################################

# Week 4.1: K-Means Clustering #

################################

## Note: clustering is considered an unsupervised learning method

# importing libraries

import random

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets.samples\_generator import make\_blobs

###################

# Generating Data #

###################

# importing library

np.random.seed(0)

# creating clusters or blobs of 5000 points at a time

# centered at the named coordinates

X, y = make\_blobs(n\_samples=5000, centers=[[4,4], [-2, -1],

[2, -3], [1, 2]], cluster\_std=0.9)

# plotting the data: iteration 1: 4 clusters

plt.scatter(X[:, 0], X[:, 1], marker='.')

# plt.show()

# initializing k means feature matrix

k\_means = KMeans(init = "k-means++", n\_clusters = 4, n\_init = 12)

# fitting feature matrix to blobs above

k\_means.fit(X)

# labelling each point

k\_means\_labels = k\_means.labels\_

k\_means\_labels

# grabbing the coordinates of the cluster centers

k\_means\_cluster\_centers = k\_means.cluster\_centers\_

k\_means\_cluster\_centers

#######################################

# Creating a data visual: iteration 1 #

#######################################

# initializing dimensions

fig = plt.figure(figsize=(6, 4))

# colors: using map, produces array given number of labels

# k\_means\_labels - helps produce coloring

colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k\_means\_labels))))

# building a plot

ax = fig.add\_subplot(1, 1, 1)

# for loop plotting data points and centroids

# k ranges from 0-3, and matches the cluster number

for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])),

colors):

# creating a list of all data points

# checking for 'is a part of' cluster

# labelling points in the set as true, else false

my\_members = (k\_means\_labels == k)

# defining the centroid, or cluster center, using function calculator

cluster\_center = k\_means\_cluster\_centers[k]

# plotting datapoints with color col.

ax.plot(X[my\_members, 0], X[my\_members, 1], 'w',

markerfacecolor=col, marker='.')

# plotting centroids distinctly

ax.plot(cluster\_center[0], cluster\_center[1], 'o',

markerfacecolor=col, markeredgecolor='k',

markersize=6)

# titling

ax.set\_title('KMeans')

# Removing x and y axis ticks

ax.set\_xticks(())

ax.set\_yticks(())

# showing the plot

# plt.show()

#########################

# Practice: iteration 2 #

#########################

# picking initial centroids is somewhat arbitrary

X, y = make\_blobs(n\_samples=5000, centers=[[-1,3], [2, -1],

[-2, -2]], cluster\_std=0.9)

# plotting the data: iteration 2: 3 clusters

plt.scatter(X[:, 0], X[:, 1], marker='.')

# plt.show()

# initializing k means feature matrix

k\_means = KMeans(init = "k-means++", n\_clusters = 3, n\_init = 12)

# fitting feature matrix to blobs above

k\_means.fit(X)

# labelling each point

k\_means\_labels = k\_means.labels\_

k\_means\_labels

# grabbing the coordinates of the cluster centers

k\_means\_cluster\_centers = k\_means.cluster\_centers\_

k\_means\_cluster\_centers

#######################################

# Creating a data visual: iteration 2 #

#######################################

# initializing dimensions

fig = plt.figure(figsize=(6, 4))

# colors: using map, produces array given number of labels

# k\_means\_labels - helps produce coloring

colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k\_means\_labels))))

# building a plot

ax = fig.add\_subplot(1, 1, 1)

# for loop plotting data points and centroids

# k ranges from 0-3, and matches the cluster number

for k, col in zip(range(len([[-1,3], [2, -1], [-2, -2]])),

colors):

# creating a list of all data points

# checking for 'is a part of' cluster

# labelling points in the set as true, else false

my\_members = (k\_means\_labels == k)

# defining the centroid, or cluster center, using function calculator

cluster\_center = k\_means\_cluster\_centers[k]

# plotting datapoints with color col.

ax.plot(X[my\_members, 0], X[my\_members, 1], 'w',

markerfacecolor=col, marker='.')

