BigML Assignment 1: Streaming Naive Bayes

Due Mon, January 28 via Blackboard

January 23, 2013

1 Important Note

This assignment is the first of three that use the Naive Bayes algorithm. You will be expected to reuse the code you develop for this assignment for future assignments. Thus, the more you adhere to good programming practices now (e.g. abstraction, encapsulation, documentation), the easier the subsequent assignments will be.

Yanbo Xu (yanbox@cs.cmu.edu) is the contact TA for this homework. Please post clarification questions to the Google Group:
`machine-learning-with-large-datasets-10-605-in-spring-2013`

2 Naive Bayes

Much of machine learning with big data involves - sometimes exclusively - counting events. Multinomial Naive Bayes fits nicely into this framework. The classifier needs just a few counters.

For this assignment we will be performing document classification using streaming Multinomial Naive Bayes. We call it streaming because the input and output of each program are read from stdin and written to stdout. This allows us to use Unix pipe "|" to chain our programs together. For example:

```
cat train.txt | java NBTrain | sort -k1,1 | java NBTest -t test.txt
```

The streaming formulation allows us to process large amounts of data without having to hold it all in memory.

Let y be the labels for the training documents and wi be the ith word in a document. Here are the counters we need to maintain:

(Y=y) for each label y the number of training instances of that class

(Y=*) here * means anything, so this is just the total number of training instances

(Y=y,W=w) number of times token w appears in a document with label y.

(Y=y,W=*) total number of tokens for documents with label y.

The learning algorithm just increments counters:

```
for each example {y [w1,...,wN]}:
    increment #(Y=y) by 1
    increment #(Y=*) by 1
    for i=i to N:
        increment #(Y=y,W=wi) by 1
    increment #(Y=y,W=*) by N
```

You may elect to use a tab-separated format for the event counters as well: eg, a pair <event,count> is stored on a line with two tab-separated fields. Classification will take a new documents with words w1,...,wN and score each possible label y with the log probability of y (as covered in class).

For now (hint, hint), you may keep a hashtable in memory, with keys like Y=news, Y=sports,W=aardvark, etc. You may NOT load all the training documents in memory. That is, you must make one pass through the data to collect the count statistics you need to do classification. Then, write these counts (feature dictionary) to disk via stdout.

Important Notes:

- At classification time, use Laplace smoothing with $\alpha = 1$ as described here: `http://en.wikipedia.org/wiki/Additive_smoothing`.
- You may assume that all of the test documents will fit into memory.
- With the exception of the test set, all files should be read from stdin and written to stdout
- Use this function to change documents into features:

```
static Vector<String> tokenizeDoc(String cur_doc) {
        String[] words = cur_doc.split("\\s+");
        Vector<String> tokens = new Vector<String>();
        for (int i = 0; i < words.length; i++) {
              words[i] = words[i].replaceAll("\\W", "");
                if (words[i].length() > 0) {
                        tokens.add(words[i]);
                }
        }
        return tokens;
}
```

3 The Data

For this assignment, we are using the Reuters Corpus, which is a set of news stories split into a hierarchy of categories. There are multiple class labels per document. This means that there is more than one correct answer to the question "What kind of news article is this?" For this assignment, we will ignore all class labels except for those ending in CAT. This way, we'll just be classifying into the top-level nodes of the hierarchy:

- CCAT: Corporate/Industrial
- ECAT: Economics
- GCAT: Government/Social
- MCAT: Markets

There are some documents with more than one CAT label. Treat those documents as if you observed the same document once for each CAT label (that is, add to the counters for all labels ending in CAT). If you're interested, a description of the class hierarchy can be found at `http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a02-orig-topics-hierarchy/rcv1.topics.hier.orig`.

The data for this assignment is at: `/afs/cs.cmu.edu/project/bigML/RCV1`
Note that you may need to issue the command *kinit* before you can access the afs files. The format is one document per line, with the class labels first (comma separated), a tab character, and then the document. There are three file sets:

```
RCV1.full.*
RCV1.small.*
RCV1.very_small.*
```

The two file sets with "small" in the name contain smaller subsamples of the full data set. They are provided to assist you in debugging your code. Each data set appears in full in one file, and is split into a train and test set, as indicated by the file suffix.

4 Deliverables

Submit a compressed archive (zip, tar, etc) of your code. With it, include a makefile so that

- The command *make demo* will build and test a classifier using the training and test files for the "very_small" dataset.

- The command *make test TESTFILE="test.txt"* builds a classifier using the "very_small" training file, and then classifies the documents in file provided (test.txt in this example).

Your classification code should print out the classification results, including the log probabilities of the best class y:

$$ln(p(Y = y)) + \sum_{w_i} ln(p(W = w_i|Y = y)) \quad (1)$$

Notice that we're using the natural logarithm here. The output format should have one test result per line, and each line should have the format:

[Label1, Label2, ...]<tab>Best Class<tab>Log prob

where **[Label1, Label2, ...]** are the true labels of the test instance, **Best Class** is the class with the maximum log probability (as in Equation 1), and the last field is the log probability. The last line of the file should give the percent correct. Here's an example of the output format:

```
[CCAT, C181, C18, GCAT] CCAT    -1042.8524
[GCAT]  CCAT    -4784.8523
...
Percent correct: 9/10=90.0%
```

You may count a document correct if the most probable class matches any of the labels (as in the first line of the example above). This way you get credit for multiply labeled documents if you get any of the labels correct.

In addition to your code an a makefile, please include a pdf document with:

1. The output of your "make demo" command, as well as the percent correct on the other two test sets (using classifiers trained with the corresponding train set).
2. Answers to the following two questions:
   (a) What changes could you make to reduce the amount of RAM required for the feature dictionary?
   (b) Right now we're basically ignoring the fact that there are multi-labeled instances in the train/test sets. How would you extend your algorithm to enable it to predict multiple labels?

5 Marking breakdown

- Code correctness and makefile functionality [**70 points**].
- 1a [**10 points**]
- 2a [**10 points**]
- 2b [**10 points**]