15	Politics_misc	8	8	2	10	1	0.8	0.888888889		
16	religion_chris	10	12	0	10	0.833333333	1	0.909090909		
17	religion_misc	7	8	3	10	0.875	0.7	0.777777778		
18	Space	10	10	0	10	1	1	1		
19	Windows	5	5	5	10	1	0.5	0.666666667		
		157	190	33	190			15.68596519		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.826								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.826			Total Class= 19					

Table 10 Training 5 (3-1-1 SMART Notation)

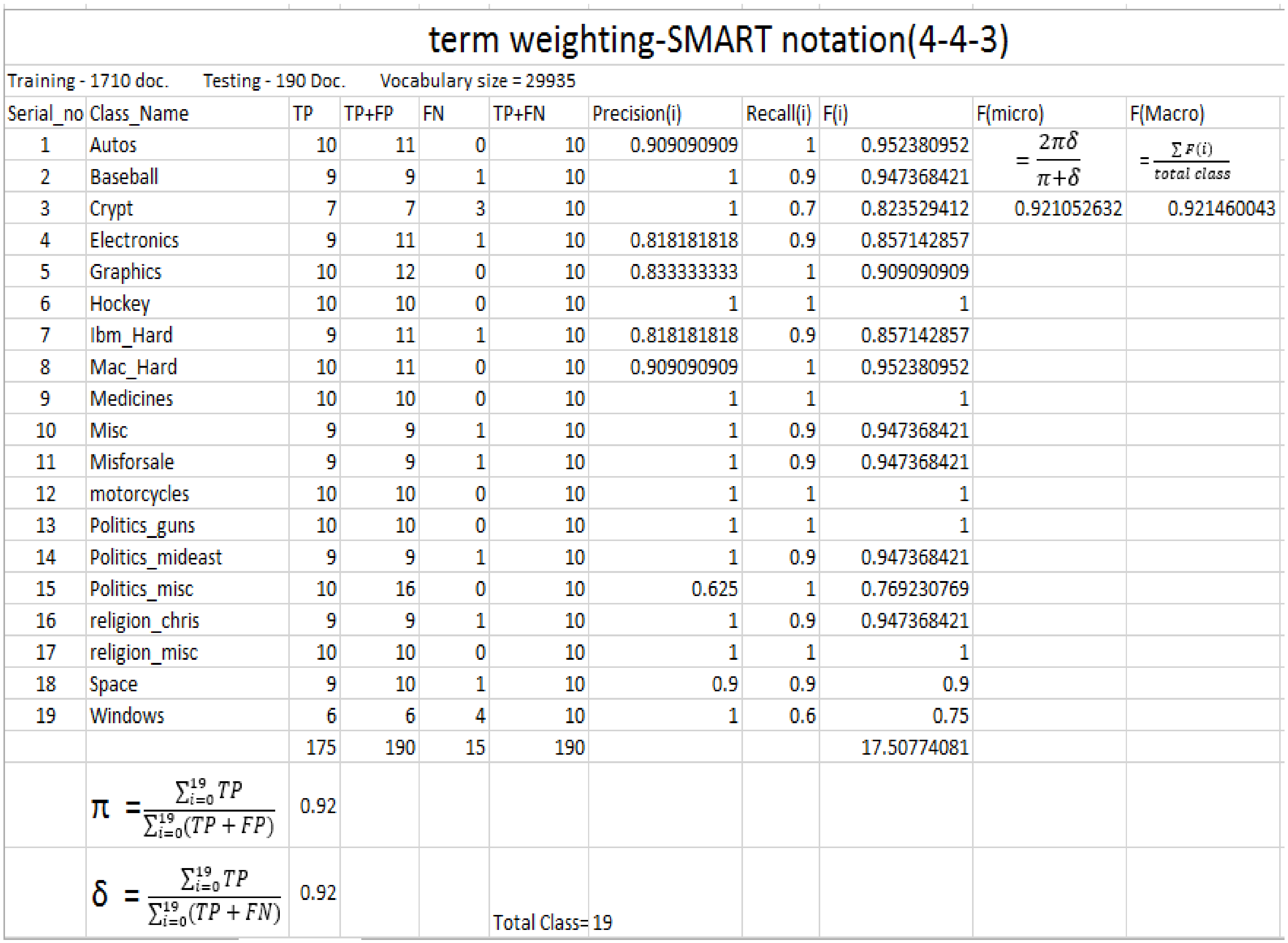


term weighting-SMART notation(4-4-3)

Training - 1710 doc. Testing - 190 Doc. Vocabulary size = 29145

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	10	14	0	10	0.714285714	1	0.833333333	$=\frac{2\pi\delta}{\pi+\delta}$	$=\frac{\sum F(i)}{total\ class}$
2	Baseball	10	11	0	10	0.909090909	1	0.952380952		
3	Crypt	9	9	1	10	1	0.9	0.947368421	0.931578947	0.930177611
4	Electronics	10	11	0	10	0.909090909	1	0.952380952		
5	Graphics	7	9	3	10	0.777777778	0.7	0.736842105		
6	Hockey	10	10	0	10	1	1	1		
7	Ibm_Hard	10	10	0	10	1	1	1		
8	Mac_Hard	9	9	1	10	1	0.9	0.947368421		
9	Medicines	10	11	0	10	0.909090909	1	0.952380952		
10	Misc	10	10	0	10	1	1	1		
11	Misforsale	9	9	1	10	1	0.9	0.947368421		
12	motorcycles	10	11	0	10	0.909090909	1	0.952380952		
13	Politics_guns	7	7	3	10	1	0.7	0.823529412		
14	Politics_mideast	10	11	0	10	0.909090909	1	0.952380952		
15	Politics_misc	10	11	0	10	0.909090909	1	0.952380952		
16	religion_chris	7	7	3	10	1	0.7	0.823529412		
17	religion_misc	10	11	0	10	0.909090909	1	0.952380952		
18	Space	10	10	0	10	1	1	1		