# plotting centroids distinctly

ax.plot(cluster\_center[0], cluster\_center[1], 'o',

markerfacecolor=col, markeredgecolor='k',

markersize=6)

# titling

ax.set\_title('KMeans')

# Removing x and y axis ticks

ax.set\_xticks(())

ax.set\_yticks(())

# showing the plot

# plt.show()

# cluster pyramid has been established

######################################

# Customer Segmentation: Application #

######################################

# importing library

import wget

# download, and save

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/Cust\_Segmentation.csv'

# wget.download(url, 'Cust\_Segmentation.csv')

# read in, print

import pandas as pd

cust\_df = pd.read\_csv("Cust\_Segmentation.csv")

cust\_df.head()

#########################################

# Customer Segmentation: Pre-processing #

#########################################

# dropping unnecessary vars

df = cust\_df.drop('Address', axis=1)

print(df.head())

# normalizing/ standardizing data:

from sklearn.preprocessing import StandardScaler

# there are a few different ways to do this

# here, using the standard deviation

X = df.values[:,1:]

X = np.nan\_to\_num(X)

Clus\_dataSet = StandardScaler().fit\_transform(X)

print(Clus\_dataSet)

###################################

# Customer Segmentation: Modeling #

###################################

# setting a global variable

clusterNum = 3

# initializing

k\_means = KMeans(init = "k-means++", n\_clusters = clusterNum,

n\_init = 12)

# fitting model to real datapoints

k\_means.fit(X)

# extracting assigned cluster

labels = k\_means.labels\_

labels

###################################

# Customer Segmentation: Insights #

###################################

# new column

df["Clus\_km"] = labels

df.head(5)

# checking centroid values, w/o resorting to a function

print(df.groupby('Clus\_km').mean())

# displays mean value for each var per cluster centroid

#############plot 1: 2d##########################

# plotting little circles for each cluster

area = np.pi \* ( X[:, 1])\*\*2

plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)

plt.xlabel('Age', fontsize=18)

plt.ylabel('Income', fontsize=16)

# plt.show()

#############plot 2: 3d##########################

# plotting 3d figure of income, by age, by education

from mpl\_toolkits.mplot3d import Axes3D

# initializing dimensions

fig = plt.figure(1, figsize=(8, 6))

plt.clf()

# declaring 3d object

ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()

# plt.ylabel('Age', fontsize=18)

# plt.xlabel('Income', fontsize=16)

# plt.zlabel('Education', fontsize=16)

# labeling axes

ax.set\_xlabel('Education')

ax.set\_ylabel('Age')

ax.set\_zlabel('Income')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))

# plt.show()

# in order to display plot within window

# plt.show()

#####################################

# Week 4.2: Hierarchical Clustering #

#####################################

# importing libraries

import numpy as np

import pandas as pd

from scipy import ndimage

from scipy.cluster import hierarchy

from scipy.spatial import distance\_matrix

from matplotlib import pyplot as plt

from sklearn import manifold, datasets

from sklearn.cluster import AgglomerativeClustering

from sklearn.datasets.samples\_generator import make\_blobs

###################

# Generating Data #

###################

# specifying cluster centers

X1, y1 = make\_blobs(n\_samples=50, centers=[[4,4], [-2, -1],

[1, 1], [10,4]], cluster\_std=0.9)

# plotting scatterplot

'''

plt.scatter(X1[:, 0], X1[:, 1], marker='o')

plt.show()

'''

############################

# Agglomerative Clustering #

############################

# aggregation: declaring an agglomerative clustering object

agglom = AgglomerativeClustering(n\_clusters = 4,

linkage = 'average')

# fitting model to the data

agglom.fit(X1, y1)

# creating a figure frame using dimensions 6 x 4

plt.figure(figsize=(6,4))

# scaling data points down, to fit closer together

# min-max range for X1

x\_min, x\_max = np.min(X1, axis=0), np.max(X1, axis=0)

# averaging distance for X1

X1 = (X1 - x\_min) / (x\_max - x\_min)

# looping to display all datapoints

for i in range(X1.shape[0]):

# replacing pts. w/ cluster value

plt.text(X1[i, 0], X1[i, 1], str(y1[i]),

color=plt.cm.nipy\_spectral(agglom.labels\_ [i] / 10.),

fontdict={'weight':'bold', 'size': 9})

# removing x ticks, y ticks, and y axis

plt.xticks([])

plt.yticks([])

# plt.axis('off')

# displauing the plot of original data, then clustering

'''

plt.scatter(X1[:, 0], X1[:, 1], marker='.')

plt.show()

'''

#########################################

# Agglom. Clustering to Dendrogram Plot #

#########################################

# printing distance matrix between features

dist\_matrix = distance\_matrix(X1, X1)

dist\_matrix

# declaring clustering object

Z = hierarchy.linkage(dist\_matrix, 'complete')

# fitting model

'''

dendro = hierarchy.dendrogram(Z)

plt.show()

'''

# declaring clustering object: iteration 2

Z2 = hierarchy.linkage(dist\_matrix, 'average')

# fitting model iteration 2

'''

dendro = hierarchy.dendrogram(Z2)

plt.show()

'''

