

Exploration on Approaches To Email (text) Classification

CS350 Project

Xuehai Zhang

Dept of CS, Univ. of Chicago
hai@cs.uchicago.edu

1. Abstract

Basic theory about text categorization and information retrieval is presented and several important algorithms for text classification are describe in details, such as the Rocchio Algorithm, TFIDF classifiers and Naïve Byes Algorithm, etc. An implementation based on Rocchio Algorithm is also discussed and evaluated. It shows that this method is reasonably efficient given fairly small training datasets. However, in order to improve the performance of text classification algorithms and construct better ones, we should take into extended feature selection such as word sequences into consideration.

1.1 Keywords

Email classification, categorization, and information retrieval.

2.Introduction

Perhaps the most-discussed phenomenon of recent years has been the rapid growth of the Internet – or more generally, the rapid growth in the number of the on line documents. This led to increased interest in intelligent methods for filtering and categorizing documents. Emails are a good example. At current time, people rely heavily upon emails to communicate with each other, so nearly everybody’s mailbox is full all the time. It is a great deal of interest in systems that allows a technically naïve user to easily construct a personalized system for filtering and classifying them. It is a good research area with many application potentials.

This paper will describe and compare methods for different learning text classifiers, focusing on the kinds of classification problems that might arise in filtering and filing personal email messages. Here we use “text” stands for email in the following parts, since we deem emails is same as text in

nature. The rules or classifiers used to perform these tasks are often trained on data rather than, or subsequent to, being constructed by hand.

The motivation and goal to perform such a study is to explore in details how “traditional” text categorization algorithms works; specifically, the implementation details of each algorithms. Thus, it might arouse us further thinking upon better improvement of the existing text classification algorithm.

3. Problem definition: Text Categorization

The text categorization problem is to classify texts (documents, emails etc) into different existing classes. However, the underlying method of how to assign documents to a class is unknown or very difficulty.

We assume that the number of categories is fixed and known and each document is assigned to exactly one of them. So, classification problem is a little more different from clustering problem. I prefer to consider it as a kind of supervised learning problem.

To put it formally, there is a set of classes C and a set of training documents D . What is more, there is a mapping function $T(d) \in C$ which assigns the documents to a class. $T(d)$ is known for the documents in the training set. What we plan to do is to find a model or hypothesis $H(d) \in C$ which identify or approximates $T(d)$. This is based on the information of the training set. Before we touch the algorithm solution, we should try to answer the following questions first.

3.1 Document representation.

For the problem of text categorization, a document is typically a string of characters. However, we should transform them to a representation that is suitable for the learning and classification. As suggested by researchers in the IR community, words seem to work well as features for many classifications. This simplicities the representation of documents from sequence of characters to sequence of words. We even could let documents be represented as a bag of words and each word is deemed as a feature, and in most cases the frequency it occurs in the documents is its values. You may see from the figure below. It is a very common technique in information retrieval and it has been proved to be effective and powerful. You may get an exact example in section 4.2, the TFIDF classifier.

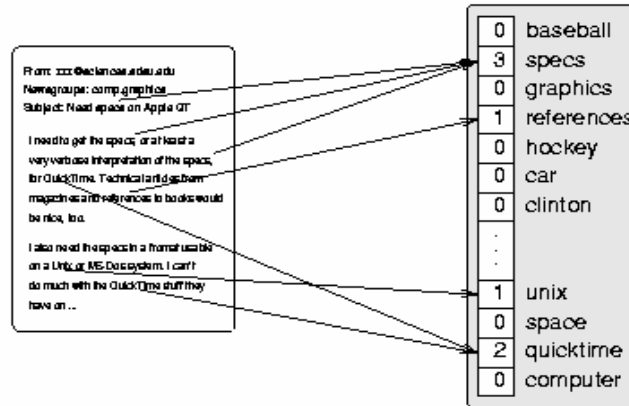


Figure: bag-of-words representation in an attribute value style.

3.2 Feature Selection

In the simple document representation, when we ignore all the additional information and use only words in the described bag-of-approach, we still end up with several tens of thousands of different words that occur in documents. Not only is using all these words time consuming but also many of them are not really important for our learning task and their usage can degrade the whole system performance. So, the methods of selection of the features directly affect the classification performance and results. But keep in mind that having too few features can make it impossible to formulate a good hypothesis, in the other hand, having features which do not help discriminate between classes add noise.

Here are some common approaches to handle the feature selection. I should say they only occupy a small portion of the whole methods we could consider to implement.

One is noisy words (terms), as well as "stop words". These words are those such as "the" or "if" that is so common that it is not useful in searches. In other words, these words do not make a significant contribution to the classification of the emails, so we can just ignore them.

The other consideration is the pruning technique we might imply on the words. We should prune those infrequent words as well as those high frequency words, because those words might be no important use.

There is still one solution we might want to take – word stemming, which reduces number of different words using a language-specific stemming algorithm. As far as I am concerned, this part is a little bit difficult.

3.3 Further thinking: Enriched document representation

By far, to the document representation, we make an assumption that all the words are independent to each other and are of equally importance. We base our algorithms on handling them individually. But this is not the real case. Actually, the words in a specific documents are supposed to be heavily

related with each other, the relation (sequence) among the words also imply the very important information of the nature of this document, thus make a key attribute to the classification of the document. So, it is natural and logical that we consider extending the document representation to a large scale. Here I want to focus on the discussion about enriching the bag-of-words representation by adding new features generated from word sequences (also called trends in the documents) as well as document topic categories. By introducing this thinking, we extend the basic representation item of a document from individual words to words and phrase (sequence of words).

3.3.1 Using the word sequence

In this discussion, we use word sequences, also known as n-grams, trends, to form the new features. We propose the basic implementation of this approach, although I do not add this part into consideration in my own implementation work. We might generate features that represent up to K words (1-grams, 2-grams, ..., K-grams) occurring in a document as a sequence (e.g., "machine learning", "word wide web"). The process of feature generation is performed in passes over documents, where i-grams are generated in the i_{th} pass only from the candidate feature of length i-1 generated in the previous pass. If you are familiar with the concept of Data Mining, this approach is very much like the large k-itemset generation used in association algorithm. We give a pseudo code for the generation of new features. In the 1st pass all single words not contained in the "stop-list" and having sufficient frequency are taken LargeNGramSet. Word sequences of size 2 up to MaxNGramSize are generated using several pruning criteria. I do not want to give many specifics; instead I want to offer something that might give you some direction of the possible improvements upon document representation.

The pseudo code:

```

1.LargeNGramSet = all single words in DocVec, notin StopWordSet and occurring >=
2.MinNGramOcc times;
3.for NgramSize =2 to MaxNGramSize do {
4. CandNGramMap ={};
5. for SymVec = DocVec[1] to DocVec[|DocVec|] do{
6. NgramQueue = []
7. for Sym = SymVec[1] to SymVec[|SymVec|] do{
8.     if (TypeOf(Sym) == word){
9.         if (Sym not in StopWordSet)
10.            if Sym in LargeNGramSet then {
11.                if (|NgramQueue| + 1 == NgramSize){
12.                    if (Concatenated(NgramQueue) in LargeNGramSet){
13.                        NgramQueue.Push(Sym);
14.                        CandNGramMap[Concatenated(NgramQueue)]++;
15.                        NgramQueue.pop();
16.                    }else{
17.                        NgramQueue.Push(Sym);
18.                        NgramQueue.Pop();
19.                    }
20.                }else{NgramQueue.Push(Sym);}
21.            }else{NgramQueue = [];}
22.        }else {NgramQueue = [];}
23.    };
24. };
25. LargeNGramSet +={Ngram:CandNGramMap[Ngram]>=MinNGramOcc};

```

```

    }
26.    return LargeGramSet;
27.

```

4. Learning algorithms (classification models)

In this paper, I try to explore several well-known text categorization algorithms, which include The Rocchio Algorithm, Naive Bayesian Classification, Support Vector Machines (SVMs), The Nearest Neighbor Classifier Algorithm. However, I focus my research on the Rocchio Algorithm (has many variances, such as Find Similar Algorithm etc.) and TFIDF classifier. I also give my implementation in C++ code in support of this algorithm. Details about implementation will be found in the following section 5.

4.1 The Rocchio Algorithm

Rocchio first brought out the Rocchio Algorithm in 1971. Although I could not find the exact texts, which has his papers, I do find a lot of description and explanation of the algorithm on the web. Actually, One-reason drives me to focus on this algorithm is the long-time importance it keeps.

The basic idea of Rocchio Algorithm is:

It constructs a weight vector for classification; the new weight vector w is generated from an existing weight vector w_1 and a set of training data. The training data here constrains within the texts. The j -th component w_j in the new weight vector is calculated as:

$$W_j = \alpha * W_{1,j} + \beta * \frac{\sum_{i \in C} x_{i,j}}{n_c} - \gamma * \frac{\sum_{i \notin C} x_{i,j}}{n - n_c}$$

Where n is the number of the training sets, $c = \{1 \leq i \leq n \text{ and } y_i = 1\}$ is the set of the positive training examples, and n_c is the number of positive tanning examples. We notice the coefficient α , β and γ and control the relative impact of the original weight vector, the positive/negative training datasets. We let $\alpha = 0$, $\beta=1$ and $\gamma=1$, then $w/|w|$ is the weight vector of unit length which maximizes

$$\frac{\sum_{i \in C} w * x}{n_c} - \frac{\sum_{i \notin C} w * x}{n - n_c}$$

That is, the difference in the mean score for positive and negative training instance.

Typically, classifiers produced with the Rocchio algorithm are restricted to having nonnegative weights, so that instead of using the raw w from above formula, one uses w' where

$$W'_j = \begin{cases} W_j & \text{if } W_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

A novel document will be classified as positive if this distance is less than some threshold T_c , which can be chosen so as to balance recall and precision in some set manner.

4.2 TFIDF Classifier.

TFIDF classifier is a variance of Rocchio Algorithm; it is based on Rocchio relevance feedback algorithm but uses TFIDF word weights. Here TF stands for Term Frequency and IDF stands for Inverse Documents Frequency.

Each documents in this algorithm is represented as a vector $d = (d_1, d_2, \dots, d_{|f|})$, the components of the representation vector is basically related the words that appear in the training corpus. Intuitively the similar documents might have similar vectors. Each dimension of the vector space represents a word.

The values of the vector elements $d(i)$ for a document d are calculated as a combination of the statistics $TF(w,d)$ and $IDF(w)$. As you may guess, $TF(w,d)$ is the number of time word w occurs in document d . $DF(w)$ is the number of documents in which the word w occurs at least once. $IDF(w)$ is calculated in this way given the $DF(w)$.

$$IDF(w) = \log \frac{|D|}{DF(w)}$$

Here $|D|$ = number of Documents in collection. From the formula above, we may conclude that the IDF of a word is low if it occurs in many documents and is highest if the word occurs in only one. So, the value $d(i)$ of feature w_i for document d is then calculated as the product

$$d^{(i)} = TF(w_i,d) * IDF(w_i)$$

Here $d^{(i)}$ = weight of Term w_i in Document d . This word weighting heuristic says the word w_i is an important indexing term for document d if it occurs frequently in it. On the other hand, words which occur in many documents are rated less important indexing terms due to their low IDF.

The TFIDF algorithm learns a class model by combining document vectors into a prototype vector c for every class C belongs to C .

$$c = \sum_{d \in C} d$$

This model can be used to classify a new document d' . Again the document is represented as a vector d' using the model we state above. We calculate the cosine of the prototype vector of each class with d' is calculated. The new document is assigned to the class with which its documents vector has the highest cosine.

$$H_{tfidf}(d') = \operatorname{argmax} \cos(d',c).$$

4.3 Naive Bayes Classifier

Bayes algorithm is another famous solution to text classification problem, it is based on a probabilistic model.

Before I state bayes algorithm, we should make some assumption: Documents are generated by drawing words from a probability distribution. All documents assigned to a particular class are generated from the probability distribution associated with this class in a number of independent trials. The i _th word of the document is generated by the i _th independent trial.

This algorithm says that to achieve the highest classification accuracy, d' should be assigned to the class for which $\Pr(C|d')$ is highest. We uses Bayes theorem to estimate the probabilities.

$$\Pr(C | d') = \frac{\Pr(d'| C) * \Pr(C)}{\sum_{c \in C} \Pr(d'| C) * \Pr(C)}$$

$\Pr(C)$, is the prior probability that a document is in class C . $\Pr(d'|C)$ is the likelihood of observing document d 's in the given class.

$\hat{\Pr}(C)$, the estimate of $\Pr(C)$, can be calculated from the fraction of the training documents that is assigned to this class:

$$\hat{\Pr}(C) = \frac{|C|}{\sum_{c' \in C} |C'|}$$

$|C|$ is the number of training documents in a class.

Actually, the estimation of $\Pr(d'|C)$ is more difficult. $\Pr(d'|C)$ is the probability of observing a document like d' in class C . Since there is a huge number of training examples to estimate this probability without prior knowledge. The assumption implies that a word's occurrence is dependent on the class the document comes from, but that it occurs independently of the other words in the document. So $\Pr(d'|C)$ can be written as :

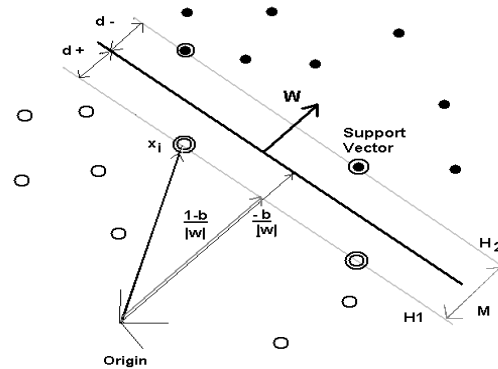
$$\Pr(d'| C) = \prod_{i=1}^{|d'|} \Pr(W_i | C)$$

w_i ranges from the sequence of words in document d' which are element of the feature vector F . A Bayesian estimate is used for

$$\hat{\Pr}(W_i | C) = \frac{1 + TF(W_i, C)}{|F| + \sum_{w' \in |F|} TF(W', C')}$$

4.4 Support Vector Machines(SVMs)

SVMs solution has only recently been gaining popularity in the learning community.



In its simplest linear form, an SVM is a hyper plane that separates a set of positive examples from a set of negative examples with maximum margin. See the above figure.

The formula for the output of a linear SVM is $u = w \cdot x - b$, where w is the normal vector to hyper plane, and x is the input vector.

In the linear case, the margin is defined by the distance of the hyper plane to the nearest of the positive and negative examples. Maximizing the margin can be expressed as an optimization

problem: minimize $\frac{1}{X |W|^2}$ subject to $y_i(w \cdot x_i - b) > 1$,

Where x_i is the i _th training example and y_i is the correct output of the SVM for the i _th training example.

5. Implementation

Having talked about so many algorithms so far, I want to talk about my experiment based on the Rocchio Algorithm and TFIDF weight strategy.

5.1 MailParser Project

Initially I want to find some existing software, which may help me do the document representation job, as well as the calculation of the term frequency and inverse document frequency. There did exist some software on the web that help to automatically index documents, but most them are used for information retrieval, I means, information query, so the indexing process is concealed in most cases. After I failed tried the following tools, such as dtsearch software, Microsoft IIS/Index Server, etc, I decide to write it by my own. The C++ code is mainly made up of three important classes: class *Cdivider*, class *CwordList*, and class *Cparser*.

Cdivider class handles with the extraction of all the emails from one single .txt file. I collect around 400 emails from the newsgroups on the web, basically they are divided into three general classes: topic about dogs, topic about java programming and topic about titanic. Each of the three topic has roughly over 100 relating emails. From the Rocchio Algorithm, part of the emails are used as the training data and based on these training examples, I construct the vector model for then three classes, then I use the left emails as the testing data to test the model I constructed.

Cparser class is the main class of the project, it generates the words vectors for all the training emails (for e.g., there is a word statistics file `dog_email1_ws.txt` corresponding to original email `dog_email1.txt`); it creates and prunes the total words list file (`total_words.txt`); it then translates the training datasets to the uniform input format (the resulting files are: `traindata_java_email.txt`, `traindata_dog_email.txt`, and `traindata_titanic_email.txt`); it generates the weights vector for the three classes(the resulting files are: `weight_java.txt`, `weight_dog.txt`, and `weight_titanic.txt`); it then use the model we calculated so far to classify the remaining testing data(the resulting file is: `testdata_result.txt`). Here `testdata_result.txt` will offer the information of the accuracy of the model we build.

CwordList class is an important supporting class, which handles with the functions of vector words, such as frequency calculation, word stemming, etc.

As to the dataset, at first I want to use the benchmark dataset provided by Reuters data set source. But I am not very familiar with the format and how to extract the desired content I want, so I give it up last.

5.2 MailParser Model Evaluation

As I said, my constructed model is based on the Rocchio Algorithm. During the implementation of this algorithm, I found there are some parts that we can take different strategy into consideration. First one is the feature-weighting vector. In my implementation, I use TFIDF weighting method. But there are a lot of different weighting strategies that might be also taken into consideration; another part lies in the calculation of the distance between the test data vector and the fixed feature vector. In my implementation I use Euclid proximities measure, but other researchers also recommend some other measures such as Cosine, Jaccard Similarity Measure.

Due to the computer resource constrains, I cannot test my code upon a large scale training data and testing data. You might have noticed, the total number of the emails of the three classes is up to only 400, which mean each class has about 100 emails. I test the code based on the following experiments:

Experiment Round	Training datasets size (equally distributed)	Testing datasets size (equally distributed)	Classification accuracy for the testing dataset
1	20*3	10*3	43.56%
2	40*3	10*3	62.33%
3	70*3	20*3	71.25%
4	90*3	20*3	76.47%

From the above results, we conclude that basically the model performs not that good as I expected. There might be many reasons to make this happen: First, I don't compare different strategies as to the parts of the models I discuss before. It is possible that there exist better solutions; Second reason may come from the dataset I used, since the sizes of both the training sets and testing sets are relatively very small, so the testing result may be affected badly by the constitution of the training datasets. So, to better this model, more work is need to dealing with different strategies and large scale training datasets.

6. Future Works

As I state in section 3.3.1, we might imply certain data mining techniques into the construction of better model for text categorization. Indeed some researchers have tried to explore such a field. I am very interested in such fields.

By far, the algorithms I states do not implement the extended feature selection. But I think it will buy us a lot of the performance of the model we want to build. But extended feature selection is a little bit hard to implement, wiser algorithm may be needed for this purpose.

7. Conclusion

Text categorization as well as other IR methods is being applied to an increasingly broad range of problems, and by implementers who are less experienced with IR systems. Motivated by such observation, we describe several important methods for text categorization problems. We implement the Rocchio algorithm and give the evaluation of the model we build, which shows reasonable performance of the learning.

8. References

Fuhr, N. (1989). Models for retrieval with probabilistic indexing. *Information Processing & Management*, 25 (1), 55-72.

Hayes, P.J. (1992). Intelligent high-volume text processing using shallow, domain-specific techniques. In P.S. Jacobs (Ed.), *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval* (pp. 227-241). Hillsdale: Lawrence Erlbaum.

Jones, W.P., & Furnas, G.W. (1987). Pictures of relevance: a geometric analysis of similarity measures. *Journal of the American Society for Information Science*, 38 (6), 420-442.

Lewis, D.D. (1995). Evaluating and optimizing autonomous text classification systems. In E. A. Fox, P. Ingwersen, & R. Fidel (Eds.), *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 246-254). New York: ACM.

Lewis, D.D., Schapire, R.E., Callan, J.P., & Papka, R. (1996). Training algorithms for linear text classifiers. In H.-P. Frei, D. Harman, P. Schäuble, & R. Wilkinson (Eds.), *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 298-306). New York: ACM.

Maron, M. (1961). Automatic indexing: an experimental inquiry. *Journal of the ACM*, 8, 404-417.

Moens, M.-F. (in press). *Automatic Indexing and Abstracting of Document Texts* (270 pp.). Boston: Kluwer Academic Publishers.

Moens, M.-F., & Dumortier, J. (in press). Automatic categorization of magazine articles. *Information Processing & Management*.

Moens, M.-F., & Uyttendaele, C. (1997). Automatic structuring and categorization as a first step in summarizing legal cases. *Information Processing & Management*, 33(6), 727-737.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), *The SMART retrieval system: Experiments in automatic document processing* (pp. 313-323). Englewood Cliffs, NJ: Prentice Hall.

Schütze, H., Hull, D. A., & Pedersen, J. O. (1995). A comparison of classifiers and document representations for the routing problem. In E. A. Fox, P. Ingwersen, & R. Fidel (Eds.), *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 229-237). New York: ACM.

Appendix

1. The Source code of MailParser Project.
2. Stop List of Words
3. Total_Words.txt
4. Traindata_java_email.txt