

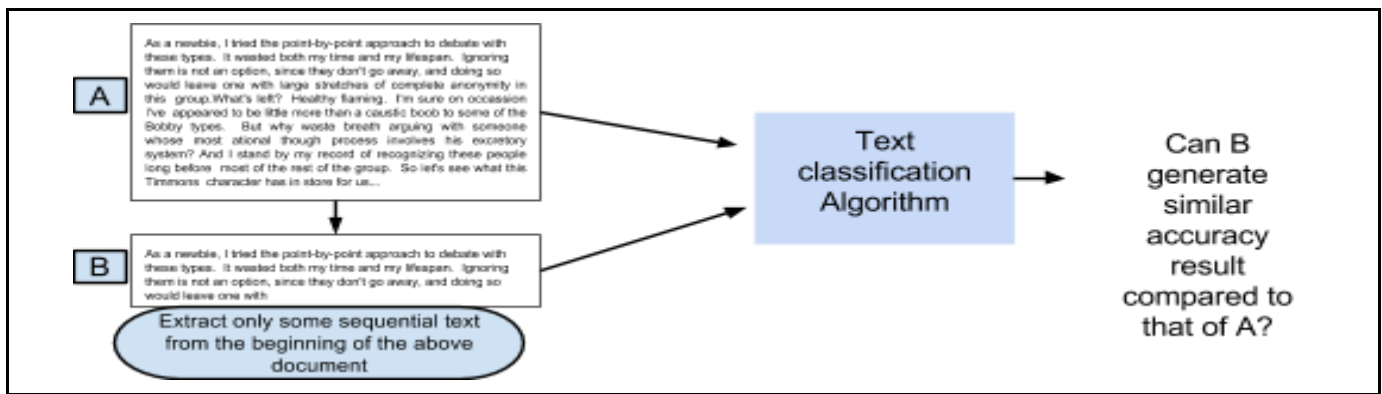
Saving Processing time for Text Classification algorithm by extracting and using only some fraction of sequential text from each document on a data set

By Yen Hoang, Anbang Xu, Taewoo Kim

1. Introduction/Abstract

In this Big Data era, there are many methods to expedite data processing such as parallelizing the process by Map Reduce. However, it is evident that the amount of whole data that needs to be processed does not change even though we apply parallelization. Based on this fact, we have explored possibilities to reduce amount of data that needs to be processed on three text classification algorithms - Naive Bayes, k Nearest Neighbor with $k=1$ (kNN), Decision Tree - on three data sets. We have found reasonable fraction parameter f which can be used to extract some proportion of sequential texts from the beginning of each test file that will be fed into these algorithms. Our results shows that extracting only 60% of sequential text from the beginning of each test file shows 90% of F-score compared to the F-score when we used 100% of sequential text from each test file.

Figure 1. Our Motivation



2. Problem Statement

Can we process some fraction of sequential text in the document, and achieve similar text classification result as we use original document? To answer this question, we will compare performance of three text classification algorithms on the three different data sets which have separate training and test data. Also, we will evaluate the impact of document size on these classification algorithms by comparing results using a fraction parameter f which is used to extract some sequential text from the beginning of each text file. Parameter f will be ranged from 0.1 to 1. By comparing f among data sets and algorithms, we will have insights whether our motivation described in the introduction can be achieved or not.

3. Data Sets

We chose to use three text categorization data sets - Reuters-21578 (hereafter Reuters), Ohsumed, 20Newsgroups since they have separated training and test set. Reuters data set consists of about 13,000

Reuters news articles categorized in 115 classes. Ohsumed data set includes about 23,000 medical abstracts from Medical Subject Headings (MeSH) categories. And 20Newsgroups includes about 19,000 news group postings categorized in 20 classes. As we can see in Table 1, size of these data sets are relatively small and entire data set can fit into memory. However, if we create quite large structures such as term frequency array across all terms and files, sometimes structure cannot fit into memory and we have to apply some optimization. We will discuss it later.

Table 1. Summary of three data sets

Data Sets	Reuters		Ohsumed		20Newsgroups	
	Training	Test	Training	Test	Training	Test
Number of Category	115	115	23	23	20	20
Number of Total Files	9,646	3,744	10,433	12,733	11,314	7,532
Number of Unique Files	7,773	3,019	6,286	7,643	9,840	6,871
Number of files belong to multi-category	1,223	436	2,929	3,600	1,428	617
Total Size	9,073KB (8.9MB)	3,656KB (3.6MB)	12,130KB (11.8MB)	15,183KB (14.8MB)	21,538KB (21MB)	13,477KB (13.2MB)
Average File Size	0.94KB	0.97KB	1.16KB	1.19KB	1.90KB	1.78KB
Average Number of Files per Category	83.9	32.6	453.6	553.6	565.7	376.6
Maximum Number of Files in a Category	2,877 (cat: earn)	1,087 (cat: earn)	1,799 (cat: C23)	2,153 (cat: C23)	600 (cat: rec.sport.hockey)	399 (cat: rec.sport.hockey)
Minimum Number of Files in a Category	1 (Only 1 file in 25 categories)	0 (No test files in 25 categories)	65 (cat: C03)	70 (cat: C03)	377 (cat: talk.religion.misc)	251 (cat: talk.religion.misc)

