B565-Data Mining
Homework #6

Due on Friday, March 31, 2023, 08:00 p.m.

Instructor: Dr. H. Kurban, Head TA: Md R. Kabir

Jash Shah

March 31, 2023

Expectation-Maximization Algorithm

This part is provided to help you implement the expectation-maximization algorithm.

function EXPECTATION-MAXIMIZATION($\mathbf{D}$, k, ϵ)
  $t \leftarrow 0$
  Randomly initialize $\boldsymbol{\mu_1^t}, ..., \boldsymbol{\mu_k^t}$ ▷ Initialization
  $\boldsymbol{\Sigma_i^t} \leftarrow \mathbf{I}, \forall i = 1, ..., k$
  $P^t(C_i) \leftarrow \frac{1}{k}, \forall i = 1, ..., k$
  repeat
    $t \leftarrow t + 1$
    for $i = 1, ..., k$ and $j = 1, ..., n$ **do** ▷ Expectation step
      $w_{ij} \leftarrow \frac{f(\boldsymbol{x_j}|\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}).P(C_i)}{\sum_{a=1}^{k} f(\boldsymbol{x_j}|\boldsymbol{\mu_a}, \boldsymbol{\Sigma_a}).P(C_a)}$ ▷ posterior probability $P^t(C_i|\boldsymbol{x_j})$
    end for
    for $i = 1, ..., k$ **do** ▷ Maximization Step
      $\boldsymbol{\mu_i^t} \leftarrow \frac{\sum_{j=1}^{n} w_{ij} \boldsymbol{x_j}}{\sum_{j=1}^{n} w_{ij}}$ ▷ re-estimate mean
      $\boldsymbol{\Sigma_i^t} \leftarrow \frac{\sum_{j=1}^{n} w_{ij} (\boldsymbol{x_j} - \boldsymbol{\mu_j})(\boldsymbol{x_j} - \boldsymbol{\mu_j})^T}{\sum_{j=1}^{n} w_{ij}}$ ▷ re-estimate covariance matrix
      $P^t(C_i) \leftarrow \frac{\sum_{j=1}^{n} w_{ij}}{n}$ ▷ re-estimate priors
    end for
  until $\sum_{i=1}^{k} ||\boldsymbol{\mu_i^t} - \boldsymbol{\mu_i^{t-1}}||^2 \leq \epsilon$
end function

Problem 1

Implement Expectation-Maximization (EM) algorithm for Gaussian mixture models (see the EM algorithm above) in *R* or *Python* and call this program G_k. As you present your code explain your protocol for [20 points]

1. initializing each Gaussian
2. maintaining k Gaussian
3. deciding ties
4. stopping criteria

R/Python Code

```
# Sample R Script With Highlighting
```

```
#Performing Data Cleaning on the dataset
import pandas as pd
df=pd.read_csv('diabetic_data.csv')
import numpy as np
df.replace({'?':np.nan},inplace=True)
df1=pd.DataFrame(df.isna().sum())
df1=df1.reset_index()
df1.columns=['Column_Names','Count_of_Nan_Values']
df2=df1[df1['Count_of_Nan_Values']!=0].sort_values(by=
['Count_of_Nan_Values'],ascending=False)
df2['Percentage_of_NAN']=df2['Count_of_Nan_Values']/len(df)*100
```

```
print('The Nan Value columns with percentage are as follows')
print(df2)
#Dropping columns with more than 40 percent null values
df.drop(['weight','payer_code','medical_specialty'],axis=1,inplac
e=True)
#Changing the readmitted column
df['readmitted'] =
df['readmitted'].replace({'>30':1,'<30':1,'NO':0})
#Replacing Age with mean
df['age'] = df['age'].replace({'[70-80)': 75, '[60-70)': 65,
'[50-60)': 55, '[80-90)': 85, '[40-50)': 45, '[30-40)': 35,
'[90-100)': 95, '[20-30)': 25, '[10-20)': 15, '[0-10)': 5})
df_diabetes=df.copy()
df_diabetes.drop(columns=
['encounter_id','patient_nbr'],axis=1,inplace=True)
imbalanced_data=['examide','metformin-rosiglitazone','metformin-
pioglitazone','glimepiride-pioglitazone','glipizide-
metformin','glyburide-
metformin','citoglipton','tolazamide','troglitazone','miglitol','
acarbose','tolbutamide','acetohexamide','chlorpropamide','nategli
nide','repaglinide']
df_diabetes.drop(columns=imbalanced_data,inplace=True)
#Label Encoding in data where there is a ordinality
from sklearn.preprocessing import LabelEncoder
ordinal_columns=['max_glu_serum', 'A1Cresult',
       'metformin', 'glimepiride', 'glipizide', 'glyburide', '
       'pioglitazone',
        'rosiglitazone', 'insulin', 'change', 'diabetesMed']
df_diabetes[['max_glu_serum', 'A1Cresult',

       'metformin', 'glimepiride', 'glipizide',
       'glyburide','pioglitazone',
       'rosiglitazone', 'insulin', 'change', 'diabetesMed']] =
       df_diabetes[['max_glu_serum', 'A1Cresult',
       'metformin', 'glimepiride', 'glipizide', 'glyburide',
       'pioglitazone',
       'rosiglitazone', 'insulin', 'change',
    'diabetesMed']].swifter.apply(LabelEncoder().fit_transform)
df_diabetes =
df_diabetes.drop(df_diabetes.loc[df_diabetes["gender"]=="Unknown/
Invalid"].index, axis=0)
ordinal_columns=['gender','race']
one_hot = pd.get_dummies(df_diabetes[['gender','race']])
df_diabetes=pd.concat([df_diabetes,one_hot],axis=1)
df_diabetes.drop(columns=
['diag_1','diag_2','diag_3','gender','race'],inplace=True)
df_diabetes_final.shape
#Scaling the Dataframe
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df=pd.DataFrame(df_diabetes_final)
scaler.fit(df)
diabetes_scaled=scaler.transform(df)
```

Problem 1 continued on next page...

```
df_diabetes_scaled=pd.DataFrame(diabetes_scaled)
df_sample_scaled=df_diabetes_scaled.copy()
df_sample_scaled=df_sample_scaled.head(10)
#df_diabetes_scaled=df_diabetes_scaled.drop(index=index)
df_diabetes_scaled.shape
def complex_euclidean_distance(u, v):
    return np.sqrt(np.sum(np.abs(u - v) ** 2))
def wcss_emm(input_dataframe,labels_array,no_of_clusters):
    from scipy.spatial.distance import cdist
    input_dataframe_clustered=input_dataframe.copy()
    input_dataframe_clustered['Labels']=labels_array

    new_centroids=input_dataframe_clustered.groupby('Labels').mea
    n()
    new_centroids=new_centroids.T
    total_error=[]
    no_of_clusters_array=np.array(labels_array)
    no_of_clusters=np.unique(no_of_clusters_array)
    for cluster in no_of_clusters:
        df_data_label_cluster=input_dataframe_clustered[input_dat
        aframe_clustered['Labels']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Labels'
        ,axis=1)
        centroids=pd.DataFrame(new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids.
        T,metric=complex_euclidean_distance)
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score(input_dataframe,labels):
    from sklearn.metrics import silhouette_score
    sample_size=5000
    sample_indices=np.random.choice(input_dataframe.shape[0],
    size=sample_size, replace=False)
    sample_data = input_dataframe[subset_indices]
    silhouette_avg=
    silhouette_score(sample_data,labels[sample_indices],
    metric='cosine')
    return silhouette_avg

def Calinski_Harbaz_score(input_dataframe,labels):
    from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs

def davies_bouldin_score(input_dataframe,labels):
    from sklearn.metrics import davies_bouldin_score
    dbs=davies_bouldin_score(input_dataframe,labels)
    return dbs

from scipy.stats import multivariate_normal
import numpy as np
```

```
def initialization_of_GMM(input_dataframe,no_of_clusters):
    '''
    The function takes scaled dataframe as input and initializes
    the GMM means,Covariances,and Weights
    '''
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    # Randomly initialize means vector
    means_vector =
    input_dataframe_values[np.random.choice(input_dataframe_value
    s.shape[0], no_of_clusters, replace=False), :]
    # Initialize covariance matrices for each cluster
    covariances_vector = np.array([np.eye(column)] *
    no_of_clusters)
    # Initialize weights from uniform distribution
    weights_vector = np.ones(no_of_clusters) / no_of_clusters
    return means_vector,covariances_vector,weights_vector


def
fit_Guassian_mixture_models(input_dataframe,no_of_clusters,max_no
_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape

    means,covariances,weights=initialization_of_GMM(input_datafra
    me,no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:

        new_log_likelihood = 0
        for index in range(no_of_clusters):
            try:
                epsilon_weight=1e-6
                cov_inv = np.linalg.pinv(covariances[index] +
                np.diag(np.ones(covariances[index].shape[0]) *
                epsilon_weight))
                new_log_likelihood=new_log_likelihood+weights[ind
                ex]*multivariate_normal.logpdf(input_dataframe_va
                lues,means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        new_log_likelihood_scalar=np.sum(new_log_likelihood)

        '''
        Calculating percentage change
        '''
        if np.abs(((np.abs(new_log_likelihood_scalar-
        previous_log_likelihood_scalar)/new_log_likelihood_scalar
        )*100))<threshold:
            print("The input Threshold was {}".format(threshold))
            print("The calculated threshold is
```

```
            {}".format(np.abs(((np.abs(new_log_likelihood_scalar-
            previous_log_likelihood_scalar)/new_log_likelihood_sc
            alar)*100))))
            break
        #else:
            #print("The input Threshold was
            {}".format(threshold))
            #print("The calculated threshold is
            {}".format(np.abs(((np.abs(new_log_likelihood_scalar-
            previous_log_likelihood_scalar)/new_log_likelihood_sc
            alar)*100))))
        previous_log_likelihood_scalar=new_log_likelihood_scalar
        posterior_probabilities =
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for index in range(no_of_clusters):
            try:
                cov_inv =
                np.linalg.pinv(covariances[index],rcond=1e-10)
            except np.linalg.LinAlgError as e:
                continue
            try:
                posterior_probabilities[:,index] =
                weights[index] *
                multivariate_normal.pdf(input_dataframe_values,
                means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        posterior_probabilities/=np.sum(posterior_probabilities,
        axis=1, keepdims=True)



        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            difference = input_dataframe_values - means[j]
            covariances[j] = np.dot((difference *
            posterior_probabilities[:, j][:, np.newaxis]).T,
            difference) / np.sum(posterior_probabilities[:, j])
            covariances[j] += np.diag(np.ones(column) * 1e-6)
            weights[j] = np.mean(posterior_probabilities[:, j])
        iteration += 1

    return means,posterior_probabilities
#Running the code Multiple Times
expectation_maximization_statistics=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
```

```
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_Guassian_mixture_models
        (df_diabetes_scaled,no_of_clusters,100,1)
        cluster_labels_original=np.array(pd.DataFrame(posterior_p
        robabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(post
        erior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:

            unique_values,value_counts=np.unique(cluster_labels_o
            riginal,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_re
            place)[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled,cl
        uster_labels_array,no_of_clusters)
        #silheoutte_score_value=silheoutte_score(df_diabetes_scal
        ed,cluster_labels_array)
        #print("C2")
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_diab
        etes_scaled,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled,cluster
        _labels_array)
        expectation_maximization_statistics.append([no_of_cluster
        s,no_of_experiments,within_sum_of_square_error,silheoutte
        _score_value,Calinski_Harbaz_score_value,dbs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_df=
pd.DataFrame(expectation_maximization_statistics,columns=
['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz_
score','davies_bouldin_score'])
expectation_maximization_statistics_df_plot=expectation_maximizat
ion_statistics_df.groupby(['No_of_Clusters']).mean().reset_index(
)[['No_of_Clusters','within_sum_of_square_error','Calinski_Harbaz
_score','davies_bouldin_score']]
expectation_maximization_statistics_df_plot
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6,10))
sns.boxplot(x=expectation_maximization_statistics_df['No_of_Clust
ers'],y=expectation_maximization_statistics_df['within_sum_of_squ
are_error'])
```

```
plt.title('Box Plot for GMM SSE (error vs no of clusters)')
plt.show()
import seaborn as sns
plt.figure(figsize=(6,10))
sns.boxplot(x=expectation_maximization_statistics_df['No_of_Clust
ers'],y=expectation_maximization_statistics_df['Calinski_Harbaz_s
core'])
plt.title('Box Plot for GMM Calinski Harabaz Score (Score vs no
of clusters)')
plt.show()
import seaborn as sns
plt.figure(figsize=(6,10))
sns.boxplot(x=expectation_maximization_statistics_df['No_of_Clust
ers'],y=expectation_maximization_statistics_df['davies_bouldin_sc
ore'])
plt.title('Box Plot for GMM Davies Bouldin Score (Score vs no of
clusters)')
plt.show()
ax =
expectation_maximization_statistics_df_plot.plot(x='No_of_Cluster
s', y='davies_bouldin_score')
ax2=expectation_maximization_statistics_df_plot.plot(x='No_of_Clu
sters', y='Calinski_Harbaz_score',secondary_y=True, ax=ax)
ax.set_xlabel('No_of_Clusters')
ax.set_ylabel('Davies Bouldin Score')
ax2.set_ylabel('Calinski_Harabaz_Score')
ax.set_title('Cluster Indices vs No of Clusters')
ax.legend(['DBS'], loc='upper left')
ax2.legend(['CHS'], loc='upper right')
plt.show()
```

