Big Data Processing Coursework
LARGE-SCALE CLUSTERING

Group AK

- Ansari, Rayyan Aziz
- Chughtai, Daniyal Asad
- Lyapina, Ekaterina
- Waqar, Muhammad Omar
- Waseef, Ahmed

Instructor: Felix Cuadorado

Introduction

During this coursework, the analysis of Stack Overflow data was created. Stack Overflow is a privately held website, which features user's questions and answers on a wide range of topics in computer programming. This website is as a platform with membership and active participation, upvoting the questions and editing them as well as the answers. The dataset of a Stack Overflow is a set of files containing posts, users, votes, comments, post history and post links.

For our coursework the following use cases was defined, which might have potential implementation in the industry.

1. An accumulates the distribution of users into junior, senior and Ninja developer by using chosen clustering technique. Clustering is performed using Post data (number of post) against Users Reputation data to find out how active and popular is the user. This shows how cluster analysis can be applied to market research for partitioning a population of users into groups. This can be useful, for user segmentation and targeting marketing campaigns.
2. Another task is performing the association of the posts answer lengths into short, medium and long cluster. This explain how cluster analysis can be applied to classify topics into level of difficulty, as certain topics requires more explanation than others.

This project creates several Spark programs to perform multiple types of computation. Moreover, project uses another approach for k-means clustering algorithm using the external library for comparison between own results and that generated by the pre-installed library.

Theoretical introduction and literature review

Clustering is one of the main data mining tasks. It is a grouping of a particular set of objects based on their characteristics, aggregating them according to their high or low inter-cluster similarities. The main distinction of the clustering task from the classification task is that the number of clusters is not known in advance. Clustering allows data points to either be a part of one cluster or allow data points to belong to the different clusters with certain degree. The goal of clustering is descriptive, as the new clusters are discovered which were not seen from raw data.

Clustering has a large number of applications spread at across various domains. Some of the most popular applications of clustering are: recommendation engines, market segmentation, social network analysis, search result grouping, medical imaging, image segmentation, anomaly detection. Clustering can also serve as a useful data-preprocessing step to identify homogeneous groups on which to build supervised models.

Clustering techniques have some requirements:

- Scalability (in terms of both time and space).
- Ability to deal with different data types.

- Minimal requirements for domain knowledge to determine input parameters.
- Able to deal with noise and outliers.
- Insensitive to order of input records.
- Incorporation of user-specified constraints.
- Interpretability and usability.

The solution of the clustering problem is fundamentally ambiguous:

- To exact formulation of the clustering problem.
- Many criteria for the quality of clustering.
- Many heuristic methods of clustering.
- Number of clusters usually is not known in advance.
- Result of clustering is subjective.

There are several different approaches to the computation of clusters.

Hierarchical clustering

Hierarchical clustering begins from building a distance matrix which contains the distances between every pair of objects based on Euclidean distance, cosine similarity or Pearson correlation coefficient. There are two types of hierarchical clustering:

1. **Bottom-Up (agglomerative):** Starting with each item in its own cluster, the best pair to merge into a new cluster is looking for. The process is repeated until all clusters are fused together. In this technique the data points are often joined in and they form groups.
2. **Top-Down (divisive):** Starting with all the data in a single cluster, every possible way to divide the cluster into two is considered. The the best division is chosen and recursively operated on both sides. Hence more groups in this top-down category is seen.

Summary of hierarchical clustering methods:

- Robust against noise.
- No need to specify the number of clusters in advance.
- Hierarchical structure maps nicely onto human intuition for some domains.
- They do not scale well: time complexity of at least O(n^2), where n is the number of total objects.
- Assumptions requires a similarity / distance measure.
- Clusters are subjective (only a tree is returned).
- Local optima are a problem. Merges are final and cannot be undone at a later time, preventing global optimisation and causing trouble for noisy, high-dimensional data.
- Interpretation of results is subjective.
- Useful if the underlying application has a taxonomy.

