HW 4.0

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

MRJob is a Python framework to make running complex Map Reduce tasks much simpler. It is capable of running sequences of MapReduce or even iterative MapReduce jobs. The really nice thing about MRJob is the almost pseudo-code like way of expressing how to execute and combine MapReduce jobs.

MRJob is not Hadoop but it can execute in a stand-alone mode to run your MapReduce jobs, useful for small scale testing. MRJob also can submit your job to Hadoop via the Streaming API, whether on a local or remote Hadoop cluster. In addition, MRJob has a very nice integration with Amazon AWS Elastic Map Reduce, allowing the researcher to focus on the MapReduce and analysis instead of the infrastructure on which to execute it.

What are the mapper_init, mapper_final(), combiner_final(), reducer_final() methods? When are they called?

MRJob defines a base class that you as the developer must override to use MRJob. The base class executes the mapper, reducer, and combiner functions when you override them in the class. The MRJob base class also provides intializer and finalizer methods for each of the mapper, combiner and reducer functions. These methods are `mapper_init()`, `combiner_init()`, `reducer_init()`, `mapper_final()`, `combiner_final()`, and `reducer_final()` respectively. The init() methods are called before the corresponding `mapper()`, `combiner()`, `reducer()` methods, allowing setup of data or other things before the method is called. The final() methods are called immediately after the `mapper()`, `reducer()` or `combiner()` methods.

HW 4.1

What is serialization in the context of MrJob or Hadoop?

Serialization is the process of converting a machine representation of an object to a format used for storage or transmission. In the context of Hadoop Streaming all input and output is treated as a character stream with keys and values separated by tabs (or another specified delimiter). In the case of MRJob, serialization consists of three types: raw, json, or pickle. Raw is text streams, json is json formatted text streams, and pickle is the Python binary serialization method.

When it used in these frameworks?

MRJob uses serialization for input and output as well as internal transmission of objects. Each place serialization is used can be defined by the type of protocol.

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

The default serialization mode for MRJob inputs is `RAWValueProtocol` which reads lines of text with no key - it's just a stream of text. The default output protocol is `JSONprotocol` which outputs JSON formatted strings separated by a tab character.

HW 4.2:

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)](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/)](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
```

V Note: #denotes comments 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.

In [ ]:
```
#make input file
import csv

with open('anonymous-msweb.data', 'r') as inputFile, open('msweblo
g.csv', 'wb') as outputFile: #get appropriate files
    writer = csv.writer(outputFile) #use csv.writer for writiing ou
tput
    lines = inputFile.readlines() #get input files
    currID = None #stores current ID
    for line in lines:
        line = line.split(',') #split on comma delimiter
        if line[0] == 'C': #if the line starts with C, that means w
e're dealing with a new customer ID
            currID = int(line[2]) #Set the customer ID
        elif line[0] == 'V': #otherwise if V, that's the visiting b
ehavior (we ignore all other lines)
            newLine = ['V']
            newLine.extend((line[1], '1', 'C', currID)) #construct
a new line
            writer.writerow(newLine) #write output
```

HW 4.3:

Find the 5 most frequently visited pages using MrJob from the output of 4.2 (i.e., transfromed log file).

In [ ]:
```
%%writefile mostvisitedpage.py
from mrjob.job import MRJob
from mrjob.step import MRStep
import heapq

class MRMostVisitedPage(MRJob):
    def mapper_get_visits(self, _, record):
        self.increment_counter('Execution Counts', 'mapper calls', 
1)
        # yield each visit in the line
        tokens = record.split(',')
        if tokens[0] == 'V':
            yield (tokens[1], 1)

    def combiner_count_visits(self, page, counts):
        self.increment_counter('Execution Counts', 'combiner call
s', 1)
        # sum the page visits we've seen so far
        yield (page, sum(counts))

    def reducer_count_visits(self, page, counts):
        self.increment_counter('Execution Counts', 'reducer_count c
alls', 1)
        # send all (num_occurrences, word) pairs to the same reduce
r.
        # num_occurrences is so we can easily use Python's max() fu
nction. yield None, (sum(counts), page)
        # discard the key; it is just None
        yield None, (sum(counts), page)

    def reducer_find_top5_visits(self, _, page_count_pairs):
        self.increment_counter('Execution Counts', 'reducer_find_ma
x calls', 1)
        # each item of page_count_pairs is (count, page),
        # so yielding one results in key=counts, value=page yield m
ax(page_count_pairs)
        return heapq.nlargest(5, page_count_pairs)


    def steps(self): return [
            MRStep(mapper=self.mapper_get_visits,
                   combiner=self.combiner_count_visits,
                   reducer=self.reducer_count_visits),
            MRStep(reducer=self.reducer_find_top5_visits)
        ]