19	Windows	9	9	1	10	1	0.9	0.947368421		
		177	190	13	190			17.67337461		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.9								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.9			Total Clas 19					

Table 11 Training 2 (4-4-3 SMART Notation)



term weighting-SMART notation(4-4-3)

Training - 1710 doc. Testing - 190 Doc. Vocabulary size = 29935

Serial_no	Class_Name	TP	TP+FP	FN	TP+FN	Precision(i)	Recall(i)	F(i)	F(micro)	F(Macro)
1	Autos	10	11	0	10	0.909090909	1	0.952380952	$= \frac{2\pi\delta}{\pi+\delta}$	$= \frac{\sum F(i)}{total\ class}$
2	Baseball	9	9	1	10	1	0.9	0.947368421		
3	Crypt	7	7	3	10	1	0.7	0.823529412	0.921052632	0.921460043
4	Electronics	9	11	1	10	0.818181818	0.9	0.857142857		
5	Graphics	10	12	0	10	0.833333333	1	0.909090909		
6	Hockey	10	10	0	10	1	1	1		
7	Ibm_Hard	9	11	1	10	0.818181818	0.9	0.857142857		
8	Mac_Hard	10	11	0	10	0.909090909	1	0.952380952		
9	Medicines	10	10	0	10	1	1	1		
10	Misc	9	9	1	10	1	0.9	0.947368421		
11	Misforsale	9	9	1	10	1	0.9	0.947368421		
12	motorcycles	10	10	0	10	1	1	1		
13	Politics_guns	10	10	0	10	1	1	1		
14	Politics_mideast	9	9	1	10	1	0.9	0.947368421		
15	Politics_misc	10	16	0	10	0.625	1	0.769230769		
16	religion_chris	9	9	1	10	1	0.9	0.947368421		
17	religion_misc	10	10	0	10	1	1	1		
18	Space	9	10	1	10	0.9	0.9	0.9		
19	Windows	6	6	4	10	1	0.6	0.75		
		175	190	15	190			17.50774081		
	$\pi = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FP)}$	0.92								
	$\delta = \frac{\sum_{i=0}^{19} TP}{\sum_{i=0}^{19}(TP+FN)}$	0.92			Total Class= 19					

