

CS 7267

MACHINE LEARNING

PROJECT 1

UNSUPERVISED LEARNING

#### INSTRUCTOR

**Dr. Zongxing Xie**

**Michael Rizig**

**001008703**

**1. ABSTRACT**

In this project, we are tasked with applying k-mean clustering to a dataset. This clustering method utilizes K number of averages, and groups the data into clusters based on their distance to the closest mean. By repeating k-mean clustering a few times, the clusters become more accurte and representative of their data. The data is then plotted to view the clustering accuracy and view the clear differences between clusters. Our kmtest dataset utilized 2 dimentional data, while the iris dataset utilizes 4 dimentional data. This means our program must adapt to any type of input data. Each dataset will be clustered with and without normalization to visualize the impact of normalizing data.

To view revison history and step by step building of this project view on my github:

<https://github.com/michaelrzg/Machine-Learning-Projects-Python>

**2. Test RESULTS**

**2.1 Clustering with K-means algorithm for kmtest dataset**

**EACH RUN STARTS WITH A RANDOM STARTING CLUSTER LOCATION.**

**Figure 2.1.a:** K=2 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION

**Figure 2.1.b:** K=3 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION

**Figure 2.1.c:** K=4 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION

**Figure 2.1.d:** K=5 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION

**Figure 2.1.e:** K=2 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION

**Figure 2.1.f:** K=3 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION

**Figure 2.1.g:** K=4 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION

**Figure 2.1.h:** K=5 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Figure 2.1: (a)** K=2 , **(b)** K=3, **(c)** K=4, **(d)** K=5, **(e)** K=2 NORMALIZED, **(f)** K=3 NORMALIZED, **(g)** K=4 NORMALIZED **(h)** K=5 NORMALIZED

**2.2 Test Results for Clustering with K-means algorithm for iris dataset**

**This dataset is a 4 dimentional dataset. EACH RUN STARTS WITH A RANDOM STARTING CLUSTER LOCATION**

**Figure 2.2.a:** K=2 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITHOUT NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.b:** K=3 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITHOUT NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.c:** K=4 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITHOUT NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.d:** K=5 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITHOUT NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.e:** K=2 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITH NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.f:** K=3 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITH NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.g:** K=4 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITH NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

**Figure 2.2.h:** K=5 This 3 dimentional graph shows 3 of the 4 dimentions of each datapoint WITH NORMALIZATION. Each group or ‘cluster’ is identifierd by their unique color.

|  |  |  |
| --- | --- | --- |
| **(a)** | **(b)** | **(c)** |
| **(d)** | **(e)** | **(f)** |
| **(g)** | **(h)** | - |

**Figure 2.2: (a)** K=2 , **(b)** K=3, **(c)** K=4, **(d)** K=5, **(e)** K=2 NORMALIZED, **(f)** K=3 NORMALIZED, **(g)** K=4 NORMALIZED **(h)** K=5 NORMALIZED

**3.Discussion**

As we can see from the test results, as we increase the number of clusters (K), our clusters get smaller and more preceise, and can be used more accuratly.However, having too many clusters can cause the clusters to overlap, taking away from the meaning of each cluster. We also observe that normalizing the dataset leads to more accurate and tighter clusters within each run. Gıven more time, it would be interesting to apply this concept to image data to see what types of images can be clustered.

**4. CODES**

**4.1 Code for K-means algorithm for kmtest dataset**

# Name: John

# Number: 123456

# Project 1

# Dataset: kmtest

**4.2 Code for K-means algorithm for iris dataset**

# Michael Rizig

# CS7247 Machine Learning

# Professor Zongxing Xie

# 8/31/24

# Assignment 1: K-means

# import plot and math tools

import matplotlib.pyplot as plot

import seaborn as sea

import numpy

import random

#function to create initial predefiend number of cluster centers (K) with data passed

# Takes in number of clusters (K) and data

# returns random cluster centers from within dataset

def createClusters(numberOfClusters,data):

# create list to store cluster means

    clusterCenters = []

    #pick random values as clusters

    for i in range (numberOfClusters):

        x = random.randint(0,len(data))

        clusterCenters.append(data[x])

    return clusterCenters

# return size of each current group

def groupSizes(K,groupings):

     # create list to store size of each current group

    groupSizes = [0 for i in range(K)]

    # find sizes of each group TODO: Fit this into other loop somehow

    for i in groupings:

        groupSizes[i[1]]+=1

    return groupSizes

# group data into clusters based on distance from each cluster center

# takes in center of clusters and groups data into closest cluster center

def groupData(clusterCenters,data):

    # list to assign groupings

    groupings = []

    #debug

    print("Randomly chosen cluster centers: ",clusterCenters)

    # for eaach data point,

    # calculate the distance between that point and each center

    for x in data:

        #list to hold each distance

        distances=[]

        #parse through each cluster center

        for cluster in clusterCenters:

            #init distanceto 0

            distance=0

            # calculate distance :  sqrt( a^2 + b^2 + c^2...)

            # and add

            for i in range(len(cluster)):

                distance += numpy.sqrt((cluster[i]-x[i])\*\*2)

            #

            #add it list for final comparison

            distances.append(distance)

        #assign the closest cluster center as group

        groupings.append((x,distances.index(min(distances))))

    #debug

    print("Grouping for each value set: ",groupings)

    return clusterCenters,groupings

#this function takes in current groupings, finds average of all values in each group

#then recalls group data to new center clusters.

def recenterGroupings(K,groupings):

    # create list to store average point of each group

    groupAverage = [[0 for i in range(len(groupings[0][0]))] for u in range(K)]

    Sizes = groupSizes(K,groupings)

    print(Sizes)

    # for each datapoint structure: ([x,y,z,...],group#),

    # go through each value in list [x,y,z,...],

    # divide it by total # of comparable values (divide each x by total appearences of x in group)

    # and add that weighted value to its appropriate spot in group avreages

    # at end of loop, we have average point of each group

    for datapoint in groupings:

        for i in range(len(datapoint[0])):

            groupAverage[datapoint[1]][i]+= datapoint[0][i] /Sizes[datapoint[1]]

    #debug

    #print(groupAverage)

    # now regroup data

    newCenters, newGroupings = groupData(groupAverage,[i[0] for i in groupings])

    return newCenters,newGroupings

#helper function for parseCSV

#checks if data is float in string format or not float

def isFloat(x):

    #try to see if passed value is float

    try:

        float(x)

        #return true if this doesnt fail

        return True

    #if we get an exception , it means that value can not be converted to float, so it not one

    except:

        #return false

        return False

#function to parse data from csv and return each set of values as list within list

# takes in string path of csv file and returns all numeric values as list of lists

def parseCSV(path):

    #open file

    file = open(path,'r',encoding='utf-8-sig')

    #place to store lines

    lines=[]

    #insert each line into lines list

    for x in file:

        lines.append(x)

    #place to store value lists

    values = []

    #split each line, then filter out non-numeric values

    for i in lines:

        x= i.split(",")

        out = [a for a in x if isFloat(a)]

        #insert into list

        values.append([float(i) for i in out])

    #return list of values list

    return values

# MAIN:

#print(parseCSV("G:\KSU\CS7267-Machine Learning\Assignments\Project 1 - Unsupervised Learning\Data\iris.csv"))

#parse Data

data = parseCSV("G:\KSU\CS7267-Machine Learning\Assignments\Project 1 - Unsupervised Learning\Data\iris.csv")

#normalize data (delete this section for non-normalized data runs)

K = 5

#generate K number of random cluster centers

clusterCenters = createClusters(K,data)

#group data based on random clusters

clusters,groupings = groupData(clusterCenters,data)

#print(groupings)

#find average of each data group, use that as new center, regroup based on average

newCenters,newGroupings = recenterGroupings(K,groupings)

#print new grouping

#print(groupSizes(len(newCenters),newGroupings))

#colors for each cluster

colors = ["red","blue","green","yellow","purple"]

#make plot 3d

plot.axes(projection='3d')

#plot each datapoint

for duple in newGroupings:

    plot.scatter(duple[0][0],duple[0][1],duple[0][2], color=colors[duple[1]])

#title, save, and show plot

plot.title(f"K={K} Iris dataset with normalization:")

plot.savefig(f"Figures/K={K} Iris-with-norm.png")

plot.show()