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

In [ ]:
```
!python mostvisitedpage.py anonymous-msweb-transformed.data
```

NB: Copied from working version on local machine. Emphasis added

no configs found; falling back on auto-configuration no configs found; falling back on auto-configuration creating tmp directory /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 0-mapper*part-00000 Counters from step 1: Execution Counts: combiner calls: 285 mapper calls: 98955 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 0-mapper-sorted sort /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 0-mapper_part-00000 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 0-reducer*part-00000 Counters from step 1: Execution Counts: combiner calls: 285 mapper calls: 98955 reducer_count calls: 285 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 1-mapper*part-00000 Counters from step 2: (no counters found) writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 1-mapper-sorted sort /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 1-mapper_part-00000 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 1-reducer*part-00000 Counters from step 2: Execution Counts: reducer_find_max calls: 1 Moving /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 1-reducer*part-00000 -> /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.738794/ 00000 Streaming final output from /var/folders/z_/rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.73879

10836 "1008" 9383 "1034" 8463 "1004" 5330 "1018" 5108 "1017"

removing tmp directory /var/folders/z_/rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostvisitedpage.rcordell.20160208.015734.73879

HW 4.4:

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

In [ ]:

```
%%writefile mostfreqvisitors.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRMostFrequentVisitors(MRJob):
    def configure_options(self):
        super(MRMostFrequentVisitors, self).configure_options()
        self.SORT_VALUES = True

    # generate a dictionary of pages and URLs for them
    def mapper_get_visits_init(self):
        # create a dictionary to use for the page URLs and ids
        self.pages = {}

    # generate keys of page,customer,url and values of 1
    def mapper_get_visits(self, _, record):
        self.increment_counter('Execution Counts', 'mapper calls', 
1)
        tokens = record.split(',')

        # the page definitions come first in the file so create a d
ictionary from them.
        if tokens[0] == 'A':
            self.pages[tokens[1]] = tokens[4].strip('"')

        # emit a key = (page_id, client_id, url) and value = 1
        elif tokens[0] == 'V':
            yield ((tokens[1], tokens[4], self.pages[tokens[1]]), 
1)
        else:
            pass

    # combine page visits by key where the key is page,customer
    def combiner_count_visits(self, key, counts):
        self.increment_counter('Execution Counts', 'combiner count 
visits', 1)
        # sum the keys we've seen so far.
        # the key is (page_id, cust_id, page_url) so we're counting 
page views by client
        yield (key, sum(counts))

    # set up instance variables to use to calculate the max visits 
to a page by a single customer
    def reducer_count_visits_init(self):
        self.current_page = None
        self.max_count = 0

    # count the visits per page per customer and also compute the m
ax visits per page by a single customer
    def reducer_count_visits(self, key, counts):
        self.increment_counter('Execution Counts', 'reducer_count v
isits', 1)
        # make sure we have sums of all keys
        s = sum(counts)
```

```
        if self.current_page == key[0]:
            if self.max_count < s:
                self.max_count = s
        else:
            if self.current_page:
                p = self.current_page
                t = self.max_count
                yield((self.current_page,'*',key[2]), t)

            self.current_page = key[0]
            self.max_count = s

        yield (key, s)

    # set up a variable to contain the current page max count value
    def reducer_find_max_visits_init(self):
        self.page_max = 0

    # yield the max visits to a page and the customers that made th
em
    def reducer_find_max_visits(self, key, counts):
        self.increment_counter('Execution Counts', 'reducer_find_ma
x visits', 1)

        # if this is the key with the max visits for the page then
stash it
        if key[1] == '*':
            self.page_max = sum(counts)
        else:
            # otherwise sum the counts and store a local copy becau
se it exhausts the generator
            p = sum(counts)
            # if this count is the same as the max visits for the p
age, yield it
            if p == self.page_max:
                yield key, p


    def steps(self): return [
            MRStep(mapper_init=self.mapper_get_visits_init,
                    mapper=self.mapper_get_visits,
                   combiner=self.combiner_count_visits,
                   reducer_init=self.reducer_count_visits_init,
                   reducer=self.reducer_count_visits),
            MRStep(reducer_init=self.reducer_find_max_visits_init,
                    reducer=self.reducer_find_max_visits)
        ]

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

