AC50001 Introduction to Data Mining and

Machine Learning: Assignment 2

Aleksandrs Tarasovs

Student ID: 140013792

AC50001 Introduction to Data Mining and Machine Learning

MSc Data Engineering

University of Dundee, 2019

# Introduction

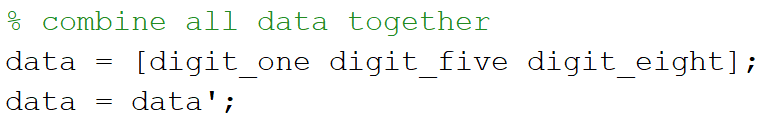
The aim of the project was to learn how to implement clustering, classification and dimensionality reduction in MATLAB. The student was provided with the MNIST handwritten digit dataset that is commonly used in data mining and machine learning. For the assignment, the student used sub-set of the MNIST dataset that included only three handwritten digits – 1, 5 and 8. Each digit contained 100 images of size 28x28 pixels.

# Requirements

The assignment was divided in three parts. According to the first part, the student had to implement principal component analysis (PCA) to reduce the dimensions of each image descriptor to two using the first two principal components. After this step was completed, the student had to choose one of clustering methods (hierarchical, k-means or GMM) and cluster data point in 2D space into three clusters. The aim of the second part was quite similar, but instead of using PCA to reduce dimensions, the student had to use linear discriminant analysis (LDA) and compare results with the first part. The third part required to solve classification problems using support vector machine (SVM) and a neural network classifier. For SVM, the student was asked to use linear and RBF kernels in a five-fold cross validation setting, and for neural networks – one hidden layer with five-fold cross validation setting. The implementation process and results of all these parts will be discussed further in the report.

# Data pre-processing

Before starting the implementation process, the student had to create a feature matrix of all MNIST digits together, so each row would represent a feature vector of each image. In the result, the 300x784 matrix was created (See code snippet 1).



***Code snippet 1*** *Part of code for data pre-processing*

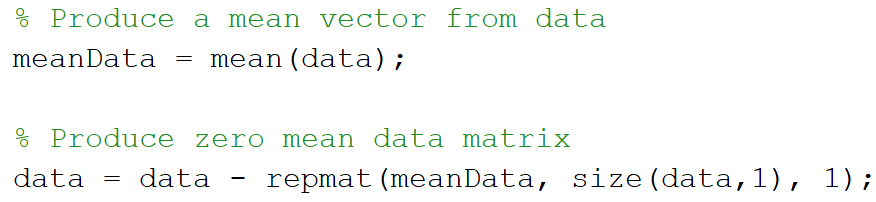
# Principal component analysis (PCA)

## Implementation

The implementation process of the PCA was based on five main stages – producing a zero-mean matrix, calculating covariance matrix, calculating eigenvectors and eigenvalues of the covariance matrix, calculating the PCA score and clustering data points into three clusters using K-means clustering algorithm.

### Produce a zero-mean matrix

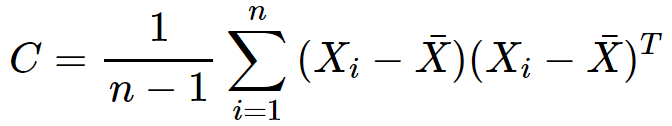
To produce a zero-mean matrix, firstly, a mean vector was created from the original feature matrix. Thereafter, the student calculated a zero-mean matrix by subtracting a mean vector from each row of the feature matrix (See code snippet 2). Subtracting the mean makes variance and covariance calculations easier by simplifying their equations.



***Code snippet 2*** *Part of code to create a zero-mean matrix*

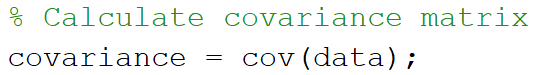
### Calculate covariance matrix

Covariance measures the variation of multiple random variables, for instance, how much four random variables vary together. The equation to calculate the covariance matrix is provided below (See figure 1).



***Figure 1*** *Equation to calculate the covariance matrix*

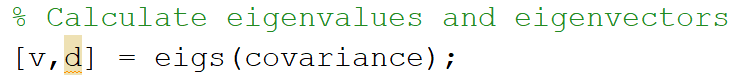
To calculate the covariance matrix in MATLAB, the student used the pre-built MATLAB command “cov” which was applied on a zero-mean matrix produced in the previous step (See code snippet 3).



***Code snippet 3*** *Part of code to calculate a covariance matrix*

### Calculate eigenvectors and eigenvalues

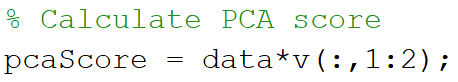
Eigenvectors and eigenvalues were calculated from the covariance matrix. Eigenvectors had to be calculated as they were used in “Calculate PCA score” step. Overall, eigenvectors are vectors on which a linear transformation can be performed without changing their direction. To calculate eigenvalues and eigenvectors, the student used prebuilt MATLAB command “eigs” (See code snippet 4).



***Code snippet 4*** *Part of code to calculate eigenvectors and eigenvalues*

### Calculate PCA score

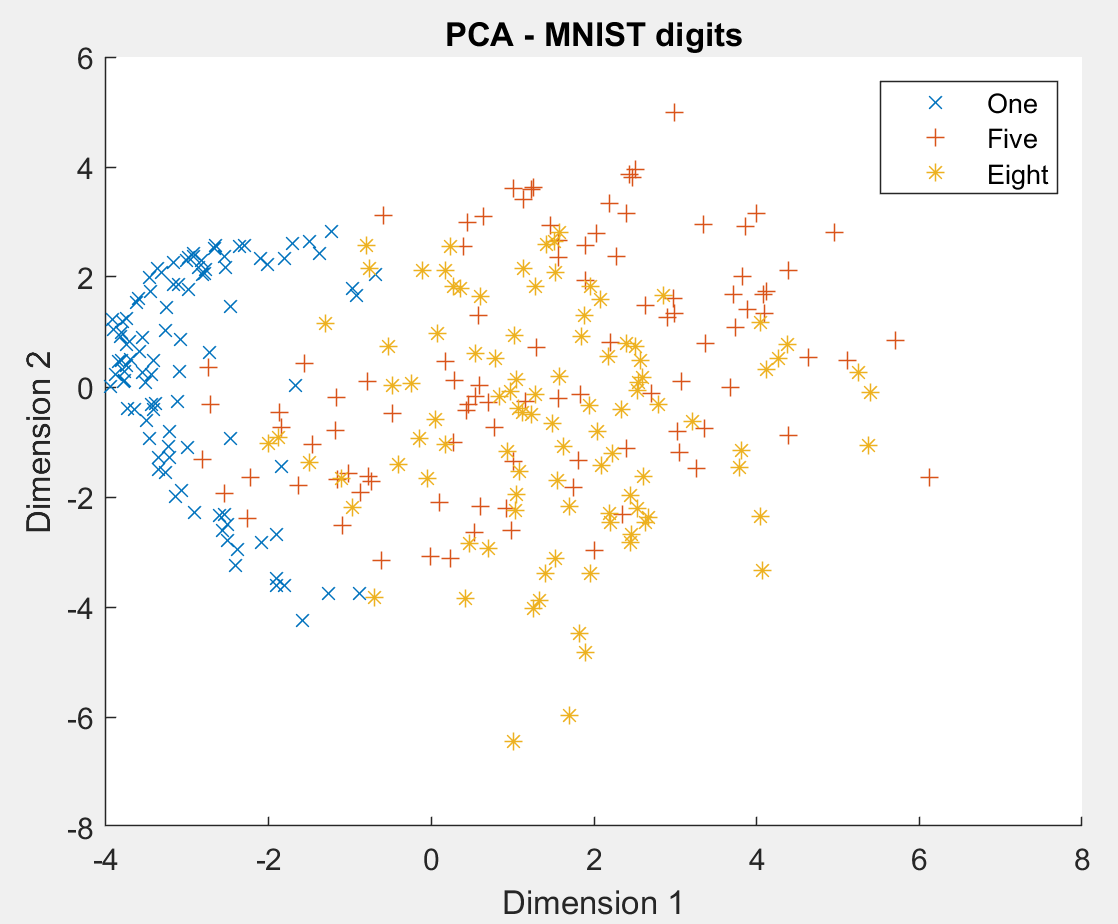
Once eigenvalues were collected a projection was calculated. To do that, the student had to multiply a matrix with eigenvectors by zero-mean feature matrix (See code snippet 5). As far as the requirement was to use only first two principal components, only first two columns of eigenvector matrix were used.



