CS 276 Programming Exercise #2 (100 points)

Assigned:	Tuesday, November 11, 2008
Due Date:	Thursday, November 20, 2008
Delivery:	All students should submit their work electronically. See below for details.
Collaboration:	You are allowed (but not required) to work in pairs for this assignment. Teams of two should only submit on copy of their work.
Late policy:	Refer to course webpage.
Honor code:	Please review the collaboration and honor code policy on the course webpage.

---

1. Overview

In this assignment, you will conduct an exercise in supervised machine learning by training classifiers for Usenet newsgroup messages. We provide you with a data set and starter code that iterates through the messages. Your task is to classify new messages into one of the Usenet newsgroup categories by training classifiers such as a Naïve Bayes classifier.

This assignment might be more challenging than practical exercise #1, but it should also be more conceptually interesting. It also offers greater opportunity for creativity and extra credit.

a. Programming language and computing environment

Although programming this assignment in Java may be the obvious choice, you are free to program it in any other language you are comfortable with. You should probably complete the assignment using the Stanford UNIX Computing Resources, possibly on one of the intensive-computing machines such as bramble, hedge, pod, or vine. See the following webpage for more details. http://www.stanford.edu/services/unixcomputing/which.html#intensive

While you are free to use any platform you choose to develop your code, your deliverables must run on **hedge** by the grading script.

b. Starter code and data set

The starter code is located in `/afs/ir.stanford.edu/class/cs276/pe2`. Make a new directory and copy everything into it.

We're using the 20 Newsgroups collection, an archive of 1000 messages from each of 20 different Usenet newsgroups. The version we selected for this assignment excludes cross-posts (messages that were posted to more than one of the 20 newsgroups), so the actual number of messages is slightly below 1000 per group. For more information, please visit http://people.csail.mit.edu/people/jrennie/20Newsgroups. A copy of the data set is located in `/afs/ir.stanford.edu/data/linguistic-data/TextCat/20Newsgroups/20news-18828`, with each newsgroup in a subdirectory containing one file per message. If you are using the Stanford UNIX Computing Resources and/or have access to the Stanford AFS file system, you should be able to read data set from its current location. There is no need to make a copy of the files.

More details about the start files:

setup.sh	Builds the code and runs MessageParser to produce parsed messages. To get started, run: `./setup.sh /afs/ir/data/linguistic-data/TextCat/20Newsgroups/ 20news-18828/` If you aren't programming in Java or make any major modifications to the MessageParser, make sure that this script still works as it will be called by the automated grading script. Please note: "setup.sh" was updated on November 17 to explicitly call Sun Java 1.6.0 instead of gij 4.2.3. Memory allocation has been doubled to account for increased memory necessary to store pointers in 64-bit computing environments such as **hedge** machines.
runNaiveBayes.sh	A wrapper around NaiveBayesClassifier.java. We will use this script to call your code, so make sure it works before you submit. Please note: "runNaiveBayes.sh" was updated on November 17 for the same issues and reasons outlined above.
MessageParser.java	Reads in the messages to produce an iterator for the data. Creates separate features for the subject and body fields, employs the stop words and the Porter stemmer and counts word frequencies. Outputs a file containing the data, including the newsgroup labels for those vectors. You may modify this file.
MessageIterator .java	Sample implementation to iterate the parsed output of MessageParser.
MessageFeatures .java	The data class for the iterator, representing a single parsed message.
NaiveBayesClassifier .java	This is the main entry point for your code. Right now it just parses the command line arguments and calls functions that you need to fill in. Feel free to add all of your code here or add additional files as needed.
Stemmer.java	Porter stemmer implementation.
english.stop	List of stop words.
cs224n/util/ Counter.java	A map from objects to doubles. Includes convenience methods for getting, setting, and incrementing element counts. (You might find this useful, but don't have to use it).
cs224n/util/ PriorityQueue.java	A priority queue based on a binary heap. (You might find this useful, but don't have to use it).

c. Related literature

A significant portion of this assignment is based on Jason Rennie's paper on improving the text classification accuracy of Naïve Bayes classifiers. You will find the paper *Tackling the poor assumptions of Naïve Bayes Text Classifier* at http://people.csail.mit.edu/jrennie/talks/icml03.pdf.

2. Write-Ups and Deliverables

This project requires you to implement a number of modifications to the Naïve Bayes classifier. Please do them step by step and get results at each step, so that even if you don't finish the whole project, you can still deliver a working partial project. Be sure to save your work at every milestone. You may wish to read the whole assignment before you start.

a. Automatic grading script

Most of your code will be graded by an automated grading script!

Please pay attention to the required inputs and outputs for each part of your program described below. In particular, do NOT output any extra text to stdout. Instead output any debugging messages that you may have to stderr (i.e. in Java use System.err.println(…)).