#############################

# Application: Vehicle Data #

#############################

# importing library

import wget

# download, and save

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/cars\_clus.csv'

# wget.download(url, 'cars\_clus.csv')

# reading in

filename = 'cars\_clus.csv'

pdf = pd.read\_csv(filename)

# printing specs

# print("Shape of dataset: ", pdf.shape)

pdf.head(5)

##########################

# Vehicle Data: Cleaning #

##########################

# printing specs

# print("Shape of dataset before cleaning: ", pdf.size)

# conversion type to num

pdf[[ 'sales', 'resale', 'type', 'price', 'engine\_s',

'horsepow', 'wheelbas', 'width', 'length', 'curb\_wgt',

'fuel\_cap', 'mpg', 'lnsales']] = pdf[['sales', 'resale',

'type', 'price', 'engine\_s',

'horsepow', 'wheelbas', 'width',

'length', 'curb\_wgt', 'fuel\_cap',

'mpg', 'lnsales']].apply(pd.to\_numeric,

errors='coerce')

# dropping missing values

pdf = pdf.dropna()

pdf = pdf.reset\_index(drop=True)

# print ("Shape of dataset after cleaning: ", pdf.size)

pdf.head(5)

###################################

# Vehicle Data: Feature Selection #

###################################

featureset = pdf[['engine\_s', 'horsepow', 'wheelbas', 'width',

'length', 'curb\_wgt', 'fuel\_cap', 'mpg']]

###############################

# Vehicle Data: Normalization #

###############################

from sklearn.preprocessing import MinMaxScaler

x = featureset.values # grabs var list into an array

min\_max\_scaler = MinMaxScaler()

# preparing to scale for comparability

feature\_mtx = min\_max\_scaler.fit\_transform(x)

print(feature\_mtx [0:5])

#####################################

# Vehicle Data: Clustering w/ Scipy #

#####################################

import scipy

leng = feature\_mtx.shape[0]

# initializing

D = scipy.zeros([leng,leng])

# calculating and storing distances

for i in range(leng):

for j in range(leng):

D[i,j] = scipy.spatial.distance.euclidean(feature\_mtx[i],

feature\_mtx[j])

import pylab

import scipy.cluster.hierarchy

Z = hierarchy.linkage(D, 'complete')

from scipy.cluster.hierarchy import fcluster

max\_d = 3

# assigning clusters to vehicle data

clusters = fcluster(Z, max\_d, criterion='distance')

clusters

# setting cluster number

k = 5

clusters = fcluster(Z, k, criterion='maxclust')

print(clusters)

# finally, plotting the dendrogram

fig = plt.figure(figsize=(18, 50))

def llf(id):

return '[%s %s %s]' % (pdf['manufact'][id],

pdf['model'][id],

int(float(pdf['type'][id])))

dendro = hierarchy.dendrogram(Z, leaf\_label\_func=llf,

leaf\_rotation=0, leaf\_font\_size=12,

orientation = 'right')

############################################

# Vehicle Data: Clustering w/ Scikit-learn #

############################################

# creating distance matrix

dist\_matrix = distance\_matrix(feature\_mtx, feature\_mtx)

print(dist\_matrix)

# declaring and fitting clustering object

agglom = AgglomerativeClustering(n\_clusters = 6, linkage = 'complete')

agglom.fit(feature\_mtx)

agglom.labels\_

# assigning labels

pdf['cluster\_'] = agglom.labels\_

# printing data

pdf.head()

import matplotlib.cm as cm

n\_clusters = max(agglom.labels\_) + 1

# coloring and labeling clusters

colors = cm.rainbow(np.linspace(0, 1, n\_clusters))

cluster\_labels = list(range(0, n\_clusters))

# creating figure frame

plt.figure(figsize=(16,14))

for color, label in zip(colors, cluster\_labels):

subset = pdf[pdf.cluster\_ == label]

for i in subset.index:

plt.text(subset.horsepow[i], subset.mpg[i],

str(subset['model'][i]),rotation=25)

plt.scatter(subset.horsepow, subset.mpg, s= subset.price\*10,

c=color, label='cluster'+str(label), alpha=0.5)

# plt.scatter(subset.horsepow, subset.mpg)

plt.legend()

plt.title('Clusters')

plt.xlabel('horsepow')

plt.ylabel('mpg')

plt.show()

# in order to display plot within window

# plt.show()

###############################

# Week 4.3: DBSCAN Clustering #

###############################

# importing libraries

import numpy as np

import pandas as pd

from sklearn.cluster import DBSCAN

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

##################

# Generating Data #

##################

# function allocates points inputted to centroids, for number of simulations run

def createDataPoints(centroidLocation, numSamples,

clusterDeviation):

# creating and storing randomly generated data in matrix shell

# feature matrix: X, target vector: y

X, y = make\_blobs(n\_samples=numSamples,

centers=centroidLocation,

cluster\_std=clusterDeviation)

# standardizing using the mean diff / var method

X = StandardScaler().fit\_transform(X)

return X, y

# assigning clusters

X, y = createDataPoints([[4,3], [2,-1], [-1,4]] , 1500, 0.5)

################################################

# Modeling our Clusters: Declaring and Fitting #

################################################

epsilon = 0.3

minimumSamples = 7

# fitting dbscan density based clustering to generated data

db = DBSCAN(eps=epsilon, min\_samples=minimumSamples).fit(X)

# checking labels, assignment

labels = db.labels\_

labels

#####################

# Identify outliers #

#####################

# creating an array of booleans for testing cluster assignment

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

core\_samples\_mask

# counts number of clusters present

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

n\_clusters\_

# subsetting to only unique values

unique\_labels = set(labels)

print(unique\_labels)

###############

# Data vizzes #

###############

# creating colors for our clusters

colors = plt.cm.Spectral(np.linspace(0, 1, len(unique\_labels)))

# plotting points using colors

'''

for k, col in zip(unique\_labels, colors):

if k == -1:

# black is used for random noise

col = 'k'

class\_member\_mask = (labels == k)

# plotting datapoints assigned clusters

xy = X[class\_member\_mask & core\_samples\_mask]

plt.scatter(xy[:, 0], xy[:, 1], s=50, c=[col], marker=u'o',

alpha=0.5)

plt.show()

'''

######################################

# Practice: contrasting with k-means #

######################################

from sklearn.cluster import KMeans

# initializing k means feature matrix

k\_means = KMeans(init = "k-means++", n\_clusters = 3, n\_init = 12)

# fitting feature matrix to blobs above

k\_means.fit(X)

# labelling each point

k\_means\_labels = k\_means.labels\_

k\_means\_labels

# grabbing the coordinates of the cluster centers

k\_means\_cluster\_centers = k\_means.cluster\_centers\_

k\_means\_cluster\_centers

# plotting little circles for each cluster

area = np.pi \* ( X[:, 1])\*\*2

'''

plt.scatter(X[:, 0], X[:, 1], s=area, c=labels.astype(np.float), alpha=0.5)

plt.show()

# very sporadic, and not fully capturing the overlay of the cluster

'''

#################################################

# 1: Weather Station Data: Download, Reading In #

#################################################

# download

import csv

import wget

# downloading data

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/weather-stations20140101-20141231.csv'

# wget.download(url, 'weather-stations20140101-20141231.csv')

filename = 'weather-stations20140101-20141231.csv'

# read in, print

pdf = pd.read\_csv(filename)

print(pdf.head())

#####################################

# 2: Weather Station Data: Cleaning #

#####################################

# subsetting rows - to remove rows w/ null values in Tm column

pdf = pdf[pd.notnull(pdf["Tm"])]

pdf = pdf.reset\_index(drop=True) # resetting row index each time we do this

print(pdf.head(5))

##########################################

# 3: Weather Station Data: Visualization #

##########################################

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

# from pylab import rcParams # included in matplotlib, pyplot

plt.rcParams['figure.figsize'] = (14,10)

llon=-140

ulon=-50

llat=40

ulat=64

pdf=pdf[(pdf['Long'] > llon) & (pdf['Long'] < ulon) &

(pdf['Lat'] > llat) & (pdf['Lat'] < ulat)]

my\_map = Basemap(projection='merc',

resolution = 'l', area\_thresh = 1000.0,

llcrnrlon=llon, llcrnrlat=llat, # min longitudes and latitudes

urcrnrlon=ulon, urcrnrlat=ulat) # max longitudes and latitudes

my\_map.drawcoastlines()

my\_map.drawcountries()

# my\_map.drawmapboundary()

my\_map.fillcontinents(color = 'white', alpha = 0.3)

# my\_map.shadedrelief() # choosing not to render this fancy layer

# collecting data based on stations

xs, ys = my\_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))

pdf['xm'] = xs.tolist()

pdf['ym'] = ys.tolist()

'''

# visualization1

for index, row in pdf.iterrows():

# x, y = my\_map(row.Long, row.Lat)

my\_map.plot(row.xm, row.ym, markerfacecolor = ([1,0,0]),

marker='o', markersize=5,

alpha=0.75)

# plt.text(x,y,stn)

plt.show()

'''

# had to forego some features, but got through the bulk of the training

#######################################

# 4: Weather Station Data: Clustering #

#######################################

from sklearn.cluster import DBSCAN

import sklearn.utils

from sklearn.preprocessing import StandardScaler

sklearn.utils.check\_random\_state(1000)

Clus\_dataSet = pdf[['xm', 'ym']]

Clus\_dataSet = np.nan\_to\_num(Clus\_dataSet)

Clus\_dataSet = StandardScaler().fit\_transform(Clus\_dataSet)

# computing density based clusters

db = DBSCAN(eps=0.15, min\_samples=10).fit(Clus\_dataSet)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

labels = db.labels\_

pdf["Clus\_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)

clusterNum = len(set(labels))

# printing few select columns

print(pdf[["Stn\_Name", "Tx", "Tm", "Clus\_Db"]].head(5))

# for outliers, cluster label is -1

print(set(labels))

#############################################

# 5: Visualizing Clusters Based on Location #

#############################################

plt.rcParams['figure.figsize'] = (14,10)

my\_map = Basemap(projection='merc',

resolution = 'l', area\_thresh = 1000.0,

llcrnrlon=llon, llcrnrlat=llat, # min longitudes and latitudes

urcrnrlon=ulon, urcrnrlat=ulat) # max longitudes and latitudes

my\_map.drawcoastlines()

my\_map.drawcountries()

# my\_map.drawmapboundary()

my\_map.fillcontinents(color = 'white', alpha = 0.3)

# my\_map.shadedrelief() # choosing not to render this fancy layer

# creating a colored map

colors = plt.get\_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

# Visualization1

for clust\_number in set(labels):

c=(([0.4,0.4,0.4])) if clust\_number == -1 else colors[np.int(clust\_number)]

clust\_set = pdf[pdf.Clus\_Db == clust\_number]

my\_map.scatter(clust\_set.xm, clust\_set.ym, color =c, marker='o',

s=20, alpha=0.85)

if clust\_number != -1:

cenx=np.mean(clust\_set.xm)

ceny=np.mean(clust\_set.ym)

plt.text(cenx,ceny,str(clust\_number), fontsize=25, color='red',)

print("Cluster "+str(clust\_number)+', Avg Temp: '+ str(np.mean(clust\_set.Tm)))

# plt.text(x,y,stn)

plt.show()

####################################################

# 6: Weather Station Data: Clustering, Iteration 2 #

####################################################

sklearn.utils.check\_random\_state(1000)

Clus\_dataSet = pdf[['xm', 'ym', 'Tx', 'Tm', 'Tn']]

Clus\_dataSet = np.nan\_to\_num(Clus\_dataSet)

Clus\_dataSet = StandardScaler().fit\_transform(Clus\_dataSet)

# computing density based clusters

db = DBSCAN(eps=0.3, min\_samples=10).fit(Clus\_dataSet)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

labels = db.labels\_

pdf["Clus\_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)

clusterNum = len(set(labels))

# printing few select columns

print(pdf[["Stn\_Name", "Tx", "Tm", "Clus\_Db"]].head(5))

# for outliers, cluster label is -1

print(set(labels))

##########################################################

# 7: Visualizing Clusters Based on Location, Iteration 2 #

##########################################################

plt.rcParams['figure.figsize'] = (14,10)

my\_map = Basemap(projection='merc',

resolution = 'l', area\_thresh = 1000.0,

llcrnrlon=llon, llcrnrlat=llat, # min longitudes and latitudes

urcrnrlon=ulon, urcrnrlat=ulat) # max longitudes and latitudes

my\_map.drawcoastlines()

my\_map.drawcountries()

# my\_map.drawmapboundary()

my\_map.fillcontinents(color = 'white', alpha = 0.3)

# my\_map.shadedrelief() # choosing not to render this fancy layer

# creating a colored map

colors = plt.get\_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

# Visualization1

for clust\_number in set(labels):

c=(([0.4,0.4,0.4])) if clust\_number == -1 else colors[np.int(clust\_number)]

clust\_set = pdf[pdf.Clus\_Db == clust\_number]

my\_map.scatter(clust\_set.xm, clust\_set.ym, color =c, marker='o',

s=20, alpha=0.85)

if clust\_number != -1:

cenx=np.mean(clust\_set.xm)

ceny=np.mean(clust\_set.ym)

plt.text(cenx,ceny,str(clust\_number), fontsize=25, color='red',)

print("Cluster "+str(clust\_number)+', Avg Temp: '+ str(np.mean(clust\_set.Tm)))

# plt.text(x,y,stn)

plt.show()

# in order to display plot within window

# plt.show()