***Code snippet 5*** *Part of code to calculate PCA score*

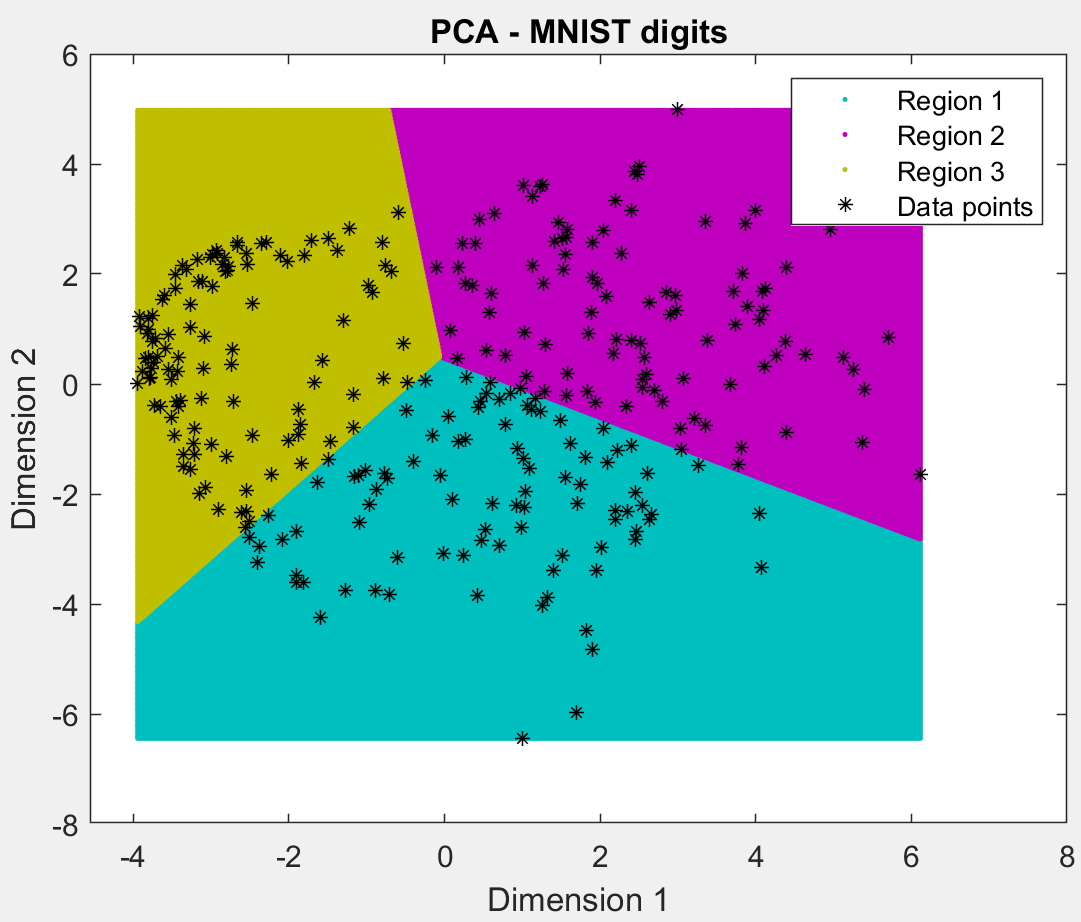
### Cluster data points into three clusters using K-means clustering algorithm

Once the PCA score was calculated, the student had to apply K-means clustering algorithm to cluster data in the 2D space into three clusters. To make a good evaluation of how well K-means clustering algorithm performed on 2D data from PCA, firstly, the student simply plotted data points that were generated by PCA projection (See figure 2).

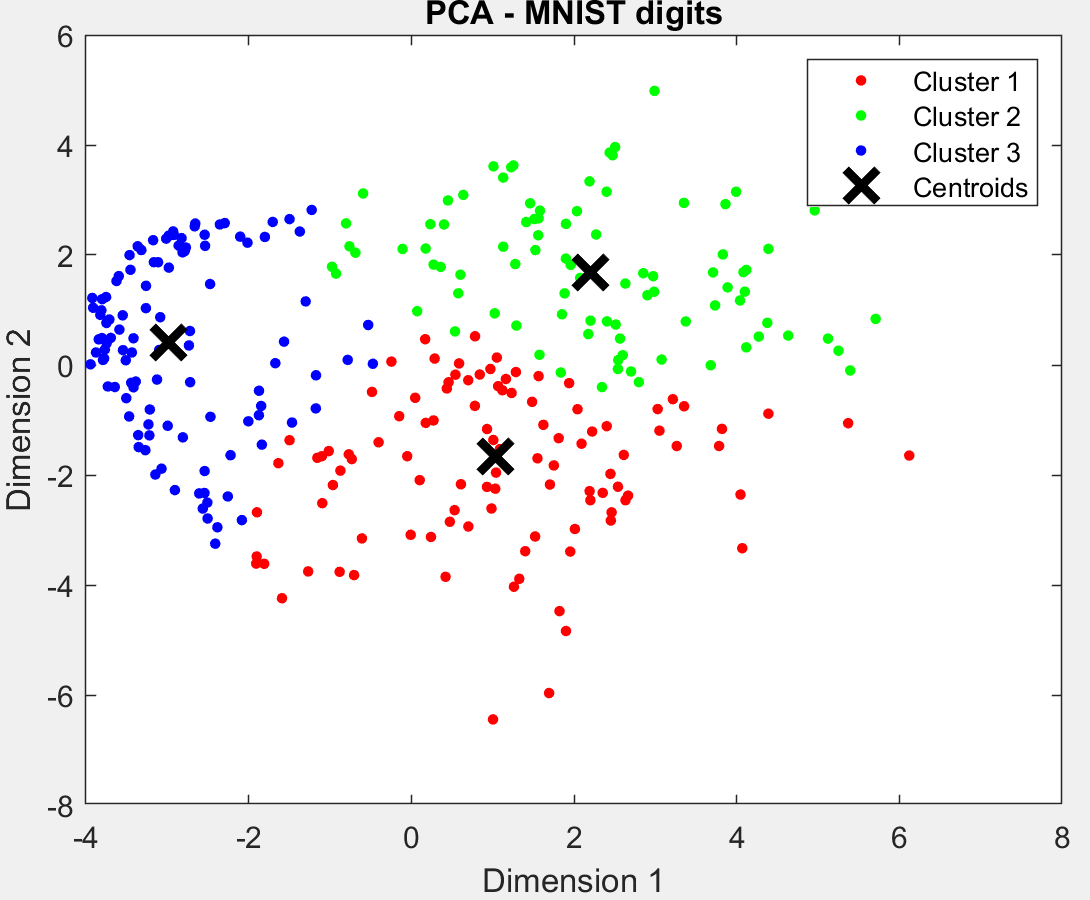


***Figure 2*** *Data after PCA projection*

As it can be seen from the plot, data points are not well separated by digits, for example, 5 and 8 digits are plotted almost in the same area. At this point, the student assumed that K-means clustering algorithm will not perform well on data from PCA projection. To prove or reject the assumption, the following two plots were created – the plot of cluster regions (See figure 3) and the plot of clusters and cluster centroids (See figure 4).



