MIDS UC Berkeley - Machine Learning at Scale

DATSCIW261 ASSIGNMENT #4

James Gray (https://github.com/jamesgray007)
jamesgray@ischool.berkeley.edu
Time of Initial Submission: 08:21 PM US Central, Friday, June 10, 2016
Time of **Resubmission**:
W261-1, Spring 2016
Week 4 Homework

References for this Assignment

- Python Hosted Documentation (https://pythonhosted.org/mrjob/)

In [ ]:
```
#!pip install mrjob
```

HW4.0

What is MrJob? How is it different to Hadoop MapReduce?

MRJob (https://pythonhosted.org/mrjob/index.html) is a programming framework to execute Hadoop Streaming jobs using Python. It was developed by Yelp in 2010 to enable parallel processing of log files. One of the primary motivations of MRJob is enable easy programming and execution of jobs with multiple steps or iterations all from one program. This enables coding and execution of complex data processing pipelines from one encapsulated code module. MRJob can be executed locally (laptop), cluster and has easy integration with AWS Elastic MapReduce.

In contrast to Hadoop MapReduce, individual code modules (e.g., mappers, reducers) are distributed off to the cluster machines unlike the MRJob code which is a class that contains all of the mapper and reducer logic. It is more difficult to orchestrate pipelines with multiple iterations and jobs in pure Hadoop MapReduce.

_What are the mapper_init, mapper_final(), combiner_final(), reducer*final() methods? When are they called?*

All of the MRJob code is written in one class (a subclass of MRJob) and these methods provide additional control prior and post execution of the mapper and reduce code. Quite often you will want to run initialization tasks such as reading a file prior to executing the mapper and reduce code and this is the purpose of the mapper_init method. You may also

want to operations such as writing files and running clean-up routines after the mapper and reduce code has processed and this is the purpose of the mapper_final and reducer_final methods. These _init and _final methods give rich control to the programmer to run pre and post operations that would be difficult to achieve in pure Hadoop MapReduce.

HW4.1

What is serialization in the context of MrJob or Hadoop?

Serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment (Wikipedia (https://en.wikipedia.org/wiki/Serialization)). Deserialization is the reverse process of turning a byte stream into structured objects.

Serialization is used in two distinct areas of distributed data processing:

- Interprocess communication
- Persistent storage

When it used in these frameworks?

In the context of MRJob and Hadoop, serialization is used to turn structured objects into a byte stream and transmit these over the network to cluster machines. The binary format is much more efficient for storing and transmitting data than raw text data.

What is the default serialization mode for input and outputs for MrJob?

The default mode for MRJob input data is raw text data and the output format is JSON. MRJob uses JSON protocol for internal transmission between processes or steps. MRJob has default and other protocols that are available:

Defaults

- INPUT_PROTOCOL = mrjob.protocol.RawValueProtocol
- INTERNAL_PROTOCOL = mrjob.protocol.JSONProtocol
- OUTPUT_PROTOCOL = mrjob.protocol.JSONProtocol

Available

- RawProtocol / RawValueProtocol
- JSONProtocol / JSONValueProtocol
- PickleProtocol / PickleValueProtocol
- ReprProtocol / ReprValueProtocol

HW 4.2 - Microsoft Logfiles

Recall the Microsoft logfiles data from the async lecture. The logfiles are described are located at:

https://kdd.ics.uci.edu/databases/msweb/msweb.html
(https://kdd.ics.uci.edu/databases/msweb/msweb.html)
http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/
(http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/)

This dataset records which areas (Vroots) of www.microsoft.com each user visited in a one-week timeframe in Feburary 1998.

Here, you must preprocess the data on a single node (i.e., not on a cluster of nodes) **from the format:**

C,"10001",10001 #Visitor id 10001
V,1000,1 #Visit by Visitor 10001 to page id 1000
V,1001,1 #Visit by Visitor 10001 to page id 1001
V,1002,1 #Visit by Visitor 10001 to page id 1002
C,"10002",10002 #Visitor id 10001

to the format:

V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001

Write the python code to accomplish this.

HW4.2 - Log File Transformation

The code below processes each line of the raw logfile data and transforms the data into the defined format above. The structure of the file is consistent with the visitor ID first and then a record for each web page visited. This new format will be used by MRJob.

