Michael Gyarmathy
CSCE 470
10/15/14

Homework 3 Report

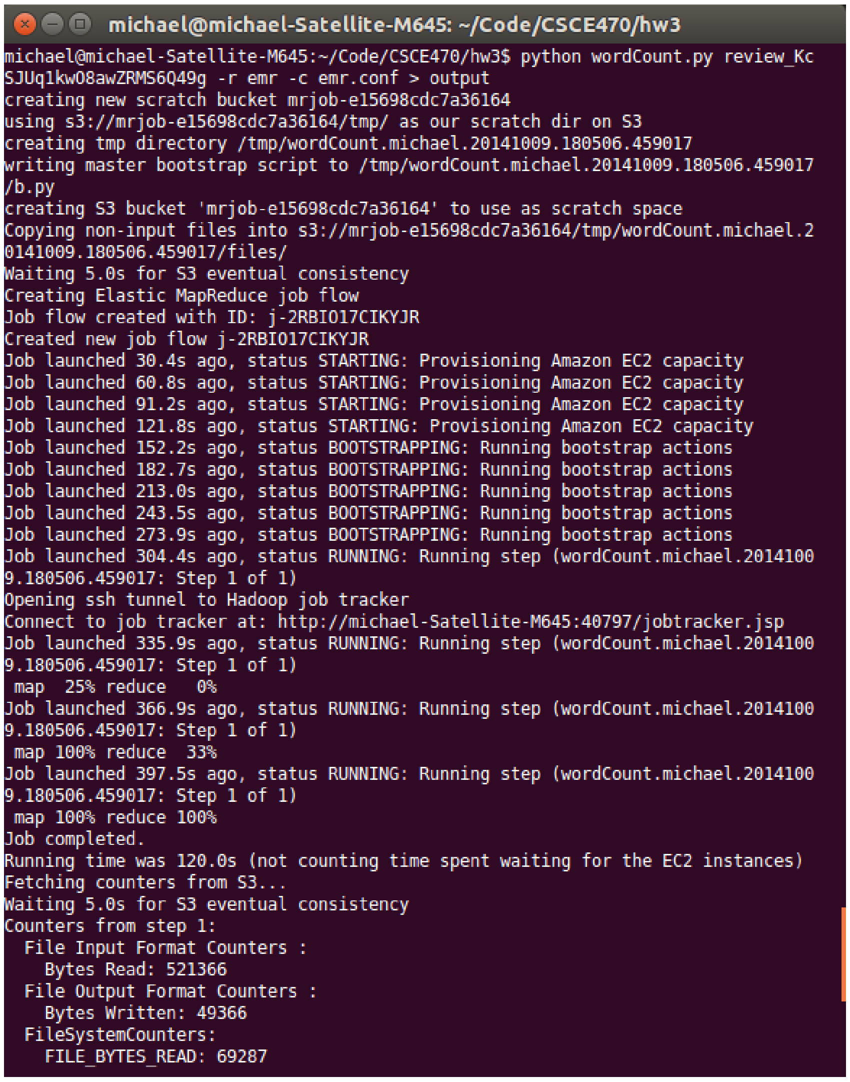
Task 1.2: Getting Started with mrjob

Ran `python wordCount.py review_KcSJUq1kwO8awZRMS6Q49g > output1-2` successfully.

Task 1.3: mrjob on EMR

Ran `python wordCount.py review_KcSJUq1kwO8awZRMS6Q49g -r emr -c emr.conf > output1-3` successfully.

`diff output1-2 output1-3` returns no difference.



```
michael@michael-Satellite-M645:~/Code/CSCE470/hw3$ python wordCount.py review_Kc
SJUq1kwO8awZRMS6Q49g -r emr -c emr.conf > output
creating new scratch bucket mrjob-e15698cdc7a36164
using s3://mrjob-e15698cdc7a36164/tmp/ as our scratch dir on S3
creating tmp directory /tmp/wordCount.michael.20141009.180506.459017
writing master bootstrap script to /tmp/wordCount.michael.20141009.180506.459017
/b.py
creating S3 bucket 'mrjob-e15698cdc7a36164' to use as scratch space
Copying non-input files into s3://mrjob-e15698cdc7a36164/tmp/wordCount.michael.2
0141009.180506.459017/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-2RBIO17CIKYJR
Created new job flow j-2RBIO17CIKYJR
Job launched 30.4s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 60.8s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 91.2s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 121.8s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 152.2s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 182.7s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 213.0s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 243.5s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 273.9s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 304.4s ago, status RUNNING: Running step (wordCount.michael.2014100
9.180506.459017: Step 1 of 1)
Opening ssh tunnel to Hadoop job tracker
Connect to job tracker at: http://michael-Satellite-M645:40797/jobtracker.jsp
Job launched 335.9s ago, status RUNNING: Running step (wordCount.michael.2014100
9.180506.459017: Step 1 of 1)
 map  25% reduce   0%
Job launched 366.9s ago, status RUNNING: Running step (wordCount.michael.2014100
9.180506.459017: Step 1 of 1)
 map 100% reduce  33%
Job launched 397.5s ago, status RUNNING: Running step (wordCount.michael.2014100
9.180506.459017: Step 1 of 1)
 map 100% reduce 100%
Job completed.
Running time was 120.0s (not counting time spent waiting for the EC2 instances)
Fetching counters from S3...
Waiting 5.0s for S3 eventual consistency
Counters from step 1:
  File Input Format Counters :
    Bytes Read: 521366
  File Output Format Counters :
    Bytes Written: 49366
  FileSystemCounters:
    FILE_BYTES_READ: 69287
```

task1-3.png - AWS EMR up and running!

