BigML Assignment 1: Streaming Naive Bayes

Due Tuesday, January 20. No submission required

January 15, 2015

You can work in groups. However, no written notes can be shared, or taken during group discussions. You may ask clarifying questions on Piazza. However, under no circumstances should you reveal any part of the answer publicly on Piazza or any other public website. The intention of this policy is to facilitate learning, not circumvent it. Any incidents of plagiarism will be handled in accordance with CMU's Policy on Academic Integrity.

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: _____ (e.g. *Jane explained to me what is asked in Question 3.4*)
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: _____ (e.g. *I pointed Joe to section 2.3 to help him with Question 2*).

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.

Rahul Goutam (rgoutam@cs.cmu.edu) and Yipei Wang (yipeiw@cmu.edu) are the contact TAs for this homework. Please post clarification questions to Piazza site.

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 | java NBTest 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 document with words w1,...,wN and score each possible label y with the log probability of y (as covered in class).

For now (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

You should implement the algorithm by yourself instead of using any existing machine learning toolkit.

The training code NBTrain.java should be able to run, using commands:

```
cat train.txt | java NBTrain
```

This will output the streaming counts for the NB model, in the tab-separated two column form:

```
Y=CCAT,W=he 3.0
Y=CCAT,W=saw 1.0
Y=CCAT,W=her 3.0
Y=CCAT,W=duck 4.0
Y=CCAT,W=or 1.0
Y=CCAT,W=* 123.0
Y=CCAT 10.0
Y=* 10.0
...
```

The test code provide a one-per-line prediction for each test case, so the full streaming command is:

```
cat train.txt | java NBTrain | java NBTest test.txt
```

which produces the following output:

```
Best Class<tab>LogProbability
```

Here, the Best Class is the class with the maximum log probability.

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

Note that we will be using natural logarithm. Here's an example of the output format:

```
CCAT -1042.8524
GCAT -4784.8523
...
```

5 Submission

You are not required to submit anything for this assignment. This assignment is like a personal milestone for you and you should aim to complete it by Tuesday (1/20/2015). The next assignment will build on top of this assignment.