In [ ]:
```
!python mostfreqvisitors.py anonymous-msweb-transformed.data > max_
page_visits_customer.output
```

NB from working notebook on local machine

no configs found; falling back on auto-configuration no configs found; falling back on auto-configuration ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner' creating tmp directory /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 0-mapper*part-00000 Counters from step 1: Execution Counts: combiner count visits: 98654 mapper calls: 98955 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 0-mapper-sorted sort /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 0-mapper_part-00000 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 0-reducer*part-00000 Counters from step 1: Execution Counts: combiner count visits: 98654 mapper calls: 98955 reducer_count visits: 98654 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 1-mapper*part-00000 Counters from step 2: (no counters found) writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 1-mapper-sorted sort /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 1-mapper_part-00000 writing to /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 1-reducer*part-00000 Counters from step 2: Execution Counts: reducer_find_max visits: 98938 Moving /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/s 1-reducer*part-00000 -> /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/o 00000 Streaming final output from /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269/o removing tmp directory /var/folders/z/*rfp5q2cd6db13d19v6yw0n8w0000gn/T/mostfreqvisitors.rcordell.20160208.053219.800269

In [ ]:
```
!cat max_page_visits_customer.output | head -100
```

["1000", "10001", "/regwiz"] 1 ["1000", "10010", "/regwiz"] 1 ["1000", "10039", "/regwiz"] 1 ["1000", "10073", "/regwiz"] 1 ["1000", "10087", "/regwiz"] 1 ["1000", "10101", "/regwiz"] 1 ["1000", "10132", "/regwiz"] 1 ["1000", "10141", "/regwiz"] 1 ["1000", "10154", "/regwiz"] 1 ["1000", "10162", "/regwiz"] 1 ["1000", "10166", "/regwiz"] 1 ["1000", "10201", "/regwiz"] 1 ["1000", "10218", "/regwiz"] 1 ["1000", "10220", "/regwiz"] 1 ["1000", "10324", "/regwiz"] 1 ["1000", "10348", "/regwiz"] 1 ["1000", "10376", "/regwiz"] 1 ["1000", "10384", "/regwiz"] 1 ["1000", "10409", "/regwiz"] 1 ["1000", "10429", "/regwiz"] 1 ["1000", "10454", "/regwiz"] 1 ["1000", "10457", "/regwiz"] 1 ["1000", "10471", "/regwiz"] 1 ["1000", "10497", "/regwiz"] 1 ["1000", "10511",

"/regwiz"] 1 ["1000", "10520", "/regwiz"] 1 ["1000", "10541", "/regwiz"] 1 ["1000", "10564", "/regwiz"] 1 ["1000", "10599", "/regwiz"] 1 ["1000", "10752", "/regwiz"] 1 ["1000", "10756", "/regwiz"] 1 ["1000", "10861", "/regwiz"] 1 ["1000", "10935", "/regwiz"] 1 ["1000", "10943", "/regwiz"] 1 ["1000", "10969", "/regwiz"] 1 ["1000", "11027", "/regwiz"] 1 ["1000", "11050", "/regwiz"] 1 ["1000", "11410", "/regwiz"] 1 ["1000", "11429", "/regwiz"] 1 ["1000", "11440", "/regwiz"] 1 ["1000", "11490", "/regwiz"] 1 ["1000", "11501", "/regwiz"] 1 ["1000", "11528", "/regwiz"] 1 ["1000", "11539", "/regwiz"] 1 ["1000", "11544", "/regwiz"] 1 ["1000", "11685", "/regwiz"] 1 ["1000", "11695", "/regwiz"] 1 ["1000", "11723", "/regwiz"] 1 ["1000", "11766", "/regwiz"] 1 ["1000", "11774", "/regwiz"] 1 ["1000", "11779", "/regwiz"] 1 ["1000", "11898", "/regwiz"] 1 ["1000", "11964", "/regwiz"] 1 ["1000", "12017", "/regwiz"] 1 ["1000", "12020", "/regwiz"] 1 ["1000", "12035", "/regwiz"] 1 ["1000", "12086", "/regwiz"] 1 ["1000", "12123", "/regwiz"] 1 ["1000", "12143", "/regwiz"] 1 ["1000", "12155", "/regwiz"] 1 ["1000", "12201", "/regwiz"] 1 ["1000", "12220", "/regwiz"] 1 ["1000", "12228", "/regwiz"] 1 ["1000", "12262", "/regwiz"] 1 ["1000", "12273", "/regwiz"] 1 ["1000", "12306", "/regwiz"] 1 ["1000", "12315", "/regwiz"] 1 ["1000", "12324", "/regwiz"] 1 ["1000", "12337", "/regwiz"] 1 ["1000", "12343", "/regwiz"] 1 ["1000", "12400", "/regwiz"] 1 ["1000", "12415", "/regwiz"] 1 ["1000", "12484", "/regwiz"] 1 ["1000", "12485", "/regwiz"] 1 ["1000", "12537", "/regwiz"] 1 ["1000", "12571", "/regwiz"] 1 ["1000", "12583", "/regwiz"] 1 ["1000", "12674", "/regwiz"] 1 ["1000", "12700", "/regwiz"] 1 ["1000", "12740", "/regwiz"] 1 ["1000", "12815", "/regwiz"] 1 ["1000", "12853", "/regwiz"] 1 ["1000", "12893", "/regwiz"] 1 ["1000", "12897", "/regwiz"] 1 ["1000", "12930", "/regwiz"] 1 ["1000", "12944", "/regwiz"] 1 ["1000", "12970", "/regwiz"] 1 ["1000", "12982", "/regwiz"] 1 ["1000", "13015", "/regwiz"] 1 ["1000", "13049", "/regwiz"] 1 ["1000", "13079", "/regwiz"] 1 ["1000", "13080", "/regwiz"] 1 ["1000", "13085", "/regwiz"] 1 ["1000", "13128", "/regwiz"] 1 ["1000", "13176", "/regwiz"] 1 ["1000", "13197", "/regwiz"] 1 ["1000", "13223", "/regwiz"] 1 ["1000", "13248", "/regwiz"] 1 ["1000", "13275", "/regwiz"] 1 ["1000", "13294", "/regwiz"] 1 cat: stdout: Broken pipe

In [ ]:
```
!cat max_page_visits_customer.output | tail -100
```

