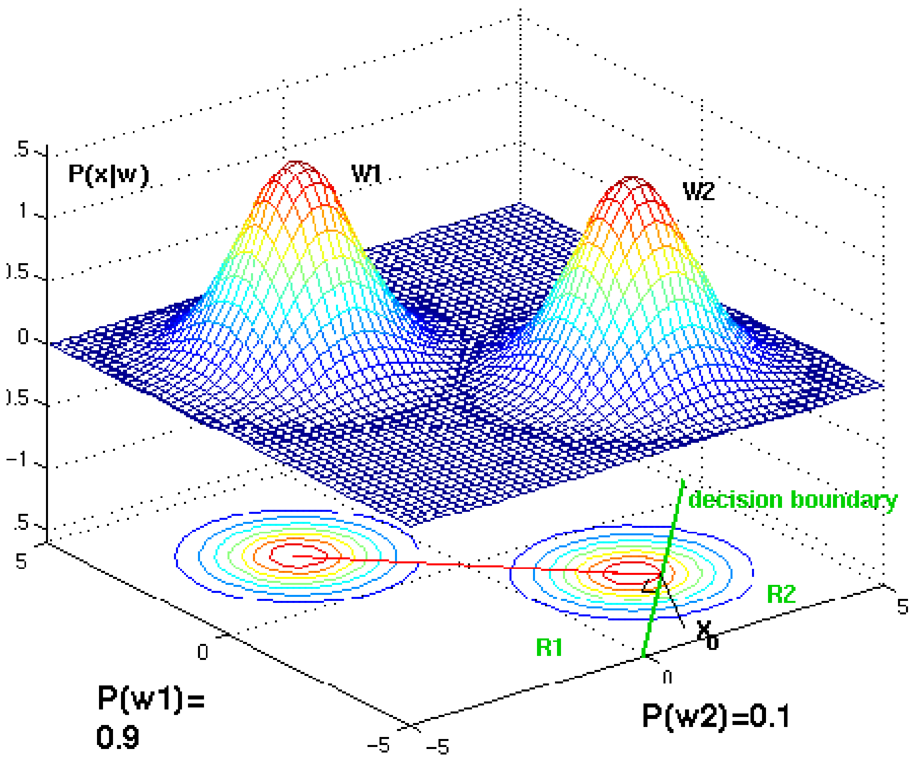
Clustering Techniques

K-MEANS
Mixture of Gaussians

Methods in Bioinformatics

Kyriakis Dimitris



Contents

1 Introduction

1.1 K-Means

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means is one of the simplest unsupervised learning algorithms. The aim of the algorithm is to classify a given data set through a certain number of clusters (assume k clusters). The main idea is to define k centroids, one for each cluster. These centers should be placed randomly because of different location causes different result. The better choice is to place them as much as possible far away from each other. So the next step is to take each point of a given data set and associate it to the nearest center and recalculate the centroids. When no point is left, the first step is completed and our early clusters are constructed. After we have these k new centroids, a new binding has to be done again between the same data and the nearest new centroid. As a result of this loop we may notice that the k centers change their location step by step until centers do not move any more or in other words there is a converge.

Disadvantages:

- The learning algorithm requires a priori specification of the number of cluster centers.
- Unable to handle noisy data and outliers.
- Algorithm fails for non-linear data set.

Distance Metrics:

- **Euclidean:** appropriate when we have continuous numerical variables and I want to reflect absolute distances. This distance takes into account every variable and doesn't remove redundancies, so if I had three variables that explain the same (are correlated), I would weight this effect by three. Moreover, this distance is not scale invariant, so generally I have to scale previously to use the distance.

$$d(x,y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2}$$

- **Manhattan:** is appropriate when we have continuous numerical variables and I want to reflect absolute distances, but we want to remove redundancies. If we have repeated variables, their repetitious effect will disappear.

$$d(x,y) = \sum_{i=1}^{D} \|x_i - y_i\|$$

- **Mahalanobis:** first transforms the variables into uncorrelated variables with variances equal to 1, and then calculates simple Euclidean distance.

$$d(x,y) = \sqrt{(x_i - y_i)^T S^{-1}(x - y)}$$

If the data have similar mean values, would indicate distances such as Manhattan and Euclidean, while in the other hand would indicate correlation distance. If you know the covariance structure of your data then Mahalanobis distance is more appropriate.

1.2 Mixture of Gaussians

Our aim is to find clusters within our data and fit them to Gaussians Distributions. Fitting a Gaussian to data does not guarantee that the resulting Gaussian will be an accurate distribution for the data. To define a Gaussian, we need to specify just two parameters:

- **μ:** which is the mean (average) of the distribution.
- **σ:** which is the standard deviation of the distribution.

We supposed to have a Gaussian Distributions for every cluster we want to construct. This technique is probabilistic, based on latent (hidden) variables z. For every data point x_n here is a corresponding latent variable z_k, which a particular element z_k is equal to 1 and all other elements are equal to 0. The values of z_k therefore satisfy $z_k \in (0, 1)$ and $\sum_k z_k = 1$. Z is the posterior probability (or responsibility). We shall define the joint distribution p(x, z) in terms of a marginal distribution p(z) and a conditional distribution $p(x \mid z)$. The marginal distribution over z is specified in terms of the mixing coefficients k, such that:

$$p(z_k = 1) = \pi_k$$

Each π_k is a weight, specifying the relative importance of Gaussian Ni in the mixture.

- Weights π_k are $0 \leq \pi_k \leq 1$.
- Weights π_k must $\sum_{k=1}^{K} \pi_k = 1$.

