K-MEANS CLUSTERING LAB

import random

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

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

%matplotlib inline

np.random.seed(0)

X, y = make\_blobs(n\_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster\_std=0.9)

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

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

k\_means.fit(X)

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

k\_means\_labels

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

k\_means\_cluster\_centers

# Initialize the plot with the specified dimensions.

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

# Colors uses a color map, which will produce an array of colors based on

# the number of labels there are. We use set(k\_means\_labels) to get the

# unique labels.

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

# Create a plot

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

# For loop that plots the data points and centroids.

# k will range from 0-3, which will match the possible clusters that each

# data point is in.

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

# Create a list of all data points, where the data points that are

# in the cluster (ex. cluster 0) are labeled as true, else they are

# labeled as false.

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

# Define the centroid, or cluster center.

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

# Plots the datapoints with color col.

ax.plot(X[my\_members, 0], X[my\_members, 1], 'w', markerfacecolor=col, marker='.')

# Plots the centroids with specified color, but with a darker outline

ax.plot(cluster\_center[0], cluster\_center[1], 'o', markerfacecolor=col, markeredgecolor='k', markersize=6)

# Title of the plot

ax.set\_title('KMeans')

# Remove x-axis ticks

ax.set\_xticks(())

# Remove y-axis ticks

ax.set\_yticks(())

# Show the plot

plt.show()

!wget -O Cust\_Segmentation.csv <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/Cust_Segmentation.csv>

import pandas as pd

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

cust\_df.head()

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

df.head()

from sklearn.preprocessing import StandardScaler

X = df.values[:,1:]

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

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

Clus\_dataSet

clusterNum = 3

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

k\_means.fit(X)

labels = k\_means.labels\_

print(labels)

df["Clus\_km"] = labels

df.head(5)

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

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()

from mpl\_toolkits.mplot3d import Axes3D

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

plt.clf()

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)

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))

AGGLOMERATIVE CLUSTERING

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

%matplotlib inline

X1, y1 = make\_blobs(n\_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster\_std=0.9)

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

agglom = AgglomerativeClustering(n\_clusters = 4, linkage = 'average')

agglom.fit(X1,y1)

# Create a figure of size 6 inches by 4 inches.

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

# These two lines of code are used to scale the data points down,

# Or else the data points will be scattered very far apart.

# Create a minimum and maximum range of X1.

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

# Get the average distance for X1.

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

# This loop displays all of the datapoints.

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

# Replace the data points with their respective cluster value

# (ex. 0) and is color coded with a colormap (plt.cm.spectral)

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})

# Remove the x ticks, y ticks, x and y axis

plt.xticks([])

plt.yticks([])

#plt.axis('off')

# Display the plot of the original data before clustering

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

# Display the plot

plt.show()

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

print(dist\_matrix)

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

dendro = hierarchy.dendrogram(Z)

!wget -O cars\_clus.csv <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/cars_clus.csv>

filename = 'cars\_clus.csv'

#Read csv

pdf = pd.read\_csv(filename)

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

pdf.head(5)

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

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')

pdf = pdf.dropna()

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

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

pdf.head(5)

featureset = pdf[['engine\_s', 'horsepow', 'wheelbas', 'width', 'length', 'curb\_wgt', 'fuel\_cap', 'mpg']]

from sklearn.preprocessing import MinMaxScaler

x = featureset.values #returns a numpy array

min\_max\_scaler = MinMaxScaler()

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

feature\_mtx [0:5]

import scipy

leng = feature\_mtx.shape[0]

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

for i in range(leng):

for j in range(leng):

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

D

import pylab

import scipy.cluster.hierarchy

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

from scipy.cluster.hierarchy import fcluster

max\_d = 3

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

clusters

from scipy.cluster.hierarchy import fcluster

k = 5

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

clusters

fig = pylab.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')

from sklearn.metrics.pairwise import euclidean\_distances

dist\_matrix = euclidean\_distances(feature\_mtx,feature\_mtx)

print(dist\_matrix)

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

fig = pylab.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\_using\_dist\_matrix, leaf\_label\_func=llf, leaf\_rotation=0, leaf\_font\_size =12, orientation = 'right')

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

agglom.fit(dist\_matrix)

agglom.labels\_

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

pdf.head()

import matplotlib.cm as cm

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

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

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

# Create a figure of size 6 inches by 4 inches.

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')

pdf.groupby(['cluster\_','type'])['cluster\_'].count()

agg\_cars = pdf.groupby(['cluster\_','type'])['horsepow','engine\_s','mpg','price'].mean()

agg\_cars

plt.figure(figsize=(16,10))