["1295", "38244", "/train_cert"] 1 ["1295", "38296", "/train_cert"] 1 ["1295", "38313", "/train_cert"] 1 ["1295", "38454", "/train_cert"] 1 ["1295", "38571", "/train_cert"] 1 ["1295", "38573", "/train_cert"] 1 ["1295", "38661", "/train_cert"] 1 ["1295", "38678", "/train_cert"] 1 ["1295", "38755", "/train_cert"] 1 ["1295", "38831", "/train_cert"] 1 ["1295", "38869", "/train_cert"] 1 ["1295", "38953", "/train_cert"] 1 ["1295", "38981", "/train_cert"] 1 ["1295", "38998", "/train_cert"] 1 ["1295", "39024", "/train_cert"] 1 ["1295", "39033", "/train_cert"] 1 ["1295", "39058", "/train_cert"] 1 ["1295", "39066", "/train_cert"] 1 ["1295", "39094", "/train_cert"] 1 ["1295", "39105", "/train_cert"] 1 ["1295", "39112", "/train_cert"] 1 ["1295", "39131", "/train_cert"] 1 ["1295", "39194", "/train_cert"] 1 ["1295", "39221", "/train_cert"] 1 ["1295", "39284", "/train_cert"] 1 ["1295", "39293", "/train_cert"] 1 ["1295", "39493", "/train_cert"] 1 ["1295", "39505", "/train_cert"] 1 ["1295", "39550", "/train_cert"] 1 ["1295", "39604", "/train_cert"] 1 ["1295", "39617", "/train_cert"] 1 ["1295", "39627", "/train_cert"] 1 ["1295", "39645", "/train_cert"] 1 ["1295", "39719", "/train_cert"] 1 ["1295", "39730", "/train_cert"] 1 ["1295", "39760", "/train_cert"] 1 ["1295", "39863", "/train_cert"] 1 ["1295", "39899", "/train_cert"] 1 ["1295", "39900", "/train_cert"] 1 ["1295", "39902", "/train_cert"] 1 ["1295", "39948", "/train_cert"] 1 ["1295", "39977", "/train_cert"] 1 ["1295", "40025", "/train_cert"] 1 ["1295", "40046", "/train_cert"] 1 ["1295", "40207", "/train_cert"] 1 ["1295", "40233", "/train_cert"] 1 ["1295", "40274", "/train_cert"] 1 ["1295", "40310", "/train_cert"] 1 ["1295", "40390", "/train_cert"] 1 ["1295", "40419", "/train_cert"] 1 ["1295", "40482", "/train_cert"] 1 ["1295", "40597", "/train_cert"] 1 ["1295", "40616", "/train_cert"] 1 ["1295", "40679", "/train_cert"] 1 ["1295", "40758", "/train_cert"] 1 ["1295", "40787", "/train_cert"] 1 ["1295", "40827", "/train_cert"] 1 ["1295", "40923", "/train_cert"] 1 ["1295", "40930", "/train_cert"] 1 ["1295", "40942", "/train_cert"] 1 ["1295", "40946", "/train_cert"] 1 ["1295", "40965", "/train_cert"] 1 ["1295", "41068", "/train_cert"] 1 ["1295", "41075",

"/train_cert"] 1 ["1295", "41093", "/train_cert"] 1 ["1295", "41100", "/train_cert"] 1 ["1295", "41117", "/train_cert"] 1 ["1295", "41175", "/train_cert"] 1 ["1295", "41183", "/train_cert"] 1 ["1295", "41207", "/train_cert"] 1 ["1295", "41255", "/train_cert"] 1 ["1295", "41269", "/train_cert"] 1 ["1295", "41273", "/train_cert"] 1 ["1295", "41367", "/train_cert"] 1 ["1295", "41429", "/train_cert"] 1 ["1295", "41580", "/train_cert"] 1 ["1295", "41594", "/train_cert"] 1 ["1295", "41598", "/train_cert"] 1 ["1295", "41692", "/train_cert"] 1 ["1295", "41715", "/train_cert"] 1 ["1295", "41730", "/train_cert"] 1 ["1295", "41748", "/train_cert"] 1 ["1295", "41843", "/train_cert"] 1 ["1295", "41953", "/train_cert"] 1 ["1295", "42065", "/train_cert"] 1 ["1295", "42146", "/train_cert"] 1 ["1295", "42161", "/train_cert"] 1 ["1295", "42198", "/train_cert"] 1 ["1295", "42234", "/train_cert"] 1 ["1295", "42241", "/train_cert"] 1 ["1295", "42262", "/train_cert"] 1 ["1295", "42313", "/train_cert"] 1 ["1295", "42353", "/train_cert"] 1 ["1295", "42385", "/train_cert"] 1 ["1295", "42497", "/train_cert"] 1 ["1295", "42516", "/train_cert"] 1 ["1295", "42568", "/train_cert"] 1 ["1295", "42576", "/train_cert"] 1 ["1295", "42600", "/train_cert"] 1 ["1295", "42616", "/train_cert"] 1

We found that no user visited a webpage more than once. That is each user visited each vroot exactly one time, meaning all visitors to a vroot are tied

Question 4.5

Implement a 1000-dimensional K-means algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using several centroid initializations and values of K:

- *(A) K=4 uniform random centroid-distributions over the 1000 words*
- *(B) K=2 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution*
- *(C) K=4 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution*
- *(D) K=4 "trained" centroids, determined by the sums across the classes.*

Report the composition as measured by the total portion of each class type (0-3) contained in each cluster, and discuss your findings and any differences in outcomes across parts A-D.

Solution

To accomplish this task in an extensible way, we wrote our `kmeans` architecture in a way that compartmentalizes different portions of the algorithm. This allows us to change properties of the `kmeans` iteration, such as initialization type, without substantially changing the code.

Driver:

Below is the driver, `kmeans.py` that is used to initialize the MRJob. It takes two arguments - initialization type and number of clusters. The initialization type can be one of the following three options:

- *uniform* - this is a random initialization using points selected from our data set
- *perturbation* - this uses the mean value of the different fields to perturb the centroids
- *trained* - this uses the class labels as groups by which centroids are constructed using averaging

The driver import five other sub-MRJob files to accomplish the different portions of the procedure, as we'll see below. The driver communicates with the other components in two main ways- pass-through arguments and flat files. The pass-through arguments are used to initialize the different portions of the algorithm, as well as specifying lookup files when necessary. Flat files are used to communicate updated cluster coordinates, and cluster labels for each customer.

Finally, the progress and results of the algorithm are presented to the terminal in readable format as the algorithm runs.

In [ ]:

