**Progress Report #2 for Comparing performance of Text classification algorithms on various fraction of sequential text of each document in the data sets**

By Yen Hoang, Anbang Xu, Taewoo Kim

## 1. Progress Summary [Anbang]

Before submitting the previous Progress Report 1, we analyzed the Reuters data set and pre-processed the data set to create the bag of words by removing stop words, applying stemming and tokenizing the documents in the training set. After the submission of that report, we have fully understood three algorithms - Decision Tree, Naive Bayes, Vector Space model using TF-IDF and cosine similarity. We have also implemented these algorithms and executed our code on the Reuters data set. In this progress report, we will discuss the algorithms, results of our code on the data set and about future plans.

## 2. Algorithms [Decision Tree - Anbang, Naive Bayes - Yen, Vector Space Model - Taewoo]

In this section, we are going to discuss about three algorithms and their implementation briefly. For more detail information, refer to the Appendix A.

**a) 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. 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.

**b) 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 word given class. In training step, we estimate probability of each class c, p(c) and each word c given class c, p(w|c). Based on the assumption about conditional independence of words, we can estimate $p(d|c) = \Pi p(w|c)$. 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 by choosing class c* with largest estimated probability, $p(c^*|d) = \max p(c|d)$.

**c) Vector Space Model using TF-IDF and cosine similarity**: 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. We create a vector

for each test file and calculate cosine similarity between two vectors. The result will be used to classify a test file to a category.

### 3. Execution Result on the Reuters Data Set [Decision Tree - Anbang, Naive Bayes - Yen, Vector Space Model - Taewoo]

Here are the execution results for three algorithms. In short, Naive Bayes performs best when regarding correctness. Although the execution time of Vector Space model is smallest among three algorithms, the correctness is much lower than other two algorithms.

**Table 1. Summary of results for three algorithms [DT: Anbang, NB: Yen, VSM: Taewoo]**

| Item | **Decision Tree** | **Naive Bayes** | **Vector Space Model** |
|---|---|---|---|
| Execution Time (sec) - Overall | 601.4 | 63.0 | **3.28** |
| - Training the Data Set | 596.2 | 39.4 | 0.79 |
| - Testing the Data Set | 5.2 | 23.6 | 2.49 |
| Number of Unique Test Files as Input | 3,019 | 3,019 | 3,019 |
| Number of Test Files categorized correctly | 2,235 | 2,603 | 1,822 |
| Correctness (Number of correct results / Number of all unique Test Files) | 74.0% | **86.2%** | 60.4% |
| Extra Information | Criterion='gini'. Depth=no limit. Features=no limit. Stored prediction class and its probability in the leaf node | The result is based on value of $\alpha = 0.08$. Execution time is time for both training and test phase | Execution Time include time for converting a vectors to unit vector to expedite the process. |

As we can see from the Table 2, since about 15% of test files belong to multiple categories ranging between 2 and 14, we created a rule to measure correctness of our algorithms. If a test file belongs to multiple categories and our code categorized it as one of multiple categories, then we regard it as a correct result. For example, if a test file $T_i$ belongs to category 'A', 'B', 'C', and Naive Bayes

categorized the $T_i$ as 'B', then that is the correct result. If the test file $T_i$ is categorized as category 'D', then the result is not correct.

**Table 2. Number of Categories where test files belong [Taewoo]**

| Number of Category that a file belongs | Number of Unique Files in this range | Number of Real Files (duplication counted) |
|---|---|---|
| 1 | 2,583 (85.6%) | 2,583 (69.0%) |
| 2~14 | 436 (14.4%) | 1,161 (31.0%) |
| Total | 3,019 (100%) | 3,744 (100%) |

From now on, we are going to analyze why our algorithms have the given results.

