The project is due in hardcopy and via provide by the last class on Thursday April 26.

**Note:** If you want to do another project of your own choice, please come to talk to me. I will accept any project that exercises and tests some of the material we covered, as long as it is not too easy or too hard. BUT you must negotiate such a project in advance.

## 1 Introduction

In recent years text categorization algorithms have become important and widely applied, for example for automatic newsfeed labeling, spam filtering, or web page categorization. In this project we study a simple algorithm that works reasonably well in many cases, and offers interesting algorithmic choices to explore as part of the algorithms class.

For this project you will work with data from the “20 newsgroups” dataset\(^1\) which we have further processed and prepared for the project. Although the data comes from 20 different newsgroups we have partitioned it somewhat arbitrarily into two groups, so that we are facing a 2-class classification problem.\(^2\) Two similar applications that can be useful to think about for further intuition are spam filtering (where the two classes are spam vs. no-spam) and identifying interesting documents for a user (where the classes are interesting vs. not-interesting for the specific user).

The main idea in applying machine learning to such a problem, is that

- Given a dataset that has previously been annotated with correct labels, we can use an algorithm (such as the one given below) to learn some general classification procedure, which we call a hypothesis.

- Given a new example, we can apply the hypothesis to predict its unknown class label. The predicted label can be used in the application context. For example in spam filtering we might mark or remove messages predicted to be spam before they get to the user.

## 2 The Algorithms

Our learning algorithm, sometimes called the “naive-Bayes algorithm” is as follows. Given a corpus of documents and their labels:

- For each class calculate count(class) which is the number of documents of that class.

\(^1\)http://people.csail.mit.edu/jrennie/20Newsgroups/

\(^2\)The following groups belong to class YES: comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, rec.autos, rec.sport.baseball, sci.crypt, misc.forsale, talk.politics.misc, talk.religion.misc, sci.religion.christian. The following groups belong to class NO: comp.sys.mac.hardware, comp.windows.x, rec.motorcycles, rec.sport.hockey, sci.electronics, sci.med, sci.space, talk.politics.guns, talk.politics.mideast, alt.atheism.
• For each word and each class calculate \( \text{count}(\text{word} \mid \text{class}) \) which is the total number of times where “word” appears in documents of type “class”.

• Add one to each of the counts from the previous two steps.

• Turn the counts into frequencies dividing \( \text{count}(	ext{class}) \) by number of documents plus the number of classes, and \( \text{count}(\text{word} \mid \text{class}) \) by the total number of words in class “class” plus the vocabulary size.

This constitutes the “learning portion” of this algorithm. Notice that this is a simple algorithm that completely ignores word order and just counts occurrences of words in different classes.

For example consider a dataset with 3 documents with a vocabulary of size 5 \( \{a, b, c, d, e\} \) as follows: document \( \{a, b, c\} \) of class 1, document \( \{a, d, e\} \) of class 1, and document \( \{b, c, e, e\} \) of class 2. Then \( \text{count}(1)=2 \), \( \text{frequency}(1) = \frac{2+1}{3+2} = \frac{3}{5} \), \( \text{count}(2)=1 \), \( \text{frequency}(2) = \frac{1+1}{3+2} = \frac{2}{5} \), \( \text{count}(a\mid1)=2 \), \( \text{frequency}(a\mid1) = \frac{2+1}{6+5} = \frac{3}{11} \), \( \text{count}(a\mid2)=0 \), \( \text{frequency}(a\mid2) = \frac{0+1}{4+5} = \frac{1}{9} \), \( \text{count}(e\mid1)=1 \), \( \text{frequency}(e\mid1) = \frac{1+1}{6+5} = \frac{2}{11} \), \( \text{count}(e\mid2)=2 \), \( \text{frequency}(e\mid2) = \frac{2+1}{4+5} = \frac{3}{9} \), and so on.

Now for application, or the “test portion” of the algorithm, we use the learned information to classify a new document. Given a new document \( d \) to be classified

• For each class calculate

\[
\text{score}(	ext{class}) = \log[\text{frequency}(	ext{class}) \cdot \prod_{w \text{ in } d} \text{frequency}(w \mid \text{class})]
\]

where the product is over word tokens \( w \) in the document \( d \) and we ignore words not in the lexicon. That is, if a word appears multiple times then it is multiplied in the product multiple times; and if a word \( w \) appears in \( d \), but it does not appear in any of the training documents then it is not included in the product.

Note that it is important to use the logarithm that turns the product into a sum and avoids underflow problems in the computation — your implementation should calculate the sum of the log values instead of the log of the product.