Table 12 Training 5 (4-4-3 SMART Notation)

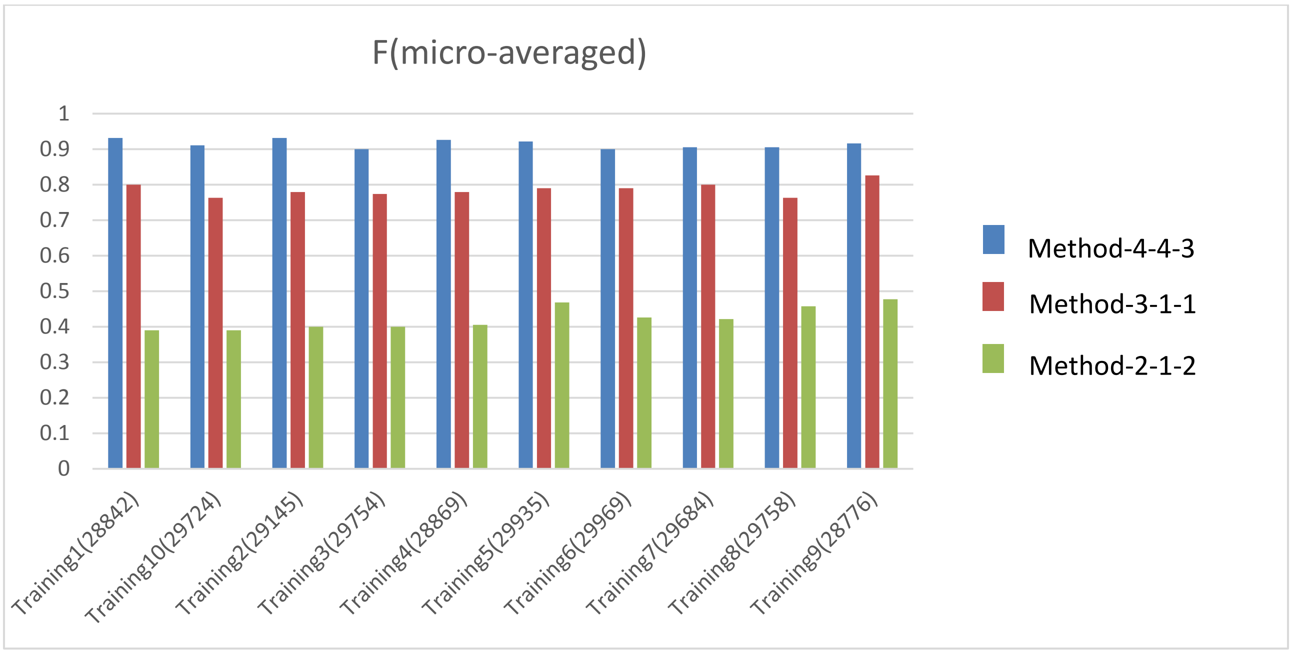


Chart 1 Comparison of F-micro

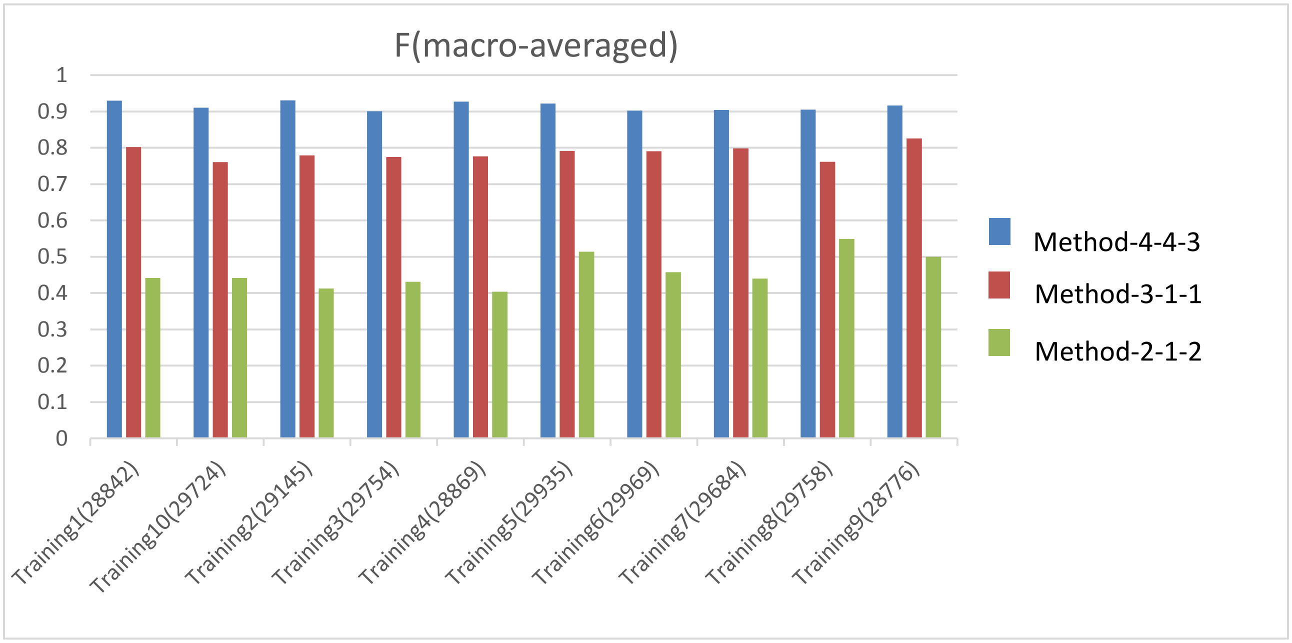


