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

Due: Thursday, Sept. 14, 2017 23:59 EST via Autolab

September 7, 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

This semester, **you are expected to use Python for this assignment**.

This assignment is worth 100 points. You will be able to reuse some of the code from the previous Naive Bayes assignment. However, unlike homework 1a, in this assignment, you will have to port your naive Bayes code to the real Hadoop environment, using the Hadoop streaming APIs to train a naive Bayes classifier.

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 Piazza, and the instructors can be reached at the following email address: *10605-Instructors@cs.cmu.edu*.

2 Naive Bayes in Hadoop

There are two parts in this assignment. In the first part, you need to re-implement Naive Bayes training in the Hadoop MapReduce framework. In the second part, you need to implement Naive Bayes testing in the Hadoop MapReduce framework. The second part counts for 20 bonus points.

2.1 Using AWS and elastic MapReduce (EMR)

Unlike homework 1a, in this part b, after setting up your EMR cluster on AWS, you will run your job in the Hadoop Streaming API mode on AWS[1]. To run your Python Hadoop jobs on AWS, You can follow the instruction document released on Piazza. We will distribute an AWS gift code to every registered student. If you don't get one, let us know.

2.2 Python MapReduce Code

We will use the Hadoop Streaming API for passing data between our Map and Reduce code via STDIN (standard input) and STDOUT (standard output). We will simply use Pythons sys.stdin to read input data and print our own output to sys.stdout. Thats all we need to do because Hadoop Streaming will take care of everything else.

[1]Note that Amazon will charge you a full hour for small jobs less than one hour.

2.3 Debugging in local machine

You should test your mapper and reducer scripts locally before using them in a real Hadoop environment. Otherwise your jobs might successfully complete but there will be no job result data at all or not the results you would have expected. You can simply use unix sort to replace the hadoop streaming module. For example, your command on local machine could run like this:

```
cat data.txt | python mapper.py | sort | python reducer.py
```

2.4 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. Check `https://wiki.pdl.cmu.edu/Stoat` for cluster usage.

To test your code, you can scp your Python scripts, the package `hadoop-streaming.jar` and dataset to the cluster and run with the following command:

```
hadoop jar local/path/to/hadoop-streaming.jar \
       -file local/path/to/mapper.py    -mapper local/path/to/mapper.py \
       -file local/path/to/reducer.py   -reducer local/path/to/reducer.py \
       -input hdfs/path/to/input       -output hdfs/path/to/output \
       -numReduceTasks num_reducers
```

You can download the package `hadoop-streaming.jar` here and rename it. 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. The output path should be a non-existing folder in HDFS. To upload your training set to HDFS, use "`hadoop fs -put path/to/data.txt`". To copy your output from HDFS, use "`hdfs dfs -get hdfs/path/to/result.txt`". For a more exhaustive list of commands, see

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

2.5 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_Happy_Hadoop_Hacking`.

2.6 About the Python Version

The Python version is 2.7.3 on OpenCloud, and 2.7.5 on Autolab. So you can write your code using Python 2.7 and submit the same copy of your code both to OpenCloud and Autolab. As for AWS, the default Python version depends on the specific machine you choose.

3 The Data

We are using a dataset extracted from DBpedia. The labels of the article are based on the types of the document. There are in total 17 (16 + other) classes in the dataset, and they are from the first level class in DBpedia ontology.

3.1 Data Format

The training data 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 testing data format is one document per line. Each line contains two columns which are separated by a single tab:

- a document id
- 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 located at `/afs/cs.cmu.edu/project/bigML/dbpedia_17fall`.You can also download data here `https://console.aws.amazon.com/s3/buckets/cmu-10605/?region=us-east-1` on S3. . For your convenience, the input path you need to fill in when launching a streaming job on EMR is `s3://cmu-10605/data/...`

Several smaller datasets are provided to you for debugging as usual. **Please use the tokenizer provided in homework 1a**.

4 Tasks

4.1 Part 1: Naive Bayes Training on Hadoop (70 Points)

In part 1, you are supposed to re-implement Naive Bayes training in Hadoop Mapreduce, and in streaming way. What you need to do can be summarized in the following steps:

- Port the naive Bayes training code into Python MapReduce Code. Organize it into **mapper.py** and **reducer.py**.
- Run the Hadoop MapReduce job on AWS with the **full dataset** with elastic MapReduce using the Streaming Program option. (**Note**: use only one reducer here.)
- Download the output file. Write a script to get the top 10 words for each class with most word counts. You should write your result into a file called top10.txt in the following format:

  Class<tab>Word<tab>Count

- Download the controller and syslog text files.
- Submit a tar ball via Autolab containing your source code and results. Check details in Deliverables Section.

