BigML Assignment 1a: Streaming Naive Bayes

November 27, 2017

1 Important Note

Assigments 1a and 1b are a pair. 1a WILL NOT be graded and you don't need to make any submission on Autolab - it's here to help you budget your time. 1b WILL be graded. You should plan to finish 1a before 1b is released so you have time to finish 1b. You need to finish all the assignments in Python.

Ning Dong (ndong1@andrew.cmu.edu) and Chen Hu (chenh1@andrew.cmu.edu) are the contact TAs for this assignment. Please post clarification questions to the Piazza, and the instructors can be reached at the following email address: *10605-Instructors@cs.cmu.edu*.

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 we won't load the training data into memory: instead we will load one document at a time, use that document to update the statistics that define the classifier, and then discard the document. The streaming formulation allows us to process large amounts of data—more than can fit 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 should 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, with field one the event, and field two the count. 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, you may keep a hashtable(note: hashtables are implemented with 'dict' in python) 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.

You will use a second invocation of your program to read statistics from stdin, and classify a test file. For example, you could do training and testing like the following, with Unix pipes.

```
cat train.txt | python NBTrain.py | python NBTest.py -t test.txt
```

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:

```
import re
def tokenizeDoc(cur_doc):
    return re.findall('\\w+',cur_doc)
```

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://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf`.

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

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 the expected output of very_small dataset to help you debug:

```
['C24', 'CCAT', 'M14', 'MCAT'] MCAT -9893.7510
['E51', 'E512', 'ECAT', 'GCAT', 'GDIP'] ECAT -3912.7886
['C15', 'C152', 'C18', 'C181', 'CCAT'] CCAT -1121.6191
['GCAT'] ECAT -1610.1366
['C13', 'CCAT', 'GCAT', 'GHEA'] CCAT -701.3665
['C13', 'CCAT', 'M11', 'MCAT'] CCAT -1453.3629
['C11', 'C13', 'CCAT', 'E12', 'ECAT', 'M13', 'M132', 'MCAT'] ECAT -2218.3008
['C31', 'CCAT'] CCAT -2285.0896
Percent correct: 7/8=0.8750
```

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. You can ignore those test instances where none of true labels ends in CAT.