AC50001 Introduction to Data Mining and Machine Learning   
**AC50001.2 Assignment: Classification and Clustering***Author: Vladislavs Ignatjevs (120015095)*

During last week I was working on Classification and Clustering assignment in attempt to create a system capable of dimension reduction using LDA, PCA and clustering using K-Means, and classification using SVM with a linear kernel, SVM with RBF kernel and Neural Network classifier. All these created components were tested/trained on limited MNIST handwritten digit database (only 3 numbers, 100 samples each).

**Q1. Principal Component Analysis (PCA)**

Applying PCA on data was the first classification problem I attempted to implement. PCA is a good way to compress data in regards to pattern recognition improvement. PCA transforms data to new, linearly uncorrelated coordinate system. It has its advantages and disadvantages. For example, if speaking about advantages, PCA is fast and easy to implement. However, if we overwrite data after applying PCA, it will not be possible to get the exact original data back, since PCA clears up the data that it considers unnecessary. As the result, we might lose significant information during the classification process. Apart from that, there are situations where we cannot apply PCA, since it works on linear spaces. For example, if the data is manifold structured, PCA will not perform an analysis in respect to expected results. The other drawback of PCA I found out comes from inability of Linear PCA to provide the information about class labels. LDA, on the other hand, solves this problem.

At the beginning I spent lots of time because I understood the task in a wrong way. Rather than applying PCA to all number data, I attempted to apply PCA on every number separately. This gave interesting results when plotted results. That was what I believe an attempt to reduce dimensions from 100 to 2 and create an interesting interpretation of most common features for numbers one, five and eight. Due to this being unsuitable for the task. I am not attaching the result of these application here, since it is not applicable for the task given, but I backed up the unused code into folder called “Unused”.

PCA can be summarized in the following steps:

1. Subtracting the mean for individual dimensions
2. Calculating covariance matrix
3. Calculating Eigen values and Eigen vectors
4. Transformation into new data

For the task allocated, I started with importing the data set and concatenating all the data sets into one larger matrix and transposing it:

%combine all data

all\_data = [digit\_one digit\_five digit\_eight]';

Then, I calculated the mean for this matrix and subtracted it from individual dimensions:

%substract the mean for individual dimenstions

all\_data = all\_data - repmat(meanAll, size(all\_data,1),1);

After doing that, it was possible to calculate covariance matrix (to see how much data vary from the mean in respect to each other). Then it was possible to calculate Eigen values and Eigen vectors from the covariance matrix calculate in the previous step. Then, by multiplying the matrix containing all data by specific Eigen vector’s values it made it possible to calculate PCA score:

score = all\_data\*v(:,1:2);

This made it ready for the data to get picked up for clustering.

**K-Means method for clustering.**

For data clustering I have used K-Means method because it was more familiar to me than the others. Using the following code I applied k-means to cluster the data into 3 clusters:

%do k-means

[idx, c] = kmeans(pca\_score,3);

%dim1

x1 = min(pca\_score(:,1)):0.01:max(pca\_score(:,1));

%dim2

x2 = min(pca\_score(:,2)):0.01:max(pca\_score(:,2));

%calculate mesh grid

[x1G,x2G] = meshgrid(x1,x2);

x\_grid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

% Assigns each node in the grid to the closest centroid

idx\_2\_region = kmeans(x\_grid,3,'MaxIter',1,'Start',c);

