

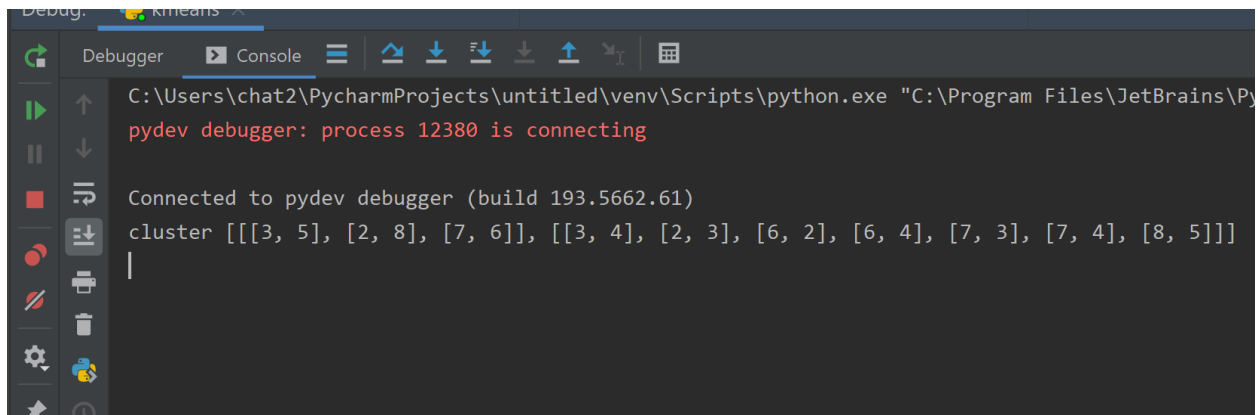
K-Means Clustering

MACHINE LEARNING ASSIGNMENT -06

BHUVANESH JEEVARATHINAM

1. Suppose we have 10 college football teams X1 to X10. We want to cluster them into 2 groups. For each football team, we have two features: One is # wins in Season 2016, and the other is # wins in Season 2017

(1) Initialize with two centroids, (4, 6) and (5, 4). Use Manhattan distance as the distance metric. Please use K-Means to find two clusters.



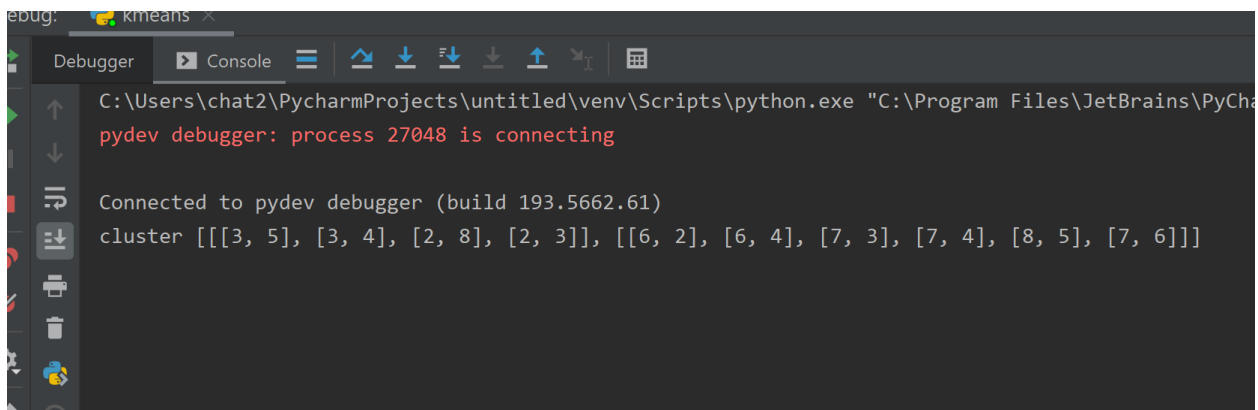
```
Debug: kmeans
Debugger Console
C:\Users\chat2\PycharmProjects\untitled\venv\Scripts\python.exe "C:\Program Files\JetBrains\Py
pydev debugger: process 12380 is connecting

Connected to pydev debugger (build 193.5662.61)
cluster [[3, 5], [2, 8], [7, 6]], [[3, 4], [2, 3], [6, 2], [6, 4], [7, 3], [7, 4], [8, 5]]
|
```

Cluster 1: [[3, 5], [2, 8], [7, 6]]

Cluster 2: [[3, 4], [2, 3], [6, 2], [6, 4], [7, 3], [7, 4], [8, 5]]

(2) Initialize with two centroids, (4, 6) and (5, 4). Use Euclidean distance as the distance metric. Please use K-Means to find two clusters.



```
debug: kmeans
Debugger Console
C:\Users\chat2\PycharmProjects\untitled\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCh
pydev debugger: process 27048 is connecting

Connected to pydev debugger (build 193.5662.61)
cluster [[[3, 5], [3, 4], [2, 8], [2, 3]], [[6, 2], [6, 4], [7, 3], [7, 4], [8, 5], [7, 6]]]
```

Cluster 1: [[3, 5], [3, 4], [2, 8], [2, 3]]

Cluster 2: [[6, 2], [6, 4], [7, 3], [7, 4], [8, 5], [7, 6]]

(3) Initialize with two centroids, (3, 3) and (8, 3). Use Manhattan distance as the distance metric. Please use K-Means to find two clusters.

[illegible]

Cluster 1: $[[3, 5], [3, 4], [2, 8], [2, 3]]$

Cluster 2: [[6, 2], [6, 4], [7, 3], [7, 4], [8, 5], [7, 6]]

(4) Initialize with two centroids, (3, 2) and (4, 8). Use Manhattan distance as the distance metric. Please use K-Means to find two clusters.

