Automatic Text Classification using a Naive Bayes Classifier

Prajwal Halasahally KeshavaReddy
pxh140930@utdallas.edu

Abstract: Automated text classifiers are becoming more important with the exponential growth of data being generated every day. The Naive Bayesian learning algorithm has proven to be a good choice in this area. This document talks about the implementation of an automatic text classifier with Naive Bayesian learning. The accuracy of the text classification is also being calculated and being recorded. Implementation has been done using Java.

Introduction: Text classification is a machine learning task aimed at automating the assignment of classes, from a set of predefined classes, to an unknown text. Bayesian methods have a good reputation for working well in this area. The approach has been a rather straight-forward version of text classifier, based on naive Bayesian learning, described in Mitchell's book Machine Learning.
The method can be said to be language-independent. The accuracy of the algorithm has been increased by removal of stop words and performing stemming.

Objective: Given a document, classify the document based on the training data.
A few numbers,
Total Number of Classes to be trained: 20
Total Number of Documents: 11314 (20 Classes)
Number of Documents in the test set that are to be classified: 2000
Initial Program Runtime: 4 Minutes Approx. (Initially) After the training set is built, a few seconds

Bayesian Learning:
Bayesian learning is a probability-driven learning algorithm, based on Bayes theorem. The Bayes theorem is given by: $p(A|B) = p(A)p(B|A)/p(B)$.
This is an interesting theory that states that we can calculate the probability of A given B based on the reverse relation. The Bayes theorem, stated in terms of this task, would be the following.

$$p(\text{Class}|\text{Document}) = p(\text{Class})p(\text{Document}|\text{Class})/p(\text{Document})$$

This can be simplified by noticing that p(Document) is a constant divider common to every calculation, and can thus be disregarded. The probability of a document, p(Document), is further simplified by only considering the words in the document (e.g. whitespaces and special characters are disregarded), and more, that the position of the words are not accounted for.

We will only consider a specific set of words, called the vocabulary, a distinct set of words. The size of the vocabulary must be carefully chosen; too many words will introduce difficulties due to space limitations, but if the words are too few, the classifier will work poorly. The words not represented in the vocabulary will be disregarded in the classification.

Learning Approach:

a. All texts making up each class are collected, i.e., each class is formed as if it simply was constituted of one text file.
b. A frequency list is compiled, and certain parts of the word entries are collected. In this step we also take care of removing the stop words and stemming each word. With stemming we do see an increase in computational time but there is a trade-off for increase in accuracy. We make use of an open source stemmer implementation.
c. The probability of each category, p(C), is computed. This means that the likelihood of each category showing up in random text is to estimated. This is the prior probability.
d. For each category: for each word appearing in the vocabulary, the conditional probability is estimated by the following formula:

$$1+c_w / c_c \text{ at}+ c_v \text{ oc}$$

Where c_w is the number of times the word occurs in the category, c_c at the number of words in the category, and $c_v$oc is the size of the vocabulary

After the prior and the conditional probabilities have been calculated, the individual probabilities for each of the text document is calculated and the document given a class with the highest probability is chosen as the final class.

Implementation of the classifier in Java:

The program consists of two modules, the training & test module and the calculating accuracy module.

Let us first discuss about the Train and Test module.
This module takes 3 input parameters via the command line arguments; the Train Folder, Test Folder and the path to generate the output file.

Training:
The training folder consists of several folders namely the class folder and inside them the respective documents. We traverse the document folder wise, record the folder name and read the individual file in each folder.
Stop words are eliminated. A standard stop word removal file has been used.
Stemming is also performed on each word being read, an open source implementation of a stemmer has been made use of.
The words are added to a hash map and the counter for each word is also stored as a value in the hash map. Other details such as the vocabulary size and the total class word count is also recorded. So the final has map would contain the class name as the key and words with their respective counts as the values. We consider each class containing a huge text file by merging all the contents and finding the cumulative count of words.

