K-Means Clustering

> This Notebook contains the implementation for data/feature extraction and our own K-Means cluster implementation for a single feature. 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

<font color=red>'pip install findspark --user'</font>

Set environment variable

<font color=red>'export SPARK_HOME=/usr/lib/spark'</font>

Step1: Import relevent libraries and initialize spark context

In [3]:

```
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* data from the stack overflow.
- 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 [4]:

```
###Full Data Set
#posts = SC.textFile("/data/stackoverflow/Posts")

###Fraction of data set with '0.5' meaning 50%
posts = SC.textFile("/data/stackoverflow/Posts").sample(False,0.005,12345)
```

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 and third cells contain the transformations (maps and filters) used to get relevent RDD's for clustering.

In [5]:

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

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

import re
def getBodyLength(input):
    try:
        tree = ET.fromstring(input)
        if 'Body' in tree.attrib:
            a = re.sub(r'\<[^>]*\>', '', tree.attrib['Body'])
            return len(a)
        else:
            return None
    except:
        return None
```

In [6]:

```
###########################Input RDD for 1D Clustering of length of accepted anse
wers ###################################
##Get accepted answer postID's
answerIds = posts.map(getAcceptedAnswerIds)\
.filter(lambda f : f is not None)

##Send the list to each node
IdsToRetrieve = SC.broadcast(answerIds.collect())

##Filter out the posts that are accepted answers
acceptedAnswerPosts = posts.filter(lambda f : getPostId(f) in
IdsToRetrieve.value)

##Get the character length of the accepted answer posts
clusterInput = acceptedAnswerPosts.map(getBodyLength)\
.filter(lambda f : f is not None)
```

K-Means Algorithm implementation:

> We have a class defined as a KMeansModel 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 [7]:

```
class KMeansModel:
    ##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):
        bestIndex = 0
        closest = 100000
        for i in range(len(self.centers)):
            tempDist = (p - self.centers[i]) ** 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):
        closest = 100000
        for i in range(len(self.centers)):
            tempDist = (p - self.centers[i]) ** 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(10)
        tempDist = float(100)

        while tempDist > convergeDist:
            assignedPoints = data.map(lambda p: (self.assignCluster(p), (p, 1)))

            pointStats = assignedPoints.reduceByKey(lambda p1_c1, p2_c2:
(p1_c1[0] + p2_c2[0], p1_c1[1] + p2_c2[1]))

            newCenters = pointStats.map(lambda st: (st[0], st[1][0] / st[1]
[1])).collect()

            sumDist = 0
            for (iK, p) in newCenters:
                sumDist = sumDist + ((self.centers[iK] - p) ** 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 = KMeansModel(clusterInput.takeSample(False, 3,1))
model.TrainModel(clusterInput)
print("Final centers: " + str(model.centers))
```

```
Initial centers: [175, 2558, 1131]
```

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

In [20]:

```
datapoints = clusterInput.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

<font color=red>'hadoop fs -getmerge output/data PostLengthData.txt'</font>

Final Step: Stop spark context

In [ ]:

```
SC.stop()
```