```
%%writefile kmeans.py
from __future__ import division

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

from init_centroids import initializeCentroids
from assign_clusters import assignClusters
from update_centroids import updateCentroids
from get_error import getError
from diagnostics import diagnostics

import cPickle as pickle
from collections import defaultdict
import sys

# Storage files
trackerFile = './.tracker'
centroidFile = './.clusters'
scoreFile = './.scores'
dataFile = 'topUsers_Apr-Jul_2014_1000-words.txt'


def getName(obj, namespace):
        return [name for name in namespace if namespace[name] is ob
j]


def extractValues(job, runner):
        output = defaultdict(int)
        for line in runner.stream_output():
                key, value = job.parse_output_line(line)
                output[key] = value

        return output


def dumpToFile(variable, filename):
        with open(filename, 'w') as f:
                pickle.dump(variable, f)


def dumpToTracker(variable, filename):
        with open(filename, 'a') as f:
                f.write('dumping...' + '\n')
                f.write(str(variable) + '\n')
                f.write('dump complete.' + '\n')


def runJob(method, args, dFile=centroidFile):
        job = method(args=args)

        methodName = getName(method, globals())[0]
        print '\n\t' + 'Running ' + methodName + '...'
```

```
        with job.make_runner() as runner:

                # Surpress console
                runner.run()

                result = extractValues(job, runner)
                dumpToTracker(result, trackerFile)

                print '\t' + 'Complete: ' + methodName

                if dFile:
                        dumpToFile(result, dFile)

                else:
                        return result


if __name__ == '__main__':

        args = sys.argv[1:]

        # Clear files
        open(trackerFile, 'w').close()
        open(centroidFile, 'w').close()
        open(scoreFile, 'w').close()

        # Step 1: Create initial clusters
        init = '--' + str(args[0])
        numClusters = '--k=' + str(args[1])

        # runJob(initializeCentroids, args=[dataFile, '--perturbati
on', '--k=4'])
        runJob(initializeCentroids, args=[dataFile, init, numCluste
rs])

        # Loop initializations
        priorError = 1
        threshold = 0.0001
        maxIter = 10

        # Loop
        for i in range(maxIter):

                print '\n' + 'Iteration ' + str(i) + '.'

                # Score based on clusters
                centroidArg = '--centroids='+centroidFile
                runJob(assignClusters, args=[dataFile, centroidAr
g], dFile=scoreFile)

                # Update clusters
                scoreArg = '--scores='+scoreFile
                runJob(updateCentroids, args=[dataFile, centroidAr
```

```
g, scoreArg])

                # Get error
                error = runJob(getError, args=[dataFile, centroidAr
g, scoreArg], dFile=None)
                currentError = error.values()[0]

                # Check threshold
                if priorError - currentError <= threshold:

                        # Get diagnostics
                        diag = runJob(diagnostics, args=[dataFile,
scoreArg], dFile=None)
                        print '\n' + 'Purity characteristics: ' +
'\n'

                        # Get cluster info
                        for cluster, clusterDiag in diag.iteritems
():
                                print '\t' + 'Cluster ' + str(clust
er) + ':'

                                for label, portion in clusterDiag.i
teritems():
                                        print '\t\t' + 'Label: ' +
str(label) + ', ' + 'Portion: ' + str(portion)

                                print ''

