# FMAN45 Machine Learning, spring 2020
# Assignment 2

Salma Kazemi Rashed

May 2020

## 1 Solving a nonlinear kernel SVM with hard constraints.

**Task 1**: Compute the kernel matrix $\boldsymbol{K}$ using the data from the table.

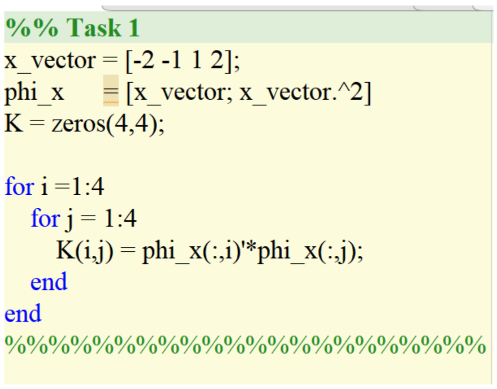
| i | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $x_i$ | -2 | -1 | 1 | 2 |
| $y_i$ | +1 | -1 | -1 | +1 |

Table 1: One-dimensional binary classification problem with classes '+1' and '-1':

$$\phi(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix} \tag{1}$$

The calculated kernel matrix equals:

$$\boldsymbol{K} = \begin{pmatrix} 20 & 6 & 2 & 12 \\ 6 & 2 & 0 & 2 \\ 2 & 0 & 2 & 6 \\ 12 & 2 & 6 & 20 \end{pmatrix} \tag{2}$$



```
%% Task 1
x_vector = [-2 -1 1 2];
phi_x    = [x_vector; x_vector.^2]
K = zeros(4,4);

for i =1:4
   for j = 1:4
      K(i,j) = phi_x(:,i)'*phi_x(:,j);
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 1: Kernel matrix calculation.

**Task 2**: Solve the maximization problem in Eq. (5) for $\alpha$ numerically, using the data in table 1. You may use, without a proof that, for the data in table 1, the solution satisfies $\alpha = \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$.

$$\underset{\alpha_1,\ldots,\alpha_4}{\text{maximize}} \quad \sum_{i=1}^{4} \alpha_i - \frac{1}{2} \sum_{i=1}^{4} \sum_{j=1}^{4} \alpha_i \alpha_j y_i y_j \boldsymbol{K}(x_i, x_j) \tag{3}$$

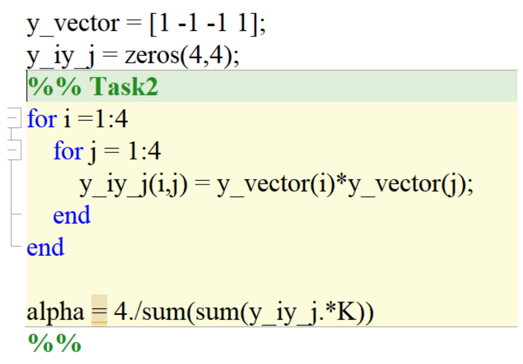
By using $\alpha = \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$, Eq. (5) will be simplified as :

$$\underset{\alpha}{\text{maximize}} \quad 4\alpha - \frac{1}{2} \sum_{i=1}^{4} \sum_{j=1}^{4} \alpha^2 y_i y_j \boldsymbol{K}(x_i, x_j) \tag{4}$$

Thus, $\alpha$ would be

$$\alpha = \frac{4}{\sum_{i=1}^{4} \sum_{j=1}^{4} y_i y_j \boldsymbol{K}(x_i, x_j)} \tag{5}$$

By using data from table (1), we will have :



```
y_vector = [1 -1 -1 1];
y_iy_j = zeros(4,4);
%% Task2
for i =1:4
    for j = 1:4
        y_iy_j(i,j) = y_vector(i)*y_vector(j);
    end
end