**a) Decision Tree Analysis [Anbang]**

(1) As we mentioned in the "Algorithm description", the most important part of the decision tree algorithm is how to pick 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**. The Gini metric tells us the probability of a random datum being mislabelled if we assign labels according to the probability vector. Entropy tells us, given that we know a data point is in the node, what is our expected information gain (in bits). 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). For example, suppose that we want to 3 discretized values. "said" is a feature in our input data and it's frequencies are - 10, 15, 20, 21, 26, 45, 88. Then, we simply discretize it to 3 ranges, [0 - 30), [30 - 60), [60, 90). Compared with the previous one, discretizing can save a lot of time when finding the best pivot.

(2) After training, the data structure of the decision tree will look like this: (text output of a simple example)

**Figure 1. Text Output of the Decision Tree [Anbang]**

```
depth:0 split_attribute: countri frequence_pivot: 0.0
depth:1 split_attribute: decis frequence_pivot: 0.0
depth:1 split_attribute: adapt frequence_pivot: 0.0
depth:2 split_attribute: four frequence_pivot: 0.0
depth:2 split_attribute: confus frequence_pivot: 0.0
depth:2 split_attribute: deterior frequence_pivot: 0.0
depth:2 prediction: {'soybean': 1.0}
depth:3 prediction: {'copper': 0.006024096385542169, 'gold': 0.018072289156626505, 'money-fx': 0.048192771084337352, 'ipi': 0.012048192771084338, 'trade': 0.03614457831325301, 'reserves': 0.006024096385542169, 'soybean': 0.030120481927710843, 'ship': 0.018072289156626505, 'alum': 0.006024096385542169, 'sugar': 0.012048192771084338, 'rubber': 0.006024096385542169, 'veg-oil': 0.012048192771084338, 'interest': 0.018072289156626505, 'crude': 0.048192771084337352, 'jobs': 0.006024096385542169, 'carcass': 0.006024096385542169, 'gas': 0.006024096385542169, 'gnp': 0.006024096385542169, 'earn': 0.40361445783132532, 'stg': 0.006024096385542169, 'money-supply': 0.018072289156626505, 'acq': 0.18072289156626506, 'grain': 0.084337349397590355, 'retail': 0.006024096385542169}
depth:3 prediction: {'earn': 0.23529411764705882, 'money-fx': 0.23529411764705882, 'nat-gas': 0.058823529411764705, 'sugar': 0.058823529411764705, 'grain': 0.17647058823529413, 'crude': 0.11764705882352941, 'ship': 0.11764705882352941}
depth:3 prediction: {'crude': 0.5, 'gnp': 0.5}
depth:3 prediction: {'soybean': 1.0}
depth:3 prediction: {'stg': 1.0}
depth:3 prediction: {'crude': 1.0}
~
```

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.

(3) 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.