***Figure 3*** *K-means cluster regions*



***Figure 4*** *K-means clusters and cluster centroids*

From the results, it can be seen that the assumption was proved, K-means clustering did not perform well on 2D data after PCA projection. While K-means algorithm worked quite well to cluster images of digit 1, it did not perform well on digits 5 and 8. For example, it can be seen that data points of digits 5 and 8 are spread mostly all over the plot and the K-means in many cases clusters fives as eights and backwards.

# Linear discriminant analysis (LDA)

## Implementation

There are two different ways to implement a linear discriminant analysis – LDA for two class problems and LDA for multiple class problems. The following steps had to be produced for two class clustering problem:

* Assign data classes
* Calculate means for each class
* Calculate covariance matrix for each class
* Calculate within-class matrix by summing up covariance matrices of each class
* Calculate the LDA projection vector by multiplying inverse within-class matrix by transpose difference of means
* Calculate the projection score

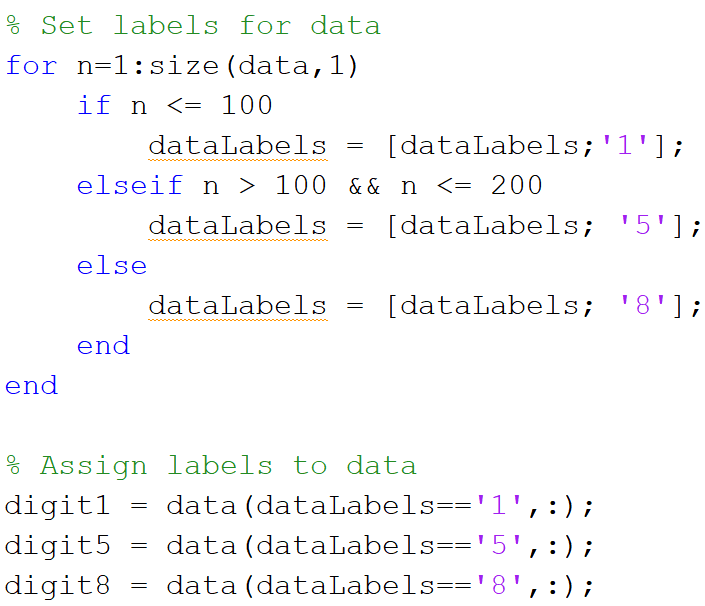
The other way to calculate LDA for multiple class clustering problem was slightly different:

* Assign data classes
* Calculate means for each class
* Calculate overall mean by summing up class means and dividing on the number of classes
* Calculate within-class matrix by summing up covariance matrices of each class
* Calculate between-class matrix
* Calculate LDA projection by multiplying inverse within-class matrix by between-class matrix
* Calculate eigenvalues and eigenvectors
* Calculate LDA score

As far as the problem discussed in the requirements of the assignment included three classes – 1, 5 and 8, the student had to implement the LDA algorithm for multiple class clustering problem. Some steps of calculating LDA score are same as for PCA, so only some parts of LDA score calculation process will be discussed in this part of the report.

### Assign data classes

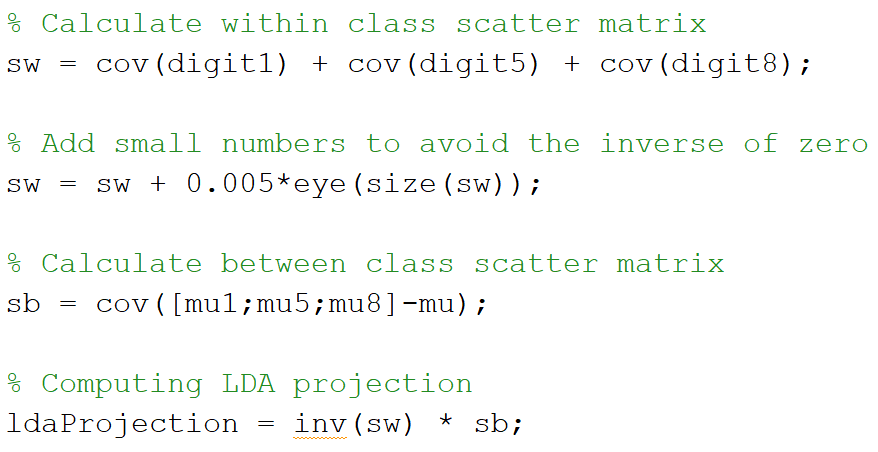
As far as LDA is a supervised transformation technique while PCA is unsupervised, the student had to assign labels for each data class. Overall, data was separated in three parts and for each subset of data a unique label was assigned (See code snippet 6).



***Code snippet 6*** *Part of code to assign labels and separate data based on labels*

### Calculate LDA projection

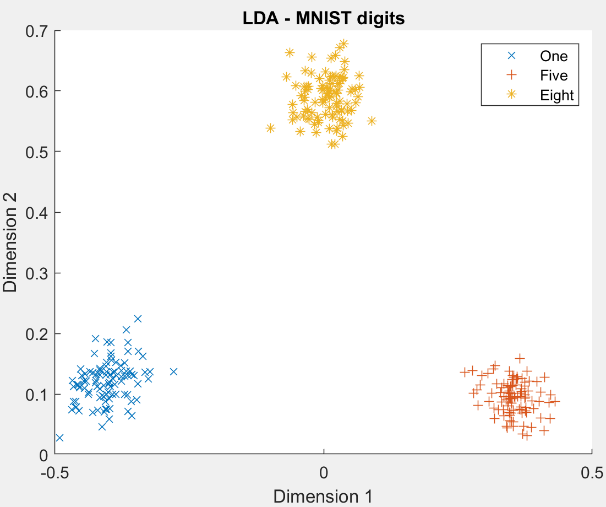
In order to calculate the LDA projection, within-class and between-class matrices had to be calculated in advance. Firstly, within-class matrix was calculated by summing up covariance matrices of each class. Thereafter between-class matrix was calculated by subtracting the overall mean vector from mean vector of each class and calculating the covariance matrix of that. And finally, the LDA projection was calculated by multiplying inverse within-class matrix by between-class matrix (See code snippet 7). During the development process the student faced a problem of calculation an inverse within-class matrix. The problem appeared because the within-class matrix contained nulls. To solve the problem, the student decided to add small to the within-class matrix to avoid the inverse of null (See code snippet 7).



