BigML Assignment 2: Small Memory Footprint Streaming Naive Bayes

Due Wed, Feb 5 before class (1:29pm) via Autolab
Late submission: 50% credit before Fri, Feb 7, 1:29pm via Autolab

1 Important Note

This assignment is the second of three that use the Naive Bayes algorithm. You will be expected to reuse the code you develop for this assignment for a future assignment, and **you are expected to use Java 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 with limited memory

The algorithm for this assignment is exactly the same as for assignment 1, but now we will constrain the size of the memory footprint. All Java commands must be followed with **-Xmx128m** which limits the size of the Java heap space. Also, there will be NO required tokenizer in the assignment, so feel free to use your own. In this homework, you are allowed to write the temporary counts to disk to make your streaming counts fit in the memory.

3 The Data

For this assignment, we will be classifying Wikipedia articles by the languages in which they are also available. The data appears at `/afs/cs.cmu.edu/project/bigML/dbpedia/`

The data format is the same as for Assignment 1. There is one instance per line. The first token of each line is the (comma separated) list of class names , then a tab, then the data. Please do multi-label learning and evaluation as described for Assignment 1.

There are 6 data sets for this assignment. Again, they are in increasing size so that you can debug your code on smaller data. The files that start with *abstract* include Wikipidia text. The files that start with *links* are the outlinks from each wikipedia page.

```
abstract.small.test
abstract.small.train
```

```
abstract.test
abstract.train
abstract.tiny.test
abstract.tiny.train
links.small.test
links.small.train
links.test
links.train
links.tiny.test
links.tiny.train
```

4 Deliverables

Submit your implementations via AutoLab. You should implement the algorithm by yourself instead of using any existing machine learning toolkit. You should upload your code (including all your function files) along with a **report**, which should solve the following questions:

1. Compare and discuss the performance for the full links vs full abstract data. (5 points)

2. Using a local copy of the RCV1.small_train.txt file from Assignment 1, compare the performance of creating your Naive Bayes feature dictionary for last assignment and this assignment. Time all parts of the dictionary creation (including, for example, sorting and combining counts for your Assignment 2 solution). Average over 10 calls. Please include in your write up the commands you used to do this comparison. (10 points)

3. How can we get the most informative features of a specific class given the trained naive Bayes model? In other words, if we are interested in figuring out the most predictive features for a class of wikipedia articles (e.g. German), what can we do? This is an open question. (5 points)

4. Answer the questions in the collaboration policy on page 1.

The training code NBTrain.java should be able to run without the out-of-memory issue, using the command:

```
cat train.txt | java -Xmx128m NBTrain
```

This will output the streaming counts for the NB model, in the tab-separated two column form that was specified in the previous homework. The test code provide a one-per-line prediction for each test case, so the full streaming command is:

```
cat train.txt | java -Xmx128m NBTrain | sort -k1,1 | \
java -Xmx128m MergeCounts | java -Xmx128m NBTest test.txt
```

Here, MergeCounts.java is a function that you need to write to merge the counts of multiple occurrances of the same hash key. The test code produces output in the following format:

```
Best Class<tab>Log prob
```

which is essentially the classification results, including the log probabilities of the best class y:

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

Here, Best Class is the class with the maximum log probability (as in Equation 1), and the last field is the log probability. L is the length of the testing document. Note that we're using the natural logarithm here. Here's an example of the output format:

```
pt  -1042.8524
ru  -4784.8523
...
```

You should tar gzip the following items into **hw2.tgz** and submit to the homework 2 assignment via Autolab:

- NBTrain.java
- MergeCounts.java
- NBTest.java
- and all other auxiliary functions you have written
- report.pdf

Tar gzip the files directly using "tar -cvf hw2.tgz *.java report.pdf". Do **NOT** put the above files in a folder and then tar gzip the folder. You do not need to upload the saved temporary files. Please make sure your code is working fine on linux.andrew.cmu.edu machines before you submit.

5 Submission

You must submit your homework through Autolab via the "Homework2: Small Memory Streaming Naive Bayes" link. In this homework, we provide an additional tool called "Homework2-validation":

- Homework2-validation: You will be notified by Autolab if you can successfully finish your job on the Autolab virtual machines. Note that this is not the place you should debug or develop your **algorithm**. All development should be done on linux.andrew.cmu.edu machines. This is basically a Autolab debug mode. There will be **NO** feedback on your **performance** in this mode. You have unlimited amount of submissions here. To avoid Autolab queues on the submission day, the validation link will be closed 24 hours prior to the official deadline. If you have received a score of 1000, this means that you code has passed the validation.
- Homework2: Small Memory Streaming Naive Bayes: This is where you should submit your validated final submission. You have a total of **15 possible submissions**[1]. Your performance will be evaluated, and feedback will be provided immediately.

6 Grading

You will be graded based on the memory usage (25 points) and runtime (25 points) for your training code, the validation of your testing code (5 points), and your final test performance (25 points). Note that the datasets in HW2 could be much more difficult than previous homework[2]. And there could be tradeoffs among the memory usage, runtime, and the test performance, so it is very unlikely that one will receive a total as high as the previous homework. The report will be graded manually (20 points).

7 Hints/Good to know

Unix sort can handle very large files. This is helpful when you need to collect together the counts for a particular key. To ensure sort behaves properly, you should set the following environment variable:

```
LC_ALL='C'
```

Get it to sort using tabs by setting this flag:

```
-t $'\t'
```

And, when files are large it may be a good idea to tell sort where to store its temporary files:

```
-T /some/dir
```

[1] There are obviously some tuning efforts in this homework, so we relaxed the limits for this assignment.
[2] For example, comparing to hw1, there are more class labels in this homework.