DATASCI W261: Machine Learning at Scale

Nick Hamlin (nickhamlin@gmail.com)
Tigi Thomas (tgthomas@berkeley.edu)
Rock Baek (rockb1017@gmail.com)
Hussein Danish (husseindanish@gmail.com)

Time of Submission: 9:23 PM EST, Wednesday, Feb 10, 2016
W261-3, Spring 2016
Week 4 Homework

Submission Notes:

- For each problem, we've included a summary of the question as posed in the instructions. In many cases, we have not included the full text to keep the final submission as uncluttered as possible. For reference, we've included a link to the original instructions in the "Useful Reference" below.
- Problem statements are listed in *italics*, while our responses are shown in plain text.
- We've included the full output of the mapreduce jobs in our responses so that counter results are shown. However, these don't always render nicely into PDF form. In these situations, please reference the complete rendered notebook on Github (https://github.com/nickhamlin/mids_261_homework/blob/master/HW4/MIDS-W261-2015-HWK-Week04-Hamlin-Thomas-Baek-Danish.ipynb)

Useful References:

- **Original Assignment Instructions (https://www.dropbox.com/sh/m0nxsf4vs5cyrp2/AACYOZQ3hRyGtHoPt33ny_Pza/HW Questions.txt?dl=0)**
- Most frequent word example in mrjob (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/nd2wow1t3y77jqk/MrjobMostUsedWorc
- kmeans example in mrjob (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/5qwejmygaievrzt/MrJobKmeans.ipynb)
- Microsoft anonymous web data background info (https://kdd.ics.uci.edu/databases/msweb/msweb.html)

HW4.0.

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

MrJob is a convenient, easy to use MapReduce library implemented in Python. The MrJob library simplifies writing and running of Hadoop Streaming jobs.

With the standard Hadoop MapReduce paradigm using Hadoop Streaming, one has to provide separate Mapper and Reducer scripts/code and invoke the streaming job providing one such mapper and readucer at a time. Althought this provides much control over the process, pipelining multiple Map and Reduce steps, or iteratively calling the same map-reduce tasks become very cumbersome. MrJob simplifies this, by allowing developers to write one Map Reduce program with the mapper and reducer as different methods in a MapReduce class. This allows for very convenient testing, debugging and considerably simplifies the creation and execution of MapReduce Job pipelines.

MrJob can be executed even without installing Hadoop providing a perfect platform for prototyping. The code will then simply work within a Hadoop setting requiring no further code changes. MrJob also has extensive integration with Amazon Elastic MapReduce and the same code can be run on Amazon EMR with just a few configuration settings. For more information, see the [MRJob source code (https://github.com/Yelp/mrjob)](https://github.com/Yelp/mrjob) and the [corresponding docs (https://pythonhosted.org/mrjob/guides/why-mrjob.html)](https://pythonhosted.org/mrjob/guides/why-mrjob.html).

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

With MrJob, you write implementation scripts for your Mapper and Reducer as methods of a subclass of MRJob. This script is then invoked once per task by Hadoop Streaming, which starts your script, feeds it stdin, reads stdout, and finally closes it. Based on how you have defined your mapper and reducer step functions MrJob will invoke each of them.

However, it is common to require some initialization or finalization code to be run before or after the various mapper / reducer steps. MrJob lets you write such start-up and tear-down methods to run at the beginning (_init()) and end (_final() of the various mapper/reducer process: via the *_init()* and *_final()* methods:

These methods can be used to load support files and or write out intermediate files during the various map and reduce steps. This allows for efficient sharing of common files within the same node while it processes different data chunks.

HW4.1

What is serialization in the context of MrJob or Hadoop?

Serialization is the process of turning structured objects (eg. an instance of an object oriented class) into a byte stream for transmission over a network or for writing to persistent storage. Serialization appears in two quite distinct areas of distributed data processing:

- For inter-process communication - often via RPC (Remote Procedure Calls) where objects are serialized for efficient transmission over the network from one process to the other.

- For persistent storage - objects are serialized to disk for efficient storage.

Objects that are serialized for storage or transmission can be deserialized, which is the reverse process of turning a byte stream back into a series of structured objects. In distributed computing, the distributed/connected processing nodes often serialize data to pass over the network to another node and the receiving node deserializes to load the object back as instance.

When it used in these frameworks?