Self Organising Maps

This clustering method based on a similarity measure related to the calculation of Euclidean distances. The idea of this principle is to find a winner-takes-all neuron to find the most closely matching case. The SOM was proposed on the idea that systems can be design to

emulate the collective co-operation of the neurons. Collectivism can be realised by feedback and thus can also be realised in the network, where many neighbouring neurons react collectively upon being activated by events. The SOM is an established paradigm in AI and cognitive modelling being the basis of unsupervised learning. Despite its excellent performance, there are major issues related to the methods of identifying the best matching unit and its slow processing time. The original SOM's winner-takes-all competitive learning algorithm often performs poorly in differentiating sequential or temporal patterns. Original SOM is unable to provide system user with visual information.

Advantages

- Data mapping is easily interpreted and understand by human. The reduction of dimensionality and grid clustering makes it easy to observe similarities in the data. SOMs factor in all the data in the input to generate these clusters and can be altered such that certain pieces of data have more/less of an effect on where an input is placed.
- The algorithm is simple enough to make the training process easy to understand and alter as needed.
- They classify data well and then are easily evaluate for their own quality so you can actually calculated how good a map is and how strong the similarities between objects are.
- Some reports also mention that SOM is fully capable of clustering large, complex data sets. With a few optimisation techniques, a SOM can be trained in a short amount of time.

Disadvantages

- It requires necessary and sufficient data in order to develop meaningful clusters. Difficult to determine what input weights to use. Getting the right data is also difficult. Unfortunately, there is a need for a value for each dimension of each member of samples in order to generate a map. Sometimes this simply is not possible and often it is very difficult to acquire all of this data so this is a limiting feature to the use of SOMs often referred to as missing data.
- Relies on a predefined distance in feature space. Requires that nearby points behave similarly. Every SOM is different and finds different similarites among the sample vectors. SOMs organise sample data so that in the final product, the samples are usually surrounded by similar samples, however similar samples are not always near each other. So a lot of maps need to be constructed in order to get one final good map.
- Very computationally expensive.
- Another problem with SOMs is that it is often difficult to obtain a perfect mapping where groupings are unique within the map. Instead, anomalies in the map often generate where two similar groupings appear in different areas on the same map. Clusters will often get divided into smaller clusters, creating several areas of similar neurons. This can be prevented by initialising the map well, but that is not an option if the state of the final map is not obvious.
- magnification factors not well understood (at least to my best knowledge).
- 1D topological ordering property does not extend to 2D.

- slow training, hard to train against slowly evolving data.

Expectation-Maximisation (EM) algorithm

This algorithm is a way to find maximum-likelihood estimates for model parameters when the data set is incomplete. This algorithm is based on finding the certain parameters that would define final clusters. This method does not assign the cluster directly, is works with the probability of a point to belong to a certain cluster. Beside working with the missing data, this algorithm is useful when the likelihood function has a form that does not allow convenient analytical methods of investigation, but allows serious simplification if additional unobserved latent variables are introduced into the model. Examples of applied problems for the second use case are the problems of image recognition, image reconstruction.

Advantages

- Good for fitting mixture distributions.
- Based on statistics.
- Linear complexity increase with the enhancement of the data set.
- Tolerant to the noise and missing data.
- Ability to built the desired amount of clusters.
- Quick convergence in case of successful initialisation.

Disadvantages

- The assumption of the normality of all data measurements is not always satisfied.
- If the initialisation fails, the convergence of the algorithm may turn out to be slow. The algorithm can stop at a local minimum and give a quasi-optimal solution. It sometimes needs a few random starts to find the best model because the algorithm can hone in on some local maxima that isn't that close to the (optimal) global maxima. In other words, it can perform better if you force it to restart and take that "initial guess" from Step 1 over again. From all the possible parameters, you can then choose the one with the greatest maximum likelihood.
- It does not produce standard errors as a by-product.
- It works best when you only have a small percentage of missing data and the dimensionality of the data isn't too big. The higher the dimensionality, the slower the E-step; for data with larger dimensionality.