By definition, the marginal distribution of x is given by summing the joint distribution over all possible z and is a Gaussian mixture model:

$$p(x) = \sum_{k=1}^{K} \pi_k N(x \mid \mu_k, \Sigma_k)(1)$$

Another quantity that will play an important role is the conditional probability of z given x. We shall use $\gamma(z_k)$ to denote $p(z_k = 1 \mid x)$, whose value can be found using Bayes' theorem:

$$\gamma(z_k) = \frac{\pi_k N(x \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x \mid \mu_j, \Sigma_j)}$$

We shall view π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed x. However, we do not know which mixture each x_n belongs to. If we knew μ_k and σ_k for each N_k, we could probabilistically assign each x_n to a component. So if we could maximize the parameters of our model, we could find the Gaussians that our data fit.
An elegant and powerful method for finding maximum likelihood solutions for models with latent variables is called the expectation-maximization algorithm, or EM algorithm (Dempster et al., 1977; McLachlan and Krishnan, 1997). The E- step is to evaluate the responsibilities using the above parameter. The M-step is the next step where we are estimate the parameters using : Setting the derivatives of $\ln p(X \mid \pi, \mu,)$ with respect to the means μ_k of the Gaussian components to zero, we obtain:

$$\mu_k^{new} = \tfrac{1}{N_k} \textstyle\sum_{n=1}^{N} \gamma(z_{nk}) x_n, \; N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \text{ and } \pi_k^{new} = \tfrac{N_k}{N}$$

We can interpret Nk as the effective number of points assigned to cluster k. Now, if we set the derivative of log-likelihood with respect to Sk and setting to zero, making use of the result for the maximum likelihood solution for the covariance matrix of a single Gaussian:

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

Finally we evaluate the the log likelihood in order to maximize (1), and check for convergence:

$$\ln p(X \mid \pi, \mu,) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_j N(x \mid \mu_j, \Sigma_j\}$$

1.3 Shilouettte Score

After cluster analysis,it is crucial to evaluate the "goodness" of the resulting cluster, in order to avoid finding patterns in noise and compare clustering with other algorithms. Numerical measures that are applied to judge various aspects of cluster validity.

- **External Index:** Used to measure the extent to which cluster labels match externally supplied class labels.
- **Internal Index:**Used to measure the goodness of a clustering structure without respect to external information. Result is evaluated based on the data that was clustered itself. Sum of Squared Error (SSE)
- **Relative Index:** Used to compare two different clustering or clusters.

The internal validation that used in this project was the silhouette coefficient. The silhouette coefficient contrasts the average distance to elements in the same cluster with the average distance to elements in other clusters. Objects with a high silhouette value are considered well clustered, objects with a low value may be outliers. This index works well with k-means clustering, and is also used to determine the optimal number of clusters.

$$S(i) = \frac{b(i) - a(i)}{max\{a(i, b(i)\}}, -1 \leq s(i) \leq 1$$

a(i) is the average dissimilarity (distance) of datum i with all other data within the same cluster (the smaller the value, the better the assignment), b(i) be the lowest average dissimilarity of i to any other cluster.

- s(i) = 1 the datum is appropriately clustered.
- $s(i) = -1$ the datum should be clustered in its neighbouring cluster.
- s(i) = 0 The datum is on the border of two natural cluster.

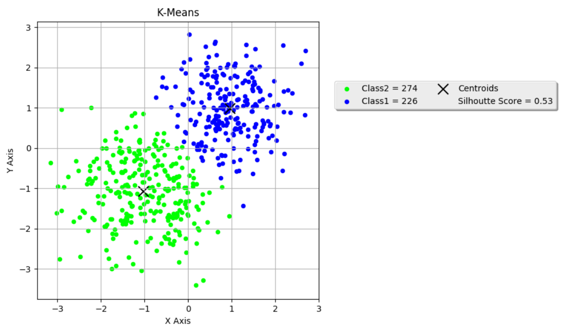
2 Analysis

2.1 Theoretical Problem

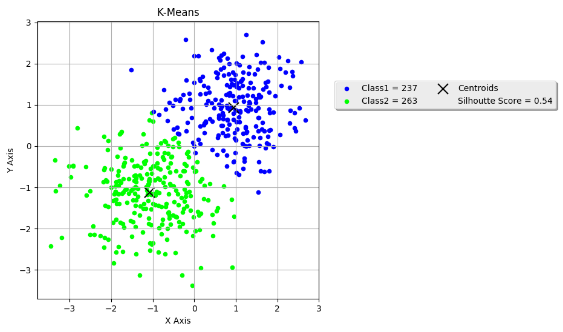
Create a data set of random, normally distributed data with 2 dimensions (D) and 500 observations (N). The first 220 observations are generated from a multivariate gaussian with mean m=[1 1] and S = 0.5 I, while the last 280 observations come from a gaussian with m=[-1 -1] and S = 0.75 I.

```
Size_1 = 220; Size_2 = 280
D =2
mean_1 =[1,1] ; cov_1 = 0.5*np.eye(D)
mean_2 = [-1,-1] ; cov_2 = 0.75*np.eye(D)
X1 = np.random.multivariate_normal(mean_1, cov_1, size=Size_1).T
X2 = np.random.multivariate_normal(mean_2, cov_2, size=Size_2).T
X = np.concatenate((X1,X2),axis=1)
```