**Results:**

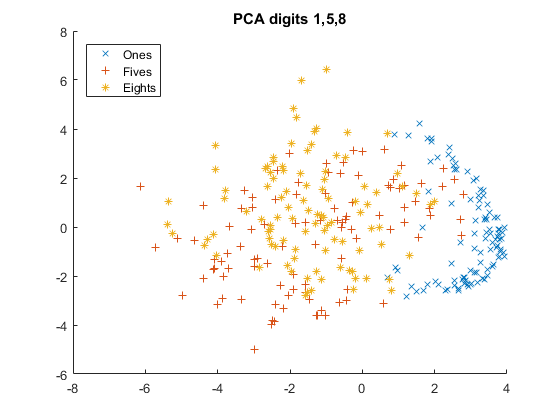


Figure 1: data after PCA projection

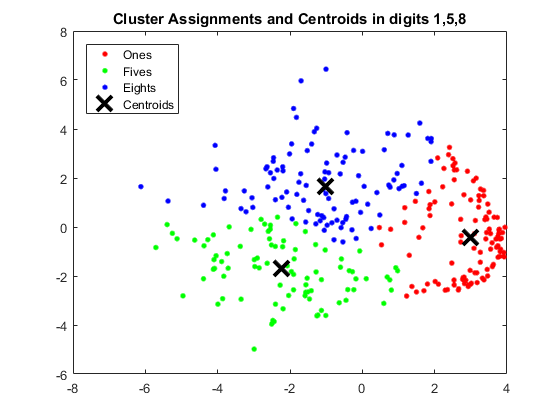


Figure 2: Data after PCA projection with Centroids

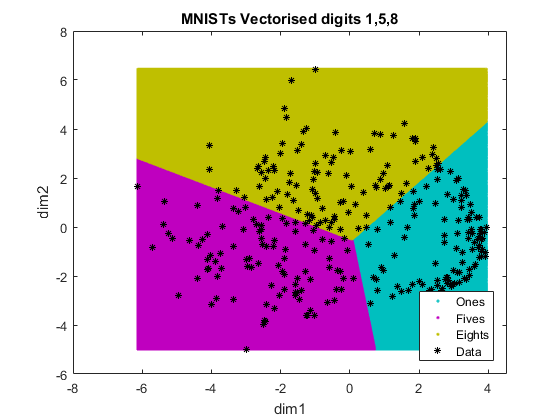


Figure 3: a more visualised plot of PCA projection on data, divided by regions for each of the classes

From the figures above we can see how PCA made it possible to classify all the data we had in the matrix. We can clearly see that due to classification some of the data got lost. For example, from the first figure, we can see that “Fives” data is spread all over the figure, but then classifier choses data “Eights” over “Fives”. Same for projection of “Ones”, if we compare figure one with figure 2 and figure 3, we can see that there is more “ones” data shown on figure 2 and figure 3 than in original projection on figure 1. I believe this happens in the result of implication of PCA discussed above. However, in respect of PCA, I believe that good results were achieved, especially if taking in mind, that it performed dimensionality reduction on data from 100 dimensions into 2 dimensions.

**Q2. Linear Discriminant Analysis (LDA)**

From what I learned, there are two types of Linear Discriminant Analysis applications. These are:

***Linear Discriminant Analysis on 2-class classification problem.***

The summary of steps for 2-class LDA:

1. Calculating means for each class
2. Calculating covariance matrices for each class
3. Calculating “within class scatter matrix” by summing up class covariance matrices
4. Computing Projection Vector by multiplying inverted “within class scatter matrix” on transposed difference of class means

***Linear Discriminant Analysis on multi-class classification problem.***

The summary of steps for n-class LDA:

1. Calculating means for each class and then calculate the mean of the class means
2. Calculating covariance matrices for each of the classes
3. Calculating “within class scatter matrix” by summing up all the covariance matrices
4. Calculating “between class scatter matrix”:
   1. for each of the classes:  
      multiplying number of size of the class by the difference of class means and mean of all class means, multiplied by the difference of class means and mean of all class means transposed.
   2. Summing up the results of above calculation from each class
5. Computing Eigen vectors (projection vectors) and Eigen values from inverted “within class scatter mask” multiplied by between class scatter mask
6. Calculating score by multiplying the matrix containing all data by specific Eigen vector’s values

If comparing LDA to PCA it is essential to mention that unlike PCA, LDA can pick up class labels, it can be used for jobs that PCA is not capable of doing. However, unlike expectations LDA is not guaranteed to perform better than PCA.