***Code snippet 7*** *Part of code to calculate LDA projection*

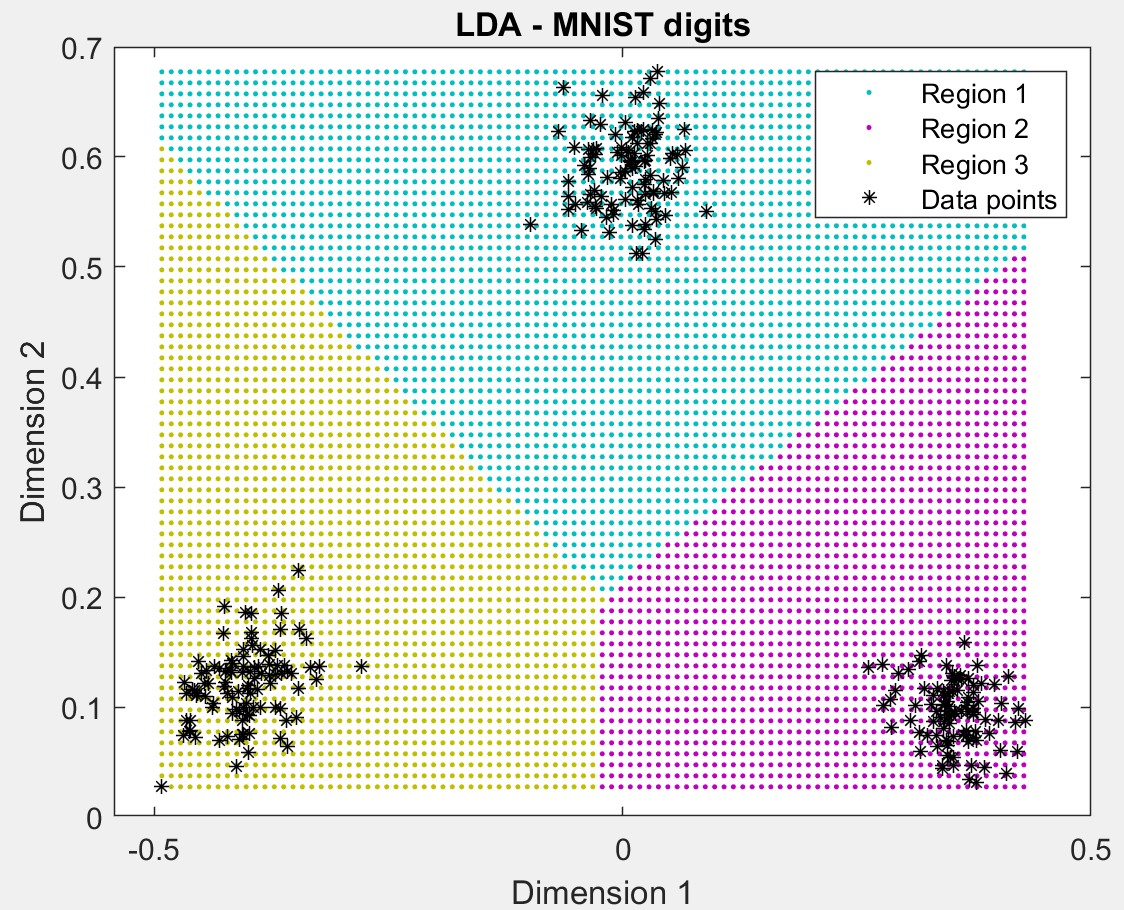
### Cluster data points into three clusters using K-means clustering algorithm

Once the LDA score was calculated, the student had to apply K-means clustering algorithm to cluster data in the 2D space into three clusters. To make a good evaluation of how well K-means clustering algorithm performed on 2D data from LDA, firstly, the student simply plotted data points that were generated by LDA projection (See figure 5).

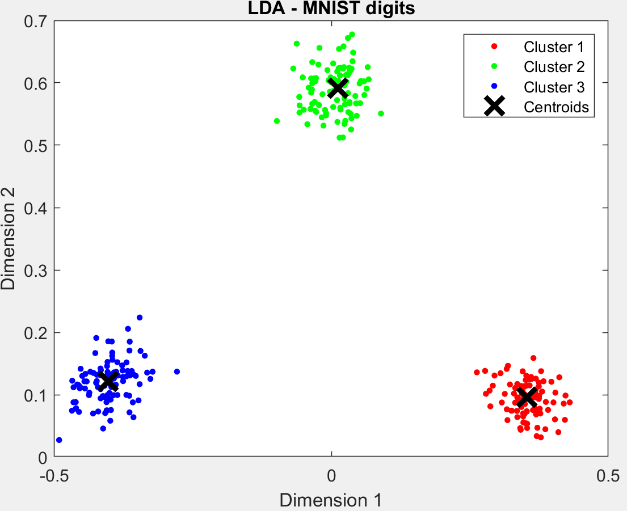


***Figure 5*** *Data after LDA projection*

The plot shows that the data classes are split well, so the student assumed that the K-mean clustering algorithm will perform much better than it performed on data after PCA projection. To prove or reject the assumption, the following two plots were created – the plot of cluster regions (See figure 6) and the plot of clusters and cluster centroids (See figure 7).



***Figure 6*** *K-means cluster regions*



***Figure 7*** *K-means clusters and cluster centroids*

From the results, it can be seen that the assumption was proved, K-means clustering performed really well on 2D data after LDA projection. Each cluster included 100 data points of each class, so there was no data loss at all. As far as there was no loss, it can be considered K-means clustering algorithm applied on 2D data after LDA projection is perfect.

# Support vector machine (SVM)

## Implementation

At this stage, the student was required to solve a two-class classification problem using SVM supervised learning algorithm. The problem had to be solved using two SVM kernels – RBF and linear kernel. At the beginning, the student attempted to solve the problem using the external library “LIBSVM” that was provided on the labs, but unfortunately there was no way found to solve the problem using 5-fold cross validation. Since there was no way to solve the problem as it was required, the student switched to another way of SVM implementation – using default MATLAB commands. As far as the first way of the implementation have not met the requirements, it will not be discussed in the report, but the code can be viewed by accessing SVM/SVM.m file.

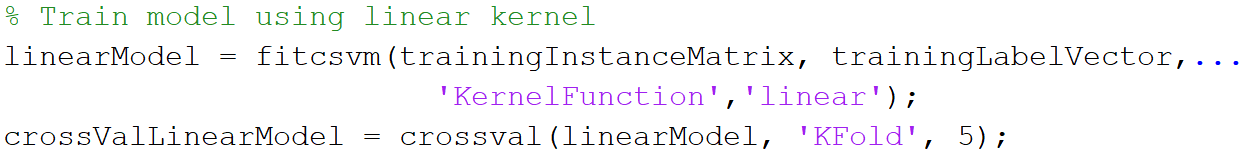
The following implementation steps were performed to build SVM classifier:

* Assign data labels (same as for LDA – 1, 5, 8)
* Assign data classes
* Divide data into training and testing sets with the ratio of 4:1
* Get indexes for training and testing samples
* Create training and testing labels ground truth, and create training and testing data matrices
* Train model using linear and RBF kernels
* Apply 5-fold cross validation
* Test classifier
* Plot confusion matrices and ROC curves to compare results

Data classes were assigned based on the digit from the MNIST dataset. Class 0 was assigned to ones and eight, and class 1 was assigned to fives. Once all the steps of data preparation were completed, the student started the implementation process of model training. The model was trained using RBF and linear kernels. The implementation process and achieved results are presented below.