In [3]:
```
%%writefile process_logfile.py
#!/usr/bin/python
## logfile.py
## Author: James Gray
## Description: convert raw log file dataset into new defined format

# open CSV file
from csv import reader
with open('anonymous-msweb.data','rb') as f:
    data=f.readlines()

for i in reader(data):
    if i[0]=='C':
        visitor_id=i[1] #Store visitor id
        continue
    if i[0]=='V':
        print i[0]+','+i[1]+','+i[2]+',C,'+visitor_id #Append visitor_i
```

```
Overwriting process_logfile.py
```

In [4]:
```
# let's test the script file to ensure we properly converting the data
!chmod a+x process_logfile.py

!python process_logfile.py > ms_logs.txt
!cat ms_logs.txt | head -10
```

```
V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001
V,1001,1,C,10002
V,1003,1,C,10002
V,1001,1,C,10003
V,1003,1,C,10003
V,1004,1,C,10003
V,1005,1,C,10004
V,1006,1,C,10005
cat: write error: Broken pipe
```

HW 4.3 - Top Five Pages

Find the 5 most frequently visited pages using MrJob from the output of 4.2 (i.e., transfromed log file). This code is similar to the standard word count example but there is a 2nd reducer step that operates as a sort function using:

mapreduce.partition.keycomparator.options': '-k2,2nr -k1,1'

that will sort the reducer output by page views (2nd field) and then web page ID.

In [18]:
```
#Use this to make sure we reload the MrJob code when we make changes
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [5]:
```
%%writefile TopFivePages.py
from mrjob.job import MRJob
#from mrjob.step import MRJobStep
from mrjob.step import MRStep
import csv

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class MRGetTopFive(MRJob):

    def mapper_count_visits(self, _, line):
        record = csv_readline(line)
        if record[0] == 'V':
            yield record[1], 1

    def reducer_sum_visits(self, page_id, counts):
        yield page_id, sum(counts)

    def reducer_sort_visits(self, page_id, counts):
        yield page_id, sum(counts)

    def steps(self):
        JOBCONF_STEP2 = {
            'mapreduce.job.output.key.comparator.class': 'org.apache.ha
            'stream.num.map.output.key.field': 2,
            'stream.map.output.field.separator':',',
            'mapreduce.partition.keycomparator.options': '-k2,2nr -k1,1
            'mapreduce.job.reduces': '1',
        }
        return [
            self.mr(mapper=self.mapper_count_visits,   # STEP 1:  count
                    reducer=self.reducer_sum_visits),
            self.mr(jobconf=JOBCONF_STEP2,
                    reducer=self.reducer_sort_visits)  # STEP 2:  sort
        ]

if __name__ == '__main__':
    MRGetTopFive.run()
```

```
Writing TopFivePages.py
```

In [23]:
```
# set file priveleges
!chmod a+x TopFivePages.py
```

HW4.3 - Execute MRJob from Command Line

In [7]:
```
!python TopFivePages.py -r hadoop ms_logs.txt
```

```
No configs found; falling back on auto-configuration
Creating temp directory /tmp/TopFivePages.hadoop.20160610.224207.6
00433
Looking for hadoop binary in $PATH...
Found hadoop binary: /usr/bin/hadoop
Using Hadoop version 2.7.2
Copying local files to hdfs:///user/hadoop/tmp/mrjob/TopFivePages.
hadoop.20160610.224207.600433/files/...
Looking for Hadoop streaming jar in /home/hadoop/contrib...
Looking for Hadoop streaming jar in /usr/lib/hadoop-mapreduce...
Found Hadoop streaming jar: /usr/lib/hadoop-mapreduce/hadoop-strea
ming.jar
Running step 1 of 2...
  packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-2.7.2-amzn-
1.jar] /tmp/streamjob3958455242306075430.jar tmpDir=null
  Connecting to ResourceManager at ip-172-31-7-251.us-west-1.compu
te.internal/172.31.7.251:8032
  Connecting to ResourceManager at ip-172-31-7-251.us-west-1.compu
te.internal/172.31.7.251:8032
```

Top Five Output

