DATASCI W261: Machine Learning at Scale

Assignment Week 4

Minhchau Dang (minhchau.dang@berkeley.edu)
Miki Seltzer (miki.seltzer@berkeley.edu)
W261-2, Spring 2016
Submission: 2016-Feb-10 9:30pm

HW 4.0

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

MRJob is a framework provided by Yelp as an alternate way to write Hadoop streaming jobs.

It allows you to write the mappers, combiners, and reducers as a single Python class. In doing so, it eliminates the boilerplate line parsing code found in Hadoop streaming jobs. It also provides a "map reduce step" (`MRStep`) abstraction that allows you to chain together a pre-determined set of map reduce steps.

MRJob is more limited than standard MapReduce in the sense that only those abstractions that have been implemented in the framework are available (for example, secondary sort does not work cleanly).

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

`mapper_init`, `combiner_init`, and `reducer_init` are called before the mapper, combiner, and reducer phases, respectively. `mapper_final`, `combiner_final`, and `reducer_final` are called after the mapper, combiner, and reducer phases, respectively.

These functions are used as part of the setup and tear-down of a MapReduce phase. They are equivalent to the commands that run before and after the stream iteration loop in a traditional Hadoop streaming job.

HW 4.1

What is serialization in the context of MrJob or Hadoop?

Serialization is the process of encoding the data between each of the different phases of a MapReduce job.

When it used in these frameworks?

Serialization is used whenever data is written to persistent storage (such as in creating spill files or creating output files) or transmitted over over the network.

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

- The default serialization mode for the input protocol (data sent during the first mapping phase) is `RawValueProtocol`, which assumes that there are no keys and the whole line should be treated as the value.
- The default serialization mode for internal communication (data sent between phases) is `JSONProtocol` which encodes the data in JSON.
- The default serialization mode for the output protocol (the final step) is `JSONProtocol` which encodes the data in JSON.

HW4.2: Write the Python code to preprocess the Microsoft logfile data on a single node into the appropriate format.

In [17]:
```
import csv

# Create files to write the processed data to
outfile = open('anonymous-msweb-preprocess.data', 'wb')
outWriter = csv.writer(outfile)
attributes = open('attributes.csv', 'wb')
attWriter = csv.writer(attributes)

with open('anonymous-msweb.data', 'r') as infile:
    for line in csv.reader(infile):

        # Check if this line is an attribute
        # If it is, write to the attributes file
        if line[0] == 'A':
            attWriter.writerow([line[1], line[3], line[4]])

        # Check if this line is a visitor ID
        # If it is, save it to memory
        elif line[0] == 'C':
            visitor_id = line[2]

        # Check if this line is a visit with a page ID
        # If it is, write it with the current visitor_id
        elif line[0] == 'V':
            outWriter.writerow([line[0], line[1], line[2], 'C', visitor_id])

outfile.close()
attributes.close()
```

In [1]:
```
# Test out our code above
!head anonymous-msweb-preprocess.data
```

```
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
```

HW 4.3: Find the 5 most frequently visited pages using MrJob from the output of 4.2 (i.e., transformed log file).

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [55]:
```
%%writefile mr_pageCount.py
from mrjob.job import MRJob
from mrjob.step import MRStep
import csv

def read_csvLine(line):
    # Given a comma delimited string, return fields
    for row in csv.reader([line]):
        return row

class MRPageFreqCount(MRJob):

    # Mapper1: emit page_id, 1
    def mapper_count(self, _, line):
        fields = read_csvLine(line)
        yield fields[1], 1

    # Combiner1: aggregate page_id
    def combiner_count(self, page, counts):
        yield page, sum(counts)

    # Reducer1: aggregate page_id
    def reducer_count(self, page, counts):
        yield page, sum(counts)

    # Mapper2: invert page and counts to sort
    def mapper_sort(self, page, counts):
        yield '%010d' % counts, page

    # Reducer2: identity
    def reducer_sort(self, counts, page):
        for p in page:
            yield int(counts), p

    # Multi-step pipeline definition
    def steps(self):
        return [
            MRStep(mapper=self.mapper_count,
                   combiner=self.combiner_count,
                   reducer=self.reducer_count),
            MRStep(mapper=self.mapper_sort,
                   reducer=self.reducer_sort)]



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

```
Overwriting mr_pageCount.py
```

In [67]:
```
from mr_pageCount import MRPageFreqCount

mr_job = MRPageFreqCount(args=['anonymous-msweb-preprocess.data'])
output = []

with mr_job.make_runner() as runner:
    # Run MRJob
    runner.run()

    # Write stream_output to file
    for line in runner.stream_output():
        output.append(mr_job.parse_output_line(line))
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
```

In [72]:
```
print '{:8s}{:>7s}'.format('page_id', 'count')
print '----------------'
for i in range(5):
    print '{:8s}{:7,d}'.format(output[-i-1][1], output[-i-1][0])
```