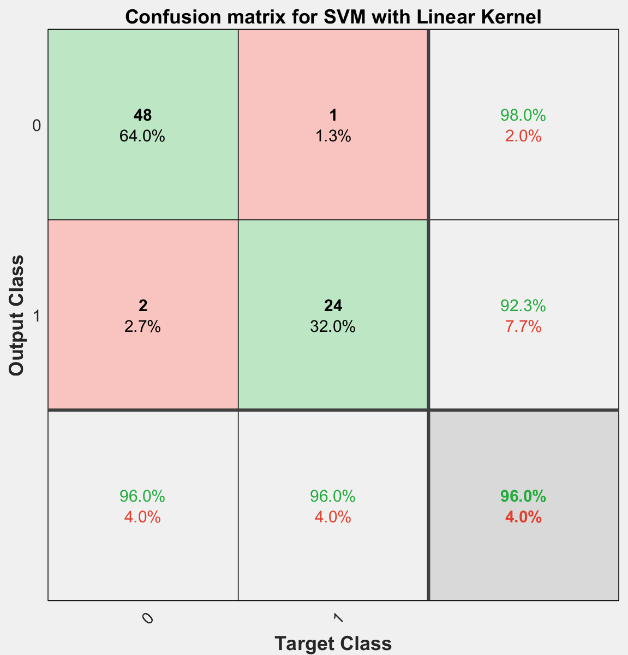
### SVM with a linear kernel

To train model, the pre-build MATLAB command “fitcsvm” was used. Firstly, the model was trained without providing any additional parameters (See code snippet 8). The only used additional parameter was “KernelFunction” to select linear kernel. Once the model was trained, it was cross validated using 5-fold cross validation (See code snippet 8) and tested.

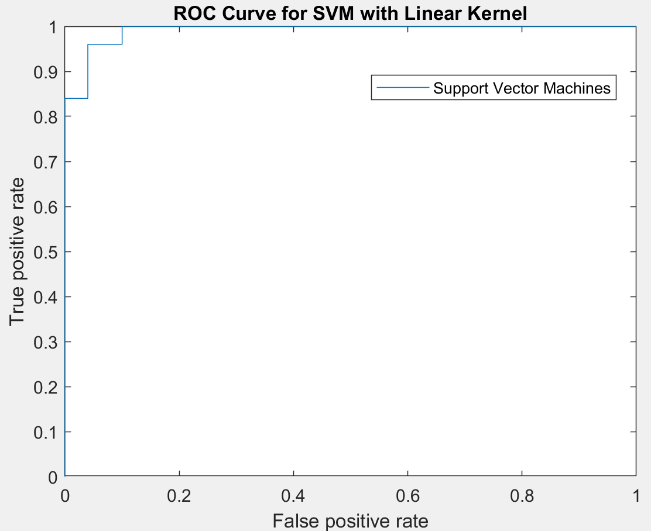


***Code snippet 8*** *Part of code to train SVM model using linear kernel*

Surprisingly, the trained SVM model provided quite good results from the first attempt. The model accuracy was 94%. To show results on testing data, the student plotted a confusion matrix (See figure 8) and ROC curve (See figure 9). The area under the curve (AUC) was 0.9912.



***Figure 8*** *Confusion matrix for SVM with linear kernel (Accuracy=96%)*



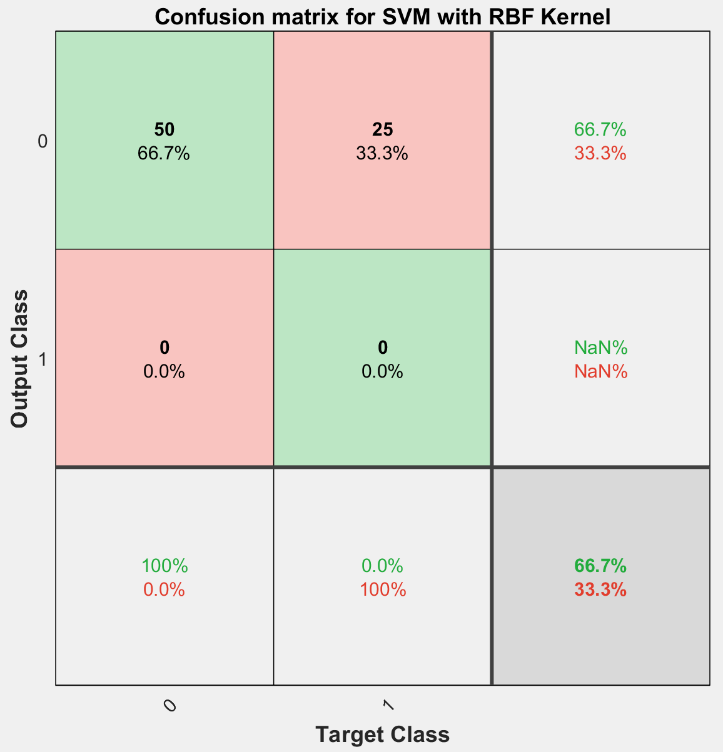
***Figure 9*** *ROC curve for SVM with linear kernel (AUC=0.9912)*

According to the confusion matrix, the accuracy of default SVM classifier with a linear kernel was 96%. Only three images were misclassified – two false negatives and one false positive. Speaking about penalty parameters optimization, the student made an attempt to optimize such parameters as kernel scale, alpha and kernel offset, but there was no better result.