For the task allocated, I had to use 3-class LDA. I followed the steps I mentioned above to produce the 3-class LDA, but got stuck on the problem of calculating inverted “within-class scatter matrix”. Due to the data we have, the result of summing up class covariance matrices produced a matrix that had many zero values in it. It is obvious that it makes no sense to attempt to calculate inverse of 0 value. I solved the problem by adding bias to “within class scatter matrix” (I have picked some small number and to the “within-class scatter matrix“) in attempt to produce the matrix that would be suitable for inverting:

%addding small number to avoid inv on zero

dc=0.00001\*eye(size(sw));

sw\_new=sw+dc;

inverted\_SW=inv(sw\_new)\*sb;

It finally made it possible to calculate projection vectors and plot the results:

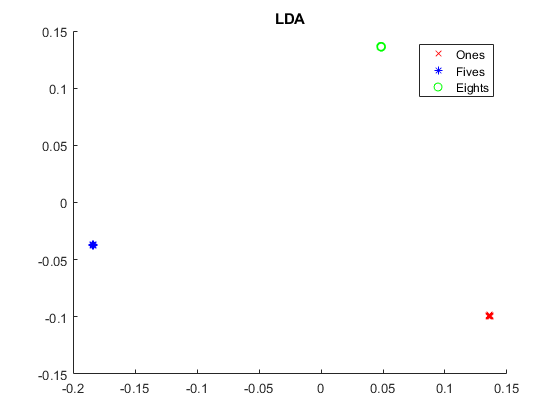
% computing the projection vectors:

[v1,d] = eig(inverted\_SW);

% calculating score

score = (all\_data\*v1(:,1:2));

**Results:**



From what we can see on the figure, LDA managed to classify classes and distinguishes the data. Compared to PCA, LDA treats data in slightly different way. By looking at PCA centroids, I believe that PCA performed better than LDA in this particular problem. Also, I believe that there is a bug somewhere in my LDA implementation, the results are not what I expected.

**Q3. Two-class problem classification**

For this question, I had to separate data for “Fives” from the rest of the data and treat it as two-class classification problem.

**Support Vector Machine (SVM)**

**SVM with linear kernel**

The SVM classifier with linear network was developed for use with libsvm library (version 3.2.2) that was provided. The following steps were replicated in order to develop SVM with linear kernel:

1. Combining all data samples into one
2. Scaling the data in each column into the range [0,1]
3. Computing data labels
4. Preparing vectors for “fives” against the rest (‘1’ meaning “fives”, ‘0’ the rest of the data)
5. Performing data partition
6. Getting the index for training and test samples
7. Creating training label ground truth and creating training data
8. Creating testing label ground truth and creating test data
9. Training SVM using linear kernel
10. Classifying

%combine all data

all\_data = [digit\_one digit\_five digit\_eight];

all\_data\_label = [];

%scale the data in each column of all data into the range of [0, 1]

for i=1:max(size(all\_data,2))

t = all\_data(:,i);

t = (t-min(t(:)))./(max(t(:))-min(t(:)));

all\_data(:,i) = t;

end

whos;

%get labels (digit one: 1-100, digit five 101-200, digit eight 201-300)

for k=1:size(all\_data,2)

if k<= 100

all\_data\_label = [all\_data\_label;1];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;5];

end

if k > 200

all\_data\_label = [all\_data\_label;8];

end

end

% preparing the vectors for 'fives' against the rest

a = zeros(size(all\_data\_label));

for i=1:max(size(a))

a(i) = isequal(all\_data\_label(i),5);

% in the vector a, 1 means 'fives'; and ‘0’ means rest

end

cvo = cvpartition(a,'k',2);

% get the index of training samples

trIdx = cvo.training(1);

% get the index of the test samples

teIdx = cvo.test(1);

% creating the training label ground truth

training\_label\_vector = a(trIdx);

%creating the training data

training\_instance\_matrix = all\_data(trIdx,:);

% creating the testing label ground truth

test\_label\_vector = a(teIdx);