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

subset = agg\_cars.loc[(label,),]

for i in subset.index:

plt.text(subset.loc[i][0]+5, subset.loc[i][2], 'type='+str(int(i)) + ', price='+str(int(subset.loc[i][3]))+'k')

plt.scatter(subset.horsepow, subset.mpg, s=subset.price\*20, c=color, label='cluster'+str(label))

plt.legend()

plt.title('Clusters')

plt.xlabel('horsepow')

plt.ylabel('mpg')

DENSITY BASED CLUSTERING

# Notice: For visualization of map, you need basemap package.

# if you dont have basemap install on your machine, you can use the following line to install it

!conda install -c conda-forge basemap matplotlib==3.1 -y

# Notice: you maight have to refresh your page and re-run the notebook after installation

import numpy as np

from sklearn.cluster import DBSCAN

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

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

%matplotlib inline

def createDataPoints(centroidLocation, numSamples, clusterDeviation):

# Create random data and store in feature matrix X and response vector y.

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

cluster\_std=clusterDeviation)

# Standardize features by removing the mean and scaling to unit variance

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

return X, y

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

epsilon = 0.3

minimumSamples = 7

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

labels = db.labels\_

labels

# Firts, create an array of booleans using the labels from db.

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

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

core\_samples\_mask

# Number of clusters in labels, ignoring noise if present.

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

n\_clusters\_

# Remove repetition in labels by turning it into a set.

unique\_labels = set(labels)

unique\_labels

# Create colors for the clusters.

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

# Plot the points with colors

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

if k == -1:

# Black used for noise.

col = 'k'

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

# Plot the datapoints that are clustered

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

plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)

# Plot the outliers

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

plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)

!wget -O weather-stations20140101-20141231.csv <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/weather-stations20140101-20141231.csv>

import csv

import pandas as pd

import numpy as np

filename='weather-stations20140101-20141231.csv'

#Read csv

pdf = pd.read\_csv(filename)

pdf.head(5)

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

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

pdf.head(5)

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

from pylab import rcParams

%matplotlib inline

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

llon=-140

ulon=-50

llat=40

ulat=65

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 longitude (llcrnrlon) and latitude (llcrnrlat)

urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my\_map.drawcoastlines()

my\_map.drawcountries()

# my\_map.drawmapboundary()

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

my\_map.shadedrelief()

# To collect 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()

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)

# Compute DBSCAN

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))

# A sample of clusters

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

set(labels)

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

from pylab import rcParams

%matplotlib inline

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

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

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

llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (llcrnrlat)

urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my\_map.drawcoastlines()

my\_map.drawcountries()

#my\_map.drawmapboundary()

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

my\_map.shadedrelief()

# To create a color 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)))

from sklearn.cluster import DBSCAN

import sklearn.utils

from sklearn.preprocessing import StandardScaler

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)

# Compute DBSCAN

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))

# A sample of clusters

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

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

from pylab import rcParams

%matplotlib inline

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

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

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

llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (llcrnrlat)

urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my\_map.drawcoastlines()

my\_map.drawcountries()

#my\_map.drawmapboundary()

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

my\_map.shadedrelief()

# To create a color 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)))

CONTENT BASED RECOMENDING SYSTEM

!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%205/data/moviedataset.zip

print('unziping ...')

!unzip -o -j moviedataset.zip

#Dataframe manipulation library

import pandas as pd

#Math functions, we'll only need the sqrt function so let's import only that

from math import sqrt

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#Storing the movie information into a pandas dataframe

movies\_df = pd.read\_csv('movies.csv')

#Storing the user information into a pandas dataframe

ratings\_df = pd.read\_csv('ratings.csv')

#Head is a function that gets the first N rows of a dataframe. N's default is 5.

movies\_df.head()

#Using regular expressions to find a year stored between parentheses

#We specify the parantheses so we don't conflict with movies that have years in their titles

movies\_df['year'] = movies\_df.title.str.extract('(\(\d\d\d\d\))',expand=False)

#Removing the parentheses

movies\_df['year'] = movies\_df.year.str.extract('(\d\d\d\d)',expand=False)

#Removing the years from the 'title' column

movies\_df['title'] = movies\_df.title.str.replace('(\(\d\d\d\d\))', '')

#Applying the strip function to get rid of any ending whitespace characters that may have appeared

movies\_df['title'] = movies\_df['title'].apply(lambda x: x.strip())

movies\_df.head()

#Every genre is separated by a | so we simply have to call the split function on |

movies\_df['genres'] = movies\_df.genres.str.split('|')

movies\_df.head()

#Copying the movie dataframe into a new one since we won't need to use the genre information in our first case.

moviesWithGenres\_df = movies\_df.copy()

#For every row in the dataframe, iterate through the list of genres and place a 1 into the corresponding column

for index, row in movies\_df.iterrows():

for genre in row['genres']:

moviesWithGenres\_df.at[index, genre] = 1

#Filling in the NaN values with 0 to show that a movie doesn't have that column's genre

moviesWithGenres\_df = moviesWithGenres\_df.fillna(0)

moviesWithGenres\_df.head()

ratings\_df.head()

#Drop removes a specified row or column from a dataframe

ratings\_df = ratings\_df.drop('timestamp', 1)

ratings\_df.head()

userInput = [

{'title':'Breakfast Club, The', 'rating':5},

{'title':'Toy Story', 'rating':3.5},

{'title':'Jumanji', 'rating':2},

{'title':"Pulp Fiction", 'rating':5},

{'title':'Akira', 'rating':4.5}

]

inputMovies = pd.DataFrame(userInput)

inputMovies

#Filtering out the movies by title

inputId = movies\_df[movies\_df['title'].isin(inputMovies['title'].tolist())]

#Then merging it so we can get the movieId. It's implicitly merging it by title.

inputMovies = pd.merge(inputId, inputMovies)

#Dropping information we won't use from the input dataframe

inputMovies = inputMovies.drop('genres', 1).drop('year', 1)

#Final input dataframe

#If a movie you added in above isn't here, then it might not be in the original

#dataframe or it might spelled differently, please check capitalisation.

inputMovies

#Filtering out the movies from the input

userMovies = moviesWithGenres\_df[moviesWithGenres\_df['movieId'].isin(inputMovies['movieId'].tolist())]

userMovies

#Resetting the index to avoid future issues

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

#Dropping unnecessary issues due to save memory and to avoid issues

userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('year', 1)

userGenreTable

inputMovies['rating']

#Dot produt to get weights

userProfile = userGenreTable.transpose().dot(inputMovies['rating'])

#The user profile

userProfile

#Now let's get the genres of every movie in our original dataframe

genreTable = moviesWithGenres\_df.set\_index(moviesWithGenres\_df['movieId'])

#And drop the unnecessary information

genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('year', 1)

genreTable.head()

genreTable.shape

#Multiply the genres by the weights and then take the weighted average

recommendationTable\_df = ((genreTable\*userProfile).sum(axis=1))/(userProfile.sum())

recommendationTable\_df.head()

#Sort our recommendations in descending order

recommendationTable\_df = recommendationTable\_df.sort\_values(ascending=False)

#Just a peek at the values

recommendationTable\_df.head()

#The final recommendation table

movies\_df.loc[movies\_df['movieId'].isin(recommendationTable\_df.head(20).keys())]

COLLABRATIVE FILTERING

!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%205/data/moviedataset.zip

print('unziping ...')

!unzip -o -j moviedataset.zip

#Dataframe manipulation library

import pandas as pd

#Math functions, we'll only need the sqrt function so let's import only that

from math import sqrt

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#Storing the movie information into a pandas dataframe

movies\_df = pd.read\_csv('movies.csv')

#Storing the user information into a pandas dataframe

ratings\_df = pd.read\_csv('ratings.csv')

#Head is a function that gets the first N rows of a dataframe. N's default is 5.

movies\_df.head()

#Using regular expressions to find a year stored between parentheses

#We specify the parantheses so we don't conflict with movies that have years in their titles

movies\_df['year'] = movies\_df.title.str.extract('(\(\d\d\d\d\))',expand=False)

#Removing the parentheses

movies\_df['year'] = movies\_df.year.str.extract('(\d\d\d\d)',expand=False)

#Removing the years from the 'title' column

movies\_df['title'] = movies\_df.title.str.replace('(\(\d\d\d\d\))', '')

#Applying the strip function to get rid of any ending whitespace characters that may have appeared

movies\_df['title'] = movies\_df['title'].apply(lambda x: x.strip())

movies\_df.head()

#Dropping the genres column

movies\_df = movies\_df.drop('genres', 1)

#Drop removes a specified row or column from a dataframe

movies\_df.head()

ratings\_df.head()

ratings\_df = ratings\_df.drop('timestamp', 1)

ratings\_df.head()

userInput = [

{'title':'Breakfast Club, The', 'rating':5},

{'title':'Toy Story', 'rating':3.5},

{'title':'Jumanji', 'rating':2},

{'title':"Pulp Fiction", 'rating':5},

{'title':'Akira', 'rating':4.5}

]

inputMovies = pd.DataFrame(userInput)

inputMovies

#Filtering out the movies by title

inputId = movies\_df[movies\_df['title'].isin(inputMovies['title'].tolist())]

#Then merging it so we can get the movieId. It's implicitly merging it by title.

inputMovies = pd.merge(inputId, inputMovies)

#Dropping information we won't use from the input dataframe

inputMovies = inputMovies.drop('year', 1)

#Final input dataframe

#If a movie you added in above isn't here, then it might not be in the original

#dataframe or it might spelled differently, please check capitalisation.

inputMovies

#Filtering out users that have watched movies that the input has watched and storing it

userSubset = ratings\_df[ratings\_df['movieId'].isin(inputMovies['movieId'].tolist())]

userSubset.head()

#Groupby creates several sub dataframes where they all have the same value in the column specified as the parameter

userSubsetGroup = userSubset.groupby(['userId'])

userSubsetGroup.get\_group(1130)

#Sorting it so users with movie most in common with the input will have priority

userSubsetGroup = sorted(userSubsetGroup, key=lambda x: len(x[1]), reverse=True)

userSubsetGroup[0:3]

userSubsetGroup = userSubsetGroup[0:100]

#Store the Pearson Correlation in a dictionary, where the key is the user Id and the value is the coefficient

pearsonCorrelationDict = {}

#For every user group in our subset

for name, group in userSubsetGroup:

#Let's start by sorting the input and current user group so the values aren't mixed up later on

group = group.sort\_values(by='movieId')

inputMovies = inputMovies.sort\_values(by='movieId')

#Get the N for the formula

nRatings = len(group)

#Get the review scores for the movies that they both have in common

temp\_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]

#And then store them in a temporary buffer variable in a list format to facilitate future calculations

tempRatingList = temp\_df['rating'].tolist()

#Let's also put the current user group reviews in a list format

tempGroupList = group['rating'].tolist()

#Now let's calculate the pearson correlation between two users, so called, x and y

Sxx = sum([i\*\*2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float(nRatings)

Syy = sum([i\*\*2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(nRatings)

Sxy = sum( i\*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingList)\*sum(tempGroupList)/float(nRatings)

#If the denominator is different than zero, then divide, else, 0 correlation.

if Sxx != 0 and Syy != 0:

pearsonCorrelationDict[name] = Sxy/sqrt(Sxx\*Syy)

else:

pearsonCorrelationDict[name] = 0

pearsonCorrelationDict.items()

pearsonDF = pd.DataFrame.from\_dict(pearsonCorrelationDict, orient='index')

pearsonDF.columns = ['similarityIndex']

pearsonDF['userId'] = pearsonDF.index

pearsonDF.index = range(len(pearsonDF))

pearsonDF.head()

topUsers=pearsonDF.sort\_values(by='similarityIndex', ascending=False)[0:50]

topUsers.head()

topUsersRating=topUsers.merge(ratings\_df, left\_on='userId', right\_on='userId', how='inner')

topUsersRating.head()

#Multiplies the similarity by the user's ratings

topUsersRating['weightedRating'] = topUsersRating['similarityIndex']\*topUsersRating['rating']

topUsersRating.head()

#Applies a sum to the topUsers after grouping it up by userId

tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weightedRating']]

tempTopUsersRating.columns = ['sum\_similarityIndex','sum\_weightedRating']

tempTopUsersRating.head()

#Creates an empty dataframe

recommendation\_df = pd.DataFrame()

#Now we take the weighted average

recommendation\_df['weighted average recommendation score'] = tempTopUsersRating['sum\_weightedRating']/tempTopUsersRating['sum\_similarityIndex']

recommendation\_df['movieId'] = tempTopUsersRating.index

recommendation\_df.head()

recommendation\_df = recommendation\_df.sort\_values(by='weighted average recommendation score', ascending=False)

recommendation\_df.head(10)

movies\_df.loc[movies\_df['movieId'].isin(recommendation\_df.head(10)['movieId'].tolist())]