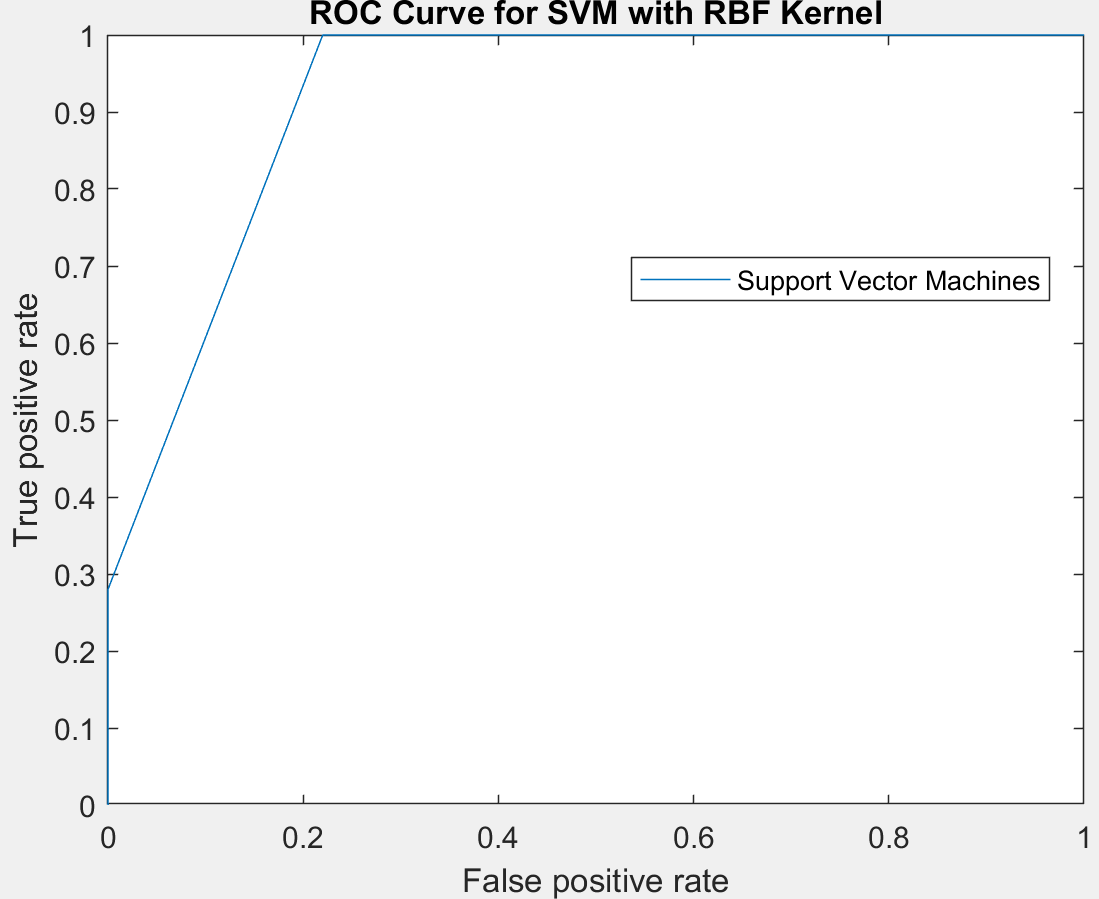
### SVM with an RBF kernel

The implementation steps for the SVM with an RBF kernel were exactly same as for SVM with a linear kernel. The only difference was that instead of parameter “KernelFunction” linear, the student had to use “KernelFunction” RBF. The MATLAB default settings for RBF kernel did not suit well to this problem, the model accuracy was only 66.66%.

The results are provided below (See figure 10 and 11).