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.

4.2 Part 2: Naive Bayes Testing on Hadoop (20 EXTRA Points)

Warning from TAs (not from compiler, so better not ignore it) It's wise to finish other tasks before marching on the bonus. It usually takes you far more time than part 1. This bonus would not benefit you in grades, if you already got an A+ at the end of semester. If you are still curious and interested to explore, go ahead!

In part 2, you are supposed to implement Naive Bayes testing in streaming way. You can refer to slides and Professor Cohen's notes(`http://www.cs.cmu.edu/~wcohen/10-605/notes/scalable-nb-notes.pdf`). What you need to do is summarized as following:

- Write a pipeline that accepts training set and test set filename as input. The output should be your classification results for the original document in test set, where each row contains two tab-separated fields - headline and predicted label.

  Headline<tab>Label

The pipeline could consist of multiple mappers and reducers. For local/Autolab test purpose, write a Makefile to automate the whole process. With running the following command, your pipeline gets the path passed in arguments **train, test** as input, and output the result to the **output**.

```
make run train=../data/train/abstract.tiny.train \
test=../data/test/abstract.tiny.test output=tiny_output
```

Below is a simple reference Makefile. It's just for people who don't have experience writing Makefile, and is not what your Makefile should look like! You may have different number of mappers and reducers in your pipeline.

```
run:
<tab>cat ${train} | python mapper_1.py | sort | python reducer_1.py > tmp_1;
<tab>cat ${test} | python mapper_2.py | sort | python reducer_2.py > tmp_2;
<tab>cat tmp_1 tmp_2 | do something... > ${output};
```

- Optionally run your code on the full dataset on AWS. You don't need to submit any result to Autolab.
- Grades in part 2 will be based on accuracy of your classification results. So feel free to explore various tricks to improve the accuracy. For example, try removing stop words, filtering rare words, adjusting smoothing parameters, etc.
- You are required to implement it in streaming way as it is in Part 1. That is, you can only use constant memory.

4.3 Deliverables

In this homework, you will need to follow the exact naming of files.

For part 1, You must have the mapper.py and reducer.py. Your MapReduce program need to execute the following commands successfully:

```
cat abstract.full.train | python mapper.py | sort | python reducer.py
```

Important: you should still use the tokenizer provided in homework 1a, and output in the format of:

```
Y=* 200
Y=Agent 100
Y=Agent,W=* 1000
Y=Agent,W=her 10
Y=Agent,W=duck 5
...
```

For part 2, you could design your own MapReduce pipelines, but you should provide a Makefile as specified in Section 4.2. What your pipeline writes into destination file would be something like (XXX is your predicted label):

```
%C3%85selistraumen_Bridge XXX
%C5%81upiny,_Masovian_Voivodeship XXX
191_Peachtree_Tower XXX
1950_Nebraska_Cornhuskers_football_team XXX
1952_Roller_Hockey_World_Cup XXX
...
```

To summarize, you should tar the following items into **hw1b.tar** and submit to the homework 1b assignment via Autolab:

- mapper.py
- reducer.py
- controller.txt
- syslog.txt
- top10.txt
- Makefile and all Python MapReduce codes for part 2

Tar the files directly using

```
tar -cvf hw1b.tar *.py *.txt Makefile
```

Do **NOT** put the above files in a folder and then tar the folder. You do not need to upload the saved temporary files. Remember also to submit the report to a separate Autolab link (More in Submission Section).

4.4 Report (30 Points)

For this assignment, you need to submit a **report**, which answers 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 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 first

log line with "map 100%" 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. (10 points)

3. 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.

   Assume you are given a query Q, containing keywords $q_1, \ldots, q_n$ and a corpus like the one 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 the stream and sort pattern to calculate the 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)
....
```

Try to implement an algorithm with 3 mapreduces. You should write down the output of each mapper or reducer. (20 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.

5 Submission

You must submit your homework through Autolab. There are 3 entries in Autolab - Validation, Submission and Report.

- 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**. 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.

- HW1b:Report - 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

- Part 1 counts up to 70 points. If you are able to successfully run the job on full dataset with AWS EMR and get correct statistics, you will receive 40 points. The successful run of your AWS job should be reflected in your submitted log files (20 points) and correct top10.txt result (20 points).

  We will also test your Python MapReduce code on Autolab. You will receive another 30 points if all the counts that your MapReduce program generates for the Naive Bayes model are same as reference .Do not submit someone else's files or codes including that of past students of the course. Autolab runs a cheat checker, and we have zero tolerance for any cheating in the course.

- Part 2 counts up to 20 points as bonus. You will be graded based on your accuracy in test set.

- The report will be graded manually and its questions are worth 30 points.