```
Streaming final output from hdfs:///user/hadoop/tmp
"1008"  10836
"1034"  9383
"1004"  8463
"1018"  5330
"1017"  5108
```

HW4.3 - Execute MRJob from Driver

In [24]:
```
%reload_ext autoreload
%autoreload 2

from TopFivePages import MRGetTopFive
mr_job = MRGetTopFive(args=['ms_logs.txt'])

with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        print (mr_job.parse_output_line(line))
```

```
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
WARNING:mrjob.job:mr() is deprecated and will be removed in v0.6.
0. Use mrjob.step.MRStep directly instead.
```

HW 4.4 - Most Frequent Visitor

Find the most frequent visitor of each page using MrJob and the output of 4.2 (i.e., transformed log file). In this output please include the webpage URL, webpageID and Visitor ID.

The first task is to create the master list of web pages and URL from the raw web log data.

HW4.4 - Create master file of web page URLs from raw log files

In [27]:
```
%%writefile generate_url.py
#!/usr/bin/python
## generate_url.py
## Author: James Gray
## Description: create master list of URL's from raw weblog data

# open CSV file
from csv import reader
with open('anonymous-msweb.data','rb') as f:
    data=f.readlines()

for i in reader(data):
    if i[0]=='A':
        print i[1]+','+i[3]+','+i[4]
```

```
Overwriting generate_url.py
```

Verify Master list of web pages

In [29]:
```
!chmod a+x generate_url.py
!python generate_url.py > msft_urls.txt
!cat msft_urls.txt | head -15
```

```
1287,International AutoRoute,/autoroute
1288,library,/library
1289,Master Chef Product Information,/masterchef
1297,Central America,/centroam
1215,For Developers Only Info,/developer
1279,Multimedia Golf,/msgolf
1239,Microsoft Consulting,/msconsult
1282,home,/home
1251,Reference Support,/referencesupport
1121,Microsoft Magazine,/magazine
1083,MS Access Support,/msaccesssupport
1145,Visual Fox Pro Support,/vfoxprosupport
1276,Visual Test Support,/vtestsupport
1200,Benelux Region,/benelux
1259,controls,/controls
```

HW4.4 - MR Job Configuration

In [20]:
```
#Use this to make sure we reload the MrJob code when we make changes
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [25]:

```
%%writefile MostFrequentVisitor.py
#!/usr/bin/python
## mostfrequentvisitor.py
## Author: James Gray
## Description: calculate the most frequent visitor of each web page. F

# Example data format
# V, webpageID,count,C,visitorID
# V,1000,1,C,10001
# V,1001,1,C,10001
# V,1002,1,C,10001

import csv
from collections import Counter
from operator import itemgetter

from mrjob.job import MRJob
from mrjob.step import MRStep

def csv_readline(line):
    """return a list of strings for each row"""
    for row in csv.reader([line]):
        return row

class MRFrequentVisitor(MRJob):

    def mapper_page_views(self, line_no, line):
        """Extracts the page that was visited and the visitor id"""
        row = csv_readline(line)
        # emit KV pairs of webpageID, visitorID
        if row[0] == 'V':
            yield row[1],row[4]

    def reducer_load_urls(self):
        """Load file of page URLs into reducer memory"""
        with open('msft_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Populuate URLs into a dictionary
            self.url_dict[int(i[0])]=i[2]

    def reducer_sum_views_by_visitor(self, vroots, visitor):
        """Summarizes visitor counts for each page,
        yields one record per page with the visitor responsible for
        the most views on that page"""

        # use a Counter to store the number of page views by user
        visitors=Counter()
        for i in visitor:
            visitors[i]+=1 #Aggregate page views for all visitors
        output= max(visitors.iteritems(), key=itemgetter(1))[0] #Find v
        yield (str(vroots)),(output,visitors[output],self.url_dict[int(
```

```
    def steps(self):
        return [MRStep(mapper=self.mapper_page_views,
                       reducer_init=self.reducer_load_urls,
                       reducer=self.reducer_sum_views_by_visitor)]

if __name__ == '__main__':
    FreqVisitor.run()
```

```
Overwriting MostFrequentVisitor.py
```