                        print '\n' + 'RMSE: ' + str(currentError) +
'\n'
                        break

                else:
                        priorError = currentError
```

Initializing the Centroids

The centroids initialization is handled by `init_centroids.py`. This is the MRJob portion that handles different initialization arguments through an `add_passthrough_option` which takes input from the driver. To handle the different initializations, there are six separate map and reduce tasks to handle the three cases.

Additionally, there is basic error-handling in case the appropriate options are not provided. Ultimately results of this job are yielded to a `extractValue()` parser in the driver to appropriately collate and pickle the results. The pickling allows one to systematically transfer Python data objects between the MRJob's in an asynchronous fashion.

In [ ]:

```
%%writefile init_centroids.py
from __future__ import division

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

import numpy as np
from collections import defaultdict
import string
import random
import cPickle as pickle

class initializeCentroids(MRJob):

        def __init__(self, args):
                MRJob.__init__(self, args)

        def configure_options(self):
                super(initializeCentroids, self).configure_options
()

                self.add_passthrough_option(
                        '--k', type='int', default=4, help='k: numb
er of clusters')

                self.add_passthrough_option(
                        '--uniform', action='store_true', default=F
alse, help='uniform: even cluster initialization')

                self.add_passthrough_option(
                        '--perturbation', action='store_true', defa
ult=False, help='perturbation: randomized cluster initialization')

                self.add_passthrough_option(
                        '--trained', action='store_true', default=F
alse, help='trained: class-based cluster initialization')


        def uniform_init_centroids_map(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                total = line[2]
                body = [x / total for x in line[3:]]

                k = self.options.k
                assignment = np.random.randint(k)

                yield assignment, body

        def uniform_get_centroids(self, assignment, arrays):

                # Parse iteratble
                arrays = list(arrays)
```

```
                # Get random element
                random.shuffle(arrays)
                newCluster = arrays[0]

                # Cluster labels
                s = string.ascii_uppercase

                yield s[assignment], newCluster



        def perturbation_init_centroids_map(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                total = line[2]
                body = [x / total for x in line[3:]]

                yield None, body

        def perturbation_get_centroids(self, _, totals):

                # Find the mean
                k = self.options.k
                s = string.ascii_uppercase
                clusterCenter = [np.mean(x) for x in zip(*totals)]

                # Emit
                for i in range(k):
                        cluster = [x + np.random.sample() for x in
clusterCenter]
                        yield s[i], cluster



        def trained_init_centroids_map(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                total = line[2]
                label = line[1]
                body = [x / total for x in line[3:]]

                yield label, body

        def trained_get_centroids(self, label, arrays):

                # Compute new cluster
                arrays = list(arrays)
                newCluster = [np.mean(x) for x in zip(*arrays)]

                # Cluster labels
                s = string.ascii_uppercase
```

```
                yield s[label], newCluster



        def steps(self):

                if self.options.uniform:
                        return [MRStep(mapper=self.uniform_init_cen
troids_map,
                                                                reducer=sel
f.uniform_get_centroids)]

                elif self.options.perturbation:
                        return [MRStep(mapper=self.perturbation_ini
t_centroids_map,
                                                                reducer=sel
f.perturbation_get_centroids)]

                elif self.options.trained:
                        return [MRStep(mapper=self.trained_init_cen
troids_map,
                                                                reducer=sel
f.trained_get_centroids)]

                else:

                        # No initialization
                        raise ValueError('ERROR: Please enter initi
alization type. See help for more details.')


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

Assigning Cluster ID's

Here, we describe the MRJob used to assign cluster ID's to the various points in our training data. Here, the pickled cluster characteristics dumped to `.clusters` is read in the reducer. The distance to each cluster is computed, and the `argmin` is returned for each customer ID.

The results of the assignment are pickled by the driver in `.scores`, which are used in the following MRJob segment used to update the centroid coordinates.

In [ ]:

```
%%writefile assign_clusters.py
from __future__ import division

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

import cPickle as pickle
import numpy as np

from collections import defaultdict

class assignClusters(MRJob):

        def __init__(self, args):
                MRJob.__init__(self, args)

        def configure_options(self):
                super(assignClusters, self).configure_options()
                self.add_file_option('--centroids',
                        help='pointer to centroids file. See main r
unner for details.')

        def mapper_diff_comp(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                custID = line[0]
                total = line[2]
                body = [x / total for x in line[3:]]

                # Read clusters
                with open(self.options.centroids, 'r') as f:
                        clusters = pickle.load(f)

                # Compute distances
                for clusterID, cluster in clusters.iteritems():

                        cluster = np.array(cluster)
                        body = np.array(body)
                        dist = np.linalg.norm(body-cluster)

                        yield custID, [clusterID, dist]


        def reducer_find_min_cluster(self, custID, distArray):

                # Find closest cluster
                distArray = np.array(list(distArray))
                clusterIndex = np.argmin(distArray[:, 1])
                clusterID = distArray[clusterIndex, 0]

                yield custID, clusterID

        def steps(self):
```

```
                return [MRStep(mapper=self.mapper_diff_comp,
                                                reducer=self.reduce
r_find_min_cluster)]

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

Update Centroids

Once the scores have been written, the cluster centers must be updated given the new class labels. Occasionally, some clusters may be eliminated or be reduced to singletons depending on their initialization. The centroids handling is flexible enough to allow this case when scoring and updating cluster centers.

The scores are read in the mapper, where the cluster ID is extracted and attached to each customer. Then, the reducer reads the centroid coordinates and computes a mean over the dimensions of the matching customer cluster labels. The shuffling between the mapper and reducer step guarantees that each cluster label is grouped with the respective arrays necessary for updating the centroids.

The update criteria is set to `i=10` absolute iterations, or a minimum `RMSE` change of `0.0001` between iterations.

In [ ]:

