BigML Assignment 4: Naive Bayes and Phrases on Hadoop

Naive Bayes Part Due Wednesday, February 27, 2013 via Blackboard
Phrase Finding Part Due Wednesday, March 6, 2013 via Blackboard

Out February 13, 2013

1 Important Note

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

This assignment is worth a total of 200 points because there are conceptually two assignments here: the Hadoop implementation of Naive Bayes (Assignment 2) and the Hadoop implementation of Phrase finding (Assignment 3). If you choose not to do this assignment, you will be counted as missing **two** assignments. If you choose to do only one of the two assignments, either Hadoop Naive Bayes, or Hadoop phrase finding, you will be counted as missing **one** assignment.

Bin Zhao (binzhao@cs.cmu.edu) is the contact TA for Hadoop naive Bayes, and Yanbo Xu (yanbox@cs.cmu.edu) is the contact TA for Hadoop phrase finding. Please post clarification questions to the Google Group:
`machine-learning-with-large-datasets-10-605-in-spring-2013`

2 Introduction

For this assignment, you will re-implement Phrase finding and Naive Bayes (Assignments 2 and 3) for the Hadoop Mapreduce framework. You are free (and encouraged) to reuse your code from those assignments.

3 Getting started

3.1 Installing Hadoop Virtual Appliance (Optional)

You may install a Hadoop virtual appliance on your own machine for ease of debugging, however, this step is optional, and you can directly run your code on Andrew cluster or AWS.

Cloudera has offered a single node virtual appliance. Follow the instructions on the page
`https://ccp.cloudera.com/display/SUPPORT/Cloudera%27s+Hadoop+Demo+VM+for+CDH4`

to download the image and also VMware Player (for Windows and Linux) or VMware Fusion (for Mac). Start up the VM. Inside the VM, Hadoop is running in what's called "pseudo-distributed mode", which means that all the daemon processes are running on the same machine and communicating via loopback.

3.2 Using AWS

We have distributed AWS credits to every registered student. If you have not got one, let us know. Here are a few hints for running jobs on AWS:

3.2.1 Submitting a Jar job

The easiest way to submit jobs to AWS is via the commandline. You need to install the elastic mr command line interface - there are many versions for many different languages. I used the ruby one and found it to be painless to install/use

`http://aws.amazon.com/code/Elastic-MapReduce/2264`

You can also start a job with the AWS console's web interface - go to the elastic mapreduce tab in your AWS console, click create new jobflow, choose the custom JAR jobtype and the rest is pretty self-explanatory. I find this takes longer than the command line, but if youre more comfortable with a GUI, that's fine.

3.2.2 Viewing job progress

Tutorial for viewing the jobtracker on your local machine (via proxy)

`http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/UsingtheHadoopUserInterface.html`

(You can also ssh into the machine using the command line interface, and then use the lynx commands in the login preamble to view the job tracker)

4 The Data

We are using the data from Assignments 2 & 3. They are still on AFS, and they are also in this public s3 bucket: `s3://bigml-shared/phrases` and `s3://bigml-shared/webdocs`

The data format is the same as for Assignments 2 & 3, with one small change: The phrase data has format:

```
<text>\t<year>\t<text>\t<year>\t<count>
```

Where text is the unigram/bigram, count is the number of times the text occurred in books during the decade indicated by year. This change is a byproduct of changing a SequenceFile to a TextFile. If you ignore the first two fields of the tab-separated line, the format is the same as you saw for Assignment 3.

5 One Small Change

In the interest of freshness, this time, use the bigrams/unigrams from the 1960s as your corpus, and those from 1970-1990 as the background corpus.

6 Deliverables

Submit a compressed archive (zip, tar, etc) of your code. Please also include the controller and syslog files **from AWS** for each of the mapreduce jobs in both of your pipelines. The controller and syslog files can be downloaded from the AWS console. Simply go to the Elastic Mapreduce tab. Click your job in the list, and then the debug button, the syslog link, and save the resulting file. Please include a pdf document with answers to the questions below.

Part A: Naive Bayes

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. Set the number of reducers to 10, and run your hadoop job on AWS. Tell us how much it costs to run your job on AWS.

For the Naive Bayes pipeline, include the controller and syslog files created when you process the full abstracts dataset. Create a folder called `naivebayes`. In it, put the controller and syslog files from each of your Mapreduces, named syslogN and controllerN for $N = 1..n$ where n is the number of Mapreduce jobs in your pipeline. Create a README file briefly describing each of the steps of your pipeline, numbered in the same order as the controller/syslog files.

Part B: Phrase Finding

1. What are the top 20 phrases (sorted by *total score*) from the *full data set only* (again, with the 1960s as the foreground corpus this time).

2. What do your phrase finding results tell you about the 1960s? Are there any interesting phrases?

3. Tell us about your experience using Hadoop for this portion of the assignment. Was it easy to port your code over from the Assignment 3? Is there anything that would have made the process easier?

For the Phrases pipeline, include the controller and syslog files created when you process the full dataset. Create a folder called `phrases`. In it, put the controller and syslog files from each of your mapreduces, named syslogN and controllerN for $N = 1..n$ where n is the number of mapreduce jobs in your pipeline. Create a README file briefly describing each of the steps of your pipeline, numbered in the same order as the controller/syslog files.

7 Marking breakdown

7.1 Naive Bayes

- Code correctness [**75 points**].
- Part A: Question 1 [**10 points**]
- Part A: Question 2 [**5 points**]
- Controller/Syslog files [**10 points**]

7.2 Phrase Finding

- Code correctness [**70 points**].
- Part B: Question 1 [**10 points**]
- Part B: Question 2 [**5 points**]
- Part B: Question 3 [**5 points**]
- Controller/Syslog files [**10 points**]

8 Hints/Good to know

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.