K-Means Clustering

> This Notebook contains the implementation for data/feature extraction and our own K-Means cluster implementation for a two features. The code is divided among cells to help understand the code easily. Explainination of each cell is at the top of the cell.

NOTE: In order to run this code on a cluster do the following in the termianl first

Environment Setup

Commands to be run in terminal:

Only need to run this command once 'pip install findspark --user'

Set environment variable 'export SPARK_HOME=/usr/lib/spark'

Step1: Import relevent libraries and initialize spark context

In [22]:
```
import findspark
findspark.init()
import pyspark
from pyspark import SparkContext
import xml.etree.ElementTree as ET
SC = pyspark.SparkContext(appName="KMeans Implementation")
```

Step2: Setup input RDD's

- We are using *posts* and *users* files from the stack overflow data.
- Due to large amount of data and cluster taking alot of time to run the job, we took 0.5 of the *posts* data to process.
- This code would work for larger dataset as well but due to load on cluster and time constraint to optimize the code even further, we have used a fraction of the data set for now.
- The commented out lines of code in the following cell would be used to get complete data set.

In [36]:
```
###Full Data Set
#posts = SC.textFile("/data/stackoverflow/Posts")
users = SC.textFile("/data/stackoverflow/Users")

###Fraction of data set with '0.5' meaning 50%
posts = SC.textFile("/data/stackoverflow/Posts").sample(False,0.5,12345)
#users = SC.textFile("/data/stackoverflow/Users").sample(False,0.7,12345)
```

Step3: Data/Feature Extraction

- Following cells contain the functions used to convert the raw data that is in XML format into a format understood by our algorithm.
- The first cell contains the function used for feature extraction.
- The second cell contains the transformations (maps and filters) used to get relevent RDD's for clustering.

In [37]:
```
def getOwnerId(input):
    try:
        tree = ET.fromstring(input)
        if 'OwnerUserId' in tree.attrib:
            a = int(tree.attrib['OwnerUserId'])
            return a
        else:
            return None
    except:
        return None

def getUserId(input):
    try:
        tree = ET.fromstring(input)
        if 'Id' in tree.attrib:
            a = int(tree.attrib['Id'])
            return a
        else:
            return None
    except:
        return None

def getUserReputation(input):
    try:
        tree = ET.fromstring(input)
        if 'Reputation' in tree.attrib:
            a = int(tree.attrib['Reputation'])
            return a
        else:
            return None
    except:
        return None
```

In [38]:
```
###############Input RDD for 2D Clustering of user reputation score against numbe
r of posts they have ####################
##Get number of posts for each users
noOfPostsAgainstUser = posts.map(lambda post: (getOwnerId(post), 1))\
.map(lambda x: (x[0],x[1]))\
.filter(lambda x: x[0] is not None)\
.reduceByKey(lambda x,y : x+y)

##Get reputatition score for each user
repAgainstUser = users.map(lambda post: (getUserId(post),
getUserReputation(post)))\
.map(lambda x: (x[0],x[1]))\
.filter(lambda x: x[0] is not None and x[1] is not None)

##Join posts count with their score for each user
repAgainstPost = repAgainstUser.join(noOfPostsAgainstUser).map(lambda x: x[1])
```

K-Means Algorithm implementation:

We have a class defined as a KMeansModel2D which has the relevent functions to train itself on the data provided to it and other functions e.g a function that can be used to assign cluster to a data point.

In [39]:
```
class KMeansModel2D:
    ##Initialize the model with some cluster centers.
    def __init__(self, centers):
        self.centers = centers

    ##Function used to determine assigned cluster for a data point
    def assignCluster(self,p):
        import math
        bestIndex = -1
        closest = 100000
        for i in range(len(self.centers)):
            tempDist = math.sqrt((self.centers[i][0] - p[0])**2 +
(self.centers[i][1] - p[1])**2 )
            if tempDist < closest:
                closest = tempDist
                bestIndex = i

        return bestIndex
    ##Method to calculate minimum distance of a point to the closest cluster
    def getMinDistance(self, p):
        import math
        closest = 100000

        for i in range(len(self.centers)):
            tempDist = math.sqrt( (self.centers[i][0] - p[0])**2 +
(self.centers[i][1] - p[1])**2 )
            if tempDist < closest:
                closest = tempDist

        return tempDist

    ##Method to train the model with given data
    def TrainModel(self,data):
        ##Print the initially assigned clusters which should be random
        print("Initial centers: " + str(self.centers))
        ##Run the algorithm until cluster movement(summed distance of updated cen
ters and previous ones) in each
        ##iteration is less then our threshold value (convergeDist)
        convergeDist = float(20)
        tempDist = float(100)

        while tempDist > convergeDist:
            closests = data.map(lambda p: (self.assignCluster(list(p)), (p, 1)))
            pointStats = closests.reduceByKey(
            lambda p1_c1, p2_c2: ((p1_c1[0][0] + p2_c2[0][0],p1_c1[0][1] +
p2_c2[0][1]), p1_c1[1] + p2_c2[1]))

            newPoints = pointStats.mapValues(
            lambda st: (st[0][0]/st[1], st[0][1]/st[1])).collect()

            sumDist = 0
            for (iK, p) in newPoints:
                import math
                sumDist = sumDist + math.sqrt((self.centers[iK][0] - p[0])**2 +
(self.centers[iK][1] - p[1])**2 )
                self.centers[iK] = p

            tempDist = sumDist
```

Step4: Instentiate KMeans class and train the model.

- Create KMeans Class with three random data points.(We are clustering it into three clusters)
- Train the model
- Print the cluster centers.

In [ ]:
```
model = KMeansModel2D(repAgainstPost.takeSample(False, 3,1))
model.TrainModel(repAgainstPost)
print("Final centers: " + str(model.centers))
```

Step5: Using our trained model, get the clustered data points and save it in output folder on our cluster

In [ ]:
```
datapoints = repAgainstPost.map(lambda p: (model.assignCluster(p), p))
datapoints.saveAsTextFile('output/data')
```

Step6: Copy the output to local file system

NOTE: Run this command in terminal to save the output as text file

'hadoop fs -getmerge output/data UserRepAgainstPosts.txt'

Final Step: Stop spark context

In [41]:
```
SC.stop()
```