CSCI 620

Assignment 8

Abhishek Shah, as5553

-- create collection with given conditions

```
db.Movies.aggregate([{$match:{$and:[{numvotes:{$gt:10000}},{type:'movie'}]}},{$out:'Movies3'}])
```

Question 1

```
import pandas as pd
import pymongo
from pymongo.write_concern import WriteConcern
client = pymongo.MongoClient("mongodb://localhost:27017/")

database = client["imdb620"]
collection = database['Movies3']
df = pd.DataFrame(list(collection.find()))

def normalizeData(columnValues):
    return (columnValues - min(columnValues)) / (max(columnValues) - min(columnValues))

column1 = ['startyear']
column2 = ['avgrating']
df['startyearNorm'] = df[column1].apply(normalizeData)
df['avgratingNorm'] = df[column2].apply(normalizeData)
NormalizedData = df.to_dict(orient='records')
database['Movies4'].insert_many([i for i in NormalizedData], ordered=True)
```

```
-- after normalizing data (Compass)

db.Movies4.aggregate([
    { '$addFields': {
      'kmeansNorm' : ['$startyearNorm','$avgratingNorm'] }
    },
     {'$out':'Movies4'}
 ])
```

Question 2

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
database = client["imdb620"]
collection = database['Movies4']
collection2 = database['Centroid']

def selectRandomDocs(k, g):
    count = 1
    randomDocs = collection.aggregate([
        {"$unwind": "$genres"},
        {"$match": {"genres": g}},
        {"$sample": {"size": k}}
    ])
    collection2.delete_many({})
    for doc in randomDocs:
        collection2.update_one({
            "_id": count,
        }, {
            "$set": {
                "kmeansNorm": doc["kmeansNorm"]
            }
        }, upsert = True)
        count += 1
```

```
    randomDocCentroids = collection2.find({"kmeansNorm": {"$exists": "true"}})
    return randomDocCentroids
```

Question 3

```
import math
from statistics import mean
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
database = client["imdb620"]
collection = database['Movies4']
collection2 = database['Centroid']

def eucDistance(kMeanNormData, kMeansNormCentroid):
    xDiff = abs(kMeanNormData[0] - kMeansNormCentroid[0])
    yDiff = abs(kMeanNormData[1] - kMeansNormCentroid[1])
    return math.sqrt((xDiff * xDiff) + (yDiff * yDiff))



def assigningClusters(k, g, centroidsData):
    collection.update_many({},{"$unset": {"cluster": ""}})
    sameGenreDocs = collection.aggregate([
        {"$unwind": "$genres"},
        {"$match": {"genres": g}},
    ])
    sameGenreMovies = [doc for doc in sameGenreDocs]

    # centroidsData = selectRandomDocs(k, g)
```

```
    allCentroids = [doc for doc in centroidsData]
    eucDistData = []

    for i in sameGenreMovies:
        eucDistData.clear()
        for j in allCentroids:
            kMeanNormData = i['kmeansNorm']
            kMeanNormCentroid = j['kmeansNorm']
            eucDist = eucDistance(kMeanNormData, kMeanNormCentroid)

            eucDistData.append(abs(eucDist))
        minEucDistance = min(eucDistData)
        minValueCentroid = eucDistData.index(minEucDistance) + 1
        collection.update_one(
            {"_id": i["_id"]},
            {"$set": {
                "cluster": minValueCentroid
            }
            })

    allClustersData = collection.find({"cluster": {"$exists": "true"}})
    allClusters = [i for i in allClustersData]
    return allClusters


def updateCentroids(k, allClustersData):
    count = 1
    sse = []
    for i in range(1, k+1):
        newCentroids = []
```

```
        clusters = collection.aggregate([
            {"$match": {"cluster": i}}
        ])
        clustersData = [i for i in clusters]
        startyearNormal = [j["kmeansNorm"][0] for j in clustersData]
        startyearMean = mean([j["kmeansNorm"][0] for j in clustersData])
        avgratingMean = mean([j["kmeansNorm"][1] for j in clustersData])
        newCentroids.extend([startyearMean, avgratingMean])

        for a in startyearNormal:
            sse.append(math.pow((a - startyearMean), 2))

        collection2.update_one(
            {"_id": count},
            {"$set": {
                "kmeansNorm": newCentroids
            }
            }, upsert=True)
        count += 1
    updatedCentroids = collection2.find({"kmeansNorm": {"$exists": "true"}})
    allCentroids = [i for i in updatedCentroids]
    return allCentroids, sse
```

Question 4

```
import math
from numpy import mean
import matplotlib.pyplot as plt
import pymongo
```

```
client = pymongo.MongoClient("mongodb://localhost:27017/")
database = client["imdb620"]
collection = database['Movies4']
collection2 = database['Centroid']


def selectRandomDocs(k, g):
    count = 1
    randomDocs = collection.aggregate([
        {"$unwind": "$genres"},
        {"$match": {"genres": g}},
        {"$sample": {"size": k}}
    ])
    collection2.delete_many({})
    for doc in randomDocs:
        collection2.update_one({
            "_id": count,
        }, {
            "$set": {
                "kmeansNorm": doc["kmeansNorm"]
            }
        }, upsert=True)
        count += 1
    randomDocCentroids = collection2.find({"kmeansNorm": {"$exists": "true"}})
    return randomDocCentroids


def eucDistance(kMeanNormData, kMeansNormCentroid):
    xDiff = abs(kMeanNormData[0] - kMeansNormCentroid[0])
    yDiff = abs(kMeanNormData[1] - kMeansNormCentroid[1])
```

```
    return math.sqrt((xDiff * xDiff) + (yDiff * yDiff))



def assigningClusters(k, g, centroidsData):
    collection.update_many({}, {"$unset": {"cluster": ""}})
    sameGenreDocs = collection.aggregate([
        {"$unwind": "$genres"},
        {"$match": {"genres": g}},
    ])
    sameGenreMovies = [doc for doc in sameGenreDocs]

    # centroidsData = selectRandomDocs(k, g)
    allCentroids = [doc for doc in centroidsData]
    eucDistData = []

    for i in sameGenreMovies:
        eucDistData.clear()
        for j in allCentroids:
            kMeanNormData = i['kmeansNorm']
            kMeanNormCentroid = j['kmeansNorm']
            eucDist = eucDistance(kMeanNormData, kMeanNormCentroid)


            eucDistData.append(abs(eucDist))
        minEucDistance = min(eucDistData)
        minValueCentroid = eucDistData.index(minEucDistance) + 1
        collection.update_one(
            {"_id": i["_id"]},
            {"$set": {
                "cluster": minValueCentroid
            }
```

```
        })

    allClustersData = collection.find({"cluster": {"$exists": "true"}})
    allClusters = [i for i in allClustersData]
    return allClusters


def updateCentroids(k, allClustersData):
    count = 1
    sse = []
    for i in range(1, k + 1):
        newCentroids = []
        clusters = collection.aggregate([
            {"$match": {"cluster": i}}
        ])
        clustersData = [i for i in clusters]
        startyearNormal = [j["kmeansNorm"][0] for j in clustersData]
        startyearMean = mean([j["kmeansNorm"][0] for j in clustersData])
        avgratingMean = mean([j["kmeansNorm"][1] for j in clustersData])
        newCentroids.extend([startyearMean, avgratingMean])

        for a in startyearNormal:
            sse.append(math.pow((a - startyearMean), 2))

        collection2.update_one(
            {"_id": count},
            {"$set": {
                "kmeansNorm": newCentroids
            }
            }, upsert=True)
```

```
        count += 1
    updatedCentroids = collection2.find({"kmeansNorm": {"$exists": "true"}})
    allCentroids = [i for i in updatedCentroids]
    return allCentroids, sse



def generate(g):
    sseFinal = []
    sseSum = 0
    kValue = []
    for i in range(10, 55, 5):
        kValue.append(i)
        initial_centroids = selectRandomDocs(i, g)
        datapoints = assigningClusters(i, g, initial_centroids)
        new, sse = updateCentroids(i, datapoints)
        # sseSum  = sum([i for i in sse])
        iterations = 0
        while iterations < 10:
            print("iter: ", iterations)
            if len(new) == 1:
                break
            datapoints = assigningClusters(i, g, new)
            new, sse = updateCentroids(i, datapoints)
            sseSum += sum([i for i in sse])
            iterations += 1
        sseFinal.append(sseSum)
    plt.plot(kValue, sseFinal)
    plt.show()
```

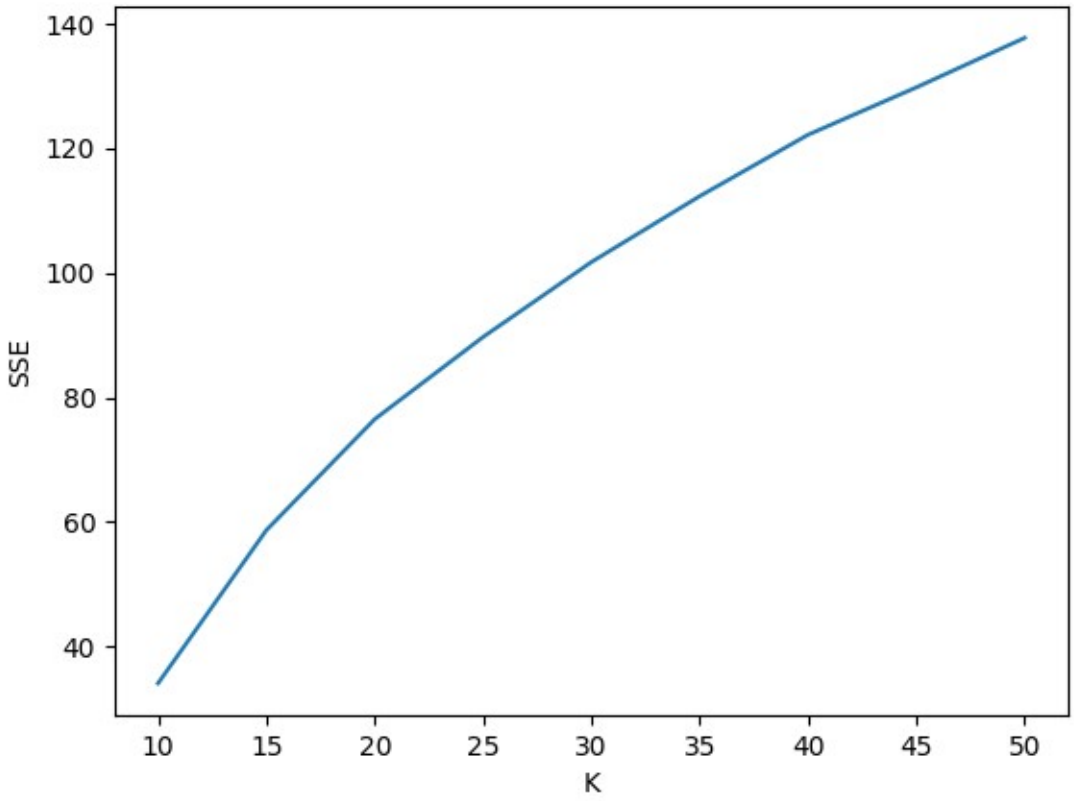
generate('Action')

generate('Horror')

generate('Romance')

generate('Sci-Fi')

generate('Thriller')



(All the graphs are almost similar. I think, I messes up while calculating SSE)

Question 5

I was unable to plot the correct graph. But I do understand the question.

Logic

After I plot the graph for a particular Genre.

I then need to see in the graph, the best value of k (convergence/elbow point).

Run the kMeans for that best value of k.

Now, in the Movies4 collection, I look for clusters assigned.

For every cluster in the document, I look at their kmeansNorm attribute and note them down.

I observe that, all the documents that are assigned to the same cluster have very near/close values

for the kmeansNorm attribute.

That is, same clusters are assigned to documents when their kmeansNorm value are very closely or almost identical/equal wrt each other

I repeat the process, right from plotting the graph, checking the best k value to verifying clusters by checking their kmeansNorm for all the five generes, I ran question 4 code)

Eg:

For **Genre = 'Horror'**

Say for k = 10

Here are some kmeansNorm values for few documents from mongoDB compass, for **cluster = 1**

kmeansNorm:

0.9439252336448598

0.7469879518072289

kmeansNorm:

0.9532710280373832

0.7590361445783131

kmeansNorm:

0.9439252336448598

0.7590361445783131

kmeansNorm:

0.9439252336448598

0.7349397590361445

As you can see, kmeansNorm are very closely related and therefore, all these documents are assigned to the same cluster.

For **Genre = ‘Action’**

Say for k = 10

Here are some kmeansNorm values for few documents from mongoDB compass, for **cluster = 1**

kmeansNorm:

0.9439252336448598

0.5662650602409638

kmeansNorm:

0.8411214953271028

0.6024096385542168

kmeansNorm:

0.8504672897196262

0.49397590361445776

As you can see, kmeansNorm are very closely related and therefore, all these documents are assigned to the same cluster.

For **Genre = ‘Romance’**

Say for k = 10

Here are some kmeansNorm values for few documents from mongoDB compass, for **cluster = 3**

kmeansNorm:

0.819252336448598

0.6169879518072289

kmeansNorm:

0.8232710280373832

0.6290361445783131

As you can see, kmeansNorm are very closely related and therefore, all these documents are assigned to the same cluster.

For **Genre = ‘Sci-Fi’**

Say for k = 10

Here are some kmeansNorm values for few documents from mongoDB compass, for **cluster = 2**

kmeansNorm:

0.9139252336448598

0.7169879518072289

kmeansNorm:

0.9132710280373832

0.7290361445783131

kmeansNorm:

0.9039252336448598

0.7290361445783131

As you can see, kmeansNorm are very closely related and therefore, all these documents are assigned to the same cluster.

For **Genre = ‘Thriller’**

Say for k = 10

Here are some kmeansNorm values for few documents from mongoDB compass, for **cluster = 1**

kmeansNorm:

0.9139252336448598

0.7269879518072289

kmeansNorm:

0.9232710280373832

0.7190361445783131

kmeansNorm:

0.9239252336448598

0.7130361445783131

kmeansNorm:

0.9139252336448598

0.7149397590361445

As you can see, kmeansNorm are very closely related and therefore, all these documents are assigned to the same cluster.