Although the Hadoop framework uses Serialization extensively, within the MrJob implementation serialization is used in a limited fashion. For eg., Input and Outputs in MrJob are not serialized - they have to adhere to the Raw Text or JSON protocols. However, for internal transfers between the various mappers and reducers, MrJob suppors the binary Pickle Value protocol.

Since the Hadoop MapReduce paradigm works in a distributed fashion to process multiple chunks of data, transferring data over the network is unavoidable and Hadoop accomplishes this interprocess communication between nodes via remote procedure calls or RPCs. The RPC protocol uses serialization to make the message into a binary stream to be sent to the remote node, which receives and deserializes the binary stream into the original message.

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

The default serialization mode for input data is raw values (lines of raw text without keys). For internal information transfer between job steps, as well as final output, MRJob uses a JSON protocol. Though custom protocols can be written, MRjob does not support binary serialization. Below is a summary of the serialization options available in MrJob.

- 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

Problem Statement

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

Implementation

We can solve this problem simply by iterating through the file. Because the rows are in order, every time we encounter a new visitor, we can save their ID to be applied to each subsequent view record until a new visitor record is reached. Also, while it's not explcitly asked for in this problem, we'll run a second batch of code to save the clean URL data to its own file since we'll need this data for HW 4.4.

In [ ]:
```
%%writefile convert_msdata.py
#HW 4.2 - Attach customer IDs to page view records

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
```

In [ ]:
```
%%writefile create_urls.py
#HW 4.2 - Extract URLs (not explicitly required, but for later use in 4

#Save only results from 'A' rows into their own file for easy URL acces
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]
```

In [136]:
```
#Make files executable, convert data, and view some example results to
#!chmod +x convert_msdata.py create_urls.py
!python convert_msdata.py > clean_msdata.txt
!cat clean_msdata.txt | head -10
!python create_urls.py > ms_urls.txt
!cat ms_urls.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: stdout: Broken pipe
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
```

HW 4.3

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

To do this, we have two separate jobs. The first extracts each page view from the dataset in the mapper and the reducer simply aggregates these counts together and emits one row per page with the total number of views as the value. The second job handles the sorting. Normally in this situation, we'd use an identity mapper/reducer and handle the sorting via the shuffle, but in this case we've moved this logic into the reducer because of the bugs in MRJob's secondary sort when running jobs locally.

In [1]:
```
#Use this to make sure we reload the MrJob code when we make changes
%load_ext autoreload
%autoreload 2
```

In [621]:

```
%%writefile top_pages.py
#HW 4.3 - MRJob Definition
import csv

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


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

class TopPages(MRJob):

# Normally, we'd use the shuffle to do the sort.  However, the bug
# in comparators when running local MRJobs makes this an untenable solu
# so we'll settle for doing the sort in the second-stage reducer instea

#     def jobconf(self):
#         orig_jobconf = super(TopPages, self).jobconf()
#         custom_jobconf = {  #key value pairs
#             'mapred.output.key.comparator.class': 'org.apache.hadoop.
#             'mapred.text.key.comparator.options': '-k1,1nr',
#             'mapred.reduce.tasks': '1',
#         }
#         combined_jobconf = orig_jobconf
#         combined_jobconf.update(custom_jobconf)
#         self.jobconf = combined_jobconf
#         return combined_jobconf

    def mapper_extract_views(self, line_no, line):
        """Extracts the Vroot that was visited"""
        cell = csv_readline(line)
        if cell[0] == 'V':
            yield cell[1],1

    def reducer_sum_views(self, vroot, visit_counts):
        """Sumarizes the visit counts by adding them together,yield the
        total = sum(i for i in visit_counts)
        yield None,(total, vroot)


    # discard the key; it is just None
    def reducer_find_top_views(self,_, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word

        output=sorted(word_count_pairs)[-5:]
        output.reverse()
        for i in output:
            yield (i[1],i[0])
    def steps(self):  #pipeline of Map-Reduce jobs
        return [
            MRStep(mapper=self.mapper_extract_views,       # STEP 1: vi
                    reducer=self.reducer_sum_views) ,
            MRStep(reducer=self.reducer_find_top_views) # Step 2: sort
        ]

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