Before we test your code, we will run setup.sh once. You don't need to modify this file to complete the assignment, but if you make significant changes to MessageParser/Iterator or choose not to program in Java, you may need to. Then for each part of the assignment below, we will call

```
./runNaiveBayes.sh <mode> <train>
```

where mode specifies which part of the assignment you should run and train is the location of a training data file produced by MessageParser.

b. Required components

«Deliverable #1» Multivariate Naïve Bayes classifier

Write and implement a multivariate binomial Naïve Bayes classifier. Make sure you take into account issues discussed in the lecture such as underflow prevention and over-fitting issues. We will call your code by:

```
./runNaiveBayes.sh "binomial" <train>
```

You should read in the file of training data and train your model. Then for the first 20 messages in the training data for each of the newsgroups, output the log-probability (in natural log as outputted by `Math.log()`) that it belongs to each of the newsgroups in the training data.

Output one message per line, with the individual log-probabilities separated by tabs. The order of the log-probabilities on the line should be in the order of the newsgroup number (0-19). For example if there were only 3 newsgroups, the first two lines of your output should be formatted like:

```
-0.397940 \t -0.522878 \t -0.522878
-0.698970 \t -0.602060 \t -0.259637
```

You may call the function `outputProbability(double[])` in NaiveBayesClassifier.java to output a line of tab-separated values.

«Deliverable #2» Feature selection using χ^2

Now add feature selection using Chi-Square. We will call your code by:

```
./runNaiveBayes.sh "binomial-chi2" <train>
```

For each newsgroup in the training set, output the 20 best words (tab-separated, one newsgroup per line). Then, retrain your multivariate classifier, using as features only the words that appear in the top 300 list of at least one newsgroup. Output your new predictions for the first 20 messages in each newsgroup in the same format outlined in deliverable #1.

«Deliverable #3» Multinomial Naïve Bayes Classifier

Write and implement a multinomial Naïve Bayes classifier in Java as described in lecture notes. Make sure that you account for underflow and over-fitting.

For the first 20 messages in each newsgroup, output (tab-separated, one per line) the newsgroup number that you predict it belongs to. (We will call your code with a mode of "`multinomial`").

«Deliverable #4» *k*-fold cross-validation.

So far, we've been training and testing on the same data set. That's generally considered bad practice as what we are really interested in is the ability of the classifier to correctly classify NEW documents. So, for this part you will write a *k*-fold cross validation to test the accuracy of your classifier.

We suggest the value k=10 as a start. The data set is separated into two sets, called the training set and the testing set. The classifier is trained using the training set only. The classifier is then tested on the testing set. ~~The errors it makes are accumulated to give the mean absolute test set error which gives the accuracy of the classifier..~~ In *k*-fold cross validation, the data set is divided into *k* subsets, and the process above is repeated *k* times. Each time, one of the *k* subsets is used as the test set and the other *k*-1 subsets are put together to form a training set. The error is then averaged over the *k* folds, which gives the accuracy (i.e., accuracy = 1 – error) of the classifier.

In your written report, give the average cross validation error numbers for both multivariate and multinomial Naïve Bayes with and without feature selection. We will not explicitly call your code for this part.

«Deliverable #5» Improving the Naïve Bayes Classifier

A lot of recent machine learning work emphasizes the advantages of discriminative classifiers, but a recent paper by Rennie et al. suggests that various normalizations can achieve a lot of the same gains within a basically Naïve Bayes classifier. Investigate this by trying to improve your MNB classifier using the extensions described in the paper: *Tackling the poor assumptions of Naïve Bayes Text Classifier* (http://people.csail.mit.edu/jrennie/talks/icml03.pdf).

We want you to implement Transformed Weight-normalized Complement Naïve Bayes (TWCNB). Please implement it in the following increments and collect data after each step.

i. Complement Naïve Bayes (CNB)
ii. Weight-normalized Complement Naïve Bayes (WCNB)
iii. Transformed Weight-normalized Complement Naïve Bayes (TWCNB)

Comment on the effects of each step in your report. We will only run your code once for this part with mode "`twcnb`". You should output the results from part (iii), though partial credit will be given for (i) or (ii). Output your predictions in the same format as specified in deliverable #3.

Make sure to read this paper before you start implementing the basic MNB classifier because a modularized implementation will make implementation of this extension much easier.

«Deliverable #6» Experimenting with different techniques

Another way to improve a classifier is with various domain specific features. Remember that anything can be a feature, not just individual words (for example, word pairs, the presence of a $ sign, etc.). Consider and test any other promising ways of improving your classifier. Other possible techniques you may wish to consider include lowercasing, stemming and upweighting of different zones of documents. We will not explicitly call your code for this part of the exercise, but be sure to summarize what you implemented and the effects of your code.

c. Optional components

This portion can be worth up to 25 points (5% of your overall class grade) but you must implement the required portions before you attempt this. Rather than sticking to Naïve Bayes classifiers, an avenue for possible extended investigation is to experiment with different classification methods and their relative advantages and disadvantages.