K-means

K-means is one of the most popular clustering techniques. The main concept behind k-means clustering is to find k-centres and one centre is defined for each cluster. Clusters are then grouped to their centers according to their similar features. The advantage of k-means clustering is that it allows the user to interpret and analyse the clusters after the grouping of the data according to the similar features and extract whatever information it needs on it.

Algorithm

1. Decide on a value for k, the number of clusters.

2. Initialise the k-cluster centers (randomly, if necessary).
3. Decide the class memberships of the N objects by assigning them to the nearest cluster center.
4. Re-estimate the k-cluster centers, by assuming the memberships found above are correct.
5. Repeat 3 and 4 until none of the N objects changed membership in the last iteration.

Advantages

- Fastest (each iteration is linear).
- Simple, easy to implement and debug.
- Intuitive objective function: optimises intra-cluster similarity.

Disadvantages

- Sensitive to the initial appropriation (often terminates at a local optimum) and outliers.
- Unable to handle noisy and categorical data (when it is difficult if not event impossible to define a mean).
- Need to the number of clusters in advance.

How to work with disadvantages

1. Make several random clusterisations.
2. Choose the best according to the functionality of the quality smoothly add the number of clusters k.
3. Assign members based on current centers.
4. Re-estimate centers based on current assignment.

Difference between k-means and other clustering techniques.

- Hierarchical clustering cannot handle data in large quantity as well as k-means clustering. This is because the time complexity of k-means is linear while that of hierarchical clustering is quadratic.
- In k-means clustering, since we start with random choice of clusters, the results produced by running the algorithm multiple times might differ. While results are reproducible in hierarchical clustering.
- K-means is found to work well when the shape of the clusters is hyper spherical.
- K-means clustering requires prior knowledge of k.

Taking into the account properties of all algorithms and stater business objectives, k-means algorithm was chosen as the most simple and universal was to solve the stated teasks.

Project description

Methodology

To run Jupyter Notebook the following commands should be run on the terminal:
$ pip install findspark – user (Only need to run this command the first time)
$ export SPARK_HOME=/usr/lib/spark
$ jupyter notebook

Implementation of the project

We have used Jupyter Notebook to implement our project and the notebook file is shared with this report. The code along with the explanation can be found in the appendix for a better understanding. While using the provided environment, several problems with “numpy” library had appeared so custom implementations of a needed functions were made. This has restricted the total implementation to be generalised enough to use any number of features to train the model. Therefore there is a need to make a few changes in the code according to the number of features being used. The total project consists of two separate notebooks for single feature and 2 features implementation of the same algorithm:

Project uses k-means clustering algorithm to find out underlying patterns. The current report contains code with the comments, the results alongside the explanation and visual representation of different types of computations.

- 2D_array.ipynb - visialisation of the second task
- Answered posts length K-Means Clustering .ipynb - clustering of the 1 task
- Libraryimp.docx - implementation of the standard library
- User reputation against posts K-Means Clustering .ipynb - implementation of the 2 task

1. Getting data

First step is to load the relevant data files into RDD's. The whole data set was decided not to be used to avoid the cluster overload and to provide the ability to evaluate faster the code logic work via analysing the preliminary results. The method of subsetting the data has been explained in the code attached in appendix.