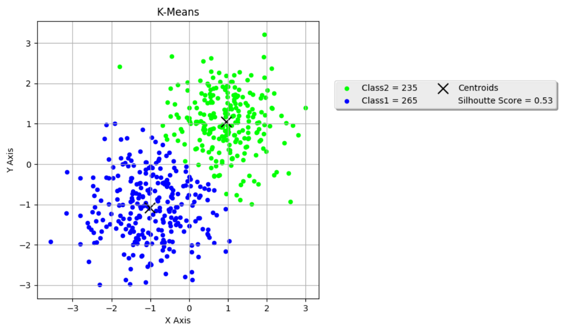
Cluster = 2



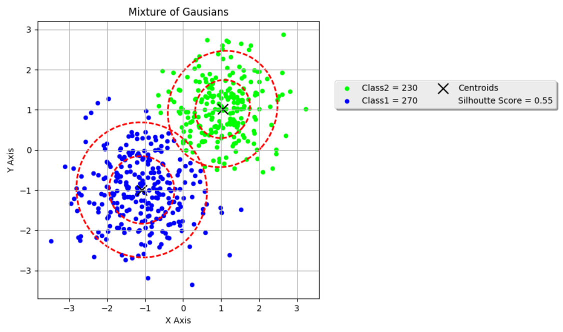
(a) Euclidean



(b) Manhattan



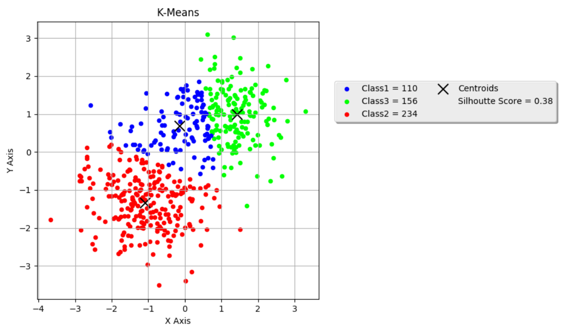
(c) Mahalanobis



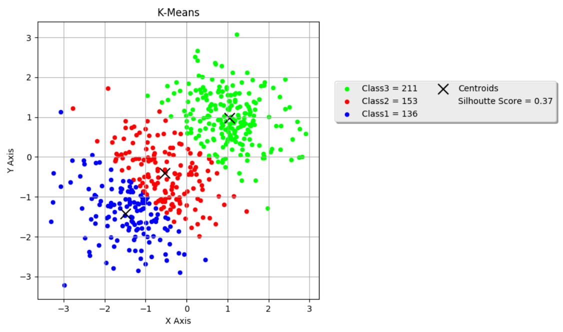
(d) Mixture of Gaussians

Figure 1: Random Data grouped in 2 clusters, with 5 repetitions and maxiters =30. A) Euclidean Distance with Silhouette score = 0.54 and time = 1.46sec. B) Manhattan Distance with Silhouette score = 0.53 and time = 1.298sec. C) Mahalanobis Distance with Silhouette score = 0.53 and time = 1.54sec. D) Mixture of Gaussian Silhouette score = 0.55 and time = 23sec.

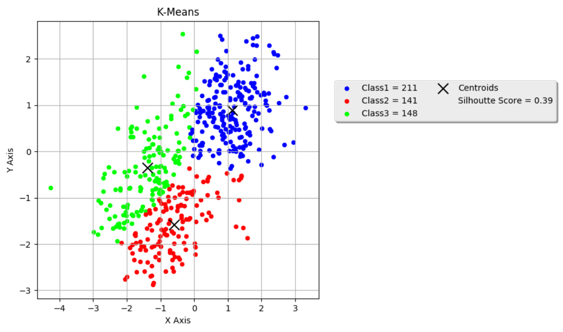
Cluster = 3



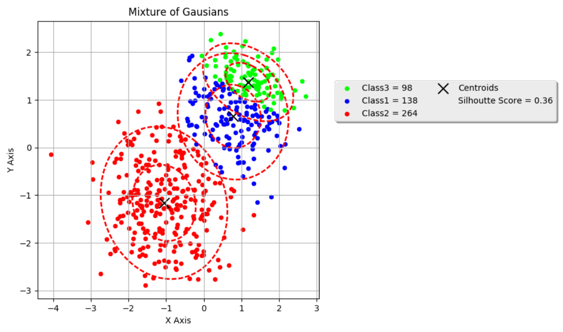
(a) Euclidean



(b) Manhattan



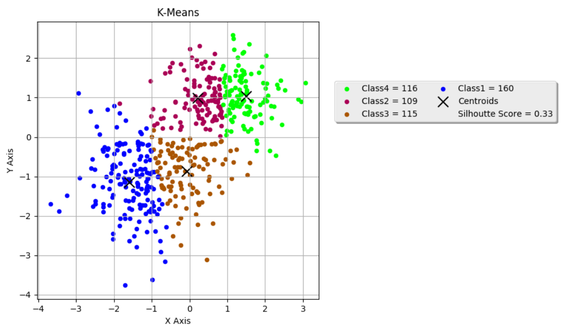
(c) Mahalanobis



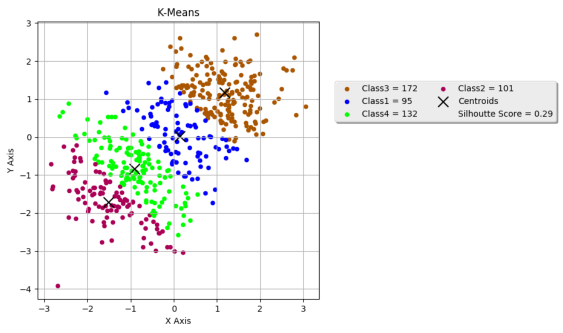
(d) Mixture of Gaussians

Figure 2: Random Data grouped in 3 clusters, with 5 repetitions and maxiters =30. A) Euclidean Distance with Silhouette score = 0.38 and time = 3.39sec. B) Manhattan Distance with Silhouette score = 0.37 and time = 4.24sec. C) Mahalanobis Distance with Silhouette score = 0.39 and time = 1.147sec. D) Mixture of Gaussian Silhouette score = 0.42 and time = 26.8sec.