Michael Gyarmathy
CSCE 470
10/15/14

Task 1.4: The Yelp Dataset Challenge

I had issues configuring mrjob and AWS to allow me to use http://s3.amazonaws.com/csce470/review.json, so I downloaded the review file onto my machine, uploaded it to my own S3 bucket, and used that bucket as my input path:

```
python wordCount.py s3://review-json/review.json -r emr -c emr.conf >
output1-4
```

To process the output from the EMR job, I wrote a short program to sort the results by term frequency and return the top-10 most frequent terms:

```
python task1-4.py output1-4
```

```
[('the', 7518615), ('and', 4815331), ('i', 4443261), ('a', 3835806),
('to', 3452557), ('was', 2462867), ('it', 2311357), ('of', 2254268),
('is', 1790944), ('for', 1707172)]
```

Expanding this list out further, the top-3 most frequent terms which are not considered stop words are: `[('good', 696455), ('place', 664741), ('food', 636999)]`.

Task 2.1: Basic Similarity

```
python task2-1.py "Query 1" "Query 2" N
```
(e.g. python task2-1.py “Microsoft CEO” “Satya Nadella” 1)

Query 1	Query 2	N	Jaccard Similarity
Microsoft CEO	Satya Nadella	1	0.0833333333333
Microsoft CEO	Satya Nadella	10	0.26618705036
Microsoft CEO	Satya Nadella	50	0.18961038961
Microsoft CEO	Bill Gates	1	0.025641025641
Microsoft CEO	Bill Gates	10	0.0805369127517
Microsoft CEO	Bill Gates	50	0.0827067669173
US President	George Bush	1	0.107142857143
US President	George Bush	10	0.0964467005076
US President	George Bush	50	0.0931506849315

US President	Barack Obama	1	0.0909090909091
US President	Barack Obama	10	0.0909090909091
US President	Barack Obama	50	0.126200274348
Barack Obama	George Bush	1	0.0
Barack Obama	George Bush	10	0.0602409638554
Barack Obama	George Bush	50	0.0906593406593
football	soccer	1	0.0277777777778
football	soccer	10	0.0490797546012
football	soccer	50	0.102605863192
aggies	longhorns	1	0.0
aggies	longhorns	10	0.0743243243243
aggies	longhorns	50	0.10284463895
aggies	kevin sumlin	1	0.0
aggies	kevin sumlin	10	0.111111111111
aggies	kevin sumlin	50	0.140529531568

Task 2.2: Improved Similarity

```
python task2-2.py "Query 1" "Query 2" N
```
(e.g. python task2-1.py “Microsoft CEO” “Satya Nadella” 1)

Query 1	Query 2	N	Improved Similarity
Microsoft CEO	Satya Nadella	1	0.0526315789474
Microsoft CEO	Satya Nadella	10	0.25
Microsoft CEO	Satya Nadella	50	0.290322580645
Microsoft CEO	Bill Gates	1	0.0
Microsoft CEO	Bill Gates	10	0.0

Microsoft CEO	Bill Gates	50	0.025641025641
US President	George Bush	1	0.04
US President	George Bush	10	0.025641025641
US President	George Bush	50	0.025641025641
US President	Barack Obama	1	0.0416666666667
US President	Barack Obama	10	0.025641025641
US President	Barack Obama	50	0.025641025641
Barack Obama	George Bush	1	0.05
Barack Obama	George Bush	10	0.025641025641
Barack Obama	George Bush	50	0.0526315789474
football	soccer	1	0.0
football	soccer	10	0.025641025641
football	soccer	50	0.0526315789474
aggies	longhorns	1	0.0
aggies	longhorns	10	0.025641025641
aggies	longhorns	50	0.111111111111
aggies	kevin sumlin	1	0.0344827586207
aggies	kevin sumlin	10	0.142857142857
aggies	kevin sumlin	50	0.25

Task 2.3: Explain Yourself!

Personally, I think Jaccard Similarity is a decent similarity function, particularly because it's fast and easy to implement. However, in the case of finding similarity between Twitter queries, Jaccard performs much better after a few slight tweaks. Rather than calculating Jaccard on the entire set of words returned from the query, I limited each set to the top 20 most frequent terms in the result. This is kind of a mashup between using TF scores and Jaccard. Since stop words are often the most frequent terms, but carry little meaning, I removed all stop words from the query results so that the top 20 term set would be much more meaningful. The stop words I removed were the list of words from Homework 1 with the addition of web/twitter-related terms such as http(s), and RT.

As N increases from 1 to 50, I think you get more accurate similarity scores because the list of top terms that best represent that query begin to become more distinct with increasing amounts of tweets. When N is 1, you'll often get a similarity score of 0.0 because, in most cases, the probability that the sets of terms from the two queries do not intersect at all. This is because Twitter tends to return a lot of irrelevant spam results when you query for a specific topic or keyword. One way to alleviate this phenomenon is to add `result_type="popular"` to your query string, however, this is still slightly susceptible to spam results.

With both vanilla Jaccard and my improved Jaccard similarity functions, it seems that query pairs that are more similar to one another based on current events will score higher than those that aren't as similar. This is likely because Twitter returns the most recent tweets in its search results. In many cases, a tweet's timestamp takes precedence over its relevance to the query. This is why the query pair <Microsoft CEO/Satya Nadella> is much more similar compared to <Microsoft CEO/Bill Gates>.