```
page_id   count
----------------
1008     10,836
1034      9,383
1004      8,463
1018      5,330
1017      5,108
```

HW 4.4: Find the most frequent visitor of each page using MrJob and the output of 4.2 (i.e., transformed log file). In this output please include the webpage URL, webpageID and Visitor ID.

In [111]:
```
%%writefile mr_visitorCount.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from collections import Counter
import csv

def read_csvLine(line):
    # Given a comma delimited string, return fields
    for row in csv.reader([line]):
        return row

class MRTopVisitorCount(MRJob):

    # Mapper1: emit page_id, 1
    def mapper_count(self, _, line):
        fields = read_csvLine(line)
        yield fields[1], (fields[4], 1)

    # Reducer1: aggregate page_id
    def reducer_count(self, page, counts):
        count = Counter()
        for visitor in counts:
            count.update({visitor[0]:visitor[1]})
        yield page, count

    # Mapper2: invert page and counts to sort
    def mapper_sort(self, page, counts):
        top = Counter(counts).most_common(1)
        yield page, (top[0][0], top[0][1])

    # Reducer2: identity
    def reducer_sort(self, page, visitor_count):
        for v in visitor_count:
            yield page, v

    # Multi-step pipeline definition
    def steps(self):
        return [
            MRStep(mapper=self.mapper_count,
                   reducer=self.reducer_count),
            MRStep(mapper=self.mapper_sort,
                   reducer=self.reducer_sort)]



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

```
Overwriting mr_visitorCount.py
```

In [113]:
```
from mr_visitorCount import MRTopVisitorCount

mr_job = MRTopVisitorCount(args=['anonymous-msweb-preprocess.data'])
output = []

with mr_job.make_runner() as runner:
    # Run MRJob
    runner.run()

    # Write stream_output to file
    for line in runner.stream_output():
        output.append(mr_job.parse_output_line(line))
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
```

In [135]:
```
import csv

attributes = {}

with open('attributes.csv','r') as myfile:
    for fields in csv.reader(myfile):
        attributes[fields[0]] = fields[2]

print '{:10s}{:<28s}{:>11s}{:>7s}'.format('page id',
                                          'page URL',
                                          'visitor id',
                                          'count')
print '-------------------------------------------------------'
for item in output:
    print '{:10s}{:<28s}{:>11s}{:7d}'.format(item[0],
                                            attributes[item[0]],
                                            item[1][0],
                                            item[1][1])
```

```
page id   page URL                     visitor id  count
-------------------------------------------------------
1000      /regwiz                         36585      1
1001      /support                        35232      1
1002      /athome                         35235      1
1003      /kb                             35546      1
1004      /search                         35540      1
1005      /norge                          10004      1
1006      /misc                           27495      1
1007      /ie_intl                        19492      1
1008      /msdownload                     35236      1
1009      /windows                        16073      1
1010      /vbasic                         20915      1
1011      /officedev                      40152      1
1012      /outlookdev                     23657      1
1013      /vbasicsupport                  32727      1
1014      /officefreestuff                20914      1
1015      /msexcel                        15125      1
1016      /excel                          35542      1
1017      /products                       35234      1
1018      /isapi                          35237      1
1019      /mspowerpoint                   16765      1
1020      /msdn                           23991      1
1021      /visualc                        25260      1
1022      /truetype                       15906      1
1023      /spain                          16079      1
1024      /iis                            20447      1
1025      /gallery                        35234      1
1026      /sitebuilder                    23990      1
1027      /intdev                         15450      1
1028      /oledev                         11191      1
1029      /clipgallerylive                33083      1
1030      /ntserver                       22468      1
1031      /msoffice                       22505      1
1032      /games                          35542      1
1033      /logostore                      19483      1
1034      /ie                             35540      1
1035      /windowssupport                 35546      1
1036      /organizations                  22505      1
1037      /windows95                      19490      1
1038      /sbnmember                      32230      1
1039      /isp                            26948      1
1040      /office                         39325      1
1041      /workshop                       35234      1
1042      /vstudio                        40159      1
1043      /smallbiz                       33738      1
1044      /mediadev                       40224      1
1045      /netmeeting                     20917      1
1046      /iesupport                      18496      1
1048      /publisher                      33083      1
1049      /supportnet                     33329      1
1050      /macoffice                      30757      1
1051      /scheduleplus                   32702      1
1052      /word                           20914      1
1053      /visualj                        36585      1
1054      /exchange                       42285      1
1055      /kids                           33326      1
1056      /sports                         15907      1
1057      /powerpoint                     39792      1
1058      /referral                       19490      1
1059      /sverige                        33081      1
1060      /msword                         20914      1
1061      /promo                          20916      1
1062      /msaccess                       36585      1
1063      /intranet                       39793      1
```

Note that in this data set, every visitor goes to each page a maximum of one time. Thus, there really is no "most frequent visitor" for each page, since the top visitor is tied with every other visitor who has been to the page.