```
Overwriting top_pages.py
```

In [ ]:
```
#Make file executable if it's not already
!chmod +x top_pages.py
```

In [622]:
```
#HW 4.3 - Driver Function
from top_pages import TopPages
import csv

def run_4_3():
    mr_job = TopPages(args=['clean_msdata.txt'])
    with mr_job.make_runner() as runner:
        runner.run()
        for line in runner.stream_output():
            print mr_job.parse_output_line(line)

run_4_3()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
```

```
('1008', 10836)
('1034', 9383)
('1004', 8463)
('1018', 5330)
('1017', 5108)
```

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.

Here we use a single job. The mapper extracts page views from the data along with the corresponding visitor id, while the reducer aggregates the results together, locates the most frequent visitor to each page, and returns the page, visitor, view count, and URL. In addition, we use a reducer_init step to make sure that the reducer has access to the URL data in memory, since it's stored in a separate file.

In [10]:
```
%%writefile freq_visitor.py
# HW 4.4 - MRJob Code

import csv
from collections import Counter
from operator import itemgetter

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


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

class FreqVisitor(MRJob):

    def mapper_extract_views(self, line_no, line):
        """Extracts the page that was visited and the visitor id"""
        cell = csv_readline(line)
        #Ignore any irrelevant messy data, though hopefully we don't ha
        if cell[0] == 'V':
            yield cell[1],cell[4]

    def reducer_load_urls(self):
        """Load file of page URLs into reducer memory"""
        with open('ms_urls.txt','rb') as f:
            urls=csv.reader(f.readlines())
        self.url_dict={}
        for i in urls:
            #Saving the URLs into a dictionary will make it easy to acc
            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"""
        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_extract_views,
                       reducer_init=self.reducer_load_urls,
                       reducer=self.reducer_sum_views_by_visitor)]

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

```
Overwriting freq_visitor.py
```

In [22]:
```
#HW 4.4 - Driver Function
from freq_visitor import FreqVisitor
import csv

def run_4_4():
    mr_job = FreqVisitor(args=['clean_msdata.txt','--file','ms_urls.txt
    with mr_job.make_runner() as runner:
        runner.run()
        print "======== RESULTS =========="
        print "PAGE | VISITOR ID | # VISITS | PAGE URL "
        print "----------------------------------------"
        for line in runner.stream_output():
            output=mr_job.parse_output_line(line)
            #This code looks a little weird, but makes the output easie
            print str(output[0])+'  '+str(output[1][0])+'        '+str(

run_4_4()
```

```
1120  10241        1            /switch
1121  41018        1            /magazine
1122  25185        1            /mindshare
1123  22506        1            /germany
1124  30187        1            /industry
1125  27594        1            /imagecomposer
1126  10272        1            /mediamanager
1127  39790        1            /netshow
1128  10286        1            /msf
1129  20067        1            /ado
1130  18495        1            /syspro
1131  40053        1            /moneyzone
1132  20613        1            /msmoneysupport
1133  32647        1            /frontpagesupport
1134  16071        1            /backoffice
1135  33243        1            /mswordsupport
1136  33329        1            /usa
1137  16470        1            /mscorp
1138  17452        1            /mind
1139  23476        1            /k-12
```

HW 4.5

Problem Statement

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the 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, classifieds postings, advertisements, etc...).

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

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

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

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342)
http://arxiv.org/abs/1508.01843 (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 users by 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 distributions are generally heavy-tailed power-laws (often called Zipf distributions), and are very rare in the larger 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 (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.

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, and discuss your findings and any differences in outcomes across parts A-D.

Note that you do not have to compute the aggregated distribution or the class-aggregated distributions, which are rows in the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt

HW 4.5 - Setting up the mrjob

First, we modify the MRJob class to run a single iteration of the K-Means algorithm. The mapper_init function takes a text file containing initial centroid positions and loads it into memory on each mapper.

Next, the mapper method runs the expectation step and emits a record for each point in the main dataset and its corresponding cluster assignment based on the current centroid locations. This is done using the helper function from the class example that we have to make sure to define in advance. The mapper's output also includes the points actual class so that we can evaluate the results of our cluster at the end.

The maximization step takes place in the reducer, which aggregates results for each predicted class and computes the new corresponding centroid location. A combiner sits between the mapper and reducer to help with intermediate aggregation. Finally, the new centroid locations are written back to disk.