«Optional Deliverable #7» Support vector machines

Train a SVM using a machine-learning library such as SVMlight (http://svmlight.joachims.org) to classify the newsgroup data. You might want to experiment with different kernels and compare SVM results with results you achieved using Naïve Bayes, and comparatively to the results presented in class from Dumais et al. or Joachims.

Since SVMlight is written in C, you may consider LIBSVM (http://www.csie.ntu.edu.tw/~cjlin/libsvm) if you prefer Java.

«Optional Deliverable #8» Other classifiers

Try comparative experiments with other different classifiers (e.g., decision trees, nearest neighbor methods) using an off the shelf workbench such as WEKA (http://www.cs.waikato.ac.nz/ml/weka). Report and compare your results with results you achieved with an SVM and/or Naïve Bayes.

«Optional Deliverable #9» Additional feature selections

Try implementing different feature selection methods described by Schneider. (http://acl.ldc.upenn.edu/P/P04/P04-3024.pdf)

d. Written report

In addition to your code, please submit a **brief report** (6 – 8 pages including figures. We will not read beyond the eighth page) summarizing your work.

- Focus on the comparative nature of your results from different classifiers and/or feature selection techniques. (e.g. You should report parameters you used in your algorithms and compare your results obtained from various components of this exercise.)

- Use figures/graphs/tables to show the numbers from your results and to illustrate the effects of different combinations of classifiers/techniques.

- Emphasize interesting experimental results and discuss the reasons for such results (e.g. "Based on our numbers, we found weight normalization was ineffective because …")

- You may assume we have knowledge of techniques covered in the lectures. (e.g. You need NOT explain what a Naïve Bayes classifier is or how to implement it.)

Your report should include the following:

i. Accuracy of multivariate and multinomial Naïve Bayes classifiers.
ii. Effects of and a discussion on χ^2 feature selection.
iii. Changes in accuracy due to and implications of applying *k*-fold cross-validation.
iv. Comparison of Naïve Bayes classifier with Transformed Weight-normalized Complement Naïve Bayes (TWCNB)
v. Results from applying domain-specific techniques.
vi. Anything interesting you might have realized while doing this project. Be sure to mention what you did that merits the extra credits.

Your report may also include the following for extra credits:

vii. Comparison of NB and TWCNB with other classifiers if you choose to implement optional deliverables #7 or #8.
viii. Results from and discussion on additional feature selection if you choose to implement optional deliverable #9.

e. Submission

Your submission should include:

- Any code necessary to compile and run your program from deliverables #1, #2, #3, and #5. Your code must compile and run in the submission directory (no hard-coding please).

- Your written report in PDF or Microsoft Word format. Your report should be named as either "report.pdf" or "report.doc".

- README file indicating how to run your code and implementation detail that you feel is necessary for us to know to grade your work.

When you're ready to submit, change to the directory where all of your files are located and run the submission script:

```
/usr/class/cs276/bin/submit-pe
```

f. Additional notes

The optional extra credit parts will require you to convert the training data into the format of the machine learning library you use. Submit whatever code you write for extra credit along with everything else and specify in your README file how we can call it.

In addition, training certain classifiers may involve intensive computation. The Stanford UNIX Computing Resources are a shared by the entire university and should not be abused. To be fair and limit the load on the cluster, we ask that each group use a maximum of five intensive processes in total. If you are running a long experiment, you should `nice` the processes, like this:

```
% nice myprog parameters
```

If you have extra access to computing resources, we encourage you to use them, as long as your code compiles and runs on **hedge** as well.

By default Java uses only a fixed memory pool, which will probably be too small for the experiments you run. You can use the following command to set the maximum size of the heap allocated to your java virtual machine. Following command sets it to 1000 mega-bytes.

```
java –Xmx1000m
```

---

3. Grading Criteria

To earn full credit on this assignment, your group must submit:

a. Implementation: Working code that can be called by the grading script (60 points)

- 15 points: Multivariate Naïve Bayes classifier as specified in deliverable #1.
- 10 points: Feature selection using χ^2 as specified in deliverable #2.
- 15 points: Multinomial Naïve Bayes classifier as specified in deliverable #3.
- 20 points: Extensions of Naïve Bayes classifier as specified in deliverable #5.

b. Implementation: Summary of the results from your code (20 points)

- 10 points: *k*-fold cross validation as outlined in deliverable #4.
- 10 points: Combinations of different techniques as outlined in deliverable #6.

c. Written report (20 points)

- Clear and concise presentation of your results.
- Comparison of classifiers and techniques.
- Interesting results and discussion of what you observed.

d. Extra credits (up to 25 points)

- Other classifiers and/or feature selection techniques from optional deliverables #7 to #9.
- Brief explanation of what you did and what you found.

Have fun! Good luck!