HW 4.5: Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users.

In [17]:
```
# First we need to normalize the data
infile = open('topUsers_Apr-Jul_2014_1000-words.txt', 'r')
outfile = open('topUsers_normalized.txt', 'w')

for line in infile:
    line = line.split(',')
    user_id = line[0]
    true_class = line[1]
    total = float(line[2])
    data = line[3:]
    data = [int(d) / total for d in data]
    outfile.write(','.join(str(j) for j in data) + '\n')

infile.close()
outfile.close()
```

In [2]:
```
from numpy import random
def generate_centroids(centroid_type, k):
    # Generate initial centroids using uniform distribution
    centroid_points = []
    num = []

    # We want to randomly select a user to initialize each cluster
    if centroid_type == "uniform":
        data = []
        with open('topUsers_normalized.txt', 'r') as myfile:
            for line in csv.reader(myfile):
                fields = [float(i) for i in line]
                data.append(fields)

        seeds = [random.randint(0,1000) for i in range(k)]

        for i in range(k):
            centroid_points.append(data[seeds[i]])

    # We want to find the centroid for each true class using the summary file
    elif centroid_type == "trained":
        with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt','r') as myfile:
            for line in csv.reader(myfile):
                if line[0] == 'CODE':
                    total = float(line[2])
                    point = [int(i)/total for i in line[3:]]
                    centroid_points.append(point)

    # We need to find the data-wide centroid first
    # Add random noise to this data-wide centroid to get the second centroid
    elif centroid_type == "perturbed":
        # Read in the user-wide aggregate, then normalize
        with open('topUsers_Apr-Jul_2014_1000-words_summaries.txt','r') as myfile:
            for line in csv.reader(myfile):
                if line[0] == "ALL_CODES":
                    total = float(line[2])
                    point = [int(i)/total for i in line[3:]]

        # Perturb the original centroid
        for i in range(k):
#             perturbed_centroid = [sum(x) for x in zip(point, [random.uniform(-1,1)/100 for i
            perturbed_centroid = [sum(x) for x in zip(point, random.sample(1000)/100)]
            normed = [i/sum(perturbed_centroid) for i in perturbed_centroid]
            centroid_points.append(normed)

    with open('Centroids.txt', 'w+') as f:
            f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

    return centroid_points
```

In [3]:
```
import csv
def report_composition():
    myfile = open('Centroids.txt','r')
    final_centroids = [map(float,s.split('\n')[0].split(',')) for s in myfile.readlines()]
    myfile.close()

    k = len(final_centroids)
    true_k = 4
    counts = [[0 for x in range(k)] for y in range(true_k)]
    pred_count = [0 for x in range(k)]
    cluster_id = ['A','B','C','D']

    with open('topUsers_Apr-Jul_2014_1000-words.txt') as myfile:
        for line in csv.reader(myfile):
            true_class = int(line[1])
            point = [float(i) for i in line[3:]]
            pred_class = int(MinDist(point, final_centroids))
            counts[true_class][pred_class] += 1.0
            pred_count[pred_class] += 1.0

    print '{:>4s}   |{:>10s}'.format('','pred')
    print '{:>4s}   |'.format('true'),
    for i in range(k):
        print '{:>10s}'.format(cluster_id[i]),
    print '\n-----------------------------------------------------',
    for i in range(true_k):
        print '\n{:4d}   |'.format(i),
        for j in range(k):
            print '{:10.2%}'.format(counts[i][j] / pred_count[j]),
    print '\n-----------------------------------------------------'
    print '{:4s}   |'.format(''),
    for i in range(k):
        print '{:10.2%}'.format(1),
```

Kmeans code provided:

In [15]:
```
%%writefile Kmeans.py
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRStep
from itertools import chain
import os

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

#Check whether centroids converge
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


class MRKmeans(MRJob):

    centroid_points=[]
    k = 4
    n = 1000

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

    #load centroids info from file
    def mapper_init(self):
        myfile = open('Centroids.txt','r')
        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in myfile.readline
        myfile.close()
        self.k = len(self.centroid_points)

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        value = list(D)
        value.append(1)
        yield int(MinDist(D,self.centroid_points)), tuple(value)

    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        sums = [0 for i in range(self.n+1)]
        for point in inputdata:
            for i in range(self.n+1):
                sums[i] += point[i]
        yield idx, tuple(sums)

    #Aggregate sum for each cluster and then calculate the new centroids
    def reducer(self, idx, inputdata):
        centroids = []
        num = [0]*self.k
        for i in range(self.k):
            centroids.append([0 for i in range(self.n)])
        for point in inputdata:
            count = float(point[-1])
            num[idx] += count
            for i in range(self.n):
                centroids[idx][i] += point[i]

        for i in range(len(centroids[idx])):
            centroids[idx][i] = centroids[idx][i]/num[idx]
        with open('/tmp/Centroids.txt', 'a') as f:
            f.writelines(','.join(str(j) for j in centroids[idx]) + '\n')
        yield idx, tuple(centroids[idx])
```

Driver provided:

In [16]:
```
from numpy import random
from Kmeans import MRKmeans, stop_criterion, MinDist
#mr_job = MRKmeans(args=['topUsers_normalized.txt', '-r', 'hadoop', '--hadoop-home','/usr/'])
mr_job = MRKmeans(args=['topUsers_normalized.txt', '--file', 'Centroids.txt'])

# Turn this into a function so that we can run it easily
# Function takes centroids created by generate_centroids function

def run_kmeans(centroid_points):
    !rm /tmp/Centroids.txt

    # Update centroids iteratively
    i = 0
    while(1):

        # save previous centoids to check convergency
        centroid_points_old = centroid_points[:]
#         print "iteration"+str(i)+":"
        with mr_job.make_runner() as runner:
            runner.run()

            # stream_output: get access of the output
            for line in runner.stream_output():
                key, value =  mr_job.parse_output_line(line)
#                 print key, value
                centroid_points[key] = value
#         print "\n"
        i = i + 1
        !mv /tmp/Centroids.txt .
        if(stop_criterion(centroid_points_old,centroid_points,0.001)):
            break

    report_composition()
    print "\n\nConverged after", i, "iterations"
#     print "Centroids\n"
#     print centroid_points
```

In [18]:
```
run_kmeans(generate_centroids("uniform", 4))
```

```
rm: cannot remove `/tmp/Centroids.txt': No such file or directory

WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:

       |      pred
true   |          A          B          C          D
-----------------------------------------------------
   0   |     97.31%      8.19%     85.63%     80.00%
   1   |      0.38%     52.05%      0.19%      0.00%
   2   |      0.77%     28.65%      0.00%      7.50%
   3   |      1.54%     11.11%     14.18%     12.50%
-----------------------------------------------------
       |    100.00%    100.00%    100.00%    100.00%

Converged after 5 iterations
```

In [19]:
```
run_kmeans(generate_centroids("perturbed", 2))
```

```
rm: cannot remove `/tmp/Centroids.txt': No such file or directory

WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:

       |      pred
true   |          A          B
-----------------------------------------------------
   0   |     89.88%     12.63%
   1   |      0.25%     46.84%
   2   |      0.37%     26.84%
   3   |      9.51%     13.68%
-----------------------------------------------------
       |    100.00%    100.00%

Converged after 6 iterations
```

In [20]:
```
run_kmeans(generate_centroids("perturbed", 4))
```

```
rm: cannot remove `/tmp/Centroids.txt': No such file or directory

WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:

       |      pred
true   |          A          B          C          D
-----------------------------------------------------
   0   |     95.94%      9.41%     20.29%     78.72%
   1   |      0.14%     52.35%      0.00%      2.13%
   2   |      0.14%     28.82%      0.00%      8.51%
   3   |      3.78%      9.41%     79.71%     10.64%
-----------------------------------------------------
       |    100.00%    100.00%    100.00%    100.00%

Converged after 8 iterations
```

In [21]:
```
run_kmeans(generate_centroids("trained", 4))
```

```
rm: cannot remove `/tmp/Centroids.txt': No such file or directory

WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by defau
WARNING:mrjob.runner:

       |      pred
true   |          A          B          C          D
-----------------------------------------------------
   0   |     98.13%      0.00%     33.33%     17.98%
   1   |      0.00%     94.44%     24.69%      0.00%
   2   |      0.43%      5.56%     29.63%      0.00%
   3   |      1.44%      0.00%     12.35%     82.02%
-----------------------------------------------------
       |    100.00%    100.00%    100.00%    100.00%

Converged after 5 iterations
```

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

The trained centroids have the highest purity out of the 4 different centroid initializations. However, it is unlikely that we will have the true class of the documents that we are clustering on -- thus it is not a methodology that we can reliably use in all situations.

Choosing random cluster intializations from a uniform distribution have wildly different results depending on the initial clusters chosen. It is easy to see that one could get significantly different results due to random differences in the points chosen as cluster seeds.

Comparatively, the perturbation centroids did not vary as widely as part A. When only two clusters were used, the highest percentage class in each cluster was class 0 (human). However, when four clusters were used, the four clusters each had a different highest percentage class.