% creating the test data

test\_instance\_matrix = all\_data(teIdx,:);

%training SVM using a linear kernel

model = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 0');

%classification on the test data:

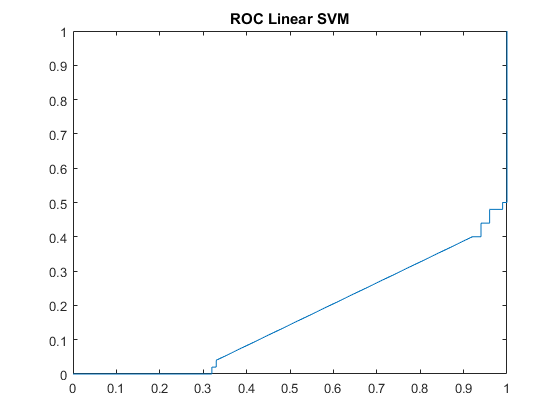
[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, model);

**Results:**

I have ran the script around 20 times in attempt to find the best results. I am providing the result set with highest accuracy percentage that was achieved after the number of script executions. For the results I consider the best, optimization finished at 140th iteration providing the following:

|  |
| --- |
| **nu** = 0.504592 |
| **obj** = -65.854545, **rho** = -0.999837 |
| **nSV** = 88, **nBSV** = 69 |
| **Total nSV** = 88 |
| **Accuracy** **= 82.6667% (124/150)** (classification) |

I have also plotted the Receiver operating characteristic (ROC) curve for this result



From what I can see from results, the classification job was very successful. I expected SVM with Linear kernel to provide lower accuracy. What is interesting that although the classification accuracy values was high, it was quite unstable and vary. Through the number of times I ran classification job it varied from 73% to 82.7%.

**SVM with RBF (radial basis function) kernel**

The steps to replicate SVM classifier with RBF kernel are exactly the same as for SVM with linear kernel, the only difference appear in the training technique:

%training SVM using a linear kernel

modelDef = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.07');

Due variable G being available this time, I decided to change it number of times and produce number of model. I have picked different values for “g” (0.07,0.01,0.20,0.50,0.91) to see how it changes the result. The best results I have achieved after running the classification job for around 20 times:

|  |  |
| --- | --- |
| G = 0.07 (default value) *finished at iter 131* | G = 0.01 (smaller value) *finished at iter 70* |
| **nu** = 0.640605 | **nu** = 0.666667 |
| **obj** = -94.462968, **rho** = -0.999798 | **obj** = -99.967466, **rho** = -0.999836 |
| **nSV** = 109, **nBSV** = 93 | **nSV** =111, **nBSV** = 97 |
| **Total nSV** = 109 | **Total nSV** = 111 |
| **Accuracy** **= 66% (99/150)** | **Accuracy** **= 66.6667% (100/150)** |

|  |  |
| --- | --- |
| G = 0.20 (higher value) *finished at iter 152* | G = 0.50 (even higher value) *finished at iter 132* |
| **nu** = 0.599665 | **nu** = 0.526810 |
| **obj** = -85.704811, **rho** = -0.999760 | **obj** = -75.200986, **rho** = -0.999928 |
| **nSV** =108, **nBSV** = 86 | **nSV** =96, **nBSV** = 77 |
| **Total nSV** = 108 | **Total nSV** = 96 |
| **Accuracy** **= 67.3333% (101/150)** | **Accuracy** **= 72% (108/150)** |

|  |
| --- |
| G = 0.91 (highest value) *finished at iter 126* |
| **nu** = 0.480768 |
| **obj** = -67.443118, **rho** = -0.999913 |
| **nSV** =89, **nBSV** = 70 |
| **Total nSV** = 89 |
| **Accuracy** **= 76.6667% (115/150)** |

What I figured out through the amount of times I ran the classification job was rather interesting for me. By setting up “g” value to or 0.01, 0.07, 0.20 the accuracy was lover then the rest, but more stable. During around 20 times of running classification job it the accuracy suffered from small changes (0.5%-2%). However, by increasing “g” to 0.9 I have managed to get the highest accuracy, but the accuracy became very unstable. During around 20 times of running classification job the accuracy was suffering from large changes (5%-10%).