Chart 2 Comparison of F-macro

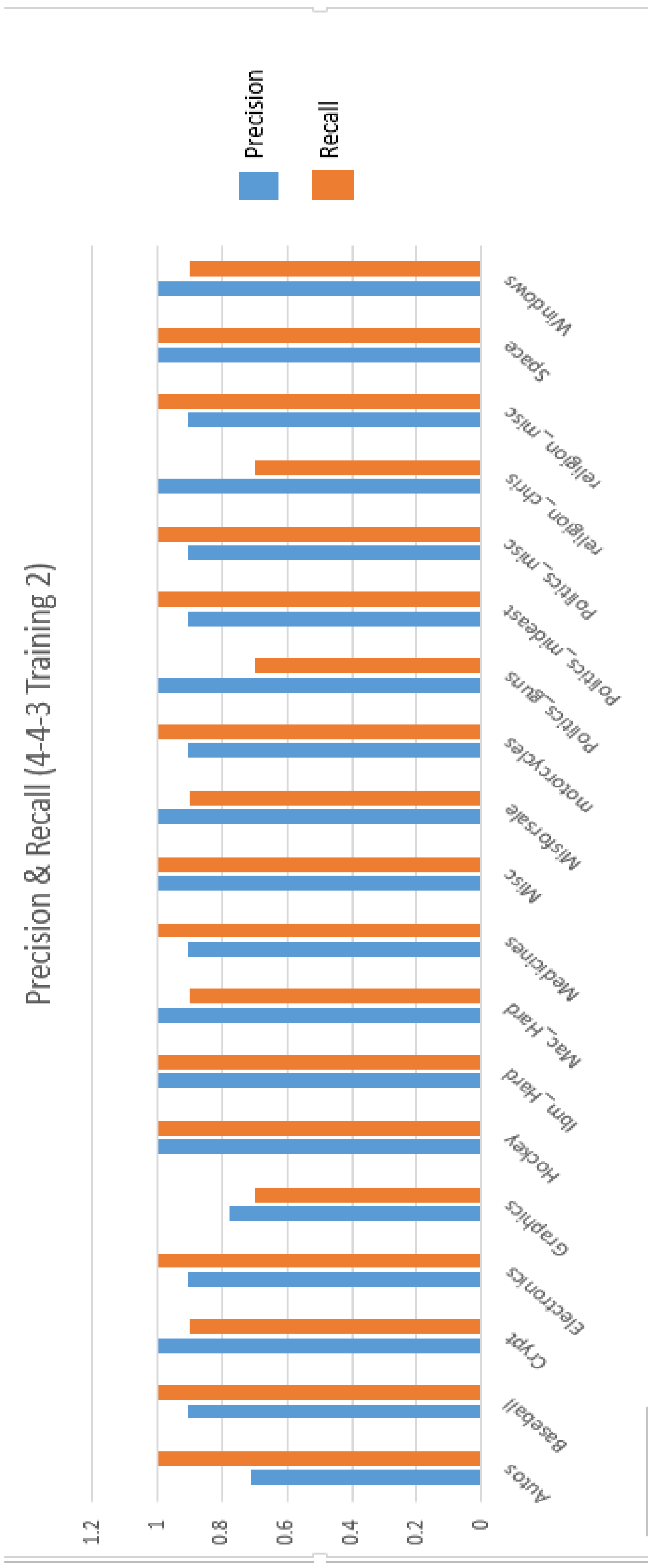


Chart 3 Class-wise Precision & Recall (4-4-3 training 2)