The screenshot shows the PyCharm interface with the 'Debug' tab selected. The console output indicates a successful connection to a pydev debugger running on a remote machine.

```

C:\Users\chat2\PycharmProjects\untitled\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm 2019.3.1\plugins\pydev\pydevd.py" --no-redirect C:\Users\chat2\PycharmProjects\untitled\venv\Scripts\python.exe
pydev debugger: process 2944 is connecting

Connected to pydev debugger (build 193.5662.61)
cluster [[[3, 5], [3, 4], [2, 3], [6, 2], [6, 4], [7, 3], [7, 4]], [[2, 8], [8, 5], [7, 6]]]

```

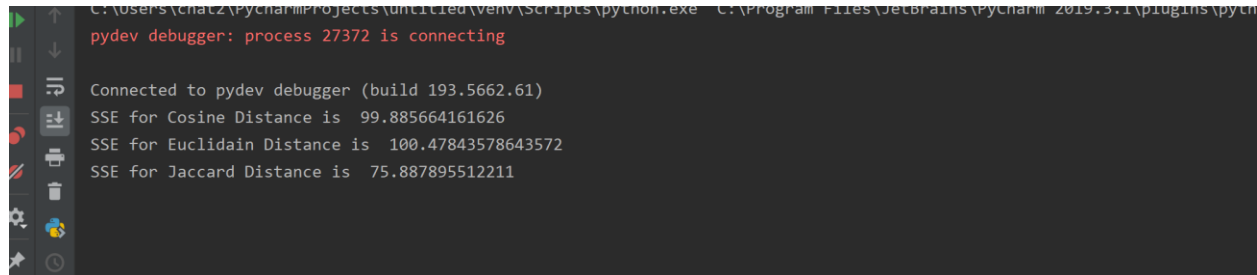
Cluster 1: [3, 5], [3, 4], [2, 3], [6, 2], [6, 4], [7, 3], [7, 4]]

Cluster 2: $[[2, 8], [8, 5], [7, 6]]$

Q1: Run K-means clustering with Euclidean, Cosine and Jaccard similarity. Specify K= the number of categorical values of y (the variable of label). Compare the SSEs of Euclidean-K means Cosine-K-means, Jaccard-K-means. Which method is better and why?

Solution:

K= 3 (No of Categories)



```
C:\Users\chatz\PycharmProjects\untitled\venv\Scripts\python.exe C:\Program Files\JetBrains\PyCharm 2019.3.1\plugins\python\pydev\pydev debugger: process 27372 is connecting

Connected to pydev debugger (build 193.5662.61)
SSE for Cosine Distance is 99.885664161626
SSE for Euclidean Distance is 100.47843578643572
SSE for Jaccard Distance is 75.887895512211
```

SSE for Cosine- 99.885664161626

SSE for Euclidean – 100.47843578643572

SSE for Jaccard - 75.887895512211

SSE - (Euclidean > Cosine > Jaccard)

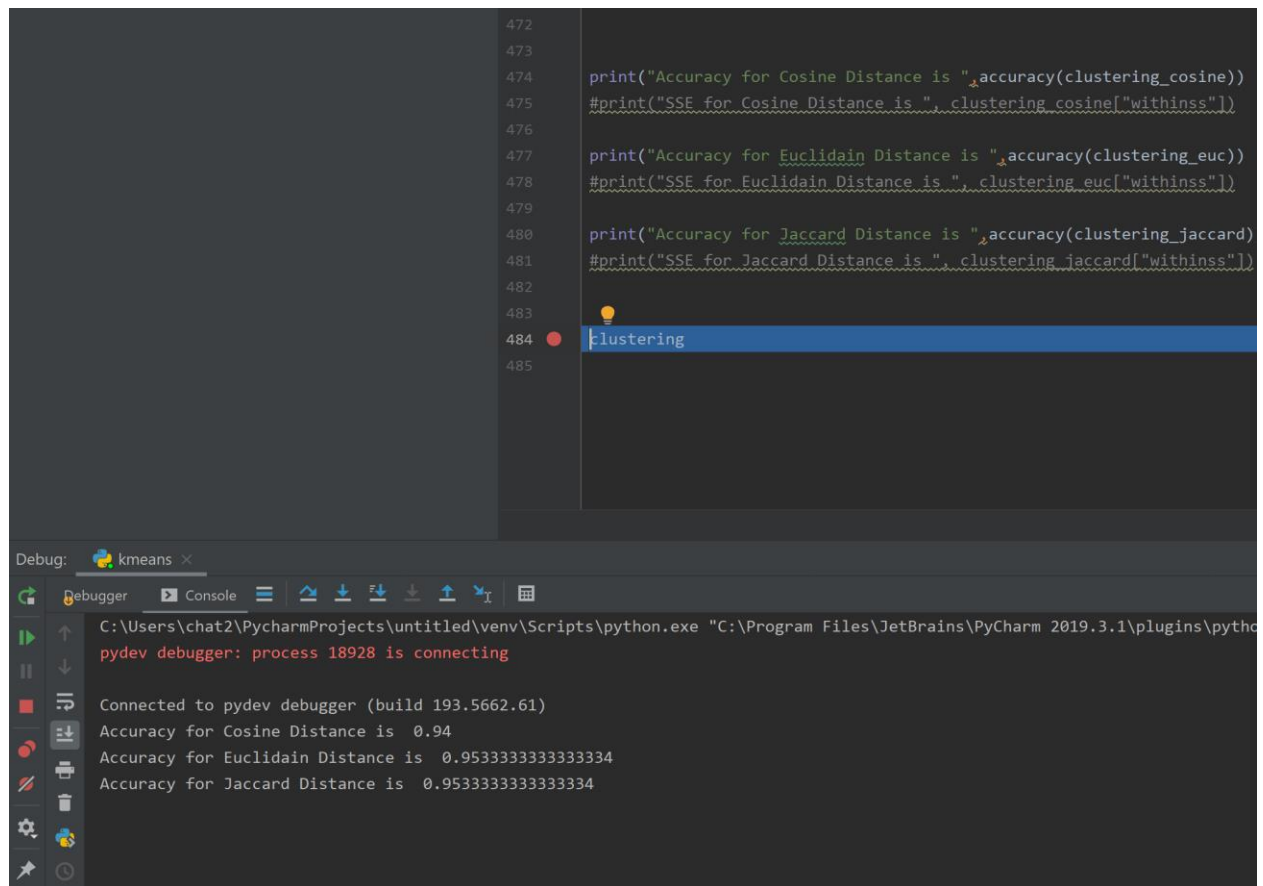
SSE for Jaccard is lowest compared to Euclidean or Cosine. SSE in K means is basically the distance between the centroid and instances. We can say that when the distance is less between them the clusters are more densely populated. Here we can see that clusters measured through Jaccard method seems to be denser and more grouped compared to rest two. Hence Jaccard method is best. Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets

Q2: Compare the accuracies of Euclidean-K-means Cosine-K-means, Jaccard-K-means. First, label each cluster with the label of the highest votes. Later, compute the accuracy of the K means with respect to the three-similarity metrics. Which metric is better and why?

Accuracy Cosine – 0.94

Accuracy Euclidean – 0.9533

Accuracy Jaccard – 0.96



```
472
473
474 print("Accuracy for Cosine Distance is ", accuracy(clustering_cosine))
475 #print("SSE for Cosine Distance is ", clustering_cosine["withinss"])
476
477 print("Accuracy for Euclidean Distance is ", accuracy(clustering_euc))
478 #print("SSE for Euclidean Distance is ", clustering_euc["withinss"])
479
480 print("Accuracy for Jaccard Distance is ", accuracy(clustering_jaccard))
481 #print("SSE for Jaccard Distance is ", clustering_jaccard["withinss"])
482
483
484 clustering
485
```

Debug: kmeans x

Debugger Console

C:\Users\chat2\PycharmProjects\untitled\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm 2019.3.1\plugins\pytho
pydev debugger: process 18928 is connecting

Connected to pydev debugger (build 193.5662.61)

Accuracy for Cosine Distance is 0.94
Accuracy for Euclidean Distance is 0.9533333333333334
Accuracy for Jaccard Distance is 0.9533333333333334

SSE - (Euclidean < Cosine < Jaccard)

Accuracy for Jaccard is comparatively Higher than Cosine and Euclidean. Jaccard uses similarity between instances and Centroids, Hence the groups formed by Jaccard is comparatively denser and far from rest of the categories Hence it is more accurate compared to other. When SSE is lesser the Accuracy is also higher. This means the distance between the centroid and the instances is less which signifies that the clusters are more denser and have more distance between each clusters.

Q3: Which of Euclidean-K-means, Cosine-K-means, Jaccard-K-means requires more iterations and times and why?

Both Cosine and Euclidean takes 8 iterations whereas Jaccard takes 4 iterations. This happen because Jaccard can easily group since the metrics is based on similarity of data. Whereas Euclidean used Euclidean distance and hence it requires lot of time and iterations for grouping. Moreover, Jaccard is itself calculated by comparing the similarity which checks for grouping in the very first place. Hence the clusters are formed easily without the necessity for further iterations.

Q4: Compare the SSEs of Euclidean-K-means Cosine-K-means, Jarcard-K-means with respect to the following three terminating conditions:

- **when there is no change in centroid position**
- **when the SSE value increases in the next iteration**
- **when the maximum preset value (100) of iteration is complete**

When the SSE value increases in the next Iteration. It means that the clusters is not grouping but it is increasing in size and more widespread. In this case it is difficult for the clusters to be formed. So, Increasing SSE requires more number of Time and Iterations compared to other.

Q5: Understanding K-Means:

- *List the general idea of K-means clustering algorithm?*

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The '*means*' in the K-means refers to averaging of the data; that is, finding the centroid.

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

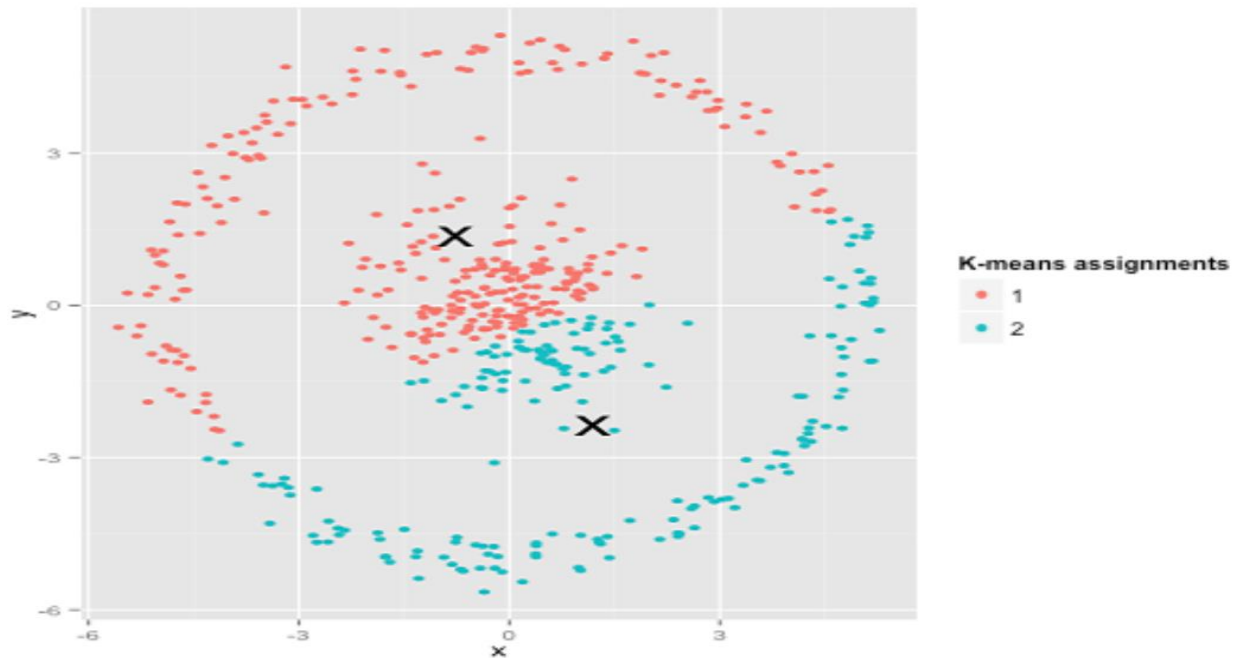
2.Please give a scenario in which K-means cluster may not work very well?