In [33]:

```
%%writefile mrkmeans.py
from __future__ import division
from math import sqrt
from operator import itemgetter
from collections import Counter

import numpy as np

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

#Calculate find the nearest centroid for data point
def MinDist(datapoint, centroid_points):
    datapoint = np.array(datapoint)
    centroid_points = np.array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = np.argmin(list(diffsq.sum(axis = 1)))
    return minidx

class MRKmeans(MRJob):

    def __init__(self, *args, **kwargs):
        super(MRKmeans, self).__init__(*args, **kwargs)
        #Initializing these values here makes them available to the cla
        self.k = 0 #Number of clusters to create
        self.centroid_points=[] #List of centroid vectors

    def steps(self):
        return [
            MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer
            )
        ]

    def mapper_init(self):
        """
        Load locations of existing centroids into memory as a list with
        """
        self.centroid_points=[map(float,s.split('\n')[0].split(',')) fo
        open('Centroids.txt','w').close() #This wipes the file once we'
        self.k=len(self.centroid_points)

    def mapper(self,_,line):
        """
        For each point sent through the stream:
        - Normalize each point by the total number of words in the docu
        - Calculate the closest centroid
        - Emit records where...
            -Key=(<current cluster asst>,<correct cluster asst>)
            [illegible]
            -Value=(1,<normalized vector for that point>)
        """

        line=line.split(',')
        line_id,cluster,total_words=int(line[0]),int(line[1]),float(lin
        D=(map(float,line[3:])) #Convert point to floats
        D=[i/total_words for i in D] #Normalize by total words
        idx=int(MinDist(D,self.centroid_points)) #Calculate closest cen
        class_counts=np.zeros(4) #Pass actual cluster assignment throug
        class_counts[cluster]+=1
        yield idx,(list(class_counts),1,D) #We convert the class_counts

    def combiner(self,idx,inputdata):
        """
        For each row sent by the mapper, calculate partial sum for new
        - Initialize a blank 1000 element list
        - Add all intermediate values together for that list
        - Emit records where...
            -Key=Index of centroid that should be updated with the asso
            -Value=(<number of points represented in the vector>,<vecto
        """

        temp_row=np.zeros(1000) #Initialize aggregated vector
        num=0
        class_counts=np.zeros(4)
        for v in inputdata: #Calculate intermediate sums
            class_counts+=v[0] #records will come in with a real cluste
            num+=v[1]
            temp_row+=v[2]
        yield idx,(list(class_counts),num,list(temp_row))

    def reducer(self,idx,inputdata):
        """
        For each incoming row:
        - Calculate final sum of vector elements using the same approac
        - Divide by the number of points in the cluster to calculate th
        - Store updated centroids to disk
        - Emit location of new centroids
        """
        centroid=np.zeros(1000)
        class_counts=np.zeros(4)
        num=0

        for v in inputdata:
            class_counts+=v[0] #Aggregate actual class assignments cont
            num+=v[1] #Track total word count for normalization
            centroid+=v[2]

        centroid/=num #Normalize aggregated new centroid vector by numb

        #Save new centroid locations to file
        with open('Centroids.txt','a') as f:
            f.writelines(','.join(map(str,centroid))+'\n')
        yield idx,(list(class_counts),list(centroid))
```

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

```
Overwriting mrkmeans.py
```

HW 4.5 - Running iterative MRJobs

Once we've established our kmeans class, we need to set up a driver structure to run it and make sense of the results. First, we define a stopping criterion based on the class example that checks how much the centroids have moved from iteration to iteration. If this delta is above a threshold, we'll continue to iterate.

Next, we set up a function to run the job itself that accepts a list of centroid points and a value for K. This will make it possible to recycle our code to answer each of the four questions posted. The main function will save the starting centroids to disk, then repeatedly call the MRJob we defined above until our stopping criterion is met. At this point, class summaries are calculated and displayed.