**Figure 2. Depth vs Correctness [Anbang]**

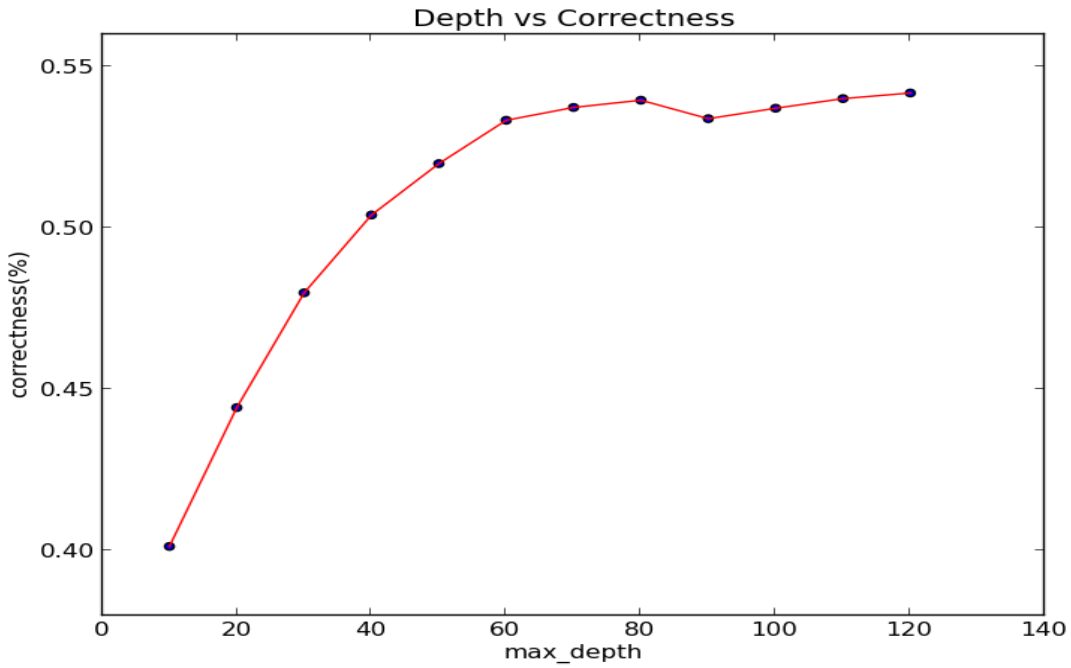
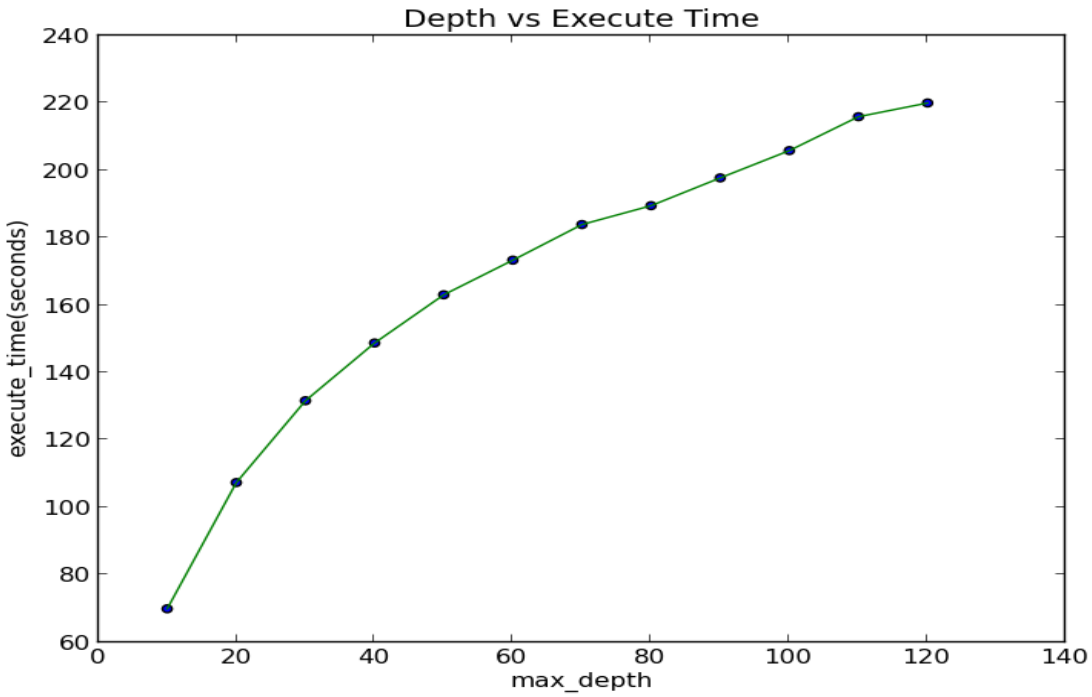



**Figure 3. Depth vs Execution time [Anbang]**




Here, we plot the two diagrams. Figure 2 is to explore how max depth affects the correctness and Figure 3 is to explore how max depth affects the execution time. In Figure 2, we can find that when the max depth reaches 70 - 80, the correctness has reached the maximum value. In Figure 3, we can see that if we set max depth greater than 80, the execution time keeps increasing but the correctness will not increase. Thus, for the decision tree algorithm, we need to figure out the best maximum depth. This will provide us the highest/ideal correctness and reasonable execution time.

**b) Naive Bayes Analysis [Yen]**

To avoid the problem with word ***w*** that never appears in class ***c***, that means p(w|c) = 0, we need to use smoothing parameters to get estimate probability. At the beginning, we use a formula from the lecture [1] $P(x_j = 1|c_k) = (n_{jk} + \alpha)/(n_k + \alpha + \beta)$ with smoothing parameters $\alpha = 1$, $\alpha + \beta = 1000$, $x_j$ to check if word ***j*** is in document, $n_k$ is total number of word in class ***k***, $n_{jk}$ is the frequency of word ***j*** in class ***k***. However, the results were lower. The accuracy of test set was 0.4528 and the accuracy of training set was 0.5290. Then, we tried other smoothing formula from [4] $P(x_j = 1|c_k) = (n_{jk} + \alpha)/(n_k + \alpha * |V|)$ where $|V|$ is size of vocabulary from training data set and $\alpha$ is smoothing parameter. With this formula, we tried different values of $\alpha$ ranging from 1 to 10 with step 1. As indicated in Figure 4, the accuracy of both training set and test set were best with $\alpha = 1$, the test accuracy was 0.82212 and the training set was 0.89013. By trying with $\alpha = 100$, we had the test accuracy 0.6505465 and training accuracy

0.800205. Therefore, we have realized that the accuracy decreases when $\alpha$ increases. One natural question after this step can be: "do the accuracies increase forever when we decrease $\alpha$ such that it still be greater than zero?" To answer this question, we tried with small value of $\alpha$ from 0.01 to 1 with step = 0.01. As we can see in Figure 5, maximum of test accuracy can be given as 0.862206 with $\alpha = 0.08$. So, we have chosen $\alpha = 0.08$ in the experiments with our implementation.

**Figure 4. Dependencies of accuracies with values of alpha [Yen]**

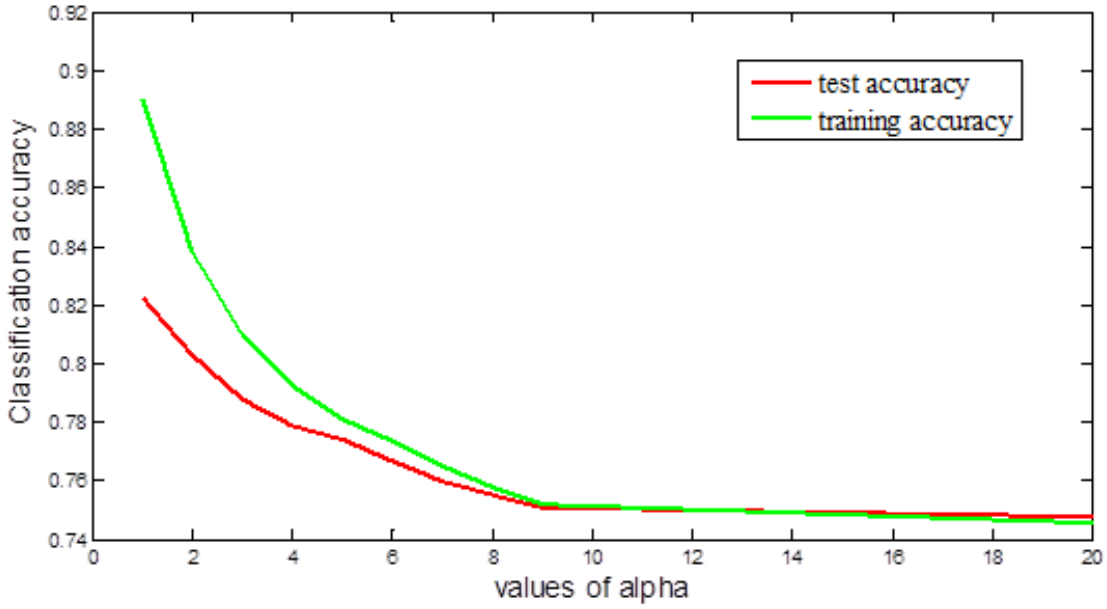
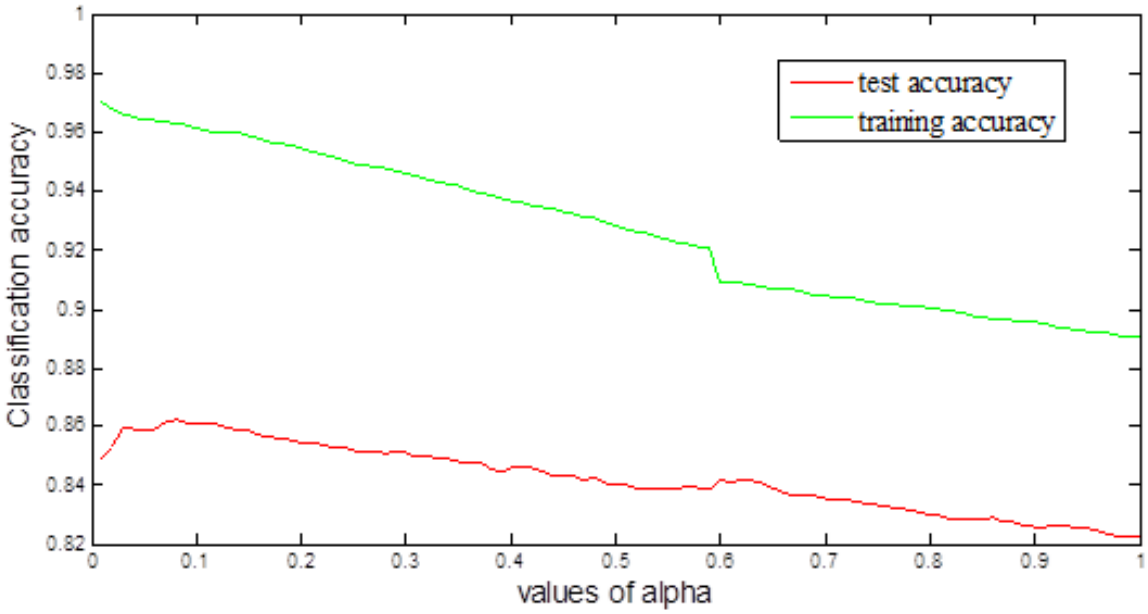



**Figure 5. Dependencies of accuracies with small values of alpha [Yen]**




**c) Vector Space Model Analysis [Taewoo]**

Performance of Vector Space Model on the Reuters data set was lower than other two algorithms. This is because the assumption of this algorithm is the frequency distribution of each vector should be similar in order to get proper text classification result. However, in the training set, for each category, on average, 83 files exist. The distribution of frequency in one category can be different from each file. Furthermore, there is no guarantee that test file is carefully chosen best to reflect this frequency distribution of training categories. If we check following table, we can see that the rate of incorrect result for test files is similar to the rate of training files in the given categories for the total number of files.

**Table 3. The percentage of test files incorrectly categorized as given category [Taewoo]**

| Category | Percentage of Test Files which i incorrectly Categorized as le Category | Percentage of Test Files in the le category |
|---|---|---|
| acq | 14.2% | 17.1% |
| interest | 6.2% | 3.6% |
| earn | 4.0% | 29.8% |

| | | |
|---|---|---|
| copper | 3.4% | 0.5% |
| nat-gas | 3.3% | 0.8% |
| trade | 3.2% | 3.8% |
| crude | 3.0% | 4.0% |
| gnp | 2.5% | 1.0% |
| cpi | 2.3% | 0.7% |
| corn | 2.2% | 1.9% |
| money-supply | 2.0% | 1.4% |

Also, there are some cases where categories are related to each other. For example, test file '0011323' was categorized as 'money-supply'. The correct category is 'interest' or 'money-fx'. However, these three categories are related with each other. These three categories includes similar words. Therefore, there are high possibilities that if the frequency distribution of test file is different from the training category, it can be categorized as related category like this case. Another example is '0011445' file. It was categorized as 'interest'. However, the correct category is 'money-fx' or 'yen'. These categories are related, too. In order to boost correctness of this algorithm, more step to refine the result will be needed such as including the test file when calculating IDF score. However, it will take more execution time since we need to calculate IDF score again for every time new test file is given as an input.

**4. Future Plan [Yen]**

First, we will execute other reference code from external libraries such as scikit-learn to compare with our results if some reference packages exist. By doing that, we can evaluate correctness of our code. Second, we will begin to evaluate the impact of various fraction of sequential text extracted from original document (article) when it is used as input instead of each original document by using fraction number parameter which ranges 10% to 100% to extract some fraction of sequential text in the original document. For example, we will extract first 10% of original text from article and apply our algorithm and compare that results with the result when we use original article as a whole. Finally, if time permits, we will use other data set to compare results between data sets.

**5. Reference [Anbang, Yen, Taewoo]**

[1] Padhraic Smyth (Winter 2014) Text Classification [pdf slides]. Retrieved from http://www.ics.uci.edu/~smyth/courses/cs277/public_slides/text_classification.pdf

[2] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze.*Introduction to information retrieval*. Vol. 1. Cambridge: Cambridge university press, 2008.

[3] Wiki page, "Decision tree". http://en.wikipedia.org/wiki/Decision_tree

[4] Paul E. Utgoff and Carla E. Brodley, (1990). 'An Incremental Method for Finding Multivariate Splits for Decision Trees', Machine Learning: Proceedings of the Seventh International Conference, (pp.58). Palo Alto, CA: Morgan Kaufmann

**Appendix A. Algorithm description and Our Implementation [DT: Anbang, NB: Yen, VSM: Taewoo]**

**a) Decision Tree**

i. Description

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[3]. According to [4], it is commonly used for gaining information for the purpose of decision-making. It starts with a root node on which it is for users to take actions. From this node, users split each node recursively according to decision tree learning algorithm. The final result is a decision tree in which each branch represents a possible scenario of decision and its outcome. Here learning function is represented by a decision tree. It classifies instances by traversing from root node to leaf node, testing the attribute specified by each path node, then moving down the tree branch according to the attribute value in the given set. This process is the repeated at the sub-tree level.

ii. Our Implementation

- Training step

1) load learning sets first, create decision tree node "root", and add learning set S into root node as its subset

2) for root, we compute Entropy[3](root.subset) first

3) if Entropy(root.subset) == 0, then root.subset consists of records all with the same value for the categorical attribute, return a leaf node with decision attribute and decision pivot

4) else, then compute information gain for each attribute left(have not been used in splitting), find attribute A and its best pivot P with Maximum(IG(S, A, P)). Create child nodes of this root and add it to root in the decision tree

5) for each child of the root, apply this algorithm recursively until reaching node that has entropy = 0 or reach leaf node

- Testing step

1) start from the root, based on the prediction on each node, go to the branch the node predicts until reach the leaf

2) when reaching the leaf, the prediction maybe only one category and then this category should be prediction class. Otherwise(many categories), pick the prediction class which has highest probability is the final prediction class.

3) compare the prediction class of each document with the real class in the test test, compute the rate of correctness

**b) Naive Bayes**

i. Description

Naïve Bayes classification is a method based on Bayes rule. For document d and class c, we have p(d|c)*p(c) = p(c|d)*p(d) with p(d|c) is probability of document d given class c, p(c) is probability of class c, p(c|d) is probability of class c given d and p(d) is probability of document d. In this method, we have two assumptions: bag of words assumption, in that position of word does not matter and conditional independence, it means probabilities of words in document given class c, p(w|c) are independent given class c. So we can estimate p(d|c) = ∏p(w|c), with w is word in document d. From the training phase, we can estimate p(c) by ratio of number of documents in class c and number of documents in data set. Whereas, p(w|c) is estimated by ratio of number of word w in class c and number of all words in class c. In test phase of Naïve Bayes, we try to estimate p(c|d) for all class c and assign d into class c* such as p(c*|d) is maximum value of p(c|d). We have p(c|d) = p(d|c)*p(c)/p(d) = ∏p(w|c)*p(c)/p(d), so to compare p(c|d), we just need to compare ∏p(w|c)*p(c). It can be estimated from parameters that we got from training phase.

ii. Our Implementation

- Training step

1) Load training set, create vocabulary

2) Compute p(c) for each class c and store in prob_category: { 'acq':0.015, 'alum':0.031 ...}

3) Compute p(w|c) for each word w in vocabulary and each class c and store in prob_conditional {'livestock':{'stock': 0.001116, 'url': 1.5417e-0.5, 'ryoka': 1.5417e-0.5, ...}, ...}

We estimate p(w|c) by formula $p(w|c) = (n_{wc} + \alpha)/(n_c + \alpha * |V|)$ with $n_{wc}$ is frequency of word w in class c and $n_c$ is total number of all words in class c. We need to use smoothing parameter $\alpha$ to avoid the case p(w|c) = 0 when we can't find word w in class c. Also to avoid underflow we compare p(c|d) by converting multiplication into addition by using log: log(∏p(w|c)*p(c)) = log(p(c)) + ∑log(p(w|c))

- Test step

for each document d in data set

1) extract set of words w from d and store in list W

2) for each class c, we estimate log(p(c|d)) (and store in dictionary score) by formula score(c, d) = log(p(c)) + ∑log(p(w|c))

3) compare score(c, d) and assign d into class c* with score(c*, d) = max(score(c, d))

**c) Vector Space Model using TF-IDF and cosine similarity**

i. Description

In this model, we create a vector for each document by using bag of words represented in TF-IDF value and calculate cosine similarity between these two vectors to apply text classification.

ii. Our Implementation

- Training Step

1) For each category, gather tokens and their frequencies from each file. Then, we will have a vector $\boldsymbol{V_i}$ {frequency of term1 in the category, frequency of term2 in the category… } for each category ***i***.

2) Normalize each vector $\boldsymbol{V_i}$ by dividing frequency of each term by sum of term frequencies in the vector for category ***i***. This process will ensure that length of document will not be considered. For example, if we have {t1:10, t2:30}, then we will have {t1:0.25, t2:0.75} after length-normalization.

3) Create a list of terms ***T*** across all categories. For each term **t** in ***T***, create IDF score for **t** by using this equation: 1 + log(Number of All categories / Number of category which has **t** in it).

4) For each term **t** in the length-normalized vector $\mathbf{V}_i$, calculate TF-IDF score by multiplying it's normalized TF and IDF score for that term **t**. Repeat this process for each vector. Then we will have a length-normalized TF-IDF vector $\mathbf{V}_i$ for each category.

5) To make calculation of the cosine similarity simpler, make vector $\mathbf{V}_i$ as a unit vector in that length of each vector $\mathbf{V}_i$ will be 1.

- Test Step

1) For every test file, identify tokens and it's frequency. After this process, we will have a vector $V_{Testi}$ {frequency of term1 in the test file, frequency of term2 in the test file ...} for each test file ***i***.

2) Normalize vector $V_{Testi}$ by using sum of term frequencies in that vector. You can apply IDF of each term ***t*** in the training set to this vector to create a TF-IDF vector.

3) As what we did in the training step, to make calculation of the cosine similarity simpler, make vector $\mathbf{V}_{Testi}$ as a unit vector in that length of each vector $V_{Testi}$ will be 1.

4) For each vector created in the training step, calculate cosine similarity between the vector which represents a test file and trained categories vector by this equation: $cos\Theta = |V_{testi}| \cdot |V_i| =$ [tf-idf of t1 in the $V_{Testi}$ * tf-idf of that term in the $V_i$ + … tf-idf of $t_n$ in the $V_{Testi}$ * tf-idf of that term in the $V_i$]. Pick a vector where $cos\Theta$ score is highest. Test file $V_{Testi}$ is categorized as the category where $V_i$ represents.