**Comparing SVM with RBF to SVM with Linear:**

From what it is possible to see on the results of SVM with linear kernel and SVM with RBF kernel, the SVM with linear kernel provides higher accuracy (10% more accurate than best result of SVM with RBF kernel). At the same time, at the higher result it is much more stable. I believe it is possible to imrove the result of SVM with RBF kernel by finding the best size combination of training/testing set size.

**ROC curves for the results of SVM with RVF kernel:**

|  |  |
| --- | --- |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g01.png  Figure 4: ROC plot SVM with RBF kernel (g=0.01) | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g02.png  Figure 5: ROC plot SVM with RBF kernel (g=0.20) |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g05.png  Figure 6: ROC plot SVM with RBF kernel (g=0.50) | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g07.png  Figure 7: ROC plot SVM with RBF kernel (g=0.07) |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g091.png  Figure 8: ROC plot SVM with RBF kernel (g=0.91) |

**Neural Network classifier**

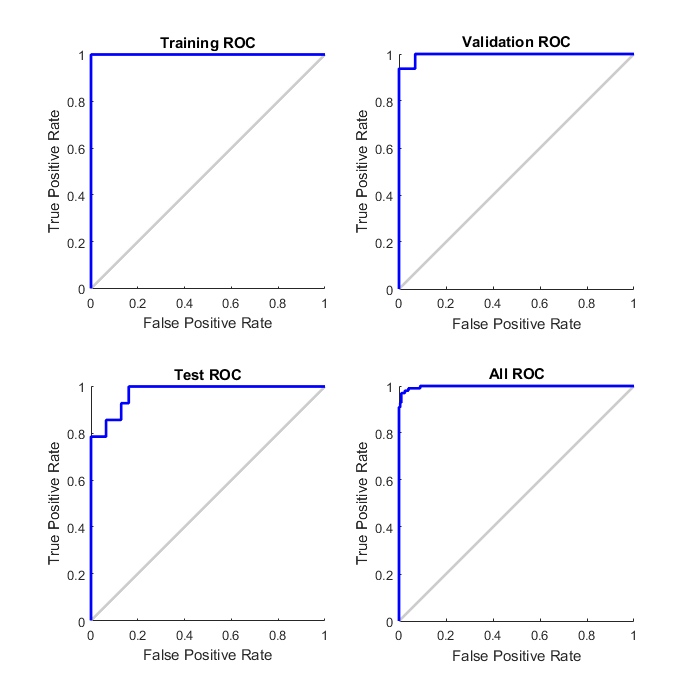
For this task I have used Matlab’s build in “Neural Net Pattern Recognition” tool (nprtool). I have created a neural network with the following samples: 70% training, 15% validation, 15% testing. I have used 1 hidden layer. I have never applied 5-fold cross validation setting to the data due to lack of time. The correct process would be to split the data into 5 chunks and perform training 5 times in attempt to get better results.

**Results:**

In attempt to find the best results, 5 same neural network classifiers were created and trained. The best performance I could achieve was 81%. This is smaller better performance than SVM with Linear Kernel, but worse than maximum accuracy I could achieve with SVM with RBF kernel.

|  |  |
| --- | --- |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\diagram.png  Figure 9: Neural Network diagram | C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\results.png  Figure 10: Neural network training results |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\training_state.png  Figure 11: Neural network training state | C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\error_histogram.png  Figure 12: Neural network error histogram |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\performance.png  Figure 13: Neural network performance | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\NeuralNet\confusion.png  Figure 14: Neural network confusion matrix |

**ROC curves for Neural Network:**



**Comparing results in Q3**

The parameter we would be interested in is ROC curve plotted from results of performing classifier testing.

**Instructions on running the code:**

**Code:**