```
%%writefile update_centroids.py
from __future__ import division

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

import cPickle as pickle
import numpy as np
from collections import defaultdict

class updateCentroids(MRJob):

        def __init__(self, args):
                MRJob.__init__(self, args)

        def configure_options(self):
                super(updateCentroids, self).configure_options()

                self.add_file_option('--scores',
                        help='pointer to scores file. See main runn
er for details.')

                self.add_file_option('--centroids',
                        help='pointer to centroids file. See main r
unner for details.')

        def mapper_telegraph_id(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                custID = line[0]
                total = line[2]
                body = [x / total for x in line[3:]]

                # Load scores
                with open(self.options.scores, 'r') as f:
                        scores = pickle.load(f)

                # Get individual
                label = scores[custID]
                yield label, body

        def reducer_emit_clusters(self, label, arrays):

                # Compute new cluster
                arrays = list(arrays)
                newCluster = [np.mean(x) for x in zip(*arrays)]

                yield label, newCluster


        def steps(self):
                return [MRStep(mapper=self.mapper_telegraph_id,
                                                reducer=self.reduce
```

```
r_emit_clusters)]

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

Getting Error

After the centroids have been updated, we compute the `RMSE` for the new cluster centers. This is job is important as it allows the driver to terminate the algorithm if the `RMSE` fails to decrease by a minimum threshold after each iteration.

Anecdotally, when running this K-Means implementation over the different option, the `RMSE` threshold will be a much stronger deciding factor as to the runtime of the algorithm as compared with the capping the absolute iteration count. This fits with the intuition one expects when updating centroid centers.

In [ ]:

```
%%writefile get_error.py
from __future__ import division

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

import cPickle as pickle
import numpy as np

from collections import defaultdict

class getError(MRJob):

        def __init__(self, args):
                MRJob.__init__(self, args)

        def configure_options(self):
                super(getError, self).configure_options()

                self.add_file_option('--centroids',
                        help='pointer to centroids file. See main r
unner for details.')

                self.add_file_option('--scores',
                        help='pointer to scores file. See main runn
er for details.')

        def mapper_telegraph_id(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                custID = line[0]
                total = line[2]
                body = [x / total for x in line[3:]]

                # Load scores
                with open(self.options.scores, 'r') as f:
                        scores = pickle.load(f)

                # Get individual
                label = scores[custID]
                yield label, body

        def reducer_compute_dist(self, label, vectors):
                vectors = np.array(list(vectors))

                # Load centroids
                with open(self.options.centroids, 'r') as f:
                        centroids = pickle.load(f)

                cluster = centroids[label]
                dist = sum([np.linalg.norm(body-cluster) for body i
n vectors])
```

```
                yield None, dist

        def reducer_rmse(self, _, dists):

                # Load scores
                with open(self.options.scores, 'r') as f:
                        scores = pickle.load(f)

                N = len(scores)

                RMSE = np.sqrt(sum(dists) / N)

                yield None, RMSE

        def steps(self):
                return [MRStep(mapper=self.mapper_telegraph_id,
                                                reducer=self.reduce
r_compute_dist),

                                MRStep(reducer=self.reducer_rmse)]


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

Diagnostics- Purity Values

After the iterations have been terminated, a final MRJob is called to compute the purity of the final round of clusters. This loops through the initial training data as well as latest customer scores to determine the proprtion of each customer label within each cluster.

To prevent confusion, customer labels are assigned a number (0-3) and cluster labels are assigned an uppercase ASCII character (A,B,C...).

In [ ]:

```
%%writefile diagnostics.py
from __future__ import division

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

import cPickle as pickle
import numpy as np

from collections import defaultdict

class diagnostics(MRJob):

        def __init__(self, args):
                MRJob.__init__(self, args)

        def configure_options(self):
                super(diagnostics, self).configure_options()

                self.add_file_option('--scores',
                        help='pointer to scores file. See main runn
er for details.')

        def mapper_load_scores(self, _, line):

                # Parse data
                line = [int(x) for x in line.split(',')]
                custID, label = line[:2]

                # Load scores
                with open(self.options.scores, 'r') as f:
                        scores = pickle.load(f)

                # Get individual
                clusterID = scores[custID]
                yield clusterID, label

        def reducer_compute_diagnostics(self, clusterID, labels):

                # Get counts
                labels = list(labels)
                uniqueLabels = set(labels)
                labelTally = defaultdict(int)

                # Assign
                for label in uniqueLabels:
                        labelTally[label] = labels.count(label)

                total = sum(labelTally.values())

                # Get portions
                for key, value in labelTally.iteritems():
                        labelTally[key] = value / total
```

```
                # Emit
                yield clusterID, labelTally


        def steps(self):
                return [MRStep(mapper=self.mapper_load_scores,
                                                reducer=self.reduce
r_compute_diagnostics)]


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

Part (A)

Below we implement the solution for part (A). Here, we use a uniform initialization with 4 clusters. Notice the progress of the algorithm and final results are printed concisely as the algorithm progresses.