Data Cleaning and Exploratory Data Analysis of data

- Reasons for dropping Imbalanced data
  - If a column in a dataset has the same value for all data points,it will not be useful for clustering. This is because clustering algorithms rely on differences or similarities between data points to group them into clusters. When a column has the same value for greater than 95 percent of datapoints, There is less variability in those datapoints which will introduce Skewness in the clustering process. Additionally,it means that this column provides no useful information for distinguishing between data points. Since K-Means is a distance based clustering algorithm,columns having less variability will dominate the results and hence other seemingly important columns will have less impact.
- Handling other categorical variables
  - Since the other columns have a particular order in the dosage/Value we can perform label encoding to these columns, An example is shown by the value counts of Maxgluserum Column None 96420 Norm 2597 greater than 200 1485 greater than 300 1264 Hence we can do label encoding and this wont affect the distance metric since the values are determined by types as mentioned above with None being mapped to 0
- Reasons for Eliminating three columns diag1,diag2,diag3
  - The diag1, diag2, and diag3 columns in the Diabetes dataset contain the ICD-9 codes for the primary, secondary, and additional diagnoses of the patients. Each ICD-9 code corresponds to a specific medical condition or diagnosis, and there are thousands of possible codes. Including these columns in the clustering analysis can result in a high- dimensional dataset with a large number of unique values, which can make it difficult to identify meaningful clusters or interpret the results. Moreover, the presence of these columns can lead to overfitting, as the clustering algorithm may focus too much on these columns and generate clusters based on ICD-9 codes rather than underling patterns in the data

Discussion of Initialization of Gaussians

- Gaussian Mixture model is a parametric technique that makes some assumption about the data. Expectation Maximization is a method to solve for the parameters of the model.
- The main assumption made is Data is distributed normally and we use probabilistic technique to assign clusters.
- Gaussian Mixture Model is a soft clustering algorithm that assigns a probability value for every data point and the value corresponds to the probability that the data point belongs to a cluster.
- **Initialization of Gaussians**
- The means vector is initialized by randomly selecting no of clusters rows from the input dataframe and using them as the initial means for each Gaussian distribution in the mixture.
- The covariances vecor is initialized by creating no of clusters identity matrices of size column, where column is the number of columns (features) in the input data. These identity matrices represent the initial covariance matrices for each Gaussian distribution in the mixture.
- Finally, the weights vector is initialized with equal weights for each Gaussian distribution, which means that all clusters are initially considered to be equally likely.

Discussion of Maintaining k Gaussians

- GMM is a soft clustering method that assigns probabilities to every point belonging to a cluster.
- Main steps are Expectation Step where we calculate Posterior Probabilities of a data point belonging to a particular cluster.
- In the maximization step we maximize the log likelihood of observing the data given the mean and covariances vector.Thus we iteratively update the means and covariances until we get the best log likelihood values
- Thus GMM maintains k centroids by updating the means of the k Gaussian distributions in each iteration of the algorithm using the E-step and M-step. The means represent the centroids of the clusters or Gaussian distributions in the feature space

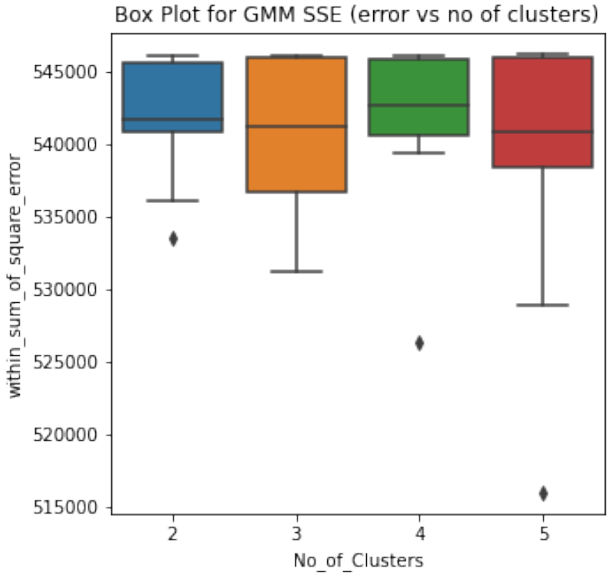
Discussion of Deciding Ties

- Since GMM gives probability values to a data point there can be cases where a data point can have same probability values of belonging to Cluster A as well as Cluster B
- To handle ties in GMM, my implementation use a tie-breaking rule, which randomly assigns the data point to one of the tied clusters with equal probability. This approach ensures that the data point is assigned to one of the clusters, but it introduces randomness into the clustering process.

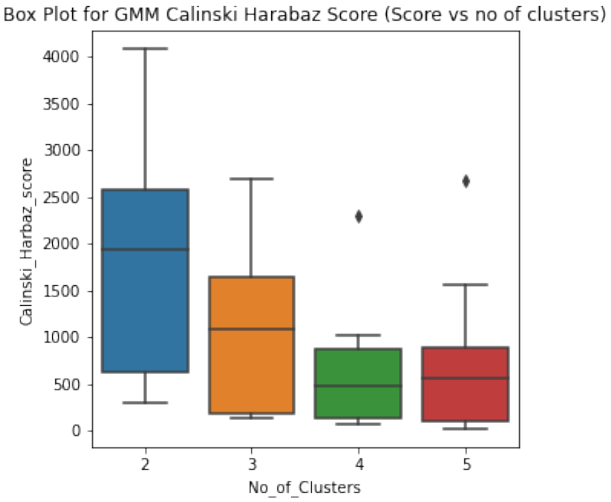
Discussion of Stopping Criteria

- The Alogrithm converges when mean updation is less that a threshold, that means if my previous mean's and my current mean's euclidean distance is less than a certain threshold the algorithm converges.
- My algorithm also implements Log likelihood as convergence criteria, which converges when the change in log likelihood is less than a threshold value which is quite low 1e-6.
  - **Why Log likelihood Convergence method is better?**
  - The log likelihood of the data given the model measures how well the model fits the observed data. In GMM, the EM algorithm iteratively updates the model parameters to maximize the log likelihood of the observed data. When the algorithm converges, the log likelihood stops increasing significantly and stabilizes.
  - The log likelihood change convergence technique tracks the change in log likelihood between iterations. The algorithm continues iterating until the change in log likelihood falls below a certain threshold, indicating that the algorithm has converged.
  - Updating the mean, on the other hand, is a parameter-specific convergence criterion. While updating the mean can be used to monitor the convergence of the algorithm, it is more sensitive to the initialization of the means and the scaling of the data. This can lead to premature convergence or slow convergence in some cases.

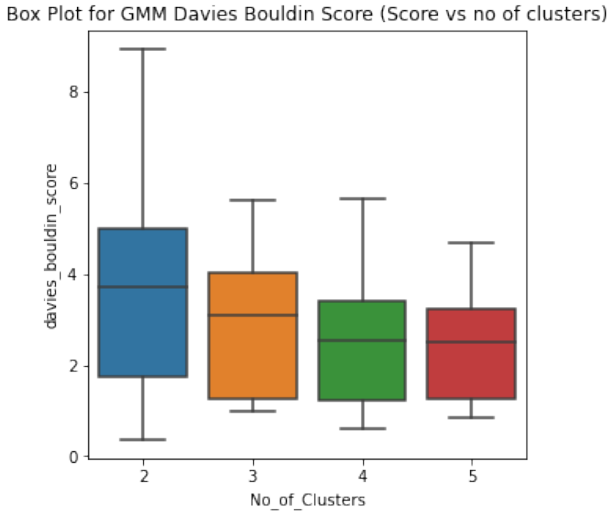
Plot/s


Box Plot for GMM SSE (error vs no of clusters)
within_sum_of_square_error
545000
540000
535000
530000
525000
520000
515000
2
3
4
5
No_of_Clusters

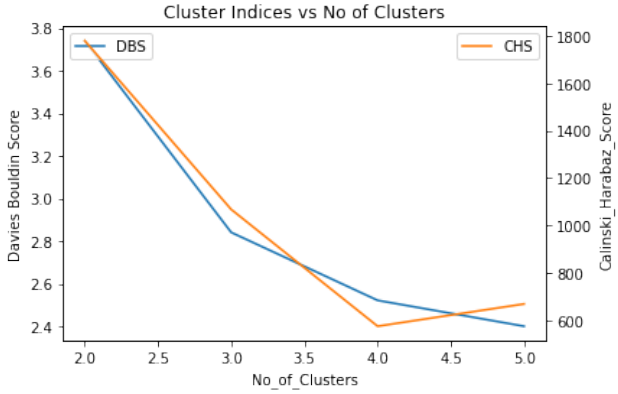
(1)


Box Plot for GMM Calinski Harabaz Score (Score vs no of clusters)
Calinski_Harbaz_score
4000
3500
3000
2500
2000
1500
1000
500
0
2
3
4
5
No_of_Clusters

(2)



(3)



(4)

Discussion of Experiments

- Discussion about the experiments.
- Have ran the code for each cluster 20 times to generalize the results
- The indices used to measure clustering are
  - Within Sum of Squares (WSS) is a measure of the total sum of distances between each point in a cluster and its centroid.
  - Davies-Bouldin score (DBS) is a measure of the similarity between clusters. It is based on the average distance between centroids of different clusters and the distance between points in the same cluster. Lower values of DBS indicate better clustering solutions.
  - Calinski-Harabasz score (CHS) is a measure of the ratio of the between-cluster variance to the within-cluster variance. It evaluates the quality of the clustering solution by comparing the separation between clusters to the dispersion within clusters. Higher values of CHS indicate better clustering solutions

Problem 2

Run your program, G_k, over the Diabetes data set and compare G_k with C_k (your k-means program from homework 4). Click on the below link to download the data set [50 points].

- Diabetes 130-US Hospitals Data Set

Answer the following questions:

1. Initialize G_k and C_k with the same set of initial points (initial centroids for C_k and μ_i-s for G_k are identical) and run them for $k = 2, \ldots, 5$ for 20 runs each. Compare G_k and C_k using two different appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results.

R or Python script

```
# Sample R Script With Highlighting
```

```
import numpy as np
import swifter
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cdist
import time
def get_random_centroids(input_dataframe,no_of_clusters):
    list_of_centroids = []
    for cluster in range(no_of_clusters):
        random_centroid = input_dataframe.swifter.apply(lambda
        x:float(x.sample()))
        list_of_centroids.append(random_centroid)

    centroid_df=pd.concat(list_of_centroids,axis=1)
    centroid_df.index.name='Cluster_Assigned'
    return centroid_df

def get_labels(input_dataframe,centroid_df):
    euclidean_distances = centroid_df.swifter.apply(lambda x:
    np.sqrt(((input_dataframe - x) ** 2).sum(axis=1)))
    return pd.DataFrame(euclidean_distances.idxmin(axis=1))


def get_new_centroids(df_clustered_label,input_dataframe):
    df_original_label_join=input_dataframe.join(df_clustered_lab
    el)
    df_original_label_join.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    new_centroids=df_original_label_join.groupby('Cluster_Assign
    ed').mean()

    return new_centroids.T


def
kmeans_llyod(input_dataframe,no_of_clusters,threshold,no_of_iter
```