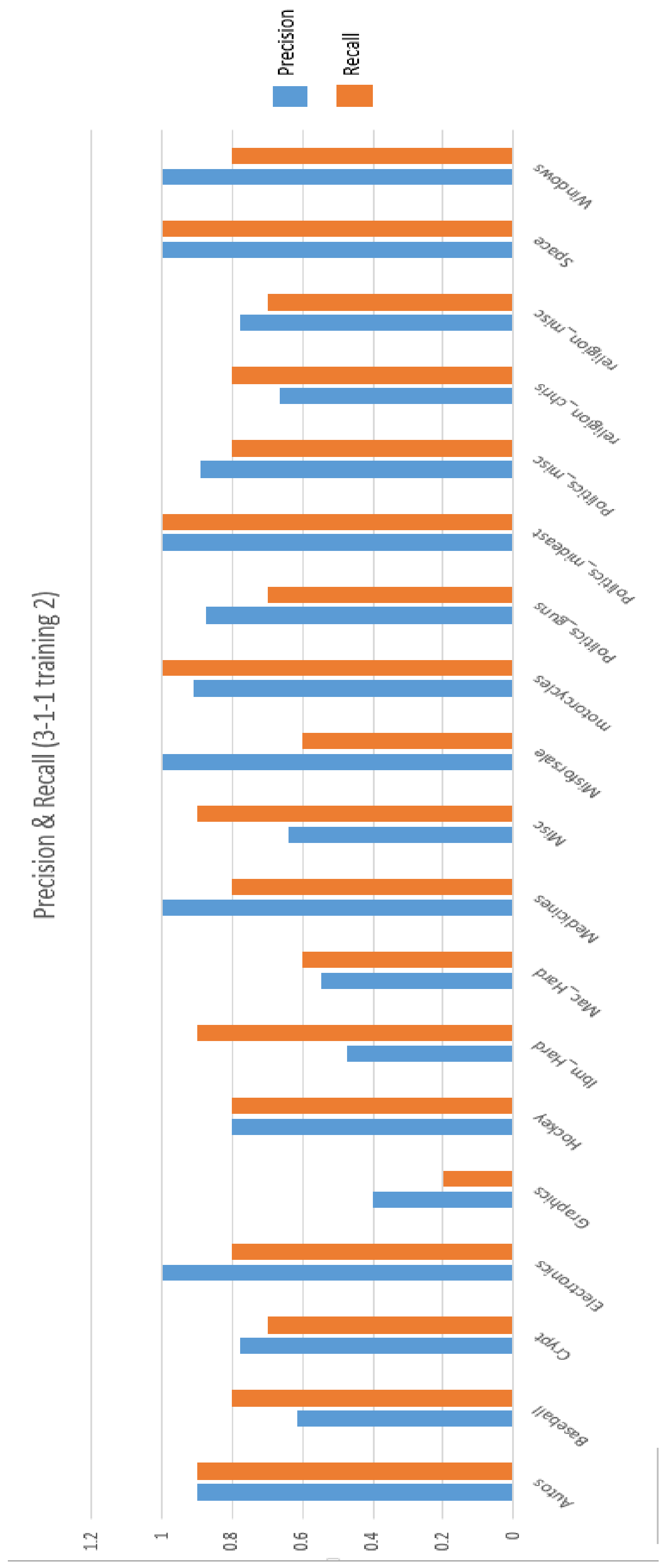


Chart 4 Class-wise Precision & Recall (3-1-1 training 2)