alpha = 4./sum(sum(y_iy_j.*K))
%%
```

Figure 2: $\alpha$ calculation.

and the optimal $\alpha$ equals $\alpha = 0.1111$. Besides, by sweeping $\alpha = 0 : 0.4$, the optimal value for $\alpha$ will be achieved as 0.1111.

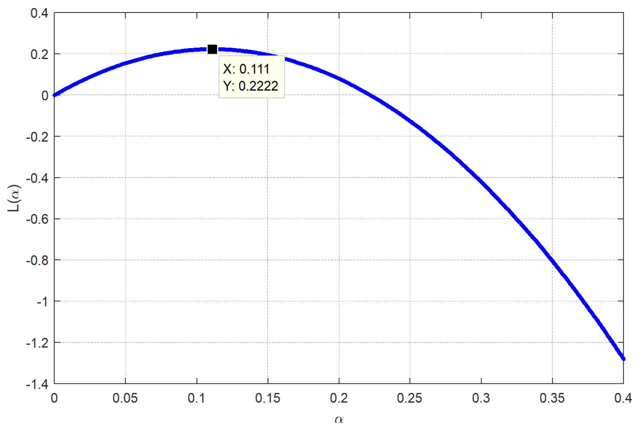


Figure 3: objective function vs $\alpha$ .

**Task 3**: For the data-target pairs in table 1, reduce the classifier function (7) to the most simple the simplest possible form, leading to a simple polynomial in x.

$$
\begin{aligned}
& g(x) = \sum_{j=1}^{4} \alpha_j y_j k(x_j, x) + b = \\
& \alpha(y_1 k(x_1, x) + y_2 k(x_2, x) + y_3 k(x_3, x) + y_4 k(x_4, x)) + b = \\
& \alpha(y_1(-2x + 4x^2) + y_2(-x + x^2) + y_3(x + x^2) + y_4(2x + 4x^2)) + b = \\
& \alpha(6x^2) + b
\end{aligned} \tag{6}
$$

Thus, the simplest form of $g(x)$ is $g(x) = 6\alpha x^2 + b$, where we have $\alpha = 0.1111$ and $b = -1.665$ from averaging over four support vectors. The classifier function is depicted in Fig. (4).

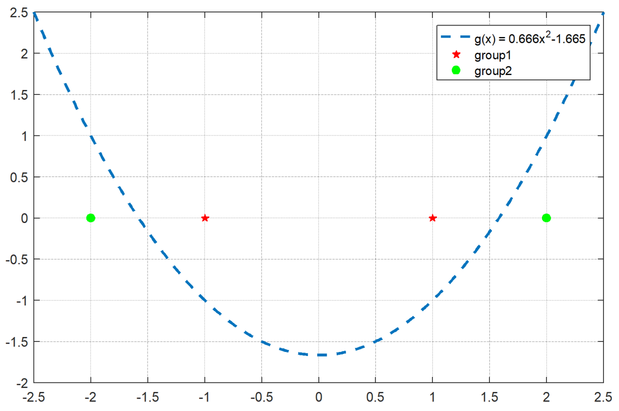


Figure 4: Classifier function as $g(x) = 0.666x^2 - 1.665$.

**Task 4**: With the same kernel $k(x, y)$ as above, what is the solution $g(x)$ of the nonlinear kernel SVM with hard constraint on the data set in table (2)? Explain how you got to this solution. .

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $x_i$ | -3 | -2 | -1 | 0 | 1 | 2 | 4 |
| $y_i$ | +1 | +1 | -1 | -1 | -1 | +1 | +1 |

Table 2: One-dimensional binary classification problem with classes '+1' and '-1':

From the KKT conditions we know that:

$$\alpha_i(y_i(\boldsymbol{w}^T\phi(x) + b) - 1) = 0 \tag{7}$$

Thus, only for support vectors we have $\alpha_i \neq 0$. support vectors are $-2, -1, 1, 2$ points. It means we have $\alpha_1 = \alpha_4 = \alpha_7 = 0$ and $\alpha_2 = \alpha_3 = \alpha_5 = \alpha_6 = \alpha$. The problem is turned into the same problem as Task3 since the support vectors are the same. By calculating new kernel matrix we have:

$$\boldsymbol{K} = \begin{pmatrix} 90 & 42 & 12 & 0 & 6 & 30 & 132 \\ 42 & 20 & 6 & 0 & 2 & 12 & 56 \\ 12 & 6 & 2 & 0 & 0 & 2 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 2 & 0 & 0 & 2 & 6 & 20 \\ 30 & 12 & 2 & 0 & 6 & 20 & 72 \\ 132 & 56 & 12 & 0 & 20 & 72 & 272 \end{pmatrix} \tag{8}$$

By using the coefficients of support vectors we will have again $\alpha = 0.1111$ and $b = -1.6665$. Therefore, the classifier function equals $g(x) = 0.666x^2 - 1.665$ as before (depicted in Fig. 5).

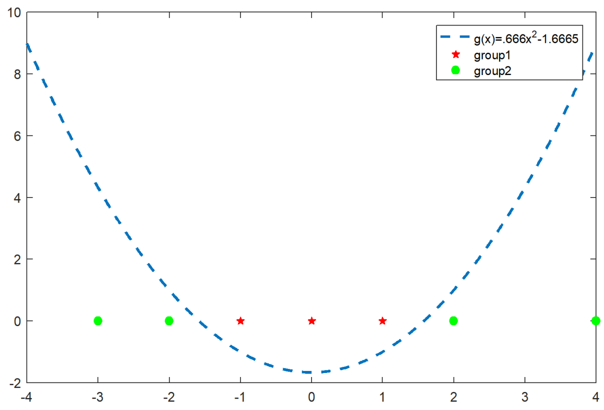


Figure 5: Classifier function as $g(x) = 0.666x^2 - 1.665$.