• Then predict the class with the highest score.

For example, if we use the corpus above and want to predict the label of \( \{a, e, c, f\} \) then we ignore \( f \) and compare the logs of \( \frac{3}{5} \cdot \frac{3}{11} \cdot \frac{2}{11} \cdot \frac{2}{11} = 0.0054 \) (class 1) and \( \frac{2}{5} \cdot \frac{3}{9} \cdot \frac{3}{9} = 0.0049 \) (class 2) and pick class 1 as the predicted label.

For this project we consider a variation on the algorithm whose intention is to reduce the memory requirement and possibly run time of the algorithm in the test portion, i.e. when it is applied to new data. This comes at a cost of increase in run time for the learning portion, and possibly a decrease in the quality of prediction. The idea is as follows.

After finding all the frequencies in the previous algorithm we score each word by its ability to discriminate the classes (in this part we restrict the discussion to 2-class problems):

• For each word calculate

\[
\text{delta}(\text{word}) = |\log(\text{frequency}(\text{word}|\text{class}1)) - \log(\text{frequency}(\text{word}|\text{class}2))|
\]
Then pick the $n$ most discriminating words, that is, those with highest delta, and redefine the vocabulary to include only these words.

The final vocabulary size $n$ is a parameter of the algorithm that we vary in the evaluation below. Therefore the learning portion does an extra step of pruning the vocabulary and its run time is longer. The test portion is done exactly as before, but it now has a smaller vocabulary, and can be deployed with a much smaller memory signature.

3 Your Task

3.1 Implementation

Implement the basic algorithm and its variation to be as efficient as possible for both learning portion and test portion. In order to do this you should think about the operations needed by the algorithm, and data structures that support these efficiently.

All non elementary data structures (anything beyond integers, floating point values, arrays and strings) and all details of the algorithms (including sort, hash, etc) should be implemented from scratch and should not use existing libraries.

You are free to choose any language for your implementation as long as you adhere to implementing all non elementary constructs yourself and as long as your code can be run on the CS system. Specifically it must be able to run on linux.cs.tufts.edu.

3.2 Evaluation

Run the learning portion of the program on the data in /comp/160/files/project/train/\(^3\) The directory includes every example as a separate text file. The files have been taken from the web site given above and have been “cleaned” by removing special characters. You can treat any sequence of characters other than space as a “word” for your lexicon. The labels of the examples are given in the file index.Full in the same directory.

Then calculate the accuracy of the predictions on the data in /comp/160/files/project/test/ where again the labels (that you need to figure out if the prediction is correct) are in the file index.Full in the same directory. The accuracy is the fraction of test documents for which the label is predicted correctly by our algorithm. We emphasize that these documents are distinct from the ones in the train portion so that this gives an estimate of the performance we might expect on new unseen data.

Repeat the above for the truncated method with $n$ going from 200 to to the full vocabulary size (decide on some intermediate values). For each of these, record the train time, the test time to classify all test documents, and classification accuracy.

Plot the results (times and accuracy) as a function of $n$. Then consider the results: is the modified version useful? what values of $n$ provide a good operating point for the algorithm?

Two suggestions for your work. The dataset is large but you do not need to copy the text files into your own space. Instead you can copy the index file and use it and the path to read the given files.

\(^3\)Note that this is a path on the CS file system and it is not accessible via the web.
Second, when developing and debugging your code it might be advisable to take a small subset of the documents (remove some lines from the index files) and work with the small subset. Then when the program is stable you can run on the full dataset.

3.3 Report

Write a short report that (1) explains what data structures and algorithms you use in your implementation, (2) explains why you chose these, (3) presents the results using the plots from the previous section and any numerical summaries you find useful, (4) analyzes the results in terms of how successful the algorithm is and the effect of $n$ on the results.

[extra work] Those wishing to explore further might design different implementations and compare the effect of implementation on performance. Or you might try to improve the predictions of the algorithm while still making the work interesting from an algorithms course perspective. However, no such work is required as part of the project.

3.4 What to Submit

You should submit

- A printout of: all your code for this project, and the report. Please make sure that your code is well documented, and that as part of the documentation you explain how to run the code on the CS system.

- Submit the same (code and report) through the provide system. Please put all the files from the previous item into a zip or tar archive, for example call it project.zip. Then submit using provide 160 proj project.zip.

The projects will be graded based on the (1) amount of work done, (2) quality of the code, (3) documentation of the code, (4) explanation of choices of algorithms and data structures used, and (5) quality of the report.