Homework 6
Naive Bayes Learning to Classify Text

15-681: Machine Learning
Tom Mitchell
Carnegie Mellon University

Due: December 2, 1997

Introduction

This assignment gives you an opportunity to apply a Naive Bayes learner to the problem of classifying text. You will use the program Rainbow, written by Andrew McCallum. Rainbow provides a suite of learning methods for text classification, including the method described in section 6.10 of the Machine Learning textbook.

Installing the code

To run the rainbow program, you need to be on a SunOS machine. For example, one of the unixXX servers will do fine. If you absolutely positively want to run this on another architecture, send mail to Frank (dellaert@cs.cmu.edu) and ask for the location of the source code.

You should copy the executables and data from the directory

```
/afs/cs/project/theo-20/www/rainbow/hw6
```

However, since this directory contains some subdirectories, you might find it more convenient to copy the tar file and untar it by issuing the following unix commands:

```
cp /afs/cs/project/theo-20/www/rainbow/hw6.tar .
tar xf hw6.tar
rm hw6.tar
```

The Naive Bayes Code

Brief documentation for Rainbow is available by typing `./rainbow --help`.

Data format: Rainbow assumes each training or test document is an ASCII file stored in Unix. To provide training data, you must create a directory containing no files, then create one subdirectory per target value and place the files in these subdirectories. An example is the subdirectory mini_newsgroups in the `hw6/` directory. Here you will find two subdirectories named `alt.atheism` and `misc.forsale`. Each of these subdirectories contains 100 files, containing 100 newsgroup articles drawn from the corresponding newsgroup. You will use these files to train rainbow to classify new articles into one of these two newsgroups.

Setting up Rainbow: To use Rainbow to implement the Naive Bayes classifier described in the textbook, first type the following on a single line to UNIX to define the alias `naivebayes`

```
alias naivebayes './rainbow -m naivebayes --no-stoplist --skip-header
                            --prune-vocab-by-occur-count=1 -d "./learned-model"'
```

This tells Rainbow to use naivebayes as its learning method, to consider every word found in every document, to skip header information in documents (e.g., subject, title, etc. lines in newsgroup articles), and to store its learned model on the subdirectory ./learned-model/ of the current directory (the directory to which you are currently connected). Type `./rainbow --help` for further documentation on command line arguments to Rainbow.

The questions below will step you through the process of training and testing a classifier, respectively. If needed, more verbose (but somewhat dated) documentation on Rainbow is also available at

```
http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes/gentle\_intro.html.
```

Question 1: Training

Train a Naive Bayes classifier to classify newsgroup articles as `alt.atheism` versus `misc.forsale`, by typing:

```
naivebayes  --index mini_newsgroups/*
```

This should produce output something like:

```
Class 'alt.atheism'
  Counting words... files : unique-words ::      100 :   4152
Class 'misc.forsale'
  Counting words... files : unique-words ::      100 :   6330
Class 'alt.atheism'
  Gathering stats... files : unique-words ::      100 :   6330
Class 'misc.forsale'
  Gathering stats... files : unique-words ::      100 :   6330
Making vector-per-class... words ::     30
```

Notice Rainbow should now have created a subdirectory for you named "./learned-model" In it you will find a number of files.

After the training has finished, examine the words with highest information gain for discriminating the two target classes. Use the `--infogain-vector` command line argument to do this, as in

```
naivebayes --infogain-vector=10
```

1. Do these make sense? *Before proceeding with the next item*, predict for each of these 10 words in which newsgroup they are more likely to occur (this is simply to make you think about the problem: no points will be deducted for guessing wrong).
2. Now, examine the exact word frequencies in the training set by using the `--print-word-counts` command line argument. List the word frequencies you find for each of the 10 words. Do the actual numbers agree with your intuition ?
3. If everything went right, the word 'of' should have appeared among the 10 most informative. What newsgroup is it more strongly associated with ? Why would you expect such a common word to be associated more strongly with this newsgroup, rather than the other one ?
4. If we would add more newsgroups than just these two, do you expect the word 'of' to carry more or less information ? Why ?

Question 2: Querying

You may ask Rainbow to classify new documents using the model we just trained. Type the following on a single line to apply the learned model to one of the training examples. *Note the use of double quotes around the file name.*

```
naivebayes --query="mini_newsgroups/misc.forsale/75987"
```

The output produced by Rainbow for this query is a list of the classes, each followed by a number. This number is the probability $P(class|document)$ calculated using the Naive Bayes formula. Note in this example, the probabilities output for the two classes are 1 and 0:

```
mini_newsgroups/misc.forsale 1
mini_newsgroups/alt.atheism 0
```

You can command Rainbow to classify all the data in the training set by using the command

```
naivebayes --test-files mini_newsgroups/
```

Each line of output lists a filename, its true classification, and the posterior probabilities assigned to each class by the learned Naive Bayes model. Notice this output is a bit verbose. You can put it into a file

```
naivebayes --test-files mini_newsgroups/ > f
```

For your convenience, we have provided a Perl script that you may use to look at a summary of the results (i.e., number of examples correctly classified, incorrectly classified). To try it, type

```
./rainbow-stats < f
```

1. How accurate is Rainbow over the training set? Report the confusion matrix and accuracy calculated by rainbow-stats.

Question 3: Testing on a Holdout Set

You can tell Rainbow to test the classification performance on a hold-out set, i.e. a portion of the data that will not be used for training the classifier. This should give much more trustworthy results than testing on the training data itself. For example, to test the performance on a holdout set consisting of 40% of the data, use:

```
naivebayes --test-percentage=40 --test=1 > h
```

1. Plot a graph showing the test set accuracy as you vary the number of training examples. Use training sets of size 2, 6, 20, and 70, using the remaining examples as your test set (Since there are 200 datapoints in total, this is easily translated into holdout-set percentages). On your graph include 95% confidence interval estimates of the true accuracy at each point.

2. In a technique called k-fold cross-validation (page 112 of the book), we use a different random partitioning of the data for k different times. The constant k can be specified using `--test=k`. Plot a second graph like the one above, but now using k=10 for each data point (Note: just hand in the graph with the confidence intervals. There is no need to print out all the confusion matrices produced by rainbow-stats).

3. If you were to use these plots to assess the quality of your text-classification technique, which one would you trust more ? Why ?

Question 4: Limited Vocabulary

1. Try limiting the learner's vocabulary to just the 10 words with the highest information gain. Try this for the holdout set size of your choice(s) and report the results. See the Rainbow command line argument `--use-vocab-in-file=FILE`

Optional Extra Credit

Note: Robotics PhD students can choose to use this as a project to obtain graduate core credit.

Pick a different dataset you are interested in and see whether you can learn something interesting. For example, you might like to automatically classify your email into subfolder, or your files into subdirectories, or web pages according to whether you'd add them to your bookmarks list...

More specifically,

- Describe your intended application and the type of data you collect.
- Describe the design of your learning algorithm, explain any interesting design choices.
- Report your experimental results.
- Turn in a writeup of 2-4 pages. Absolutely no writeups longer than 8 pages (unless used as graduate course project) !
- IMPORTANT: if you have no headers in your documents, then redefine your naivebayes alias to omit the `--skip-header` flag.