Table of Contents

In [78]:
```
%%javascript
/***********************************************************************************
*****************
Known Mathjax Issue with Chrome - a rounding issue adds a border to the right
 of mathjax markup
https://github.com/mathjax/MathJax/issues/1300
A quick hack to fix this based on stackoverflow discussions:
http://stackoverflow.com/questions/34277967/chrome-rendering-mathjax-equations
-with-a-trailing-vertical-line
***********************************************************************************
****************/

$('.math>span').css("border-left-color","transparent")
```

MIDS - w261 Machine Learning At Scale

Course Lead: Dr James G. Shanahan (**email** Jimi via James.Shanahan *AT* gmail.com)

Assignment - HW4

Name: David Skarbrevik
Class: MIDS w261 Summer 2017 Section 1
Email: skarbrevik@iSchool.Berkeley.edu
StudentId 3032124317 **End of StudentId**
Week: 4

Due Time: HW is due the Tuesday of the following week by 8AM (West coast time). I.e., Tuesday, Feb 7, 2017 in the case of this homework.

Table of Contents

1 Instructions

Back to Table of Contents

MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #4

Version 2017-26-1

IMPORTANT

This homework can be completed locally on your computer

=== INSTRUCTIONS for SUBMISSIONS ===

Follow the instructions for submissions carefully.

NEW: Going forward, each student will have a `HW-<user>` repository for all assignments.

Click this link to enable you to create a github repo within the MIDS261 Classroom: https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8 (https://classroom.github.com/assignment-invitations/3b1d6c8e58351209f9dd865537111ff8) and follow the instructions to create a HW repo.

Push the following to your HW github repo into the master branch:

- Your local HW4 directory. Your repo file structure should look like this:

```
HW-<user>
    --HW3
       |__MIDS-W261-HW-03-<Student_id>.ipnb
       |__MIDS-W261-HW-03-<Student_id>.pdf
       |__some other hw3 file
    --HW4
       |__MIDS-W261-HW-04-<Student_id>.ipnb
       |__MIDS-W261-HW-04-<Student_id>.pdf
       |__some other hw4 file
    etc..
```

2 Useful References

Back to Table of Contents

- See async lecture and live lecture

HW Problems

Back to Table of Contents

HW4.0

Back to Table of Contents

What is MrJob? How is it different to Hadoop MapReduce? What are the mapper_init, mapper_final(), combiner_final(), reducer_final() methods? When are they called?

MRJob is a framework for running Hadoop MapReduce jobs in a python environment. It allows a very expressive way to specify jobs and simplifies the act of performing multiple jobs.

The mapper_init() method is a refers to code that can be executed prior to any map task executing, while the mapper_final() refers to code that can be executed after a map task has completed. The same logic applies to combiner_final() and reducer_final() for the combiner and reducer stages respectively.

HW4.1

Back to Table of Contents What is serialization in the context of MrJob or Hadoop? When it used in these frameworks? What is the default serialization mode for input and outputs for MrJob?

In [1]:
```
%%bash
curl -L http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/ano
nymous-msweb.data -o anonymous-msweb.data
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
nt
                                 Dload  Upload   Total   Spent    Left  Speed
100 1389k  100 1389k    0     0  1821k      0 --:--:-- --:--:-- --:--:-- 2096
k
```

In [ ]:
```
```

HW4.2 Preprocess log file data

Back to Table of Contents

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

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

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

Data Format

```
The data is in an ASCII-based sparse-data format called "DST". Each line of the dat
a file starts with a letter which tells the line's type. The three line types of in
terest are:
-- Attribute lines:
For example, 'A,1277,1,"NetShow for PowerPoint","/stream"'
Where:
  'A' marks this as an attribute line,
  '1277' is the attribute ID number for an area of the website (called a Vroot),
  '1' may be ignored,
  '"NetShow for PowerPoint"' is the title of the Vroot,
  '"/stream"' is the URL relative to "http://www.microsoft.com (http://www.microsof
t.com)"



Case and Vote Lines:
For each user, there is a case line followed by zero or more vote lines.
For example:
  C,"10164",10164
  V,1123,1
  V,1009,1
  V,1052,1
Where:
  'C' marks this as a case line,
  '10164' is the case ID number of a user,
  'V' marks the vote lines for this case,
  '1123', 1009', 1052' are the attributes ID's of Vroots that a user visited.
  '1' may be ignored.
```

</PRE>

Here, you must transform/preprocess the data on a single node (i.e., not on a cluster of nodes) from the following 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 following format (V, PageID, 1, C, Visitor):

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

Write the python code to accomplish this transformation.

In [1]:
```
!pwd
```

```
/media/notebooks
```

In [2]:
```
!wc -l anonymous-msweb.data
```

```
131666 anonymous-msweb.data
```

In [2]:
```
C_val = []
tmp_list = []

with open('anonymous-msweb.data', 'r') as file1, open('anonymous-msweb-preproc
essed.data', 'w') as file2:
    for line in file1.readlines():
        line = line.strip()
        line = line.split(',')
        if line[0] == "C":
            C_val = [line[0]]
            C_val.append(line[2])
        elif line[0] == "V":
            tmp_list = line + C_val
            file2.write(",".join(tmp_list))
            file2.write("\n")
```

In [3]:
```
!head -10 anonymous-msweb-preprocessed.data
!wc -l anonymous-msweb-preprocessed.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
98654 anonymous-msweb-preprocessed.data
```

HW4.3 Find the most frequent pages

Back to Table of Contents

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

In [19]:
```
%%writefile MostFrequentVisits.py
#!/usr/bin/env python

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

class MRWordFreqCount(MRJob):

    SORT_VALUES = True

    def mapper_get_urls(self, _, line):
        self.increment_counter('group', 'Num_mapper_calls', 1)
        line = line.split(',')
        yield (line[1], 1)

    def reducer_sum_urls(self, url, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield (sum(counts), url)



    def steps(self):
#         JOBCONF_STEP = {
#             'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.
mapred.lib.KeyFieldBasedComparator',
#             'stream.num.map.output.key.fields': '2',
#             'stream.map.output.field.separator': '\t',
#             'mapreduce.partition.keycomparator.options': '-k2,2nr',
#             'mapreduce.job.reduces': '1'
#         }
        return [
                MRStep(mapper=self.mapper_get_urls,
                       reducer=self.reducer_sum_urls)
#                 MRStep(jobconf=JOBCONF_STEP,
#                        mapper=self.mapper_top_urls,
#                        reducer=self.reducer_top_urls)
                ]

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

```
Overwriting MostFrequentVisits.py
```

In [6]:
```
!chmod a+x MostFrequentVisits.py
```

In [20]:
```
!python MostFrequentVisits.py -r hadoop 'anonymous-msweb-preprocessed.data' >
 mrjob_presort-output
```

```
No configs found; falling back on auto-configuration
Looking for hadoop binary in $PATH...
Found hadoop binary: /usr/bin/hadoop
Using Hadoop version 2.6.0
Looking for Hadoop streaming jar in /home/hadoop/contrib...
Looking for Hadoop streaming jar in /usr/lib/hadoop-mapreduce...
Found Hadoop streaming jar: /usr/lib/hadoop-mapreduce/hadoop-streaming.jar
Creating temp directory /tmp/MostFrequentVisits.root.20170606.032734.270268
Copying local files to hdfs:///user/root/tmp/mrjob/MostFrequentVisits.root.20
170606.032734.270268/files/...
Detected hadoop configuration property names that do not match hadoop version
2.6.0:
The have been translated as follows
 mapred.output.key.comparator.class: mapreduce.job.output.key.comparator.clas
s
mapred.text.key.comparator.options: mapreduce.partition.keycomparator.options
mapred.text.key.partitioner.options: mapreduce.partition.keypartitioner.optio
ns
Running step 1 of 1...
  mapred.text.key.partitioner.options is deprecated. Instead, use mapreduce.p
artition.keypartitioner.options
  packageJobJar: [] [/usr/jars/hadoop-streaming-2.6.0-cdh5.7.0.jar] /tmp/stre
amjob975751589671531124.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Total input paths to process : 1
  number of splits:2
  Submitting tokens for job: job_1496718070925_0006
  Submitted application application_1496718070925_0006
  The url to track the job: http://quickstart.cloudera:8088/proxy/application
_1496718070925_0006/
  Running job: job_1496718070925_0006
  Job job_1496718070925_0006 running in uber mode : false
   map 0% reduce 0%
   map 50% reduce 0%
   map 100% reduce 0%
   map 100% reduce 100%
  Job job_1496718070925_0006 completed successfully
  Output directory: hdfs:///user/root/tmp/mrjob/MostFrequentVisits.root.20170
606.032734.270268/output
Counters: 51
        File Input Format Counters
                Bytes Read=1681214
        File Output Format Counters
                Bytes Written=2903
        File System Counters
                FILE: Number of bytes read=1183854
                FILE: Number of bytes written=2723201
                FILE: Number of large read operations=0
                FILE: Number of read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1681596
                HDFS: Number of bytes written=2903
                HDFS: Number of large read operations=0
                HDFS: Number of read operations=9
                HDFS: Number of write operations=2
        Job Counters
		Data-local map tasks=2
		Launched map tasks=2
		Launched reduce tasks=1
		Total megabyte-seconds taken by all map tasks=24852480
		Total megabyte-seconds taken by all reduce tasks=7814144
		Total time spent by all map tasks (ms)=24270
		Total time spent by all maps in occupied slots (ms)=24270
		Total time spent by all reduce tasks (ms)=7631
		Total time spent by all reduces in occupied slots (ms)=7631
		Total vcore-seconds taken by all map tasks=24270
		Total vcore-seconds taken by all reduce tasks=7631
	Map-Reduce Framework
		CPU time spent (ms)=8480
		Combine input records=0
		Combine output records=0
		Failed Shuffles=0
		GC time elapsed (ms)=187
		Input split bytes=382
		Map input records=98654
		Map output bytes=986540
		Map output materialized bytes=1183860
		Map output records=98654
		Merged Map outputs=2
		Physical memory (bytes) snapshot=793448448
		Reduce input groups=285
		Reduce input records=98654
		Reduce output records=285
		Reduce shuffle bytes=1183860
		Shuffled Maps =2
		Spilled Records=197308
		Total committed heap usage (bytes)=695205888
		Virtual memory (bytes) snapshot=4083478528
	Shuffle Errors
		BAD_ID=0
		CONNECTION=0
		IO_ERROR=0
		WRONG_LENGTH=0
		WRONG_MAP=0
		WRONG_REDUCE=0
	group
		Num_mapper_calls=98654
		Num_reducer_calls=285
Streaming final output from hdfs:///user/root/tmp/mrjob/MostFrequentVisits.ro
ot.20170606.032734.270268/output...
Removing HDFS temp directory hdfs:///user/root/tmp/mrjob/MostFrequentVisits.r
oot.20170606.032734.270268...
Removing temp directory /tmp/MostFrequentVisits.root.20170606.032734.27026
8...
```

In [50]:
```
!sort -nr mrjob_presort-output | head -n 5
```

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

HW4.4 Find the most frequent visitor

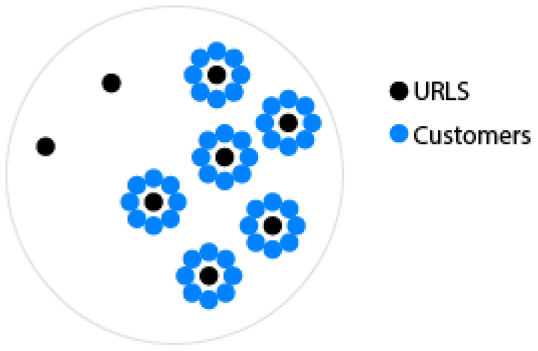
Back to Table of Contents

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. You may get a weird result. HINT: The maximum visits by any visitor to any given webpage is 1.

```
In [2]: from IPython.display import Image, HTML
```

```
In [3]: Image('ms-data.png')
```

Out[3]:



A simplified view of the data

In [59]:
```
%%writefile mostFrequentVisitors.py
#!/usr/bin/env python

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

class MRWordFreqCount(MRJob):

    SORT_VALUES = True

    def mapper_get_urls(self, _, line):
        self.increment_counter('group', 'Num_mapper_calls', 1)
        line = line.split(',')
        yield (line[, 1)

    def reducer_sum_urls(self, url, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield (sum(counts), url)



    def steps(self):
#         JOBCONF_STEP = {
#             'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.
mapred.lib.KeyFieldBasedComparator',
#             'stream.num.map.output.key.fields': '2',
#             'stream.map.output.field.separator': '\t',
#             'mapreduce.partition.keycomparator.options': '-k2,2nr',
#             'mapreduce.job.reduces': '1'
#         }
        return [
                MRStep(mapper=self.mapper_get_urls,
                       reducer=self.reducer_sum_urls)
#                 MRStep(jobconf=JOBCONF_STEP,
#                        mapper=self.mapper_top_urls,
#                        reducer=self.reducer_top_urls)
                ]

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

```
Overwriting mostFrequentVisitors.py
```

In [ ]:
```
!chmod +x mostFrequentVisitors.py
```

In [ ]:

HW4.5 Clustering Tweet Dataset

Back to Table of Contents

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 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 (https://www.dropbox.com/s/6129k2urvbvobkr/topUsers_Apr-Jul_2014_1000-words.txt?dl=0)

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 (generate 1000 random numbers and normalize the vectors)
- (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. Use use the (row-normalized) class-level aggregates as 'trained' starting centroids (i.e., the training is already done for

you!). 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 (https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0)

Row 1: Words Row 2: Aggregated distribution across all classes Row 3-6 class-aggregated distributions for clases 0-3 For (A), we select 4 users randomly from a uniform distribution [1,...,1,000] For (B), (C), and (D) you will have to use data from the auxiliary file:

- topUsers_Apr-Jul_2014_1000-words_summaries.txt (https://www.dropbox.com/s/w4oklbsoqefou3b/topUsers_Apr-Jul_2014_1000-words_summaries.txt?dl=0)

This file contains 5 special word-frequency distributions:

- (1) The 1000-user-wide aggregate, which you will perturb for initializations in parts (B) and (C), and
- (2-5) The 4 class-level aggregates for each of the user-type classes (0/1/2/3)

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

In [ ]:
```
from numpy import random
numbers = random.sample(1000)
```

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

In [ ]:
```
##################################################################################
#####
##Geneate random initial centroids around the global aggregate
##Part (B) and (C) of this question
##################################################################################
#####
def startCentroidsBC(k):
    counter = 0
    for line in open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readlin
es():
        if counter == 2:
            data = re.split(",",line)
            globalAggregate = [float(data[i+3])/float(data[2]) for i in
range(1000)]
        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
range(1000)]
        centroids.append(peturpoints)
        total = 0
        for j in range(len(centroids[i])):
            total += centroids[i][j]
        for j in range(len(centroids[i])):
            centroids[i][j] = centroids[i][j]/total
    return centroids
```

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

K-Means

K-means is a clustering method that aims to find the positions μi,i=1...k of the clusters that minimize the distance from the data points to the cluster. K-means clustering solves:

$$\arg\min_{c} \sum_{i=1}^{k} \sum_{x \in c_i} d(x, \mu_i) = \arg\min_{c} \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|_2^2$$

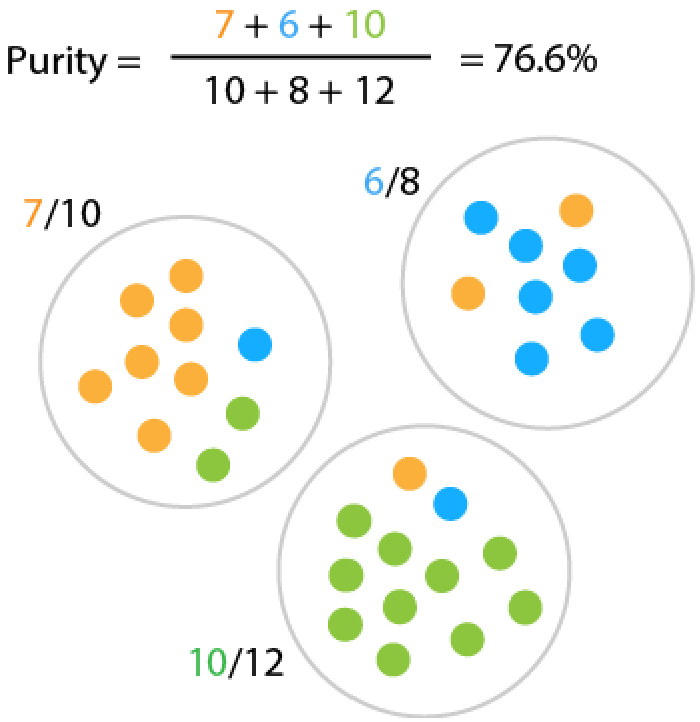
where c_i is the set of points that belong to cluster i. The K-means clustering uses the square of the Euclidean distance $d(x, \mu_i) = \|x - \mu_i\|_2^2$. This problem is not trivial (in fact it is NP-hard), so the K-means algorithm only hopes to find the global minimum, possibly getting stuck in a different solution.

K-means algorithm

The Lloyd's algorithm, mostly known as k-means algorithm, is used to solve the k-means clustering problem and works as follows. First, decide the number of clusters k. Then:

1. Initialize the center of the clusters	$\mu_i =$ some value $, i = 1, \ldots, k$
2. Attribute the closest cluster to each data point	$c_i = \{j : d(x_j, \mu_i) \leq d(x_j, \mu_l), l \neq i, j = 1, \ldots, n\}$
3. Set the position of each cluster to the mean of all data points belonging to that cluster	$\mu_i = \frac{1}{\|c_i\|} \sum_{j \in c_i} x_j, \forall i$
4. Repeat steps 2-3 until convergence	
Notation	$\|c\| =$ number of elements in c

Calculating purity



In [ ]:
```
%%writefile Kmeans.py
#!/usr/bin/env python
#START STUDENT CODE45



#END STUDENT CODE45
```

In [ ]:
```
%%writefile kmeans_runner.py
#!/usr/bin/env python
#START STUDENT CODE45_RUNNER



#END STUDENT CODE45_RUNNER
```

In [ ]:

In [ ]:

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

Back to Table of Contents

Over half a century old and showing no signs of aging, k-means remains one of the most popular data processing algorithms. As is well-known, a proper initialization of k-means is crucial for obtaining a good final solution. The recently proposed k-means++ initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the k-means++ is its inherent sequential nature, which limits its applicability to massive data: one must make k passes over the data to find a good initial set of centers. The paper listed below shows how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing k-means that have mostly focused on the post-initialization phases of k-means. The proposed initialization algorithm k-means|| obtains a nearly optimal solution after a logarithmic number of passes; the paper also shows that in practice a constant number of passes suffices. Experimental evaluation on realworld large-scale data demonstrates that k-means|| outperforms k-means++ in both sequential and parallel settings.

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

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

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

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

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

```
In [ ]:
```

4.6.1 (OPTIONAL) Apply K-MEANS||

Back to Table of Contents

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

```
In [ ]:
```

HW4.7 (OPTIONAL) Canopy Clustering

Back to Table of Contents

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

For more details on the Canopy Clustering algorithm see:

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

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

```
In [ ]:
```

4.7.1 Apply Canopy Clustering based K-MEANS

Back to Table of Contents

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

```
In [ ]:
```

Back to Table of Contents

------- END OF HOWEWORK --------