Problem 2 continued on next page...

```
ations):
    start_time=time.time()
    iteration=0

    initial_centroid=get_random_centroids(input_dataframe,no_of_
    clusters)
    same_centroid=initial_centroid
    initial_centroid_column_list=initial_centroid.columns.to_lis
    t()

    while True:

        df_cluster_label=get_labels(input_dataframe,initial_cent
        roid)

        df_new_centroids=get_new_centroids(df_cluster_label,inpu
        t_dataframe)
        new_list_of_columns=df_new_centroids.columns.to_list()
        initial_set_columns = set(initial_centroid_column_list)
        new_set_columns = set(new_list_of_columns)
        missing_columns = initial_set_columns - new_set_columns
        for col in missing_columns:
            df_new_centroids[col]=initial_centroid[col]

        from scipy.spatial.distance import euclidean
        scalar_product =
        [euclidean(initial_centroid[col],df_new_centroids[col])
        for col in initial_centroid.columns]

        threshold_calculated=float(sum(scalar_product))/no_of_cl
        usters

        iteration+=1

        if threshold_calculated<threshold:
            print("The input Threshold was
            {}".format(threshold))
            print("The calculated threshold is
            {}".format(threshold_calculated))

        if iteration>no_of_iterations:
            print("Limit for iterations has exceeded")

        if threshold_calculated<threshold or
        iteration>no_of_iterations:
            sum_of_square_error=sum_of_square_error_function(df_
            cluster_label,input_dataframe,df_new_centroids,no_of
            _clusters)
            df_cluster_label_copy=df_cluster_label.copy()
            df_cluster_label_copy.rename(columns=
            {0:'Cluster_Assigned'},inplace=True)
            labels=df_cluster_label_copy['Cluster_Assigned'].to_
            list()
```

```
            #silheoutte_score=silheoutte_score_Kmeans(input_data
            frame,labels)
            silheoutte_score=0
            chs_score=Calinski_Harbaz_score_Kmeans(input_datafra
            me,labels)
            dbs_score=davies_bouldin_score(input_dataframe,label
            s)
            end_time=time.time()
            return
            df_new_centroids,sum_of_square_error,silheoutte_scor
            e,chs_score,dbs_score,end_time-
            start_time,same_centroid
            break
        else:
            initial_centroid= df_new_centroids


def
sum_of_square_error_function(df_cluster_label,input_dataframe,df
_new_centroids,no_of_clusters):
    df_data_label=input_dataframe.join(df_cluster_label)
    df_data_label.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    total_error=[]
    for cluster in range(no_of_clusters):
        df_data_label_cluster=df_data_label[df_data_label['Clust
        er_Assigned']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Cluste
        r_Assigned',axis=1)
        centroids=pd.DataFrame(df_new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
        .T,metric='euclidean')
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import silhouette_score
    silhouette_avg = silhouette_score(input_dataframe, labels)
    return silhouette_avg

def Calinski_Harbaz_score_Kmeans(input_dataframe,labels):
    from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs


def davies_bouldin_score(input_dataframe,labels):
    from sklearn.metrics import davies_bouldin_score
    dbs=davies_bouldin_score(input_dataframe,labels)
    return dbs

from scipy.stats import multivariate_normal
import numpy as np
```

```
def initialization_of_GMM(input_dataframe,no_of_clusters):
    '''
    The function takes scaled dataframe as input and
    initializes the GMM means,Covariances,and Weights
    '''
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    # Randomly initialize means vector
    means_vector =

    np.array(get_random_centroids(input_dataframe,no_of_clusters
    ).T)
    # Initialize covariance matrices for each cluster
    covariances_vector = np.array([np.eye(column)] *
    no_of_clusters)
    # Initialize weights from uniform distribution
    weights_vector = np.ones(no_of_clusters)/no_of_clusters
    return means_vector,covariances_vector,weights_vector


def
fit_Guassian_mixture_models(input_dataframe,no_of_clusters,max_n
o_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape

    means,covariances,weights=initialization_of_GMM(input_datafr
    ame,no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:

        new_log_likelihood = 0
        for index in range(no_of_clusters):
            try:
                epsilon_weight=1e-6
                cov_inv = np.linalg.pinv(covariances[index] +
                np.diag(np.ones(covariances[index].shape[0]) *
                epsilon_weight))

                new_log_likelihood=new_log_likelihood+weights[in

                dex]*multivariate_normal.logpdf(input_dataframe_
                values,means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        new_log_likelihood_scalar=np.sum(new_log_likelihood)

        '''
        Calculating percentage change
        '''
        if np.abs(((np.abs(new_log_likelihood_scalar-
```

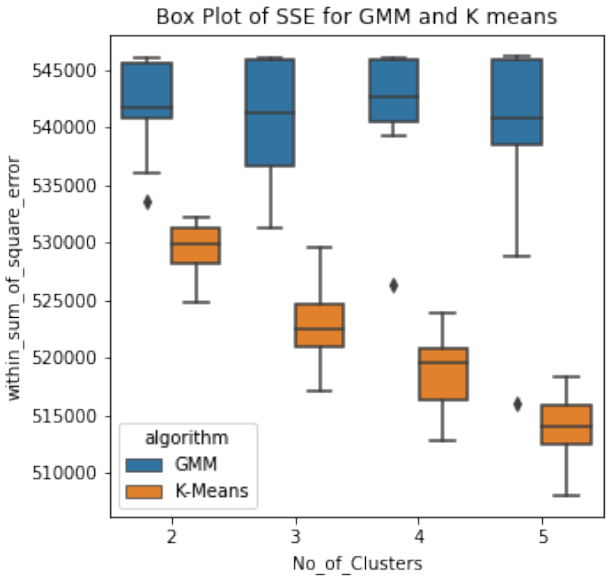
```
        previous_log_likelihood_scalar)/new_log_likelihood_scala
        r)*100))<threshold:
            print("The input Threshold was
            {}".format(threshold))
            print("The calculated threshold is


            {}".format(np.abs(((np.abs(new_log_likelihood_scalar
            -
            previous_log_likelihood_scalar)/new_log_likelihood_s
            calar)*100))))
            break



        previous_log_likelihood_scalar=new_log_likelihood_scalar

        posterior_probabilities =
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for index in range(no_of_clusters):
            try:
                cov_inv =
                np.linalg.pinv(covariances[index],rcond=1e-10)
            except np.linalg.LinAlgError as e:
                continue
            try:
                posterior_probabilities[:,index] =
                weights[index] *
                multivariate_normal.pdf(input_dataframe_values,
                means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue

        posterior_probabilities/=np.sum(posterior_probabilities,
        axis=1, keepdims=True)



        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            difference = input_dataframe_values - means[j]
            covariances[j] = np.dot((difference *
            posterior_probabilities[:, j][:, np.newaxis]).T,
            difference) / np.sum(posterior_probabilities[:, j])
            covariances[j] += np.diag(np.ones(column) * 1e-6)
            weights[j] = np.mean(posterior_probabilities[:, j])
```

```
        iteration += 1

    return means,posterior_probabilities
error_values_kmeans_same_centroid=[]
error_values_emm_same_centroid=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        final_centroids,sum_of_squared_error,sil_score,chs_score
        ,dbs_score,run_time,same_centroid=kmeans_llyod(pd.DataFr
        ame(df_diabetes_scaled),no_of_clusters,10,100)
        error_values_kmeans_same_centroid.append([no_of_clusters
        ,no_of_experiments,sum_of_squared_error,sil_score,chs_sc
        ore,dbs_score,run_time])

        means,posterior_probabilities=fit_Guassian_mixture_model
        s(df_diabetes_scaled,no_of_clusters,100,1)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:

            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled,c
        luster_labels_array,no_of_clusters)
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_dia
        betes_scaled,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled,cluste
        r_labels_array)
        error_values_emm_same_centroid.append([no_of_clusters,no
        _of_experiments,within_sum_of_square_error,silheoutte_sc
        ore_value,Calinski_Harbaz_score_value,dbs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_same_centroid_df=
pd.DataFrame(error_values_emm_same_centroid,columns=
['No_of_Clusters', 'Iteration Number',
```

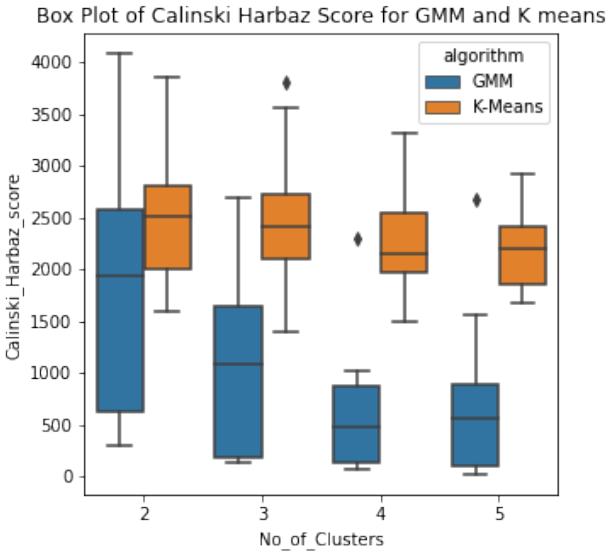
```
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])
error_values_kmeans_same_centroid_df=
pd.DataFrame(error_values_kmeans_same_centroid,columns=
['No_of_Clusters', 'Iteration
Number','within_sum_of_square_error','Silheoutte_Score','Calinsk
i_Harbaz_score','davies_bouldin_score','run_time'])

#Plotting of graphs
import seaborn as sns
expectation_maximization_statistics_df['algorithm']='GMM'
error_values_kmeans_same_centroid_df['algorithm']='K-Means'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_df[
['algorithm','No_of_Clusters','within_sum_of_square_error','Cali
nski_Harbaz_score','davies_bouldin_score']],
error_values_kmeans_same_centroid_df[['algorithm','No_of_Cluster
s','within_sum_of_square_error','Calinski_Harbaz_score','davies_
bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='within_sum_of_square_error',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means','GMM'])],ax=ax);
plt.title('Box Plot of SSE for GMM and K means')
plt.show()

import seaborn as sns
expectation_maximization_statistics_df['algorithm']='GMM'
error_values_kmeans_same_centroid_df['algorithm']='K-Means'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_df[
['algorithm','No_of_Clusters','within_sum_of_square_error','Cali
nski_Harbaz_score','davies_bouldin_score']],
error_values_kmeans_same_centroid_df[['algorithm','No_of_Cluster
s','within_sum_of_square_error','Calinski_Harbaz_score','davies_
bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='Calinski_Harbaz_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means','GMM'])],ax=ax);
plt.title('Box Plot of SSE for GMM and K means')
plt.show()
import seaborn as sns
expectation_maximization_statistics_df['algorithm']='GMM'
error_values_kmeans_same_centroid_df['algorithm']='K-Means'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_df[
['algorithm','No_of_Clusters','within_sum_of_square_error','Cali
nski_Harbaz_score','davies_bouldin_score']],
```

```
error_values_kmeans_same_centroid_df[['algorithm','No_of_Cluster
s','within_sum_of_square_error','Calinski_Harbaz_score','davies_
bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='davies_bouldin_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means','GMM'])],ax=ax);
plt.title('Box Plot of SSE for GMM and K means')
plt.show()
```

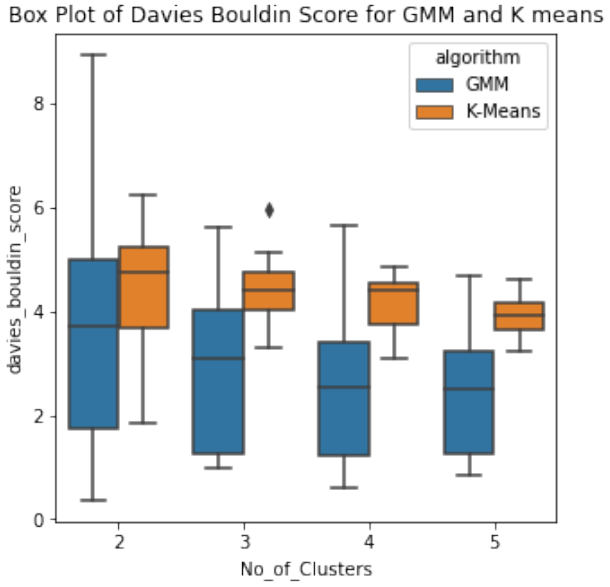
Plot/s



(5)



(6)



(7)

Discussion of Experiments

- Above experiments have initialized same sets of initital centroids, even though randomly the clusters are assigned as same for both algotithms.
- We can observe from the box plots of different clusters that:
  - The Median for Within Sum of Squared errors is slightly lower for K means and it shows K means can perform better than GMM but the difference is not huge and we cannot quantitatively conclude the same.
  - The probable reasons can be

- When the clusters in the data are well-separated and clearly distinct, it is easier for K-means to accurately assign each data point to its nearest centroid, resulting in a lower WSS. In contrast, GMM may have more difficulty accurately assigning data points to clusters if the clusters are overlapping or have complex structures, which can lead to a higher WSS.
- Well defined cluster separation is visible from Calinski Harabaz score and davies bouldin score
- Calinski-Harabasz Score (CHS) is a clustering evaluation metric used to measure the quality of clustering solutions. It calculates the ratio of the between-cluster variance to the within-cluster variance. The CHS is higher when the clusters are well separated and the within-cluster variance is small, and lower when the clusters are overlapping and the within-cluster variance is large.
- Davies-Bouldin Score (DBS) is a clustering evaluation metric used to measure the similarity between clusters. It calculates the average similarity between each cluster and its most similar cluster, based on the ratio of within-cluster and between-cluster distances. A lower DBS indicates better clustering solutions with higher intra-cluster similarity and lower inter-cluster similarity.
- Thus by the definition we can see that since davies bouldin score of K means is less than that of GMM it is showing better results.
- However choice for clustering depends on other factors as well which we need to consider

2. Run your G_k without updating the covariance matrices and priors across iterations. Compare G_k and C_k using two different appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results.

R or Python script

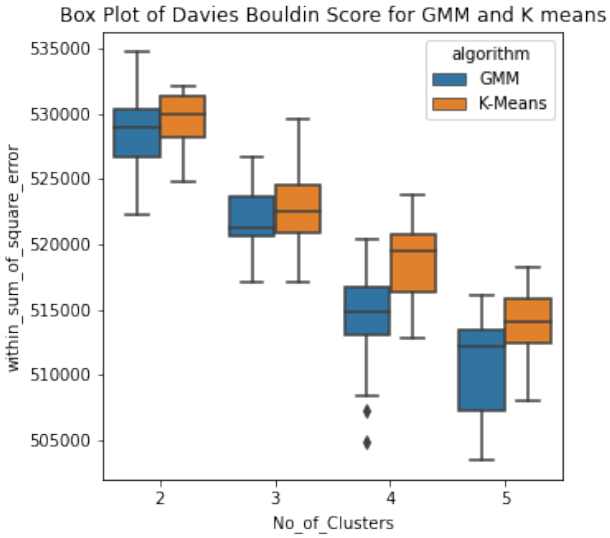
```
# Sample R Script With Highlighting
```

```
from scipy.stats import multivariate_normal
import numpy as np
from scipy.spatial.distance import euclidean

def initialization_of_GMM(input_dataframe,no_of_clusters):
    '''
    The function takes scaled dataframe as input and
    initializes the GMM means,Covariances,and Weights
    '''
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    # Randomly initialize means vector
    means_vector =
    input_dataframe_values[np.random.choice(input_dataframe_valu
    es.shape[0], no_of_clusters, replace=False), :]
    # Initialize covariance matrices for each cluster
    covariances_vector = np.array([np.eye(column)] *
    no_of_clusters)
    # Initialize weights from uniform distribution
    weights_vector = np.ones(no_of_clusters) / no_of_clusters
    return means_vector,covariances_vector,weights_vector


def
fit_Guassian_mixture_models_without_covariances(input_dataframe,
no_of_clusters,max_no_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means,covariances,weights=initialization_of_GMM(input_datafr
    ame,no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:

        new_log_likelihood = 0
        for index in range(no_of_clusters):
            try:
                epsilon_weight=1e-6
                cov_inv = np.linalg.pinv(covariances[index] +
                np.diag(np.ones(covariances[index].shape[0]) *
                epsilon_weight))

                new_log_likelihood=new_log_likelihood+weights[in
```

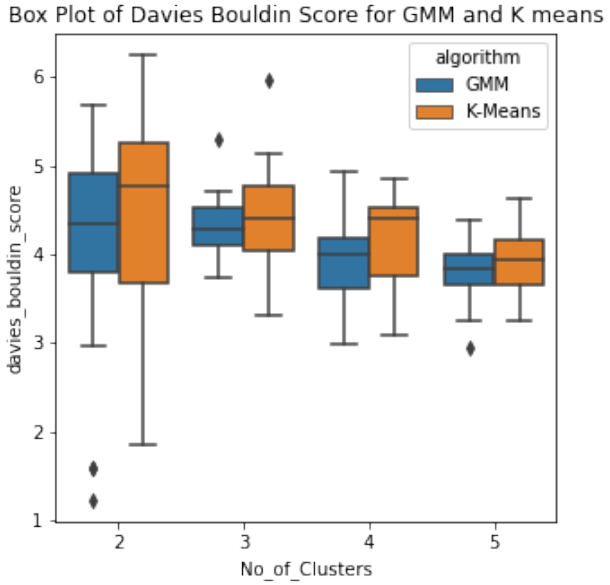
```
            dex]*multivariate_normal.logpdf(input_dataframe_
            values,means[index], cov_inv)
        except np.linalg.LinAlgError as e:
            continue
    new_log_likelihood_scalar=np.sum(new_log_likelihood)
    previous_means_df=pd.DataFrame(means)

    posterior_probabilities =
    np.zeros((len(input_dataframe_values),no_of_clusters))
    for index in range(no_of_clusters):
        try:
            cov_inv =
            np.linalg.pinv(covariances[index],rcond=1e-10)
        except np.linalg.LinAlgError as e:
            continue
        try:
            posterior_probabilities[:,index] =
            weights[index] *
            multivariate_normal.pdf(input_dataframe_values,
            means[index], cov_inv)
        except np.linalg.LinAlgError as e:
            continue




    for j in range(no_of_clusters):
        weighted_sum = np.zeros((1, means.shape[1]))
        sum_posterior = 0.0
        for i in range(row):
            weighted_sum += posterior_probabilities[i][j] *
            input_dataframe_values[i]
            sum_posterior += posterior_probabilities[i][j]
        means[j] = weighted_sum/sum_posterior


    new_means_df=pd.DataFrame(means)


    euclidean_distance=[]
    for col in new_means_df.columns:
        col_distance = euclidean(previous_means_df[col],
        new_means_df[col])
        euclidean_distance.append(col_distance)
    threshold_calculated=sum(euclidean_distance)/no_of_clust
    ers


    iteration += 1
    if threshold_calculated<threshold:
        return means,posterior_probabilities
    if iteration>max_no_of_iterations:
```

```
            return means,posterior_probabilities
expectation_maximization_statistics_without_updation=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_Guassian_mixture_model
        s_without_covariances(df_diabetes_scaled,no_of_clusters,
        100,10)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:
            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            if len(indices)==0:
                indices=[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled,c
        luster_labels_array,no_of_clusters)
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_dia
        betes_scaled,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled,cluste
        r_labels_array)
        silheoutte_score_value=0
        expectation_maximization_statistics_without_updation.app
        end([no_of_clusters,no_of_experiments,within_sum_of_squa
        re_error,silheoutte_score_value,Calinski_Harbaz_score_va
        lue,dbs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_without_updation_df=
pd.DataFrame(expectation_maximization_statistics_without_updatio
n,columns=['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])
```

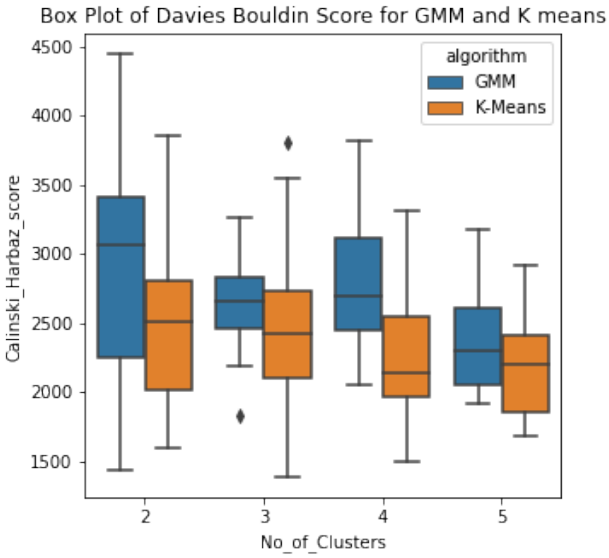
Plot/s


Box Plot of Davies Bouldin Score for GMM and K means
algorithm
GMM
K-Means
within_sum_of_square_error
535000
530000
525000
520000
515000
510000
505000
2
3
4
5
No_of_Clusters

(8)


Box Plot of Davies Bouldin Score for GMM and K means
algorithm
GMM
K-Means
davies_bouldin_score
6
5
4
3
2
1
2
3
4
5
No_of_Clusters

(9)



(10)

Discussion of Experiments

- From the graphs we can see
- The WCSS values of K means and GMM are very close to each other.
- The Calinski Bouldin Score is also very close to each other similarly for Davies Bouldin score.
- Thus from the above graph we can conclude K means and GMM converge at the same time if we do not update covariances matrix for Gaussian Mixture Models
- **Why K means and GMM converge at the same time**
  - K-means and GMM may converge at the same time if the covariances and weights vector are not updated during the GMM algorithm.
  - This is because when the covariance matrix is fixed to a certain value, the GMM algorithm becomes similar to K-means, where the data points are assigned to the nearest centroid based on the Euclidean distance.
  - In the case of GMM with fixed covariances, the algorithm still estimates the mean values and the mixing coefficients (the weights vector) using the EM algorithm.
  - The E-step of the EM algorithm calculates the probability of each data point belonging to each Gaussian distribution based on their mean values and fixed covariance matrix.
  - The M-step updates the mean values and mixing coefficients based on the calculated probabilities.
  - However it resembles K means algorithm and converges at near similar points.

3. Perform PCA over the Diabetes data set. Create a new data set, Δ_R, with using 90% of the variance. Compare G_k and C_k over Δ_R using two different appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results.

R or Python script

```
# Sample R Script With Highlighting
```

```
#Performing PCA on Data
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler

def covariance(input_dataframe):
    '''
    This function takse input as a standardized dataframe
    '''
    input_dataframe_mean =
    input_dataframe.swifter.apply(np.mean, axis=0)
    input_dataframe_centered= input_dataframe-
    input_dataframe_mean
    with tqdm(total=input_dataframe.shape[1], desc="Calculating
    Covariance Matrix") as pbar:
        cov_matrix=np.cov(input_dataframe.T)
        pbar.update()
    return cov_matrix,input_dataframe_centered

def principal_component_analysis(input_dataframe):
    '''
    This function takes input_dataframe,stadndardizes it and
    number of components as the number of components required
    by PC
    '''
    scaler = StandardScaler()
    input_dataframe_scaled
    =pd.DataFrame(scaler.fit_transform(input_dataframe))
    #Calling the covriance function
    covariance_matrix,input_dataframe_centered=covariance(input_
    dataframe_scaled)
    #Calculates Covariance Matirx
    eigen_values,eigen_vectors=np.linalg.eig(covariance_matrix)
    #Calculates Eigen Values and Eigen Vectors
    sorted_indices=np.argsort(eigen_values)
    #Sort the elements in descending order
    sorted_indices=sorted_indices[::-1]


    explained_variances = eigen_values / np.sum(eigen_values)

    variance_explained_ratios =
    pd.DataFrame(explained_variances[sorted_indices], columns=
    ["variance_explained_ratio"])
    variance_explained_ratios["cumulative_variance_explained_rat
    io"] =
    variance_explained_ratios["variance_explained_ratio"].cumsum
    ()

    #Find the number of components that explain 90% of variance
    number_of_components =
```

```
    variance_explained_ratios["cumulative_variance_explained_rat
    io"][variance_explained_ratios["cumulative_variance_explaine
    d_ratio"] <= 0.90].count() + 1

    print("Number of Principal components explain 90% of
    variance are {}".format(number_of_components))



    #Taking Top Eigen Values and Top Eigen Vectors
    top_eigen_values_indices=sorted_indices[:number_of_component
    s]
    top_eigen_vectors=eigen_vectors[:,top_eigen_values_indices]

     #Variance Calculations Plot
    explained_variances = eigen_values/np.sum(eigen_values)
    variance_explained =
    pd.DataFrame(eigen_values[top_eigen_values_indices] /
    sum(eigen_values))
    variance_explained['PC_Feature']=top_eigen_values_indices
    variance_explained_plot=pd.Series(eigen_values[top_eigen_val
    ues_indices] / sum(eigen_values))

    #Cumulative Variance Plot
    cumulative_variance_explained =
    np.cumsum(variance_explained_plot)
    cumulative_variance_explained_plot =
    pd.Series(cumulative_variance_explained)



    #Projecting Principal Components
    principal_components=input_dataframe_centered.dot(top_eigen_
    vectors)
    principal_components.columns=[f'PC{i+1}' for i in
    range(number_of_components)]



    #Calculate the loadings
    loadings =
    pd.DataFrame(top_eigen_vectors,index=input_dataframe.columns
    )

    df_principal_components=pd.DataFrame(principal_components,
    columns=[f'PC{i+1}' for i in range(number_of_components)])
    #PLotting the graph
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    ax[0].plot(np.arange(1,
    number_of_components+1),variance_explained_plot, 'o-')
```

```
    ax[0].set_xlabel('Principal Component')
    ax[0].set_ylabel('Proportion of Variance Explained')
    ax[0].set_title('Scree Plot')


    ax[1].plot(np.arange(1,
    number_of_components+1),cumulative_variance_explained_plot,
    'o-')
    ax[1].set_xlabel('Principal Component')
    ax[1].set_ylabel('Cumulative Proportion of Variance
    Explained')
    ax[1].set_title('Cumulative Scree Plot')
    plt.tight_layout()
    plt.show()

    #Correlation between PC1 and PC2

    plt.scatter(principal_components['PC1'],
    principal_components['PC2'])
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Scatter plot of PC1 against PC2')
    plt.show()

    principal_components_temp=principal_components[['PC1','PC2']]
    corr_matrix = principal_components_temp.corr()
    print('Correlation matrix:')
    print(corr_matrix)

    total_variance_explained=cumulative_variance_explained_plot[
    1]
    print("The total variance explained by first two PC's is
    {}".format(total_variance_explained))

    return
    variance_explained,loadings,principal_components,cumulative_
    variance_explained
#Running EM algorithm on reduced dataset
expectation_maximization_statistics_kmeans_plus_plus_pca=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_Guassian_mixture_model
        s_kmeans_plus_plus(principal_components,no_of_clusters,1
        00,1)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))

        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
```

```
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:
            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            if len(indices)==0:
                indices=[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        try:
            within_sum_of_square_error=wcss_emm(principal_compon
            ents,cluster_labels_array,no_of_clusters)
        except KeyError as e:
            continue
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(princi
        pal_components,cluster_labels_array)
        dbs_value=davies_bouldin_score(principal_components,clus
        ter_labels_array)
        expectation_maximization_statistics_kmeans_plus_plus_pca
        .append([no_of_clusters,no_of_experiments,within_sum_of_
        square_error,silheoutte_score_value,Calinski_Harbaz_scor
        e_value,dbs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_kmeans_plus_plus_pca_df=
pd.DataFrame(expectation_maximization_statistics_kmeans_plus_plu
s_pca,columns=['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])
# Running K means on reduced data
import numpy as np
import swifter
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cdist
import time
def kmeans_pp_init(input_dataframe,no_of_clusters):
    '''
    K-means++ is a variant of the K-means algorithm that aims
    to improve the initial centroids' selection
    in the clustering process.
    The standard K-means algorithm initializes the cluster
    centroids randomly,
    which can lead to suboptimal clustering results,
    especially if the dataset has complex or irregular
    structures.
    '''
    list_of_centroids=[]
    #Choosing the first centroid randomly
```

```
    centroid = input_dataframe.apply(lambda x:
    float(x.sample()))
    list_of_centroids.append(centroid)

    iterator=2
    while iterator<=no_of_clusters:
        '''
        Calculating the distances from the centroid to every
        data point
        If the no of centroids are more than 1 calculate the
        distance from every centroid and take minimum distance
        '''
        distances =
        np.array(np.amin(cdist(input_dataframe,list_of_centroids
        ,metric='euclidean'),axis=1))
        #Next centroid will be selected with probability
        proportional to the distance

        probs = distances / np.sum(distances)
        '''
        Selection of the next centroids
        '''
        next_centroid =

        input_dataframe.iloc[np.random.choice(len(input_datafram
        e),p=probs)]
        list_of_centroids.append(next_centroid)
        iterator+=1

    centroid_df=pd.concat(list_of_centroids,axis=1,ignore_index=
    True)
    #Naming the column as Label for ease of purpose
    centroid_df.index.name='Cluster_Assigned'


    return centroid_df


def get_labels(input_dataframe,centroid_df):
    euclidean_distances = centroid_df.swifter.apply(lambda x:
    np.sqrt(((input_dataframe - x) ** 2).sum(axis=1)))
    return pd.DataFrame(euclidean_distances.idxmin(axis=1))


def get_new_centroids(df_clustered_label,input_dataframe):
    df_original_label_join=input_dataframe.join(df_clustered_lab
    el)
    df_original_label_join.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    new_centroids=df_original_label_join.groupby('Cluster_Assign
    ed').mean()
    return new_centroids.T
```

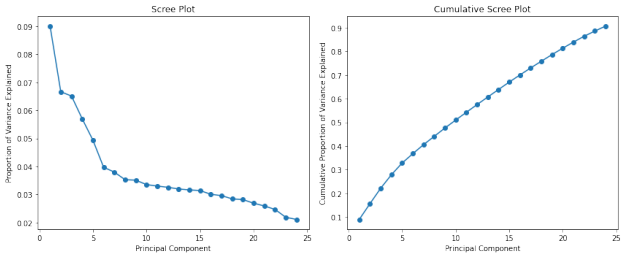
```
def kmeans_plus_plus(input_dataframe,no_of_clusters,threshold,no_of_iterations):
    start_time=time.time()
    iteration=0

    initial_centroid=kmeans_pp_init(input_dataframe,no_of_clusters)
    same_centroid=initial_centroid
    initial_centroid_column_list=initial_centroid.columns.to_list()

    while True:

        df_cluster_label=get_labels(input_dataframe,initial_centroid)
        df_new_centroids=get_new_centroids(df_cluster_label,input_dataframe)
        new_list_of_columns=df_new_centroids.columns.to_list()
        initial_set_columns = set(initial_centroid_column_list)
        new_set_columns = set(new_list_of_columns)
        missing_columns = initial_set_columns - new_set_columns
        for col in missing_columns:
            df_new_centroids[col]=initial_centroid[col]

        from scipy.spatial.distance import euclidean
        scalar_product =
        [euclidean(initial_centroid[col],df_new_centroids[col])
        for col in initial_centroid.columns]
        threshold_calculated=float(sum(scalar_product))/no_of_clusters

        iteration+=1

        if threshold_calculated<threshold:
            print("The input Threshold was
            {}".format(threshold))
            print("The calculated threshold is {}".format(threshold_calculated))

        if iteration>no_of_iterations:
            print("Limit for iterations has exceeded")

        if threshold_calculated<threshold or
        iteration>no_of_iterations:
            sum_of_square_error=sum_of_square_error_function(df_

            cluster_label,input_dataframe,df_new_centroids,no_of
            _clusters)
            df_cluster_label_copy=df_cluster_label.copy()

            df_cluster_label_copy.rename(columns=
            {0:'Cluster_Assigned'},inplace=True)
```

```
            labels=df_cluster_label_copy['Cluster_Assigned'].to_
            list()
            #silheoutte_score=silheoutte_score_Kmeans(input_data
320         frame,labels)
            silheoutte_score=0
            chs_score=Calinski_Harbaz_score_Kmeans(input_datafra
            me,labels)
            dbs_score=davies_bouldin_score(input_dataframe,label
325         s)
            end_time=time.time()
            return
            df_new_centroids,sum_of_square_error,silheoutte_scor
            e,chs_score,dbs_score,end_time-
330         start_time,same_centroid
            break
        else:
            initial_centroid= df_new_centroids


335
def
sum_of_square_error_function(df_cluster_label,input_dataframe,df
_new_centroids,no_of_clusters):
    df_data_label=input_dataframe.join(df_cluster_label)
340 df_data_label.rename(columns=
    {0:'Cluster_Assigned'},inplace=True)
    total_error=[]
    for cluster in range(no_of_clusters):
        df_data_label_cluster=df_data_label[df_data_label['Clust
345     er_Assigned']==cluster]
        df_data_label_cluster=df_data_label_cluster.drop('Cluste
        r_Assigned',axis=1)
        centroids=pd.DataFrame(df_new_centroids[cluster])
        euclidean_distance=cdist(df_data_label_cluster,centroids
350     .T,metric='euclidean')
        total_error.append(sum(euclidean_distance))
    return round(float(''.join(map(str, sum(total_error)))),3)

def silheoutte_score_Kmeans(input_dataframe,labels):
355 from sklearn.metrics import silhouette_score
    silhouette_avg = silhouette_score(input_dataframe, labels)
    return silhouette_avg

def Calinski_Harbaz_score_Kmeans(input_dataframe,labels):
360 from sklearn.metrics import calinski_harabasz_score
    chs=calinski_harabasz_score(input_dataframe,labels)
    return chs


365 def davies_bouldin_score(input_dataframe,labels):
    from sklearn.metrics import davies_bouldin_score
    dbs=davies_bouldin_score(input_dataframe,labels)
    return dbs
```

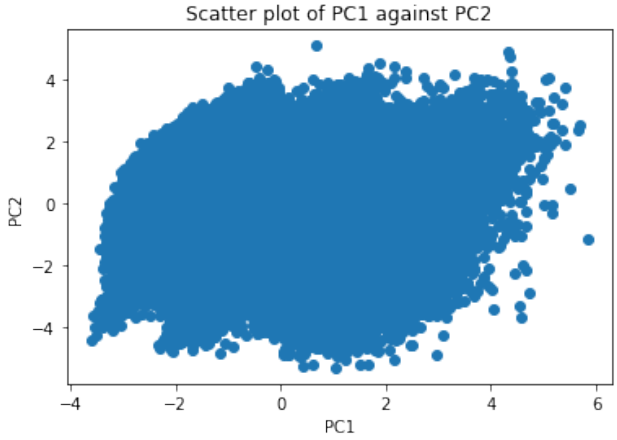
```
#Plotting graphs

import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_pca_df['alg
orithm']='GMM_PCA_K++'
error_values_kmeans_pca_df['algorithm']='K-Means++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_pca_df[['algorithm','No_of_Clusters','within_sum_o
f_square_error','Calinski_Harbaz_score','davies_bouldin_score']]
,
expectation_maximization_statistics_kmeans_plus_plus_df[['algori
thm','No_of_Clusters','within_sum_of_square_error','Calinski_Har
baz_score','davies_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='within_sum_of_square_error',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means++','GMM_PCA_K++'])],ax=ax);
plt.title('Box Plot of SSE for GMM_PCA and K means')
plt.show()

import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_pca_df['alg
orithm']='GMM_PCA_K++'
error_values_kmeans_pca_df['algorithm']='K-Means++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_pca_df[['algorithm','No_of_Clusters','within_sum_o
f_square_error','Calinski_Harbaz_score','davies_bouldin_score']]
,
expectation_maximization_statistics_kmeans_plus_plus_df[['algori
thm','No_of_Clusters','within_sum_of_square_error','Calinski_Har
baz_score','davies_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='davies_bouldin_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means++','GMM_PCA_K++'])],ax=ax);
plt.title('Box Plot of Davies Bouldin Score for GMM_PCA and K
means')
plt.show()

import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_pca_df['alg
orithm']='GMM_PCA_K++'
error_values_kmeans_pca_df['algorithm']='K-Means++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_pca_df[['algorithm','No_of_Clusters','within_sum_o
f_square_error','Calinski_Harbaz_score','davies_bouldin_score']]
```

```
,
expectation_maximization_statistics_kmeans_plus_plus_df[['algori
thm','No_of_Clusters','within_sum_of_square_error','Calinski_Har
baz_score','davies_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='Calinski_Harbaz_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin (['K-
Means++','GMM_PCA_K++'])],ax=ax);
plt.title('Box Plot of Calinski Harbaz Score for GMM_PCA and K
means')
plt.show()
```

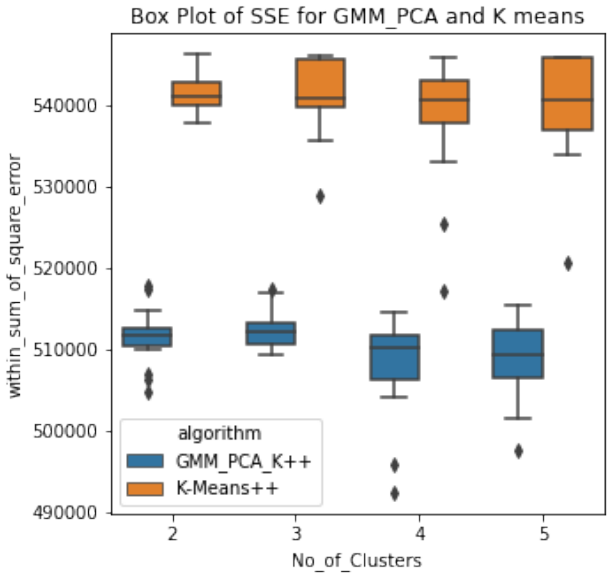
Plot/s



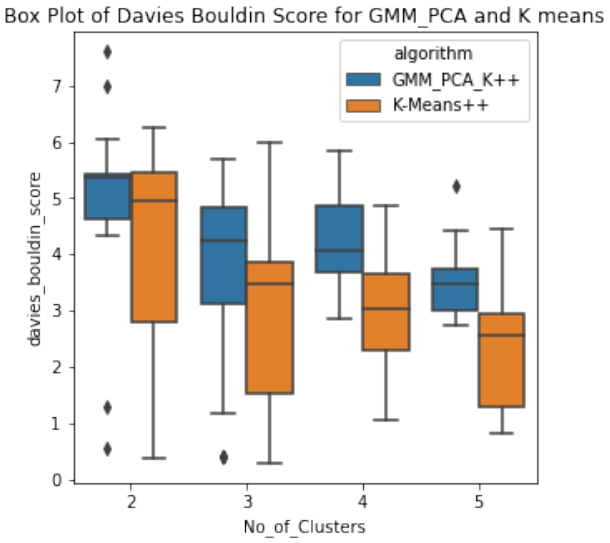
(11)



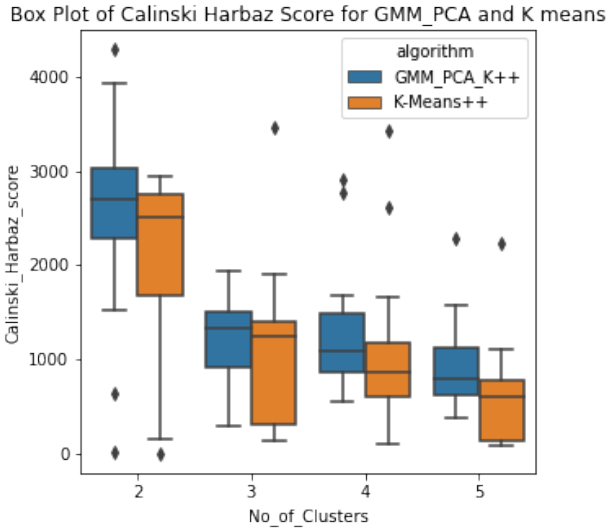
(12)


Box Plot of SSE for GMM_PCA and K means
within_sum_of_square_error
No_of_Clusters
algorithm
GMM_PCA_K++
K-Means++
540000
530000
520000
510000
500000
490000
2
3
4
5

(13)


Box Plot of Davies Bouldin Score for GMM_PCA and K means
davies_bouldin_score
No_of_Clusters
algorithm
GMM_PCA_K++
K-Means++
7
6
5
4
3
2
1
0
2
3
4
5

(14)



(15)

Discussion of Experiments

- Number of Principal components explain 90 percent of variance are 24
- We have used K means ++ Intialization to run the code for K means as well as expectation maximization and from the graphs we can see.
  - Gaussian Mixture models on reduced dataset has lower SSE as compared to K Means
  - Although the reduction in error is not significant we can see PCA helps in reduction of Error
  - The other indexes used are Calinski Harbaz score and Davies Bouldin Score, which we can see PCA has improved the cluster stability and cohesion.
  - **Why PCA helps in Gaussian Mixture Models**
  - PCA (Principal Component Analysis) is a technique used for dimensionality reduction by transforming a high-dimensional dataset into a lower-dimensional space while retaining most of the variability in the data.
  - The resulting transformed features (principal components) are linear combinations of the original features that are orthogonal to each other.
  - PCA does not directly estimate a normal distribution of the data, but it can help to approximate a normal distribution by reducing the effects of outliers and noise in the data, and by removing redundant and correlated features.
  - This is because PCA finds the directions of maximum variance in the data, which are often associated with the most informative and representative features.
  - By retaining the principal components that capture the most variance in the data, we can obtain a reduced-dimensional representation of the data that is less affected by outliers and noise and that highlights the most relevant patterns and structure in the data.

4. Run the EM algorithm for the other two different mixture models such as, Poisson. Compare all your three G_k's using two different appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results [30 points].

R or Python script

```
# Sample R Script With Highlighting
```

```
#Poisson Distribution
from scipy.stats import multivariate_normal
from scipy.special import gamma
import numpy as np
import math

def initialization_of_Poisson(input_dataframe,no_of_clusters):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means_vector =
    input_dataframe_values[np.random.choice(input_dataframe_valu
    es.shape[0], no_of_clusters,replace=False), :]
    weights_vector = np.ones(no_of_clusters)/no_of_clusters
    return means_vector,weights_vector

def
get_poisson(input_dataframe_values,means,weights,no_of_clusters)
:
    posterior_clusters=np.zeros(no_of_clusters)
    gamma_array=np.array([math.gamma(index+1) for index in
    input_dataframe_values])
    for cluster in range(no_of_clusters):

        temp=np.exp(-means[cluster])*np.power(means[cluster],inp
        ut_dataframe_values)/gamma_array
        posterior_clusters[cluster]=weights[cluster]*np.prod(tem
        p)+0.0001
    return posterior_clusters



def
fit_Poisson_mixture_models(input_dataframe,no_of_clusters,max_no
_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means,weights=initialization_of_Poisson(input_dataframe,no_o
    f_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:
        previous_means_df=pd.DataFrame(means)
        posterior_probabilities =
```

```
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for row_number in
        range(input_dataframe_values.shape[0]):
            posterior_probabilities[row_number]=get_poisson(inpu

            t_dataframe_values[row_number],means,weights,no_of_c
            lusters)
        posterior_probabilities=np.nan_to_num(posterior_probabil
        ities,nan=0)

        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            weights[j] = np.mean(posterior_probabilities[:, j])
        new_means_df=pd.DataFrame(means)
        euclidean_distance=[]
        for col in new_means_df.columns:
            col_distance = euclidean(previous_means_df[col],
            new_means_df[col])
            euclidean_distance.append(col_distance)
        threshold_calculated=sum(euclidean_distance)/no_of_clust
        ers
        iteration += 1
        if threshold_calculated<threshold:
            return means,posterior_probabilities
        if iteration>max_no_of_iterations:
            return means,posterior_probabilities
        iteration += 1

    return means,posterior_probabilities
#Running code multiple times
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_diabetes_scaled)
df_diabetes_scaled_min_max=scaler.transform(df_diabetes_scaled)
df_diabetes_scaled_min_max=pd.DataFrame(df_diabetes_scaled_min_max)
expectation_maximization_statistics_poisson=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_Poisson_mixture_models
        (df_diabetes_scaled_min_max,no_of_clusters,100,10)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
```

```
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:
            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            if len(indices)==0:
                indices=[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled_m
        in_max,cluster_labels_array,no_of_clusters)
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_dia
        betes_scaled_min_max,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled_min_ma
        x,cluster_labels_array)
        silheoutte_score_value=0
        expectation_maximization_statistics_poisson.append([no_o
        f_clusters,no_of_experiments,within_sum_of_square_error,
        silheoutte_score_value,Calinski_Harbaz_score_value,dbs_v
        value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_poisson_df=
pd.DataFrame(expectation_maximization_statistics_poisson,columns
=['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])

# Expnonential Distribution
from scipy.stats import multivariate_normal
from scipy.special import gamma
import numpy as np
import math

def initialization_of_exponential(input_dataframe,no_of_clusters):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means_vector =
    input_dataframe_values[np.random.choice(input_dataframe_valu
    es.shape[0], no_of_clusters,replace=False), :]
    weights_vector = np.ones(no_of_clusters)/no_of_clusters
    return means_vector,weights_vector

def get_exponential(input_dataframe_values,means,weights,no_of_clusters):
    posterior_clusters=np.zeros(no_of_clusters)
    for cluster in range(no_of_clusters):
        mean_temp=1/(means[cluster]+0.01)
```

```
        exponential=np.exp(-mean_temp*input_dataframe_values)/mean_temp
        posterior_clusters[cluster] = weights[cluster]*np.prod(exponential)+0.0001
    return posterior_clusters



def
fit_exponential_mixture_models(input_dataframe,no_of_clusters,ma
x_no_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means,weights=initialization_of_exponential(input_dataframe,
    no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:
        previous_means_df=pd.DataFrame(means)
        posterior_probabilities =
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for row_number in
        range(input_dataframe_values.shape[0]):
            posterior_probabilities[row_number]=get_exponential(
            input_dataframe_values[row_number],means,weights,no_
            of_clusters)
        posterior_probabilities=np.nan_to_num(posterior_probabil
        ities,nan=0)

        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            weights[j] = np.mean(posterior_probabilities[:, j])
        new_means_df=pd.DataFrame(means)
        euclidean_distance=[]
        for col in new_means_df.columns:
            col_distance = euclidean(previous_means_df[col],
            new_means_df[col])
            euclidean_distance.append(col_distance)
        threshold_calculated=sum(euclidean_distance)/no_of_clust
        ers
        iteration += 1
        if threshold_calculated<threshold:
            return means,posterior_probabilities
        if iteration>max_no_of_iterations:
            return means,posterior_probabilities
        iteration += 1

    return means,posterior_probabilities
from sklearn.preprocessing import MinMaxScaler
```

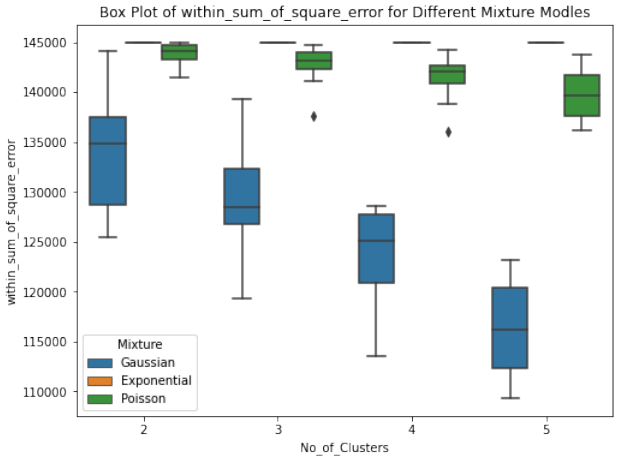
```
scaler=MinMaxScaler()
scaler.fit(df_diabetes_scaled)
df_diabetes_scaled_min_max=scaler.transform(df_diabetes_scaled)
df_diabetes_scaled_min_max=pd.DataFrame(df_diabetes_scaled_min_max)
expectation_maximization_statistics_exponential=[]
for no_of_clusters in range(2,6):
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_exponential_mixture_mo
        dels(df_diabetes_scaled_min_max,no_of_clusters,100,10)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:
            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            if len(indices)==0:
                indices=[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled_m
        in_max,cluster_labels_array,no_of_clusters)
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_dia
        betes_scaled_min_max,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled_min_ma
        x,cluster_labels_array)
        silheoutte_score_value=0
        expectation_maximization_statistics_exponential.append([
        no_of_clusters,no_of_experiments,within_sum_of_square_er
        ror,silheoutte_score_value,Calinski_Harbaz_score_value,d
        bs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_exponential_df=
pd.DataFrame(expectation_maximization_statistics_exponential,col
umns=['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])

#Running GMM on min maxed scaled dataset
from scipy.stats import multivariate_normal
import numpy as np
```

```
from scipy.spatial.distance import euclidean

def initialization_of_GMM(input_dataframe,no_of_clusters):
    '''
    The function takes scaled dataframe as input and
    initializes the GMM means,Covariances,and Weights
    '''
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    # Randomly initialize means vector
    means_vector =
    input_dataframe_values[np.random.choice(input_dataframe_valu
    es.shape[0], no_of_clusters, replace=False), :]
    # Initialize covariance matrices for each cluster
    covariances_vector = np.array([np.eye(column)] *
    no_of_clusters)
    # Initialize weights from uniform distribution
    weights_vector = np.ones(no_of_clusters) / no_of_clusters
    return means_vector,covariances_vector,weights_vector


def
fit_Guassian_mixture_models_scaled(input_dataframe,no_of_cluster
s,max_no_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    means,covariances,weights=initialization_of_GMM(input_datafr
    ame,no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:

        new_log_likelihood = 0
        for index in range(no_of_clusters):
            try:
                epsilon_weight=1e-6
                cov_inv = np.linalg.pinv(covariances[index] +
                np.diag(np.ones(covariances[index].shape[0]) *
                epsilon_weight))
                new_log_likelihood=new_log_likelihood+weights[in
                dex]*multivariate_normal.logpdf(input_dataframe_
                values,means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        new_log_likelihood_scalar=np.sum(new_log_likelihood)
        previous_means_df=pd.DataFrame(means)

        posterior_probabilities =
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for index in range(no_of_clusters):
            try:
                cov_inv = np.linalg.pinv(covariances[index],rcond=1e-10)
            except np.linalg.LinAlgError as e:
```

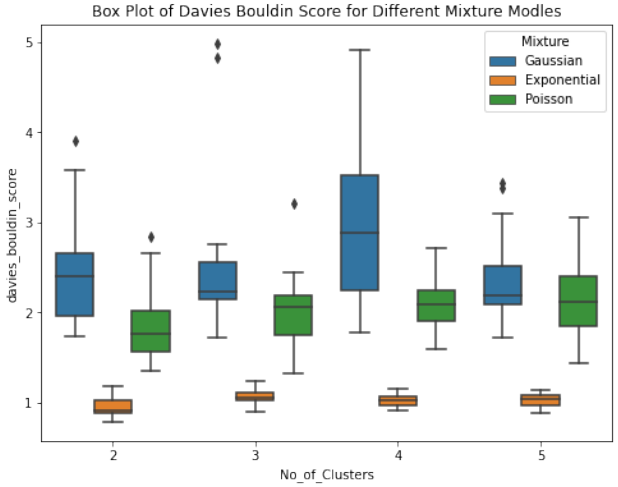
```
                continue
            try:
                posterior_probabilities[:,index] =
                weights[index] *
                multivariate_normal.pdf(input_dataframe_values,
                means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue




        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            difference = input_dataframe_values - means[j]
            covariances[j] = np.dot((difference *
            posterior_probabilities[:, j][:, np.newaxis]).T,
            difference) / np.sum(posterior_probabilities[:, j])
            covariances[j] += np.diag(np.ones(column) * 1e-6)
            weights[j] = np.mean(posterior_probabilities[:, j])

        new_means_df=pd.DataFrame(means)


        euclidean_distance=[]
        for col in new_means_df.columns:
            col_distance = euclidean(previous_means_df[col],
            new_means_df[col])
            euclidean_distance.append(col_distance)
        threshold_calculated=sum(euclidean_distance)/no_of_clust
        ers



        iteration += 1
        if threshold_calculated<threshold:
            return means,posterior_probabilities
        if iteration>max_no_of_iterations:
            return means,posterior_probabilities
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_diabetes_scaled)
df_diabetes_scaled_min_max=scaler.transform(df_diabetes_scaled)
df_diabetes_scaled_min_max=pd.DataFrame(df_diabetes_scaled_min_m
ax)
expectation_maximization_statistics_gaussian_min_max=[]
for no_of_clusters in range(2,6):
```

```
    print(no_of_clusters)
    for no_of_experiments in range(1,21):
        print(no_of_experiments)
        means,posterior_probabilities=fit_Guassian_mixture_model
        s_scaled(df_diabetes_scaled_min_max,no_of_clusters,100,1
        0)
        cluster_labels_original=np.array(pd.DataFrame(posterior_
        probabilities).idxmax(axis=1))
        cluster_labels_array=np.unique(np.array(pd.DataFrame(pos
        terior_probabilities).idxmax(axis=1)))
        list_of_clusters=np.array([i for i in
        range(0,no_of_clusters)])
        missing_clusters=set(list_of_clusters)-
        set(cluster_labels_array)
        for missing_value in missing_clusters:
            unique_values,value_counts=np.unique(cluster_labels_
            original,return_counts=True)
            values_to_replace=unique_values[value_counts > 1]
            value_to_replace=np.random.choice(values_to_replace)
            indices=np.where(cluster_labels_original==value_to_r
            eplace)[0]
            if len(indices)==0:
                indices=[0]
            random_index=np.random.choice(indices)
            new_value=missing_value
            cluster_labels_original[random_index]=new_value
        cluster_labels_array=cluster_labels_original
        within_sum_of_square_error=wcss_emm(df_diabetes_scaled_m
        in_max,cluster_labels_array,no_of_clusters)
        Calinski_Harbaz_score_value=Calinski_Harbaz_score(df_dia
        betes_scaled_min_max,cluster_labels_array)
        dbs_value=davies_bouldin_score(df_diabetes_scaled_min_ma
        x,cluster_labels_array)
        silheoutte_score_value=0
        expectation_maximization_statistics_gaussian_min_max.app
        end([no_of_clusters,no_of_experiments,within_sum_of_squa
        re_error,silheoutte_score_value,Calinski_Harbaz_score_va
        lue,dbs_value])
        print("Appended_to_dataframe")
expectation_maximization_statistics_gaussian_min_max_df=
pd.DataFrame(expectation_maximization_statistics_gaussian_min_ma
x,columns=['No_of_Clusters', 'Iteration Number',
'within_sum_of_square_error','silheoutte_score','Calinski_Harbaz
_score','davies_bouldin_score'])
```

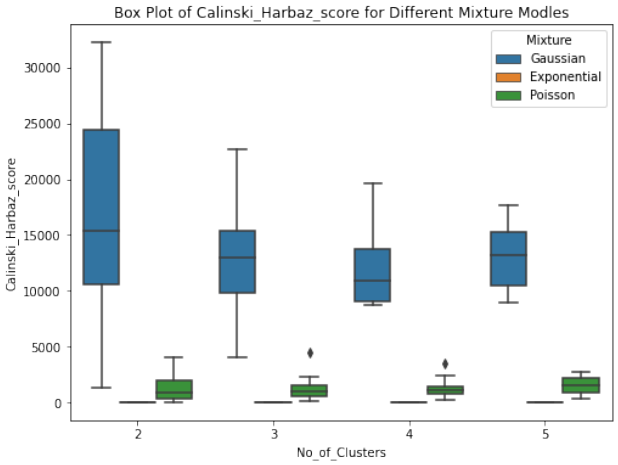
Plot/s



(16)



(17)



(18)

Discussion of Experiments

- Standardization of Data
  - In these experiments have performed min max scaler on the standardized dataset because Poisson distribution does not work with non negative numbers
- The formula used for the Distributions to calculate posterior probabilities are as follows:
- Poisson Distribution: P(X=k)=$e^{-\lambda}\lambda^k \frac{}{k!}$
- Exponential Distribution $f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$
- Normal Distribution: $f(x;\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$
  - Discussion of Graphs:
  - From the graphs we can see the Within Sum of Squared Errors median for every cluster is lower for Gaussian Distribution and goes higher in other types of distributions
  - Also the Calinski Harabaz score is better for Gaussian distribution so with Davies Bouldin Index.
  - **Why normal distribution would fit the data better?**
  - **Flexibility**:Normal distribution is a more flexible distribution than Poisson or exponential distribution. It can take on many different shapes, including symmetric, asymmetric, and bimodal. This makes it more suitable for modeling complex data sets that may exhibit multiple peaks or other non- unimodal patterns.
  - **Data EDA** After exploring the Dataset we can see the data is normally distributed with skewness and assumption of Normal is the best guess for our mixture model.
  - **Continous Variable**Normal distribution is a continuous distribution, while Poisson and exponential distributions are discrete and continuous, respectively. If the data being modeled is continuous, normal distribution may be a more appropriate choice than Poisson or exponential distribution.
  - **Central Limit Theorem**The central limit theorem states that the sum of a large number of independent and identically distributed random variables tends to follow a normal distribution, regardless of the distribution of the individual variables. This means that if the data being modeled is a sum of many independent random variables, normal distribution may provide a better approximation of the true distribution than Poisson or exponential distribution.
  - Hence from the above reasonings and the conclusions from the graph we can see why Gaussian Distribution is the best guess for our parametric clustering technique

Problem 3

Improve the EM algorithm through initialization. k-means ++ is an extended k-means clustering algorithm and induces non-uniform distributions over the data that serve as the initial centroids. Read the paper and implement this idea to improve your G_k program. Let's call the new algorithm G_{k++}. Run your new G_{k++} and G_k for $k = 2, \ldots, 5$ for 20 runs each. Compare G_k and C_k using two different appropriate cluster validity techniques, i.e., internal, external or relative indices. Plots are generally a good way to convey complex ideas quickly, i.e., box plots, whisker plots. Discuss your results.

R or Python script

```
# Sample R Script With Highlighting
```

```
# EM algorithm with K means ++ Initialization
from scipy.stats import multivariate_normal
import numpy as np

def kmeans_pp_init(input_dataframe,no_of_clusters):
    from scipy.spatial.distance import cdist
    '''
    K-means++ is a variant of the K-means algorithm that aims
    to improve the initial centroids' selection
    in the clustering process.
    The standard K-means algorithm initializes the cluster
    centroids randomly,
    which can lead to suboptimal clustering results,
    especially if the dataset has complex or irregular
    structures.
    '''
    list_of_centroids=[]
    #Choosing the first centroid randomly
    centroid = input_dataframe.apply(lambda x:
    float(x.sample()))
    list_of_centroids.append(centroid)

    iterator=2
    while iterator<=no_of_clusters:
        '''
        Calculating the distances from the centroid to every
        data point
        If the no of centroids are more than 1 calculate the
        distance from every centroid and take minimum distance
        '''
        distances =
        np.array(np.amin(cdist(input_dataframe,list_of_centroids
        ,metric='euclidean'),axis=1))
        #Next centroid will be selected with probability
        proportional to the distance

        probs = distances / np.sum(distances)
        '''
        Selection of the next centroids
        '''
        next_centroid =
        input_dataframe.iloc[np.random.choice(len(input_datafram
        e),p=probs)]
        list_of_centroids.append(next_centroid)
        iterator+=1

    centroid_df=pd.concat(list_of_centroids,axis=1,ignore_index=
    True)
    return centroid_df.T
```

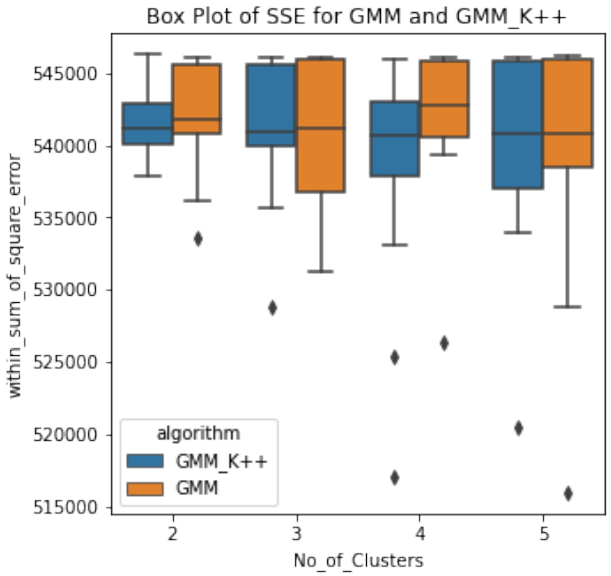
```
50
def
initialization_of_GMM_Kmeans(input_dataframe,no_of_clusters):
    '''
    The function takes scaled dataframe as input and
55  initializes the GMM means,Covariances,and Weights
    '''
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
    # Randomly initialize means vector
60  means_vector =
    np.array(kmeans_pp_init(input_dataframe,no_of_clusters))
    # Initialize covariance matrices for each cluster
    covariances_vector = np.array([np.eye(column)] *
    no_of_clusters)
65  # Initialize weights from uniform distribution
    weights_vector = np.ones(no_of_clusters) / no_of_clusters
    return means_vector,covariances_vector,weights_vector


70  def
fit_Guassian_mixture_models_kmeans_plus_plus(input_dataframe,no_
of_clusters,max_no_of_iterations,threshold):
    input_dataframe_values = input_dataframe.values
    row, column = input_dataframe_values.shape
75  means,covariances,weights=initialization_of_GMM_Kmeans(input
    _dataframe,no_of_clusters)
    iteration = 0
    previous_log_likelihood_scalar=0
    while iteration < max_no_of_iterations:
80

        new_log_likelihood = 0
        for index in range(no_of_clusters):
            try:
                epsilon_weight=1e-6
85              cov_inv = np.linalg.pinv(covariances[index] +
                np.diag(np.ones(covariances[index].shape[0]) *
                epsilon_weight))
                new_log_likelihood=new_log_likelihood+weights[in
                dex]*multivariate_normal.logpdf(input_dataframe_
90              values,means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        new_log_likelihood_scalar=np.sum(new_log_likelihood)

95      '''
        Calculating percentage change
        '''
        if np.abs(((np.abs(new_log_likelihood_scalar-
        previous_log_likelihood_scalar)/new_log_likelihood_scala
100     r)*100))<threshold:
            print("The input Threshold was
            {}".format(threshold))
```

```
            print("The calculated threshold is
            {}".format(np.abs(((np.abs(new_log_likelihood_scalar
            -
            previous_log_likelihood_scalar)/new_log_likelihood_s
            calar)*100))))
            break


        previous_log_likelihood_scalar=new_log_likelihood_scalar

        posterior_probabilities =
        np.zeros((len(input_dataframe_values),no_of_clusters))
        for index in range(no_of_clusters):
            try:
                cov_inv =
                np.linalg.pinv(covariances[index],rcond=1e-10)
            except np.linalg.LinAlgError as e:
                continue
            try:
                posterior_probabilities[:,index] =
                weights[index] *
                multivariate_normal.pdf(input_dataframe_values,
                means[index], cov_inv)
            except np.linalg.LinAlgError as e:
                continue
        posterior_probabilities/=np.sum(posterior_probabilities,
        axis=1, keepdims=True)



        for j in range(no_of_clusters):
            weighted_sum = np.zeros((1, means.shape[1]))
            sum_posterior = 0.0
            for i in range(row):
                weighted_sum += posterior_probabilities[i][j] *
                input_dataframe_values[i]
                sum_posterior += posterior_probabilities[i][j]
            means[j] = weighted_sum/sum_posterior
            difference = input_dataframe_values - means[j]
            covariances[j] = np.dot((difference *
            posterior_probabilities[:, j][:, np.newaxis]).T,
            difference) / np.sum(posterior_probabilities[:, j])
            covariances[j] += np.diag(np.ones(column) * 1e-6)
            weights[j] = np.mean(posterior_probabilities[:, j])



        iteration += 1

    return means,posterior_probabilities
#Code for Plotting Graphs
import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_df['algorit
```

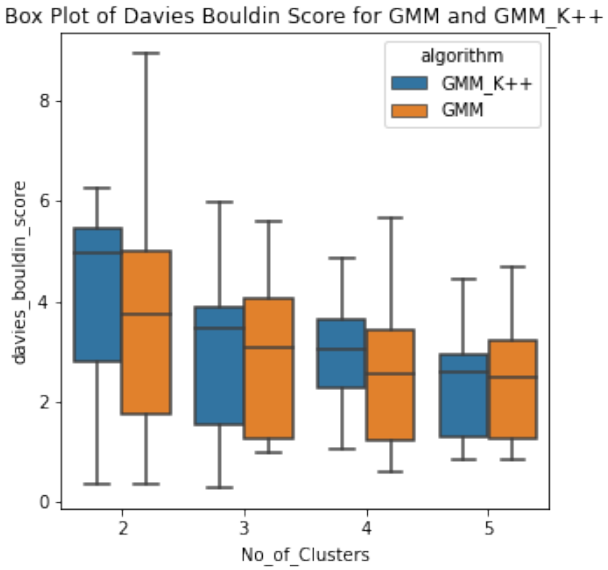
```
hm']='GMM_K++'
expectation_maximization_statistics_df['algorithm']='GMM'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_df[['algorithm','No_of_Clusters','within_sum_of_sq
uare_error','Calinski_Harbaz_score','davies_bouldin_score']],
expectation_maximization_statistics_df[['algorithm','No_of_Clust
ers','within_sum_of_square_error','Calinski_Harbaz_score','davie
s_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='Calinski_Harbaz_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin
(['GMM_K++','GMM'])],ax=ax);
plt.title('Box Plot of Calinski Harbaz Score for GMM and
GMM_K++')
plt.show()

#Comparing k++ and Gmm++
import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_df['algorit
hm']='GMM_K++'
error_values_kmeans_plus_plus_alone_df['algorithm']='K++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_df[['algorithm','No_of_Clusters','within_sum_of_sq
uare_error','Calinski_Harbaz_score','davies_bouldin_score']],
error_values_kmeans_plus_plus_alone_df[['algorithm','No_of_Clust
ers','within_sum_of_square_error','Calinski_Harbaz_score','davie
s_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='Calinski_Harbaz_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin
(['GMM_K++','K++'])],ax=ax);
plt.title('Box Plot of Calinski Harbaz Score for K++ and GMM_K++')
plt.show()

import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_df['algorit
hm']='GMM_K++'
error_values_kmeans_plus_plus_alone_df['algorithm']='K++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_df[['algorithm','No_of_Clusters','within_sum_of_sq
uare_error','Calinski_Harbaz_score','davies_bouldin_score']],
error_values_kmeans_plus_plus_alone_df[['algorithm','No_of_Clust
ers','within_sum_of_square_error','Calinski_Harbaz_score','davie
s_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
```

```
sns.boxplot(x='No_of_Clusters', y='within_sum_of_square_error',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin
(['GMM_K++','K++'])],ax=ax);
plt.title('Box Plot of Within Sum of Square Error for K++ and
GMM_K++')
plt.show()

import seaborn as sns
expectation_maximization_statistics_kmeans_plus_plus_df['algorit
hm']='GMM_K++'
error_values_kmeans_plus_plus_alone_df['algorithm']='K++'
comparison_df=pd.DataFrame()
comparison_df=pd.concat([expectation_maximization_statistics_kme
ans_plus_plus_df[['algorithm','No_of_Clusters','within_sum_of_sq
uare_error','Calinski_Harbaz_score','davies_bouldin_score']],
error_values_kmeans_plus_plus_alone_df[['algorithm','No_of_Clust
ers','within_sum_of_square_error','Calinski_Harbaz_score','davie
s_bouldin_score']]
],ignore_index=True )
fig, ax = plt.subplots(figsize=(5,5))
sns.boxplot(x='No_of_Clusters', y='davies_bouldin_score',
hue='algorithm',
data=comparison_df[comparison_df['algorithm'].isin
(['GMM_K++','K++'])],ax=ax);
plt.title('Box Plot of davies bouldin score for K++ and
GMM_K++')
plt.show()
```

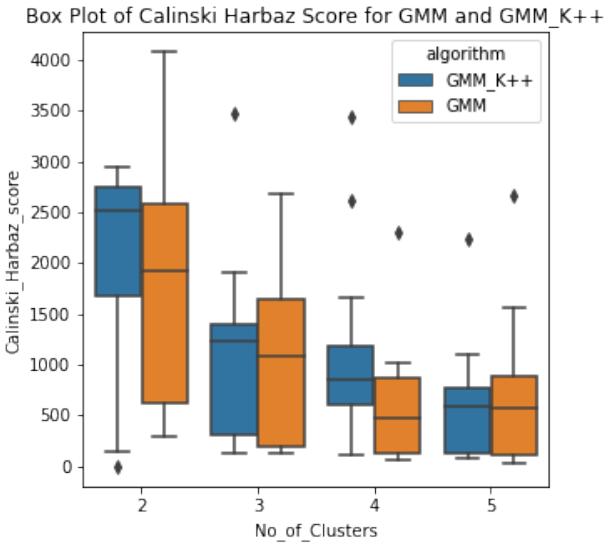
Plot/s



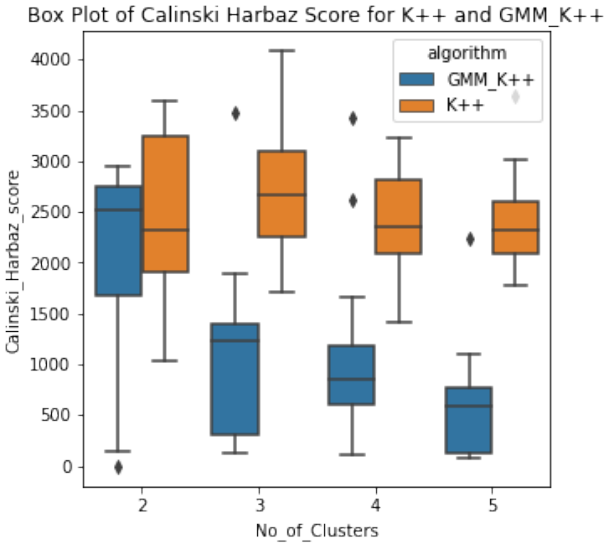
(19)


Box Plot of Davies Bouldin Score for GMM and GMM_K++
algorithm
GMM_K++
GMM
davies_bouldin_score
0
2
4
6
8
2
3
4
5
No_of_Clusters

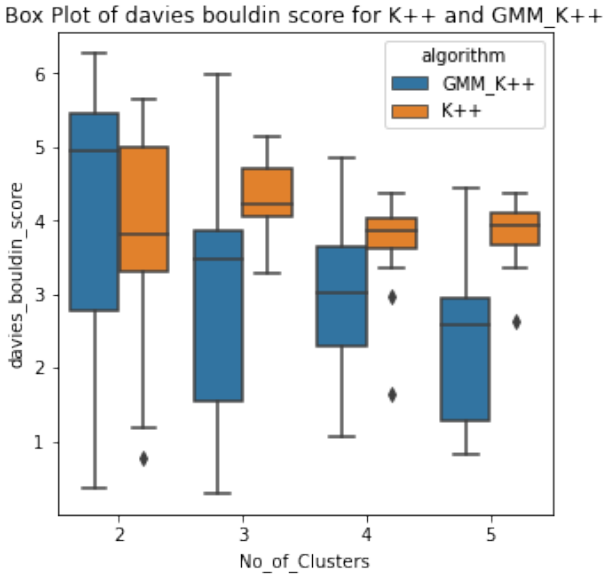
(20)


Box Plot of Calinski Harbaz Score for GMM and GMM_K++
algorithm
GMM_K++
GMM
Calinski_Harbaz_score
0
500
1000
1500
2000
2500
3000
3500
4000
2
3
4
5
No_of_Clusters

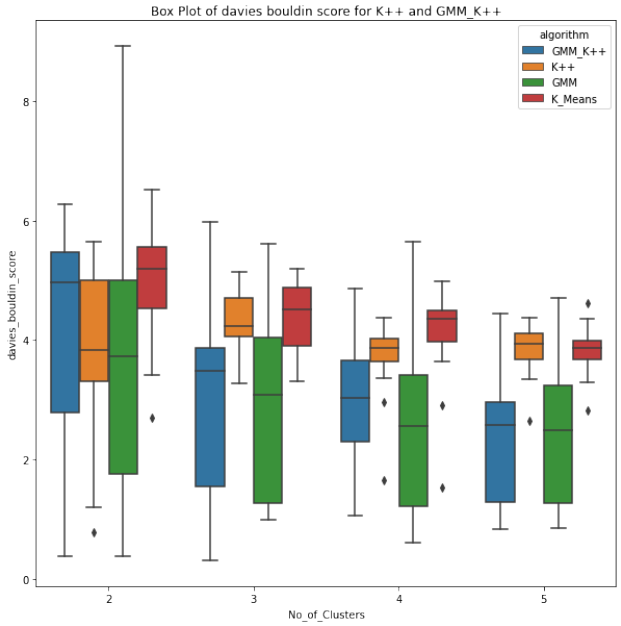
(21)


Box Plot of Calinski Harbaz Score for K++ and GMM_K++
algorithm
GMM_K++
K++
Calinski_Harbaz_score
No_of_Clusters
0
500
1000
1500
2000
2500
3000
3500
4000
2
3
4
5

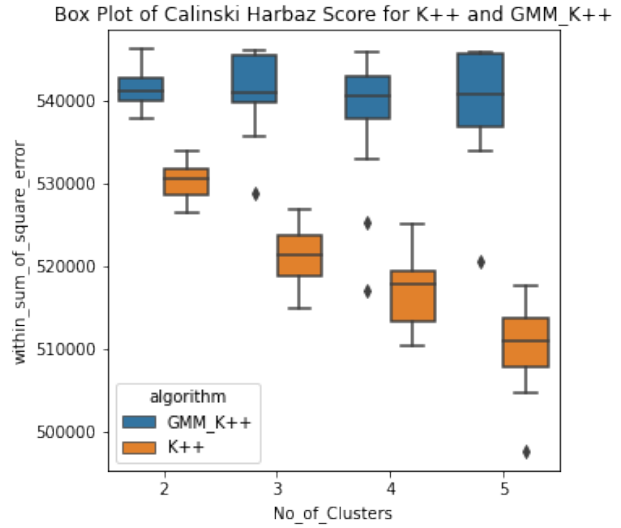
(22)


Box Plot of davies bouldin score for K++ and GMM_K++
algorithm
GMM_K++
K++
davies_bouldin_score
No_of_Clusters
1
2
3
4
5
6
2
3
4
5

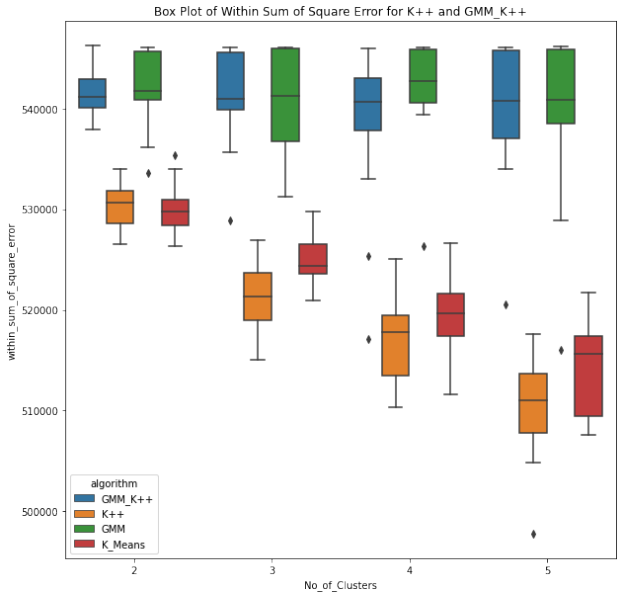
(23)


Box Plot of davies bouldin score for K++ and GMM_K++
algorithm
GMM_K++
K++
GMM
K_Means
davies_bouldin_score
No_of_Clusters
0
2
4
6
8
2
3
4
5

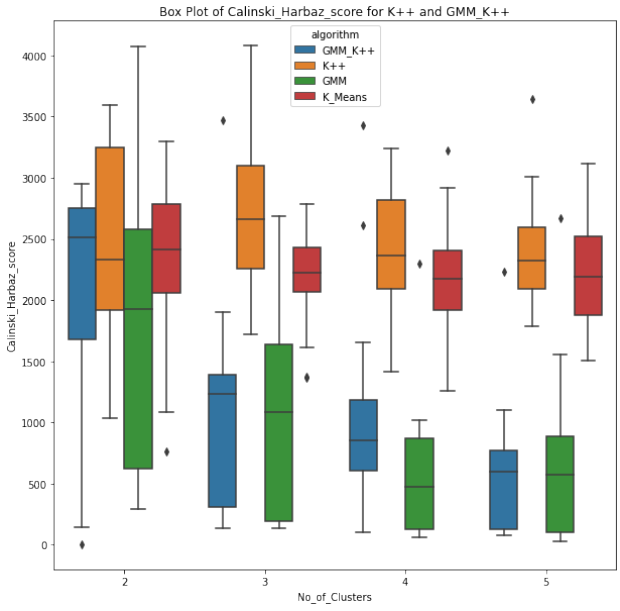
(24)


Box Plot of Calinski Harbaz Score for K++ and GMM_K++
within_sum_of_square_error
No_of_Clusters
algorithm
GMM_K++
K++
500000
510000
520000
530000
540000
2
3
4
5

(25)



(26)



(27)

Discussion of Experiments

- I have ran experiments for comparing Gaussian Mixture Models and Gaussian Mixture models with K means ++ intialization.
- Other comparison is between K means ++ and Gaussian Mixture Models with K means plus plus initialization
- The code for K means and K means ++ is from assignment 04
- Final Comparison is between all the four algorithms mentioned above.

- **GMM and GMM++**
- From the initial three graphs we can see that when we initialize Gaussian Mixture Models with K means plus plus initalization the Within Sum of Square Errors reduces although the reduction is not significant.
- However we can conclude that better initialization of means has a good effect on clustering algorithm.
- **K++ and GMM++**
- From graphs 4,5,6 we can see K means ++ performs better as it has lower within sum of square errors and higher Calinski Harabaz score.
- The probable reason why K means with better initialization works well could be:
- When the clusters in the data are well-separated and clearly distinct, it is easier for K-means to accurately assign each data point to its nearest centroid, resulting in a lower WSS. In contrast, GMM may have more difficulty accurately assigning data points to clusters if the clusters are overlapping or have complex structures, which can lead to a higher WSS.
- Well defined cluster separation is visible from Calinski Harabaz score and davies bouldin score
- Calinski-Harabasz Score (CHS) is a clustering evaluation metric used to measure the quality of clustering solutions. It calculates the ratio of the between-cluster variance to the within-cluster variance. The CHS is higher when the clusters are well separated and the within-cluster variance is small, and lower when the clusters are overlapping and the within-cluster variance is large.
- Davies-Bouldin Score (DBS) is a clustering evaluation metric used to measure the similarity between clusters. It calculates the average similarity between each cluster and its most similar cluster, based on the ratio of within-cluster and between-cluster distances. A lower DBS indicates better clustering solutions with higher intra-cluster similarity and lower inter-cluster similarity.
- Thus by the definition we can see that since davies bouldin score of K means is less than that of GMM it is showing better results.
- However choice for clustering depends on other factors as well which we need to consider
- **Comparison of All four methods**
- From graphs 7,8,9 we can see that K means ++ is a better method as compared to others.
- However Gaussian Mixture model when initialized with K means plus plus also gives better Davies Bouldin Score as well as Calinski Harabaz Score as compared to K means ++
- Thus we can conclude even though K means ++ reduces the within sum of square errors, GMM with K means plus plus initialization have better cluster definition.
- Thus the final selection of model also depends on other factors but K means ++ initialization with PCA can serve as a better recipe

Submission

You must use LaTeX to turn in your assignments. Please submit the following two files via Canvas:

1. A .pdf with the name `yourname-hw6-everything.pdf` which you will get after compiling your .tex file.
2. A .zip file with the name `yourname-hw6.zip` which should contain your .tex, .pdf, codes(.py, .ipynb, .R, or .Rmd), and a README file. The README file should contain information about dependencies and how to run your codes.