K-Means Clustering

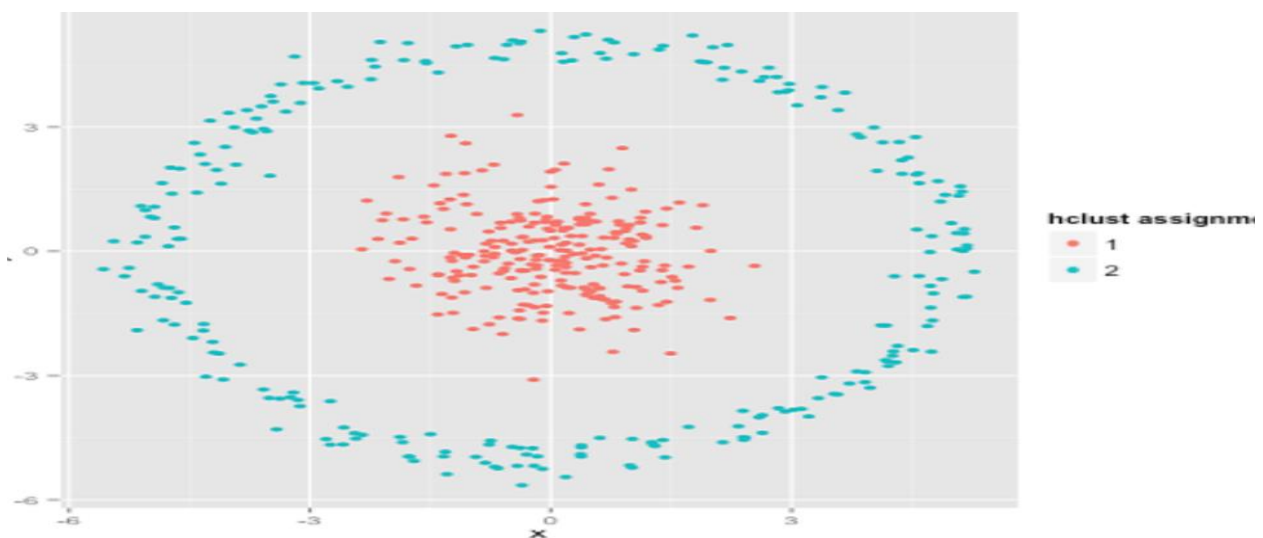
Kmeans does not perform well for *Spherical* Data.

Example Case:

The below is wrong assumption of K means for clustering.



Hierarchical Clustering provides a good assumption and efficiency



3. What is the advantage of K-Means? What are the disadvantages of K-Means?

ADVANTAGES:

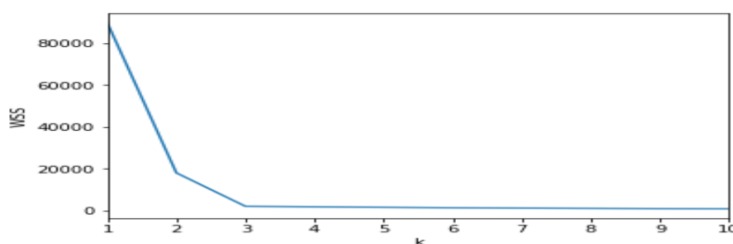
- If the variables in the dataset are huge, then k-means is computationally faster than that of hierarchical clustering when the value of K is kept small.
- The clusters produced in K-means contains the data points closer when compared to that of the hierarchical clustering.
- It is easy to implement for large data sets and adapts to new training of examples.

DISADVANTAGES:

- One of the major drawback of k-means is the inability of the algorithm to take the initialize K by itself and it creates a dependency to obtain the initial values.
- The K-means algorithms does not work well with the globular clusters.
- Centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. Consider removing the outliers before clustering.

4. The classic K-means algorithm randomly initializes K centers. Is there any better strategy for selecting K initial centers?

- **The Elbow Method**



We plot the graph between within Sum of Squares of cluster for various K. We select a point at which a perfect L paves way. This point can be considered as perfect K.

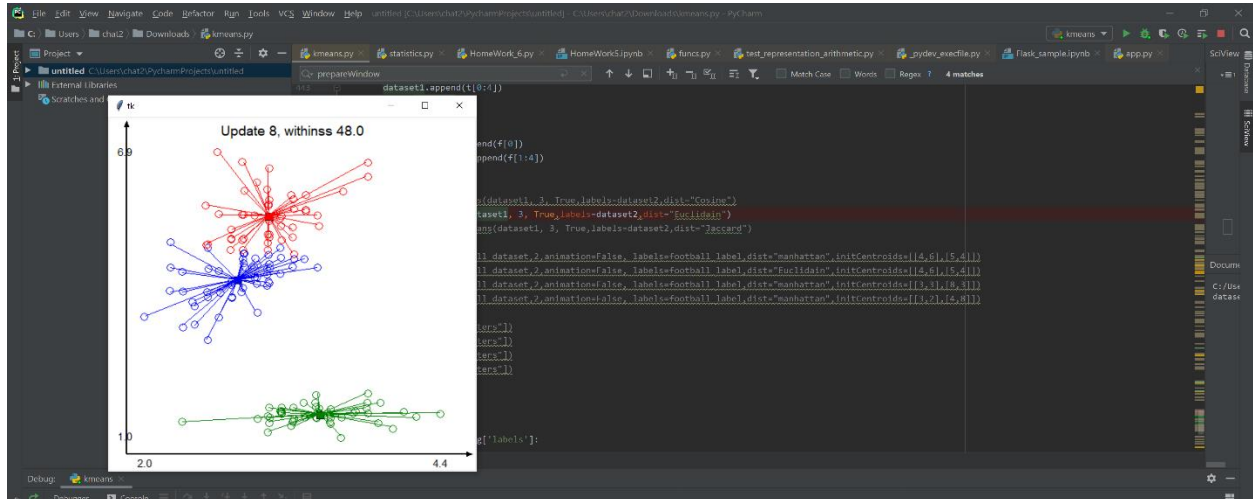
- **The Silhouette Method**

The silhouette value measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation).

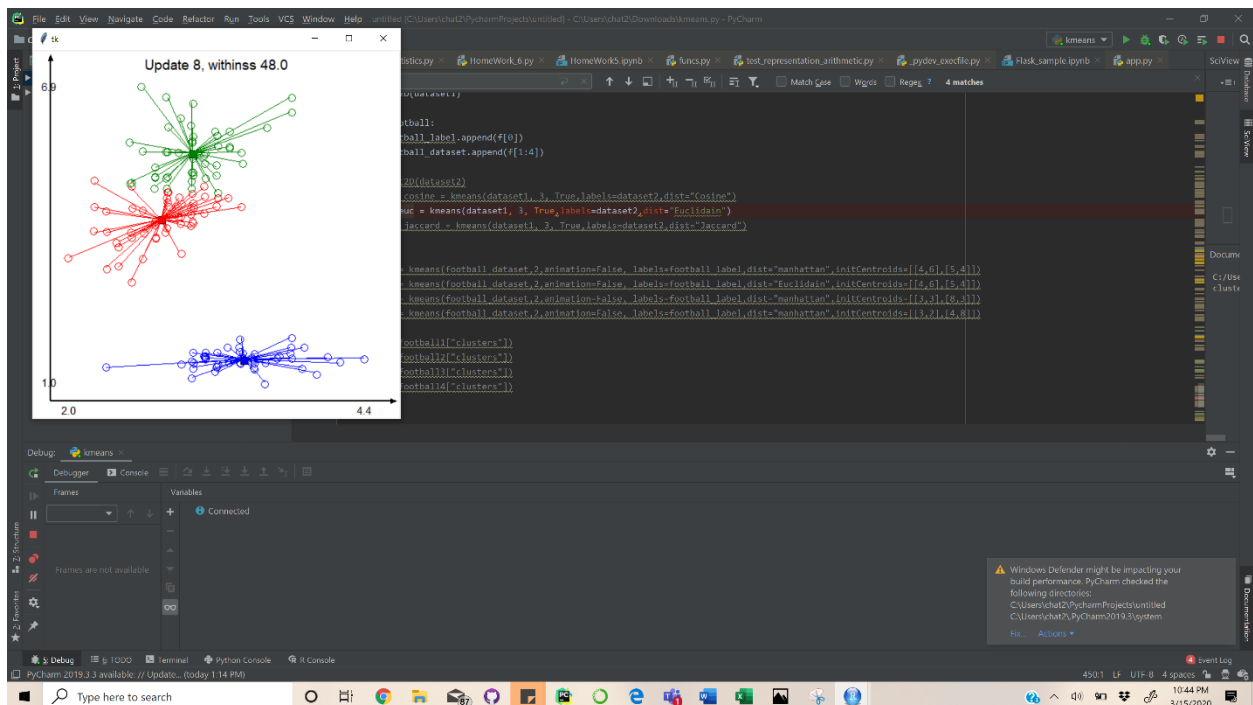
APPENDIX

Screenshots of clustering of Only Final Iterations:

Cosine:

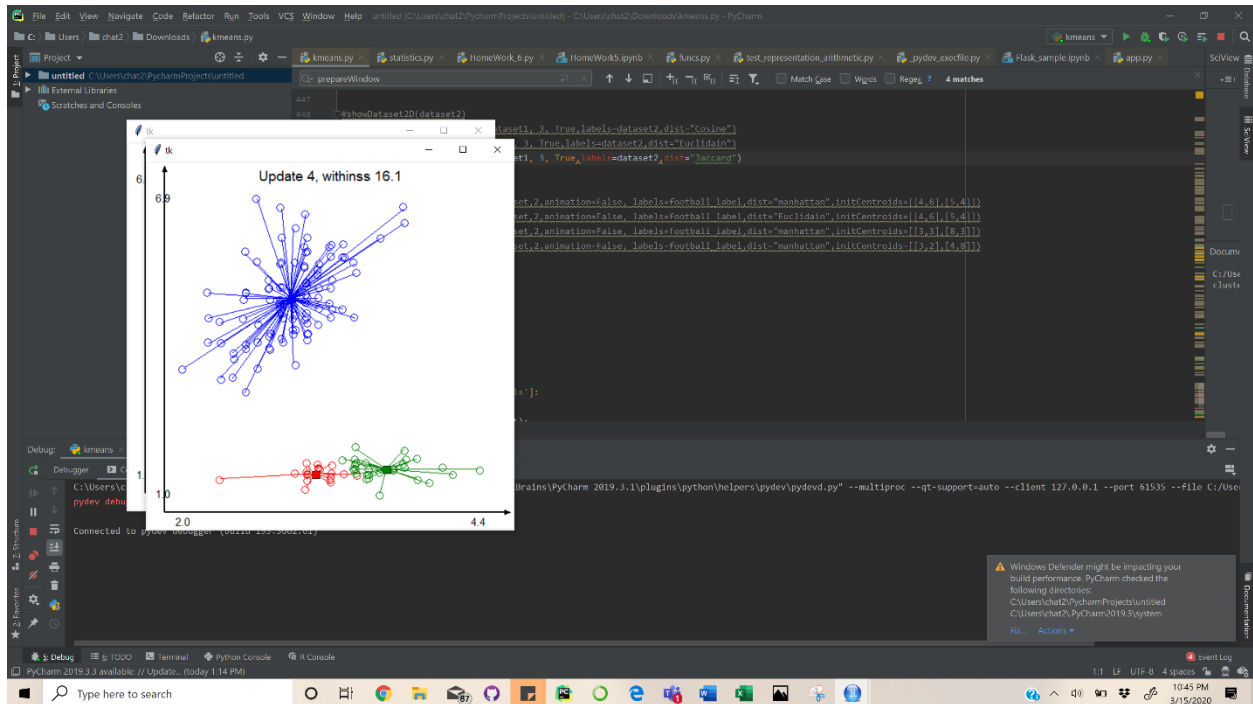


Euclidean:



K-Means Clustering

Jaccard:



Code:

Code Link:

<https://github.com/bhuvaneshkj/CAP5610-MachineLearning-Assignment>

IDE used: Pycharm

```
import math
import random
import time
from scipy import spatial
from tkinter import *
import numpy as np
from math import sqrt
from sklearn.metrics import jaccard_score
import statistics as stats
from statistics import mode
#####
# This section contains functions for loading CSV (comma separated values)
# files and convert them to a dataset of instances.
```

```

# Each instance is a tuple of attributes. The entire dataset is a list
# of tuples.
#####

# Loads a CSV files into a list of tuples.
# Ignores the first row of the file (header).
# Numeric attributes are converted to floats, nominal attributes
# are represented with strings.
# Parameters:
#   fileName: name of the CSV file to be read
# Returns: a list of tuples
def loadCSV(fileName):
    fileHandler = open(fileName, "rt")
    lines = fileHandler.readlines()
    fileHandler.close()
    del lines[0] # remove the header
    dataset = []
    # y=[]
    for line in lines:
        instance = lineToTuple(line)
        dataset.append(instance)

    # for i in range(0,len(dataset)):
    #     y.append(dataset[i][4])
    #     del dataset[i][4]
    # y
    return dataset

# Converts a comma separated string into a tuple
# Parameters
#   line: a string
# Returns: a tuple
def lineToTuple(line):
    # remove leading/trailing witespace and newlines
    cleanLine = line.strip()
    # get rid of quotes
    cleanLine = cleanLine.replace('"', '')
    # separate the fields
    lineList = cleanLine.split(",")
    # convert strings into numbers
    stringsToNumbers(lineList)
    lineTuple = tuple(lineList)
    return lineTuple

# Destructively converts all the string elements representing numbers
# to floating point numbers.
# Parameters:
#   myList: a list of strings
# Returns None
def stringsToNumbers(myList):
    for i in range(len(myList)):
        if (isValidNumberString(myList[i])):
            myList[i] = float(myList[i])

# Checks if a given string can be safely converted into a positive float.

```

```

# Parameters:
#   s: the string to be checked
# Returns: True if the string represents a positive float, False otherwise
def isValidNumberString(s):
    if len(s) == 0:
        return False
    if len(s) > 1 and s[0] == "-":
        s = s[1:]
    for c in s:
        if c not in "0123456789.":
            return False
    return True

#####
# This section contains functions for clustering a dataset
# using the k-means algorithm.
#####

def dp(x, y):
    return sum(a * b for a, b in zip(x, y))

def mg(v):
    return sqrt(dp(v, v))

def manhattan(a, b):
    return abs(a-b)

def manhatten(x, y):
    result = []
    #if len(x) == len(y) + 1:
        #x = x[1:]
    for i in range(1, len(x)):
        result.append(abs(x[i] - y[i]))

    return sum(result)

def euclidean(a, b, ax=0):
    return np.sum((a-b)**2, axis=ax)
#     return (a-b)

def cosine(a,b, ax=0):
    val = 1- np.dot(a,b) /(np.linalg.norm(a))*np.sum(np.linalg.norm(b))
#     print(val)
    return val

def jaccard(a, b, ax=0):
    return (1-np.sum(np.minimum(a,b),axis=ax)/np.sum(np.maximum(a,b),axis=ax))
def distance(instance1, instance2,dist):

    if instance1 == None or instance2 == None:

```

```

        return float("inf")
    error=0

    if(dist=="Cosine"):

        dp1 = dp(instance1[1:], instance2[1:])
        mg1 = (mg(instance1[1:]) * mg(instance2[1:]) + 0.00000000000001)
        error += (1 - ((dp1) / (mg1)))
        #error += cosine(instance1[1:],instance2[1:])

    elif(dist=="Euclidean"):
        for i in range(1, len(instance1)):
            #error += ((instance2[i] - instance1[i]) ** 2)**0.5
            error += euclidean(instance2[i],instance1[i])

    elif(dist == "norml"):
        for i in range(1, len(instance1)):
            error += (instance1[i] - instance2[i])**2
    elif(dist=="Jaccard"):
        # x = instance1[1:]
        # y = instance2[1:]
        #
        # intersection_cardinality = len(set(x).intersection(set(y)))
        # union_cardinality = len(set(x).union(set(y)))
        #error += (intersection_cardinality / float(union_cardinality))
        error+=jaccard(instance1[1:],instance2[1:])
    elif(dist=="manhattan"):
        for i in range(1, len(instance1)):
            error += manhattan(instance2[i],instance1[i])

    return error
def jaccard_similarity(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection) / union

##dummy

def meanInstance(name, instanceList):
    numInstances = len(instanceList)
    if (numInstances == 0):
        return
    numAttributes = len(instanceList[0])
    means = [name] + [0] * (numAttributes-1)
    for instance in instanceList:
        for i in range(1, numAttributes):
            means[i] += instance[i]
    for i in range(1, numAttributes):
        means[i] /= float(numInstances)
    return tuple(means)

def assign(instance, centroids,dist):
    minDistance = distance(instance, centroids[0],dist)
    minDistanceIndex = 0

```

```

    for i in range(1, len(centroids)):
        d = distance(instance, centroids[i], dist)
        if (d < minDistance):
            minDistance = d
            minDistanceIndex = i
    return minDistanceIndex

def createEmptyListOfLists(numSubLists):
    myList = []
    for i in range(numSubLists):
        myList.append([])
    return myList

def assignAll(instances, centroids, labels, dist):
    clusters = createEmptyListOfLists(len(centroids))
    labelclassclusters = createEmptyListOfLists(len(centroids))
    i=0
    for instance in instances:
        clusterIndex = assign(instance, centroids, dist)
        clusters[clusterIndex].append(instance)
        labelclassclusters[clusterIndex].append(labels[i])
        i+=1
    return clusters, labelclassclusters

def computeCentroids(clusters):
    centroids = []
    for i in range(len(clusters)):
        name = "centroid" + str(i)
        centroid = meanInstance(name, clusters[i])
        centroids.append(centroid)
    return centroids

def kmeans(instances, k, animation=False, initCentroids=None, labels=None, dist=None):
    result = {}
    if (initCentroids == None or len(initCentroids) < k):
        # randomly select k initial centroids
        random.seed(time.time())
        centroids = random.sample(instances, k)
    else:
        centroids = initCentroids
    prevCentroids = []
    if animation:
        delay = 1.0 # seconds
        canvas = prepareWindow(instances)
        clusters = createEmptyListOfLists(k)
        clusters[0] = instances
        paintClusters2D(canvas, clusters, centroids, "Initial centroids")
        time.sleep(delay)
    iteration = 0
    while (centroids != prevCentroids):
        iteration += 1
        clusters, label = assignAll(instances, centroids, labels, dist)
        if animation:
            paintClusters2D(canvas, clusters, centroids, "Assign %d" % iteration)
            time.sleep(delay)

```

```

        prevCentroids = centroids
        centroids = computeCentroids(clusters)
        withinss = computeWithinss(clusters, centroids, dist)
        if animation:
            paintClusters2D(canvas, clusters, centroids,
                            "Update %d, withinss %.1f" % (iteration, withinss))
            time.sleep(delay)
        result["clusters"] = clusters
        result["centroids"] = centroids
        result["withinss"] = withinss
        result["labels"] = label
        return result

def computeWithinss(clusters, centroids, dist):
    result = 0
    try:
        for i in range(len(centroids)):
            centroid = centroids[i]
            cluster = clusters[i]
            for instance in cluster:
                result += distance(centroid, instance, dist)

    except TypeError as err:
        print(err)

    return result

# Repeats k-means clustering n times, and returns the clustering
# with the smallest withinss
def repeatedKMeans(instances, k, n):
    bestClustering = {}
    bestClustering["withinss"] = float("inf")
    for i in range(1, n+1):
        print ("k-means trial %d," % i ,
              trialClustering = kmeans(instances, k))
        print ("withinss: %.1f" % trialClustering["withinss"])
        if trialClustering["withinss"] < bestClustering["withinss"]:
            bestClustering = trialClustering
            minWithinssTrial = i
    print ("Trial with minimum withinss:", minWithinssTrial)
    return bestClustering

#####
# This section contains functions for visualizing datasets and
# clustered datasets.
#####

def printTable(instances):
    for instance in instances:
        if instance != None:
            line = instance[0] + "\t"
            for i in range(1, len(instance)):
                line += "%.2f " % instance[i]
            print(line)

```

```

def extractAttribute(instances, index):
    result = []
    for instance in instances:
        result.append(instance[index])
    return result

def paintCircle(canvas, xc, yc, r, color):
    canvas.create_oval(xc-r, yc-r, xc+r, yc+r, outline=color)

def paintSquare(canvas, xc, yc, r, color):
    canvas.create_rectangle(xc-r, yc-r, xc+r, yc+r, fill=color)

def drawPoints(canvas, instances, color, shape):
    random.seed(0)
    width = canvas.winfo_reqwidth()
    height = canvas.winfo_reqheight()
    margin = canvas.data["margin"]
    minX = canvas.data["minX"]
    minY = canvas.data["minY"]
    maxX = canvas.data["maxX"]
    maxY = canvas.data["maxY"]
    scaleX = float(width - 2*margin) / (maxX - minX)
    scaleY = float(height - 2*margin) / (maxY - minY)
    for instance in instances:
        x = 5*(random.random()-0.5)+margin+(instance[1]-minX)*scaleX
        y = 5*(random.random()-0.5)+height-margin-(instance[2]-minY)*scaleY
        if (shape == "square"):
            paintSquare(canvas, x, y, 5, color)
        else:
            paintCircle(canvas, x, y, 5, color)
    canvas.update()

def connectPoints(canvas, instances1, instances2, color):
    width = canvas.winfo_reqwidth()
    height = canvas.winfo_reqheight()
    margin = canvas.data["margin"]
    minX = canvas.data["minX"]
    minY = canvas.data["minY"]
    maxX = canvas.data["maxX"]
    maxY = canvas.data["maxY"]
    scaleX = float(width - 2*margin) / (maxX - minX)
    scaleY = float(height - 2*margin) / (maxY - minY)
    for p1 in instances1:
        for p2 in instances2:
            x1 = margin + (p1[1]-minX)*scaleX
            y1 = height - margin - (p1[2]-minY)*scaleY
            x2 = margin + (p2[1]-minX)*scaleX
            y2 = height - margin - (p2[2]-minY)*scaleY
            canvas.create_line(x1, y1, x2, y2, fill=color)
    canvas.update()

def mergeClusters(clusters):
    result = []
    for cluster in clusters:

```



```

        result.extend(cluster)
    return result

def prepareWindow(instances):
    width = 500
    height = 500
    margin = 50
    root = Tk()
    canvas = Canvas(root, width=width, height=height, background="white")
    canvas.pack()
    canvas.data = {}
    canvas.data["margin"] = margin
    setBounds2D(canvas, instances)
    paintAxes(canvas)
    canvas.update()
    return canvas

def setBounds2D(canvas, instances):
    attributeX = extractAttribute(instances, 1)
    attributeY = extractAttribute(instances, 2)
    canvas.data["minX"] = min(attributeX)
    canvas.data["minY"] = min(attributeY)
    canvas.data["maxX"] = max(attributeX)
    canvas.data["maxY"] = max(attributeY)

def paintAxes(canvas):
    width = canvas.winfo_reqwidth()
    height = canvas.winfo_reqheight()
    margin = canvas.data["margin"]
    minX = canvas.data["minX"]
    minY = canvas.data["minY"]
    maxX = canvas.data["maxX"]
    maxY = canvas.data["maxY"]
    canvas.create_line(margin/2, height-margin/2, width-5, height-margin/2,
                       width=2, arrow=LAST)
    canvas.create_text(margin, height-margin/4,
                       text=str(minX), font="Sans 11")
    canvas.create_text(width-margin, height-margin/4,
                       text=str(maxX), font="Sans 11")
    canvas.create_line(margin/2, height-margin/2, margin/2, 5,
                       width=2, arrow=LAST)
    canvas.create_text(margin/4, height-margin,
                       text=str(minY), font="Sans 11", anchor=W)
    canvas.create_text(margin/4, margin,
                       text=str(maxY), font="Sans 11", anchor=W)
    canvas.update()

def showDataset2D(instances):
    canvas = prepareWindow(instances)
    paintDataset2D(canvas, instances)

def paintDataset2D(canvas, instances):
    canvas.delete(ALL)
    paintAxes(canvas)

```

```

drawPoints(canvas, instances, "blue", "circle")
canvas.update()

def showClusters2D(clusteringDictionary):
    clusters = clusteringDictionary["clusters"]
    centroids = clusteringDictionary["centroids"]
    withinss = clusteringDictionary["withinss"]
    canvas = prepareWindow(mergeClusters(clusters))
    paintClusters2D(canvas, clusters, centroids,
                    "Withinss: %.1f" % withinss)

def paintClusters2D(canvas, clusters, centroids, title=""):
    canvas.delete(ALL)
    paintAxes(canvas)
    colors = ["blue", "red", "green", "brown", "purple", "orange"]
    for clusterIndex in range(len(clusters)):
        color = colors[clusterIndex%len(colors)]
        instances = clusters[clusterIndex]
        centroid = centroids[clusterIndex]
        drawPoints(canvas, instances, color, "circle")
        if (centroid != None):
            drawPoints(canvas, [centroid], color, "square")
            connectPoints(canvas, [centroid], instances, color)
    width = canvas.winfo_reqwidth()
    canvas.create_text(width/2, 20, text=title, font="Sans 14")
    canvas.update()

#####
# Test code
#####

dataset = loadCSV("C:/Users/chat2/Downloads/Iris.csv")
#dataset2 = loadCSV("C:/Users/chat2/Downloads/Iris.csv")

football= loadCSV("C:/Users/chat2/Downloads/football.csv")
dataset1 = []
dataset2 = []
football_label=[]
football_dataset=[]
for t in dataset:
    dataset2.append(t[-1])
    dataset1.append(t[0:4])
showDataset2D(dataset1)

for f in football:
    football_label.append(f[0])
    football_dataset.append(f[1:4])

#showDataset2D(dataset2)
#clustering_cosine = kmeans(dataset1, 3, True, labels=dataset2, dist="Cosine")
#clustering_euc = kmeans(dataset1, 3, True, labels=dataset2, dist="Euclidean")
clustering_jaccard = kmeans(dataset1, 3, True, labels=dataset2, dist="Jaccard")

```

K-Means Clustering

```
#football1 = kmeans(football_dataset,2,animation=False,
labels=football_label,dist="manhattan",initCentroids=[[4,6],[5,4]])
#football2 = kmeans(football_dataset,2,animation=False,
labels=football_label,dist="Euclidain",initCentroids=[[4,6],[5,4]])
#football3 = kmeans(football_dataset,2,animation=False,
labels=football_label,dist="manhattan",initCentroids=[[3,3],[8,3]])
#football4 = kmeans(football_dataset,2,animation=False,
labels=football_label,dist="manhattan",initCentroids=[[3,2],[4,8]])

#print("1",football1["clusters"])
#print("2",football2["clusters"])
#print("3",football3["clusters"])
#print("4",football4["clusters"])

def accuracy(clustering):
    correct_pred = 0
    for clust in clustering['labels']:
        for label in clust:
            if label == mode(clust):
                correct_pred+=1
    return correct_pred/150

print("Accuracy for Cosine Distance is ",accuracy(clustering_cosine))
print("SSE for Cosine Distance is ", clustering_cosine["withinss"])

print("Accuracy for Euclidain Distance is ",accuracy(clustering_euc))
print("SSE for Euclidain Distance is ", clustering_euc["withinss"])

print("Accuracy for Jaccard Distance is ",accuracy(clustering_jaccard))
print("SSE for Jaccard Distance is ", clustering_jaccard["withinss"])

clustering
```