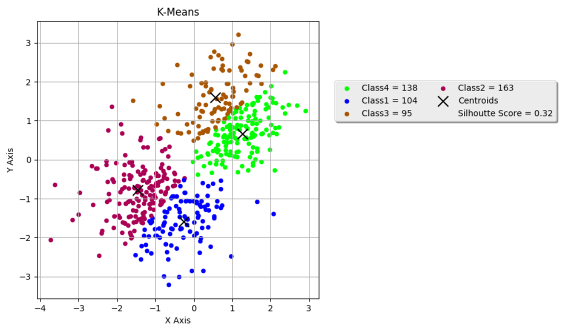
Cluster = 4



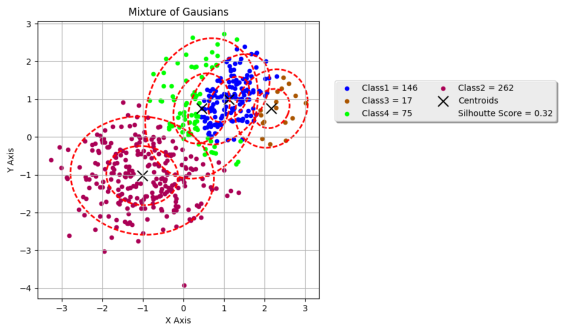
(a) Euclidean



(b) Manhattan



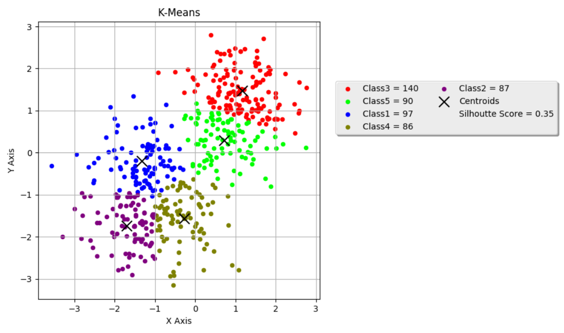
(c) Mahalanobis



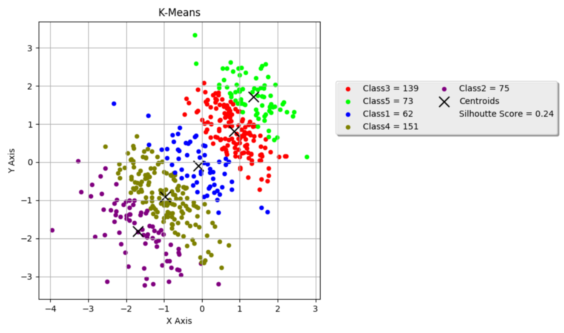
(d) Mixture of Gaussians

Figure 3: Random Data grouped in 4 clusters, with 5 repetitions and maxiters =30. A) Euclidean Distance with Silhouette score = 0.33 and time = 5.39sec. B) Manhattan Distance with Silhouette score = 0.29 and time = 5.24sec. C) Mahalanobis Distance with Silhouette score = 0.32 and time = 9.47sec. D) Mixture of Gaussian Silhouette score = 0.32 and time = 30.9 sec.

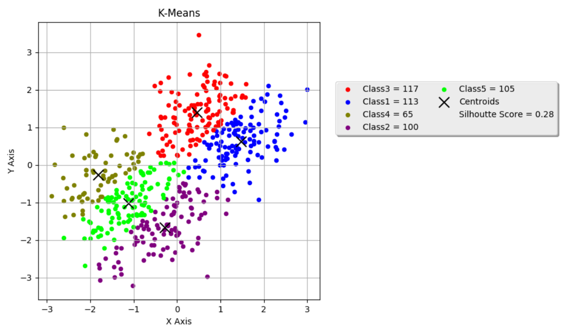
Cluster = 5



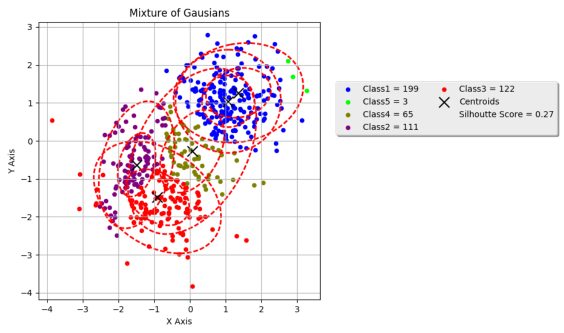
(a) Euclidean



(b) Manhattan



(c) Mahalanobis



(d) Mixture of Gaussians

Figure 4: Random Data grouped in 5 clusters, with 5 repetitions and maxiters =30. A) Euclidean Distance with Silhouette score = 0.35 and time = 7.39sec. B) Manhattan Distance with Silhouette score = 0.28 and time = 8.42sec. C) Mahalanobis Distance with Silhouette score = 0.24 and time = 9.47sec. D) Mixture of Gaussian Silhouette score = 0.27 and time = 39.7 sec.

From the charts above we can conclude to the figure 5 about the time and silhoutte score for the different clustering methods.