In [129]:

```
### K-Means Driver Code
from __future__ import division
from itertools import chain

from numpy import random

from mrkmeans import MRKmeans

def stop_criterion(centroid_points_old, centroid_points_new,T):
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag

def run_kmeans_mrjob(centroid_points,k):
    source='topUsers_Apr-Jul_2014_1000-words.txt'

    #Set up job and save centroids to file
    mr_job=MRKmeans(args=[source,'--file', 'Centroids.txt'])
    with open('Centroids.txt','w+') as f:
        f.writelines(','.join(str(j) for j in i)+'\n' for i in centroid

    #Update centroids iteratively
    i=0 #Track which iteration we're on
    while(1):
        output=[] #Initialze destination for our final results
        centroid_points_old=centroid_points[:]
        with mr_job.make_runner() as runner:
            runner.run() #stream output
            for line in runner.stream_output():
                key,value=mr_job.parse_output_line(line)
                output.append((key,value[0])) #Save our temp results.
                centroid_points[key]=value[1]
        i+=1

        #Check if stop criterion is satsfied.
        if stop_criterion(centroid_points_old,centroid_points,0.001):

            overall_total=0
            overall_max=0

            #Print final results
            print "=========RESULTS============"
            print "k-means converged after {0} iterations\n".format(str
            print ''
            print 'CLASS COUNTS BY CLUSTERS'
            print '(Rows are predicted clusters, columns are actual clu
            print '-----------------------------------------------------'
            for i,v in enumerate(output):
                output_array=np.array(v[1])
                total=sum(v[1])
                overall_total+=total
                ratios=output_array/total
                purity=max(v[1])/total
                overall_max+=max(v[1])
                print '{0} ||  {1:3.0f}   |  {2:3.0f}   |  {3:3.0f}   |
                print '{0} || ({1:0.2f}) | ({2:0.2f}) | ({3:0.2f}) | ({
                print '-------------------------------------------------
            print 'OVERALL PURITY:{0:0.2f}'.format(overall_max/overall_

            break
    return output
```

HW 4.5 - Part A

K=4 uniform random centroid-distributions over the 1000 words (generate 1000 random numbers and normalize the vectors)

Now that we've laid all the groundwork, we can run our jobs. The only differences between each of the four parts in this problem are the values of K and what process we use to intialize our centroid locations. For this initial try, we'll choose four centroids completely at random.

```
In [131]: ####### PART A ############
          from __future__ import division
          import numpy as np

          def run_part_a():
              k=4
              centroid_points=[] #Initialize list of lists for to hold K starting
              for line in range(k):
                  row=np.random.random_integers(10000, size=(1000)) #Create a vec
                  total=sum(row)
                  normalized_row=row/total #Normalize the same way we have in the
                  centroid_points.append(normalized_row)

              run_kmeans_mrjob(centroid_points,k) #Run the jobs

          run_part_a()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

==========RESULTS=============
k-means converged after 9 iterations


CLASS COUNTS BY CLUSTERS
(Rows are predicted clusters, columns are actual clusters)
--------------------------------------------------
0 ||  749   |    3   |    5   |   42   |  Purity:
  || (0.94) | (0.00) | (0.01) | (0.05) |  0.94
--------------------------------------------------
1 ||    0   |    2   |    7   |    0   |  Purity:
  || (0.00) | (0.22) | (0.78) | (0.00) |  0.78
--------------------------------------------------
2 ||    2   |    0   |    6   |   57   |  Purity:
  || (0.03) | (0.00) | (0.09) | (0.88) |  0.88
--------------------------------------------------
3 ||    1   |   86   |   36   |    4   |  Purity:
  || (0.01) | (0.68) | (0.28) | (0.03) |  0.68
--------------------------------------------------
OVERALL PURITY:0.90
```

HW 4.5 - Part B

K=2, with centroids based on random perturbations from the user-wide distribution

Here, we use the intialization function provided in the updated problem statement, which returns K centroids based on random noise added to the overall distribution. These are returned as a list of lists that we can then use in our main k-means function

In [132]:
```
import re
# Setup function for centroids for part B
def startCentroidsBC(k):
    counter = 0
    for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").
        if counter == 2:
            data = re.split(",",line)
            globalAggregate = [float(data[i+3])/float(data[2]) for i in
        counter += 1
    ## 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 rang
        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
```

In [133]:
```
####### PART B ############
def run_part_b():
    k=2
    centroid_points=startCentroidsBC(k)
    run_kmeans_mrjob(centroid_points,k)

run_part_b()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
```

```
==========RESULTS=============
k-means converged after 5 iterations


CLASS COUNTS BY CLUSTERS
(Rows are predicted clusters, columns are actual clusters)
--------------------------------------------------
0 ||  751   |    3   |   14   |   99   |  Purity:
  || (0.87) | (0.00) | (0.02) | (0.11) |  0.87
--------------------------------------------------
1 ||    1   |   88   |   40   |    4   |  Purity:
  || (0.01) | (0.66) | (0.30) | (0.03) |  0.66
--------------------------------------------------
OVERALL PURITY:0.84
```