A symbolic representation is shown below:
{class_name1= {word1=10, word2=20}, class_name2= {word1=3, word2=8}…….}

The prior probabilities for each classes are computed for and stored, i.e., given any document what is the likelihood that the document would belong to that class

$$\hat{P}(c) = \frac{N_c}{N}$$

The conditional probability for each word is calculated for and stored in another HashMap.

$$\hat{P}(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

Testing:
The test folder consists of the 2000 individual files.
Each file is read individually, and the class label for each file is generated.
As done while reading the train data, the test data is also stripped of stop words and undergoes stemming operations. The word count is stored in a hash map.
A testCount HashMap is of the following format:

{test_document_name1= {word1=10, word2=20}, test_document_name1= {word1=3, word2=8}…….}

After this HashMap has been generated we head towards finding the max probability likelihoods of the test data.

We observe that during the procedure of calculating the max probability the values would tend to infinity. Hence we calculate the Max Log Likelihood for each class.

After the class with the max likelihood is determined, an output file is generated with the test document name and its respective classification. The output document is generated in the following format:
Doc_Name1.txt 0
Doc_Name2.txt 1
Doc_Name3.txt 2

The integer values range from 0-19 and represent the class labels. The integer labels are read from a text file containing the class name and it's representing integer value.
This output document would be used for calculating the accuracy.

Accuracy Calculation:
Along with the dataset we also have a text document containing the actual classifications in the following format:
Doc_Name1.txt 0
Doc_Name2.txt 1
Doc_Name3.txt 2
.
.
.

This modules takes 2 inputs, the output file that is generated by the classifier and the dev_label file which contains the actual classifications. Both the files are read and stored in individual HashMaps. Both the HashMaps are compared for and a counter is incremented for each matching record and the accuracy is calculated.

Output Observations:

	Without Stemming	With Stemming
Total Documents Correctly Classified	1585	1597
Accuracy	79.25%	79.85%

Individual Class Accuracies:

Class Name	Accuracy
alt.atheism	83%
comp.graphics	87%
comp.os.ms-windows.misc	0%
comp.sys.ibm.pc.hardware	80%
comp.sys.mac.hardware	84%
comp.windows.x	84%
misc.forsale	63%
rec.autos	92%
rec.motorcycles	96%
rec.sport.baseball	88%
rec.sport.hockey	99%
sci.crypt	95%
sci.electronics	72%
sci.med	88%
sci.space	94%
soc.religion.christian	95%
talk.politics.guns	94%
talk.politics.mideast	95%
talk.politics.misc	64%
talk.religion.misc	46%

Note: There were 100 files under each class type.

Observation/ Output Analysis:
The outputs for each type of class is calculated. Surprisingly none of the documents in the third class (2) were classified. A satisfactory overall average of 79.85% accuracy is being achieved.

Challenges:
One main hurdle was the size of the vocabulary, and calculating the word count. This challenge has been partially overcome by the removal of stop words and performing stemming operations. It can further be decreased by performing lemmatization.
There were a lot of run-time issues faced, the main one in particular was the time duration. Initially all computations were being done on the fly and hence the program run-time was very high. This has been taken by writing the probabilities to persisting HashMaps and running the test data reading these HashMaps.

Future Enhancements:

a. The vocabulary size can further be decreased by performing lemmatization.
b. Modifying the program to run it on Apache Spark or using Hadoop MapReduce jobs would drastically reduce the run time.
c. Provide an UI to interact with the program and provide options to train and test data and show the statistics.

References:

1. Tom M. Mitchell. Machine Learning. The McGraw-Hill Companies, 1997
2. http://www.hlt.utdallas.edu/~yangl/cs6375/slides/naive_bayes.pdf
3. http://www.cse.chalmers.se/alumni/markus/LangClass/LangClass.pdf
4. http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/
5. http://tartarus.org/martin/PorterStemmer