HW4.4 - MRJob Driver

In [26]:
```
#HW 4.4 - Driver Function
from MostFrequentVisitor import MRFrequentVisitor
import csv

def run_4_4():
    mr_job = MRFrequentVisitor(args=['ms_logs.txt','--file','msft_urls.
    with mr_job.make_runner() as runner:
        runner.run()
        print "PAGE | VISITOR ID | # VISITS | WEB PAGE URL "
        print "----------------------------------------"
        for line in runner.stream_output():
            output=mr_job.parse_output_line(line)
            print str(output[0])+'  '+str(output[1][0])+'        '+str(

run_4_4()
```

```
======== RESULTS ==========
PAGE | VISITOR ID | # VISITS | PAGE URL
----------------------------------------
1000  36585        1          /regwiz
1001  23995        1          /support
1002  35235        1          /athome
1003  22469        1          /kb
1004  35540        1          /search
1005  10004        1          /norge
1006  27495        1          /misc
1007  19492        1          /ie_intl
1008  35236        1          /msdownload
1009  22504        1          /windows
1010  20915        1          /vbasic
1011  40152        1          /officedev
1012  37811        1          /outlookdev
1013  32727        1          /vbasicsupport
1014  20914        1          /officefreestuff
1015  16662        1          /msexcel
1016  25542        1          /excel
```

HW4.4 - Output

```
======== RESULTS ==========
PAGE | VISITOR ID | # VISITS | PAGE URL
----------------------------------------
1000  36585        1          /regwiz
1001  23995        1          /support
1002  35235        1          /athome
1003  22469        1          /kb
1004  35540        1          /search
1005  10004        1          /norge
1006  27495        1          /misc
1007  19492        1          /ie_intl
1008  35236        1          /msdownload
1009  22504        1          /windows
1010  20915        1          /vbasic
1011  40152        1          /officedev
1012  37811        1          /outlookdev
1013  32727        1          /vbasicsupport
1014  20914        1          /officefreestuff
1015  16662        1          /msexcel
1016  35542        1          /sql
```

HW 4.5 Clustering Tweet Dataset

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter These Twitter users use language in very different ways, and were classified by hand according criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifie postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismol police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, person promotion, etc...)

Check out the preprints of recent research, which spawned this dataset:

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342) http://arxiv.org/abs/1508.0184 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

topUsers_Apr-Jul_2014_1000-words.txt

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

Using this data, you will implement a 1000-dimensional K-means algorithm in MrJob on the use
their 1000-dimensional word stripes/vectors using several centroid initializations and values of K

Note that each "point" is a user as represented by 1000 words, and that word-frequency distribu
are generally heavy-tailed power-laws (often called Zipf distributions), and are very rare in the la
class of discrete, random distributions. For each user you will have to normalize by its "TOTAL"
column. Try several parameterizations and initializations:

(A) K=4 uniform random centroid-distributions over the 1000 words (generate 1000 random num
and normalize the vectors) (B) K=2 perturbation-centroids, randomly perturbed from the aggrega
(user-wide) distribution (C) K=4 perturbation-centroids, randomly perturbed from the aggregated
wide) distribution (D) K=4 "trained" centroids, determined by the sums across the classes. Use u
(row-normalized) class-level aggregates as 'trained' starting centroids (i.e., the training is already
for you!). Note that you do not have to compute the aggregated distribution or the class-aggrega
distributions, which are rows in the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt

Row 1: Words Row 2: Aggregated distribution across all classes Row 3-6 class-aggregated
distributions for clases 0-3 For (A), we select 4 users randomly from a uniform distribution [1,...,
For (B), (C), and (D) you will have to use data from the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt

This file contains 5 special word-frequency distributions:

(1) The 1000-user-wide aggregate, which you will perturb for initializations in parts (B) and (C), a

(2-5) The 4 class-level aggregates for each of the user-type classes (0/1/2/3)

In parts (B) and (C), you will have to perturb the 1000-user aggregate (after initially normalizing b
sum, which is also provided). So if in (B) you want to create 2 perturbations of the aggregate, sta
(1), normalize, and generate 1000 random numbers uniformly from the unit interval (0,1) twice (fo
centroids), using:

from numpy import random numbers = random.sample(1000)

Take these 1000 numbers and add them (component-wise) to the 1000-user aggregate, and the
renormalize to obtain one of your aggregate-perturbed initial centroids.