2. Data/Feature Extraction Section

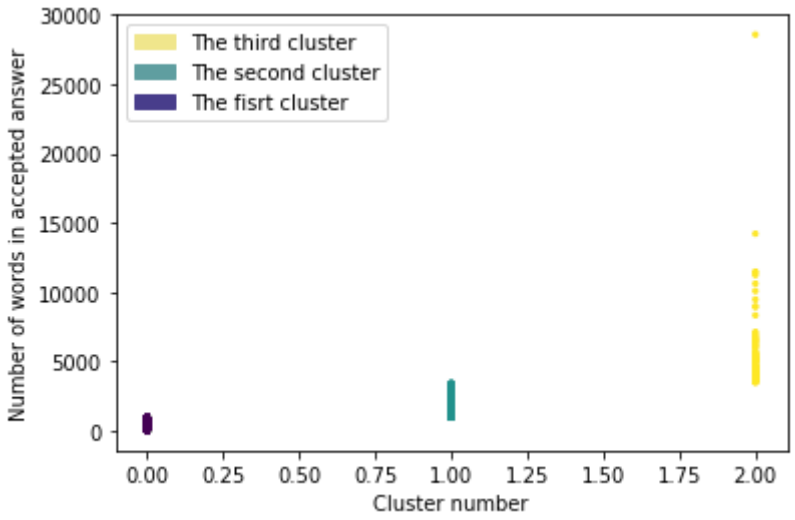
The second step is to read an XML data file, parse it and extract the relevant fields. For this purpose the package named 'xml.etree.ElementTree' was used. It parses a single line of xml input entry from which relevant attributes/fields can be accessed. The specific functions that parse and then return the value of a specific field was created. In case the parsing fails or the field is not found in any of the input, it returns 'None'. This entries were excluded from an RDDs before processing.

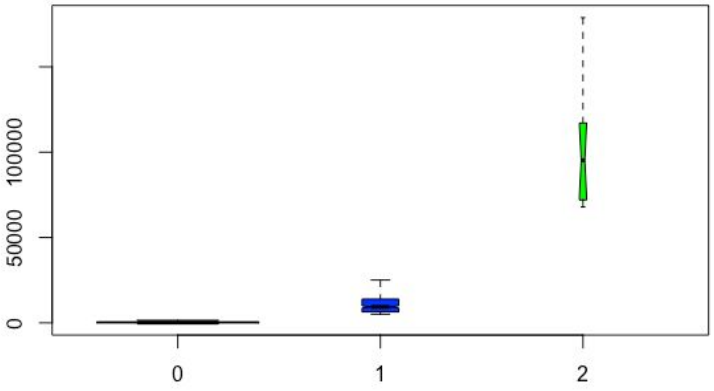
3. Print and save output

Print the cluster centres and save each data point with its assigned cluster as a text file to be visualised.

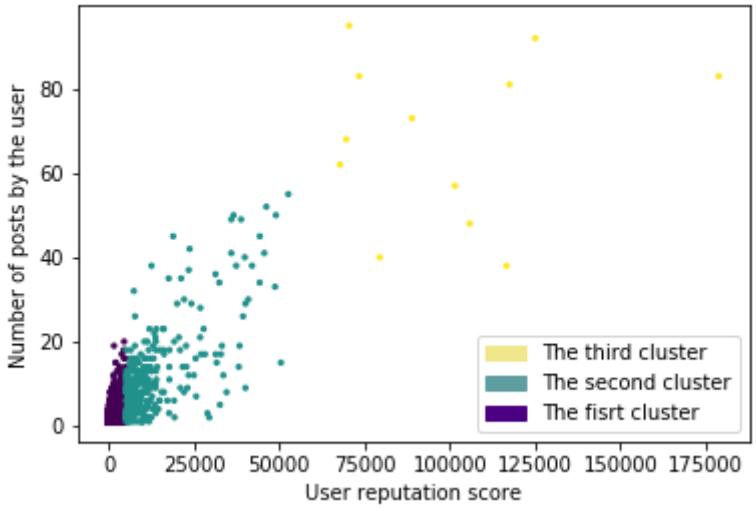
4. Results

The result of the first task can be seen on the following two graphs. The outlier in the third cluster might be observed.





The second graph represents the clusterisation for the second task:



References

1. Abbas, “Comparison between Data Clustering Algorithms”, Yormouk University, 2007.
2. Bijuraj, “Clustering and its Applications”, 2013.
3. M A Deshmukh, R A Gulhane, “Importance of Clustering in Data Mining”, 2016.
4. https://en.wikipedia.org/wiki/Cluster_analysis
5. https://www.analyticsvidhya.com/blog/2017/02/test-data-scientist-clustering/

Appendixes

1. Answered posts length 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. Explanation 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 terminal first. Environment Setup, commands to be run in the terminal:
Only need to run this command once
'pip install findspark --user'
Set environment variable
'export SPARK_HOME=/usr/lib/spark'

Step 1: Import relevant libraries and initialise Spark context.