4. Methods/Approach

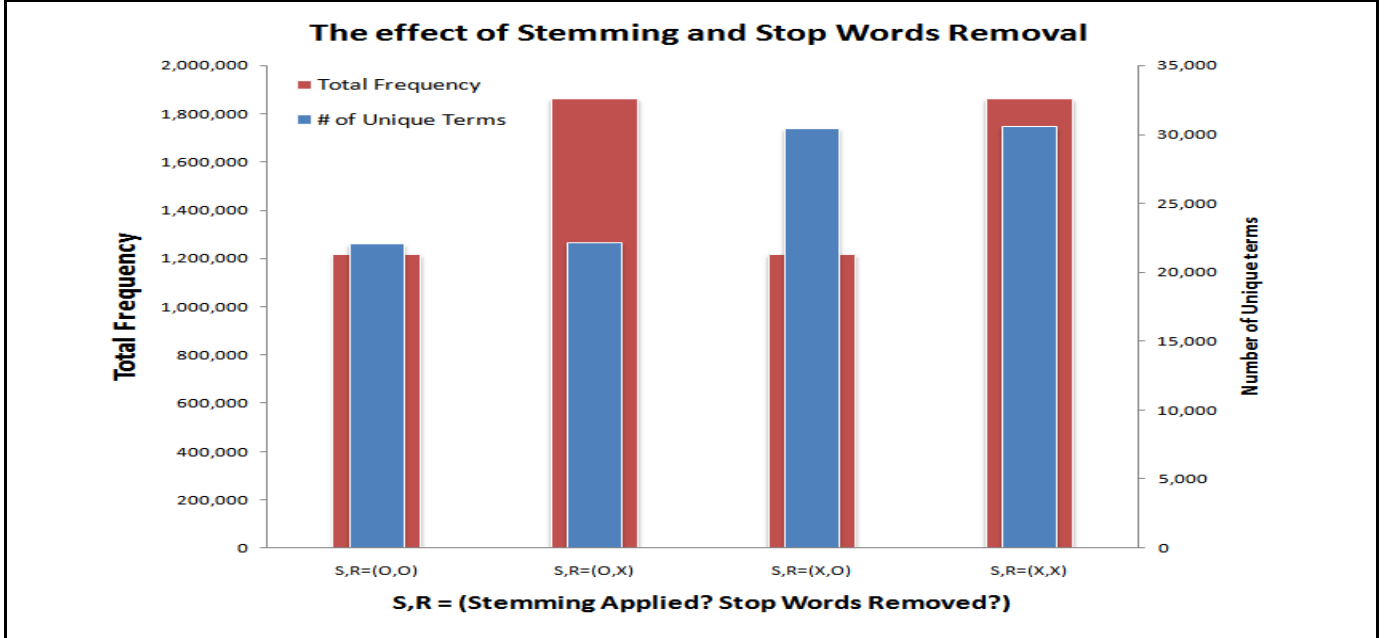
In this section, we are going to discuss pre-processing, algorithms, and our methods to get results.

1) Pre-Processing

In order to get good result, we tokenized original text file using white space as delimiter, then applied post - tokenization process such as stemming, removing stop words and non alpha numeric words. As

we can see in Figure 2, the effect of removing stop words are noticeable. On the Ohsumed data set, about 35% of total frequency can be reduced when we removed stop words in the data set. In other words, 35% of frequency in the data set comes from common stop words. Also, after we applied stemming, number of unique terms were reduced by about 28%. Therefore, we have decided to use both methods to pre-process each data set. We used the Natural Language Toolkit (NLTK) for stemming and stop words removal.

Figure 2. The effect of Stemming and Stop Words Removal on the Ohsumed data set



For Decision Tree, in addition to these methods, we had to apply another pre-processing since the number of term were large, we could not train the data set properly by using main memory even though RAM size was about 80GB. Therefore, we dropped a term if the frequency of term is less than 0.001% of total term frequency. This is common method to reduce invalid terms as suggested in [13] based on the Zipifan distribution. Interestingly, reducing rare terms also made the Decision Tree to achieve higher score than the score where rare term were not removed since noise terms were removed.

Table 2. The effect of rare term removal (frequency < 0.001% of total frequency)

Data Set	Reuters		Ohsumed		20Newsgroups	
	Training	Test	Training	Test	Training	Test
Number of Terms	20,275	13,077	22,047	24,598	106,993	75,196
Number of Terms after applying threshold	4,704	2,838	4,407	4,988	6,442	4,618
Reduce Rate(%)	76.8%	78.3%	80.0%	79.7%	93.9%	93.8%

2) Algorithms and our implementation

We chose three algorithms - Naive Bayes, kNN with $k=1$ using cosine similarity as distance measure, and Decision Tree and have implemented on our own without using reference package.

a. Naive Bayes: Naïve Bayes classification is a method based on Bayes rule and “Bag of Words” representation of document and assumption about conditional independence of words given class. In training step, we estimate probability of each class c , $p(c)$ and each word w given class c , $p(w|c)$. Based on the assumption about conditional independence of words, we can estimate $p(d|c) = \prod p(w|c)$. In test step, we use Bayes rule to estimate probability of each class c given document d , $p(c|d)$, then predict class of d as class c^* with largest estimated probability, $p(c^*|d) = \max p(c|d)$. To avoid the problem with word w that never appears in class c , that means $p(w|c) = 0$, we used smoothing parameters to get estimated probability. At the beginning, we used a formula from the lecture [12] $P(x_j = 1 | c_k) = (n_{jk} + \alpha) / (n_k + \alpha + \beta)$ with smoothing parameters $\alpha = 1$, $\alpha + \beta = 1000$ to check whether word x_j is in class c_k , with n_k is total number of word in class c_k and n_{jk} is the frequency of word x_j in class c_k . However, the results were very bad, the fraction of correctly assigned files of test set was 0.4528 and training set was 0.5290. Then, we tried other smoothing formula from [9] $P(x_j = 1 | c_k) = (n_{jk} + \alpha) / (n_k + \alpha * |V|)$ where $|V|$ is size of vocabulary from training data set and α is smoothing parameter. With this formula, we tried different value of α and we get best result was 0.862206 with $\alpha = 0.08$. So, we chose $\alpha = 0.08$ in the experiments with Naive Bayes implementation.