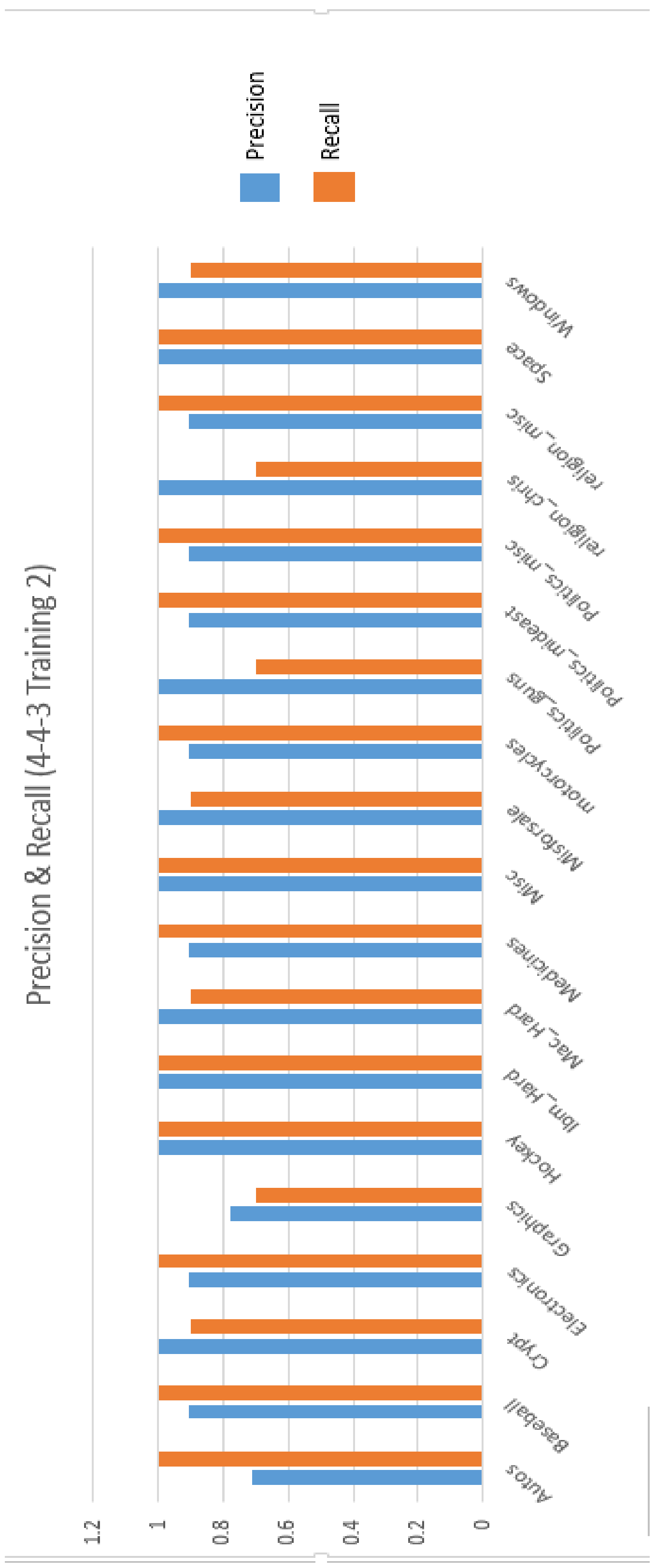


Chart 5 Class-wise Precision & Recall (2-1-2 training 2)

9. Conclusion and Future Work

In last two decades various researchers has experimented with the task of Text Document Classification using machine learning algorithms. The main aim of all this work is to improve the efficiency and accuracy of classifier. We have found that the Naïve Bayes approach we have used performs well with even large datasets.

In our project we have discussed different alternatives for document representation and explained our methodology in term weighting. We used the “bag-of-words” representation, where each distinct stemmed word is defined as a term. Stop words and mark-up tags are removed, words are stemmed by using Porter’s stemming algorithm. After the preprocessing phase, we implement and apply the leading supervised algorithms to the standard newsgroup document corpora.

In the training phase we implemented the multinomial naïve Bay model in which the classifier is trained with the help of training documents. And after the training, testing is done where a set of documents were tested and their categories were analyzed. It was found that a simple term weighting scheme gave preference to the longer documents than the shorter documents. Hence few other tem weighting techniques like logarithms, augmented term weighting were employed. Also to negate the effect of length of longer documents inverse documents frequent and normalization were also implemented. And it was found that doing so greatly improved the result as the F-measure, precision & recall each increased from around 0.45 to 0.78 and on further improvement it increased to around 0.92 with an accuracy of about 93%. Hence as a conclusion it can be said that naïve Bayes being so naïve can also produce good results if proper term weighting scheme and normalization techniques are applied.

However as a future work much work can be done on other supervised techniques like k-nearest neighbor (knn) and support vector machine (svm). These two supervised techniques have a better performance measures than Naive Bayes. Also much work can be done on dimension reduction where the size of the feature vector can be greatly decreased. Feature selection techniques like mutual information and chi square techniques can be applied to remove unwanted features.

10 Reference

1 https://en.wikipedia.org/wiki/Supervised_learning

2 https://en.wikipedia.org/wiki/Machine_learning

3 https://en.wikipedia.org/wiki/Precision_and_recall

4 https://en.wikipedia.org/wiki/F1_score

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

6 Nicola Polettini, "The Vector Space Model in Information Retrieval-Term Weighting problem"

7 Arzucan Ozgur," SUPERVISED AND UNSUPERVISED MACHINE LEARNING TECHNIQUES FOR TEXT DOCUMENT CATEGORIZATION".

8 Steven Bird, Ewan Klein, and Edward Loper, "Natural Language Processing with Python".

9 Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze,"Introduction to Information Retrieval".

10 Sebastiani, F., "Machine learning in automated text categorization", ACM Computing Surveys, Vol. 34, No. 1, pp. 1–47, 2002

11 Shweta Joshi, Bhawna Nigam," Categorizing the Document using Multi Class Classification in Data Mining".

12 A.M. Kibriya, E. Frank, G. Holmes and B. Pfaringer, "Multinomial Naïve Bayes for text Categorization Revisited".