In [ ]:
```
!python kmeans.py 'uniform' 4
```

Running initializeCentroids... No handlers could be found for logger "mrjob.runner" Complete: initializeCentroids

Iteration 0.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 1.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 2.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 3.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError

Running diagnostics...
Complete: diagnostics
```

Purity characteristics:

```
Cluster A:
    Label: 1, Portion: 0.00149253731343
    Label: 0, Portion: 0.879104477612
    Label: 3, Portion: 0.119402985075

Cluster C:
    Label: 1, Portion: 0.0111731843575
    Label: 0, Portion: 0.815642458101
    Label: 3, Portion: 0.106145251397
    Label: 2, Portion: 0.0670391061453

Cluster B:
    Label: 1, Portion: 0.676923076923
    Label: 0, Portion: 0.00769230769231
    Label: 3, Portion: 0.0307692307692
    Label: 2, Portion: 0.284615384615

Cluster D:
    Label: 0, Portion: 0.761904761905
    Label: 2, Portion: 0.238095238095
```

RMSE: 0.258822076078

Part (B)

For this portion, we implement a perturbation intitialization that displaces each cluster from the "uniform" data centroid by random amounts. We specify `k=2` clusters for this question.

In [ ]:
```
!python kmeans.py 'perturbation' 2
```

Running initializeCentroids... No handlers could be found for logger "mrjob.runner" Complete: initializeCentroids

Iteration 0.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 1.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError

Running diagnostics...
Complete: diagnostics
```

Purity characteristics:

```
Cluster B:
    Label: 1, Portion: 0.091
    Label: 0, Portion: 0.752
    Label: 3, Portion: 0.103
    Label: 2, Portion: 0.054
```

RMSE: 0.282547403061

Part (C)

We do a similar initialization here, except we ask for `k=4` clusters using the same perturbation method.

In [3]:
```
!python kmeans.py 'perturbation' 4
```

Running initializeCentroids... No handlers could be found for logger "mrjob.runner" Complete: initializeCentroids

Iteration 0.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 1.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 2.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 3.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError

Running diagnostics...
Complete: diagnostics
```

Purity characteristics:

```
Cluster C:
    Label: 1, Portion: 0.661654135338
    Label: 0, Portion: 0.00751879699248
    Label: 3, Portion: 0.0300751879699
    Label: 2, Portion: 0.300751879699

Cluster D:
    Label: 1, Portion: 0.00346020761246
    Label: 0, Portion: 0.866205305652
```

Part (D)

Finally, we use the "trained" initialization here to initialize centroids over the class labels.

In [ ]:
```
!python kmeans.py 'trained' 4
```

Running initializeCentroids... No handlers could be found for logger "mrjob.runner" Complete: initializeCentroids

Iteration 0.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 1.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 2.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError
```

Iteration 3.

```
Running assignClusters...
Complete: assignClusters

Running updateCentroids...
Complete: updateCentroids

Running getError...
Complete: getError

Running diagnostics...
Complete: diagnostics
```

Purity characteristics:

```
Cluster A:
    Label: 1, Portion: 0.00373599003736
    Label: 0, Portion: 0.932752179328
    Label: 3, Portion: 0.0473225404732
    Label: 2, Portion: 0.0161892901619

Cluster C:
    Label: 1, Portion: 0.44578313253
    Label: 0, Portion: 0.0120481927711
    Label: 3, Portion: 0.0481927710843
    Label: 2, Portion: 0.493975903614

Cluster B:
    Label: 1, Portion: 1.0

Cluster D:
    Label: 0, Portion: 0.031746031746
    Label: 3, Portion: 0.968253968254
```

RMSE: 0.255077922595

Discussion

The results of the four different initializations fit our intuition. The trained initialization did the best, with the lowest `RMSE` of the four. However, it contains a singleton cluster and is symptomatic of either poor initialization or skewed data. Since the lexigraphical data follows Zipf's law, a singleton, or especially pure clusters are unsurprising.

The two perturbation initializations perform the worst, and this is perhaps unsurprising. Given that the data is normalized prior to perturbation, adding random numbers with 0-mean is may have a dramatic impact on their sucessful path through 'updating', as the data is highly skewed. In addition, both perturbation initializations saw their clusters reduced to half of their original count, which suggests that perhaps more clustered are needed for an accurate classification to be produced.

Finally, the uniform initialization did only slightly worse than the trained initialization. This is also unsurprising, as we are picking random customers to represent our cluster centers, then updating from there. For this reason, none of the clusters will drop through the update process, as the original centroid point is going to be clusters going forward.

In [ ]: