Development of Text Classifier Using naïve Bayes Classifier

Submitted as a Project for Cpt S 540 Artificial Intelligence Course

Submitted By:

Barnan Das

School of Electrical Engineering and Computer Science
Washington State University, Pullman

Project Description
The objective of this project is to develop a text classifer using naïve Bayes classifer. In order to optimize the performance, stop words removal and a rule based Stemmer has been used.

Introduction
Text Classification is a classical problem in the area of Data Mining and Information Retrieval which deals with assigning electronic documents to one or more categories. It has a huge application in classifying web pages by topic, automatic file saving, spam filters, recommenders and information extraction. A number of machine learning approaches, like naïve Bayes Classifier, k Nearest Neighbor, Support Vector Machines, Latent Semantic Indexing, etc., have been considered to make the algorithm learn on a particular pre-classified data set and then take decisions of classification on the basis of the learning. Out of these approaches naïve Bayes Classifier is the most common and renouned one and so in this project I have implemented this approach.

Data Set Used
The 20Newsgroup (http://people.csail.mit.edu/jrennie/20Newsgroups/) data set has been used for this project. It is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. For its variety of topics, it has become a popular data set for experiments in text classification. I have used the "bydate" version which contains precisely 18846 documents sorted by date into training(60%) and test(40%) sets, does not include cross-posts (duplicates) and newsgroup-identifying headers (Xref, Newsgroups, Path, Followup-To, Date).

Algorithms
I have not explicitly included the algorithm in this report as it is an implemented of the same algorithm proposed in page 183 of Machine Learning, Tom Mitchell, McGraw Hill, 1997, with a few additions.

While considering every word, it was first checked whether the word is a stop word. If it is a stop word, it is ignored and the next word is considered. If a word is not a stop word, then it is checked whether the word is an inflected form of a base word. If it is so, the base word is considered instead of the inflicted word.

By using these techniques it was not only possible to reduce the size of the vocabulary, but also assigning the probability to the root words which are in higher frequency and hence the probability of such words would also be higher.

Note that the following assupmtions were used while developing this project:
1. A document can belong to only one class and not multiple classes.
2. Classification is independent of position of the words.

Stopwords Removal
Stop words are high frequency words in any language. As they are very common words, they have no significance in text classification. I have used a stop words list available at this link: http://web.mit.edu/course/6/6.863/share/data/corpora/stopwords/english

Stemmer
Stemming is a process of reducing inflicted words to their stem or root form. e.g. the stem for the word “stemming” is “stem”. A quite large number of stemmers have been developed so far. But in order to keep my algorithm simple I have used a rule based stemmer. I have used a list of suffixes in english and have just stripped them off. The list of suffixes are in a text file. Each line in the file is in the format of this example:

```
*ed, *
```

This rule indicates that any word that ends with “ed” should be stripped off with the suffix in order to obtain the root word.

Experimental Results

Refer to the attached output file.

The output displays the results in a tabulated format, where each row represents the result for each topic category in the order the training was conducted.

A sample run was conducted by training the algorithm on 400 documents (20 documents each from the 20 topics) and testing on 120 documents (6 from each of the 20 topics). This execution took 1158.614 seconds to complete.

The test was conducted only on this sample dataset because it was taking a long time to do so with the entire data set(as it can be seen, it took 1158.614 seconds for this minute fraction of the data set).

As this is only a minute fraction of the whole data set, I couldn't get the desired results. It obtained a 66.67% recall and a negligible Precision. The accuracy for most of the classes was quite high because this classifier essentially acts as 20 binary classifiers, deciding whether the document belongs to class x or not.

Discussion

In a multiclass text classification problem, accuracy is not the only parameter for performance measurement of the classifier. But, other parameters like Precision, Recall and F1 Measure are more important. That is why they have been included for the results.

While experimenting it was found that the program was taking more than 10 hours to train and test on the entire data set and I had to kill the process when it took so long. The reason for this is: the program reads from nearly 20,000 files which creates an overhead. The data stuctures used were mostly linked linked and array of linked list which were overwhelming due to the mammoth size of the vocabulary. Also, some of the portions of the code has $O(n^3)$ run time complexity, which made it really slow.

Future Work

There is a huge prospect of improvement for this project. Use of better data structures like hash tables can reduce the run time complexity a lot. Better stemmers could be used to do more efficient stemming. The probability of the inflected words can be considered in addition to the stem words. Position of words is also a vital area to look into.