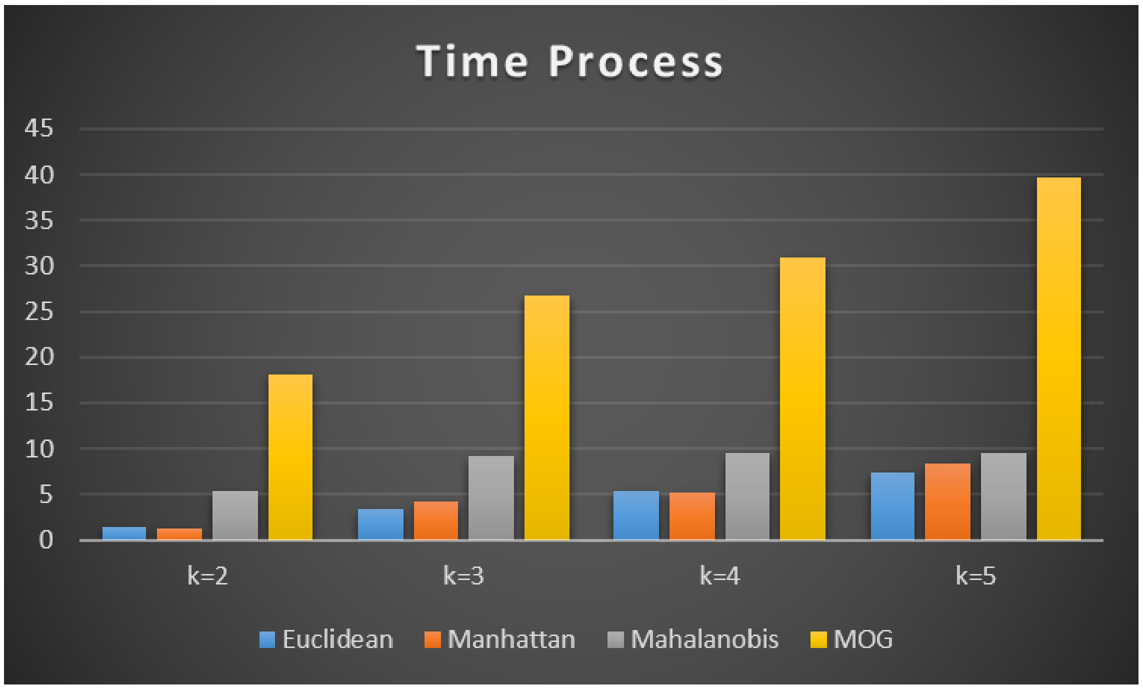


Figure 5: Speed comparison between the K-means(Euclidean, Manhattan, Mahalanobis) and Mixture of Gaussian

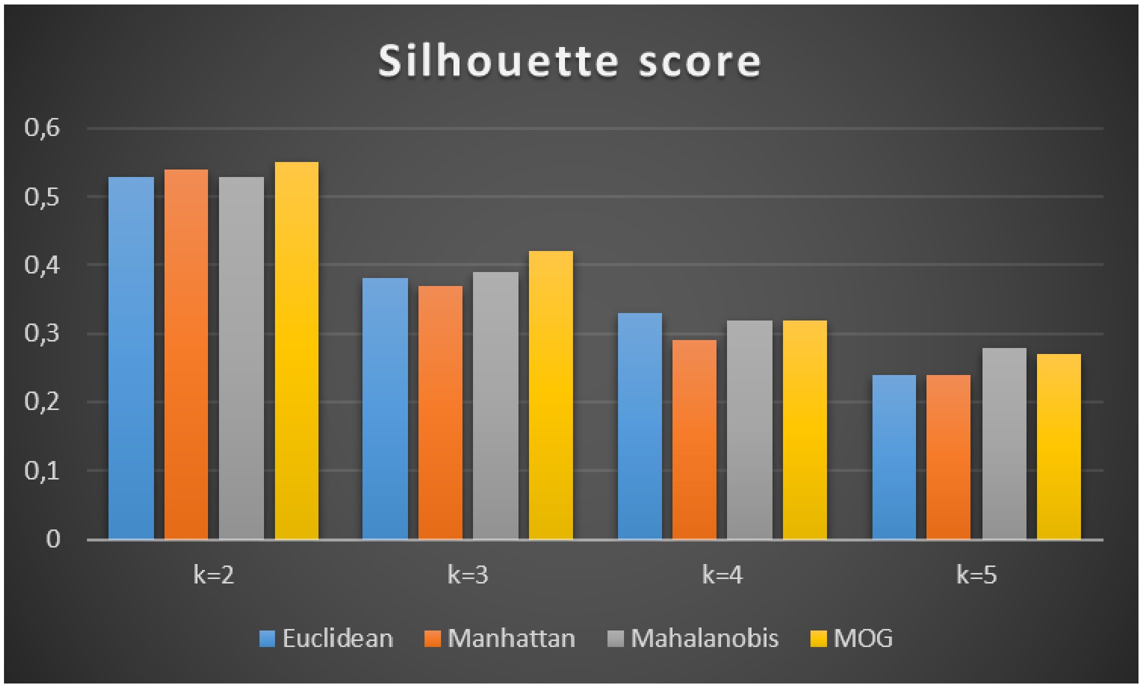


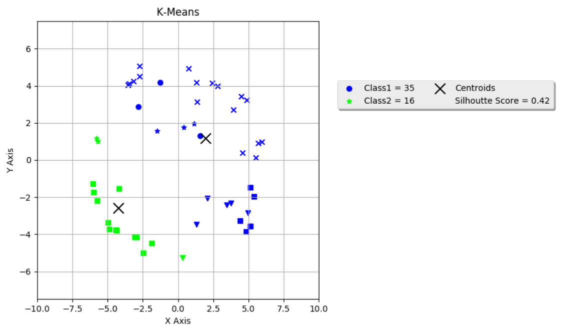
Figure 6: Silhoutte score comparison between the K-means(Euclidean, Manhattan, Mahalanobis) and Mixture of Gaussian

As we can see, the mixture of Gaussian seems more computationally complicated and demanding more time than kmeans. The score is quite the same between the two algorithms. In our data set the best choice is to cluster them in two groups.The silhouette score decreasing significantly while the number of clusters increasing.

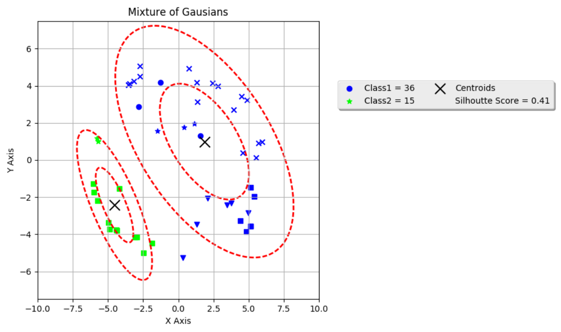
2.2 Practical Problem

The aim of this exercise is to perform clustering in a real data set. The analysis of livers of C57BL/6J mice fed a high fat diet for up to 24 weeks has shown significant body weight gain was observed after 4 weeks. The results provide insight into the effect of high-fat diets on metabolism in the liver. In order to reduce the dimension of our data set we used Kernel PCA with Hyperbolic kernel ($\delta = 4$). Then we used K-means to initialize the centroids. Then the outputted centroids used as input to Mixture of Gaussian clustering algorithm. As we can see below the K-means not always outputted the same results. And the most of the cases are false grouped.
If we consider the distribution of our data after Kernel PCA we can see that the data can be clustered in two ways. In the first one, a horizontal line can separate our data. Maybe in this case another clustering algorithm is more suitable like SVM which use the margin line. In the second case the data can clustered according to the distance from the centroids. Some times the results

K-Means with two Clusters



(a) K-means



(b) MOG

Figure 7: Kmeans with 2 clusters. The shapes are according to the original labels,where the controls interpreted as dots, the mice with normal diet as "X" and asterisk (for 2-4 weeks and for 6-8 weeks respectively) and the mice with high fat diet as triangles and square (for 2-4 weeks and for 6-8 weeks respectively

In Figure 7, the high fat diet mice seems to make a cluster together and all the other mice a second group.

K-Means with three Clusters

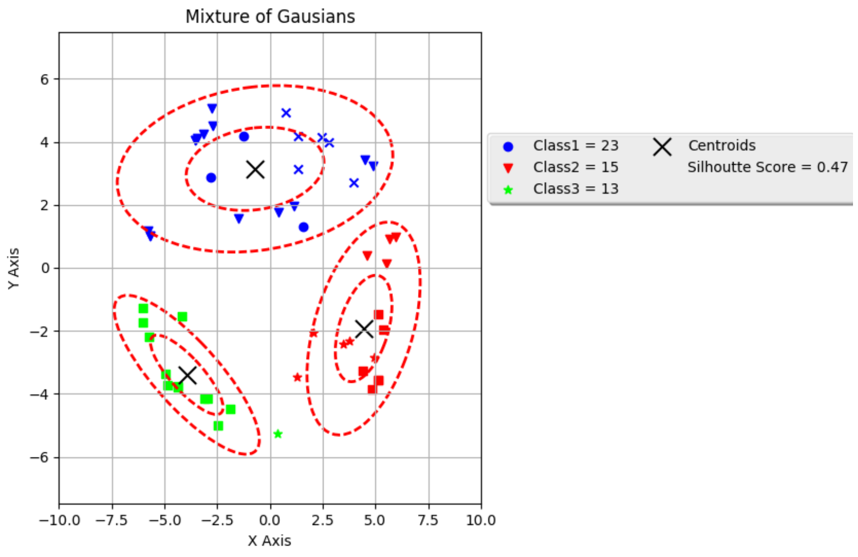


Figure 8: MOG with 3 clusters. The shapes are according to the original labels,where the controls interpreted as dots, the mice with normal diet as "X" and triangles (for 2-4 weeks and for 6-8 weeks respectively) and the mice with high fat diet as asterisk and square (for 2-4 weeks and for 6-8 weeks respectively

K-Means with two Clusters

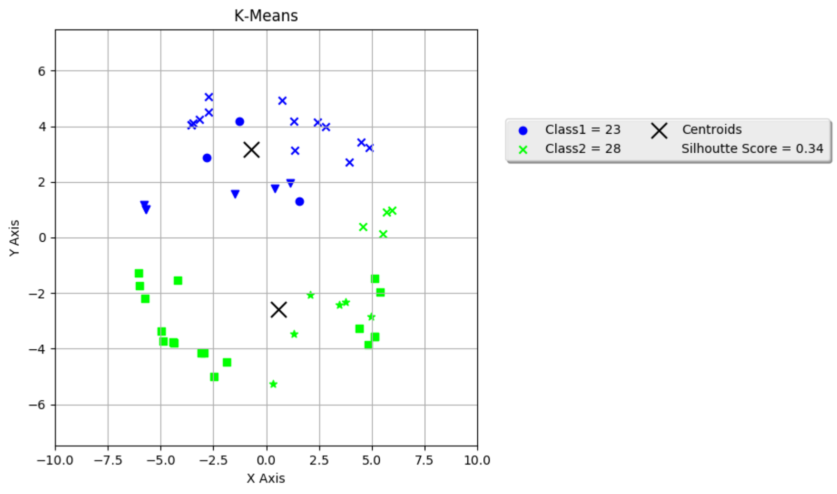


Figure 9: Kmeans with 2 clusters. The shapes are according to the original labels,where the controls interpreted as dots, the mice with normal diet as triangles and the mice with high fat diet as x

MOG with two Clusters

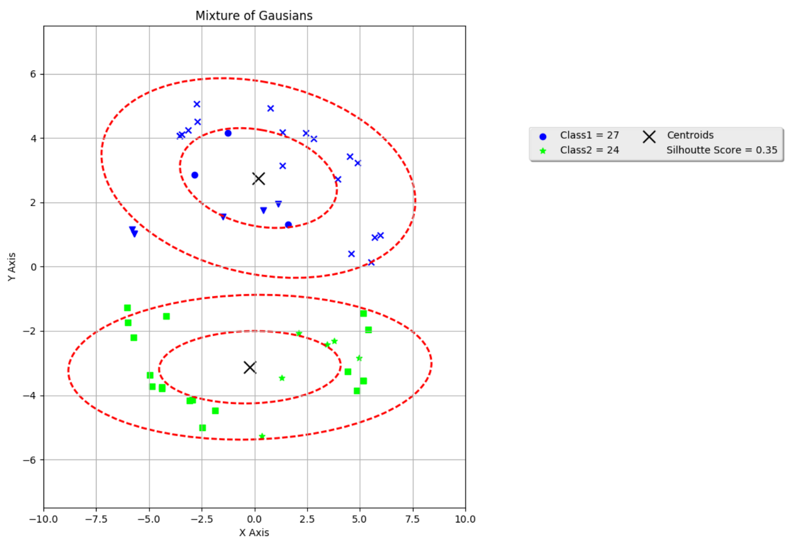


Figure 10: Mixture of Gaussianss with 2 clusters. The shapes are according to the original labels,where the controls interpreted as dots, the mice with normal diet as "X" and triangles (for 2-4 weeks and for 6-8 weeks respectively) and the mice with high fat diet as asterisk and square (for 2-4 weeks and for 6-8 weeks respectively).

K means seems to miss grouped 4 elements but when the outputted centroids used as input to MOG the 4 points was correctly placed in the same cluster, as the original labels underline.

3 Bibliography

References

[1] Christopher M. Bishop *Pattern Recognition and Machine Learning.*

[2] Tapas Kanungo, Senior Member, IEEE, David M. Mount, Member, IEEE, Nathan S. Netanyahu, Member, IEEE, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, Senior Member, IEEE *An Efficient k-Means Clustering Algorithm: Analysis and Implementation.*. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE , 27:7, 2002.

4 Appendix

File: kmeans.py

```
def kmeans(X,k,Distance,maxiters,reps):
    import scipy.spatial.distance
    ################  NORMALIZARION ####################
    X, mean_matrix, max_min = Normalization(X)
    D = X.shape[0]
    N = X.shape[1]
    S = np.cov(X)
    names = ["C"+str(x) for x in range(k)]
    e = 1e-10
    ################  Distance Measures ######################
    Eucl_func =  lambda x,y: np.linalg.norm(x.reshape(D,1)-y.reshape(D,1))
    Manha_func = lambda x,y: abs(np.sum(x.reshape(D,1)-y.reshape(D,1)))
    Mahala_func =  lambda x,y:
        np.sqrt((x-y).reshape(1,D).dot(np.linalg.inv(S)).dot((x.reshape(D,1)-y.reshape(D,1)).reshape(D,1)))[0][0]
    if Distance =="Euclidean":
        Dist_func = Eucl_func
    elif Distance =="Manhatan":
        Dist_func = Manha_func
    elif Distance =="Mahalanobis":
        S = np.cov(X)
        Dist_func = Mahala_func

    ##########################   Repetition   ################################

    for r in range(reps):
        ####### RANDOM Centrioids #######
        indx = np.random.randint(X.shape[1],size=k)
        Centr_old = X[:,indx]
        diff = 10
        max_iters = maxiters

        #######################  Convergence ################################
        while diff > e:
            ################### INITIALIZE ##################################
            max_iters -= 1
            Cluster_Dic = dict.fromkeys(names,np.zeros((D,1)))
            Cluster_Num = dict.fromkeys(names,0)
            Centr_new = np.zeros((D,k))
            labels =[]
            #################  PARSING VECTROS ##############################
            for i in range(N):
                ################### DISTANCE FROM CENTROIDS ###################
                dist=[]
                for c in range(Centr_old.shape[1]):
                    Metric_distance = Dist_func(Centr_old[:,c],X[:,i])
                    dist.append(Metric_distance)
                clash = "C"+str(dist.index(min(dist)))
                clash_num = dist.index(min(dist))
                labels.append(clash_num)
                ################### ADD VECTROR TO CLUSTER ####################
                if Cluster_Num[clash] == 0:
                    Cluster_Num[clash] += 1
                    Cluster_Dic[clash] = X[:,i].reshape(D,1)
                else:
                    Cluster_Dic[clash] =
                        np.concatenate((Cluster_Dic[clash],X[:,i].reshape(D,1)),axis=1)
                #################E  RE CALCULATE CENTROID #####################
                new = np.average(Cluster_Dic[clash],axis=1)
                Centr_new[:,clash_num] = new
            ##########  DISTANCE BETWEEN OLD AND NEW CENTROIDS ################
            diff = 0
            for cluster in range(k):
                diff += Dist_func(Centr_new[:,cluster],Centr_old[:,cluster])
            sys.stdout.write("\rDIFF_W: {:.2e}\tRepetition: {}".format(diff,r+1))
            sys.stdout.flush()
            Centr_old = np.copy(Centr_new)
            if max_iters == 0:
                break
        #################### DISTANCE OLD VS NEW CLUSTER ####################
        ################ KEEP CLUSTER WITH THE SMALLER DISTANCE ###############
        dist_C_new = 0
        for cl in Cluster_Dic.keys():
            dist_metric = {"Euclidean":'euclidean',"Manhatan":'cityblock',
                 "Mahalanobis":'mahalanobis'}
            if Distance == "Mahalanobis":
                dist_C_new += scipy.spatial.distance.pdist(Cluster_Dic[cl],dist_metric[Distance], VI=
                     np.linalg.inv(S))[0]
            else:
                dist_C_new += scipy.spatial.distance.pdist(Cluster_Dic[cl],dist_metric[Distance])[0]
        if r == 0:
             Centroids = np.copy(Centr_new)
             dist_C_old = dist_C_new
             cl_labels = labels
        else:
            if dist_C_new < dist_C_old:
                Centroids = np.copy(Centr_new)
                dist_C_old = dist_C_new
                cl_labels = labels
    print("\nCluster Distance {:.5}".format(dist_C_new))
    Centroids  = (Centroids*max_min[:,:k])+mean_matrix[:,:k]
    return(Centroids,cl_labels)
```

File2: Mixture of Gaussians.py

```
def MOG(X,k,reps,MEANS,maxiters):
    import scipy.spatial.distance
    N =X.shape[1]
    D = X.shape[0]
    names = ["C"+str(x) for x in range(k)]
    ############# NORMALIZATION ################
#    X, mean_matrix, max_min = Normalization(X)
    for rep in range(reps):
        #==================================================================#
        ######################### INITIALIZE ################################
        #==================================================================#
        ##  CREATE MEANS ##

        try:
            Mean_Matrix_old = MEANS.copy()
        except:
            indx = np.random.randint(X.shape[1],size=k)
            Mean_Matrix_old = X[:,indx] # DXK

        S_dic_Matrx_old ={}
        for q in names:
            S_dic_Matrx_old[q] = np.eye(D,D)#+np.random.uniform(D,D)
        ## CREATE      ##
         k_old  = (np.random.dirichlet(np.ones(k),size=1)[0])

        ## ZERO GAMMA ##
        Diff =1
        e = 1e-7
        Log_old = -mt.inf
        silhouette_avg_old = 0
        #==================================================================#
        ###################### CHECK CONVERGE ###############################
        #==================================================================#
        maxit = maxiters
        while Diff > e:
            maxit -=1
            Mean_Matrix_new= np.zeros((D,k))
            S_dic_Matrix_new = {}
            for q in names:
                S_dic_Matrix_new[q] = np.zeros((D,D))
             k_new  = np.zeros((1,k))[0]
            Gamma_Matrix = np.zeros((k,N))
            Marginal_Matrix = np.zeros((1,N))[0]
            labels = []

            #==============================================================#
            ####################### E-STEP ##################################
            #==============================================================#
            for n in range(N):
                #==========================================================#
                ################# CALCULATE MARGINAL ##########################
                #==============================================================#
                ###################### Responsibility ##########################
                ####### Probability that xi was generated by Gaussian Nk. #####
                #==============================================================#

                for cluster in range(k):
                    name = "C{}".format(cluster)
                    pdf_Xn =
                          k_old  [cluster]*pdf(X[:,n],Mean_Matrix_old[:,cluster],S_dic_Matrx_old[name])
                    Gamma_Matrix[cluster,n] = pdf_Xn
                    Marginal_Matrix[n] += pdf_Xn
            Marginal_Matrix_A = np.repeat(Marginal_Matrix.reshape(1,N),k,axis=0)
            Gamma_Matrix = Gamma_Matrix/Marginal_Matrix_A
            #==================================================================#
            ########################## M-STEP ################################
            #################### Mnew S_new    _new    ############################
            #==================================================================#
            labels = np.argmax(Gamma_Matrix, axis=0)
            Nk = np.sum(Gamma_Matrix,axis=1)
            N_all = np.sum(Nk)
            for n in range(N):
                for cluster in range(k):
                    Mean_Matrix_new[:,cluster] += (Gamma_Matrix[cluster,n])*X[:,n]
            for cluster in range(k):
                Mean_Matrix_new[:,cluster]= Mean_Matrix_new[:,cluster]/Nk[cluster]
            for n in range(N):
                for cluster in range(k):
                    name = "C{}".format(cluster)
                    Xtil = (X[:,n]-Mean_Matrix_new[:,cluster])
                    S_dic_Matrix_new[name] +=
                        (Gamma_Matrix[cluster,n])*Xtil.reshape(D,1).dot(Xtil.reshape(1,D))
            for cluster in range(k):
                name = "C{}".format(cluster)
                 k_new  [cluster] = Nk[cluster]/N_all
                S_dic_Matrix_new[name]= S_dic_Matrix_new[name]/Nk[cluster]

            #==================================================================#
            ########################### EVALUATE #############################
            #==================================================================#
            Log_new = np.sum(np.log(Marginal_Matrix))
            Diff = abs(Log_new - Log_old)
            sys.stdout.write("\rDIFF_W: {:.2e}\t Repetition : {}".format(Diff,rep+1))
            sys.stdout.flush()
            Log_old = Log_new
            #RE-ASSIGN OLDS ##
            Mean_Matrix_old = Mean_Matrix_new.copy()
            S_dic_Matrx_old = S_dic_Matrix_new.copy()
             k_old   =   k_new  .copy()
            if maxit == 0:
                break
        #======================================================================#
        #########################  RE CALCULATE Gnk ##########################
        #======================================================================#
        Marginal_Matrix = np.zeros((1,N))
        for n in range(N):
            for cluster in range(k):
                name = "C{}".format(cluster)
                pdf_Xn =  k_old  [cluster]*pdf(X[:,n],Mean_Matrix_old[:,cluster],S_dic_Matrx_old[name])
                Gamma_Matrix[cluster,n] = pdf_Xn
                Marginal_Matrix[0][n] += pdf_Xn
        Marginal_Matrix_A = np.repeat(Marginal_Matrix,k,axis=0)
        Gamma_Matrix = Gamma_Matrix/Marginal_Matrix_A
#        Marginal_Matrix_A = np.repeat(Marginal_Matrix,k,axis=0)
#        Gamma_Matrix = Gamma_Matrix/Marginal_Matrix_A
        labels = np.argmax(Gamma_Matrix, axis=0)
        Cluster_DIC ={}
        for cluster in range(k):
            Cluster_DIC[cluster] = X[:,labels==cluster]

        dist_C_new = 0
        Distance="Euclidean"
        for cl in Cluster_DIC.keys():
            dist_metric = {"Euclidean":'euclidean',"Manhatan":'cityblock',
                 "Mahalanobis":'mahalanobis'}
            if Distance == "Mahalanobis":
                dist_C_new += scipy.spatial.distance.pdist(Cluster_DIC[cl],dist_metric[Distance], VI=
                    np.linalg.inv(S))[0]
            else:
                dist_C_new += scipy.spatial.distance.pdist(Cluster_DIC[cl],dist_metric[Distance])[0]
        if rep == 0:
             Centroids = Mean_Matrix_new.copy()
             S_OR = S_dic_Matrix_new.copy()
             dist_C_old = dist_C_new
             cl_labels = np.ndarray.tolist(labels)
             Gamma_Matrix_F = Gamma_Matrix.copy()
        else:
            if dist_C_new < dist_C_old:
                Centroids = Mean_Matrix_new.copy()
                S_OR = S_dic_Matrix_new.copy()
                dist_C_old = dist_C_new
                cl_labels = np.ndarray.tolist(labels)
                Gamma_Matrix_F = Gamma_Matrix.copy()
    #==========================================================================#
    ################### CONVER MEANS TO CENTROID MATRIX #####################
    #==========================================================================#
#    Centroids =
     np.concatenate((Mean_Matrix_new[:,0].reshape(2,1),Mean_Matrix_new[:,1].reshape(2,1)),axis=1)
    #### RE SCALE CENTROIDS ###
#    Centroids  = (Centroids*max_min[:,:k])+mean_matrix[:,:k]
    return Centroids, cl_labels,S_OR,Gamma_Matrix_F
```