## 2 The Lagrangian dual of the soft margin SVM

**Task 5**: Show that the Lagrangian dual problem for (9) is given by (10).

$$\begin{aligned} &\underset{\boldsymbol{w},b,\zeta}{\text{minimize}} \ \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\zeta_i \\ &s.t. \ \ y_i(\boldsymbol{w}^T x_i + b) \geq 1 - \zeta_i \\ &\zeta_i \geq 0 \end{aligned} \tag{9}$$

$$\begin{aligned} &\underset{\alpha_1,\ldots,\alpha_n}{\text{maximize}} \ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j \\ &\text{subject to} \ \ 0 \leq \alpha_i \leq C \\ &\sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{10}$$

proof: By having Lagrange multipliers $\alpha_i, \beta_i$ for constraints in 9 we will have the Lagrangian as follows:

$$\begin{aligned}
&\max_{\boldsymbol{w},b,\zeta} \ell(\boldsymbol{w}, b, \zeta, \alpha, \beta) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\zeta_i - \sum_{i=1}^{n}\alpha_i(y_i(\boldsymbol{w}^T x_i + b) - 1 + \zeta_i) - \sum_{i=1}^{n}\beta_i\zeta_i \\
&\alpha_i \geq 0 \\
&\beta_i \geq 0
\end{aligned} \tag{11}$$

by taking derivatives of $\ell$ and set them to zero with respect to $\boldsymbol{w}$, $b$ and $\zeta_i$ we will have :

$$\begin{aligned}
&\frac{\partial \ell}{\partial \boldsymbol{w}} = 0 \longrightarrow \boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i x_i, \\
&\frac{\partial \ell}{\partial b} = 0 \longrightarrow \sum_{i=1}^{n}\alpha_i y_i = 0, \\
&\frac{\partial \ell}{\partial \zeta_i} = 0 \longrightarrow C - \alpha_i - \beta_i = 0
\end{aligned} \tag{12}$$

By substituting $\boldsymbol{w}$ from (12) in (11) we will have:

$$\begin{aligned}
&\ell(\boldsymbol{w}, b, \zeta, \alpha, \beta) = \frac{1}{2}(\sum_{i=1}^{n}\alpha_i y_i x_i)^T(\sum_{i=1}^{n}\alpha_i y_i x_i)) - \sum_{i=1}^{n}\alpha_i(y_i((\sum_{i=1}^{n}\alpha_i y_i x_i)^T x_i + b) - 1 + \\
&C\sum_{i=1}^{n}\zeta_i - \sum_{i=1}^{n}\alpha_i\zeta_i - \sum_{i=1}^{n}\beta_i\zeta_i = -\frac{1}{2}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n}\alpha_i + \\
&C\sum_{i=1}^{n}\zeta_i - \sum_{i=1}^{n}\alpha_i\zeta_i - \sum_{i=1}^{n}\beta_i\zeta_i
\end{aligned} \tag{13}$$

By using (12) we will have $\sum_{i=1}^{n}\zeta_i(C - \alpha_i - \beta_i) = 0$ and the Lagrangian in (13) simply turns into the dual problem as hard-margin SVM. However, since $\beta_i \geq 0$ then $C - \alpha_i = \beta_i \geq 0$. Thus, $\alpha_i \leq C$ and we had $\alpha_i \geq 0$. Finally, we will have $0 \leq \alpha_i \leq C$ and it is verified that the dual problem of (9) is given by (10).

**Task 6**: Use complementary slackness (of the KKT conditions) to show that support vectors with $y_i(\boldsymbol{w}^T x_i + b) < 1$ have coefficient $\alpha_i = C$.
By using complementary slackness and (12) we have:

$$\begin{aligned}
&\alpha_i(y_i(\boldsymbol{w}^T x_i + b) - 1 + \zeta_i) = 0 \\
&\beta_i\zeta_i = 0 \\
&C - \alpha_i - \beta_i = 0
\end{aligned} \tag{14}$$

Now for those SVs that $y_i(\boldsymbol{w}^T x_i + b) < 1$ we have $\zeta_i > 0$. Thus, $\beta_i = 0$ and that forces $\alpha_i = C$.
For SVs that are on the margin we have: $\zeta_i = 0$, Thus, $\beta_i \neq 0$ and $\alpha_i \neq 0$.
For those data points that are not SVs, we have $y_i(\boldsymbol{w}^T x_i + b) \geq 1 - \zeta_i$ and that forces $\alpha_i = 0$.

## 3 Dimensionality reduction on MNIST using PCA

**Task E1**: Compute a linear PCA and visualize the whole training data in d = 2 dimensions. Make sure that the necessary condition to apply PCA is met. Display your results in a plot where the data examples are marked with different markers and colors for each of the two classes. Make the plot clearly interpretable by using legends, adjusting font size, etc.

For this part, first, I used "pca" function of MATLAB. I normalized the train data to the $\mu = 0$ and then I fed it into the pca function. I select the two first columns of "coefs" output of the "pca" function and plot them with their corresponding labels from "train_label" column. Then, I used the "svd" function of the MATLAB and fed the normalized data to that function. I selected the first two columns of left singular vectors matrix ("'U") corresponding to the two largest absolute singular values (560.069391858583, 388.405641246140) and find the projection of the data on those singular vectors. The results of first and second part are depicted in Figs. (6, 7).

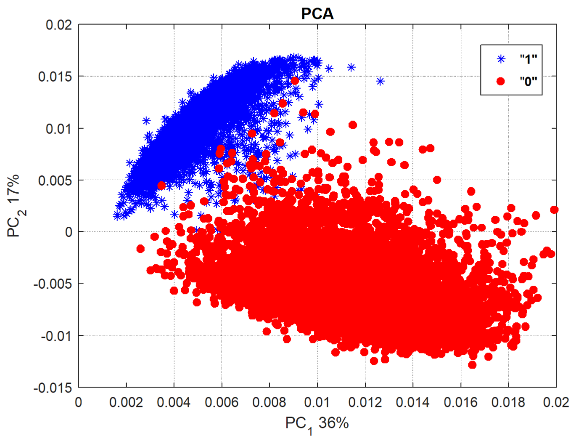


Figure 6: Principal components of MNIST training Data (0,1) in 2D dimensions using "pca" function.

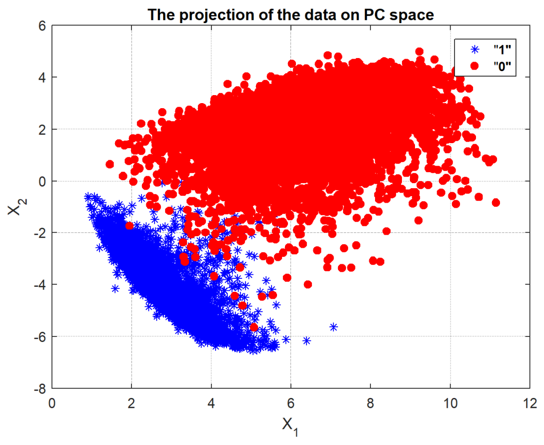
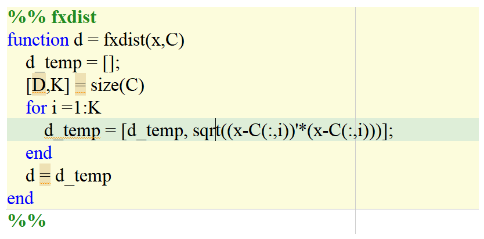


Figure 7: MNIST training Data (0,1) in 2D dimensions using "svd" function.

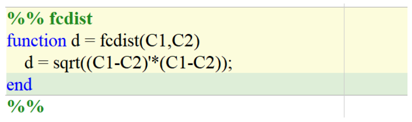
## 4 Clustering of unsupervised data using K-means

**Task E2**: Implement K-means clustering for the training data, using the provided function sketch K_means_clustering.
For this part, I implemented the following functions in order to use in K_means_clustering algorithm as Figs. (8–11).



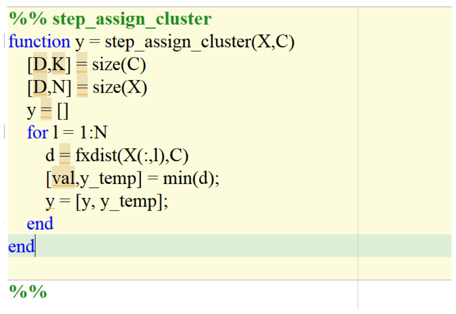
```
%% fxdist
function d = fxdist(x,C)
   d_temp = [];
   [D,K] = size(C)
   for i =1:K
      d_temp = [d_temp, sqrt((x-C(:,i))'*(x-C(:,i)))];
   end
   d = d_temp
end
%%
```

Figure 8: “fxdist” function to calculate the distance of each point from all Centroids.



```
%% fcdist
function d = fcdist(C1,C2)
   d = sqrt((C1-C2)'*(C1-C2));
end
%%
```

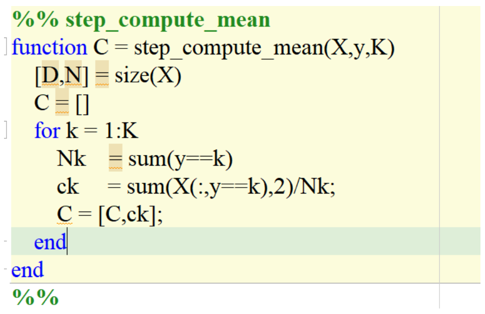
Figure 9: “fcdist” function to calculate the distance between the Centroids.



```
%% step_assign_cluster
function y = step_assign_cluster(X,C)
   [D,K] = size(C)
   [D,N] = size(X)
   y = []
   for l = 1:N
      d = fxdist(X(:,l),C)
      [val,y_temp] = min(d);
      y = [y, y_temp];
   end
end

%%
```

Figure 10: “set_assign_step” function which assigns each data point to the closest Centroid.



```
%% step_compute_mean
function C = step_compute_mean(X,y,K)
   [D,N] = size(X)
   C = []
   for k = 1:K
      Nk   = sum(y==k)
      ck    = sum(X(:,y==k),2)/Nk;
      C = [C,ck];
   end
end
%%
```

Figure 11: “set_compute_mean” function that computes the new Centroids from the mean of the points assigned to that Centroid.

After running the “K_means_clustering” algorithm for $K = 2$ and $K = 5$, the $y$ label vector achieved. After using PCA from previous task, I plot the clusters in different colors for $K = 2$ and $K = 5$ shown in Figs (12, 13). It is clear that for $K = 2$, data points are clustered very close to their real labels. However, for $K = 5$, we see overlaps among clusters. It is because we have only two groups of numbers (0,1). By forcing the algorithm to find 5 groups of data we can not see real separation among those points that are infact in one cluster.

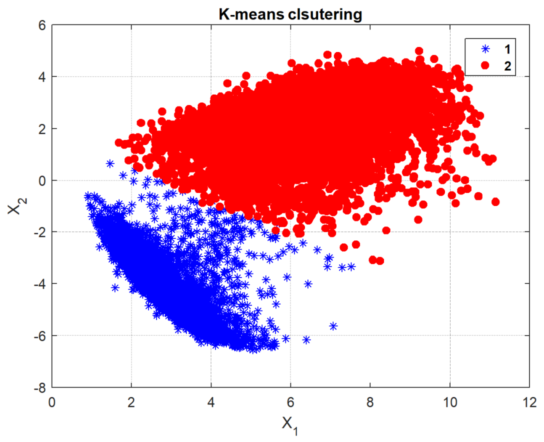


Figure 12: MNIST training Data (0,1) in 2D dimensions using "K-means" algorithm with $K = 2$ clusters.

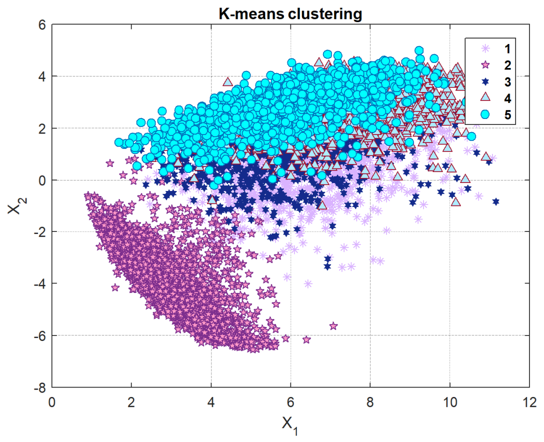


Figure 13: MNIST training Data (0,1) in 2D dimensions using "K-means" algorithm with $K = 5$ clusters.

**Task E3**: Display the $K$ centroids as images. Make a plot with $1K$ subplots illustrating the centroids from each cluster as an image, using the Matlab function imshow(). Make legends, titles or otherwise mark which cluster is displayed. Note that you have to stack the centroids back into the shape 2828 to display the image properly, using, e.g., reshape().
For this part first I converted the centroids to $28 \times 28$ images and then plot them in a subplot. The results are shown in Figs. (14, 15)

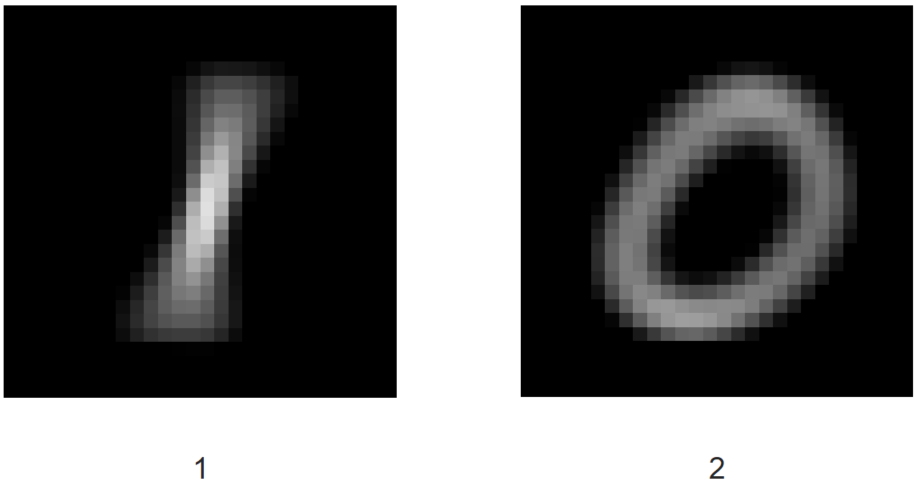


Figure 14: Images of the centroid points for each cluster for $K = 2$ clusters.

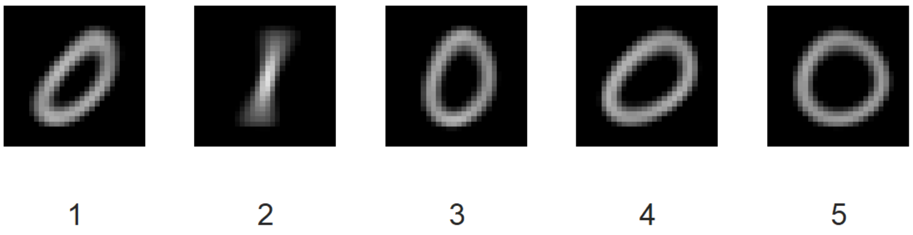


Figure 15: Images of the centroid points for each cluster for $K = 5$ clusters.

The centroid points for $K = 5$ shows that the first, third, fourth and fifth centroids are actually in "**0**" group and only the second centroid is in "**1**" group.

**Task E4**: Now you shall use K-means clustering for classification.

For this part, I first run "k-means-clustering" algorithm to get centroids and classify all the training and test data points by mapping to the closest centroid. Then I compared the assigned label with the true label and count the number of truly classified and mis-classified points. The results are shown in table 3.
The misclassfication error of the test data is less than training data in my simulations. In order to check I plotted the 2D test data with the "k-means-classifier" labels and true labels. It is even clear from the two images that only small number of points have mislabeled compared to true labels since there are two separate, clear clusters in test data. The results

are shown in Figs. (16, 17)

Table 3: K-means classification results

| Training data | Cluster | # '0' | # '1' | Assigned to class | # misclassified |
|---|---|---|---|---|---|
| | 1 | 5821 | 5 | 5826 | 5 |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | K | 102 | 6737 | 6839 | 102 |
| $N_{\text{train}} = 12665$ | | | | Sum misclassified: | 107 |
| | | | | Misclassification rate (%): | 0.8448 |
| **Testing data** | **Cluster** | **# '0'** | **# '1'** | **Assigned to class** | **# misclassified** |
| | 1 | 970 | 0 | 970 | 0 |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | K | 10 | 1135 | 1145 | 10 |
| $N_{\text{test}} = 2115$ | | | | Sum misclassified: | 10 |
| | | | | Misclassification rate (%): | 0.4728 |

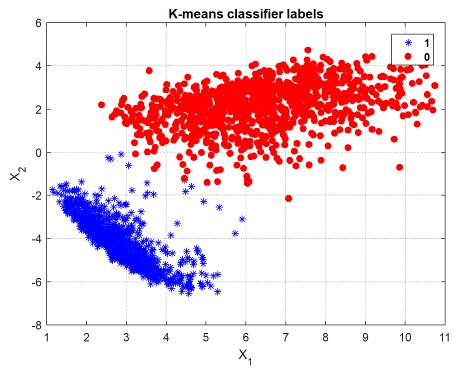


Figure 16: 2D presentation of MNIST test data by k-means classifier labels.

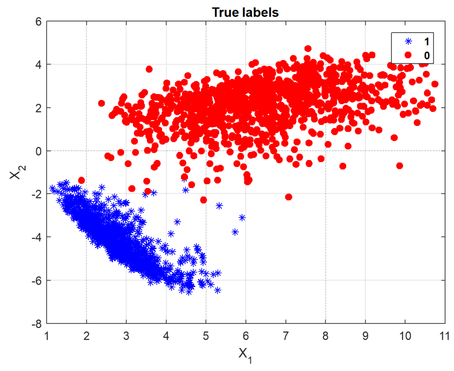


Figure 17: 2D presentation of MNIST test data by true labels.

**Task E5**: Can you lower the misclassification rate further on test data by considering a

different number of clusters $K$?
In this case, my answer is no. As I mentioned in the previous task, since we can see two clear, separate clusters in our test data which comes from originally two classes of (0,1), by changing the number of clusters we will definitely see some overlaps among clusters. However, if our test data comes from another source with different number of clusters (different from train dataset) there is a chance that by changing $K$ we get better results.

## 5 Classification of MNIST digits using SVM

**Task E6**: Use the supervised training data training_data, training_labels to train a linear SVM classifier.
For this task, I plotted the true labels and the assigned labels by linear SVM classifier shown in Figs. (18, 19). There is no misclassification for train data and one for each class of the test data.

Table 4: Linear SVM classification results

| Training data | Predicted class | True class: # '0' | # '1' |
|---|---|---|---|
| | '0' | 5923 | 0 |
| | '1' | 0 | 6742 |
| $N_{\text{train}} = 12665$ | | Sum misclassified: | 0 |
| | | Misclassification rate (%): | 0 |
| **Testing data** | **Predicted class** | **True class: # '0'** | **# '1'** |
| | '0' | 979 | 1 |
| | '1' | 1 | 1134 |
| $N_{\text{test}} = 2115$ | | Sum misclassified: | 2 |
| | | Misclassification rate (%): | 0.0946 |

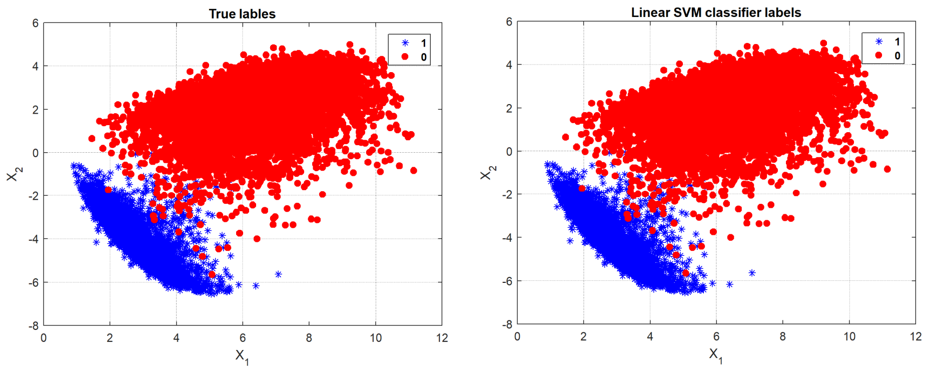


Figure 18: Comparison of True labels and the labels assigned by linear SVM classifier for train data.

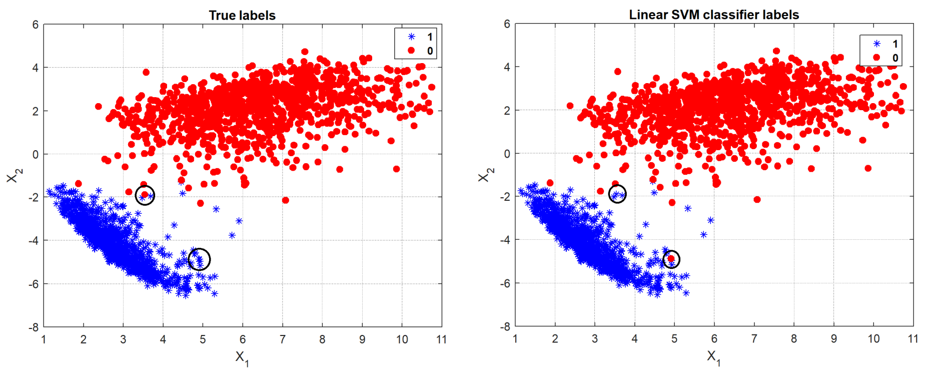


Figure 19: Comparison of True labels and the labels assigned by linear SVM classifier for test data.

**Task E7**: Use the supervised data again to train a non-linear kernel SVM classifier, using a Gaussian kernel.

For this task, I used fitcsvm(normalized_train_data',train_label,'KernelFunction','gaussian', 'KernelScale',27.2223) MATLAB function. For the kernel scale, first I tried "auto" scale and then I extract the "$\beta$" parameter from the model which was around $\beta = 27.22$. Since, our data points are too close to each other we need the Gaussian kernel falls off rapidly. By increasing *beta*, I could see only two misclassfication in test data and fourteen for train data which I depicted in Figs. (20, 21) for both train and test data. We see a better result for the test data although we train the algorithm on train data. The reason is we have a clear, more separated test data compared to the train data.

Table 5: Gaussian kernel SVM classification results

| Training data | Predicted class | True class: # '0' | # '1' |
|---|---|---|---|
| | '0' | 5915 | 6 |
| | '1' | 8 | 6736 |
| $N_{\text{train}} = 12665$ | | Sum misclassified: | 14 |
| | | Misclassification rate (%): | 0.1105 |
| Testing data | Predicted class | True class: # '0' | # '1' |
| | '0' | 978 | 0 |
| | '1' | 2 | 1135 |
| $N_{\text{test}} = 2115$ | | Sum misclassified: | 2 |
| | | Misclassification rate (%): | 0.0946 |

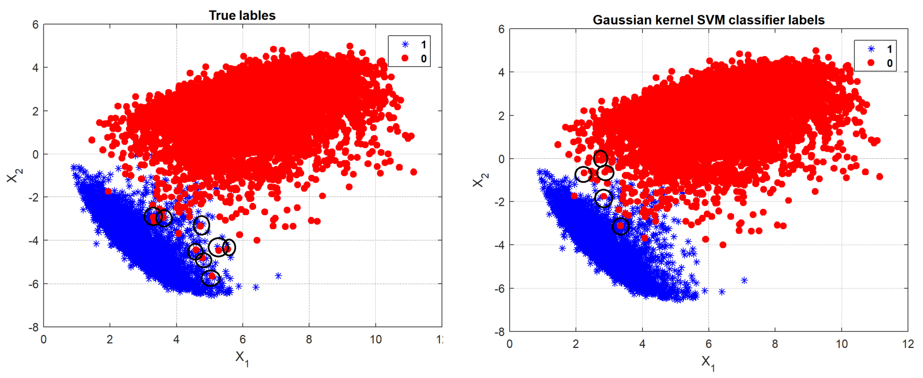


Figure 20: Comparison of True labels and the labels assigned by Gaussian kernel SVM classifier for train data.

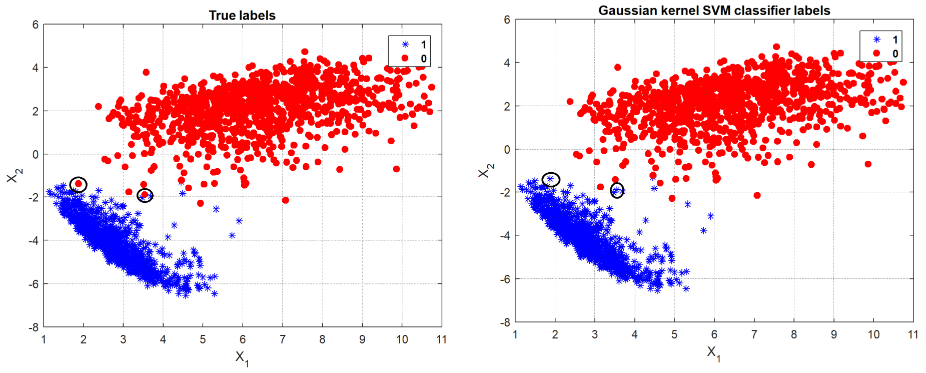


Figure 21: Comparison of True labels and the labels assigned by Gaussian kernel SVM classifier for test data.

**Task E7**: We can achieve very low misclassification rate on both train and test data with a good choice of parameters for the Gaussian kernel SVM. Can we therefore expect the same error on new images? Explain why such a conclusion would be false.

It is not true all the time. It completely depends on the unseen dataset that actually matches the parameter we have chosen for the model or not. If the new images follow the same pattern or comes from the same source there is a chance we see the same performance and the error on that. However, it is not a 100 % true conclusion. The reason is that the tuned parameters depend on the data that we trained the model on. Thus, if the behaviour of the new data differs from the data that parameters are tuned on, we will see different results. This difference could even be positive and we achieve a better result in the test, unseen data like the test data of this assignment.