HW 4.5 - Part C

K=4, with centroids based on random perturbations from the user-wide distribution

We can recycle the same approach as in part B, and simply change the value of K

In [134]:
```
####### PART C ############
def run_part_c():
    k=4
    centroid_points=startCentroidsBC(k)
    run_kmeans_mrjob(centroid_points,k)

run_part_c()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

==========RESULTS=============
k-means converged after 7 iterations


CLASS COUNTS BY CLUSTERS
(Rows are predicted clusters, columns are actual clusters)
--------------------------------------------------
0 ||  291   |    0   |    0   |   78   |  Purity:
  || (0.79) | (0.00) | (0.00) | (0.21) |  0.79
--------------------------------------------------
1 ||    1   |   86   |   36   |    4   |  Purity:
  || (0.01) | (0.68) | (0.28) | (0.03) |  0.68
--------------------------------------------------
2 ||    0   |    2   |   14   |    0   |  Purity:
  || (0.00) | (0.12) | (0.88) | (0.00) |  0.88
--------------------------------------------------
3 ||  460   |    3   |    4   |   21   |  Purity:
  || (0.94) | (0.01) | (0.01) | (0.04) |  0.94
--------------------------------------------------
OVERALL PURITY:0.85
```

HW 4.5 - Part D

K=4, "trained" centroids, determined by the sums across the classes

This version involves pulling the initial centroids from the aggregated summary of the stats. We read each in, normalize by the total words in the class, and output the result as our class centroid.

In [135]:
```
####### PART D ############
from csv import reader

def run_part_d():
    k=4
    centroid_points=[] #Initialize list of lists for to hold K starting
    source='topUsers_Apr-Jul_2014_1000-words_summaries.txt'
    users=list(open(source))
    for line in reader(users[2:]): #Skip the first two lines since we c
        line_id,cluster,total_words=line[0],line[1],float(line[2])
        D=(map(float,line[3:]))
        D=[i/total_words for i in D]
        centroid_points.append(D)

    run_kmeans_mrjob(centroid_points,k)

run_part_d()
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocol
s will be strict by default. It's recommended you run your job with
--strict-protocols or set up mrjob.conf as described at https://pyth
onhosted.org/mrjob/whats-new.html#ready-for-strict-protocols (http
s://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocol
s)
WARNING:mrjob.runner:

==========RESULTS=============
k-means converged after 5 iterations


CLASS COUNTS BY CLUSTERS
(Rows are predicted clusters, columns are actual clusters)
-----------------------------------------------------
0 ||  749   |    3   |   14   |   38   |  Purity:
  || (0.93) | (0.00) | (0.02) | (0.05) |  0.93
-----------------------------------------------------
1 ||    0   |   51   |    0   |    0   |  Purity:
  || (0.00) | (1.00) | (0.00) | (0.00) |  1.00
-----------------------------------------------------
2 ||    1   |   37   |   40   |    4   |  Purity:
```

```
2 ||     1    |    37    |    40    |     4    |  Purity:
  || (0.01) | (0.45) | (0.49) | (0.05) |  0.49
--------------------------------------------------------
3 ||     2    |     0    |     0    |    61    |  Purity:
  || (0.03) | (0.00) | (0.00) | (0.97) |  0.97
--------------------------------------------------------
OVERALL PURITY:0.90
```

HW 4.5 - Discussion of Results

We have reasonable purity in Part A (random centroids), though the algorithm takes a while to converge. In contrast, Part B has a lower value for K, and therefore takes less time to converge. However, the purity is significantly lower, which makes sense since we know there are actually four clusters in the source data. Part C converges faster than Part A, but produces lower purity. This may be because we're initializing our clusters randomly based on the aggregate of all four clusters, and so the initial centroid positions are "too close" to the overall average and miss some of the points on the periphery. Unsurprisingly, our best result comes in Part D, where our intial centroid positions are based on the known class aggregate centroids. Not only does this model generate the same purity as Part A, but it also converges in about half the time.

End of Submission