b. kNN with $k=1$ using cosine similarity as distance measure: Using Bag of Words where each word is represented by term frequency, we create a vector for each document to represent each word by using Term Frequency - Inverse Document Frequency (TF-IDF) score and calculate cosine similarity between these two vectors. Higher value means that two documents are more similar. In our implementation, each label (category in the training set) is represented as a vector by accumulating all terms and it's frequency from each file in the category. Then, we create a vector for each test file and calculate cosine similarity between this vector and a vector that represents each category. The result will be used to classify a test file to a category which has maximum cosine similarity score. Therefore, we have implemented kNN with $k=1$ using cosine similarity as distance measure in TF-IDF space.

c. Decision Tree: In short, a Decision Tree is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a decision.[1] So three main steps here: 1. Represent our data as Tree and TreeNode data structure; 2. Build/Grow the Decision Tree based on the input data; 3. Make prediction from the built tree. The key point for the Decision Tree algorithm is how to choose the best split feature and its best pivot.

For our implementation, we try to iterate through the evaluations of all the possible splits and returns the best one based on the metrics. There are two main metrics we use here: **Gini** and **Entropy** [14]. However, in fact, trying out all the possible splits is very time-consuming. Therefore, we split/divide up the continuous variable into k equal ranged groups (discretizing). After training, each non-leaf node has depth, split_attribute, split_frequency_pivot. On the contrary, each leaf node has depth and the prediction - probability that it belongs to each category if a test data point reaches it. Also, some important parameters can be adjusted, such as “criterion”, “max_depth”, “max_feature”. “**criterion**” is the function to measure the quality of a split. “**max_depth**” is the maximum depth of the tree. “**max_feature**” is the number of features to consider when looking for the best split. To measure the performance of the Decision Tree algorithm, we measured the relation between correctness and the depth. We also measured the relation between execution time and depth.

3) Our methods

First of all, to train the text classifier, we have processed each file in the training set with fraction parameter $f=1$ since our focus is to check the result of test file with various fraction parameter, not train the classifier with various fractions. For each file in the test set, we have extracted some fractions of sequential text from the beginning of original file. The fraction was ranged from 0.1 (10%) to 1 (whole file). The interval was 0.1. Therefore, for each data set, we have 10 experiment sets ($t1 \sim t10$). Since there are three data sets, we execute 30 experiments to get entire picture. To measure effectiveness of each algorithm, we have calculated micro and macro F-score to measure accuracy using Precision and Recall. As mentioned in [8], F-score can be calculated as $2RP / (R + P)$ where R is recall and P is Precision. Between macro and micro, macro average method treats each category equally. On the contrary, micro average method puts more weight on high frequency classes.

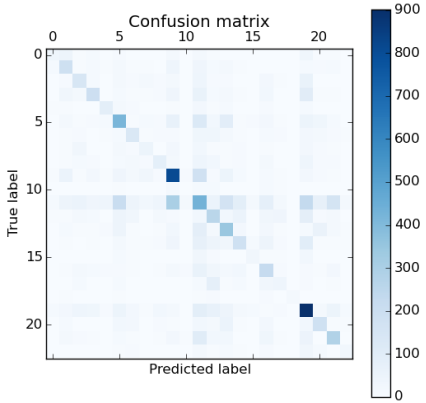
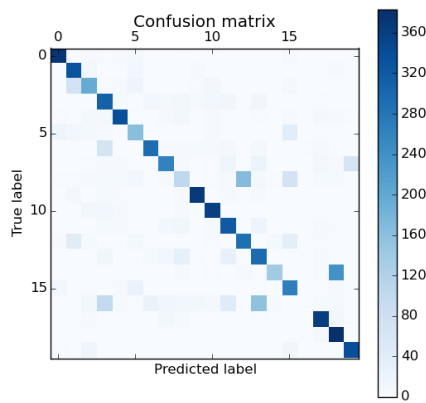
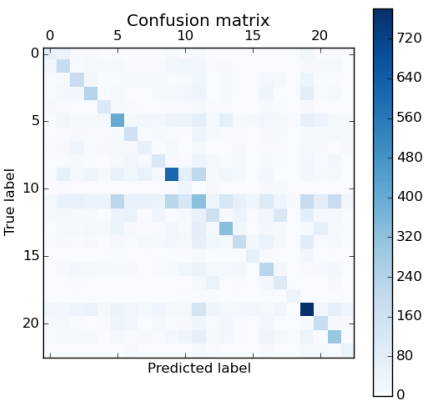
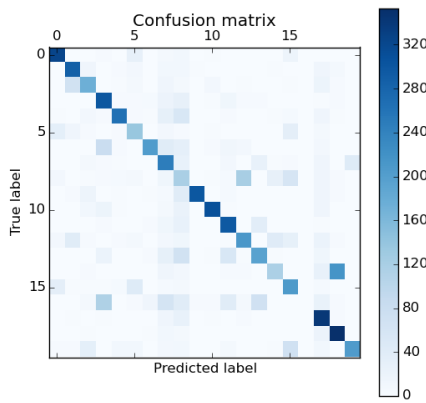
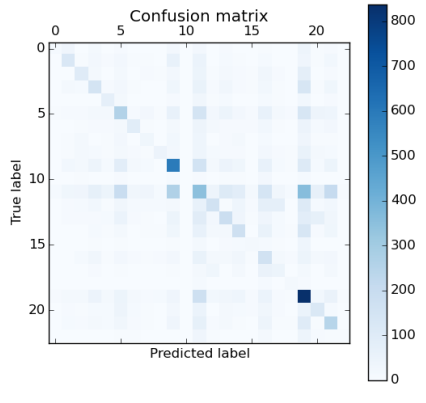
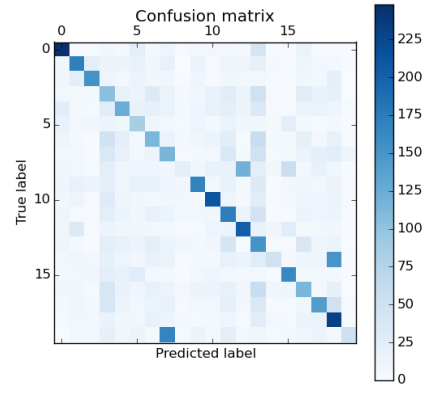
5. Results

In this section, we have measured performance of our implementation, then gather results of each algorithm on three data sets by using fraction parameter f . Based on these results, we present relative F-score graph compared between various of f and $f=1$.

1) Performance of three Algorithm on whole Fraction ($f=1$)

Before checking the result of each fraction, we checked the result using original test file. The reason is that if the result from original test file is not good, it is not desirable to use that algorithm. Here are calculated confusion Matrix for three data sets using three algorithms. In the graph, y-axis represents true label, and x-axis shows predicted label. Dense color means that number of instances located in that section is high. In the result, you can see strong diagonal line is shown in the 20Newsgroups data sets. This means that true label is predicted correctly. Actually, the best F-Score of 20Newsgroups data set ($=0.728$) among three algorithms is higher than that of Ohsumed data set ($=0.396$).

Figure 3. Confusion Matrix graph on two data sets* (f=1)

Method	Ohsumed Data Set	20Newsgroups Data Set
Naive Bayes	 <p>micro average F-Score: 0.396</p>	 <p>micro average F-score: 0.728</p>
kNN with k=1	 <p>micro average F-Score: 0.384</p>	 <p>micro average F-Score: 0.609</p>
Decision Tree	 <p>micro average F-Score: 0.246</p>	 <p>micro average F-Score: 0.371</p>

* We exclude Reuters data in this graph since it is not feasible to see 115 categories in the confusion matrix graph.

Here is the execution time and F-Score result for three algorithms on largest data set (20Newsgroups). Fraction parameter used here is $f=1$. We can see that the F-score of Naive Bayes is best among three algorithms. However, the execution time of kNN is fastest.

Table 3. Execution Time and F-Score on 20Newsgroups Data set* ($f=1$)

Algorithm	Decision Tree	Naive Bayes	kNN with $k=1$
Execution Time (s)	736.0	1,542.5	8.92
- Training (s)	731.4	738.4	0.82
- Test (s)	4.6	804.1	8.1
Micro F-Score	0.371	0.728	0.609
Macro F-Score	0.366	0.701	0.594

2) F-Score for 10 Fractions (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

For these 10 fractions, we calculated micro average F-score and macro average F-score to see their performance. As we described in the previous section, Naive Bayes performs well even when we use fraction parameter smaller than 1. kNN follows next and Decision Tree ranks the last. Since Naive Bayes took much time on 20Newsgroups data set, we present the execution time of Naive Bayes for each fraction. This table clearly shows that if we process test set by using fraction parameter $f < 1$, processing time can be saved.

Table 4. Execution Time and F-Score of Naive Bayes on the 20Newsgroups data set

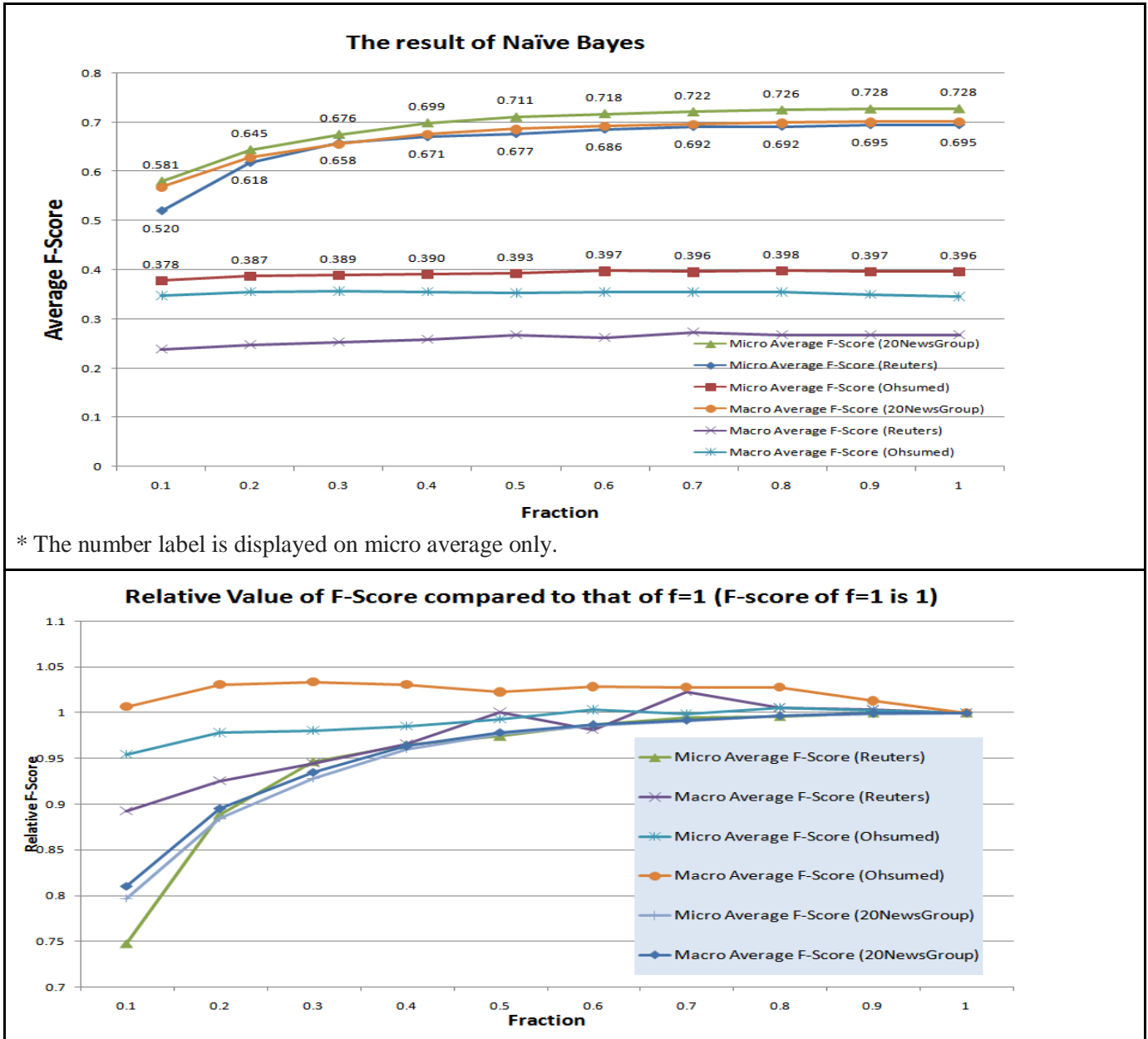
Fraction	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Execution Time (s)	874.4	976.4	1,063.3	1,144.2	1,219.7	1,290.6	1,357.9	1,423.3	1,484.6	1,542.5
- Training (s)	738.4	738.7	738.5	738.5	738.5	738.3	738.3	738.8	738.6	738.4
- Test (s)*	136.0	237.7 (+101.7)	324.8 (+87.1)	405.7 (+80.9)	481.2 (+75.5)	552.3 (+71.1)	619.6 (+67.3)	684.5 (+64.9)	746.0 (+61.5)	804.1 (+58.1)
Micro F-Score	0.580	0.644	0.676	0.699	0.711	0.718	0.722	0.725	0.727	0.728
Macro F-Score	0.568	0.628	0.655	0.675	0.686	0.692	0.696	0.699	0.700	0.701

* (+ number) means additional processing time for that fraction compared to the previous (left) fraction.

We now understood that when we use fraction parameter $f=0.3$, it can produce nearly 90% of quality level compared to the result when we use $f=1$ on Naive Bayes. Decision Tree and kNN with $k=1$ showed 90% of quality level with $f=0.6$. To see this point clear through figures, we regard the F-score of $f=1$ as 1 (100%), we draw relative score graph for each fraction. Note that when fraction reaches 0.6, all F-score reaches more than 90% of $f=1$ score. For fraction greater than 0.6, this difference becomes even smaller.

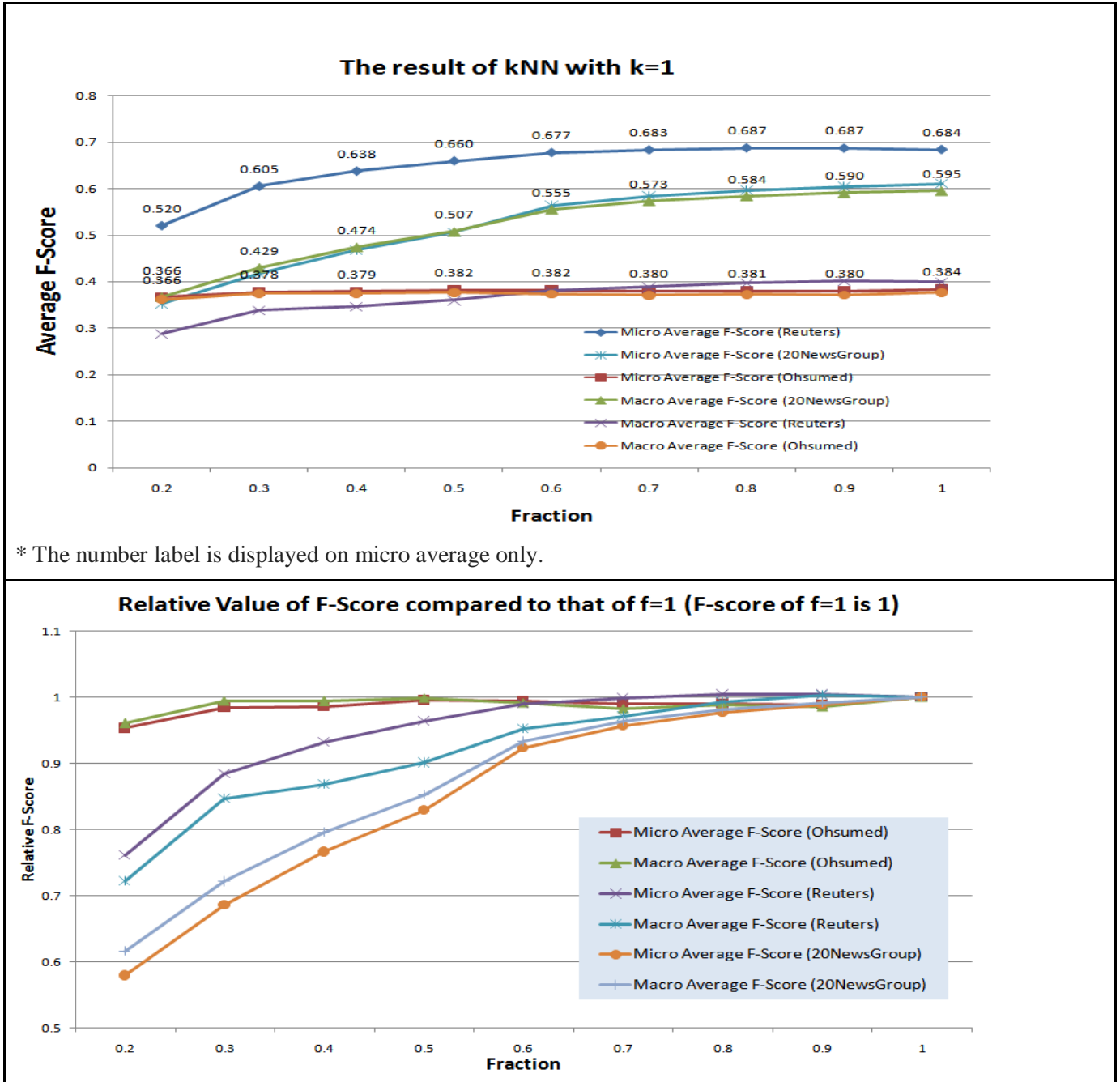
The following figure shows the performance and relative score of Naive Bayes. For Naive Bayes, when fraction reaches 0.3, all F-score hits above 90% of F-score with $f=1$. It is interesting to note that some micro F-score is greater than the micro F-score of $f=1$ in this figure.

Figure 4. Average F-Score and Relative score on three data sets using Naive Bayes



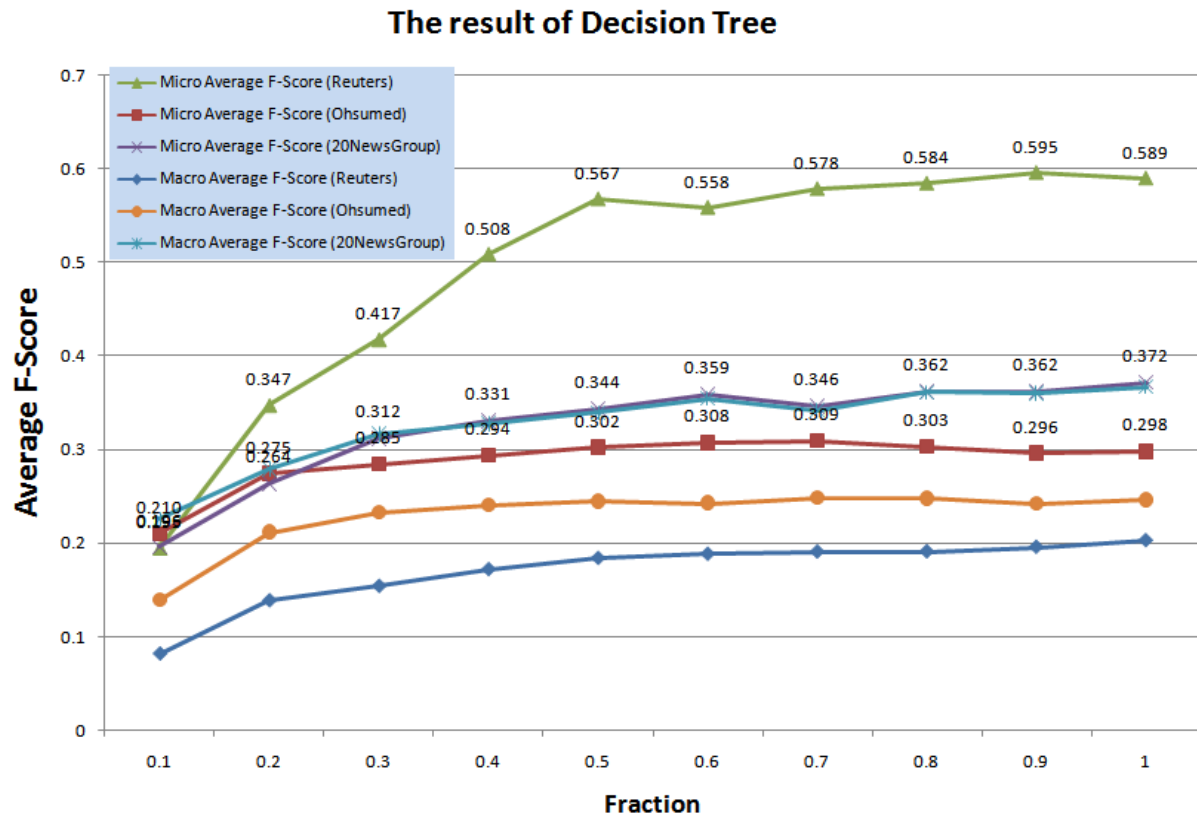
Compared to Naive Bayes, kNN performs worse. However, its execution time is faster. On the 20Newsgroups with $f=0.9$, kNN took 8.9 seconds to generate the results and it's micro F-score was 0.590. The micro average score of Naive Bayes on the 20Newsgroups with $f=0.9$ was 0.727. Still, when we use $f=0.6$, the relative F-score hits 90% level of the score when we use $f=1$.

Figure 5. Average F-Score and Relative score on three data sets using kNN with k=1

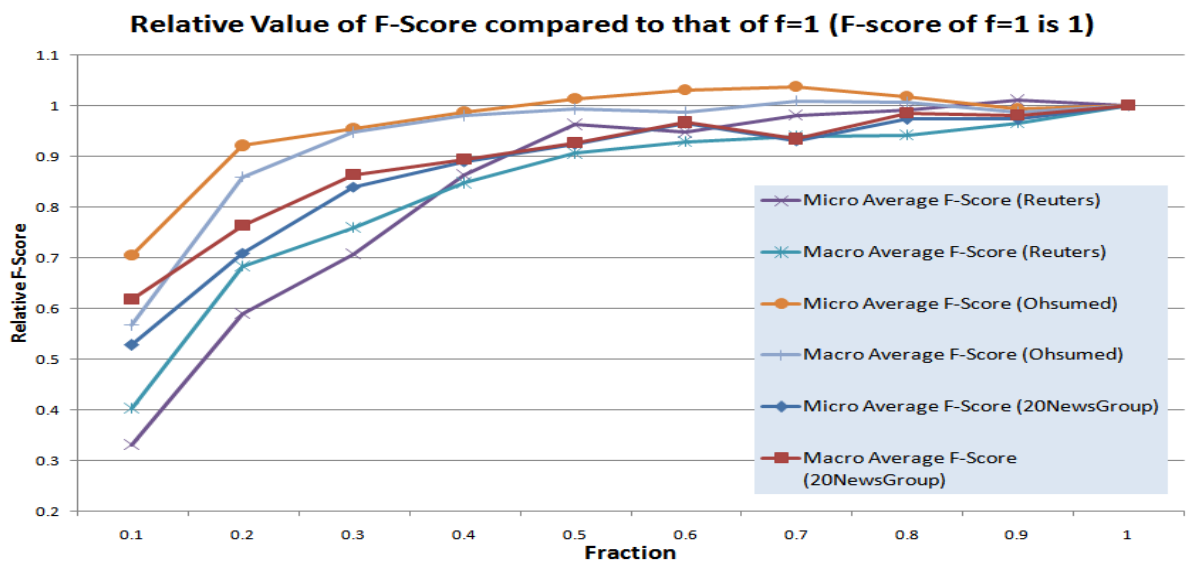


Decision Tree took almost 740 seconds to test entire test files on 20NewsGroups regardless of fraction parameter f . However, it's performance was not good compared to other algorithms. When we use fraction parameter f as 0.5, it already achieves 90% of score level of $f=1$.

Figure 6. Average F-Score and Relative score on three data sets using Decision Tree



* The number label is displayed on micro average only.



6. Discussion

1) Lessons Learned

Since concept of algorithm or pseudo code does not tell us exact structure that we need to create, much of implementation was based on our decision. Implementing Decision Tree required quite efforts compared to Naive Bayes and kNN. It required complex structure and a lot of memory. Initially, it worked on some data sets and it failed on other data sets. Therefore, we had to change data structure and implement rare term removal. However, performance of Decision Tree were not increased much compared with efforts we put. We have learned again that we need to allocate time slot more efficiently as our main goal was not to implement our own code.

2) Increasing fraction makes the accuracy raise. However, accuracy does not increase linearly.

As we expected, when we increase the fraction parameter f , the effectiveness of algorithm tends to increase, too. However, as it reaches certain level, it doesn't increase linearly. This is where we can achieve our goal because if the F-score increases linearly, there is no reason to pick some fraction of each document since in order to get high F-score, we just pick higher fraction and it tends to grow up to 1. Since this is not the case, we can conclude that we can process 60% of each document to yield similar results compared to the result which we process original document. As we suggested in the result section, this can save a lot of time especially on the large data set.

3) Discussion about each algorithm

a. Naive Bayes: Performance of Naive Bayes can be improved with feature selection.

Although the accuracy of Naive Bayes is best in the three algorithms that we implemented, it works much worse than other implementations from [10]. The micro accuracy of their algorithm for Multinomial Naive Bayes, Transformed weight – normalized complement Naïve Bayes and Support Vector Machine - SVM for 20Newsgroups are 0.848, 0.861, 0.862, respectively whereas accuracy of our Naive Bayes is 0.728. For Reuters, their accuracies are 0.739, 0.844, 0.887 and accuracy of Naive Bayes is 0.695. After considering detail of data sets and algorithm, we found the reason for the bad results is that we are using conditional independence probability to compute $p(d|c)$ with formula $p(d|c) = \prod (p(w|c))$. So, if document d does not belongs to class c but contains many words that occur frequently in c , most likely it is predicted to be in c . To improve the accuracy of Naive Bayes, we can use feature selection techniques to remove those words that don't distinguish one class with others.

b. kNN with $k=1$: Simple method is doing well.

Before beginning the experiment, we expected that Decision Tree or Naive Bayes algorithm works well and kNN with $k=1$ performs worst since kNN with $k=1$ is the most simplest algorithm among three. However, the result showed another picture. kNN ranks second among three algorithms and execution time of kNN was shortest. Considering execution time and its performance, we think that it can be used

as rough estimation of data set. For example, classifying the result before real analyst look at the result to correct.

c. Decision Tree: Too Much Information may slow down decision-making capacity

From our experiment, the F-score of the Decision Tree is worst on the three data sets. Four main reasons: a. The simple Iterative Dichotomiser 3 - ID3 - Decision Tree algorithm is not well suited for multi-label classification. The performance can be improved by hierarchical multi – label dataset[15]; b. we apply “Discretization” to preprocess dataset, but we don’t spend time figuring out the best variable of category granularity; c. Without choosing proper parameters to prune or limit tree growth, they tend to overfit or underfit the training data, making them somewhat poor predictors. But on the other hand, compared with kNN and Naive Bayes, the Decision Tree can be easily interpreted, visualized and explained. It’s a “white boxes” in the sense that the acquired knowledge can be expressed in a readable form. Another advantage is that it doesn’t require some form of normalization or scaling before we can fit it. It also implicitly performs feature selection. When we fit a Decision Tree to a training data set, the top few nodes on which the tree is split are essentially the most important variables within the data set and feature selection is completed automatically. From the time perspective, it has an exponential calculation growth while the input size is getting bigger. But it has a fast query time $O(\log N)$ after learning the Decision Tree.

4) Future Work

For future work, we can use more algorithms to check their behavior on $f=0.6$ are similar to three algorithms that we have coded. The more algorithms that we use, the more precise on our thought. Of course, more data sets can be used to gather more clearer insights. Also, as we initially planned, our thought can be tested on clustering algorithm in addition to classification algorithm. Finally, since our implementation has limitations in that we can only process one-to-one mapping. In other words, we do not assign multiple label. If we extend our feature, the result can be improved, too.

7. Related Works

The performance measure of text classification algorithm is introduced in [8]. In [8], various concept is introduced including F-score. To measure performance of our algorithm, we are using F - score. We also use accuracy measure from [8] to evaluate our algorithms.

There are many variants of Naïve Bayes model. The authors in [3] distinct between the Multinomial Naïve Bayes model - MNB and Naïve Bayes models and they suggest that it provides classification performance better than other variants of Naïve Bayes model. However, a new algorithm has been proposed, called “transformed weight – normalized complement Naïve Bayes” - TWCNB, it is a modified version of MNB [4] with better performance. This algorithm is easy to implement and has good running time. In [10], the authors do experiments with Reuters – 21578 to compare different

algorithms, the micro accuracy 0.739, 0.844, 0.887 and the macro accuracy 0.270, 0.647, 0.694 for MNB, TWCNB and SVM, respectively. So the performance of TWCNB is nearly accurate as SVM. The performance for our Naïve Bayes is not good as performance of those algorithms because we did not do depth pre – process with features (words) in text file. Many words are not useful for classification. Restricting the set of words that are used for classification makes classification more efficient and can improve generalization error. [11] describes how to select best features for text file.

For the Decision Tree, we can improve the accuracy of algorithm by using multiple Decision Trees instead of a single tree. In [5], the authors build good trees based on a modified version of the ID3 algorithm and consider predictions from these good trees, weighted by their probability and average over predictions. Some traditional Decision Tree algorithms, such as C4.5 and ID3, produced poor probability estimation so Charles X. Ling and Robert J. Yan [6] propose new algorithm for improving ranking and the Area Under the Curve - AUC of Receiver Operating Characteristics - ROC, by averaging probability estimates from all leaves of the tree instead of estimating the probability at the single leaf. In some data sets when instances may belong to multiple labels and these classes are organized in a hierarchy, namely hierarchical multi – label dataset. [15] provides several approaches to the induction of Decision Trees for hierarchical multi – label classification.

kNN with $k = 1$ using cosine similarity as distance measure. This Vector Space Model – VSM based classifier has high speed but its accuracy is not good because the Similarity Threshold is decided empirically but not mathematically [16]. In this paper [16], the authors introduces a boosting – based mechanism to adaptively compute out relatively accurate Similarity Threshold over specific dataset. There are several improvement techniques that we can apply to VSM, in [2] they improved TF, TFIDF and BM25 then maximum mutual information feature selection to archive higher precision. In [7], precision of VSM can be improved greatly by using a weight adjustment method in which the IDF function is replaced by scoring function usually used in feature selection. It is obviously that the document title contains much semantic information but it is not taken into special consideration in VSM algorithm. [17] proposes a technique to improve VSM text classification by title vector based document representation, in that weights of terms in document title should be amplified. They do experiment with documents collected from portal sites by the VIPs module in the IR system automatically, the precision and recall of new VSM are 0.872 and 0.832 respectively compare to 0.852 and 0.767 from general VSM. So, we can improve the precision and recall of SVM by several ways.

8. References

- [1] Wiki page, “Decision Tree”. http://en.wikipedia.org/wiki/Decision_tree
- [2] Diao, Lili., et al. "Using boosting mechanism to refine the threshold of VSM-based similarity in text classification." *Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on*. Vol. 3. IEEE, 2002.
- [3] Eyheramendy, Susana, David D. Lewis, and David Madigan. "On the Naive Bayes model for text

categorization." (2003).

- [4] Kibriya, Ashraf M., et al. "Multinomial Naive Bayes for text categorization revisited." *AI 2004: Advances in Artificial Intelligence*. Springer Berlin Heidelberg, 2005. 488-499.
- [5] Kwok, Suk Wah, and Chris Carter. "Multiple Decision Trees." *arXiv preprint arXiv:1304.2363* (2013).
- [6] Ling, Charles X., Yan, Robert J. "Decision Tree with better ranking." (2003).
- [7] Lu, Mingyu, et al. "SECTCS: towards improving VSM and Naive Bayesian classifier." *Systems, Man and Cybernetics, 2002 IEEE International Conference on*. Vol. 5. IEEE, 2002.
- [8] Lewis, David D., et al. "RCV1: A New Benchmark Collection for Text Categorization Research", *Journal of Machine Learning Research* 5, pp.361-397, 2004
- [9] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Vol. 1. Cambridge: Cambridge university press, 2008.
- [10] Rennie, Jason D., et al. "Tackling the poor assumptions of Naive Bayes text classifiers." *ICML*. Vol. 3. 2003.
- [11] Rennie, Jason DM. *Improving multi-class text classification with Naive Bayes*. Diss. Massachusetts Institute of Technology, 2001.
- [12] Padhraic Smyth (Winter 2014) Text Classification [pdf slides]. Retrieved from http://www.ics.uci.edu/~smyth/courses/cs277/public_slides/text_classification.pdf
- [13] Solr Wiki page, "Solr Suggester", <http://wiki.apache.org/solr/Suggester>
- [14] Utgoff, Paul E., Brodley, Carla E. 'An Incremental Method for Finding Multivariate Splits for Decision Trees', *Machine Learning: Proceedings of the Seventh International Conference*, (pp.58). 1990. Palo Alto, CA: Morgan Kaufmann
- [15] Vens, Celine, et al. "Decision Trees for hierarchical multi-label classification." *Machine Learning* 73.2 (2008): 185-214.
- [16] Yang, Zhen, et al. "Improved VSM for incremental text classification." (2008): 369-373.
- [17] Xia, Tian, and Yi Du. "Improve VSM text classification by title vector based document representation method." *Computer Science & Education (ICCSE), 2011 6th International Conference on*. IEEE, 2011.