DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF GEORGIA

CSCI6900 Assignment 1: Naïve Bayes on Hadoop

DUE: Friday, January 29 by 11:59:59pm

Out January 8, 2015

1 INTRODUCTION TO NAÏVE BAYES

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

For this assignment we will be performing document classification using Multinomial Naïve Bayes. Let y be the labels for the training documents and w_i be the i^{th} 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
```

[1]

Note that you will need to be clever with how these counters are incremented and stored. Since your Mappers will each read individual documents, multiple nodes will have to update the same counter. It's up to you to synchronize these counts. Classification will take a new documents with words $w_1, ..., w_N$ and score each possible label y with the log probability of y (as covered in class).

At classification time, use Laplace smoothing with $\alpha = 1$ as described here: `http://en.wikipedia.org/wiki/Additive_smoothing`.

Use this function to convert 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;
}
```

[*2*]

(you're free to use your own document parsing method, of course; this is just to get you started)

2 Installing Hadoop

2.1 Local machine

Installing Hadoop on your machine entails two modes of operation: "standalone" mode, where you effectively download the Hadoop .jars and dependencies and run it immediately; and "pseudo-distributed" mode, where Hadoop behaves as though it was running in a distributed environment complete with HDFS and the supporting Hadoop job daemons. The former is an excellent way to make sure your code runs correctly without NullPointerExceptions; the latter is useful for a first-pass at testing how your program will behave in a distributed environment.

See the Hadoop documentation for excellent instructions on setting up local installs of both standalone and pseudo-distributed Hadoop: `http://hadoop.apache.org/docs/r1.2.1/single_node_setup.html`. The Stanford MMDS course also has a great tutorial for setting up Hadoop locally: `http://web.stanford.edu/class/cs246/homeworks/tutorial.pdf`. It also has excellent instructions for installing a preconfigured Hadoop virtual environment (Cloudera), if you prefer that route.

2.2 AWS ELASTIC MAPREDUCE

Each registered student receives AWS credits. If you have not received your code, please let me know. For students who are auditing, Amazon often runs promotions for students signing up with AWS for the first time (if you choose the `micro` compute tier).

2.2.1 SUBMITTING A .JAR JOB

The easiest way to submit jobs to AWS is via the commandline. You need to install the elastic mapreduce command line interface; there are many versions for many different languages.

You can also start a job with the AWS console's web interface. Go to the EMR (elastic mapreduce) tab in your AWS console, click create new jobflow, choose the custom JAR jobtype and the rest is pretty self-explanatory. This takes longer than the command line but I personally find it more reliable.

3 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: `s3://uga-mmd/rcv1/`. You can view them in a web browser at: `https://s3.amazonaws.com/uga-mmd/rcv1/`. 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.*
```
[3]

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 (e.g. `RCV1.very_small_train.txt` and `RCV1.very_small_test.txt`; repeat for the `small` and `full` datasets as well).

4 Deliverables

Create a folder in your repository named `assignment1` and keep all your code for this assignment there. If my BitBucket username is `magsol`, my assignment should be in `magsol/assignment1` (you'll need multiple folders in your repository, one for each assignment as the semester progresses). I will reference the code in that folder for partial credit, but for testing your program, **you will need to also commit a .jar executable I can run on AWS: assignment1.jar**. This needs to have a commit timestamp *before* the deadline to be considered on time.

Please include a `README` file with 1) the command you issued to run your code, and 2) any known bugs you were unable to fix before the deadline. In addition, please provide answers to the following questions.

1. For parallel computing, the optimal speedup gained through parallelization is linear with respect to the number of jobs running in parallel. For example, with 5 reducers, ideally we would expect parallel computing to take 1/5 wall clock time of single machine run. However, this optimal speedup is usually not achievable. In this question, set the number of reducers in your hadoop run to 2, 4, 6, 8, 10, and record the wall clock time. Plot a curve, where the horizontal axis is the number of reducers, and the vertical axis is the wall time. Is the wall time linear with respect to the number of reducers? Explain what you observed.
2. Right now we're basically ignoring the fact that there are multiple labels on the training and testing sets. How would you extend your algorithm to predict multiple labels?

Have your classification code print out the results and put them at the end of the `README` file, including the log probabilities of the best class y:

$$\ln(Pr(Y = y)) + \sum_i \ln(Pr(W = w_i | Y = y)) \tag{4.1}$$

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 Eq. 4.1), 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%
```

[4]

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).

5 Marking breakdown

- Code correctness (commit messages, program output, and the code itself) **[70 points]**
- Question 1 **[15 points]**
- Question 2 **[15 points]**

6 Other stuff

You may consider using the `GenericOptionsParser` for passing command line arguments. It parses all arguments that have the form `-D name=value` and turns them into name/value pairs in your `Configuration` object.

START EARLY. There is not a lot of actual code required (a few hundred lines total), but it can take much longer than you think to debug and fix problems you encounter. If you need a development strategy, check out William Cohen's words of advice for his CMU big data course: `http://curtis.ml.cmu.edu/w/courses/index.php/Guide_for_Happy_Hadoop_Hacking`. You can also send messages to the mailing list if you're stuck; chances are, someone else is having the same issue.