##

Generate random initial centroids around the global aggregate

Part (B) and (C) of this question

##
def startCentroidsBC(k): counter = 0 for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readlines(): if counter == 2:
data = re.split(",",line) globalAggregate = [float(data[i+3])/float(data[2]) for i in range(1000)] count

```
    ## perturb the global aggregate for the four initializations
    centroids = []
    for i in range(k):
        rndpoints = random.sample(1000)
        peturpoints = [rndpoints[n]/10+globalAggregate[n] for n in range
    000)]
        centroids.append(peturpoints)
        total = 0
        for j in range(len(centroids[i])):
            total += centroids[i][j]
        for j in range(len(centroids[i])):
            centroids[i][j] = centroids[i][j]/total
    return centroids
```

—— For experiments A, B, C and D and iterate until a threshold (try 0.001) is reached. After convergence, print out a summary of the classes present in each cluster. In particular, report the composition as measured by the total portion of each class type (0-3) contained in each cluster, discuss your findings and any differences in outcomes across parts A-D.

HW4.5 - Part A

HW4.5 - Part B

HW4.5 - Part C

HW4.5 - Part D

HW4.6 (OPTIONAL) Scaleable K-MEANS++

Over half a century old and showing no signs of aging, k-means remains one of the most popular data processing algorithms. As is well-known, a proper initialization of k-means is crucial for obtaining a good final solution. The recently proposed k-means++ initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the k-means++ is its inherent sequential nature,

which limits its applicability to massive data: one must make k passes over the data to find a good initial set of centers. The paper listed below shows how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing k-means that have mostly focused on the post-initialization phases of k-means. The proposed initialization algorithm k-means|| obtains a nearly optimal solution after a logarithmic number of passes; the paper also shows that in practice a constant number of passes suffices. Experimental evaluation on realworld large-scale data demonstrates that k-means|| outperforms k-means++ in both sequential and parallel settings.

Read the following paper entitled "Scaleable K-MEANS++" located at:

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf (http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf)

In MrJob, implement K-MEANS|| and compare with a random initializtion when used in conjunction with the kmeans algorithm as an initialization step for the 2D dataset generated using code in the following notebook:

https://www.dropbox.com/s/lbzwmyv0d8rocfq/MrJobKmeans.ipynb?dl=0 (https://www.dropbox.com/s/lbzwmyv0d8rocfq/MrJobKmeans.ipynb?dl=0)

Plot the initialation centroids and the centroid trajectory as the K-MEANS|| algorithms iterates. Repeat this for a random initalization (i.e., pick a training vector at random for each inital centroid) of the kmeans algorithm. Comment on the trajectories of both algorithms. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

HW 4.6.1

Apply your implementation of K-MEANS|| to the dataset in HW 4.5 and compare to the a random initalization (i.e., pick a training vector at random for each inital centroid)of the kmeans algorithm. Report on the number passes over the training data, and time required to run all clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

HW4.7 (OPTIONAL) Canopy Clustering

An alternative way to intialize the k-means algorithm is the canopy clustering. The canopy clustering algorithm is an unsupervised pre-clustering algorithm introduced by Andrew McCallum, Kamal Nigam and Lyle Ungar in 2000. It is often used as preprocessing step for the K-means algorithm or the Hierarchical clustering algorithm. It is intended to speed up clustering operations on large data sets, where using another algorithm directly may be impractical due to the size of the data set.

For more details on the Canopy Clustering algorithm see:

https://en.wikipedia.org/wiki/Canopy_clustering_algorithm (https://en.wikipedia.org/wiki/Canopy_clustering_algorithm)

Plot the initialation centroids and the centroid trajectory as the Canopy Clustering based K-MEANS algorithm iterates. Repeat this for a random initalization (i.e., pick a training vector at random for each inital centroid) of the kmeans algorithm. Comment on the trajectories of both algorithms. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

HW4.7.1

Apply your implementation Canopy Clustering based K-MEANS algorithm to the dataset in HW 4.5 and compare to the a random initalization (i.e., pick a training vector at random for each inital centroid)of the kmeans algorithm. Report on the number passes over the training data, and time required to run both clustering algorithms. Also report the rand index score for both algorithms and comment on your findings.

In [ ]: