# 10605 BigML Assignment 1(b): Naive Bayes with Hadoop API

Due: Thursday, Sept. 15, 2016 23:59 EST via Autolab

August 2, 2017

## Policy on Collaboration among Students

These policies are the same as were used in Dr. Rosenfeld's previous version of 10601 from 2013. The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes are shared, or are taken at that time, and provided learning is facilitated, not circumvented. The actual solution must be done by each student alone, and the student should be ready to reproduce their solution upon request. The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved, on the first page of their assignment. Specifically, each assignment solution must start by answering the following questions in the report:

- 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".

Collaboration without full disclosure will be handled severely, in compliance with CMU's Policy on Cheating and Plagiarism. As a related point, some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions. Avoiding the use of heavily tested assignments will detract from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be (or may have been) available online, or from other people. It is explicitly forbidden to use any such sources, or to consult people who have solved these problems before. You must solve the homework assignments completely on your own. I will mostly rely on your wisdom and honor to follow this rule, but if a violation is detected it will be dealt with harshly. Collaboration with other students who are currently taking the class is allowed, but only under the conditions stated below.

## 1 Important Note

As usual, **you are expected to use Java for this assignment**.

**This assignment is worth 100 points**. Similar to part (a), this part (b) is also a relatively small assignment because you will be able to reuse some of the code from previous Naive Bayes assignments. However, unlike the Hadoop streaming settings in part (a), in this assignment, you will have to port your naive Bayes code to the real Hadoop environment using Hadoop APIs to train a naive Bayes classifier.

Lanxiao Xu (lanxiaox@andrew.cmu.edu) and Chenran Li (chenranl@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 Introduction

In this part of the assignment, you need to re-implement naive Bayes for the Hadoop MapReduce framework. Similar to part (a), you only need to write Hadoop naive Bayes training, and you do not need to care about the testing part.

### 2.1 Using AWS and elastic MapReduce (EMR)

We have already distributed the AWS gift code to every registered student. If you have not got one, let us know. Here are a few hints for running the real Hadoop jobs on AWS.

#### 2.1.1 Submitting a Jar job

Important: unlike homework 1a, in this part b, after setting up your EMR cluster on AWS, you will run your job in the Hadoop API mode on AWS.

#### 2.1.2 Viewing job progress

Tutorial for viewing the jobtracker on your local machine (via proxy) [1]. (You can also ssh into the machine using the command line interface, and then use the Linux commands in the login preamble to view the job tracker.)

[1]http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/UsingtheHadoopUserInterface.html

### 2.2 Debugging with the CMU Hadoop cluster

You have access to the OpenCloud cluster from the PDL team at CMU. You should be receiving an email soon with your login details. To login to the cluster, make sure you are on a CMU network, or using the vpn, and then ssh into shell.stoat.pdl.local.cmu.edu. Once there, you can run hadoop commands.

To test your code: First, create a jar of your Naive Bayes implementation. You need this for AWS as well. Once you have your jar, you can scp it to the cluster and run it with the following command:

```
hadoop jar name-of-jar.jar path/to/input path/to/output num_reducers
```

Note that the path to input and output is on HDFS, not the local machine. This will read input from HDFS and store output on HDFS. To upload your training set to HDFS, use "`hadoop fs -put path/to/data.txt`". For a more exhaustive list of commands, see

http://hortonworks.com/hadoop-tutorial/using-commandline-manage-files-hdfs/.

Another option is to install and run Hadoop on your local machine. While this isn't streaming, it will allow you to test your implementation.

**Note**: since the java version on OpenCloud is 1.6, the recommended way is to upload your code and the package `hadoop-core-1.0.1.jar` to the cluster. Suppose you are in the directory where there are your java source files, first invoke

```
javac -cp path/to/hadoop-core-1.0.1.jar:. *.java
```

to compile your code. Suppose your main class is "run", then type

```
jar cfe NB_train.jar run *.class
```

to generate the jar file. And finally type

```
hadoop jar NB_train.jar path/to/input path/to/output num_reducers
```

to run your job.

### 2.3 Additional Hadoop Tutorial

In case you want to study extra tutorials about Hadoop, your honorary TA Malcolm Greaves has kindly put together a wiki page here:

http://curtis.ml.cmu.edu/w/courses/index.php/Guide$_{for_{H}appy_{H}adoop_{H}acking}$.

### 2.4 About Java Version

The Java version is 1.6 on OpenCloud, and 1.8 on Autolab. So you can write your code using Java 1.6 API and submit the same copy of your code both to OpenCloud and Autolab. As for AWS, the default Java version depends on the specific machine you choose.

## 3 The Data

We are using a new dataset extracted from DBpedia. The labels of the article are mapping-based types of the document. There are in total 18 (17 + other) classes in the dataset, and they are from the first level class in DBpedia ontology.

### 3.1 Data Format

The format is one document per line. Each line contains three columns which are separated by a single tab:

- a document id
- a comma separated list of class labels
- document words

The documents are preprocessed so that there are no tabs in the body.

### 3.2 Obtaining the Data

The full dataset for assignment 1b is loacated at /afs/cs.cmu.edu/project/bigML/dbpedia$_1$6$fall/abstract.full.$

Note that you only need to submit the log files for running your code on the **full dataset**. The small dataset is provided to you for debugging as usual[2]. Similar to homework 1a, please use the provided tokenizer from homework 1a.

## 4 Deliverables

### 4.1 Steps

What you need to do in this assignment can be summarized in the following steps:

- Port the naive Bayes training code into Hadoop using Hadoop's MapReduce API.
- Run the Hadoop API MapReduce job on AWS with the **full dataset** with elastic MapReduce using the Custom Jar option. (**Note**: use only one reducer here.)
- Download the controller and syslog text files, and submit via Autolab together with the report and your source code in a tar ball.

**Hint**: The controller and syslog files from AWS for the mapreduce job can be downloaded from the AWS console. Simply go to the Elastic Mapreduce tab. Select your job in the list, click View details and expand Steps to see jobs and log files.

[2]Note that Amazon will charge you a full hour for small jobs less than one hour, so you may not want to develop or debug your code on AWS.

## 4.2 Report

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

2. 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. (The wall clock time of reducers can be inferred from syslogs - use the time between "map 100% reduce 0%" and "map 100% reduce 100%"). 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. (15 points)

3. You are performing the classical WordCount example on Hadoop MapReduce. Words in real languages follow a power-law distribution - the frequency of words is highly non-uniform and skewed, and some words occur much more in the corpus than other words. Consider an extreme case of skew: for instance, assume that 10% of the words in the document are copies of the word "the". Will this adversely impact the performance of your MapReduce job? Why or why not? If yes, suggest a fix for the same. (10 points)

4. In **information retrieval**, sometimes we need to know the most relevant documents in a corpus given a search query. This can be done by calculating the relevance score of each document to that query, and ranking the documents according to their scores.

   **Okapi BM25** is a function used to calculate relevance scores of a given query and a set of documents in a corpus. Given a query $Q$, containing keywords $q_1, \ldots, q_n$, the BM25 score of a document $D$ is:

   $$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{\text{TF}(q_i, D) \cdot (k_1 + 1)}{\text{TF}(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \quad ,$$

   where $\text{TF}(q_i, D)$ is $q_i$'s term frequency in the document $D$, $|D|$ is the length of the document $D$, and avgdl is the average document length in the corpus. $k_1$ and $b$ are free parameters, which can be treated as constants here. $\text{IDF}(q_i)$ is the inverse document frequency weight of the query term $q_i$. It can be computed as:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \quad ,$$

where $N$ is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing $q_i$.

Now given a query $Q$, containing keywords $q_1, \ldots, q_n$ and a corpus like below (With two columns separated by tab, of which the first is the document id, and the second is the content. You may assume there is no tab in the content, all words are in lower case and any one of the documents can fit in memory):

```
....
d1 \t russian police raid rights group memorial and other...
d2 \t the us embassy in russia has asked the russian government...
....
```

Outline how can you employ stream and sort pattern to calculate BM25 score of each document. The output should look like below (you don't have to sort the documents according to the scores):

```
....
(d1, score(d1, Q) = 0.08)
(d2, score(d2, Q) = 0.09)
....
```

You should write down the input and output format of each step. (15 points)

**Hint**: think of how you can get the BM25 score of each document w.r.t each keyword in the query (because the score of each document w.r.t. the query can be calculated by adding up all the scores of that document w.r.t each keyword in the query). For example, if you have:

```
....
(q1, d1, score(d1, q1) = ?)
(q1, d2, score(d2, q1) = ?)
(q1, d3, score(d3, q1) = ?)
....
(q2, d1, score(d1, q2) = ?)
(q2, d2, score(d2, q2) = ?)
```

```
(q2, d3, score(d3, q2) = ?)
....
```
it will be much easier to get the final result.

## 4.3 Autolab Implementation details

Autolab is currently running Hadoop 1.0.1. In this part of homework, you will need to follow the exact naming of the following files.

You must have the **run.java** class, which includes the main function that you call the MapReduce version of the naive Bayes trainer (e.g. NB_train_hadoop.java). There will be three arguments sent to this main function: **InputPath**, **OutputPath**, and **the number of reduce tasks**. For example, in a standalone debugging version of Hadoop, your code need to execute sucessfully via the following commands:

```
javac -cp hadoop-core-1.0.1.jar:. *.java;
java -cp hadoop-core-1.0.1.jar:hadoop/lib/*:. run InputPath OutputPath 1
```

Important: you should still use the tokenizer provided in homework 1a, and also the key output format mentioned in homework 1a. For example:

```
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
...
```

But this time Hadoop will do the sorting job for you.

You should tar the following items into **hw1b.tar** and submit to the homework 1b assignment via Autolab:

- run.java
- NB_train_hadoop.java
- controller.txt
- syslog.txt
- and all other auxiliary functions you have written
- report.pdf

Tar the files directly using "tar -cvf hw1b.tar *.java *.txt report.pdf". Do **NOT** put the above files in a folder and then tar the folder. You do not need to upload the saved temporary files.

## 5 Submission

You must submit your homework through Autolab. In this part of homework 1, there will be a validation link.

- HW1b-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 with no errors, this means that you code has passed the validation.
- HW1b-Hadoop-Naive-Bayes: This is where you should submit your tar ball. You have a total of **10 possible submissions**. Your score will be reported, and feedback will be provided immediately.

## 6 Grading

- If you are able to successfully run the job on full dataset with AWS EMR, you will receive 20 points. The successful run of your AWS job should be reflected in your submitted log files. Do not submit someone elses log files. Autolab runs a cheat checker, and we have zero tolerance for any cheating in the course.
- We will test your Hadoop code on Autolab in real time, and check the logs and the correctness of your MapReduce output. In particular, all the counts that your MapReduce program generates for the Naive Bayes model will be compared to reference counts to determine correctness of your program. This will count for 40 points. As stated earlier, do not submit anyone elses code including that of past students of the course. Such cheating will be caught by Autolab and the consequences for cheating are severe.
- The report will be graded manually and its questions carry 40 points.