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

Step 2: Setup input RDD's.

- Posts data from the Stack Overflow is used.
- Due to large amount of data and cluster taking a lot of time to run the job, 0.5% of the data posts was processed.
- This code would work for larger dataset as well.
- The commented out lines of code in the following cell would be used to get complete data set.

```
#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)
```

Step 3: Data/feature extraction.

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

```
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

#Input RDD for 1D Clustering of length of accepted answers
#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 relevant 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.

```
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 centers and previous
ones) in each
    #Iteration is less than 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
```

Step 4: Instentiate K-means class and train the model.

- Create K-means class with three random data points. (the data is clustered into three clusters).
- Train the model.
- Print the cluster centers.

```
model = KMeansModel(clusterInput.takeSample(False, 3,1))
model.TrainModel(clusterInput)
print("Final centers: " + str(model.centers))
```

Initial centers: [175, 2558, 1131]

Step 5: Using trained model, get the clustered data points and save it in output folder on our cluster.

```
datapoints = clusterInput.map(lambda p: (model.assignCluster(p), p))
datapoints.saveAsTextFile('output/data')
```

Step 6: 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 PostLengthData.txt'

Final Step: Stop Spark context.

```
SC.stop()
```

2. User reputation against posts 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. Explanation 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 terminal first. Environment Setup, commands to be run in the terminal:
Only need to run this command once
'pip install findspark --user'
Set environment variable
'export SPARK_HOME=/usr/lib/spark'

Step 1: Import relevant libraries and initialise Spark context.

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

Step 2: 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 a lot 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.

```
#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)
```

Step 3: 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.

```
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

#Input RDD for 2D Clustering of user reputation score against number 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 reputation 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 relevant 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.

```
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 centers and
previous ones) in each
    #Iteration is less than 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
```

Step 4: Instentiate K-means class and train the model.

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

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

Step 5: Using trained model, get the clustered data points and save it in output folder on the cluster.

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

Step 6: 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.

```
SC.stop()
```

3. Implementation of a standard library

```
from sklearn.cluster import KMeans
import numpy as np
from numpy import genfromtxt
X= genfromtxt('C:\\Users\\Rayyan\\Downloads\\plotthis222.csv', delimiter=',')
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
kmeans.cluster_centers_
```

We are using Scikit-learn to verify our results. The “genfromtxt” command generates a Numpy array “X” from the input data that we read. We then use the .fit method of K-means to compute K-means clustering. Finally we use the .cluster_centers_ method to return an array of cluster centers.

4. Visualisation of the results

```
from sklearn.cluster import KMeans
import numpy as np
from numpy import genfromtxt
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

X= genfromtxt('/home/qmul/plotthis.csv', delimiter=',')
type(X)

kmeans = KMeans(n_clusters=3, random_state=0).fit(X.reshape(-1,1))
kmeans.cluster_centers_

combined = np.vstack((X, kmeans.labels_)).T
print(combined)

plt.scatter(combined[:,1], combined[:,0], c=combined[:,1], s=5)

plt.ylabel('Number of words in accepted answer')
plt.xlabel('Cluster number')

yellow_patch = mpatches.Patch(color='khaki', label='The third cluster')
green_patch = mpatches.Patch(color='cadetblue', label='The second cluster')
purple_patch = mpatches.Patch(color='darkslateblue', label='The fisrt cluster')

plt.legend(handles=[yellow_patch, green_patch, purple_patch])

plt.show()
```

5. Visualisation of a boxplot

```
D <- read.table("2D.csv", header=TRUE, sep=",", stringsAsFactors = FALSE)
boxplot(D[,2] ~ D[,1], notch=T, col=(c("RED", "BLUE", "GREEN")), outline=F, varwidth=T)
```