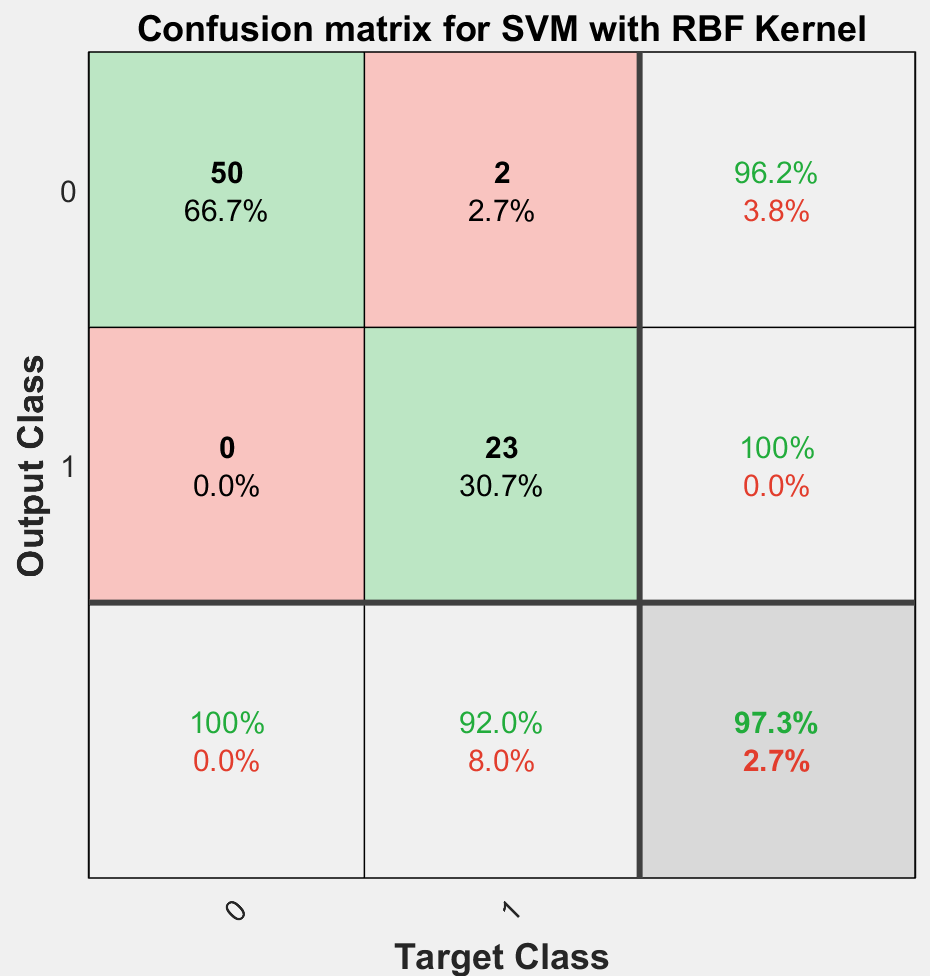


***Figure 10*** *Confusion matrix for SVM with RBF kernel (Accuracy=66.7%)*

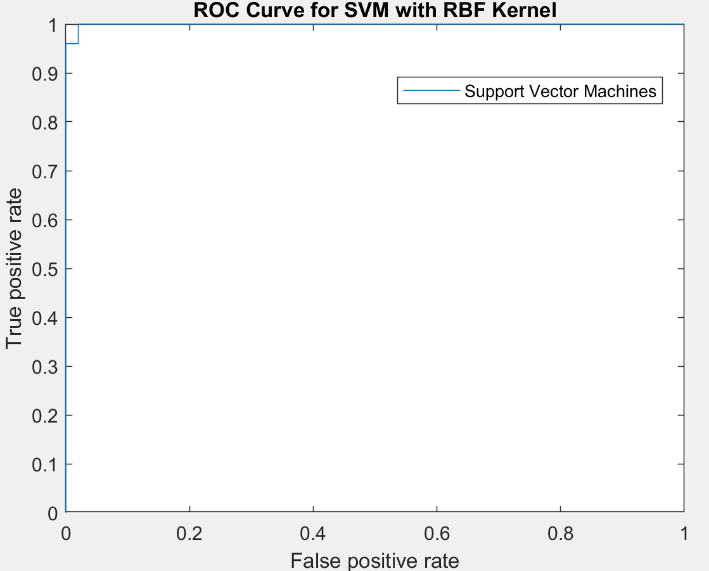


***Figure 11*** *ROC curve for SVM with RBF kernel (AUC=0.9208)*

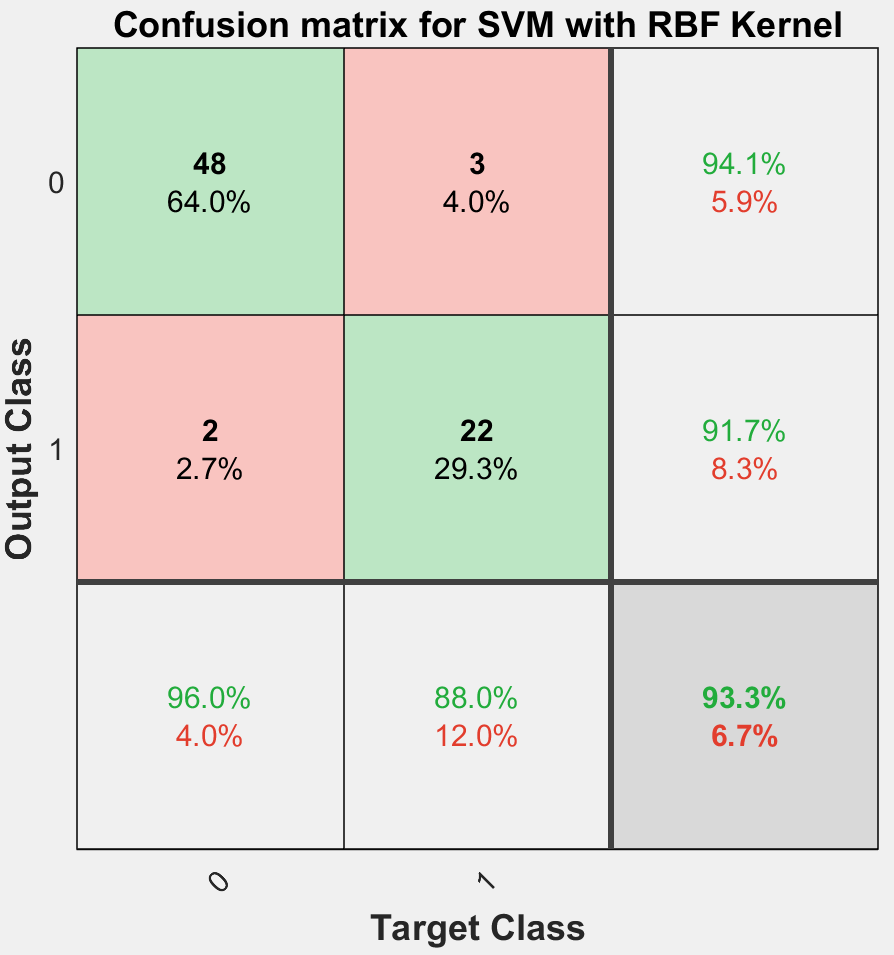
According to the confusion matrix, the classifier did not predict any images to be of class 1 – fives. It classified all images to be a class of 0 – 66.7% true negative, and 33.3% false positive. The result of the classifier was horrific, if the testing set would include only images of digit 5 (class 1), the accuracy would be 0%. To improve the results of the classifier, the student decided to change the scale of kernel – gamma parameter. The default value for kernel scale parameter was 1, so the student decided to use 10 and 20 to compare results (See figures 12, 13, 14, 15).



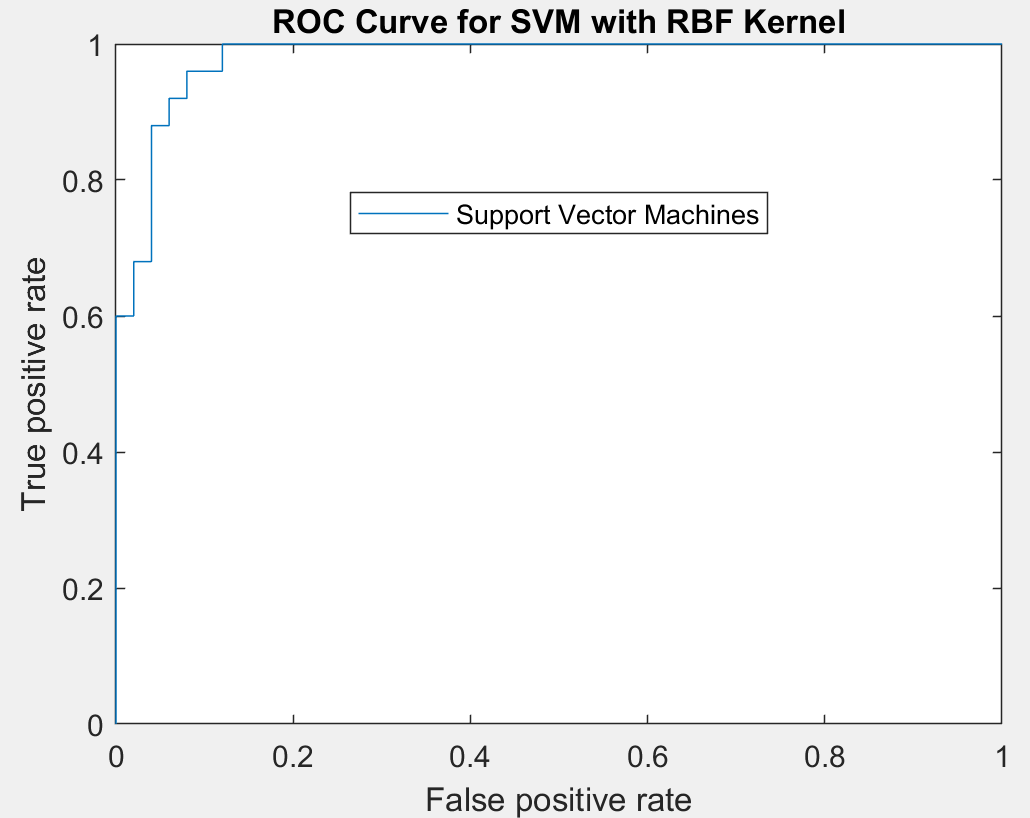
***Figure 12*** *Confusion matrix for SVM with RBF kernel – Kernel Scale=10 (Accuracy=97.3%)*



***Figure 13*** *ROC curve for SVM with RBF kernel – Kernel Scale=10 (AUC=0.992)*



***Figure 12*** *Confusion matrix for SVM with RBF kernel – Kernel Scale=20 (Accuracy=93.3%)*



***Figure 13*** *ROC curve for SVM with RBF kernel – Kernel Scale=20 (AUC=0.98)*

By comparing the figures of confusion matrices and ROC curves after parameter optimization and figures before parameter optimization, it can be seen that the SVM classifier with an RBF kernel with a kernel scare larger than one performs much better. For instance, the classifier with a kernel scale of 10 had an accuracy of 97.3% and the AUC was 0.992. Only two images of class 0 were misclassified. After playing with different penalty parameter, it was found that the best performance of the classifier is when the kernels scale is between 10 and 12, sometimes the accuracy of 100% was achieved by that.

As we can see from the results (See table 1), both classifiers performed quite well, but SVM classifier with an RBF kernel with a kernel scale of 10 performed slightly better.

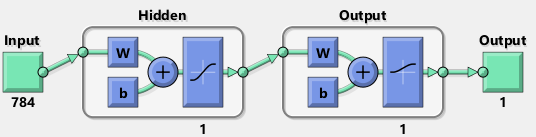
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Linear kernel | RBF kernel | RBF kernel scale=10 | RBF kernel scale=20 |
| AUC | 0.9912 | 0.9208 | **0.992** | 0.08 |
| Accuracy | 96% | 66.7% | **97.3%** | 93.3% |

***Table 1*** *Results of SVM classifiers*

# Neural networks

## Implementation

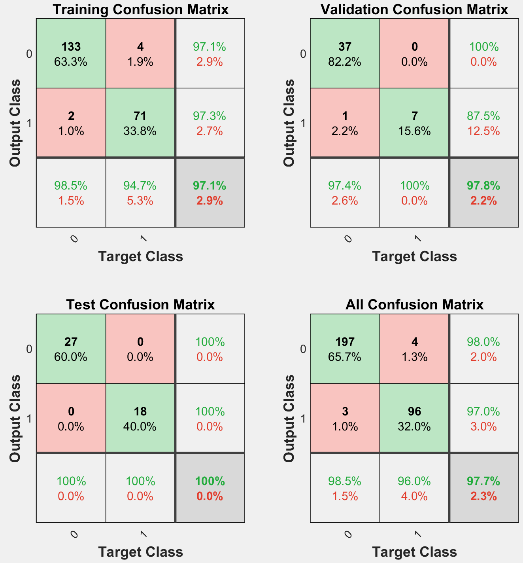
To implement neural networks, the student used MATLAB neural network recognition application (nprtool). The application allowed to meet all the requirement of the assignment. Using neural network recognition tool, the student created a neural network classifier with one hidden layer (See figure 14).



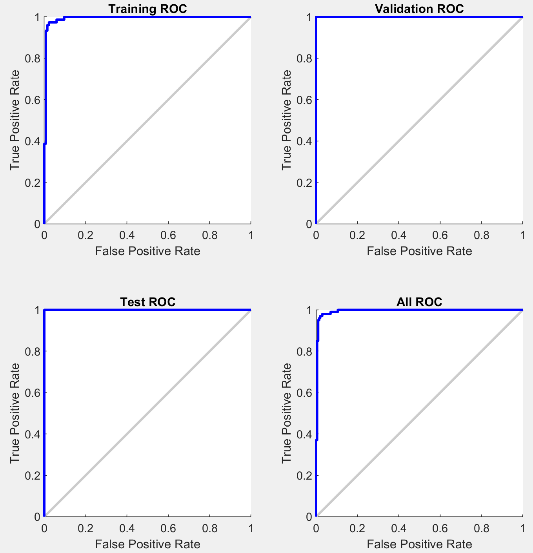
***Figure 14*** *Neural network classifier architecture*

As far as the tool offered an option to generate advanced MATLAB script that was used to create neural network classifier, the student was able to make some optimizations of the classifier to improve accuracy results. The ratio for the training, validation and testing samples was 70%:15%:15%.

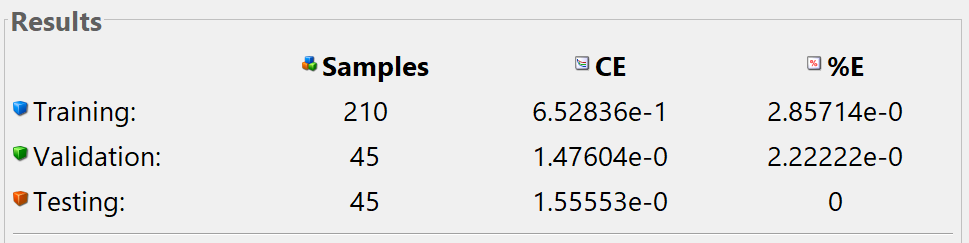
To show and compare results, the student plotted confusion matrices and ROC curves (See figures 15, 16 and 17).



***Figure 15*** *Confusion matrices for neural network classifier*

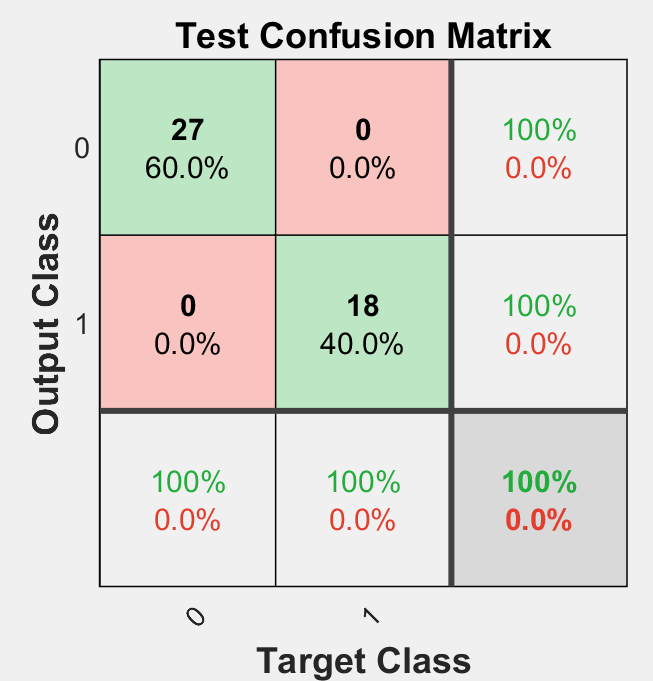


***Figure 16*** *ROC curves for neural network classifier*

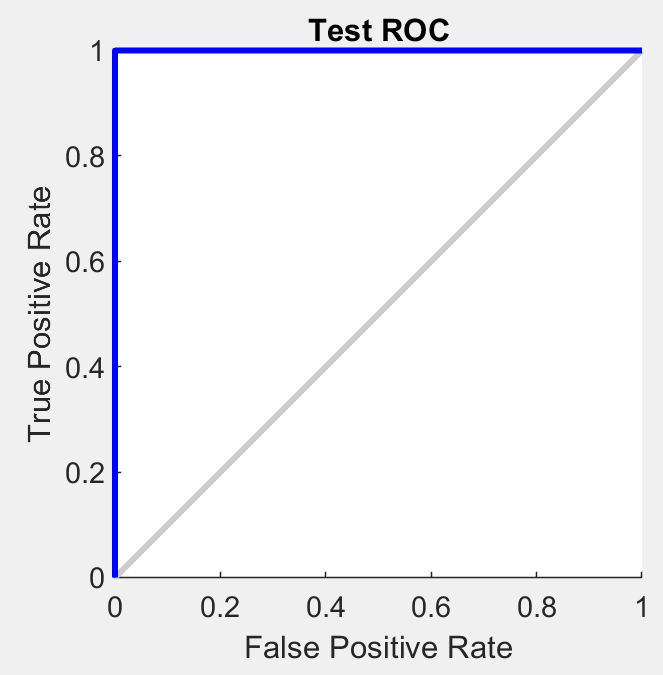


***Figure 17*** *Result table for neural networks*

As it can be seen from the result, neural networks classifier performed quite well on the provided data. The best result that was achieved by neural networks was 100% on testing data (See figures 18 and 19). While the accuracy on testing data was 100%, the accuracy of the model was 97.1% and on validation set – 97.8%. So, the overall accuracy of the classifier was 97.7% and the error rate – 2.3%.



***Figure 18*** *Confusion matrix for neural network classifier*



***Figure 19*** *ROC curves for neural network classifier*

# Conclusion

Overall, based on the requirements of the project, the student was required to implement three different machine learning mechanisms – dimensionality reduction, clustering and classification. For dimensionality reduction, principal component analysis and linear discriminant techniques were used. To implement clustering, the student used K-means clustering algorithm. As it was stated before, K-means clustering algorithm performed on 2D data after LDA projection. The data points were divided in three separate classes without any loss.

To implement classification the student used two classifiers – support vector machines and neural networks. The accuracy of all classifiers was satisfactory. With neural networks, the accuracy on testing data was 100% and AUC was equal to 1. The best accuracy achieved with SVM classifier with an RBF kernel (gamma = 10) was 97.2%.

As far as in the development process only one clustering algorithm was used, the student considered that in future, it would be interesting to implement such clustering algorithms as hierarchical clustering or GMM and compare their performances with the performance of K-mean clustering algorithm

During the development process, the student used version control system – GitHub. The link to the GitHub is <https://github.com/ATarasovs/Data-Mining-and-Machine-Learning-Assignment>.

# Instructions

The code is divided in four folders – PCA, LDA, SVM and NeuralNet. MNIST digits are stored in AC50001\_assignment2\_data.mat.

* To run K-means clustering on the data after PCA projection, open PCA folder and run main.m script.
* To run K-means clustering on the data after LDA projection, open LDA folder and run main.m script.
* To run SVM classifier:
  + For simple SVM classifier open SVM folder and run SVM.m script.
  + For SVM with 5-fold cross validation open SVM folder and run SVM\_5Fold.m script.
* To run neural networks classifier open NeuralNet folder and run main.m script.

You can access the instruction in README.txt file as well.