

CS 7267

MACHINE LEARNING

PROJECT 2

SUPERVISED LEARNING

#### INSTRUCTOR

**Dr. Zongxing Xie**

**Michael Rizig**

**001008703**

**1. ABSTRACT**

In this project, we are talked with implementing a Supervised Learning Algorithm, namely KNN, or K Nearest Neighbor. This algorithm is related to the Unsupervised Learning Algorithm K-Means as they both deal with clustering. The difference between KNN and K-Means is that KNN is a Supervised Algorithm, meaning that the data comes prelabeled and we can measure the accuracy and performance to later tune the model. First, we will discuss normalization, then we will normalize the data. After normalization we will partition the data between training and testing datasets (70-30 split respectively), and finally we will run our model with K=1,3,5,7, and 9 and measure each K values respective accuracy.

To view revision 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. Large non-precise groups, K is too small.

**Figure 2.1.b:** K=3 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION We see that 3 groups are created

**Figure 2.1.c:** K=4 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION. We see that since the data has 4 natural groups, the algorithm runs perfectly with K=4

**Figure 2.1.d:** K=5 This 2d graph shows the clusters for the kmtest dataset WITHOUT NORMALIZATION. Since we have reached more groups that the data actually has, we see the algorithm start to force a new group in places there is not one. K is too large.

**Figure 2.1.e:** K=2 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION. Once again, we see 2 non-precise groups.

**Figure 2.1.f:** K=3 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION With k=3 we see that general groups are created

**Figure 2.1.g:** K=4 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION Once again we see that since the data has 4 natural groups, the algorithm runs perfectly with K=4

**Figure 2.1.h:** K=5 This 2d graph shows the clusters for the kmtest dataset WITH NORMALIZATION Once again, since we have reached more groups that the data actually has, we see the algorithm start to force a new group in places there is not one. K is too large.

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

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

**Figure 2.2.a:** K=2 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITHOUT NORMALIZATION. We see that two general groups form.

**Figure 2.2.b:** K=3 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITHOUT NORMALIZATION. We see that 3 groups form in different sections of graph.

**Figure 2.2.c:** K=4 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITHOUT NORMALIZATION. We see 4 groups forming, it seems like overfitting as the groups are almost forced. K is too large.

**Figure 2.2.d:** K=5 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITHOUT NORMALIZATION. At this point we have way too many groups, and the meaning of the groups is dampened.

**Figure 2.2.e:** K=2 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITH NORMALIZATION. We see that two general groups form.

**Figure 2.2.f:** K=3 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITH NORMALIZATION. We again see 3 general groups form

**Figure 2.2.g:** K=4 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITH NORMALIZATION. Once again, it seems that K=4 is too large for this data set

**Figure 2.2.h:** K=5 This 3-dimensional graph shows 3 of the 4 dimensions of each datapoint WITH NORMALIZATION. K=5 is too many groups, graph loses some of its value.

|  |  |  |
| --- | --- | --- |
| **(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 precise and can be used more accurately. We see that having a small K value leads to large general groups with not too much information or identification present. 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. For the first dataset (kmtest), we can see that K=4 was the perfect value for the algorithm, but K=5 led to forcing groups that weren’t exactly unique. For the second dataset (iris) we saw that 3 was the perfect K value, and anything above loses some of its meaning. Given 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**

# 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("   ")

        x.remove('')

        print(x)

        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\kmtest.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], color=colors[duple[1]])

#title, save, and show plot

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

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

plot.show()

**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()