EE559: Project Report

Human Activity Recognition Using Smartphones Data Set

Naveen Sasalu Rajashekharappa
sasalura@usc.edu
Individual Project
No prior or parallel work
4-May-2015

Table of Contents

Abstract

The project looks at the multi-class classification of **Human Activity Recognition Using Smartphones Data Set** . This project aims at providing the best classification results possible for the dataset and also illustrates the use of *diverse pattern recognition techniques* studied in class. *Parameter selection* and *dimensionality reduction techniques* are demonstrated for each classifier. Both Distribution free classifiers and Statistical Classifiers are applied on the dataset. The dataset is not linearly separable and hence non-linear classifier like SVM with RBF kernel will perform better with right selection of parameters. The results from all the classifiers are summarized in table.

Dataset description

Human Activity Recognition Using Smartphones Data Set

> “The dataset contains data from experiments carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.
>
> The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.”

Link to dataset -
http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

Some information related to dataset
Number of data samples = **7352** (Training) + **2947** (Testing+Validations)
Number of features = **561**
Number of classes = **6**

There are **no missing or categorical data** . There was **no normalization or data formatting** done. The features are already normalized in the range -1 to 1.

Language, Toolboxes and Issues

The project is completly coded in **MATLAB** .
- **Random class assignment** : Own code
- **Min Dist to Class Means classifier** : Own code
- **Perceptron** : *prtools* (toolbox) : `perlc` (function name)
- **LS** : *prtools* (toolbox) : `fisherc` (function name)
- **SVM** : *libsvm* (toolbox) : `svmtrain` , `svmpred` (function name)
- **KNN** : *prtools* (toolbox) : `knnc` (function name)
- **naive Bayes** : *prtools* (toolbox) : `naivebc` (function name)
- **LDC** : *prtools* (toolbox) : `ldc` (function name)
- **FLD** : *prtools* (toolbox) : `fisherm` (function name)
- **PCA** : online code - link given in reference
- **Parameter selection** : Own code
- **Validaton framework** : Own code

Issues - The project was planned to be coded in **R** and most of the modules were already coded. But *Validation* and *Parameter selection* required loops in the order of 100, and **R** being highly unoptimized language started taking hours to perform single validation. Hence all the modules were re-coded in **MATLAB** which is much faster. Also the **3D plots** in MATLAB took quite a time to get correct viewing angle to see the curvature.

Summary of Classifier performance

The table below summarizes the performance of each classifier used in this project

	Correct Classification % on Test data				
Classifier	**Baseline**	**Best (Excluding FLD)**	**PCA (Dimension)**	**FLD (Dimension - 5)**	**Parameter Selection**
Min Dist to Class Means	84.52	84.52	84.50 (306)	96.17	NA
Perceptron	90.36	94.68	94.68 (153)	96.31	η=0.46
LS	96.20	96.61	96.61 (255)	96.31	NA
SVM	94.02	**96.68**	96.34 (408)	96.58	Kernel = RBF, C = 32768, gamma = 1.22e-04

KNN	87.85	90.8	90.8 (306)	96.41	K = 8
naive Bayes	86.05	87.04	85.75 (102)	95.70	N=4
LDC	83.06	95.90	95.76 (459)	96.20	R=0, S=0.2

Distribution free Classification

Ad Hoc Methods

Random Class assignment

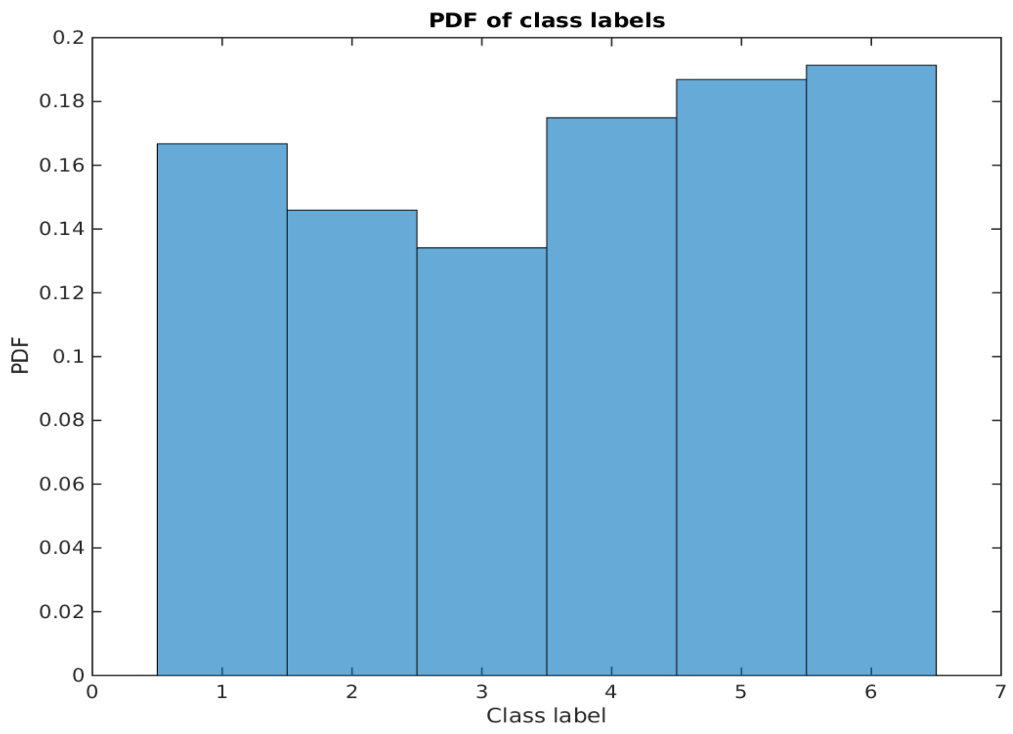
Random class assignment is one of the ad-hoc methods of classifying data without the use of any classifier. Here we do not use training data. We just generate a random number for the class label and assign it to a sample.

Without Priors

Here we generate random numbers using uniform distribution, i.e, all the class labels will have equal probability of occurrence.
This method resulted in **16.89%** correct classification. This shows that we need a trained classifier for good classification.

With Priors

Here we generate random numbers using probability of occurrence of each class in the training data as a PDF.
The PDF of class labels is as given below



Class prior probabilities

This method resulted in **17.12%** correct classification. There is a very slight improvement compared to without prior method. But we still need a trained classifier for good classification.

Minimum Distance to class means classifier

The discriminant function is given by

$$g_k(\mathbf{x}) = -\|\mathbf{x} - \mathbf{m_k}\|$$

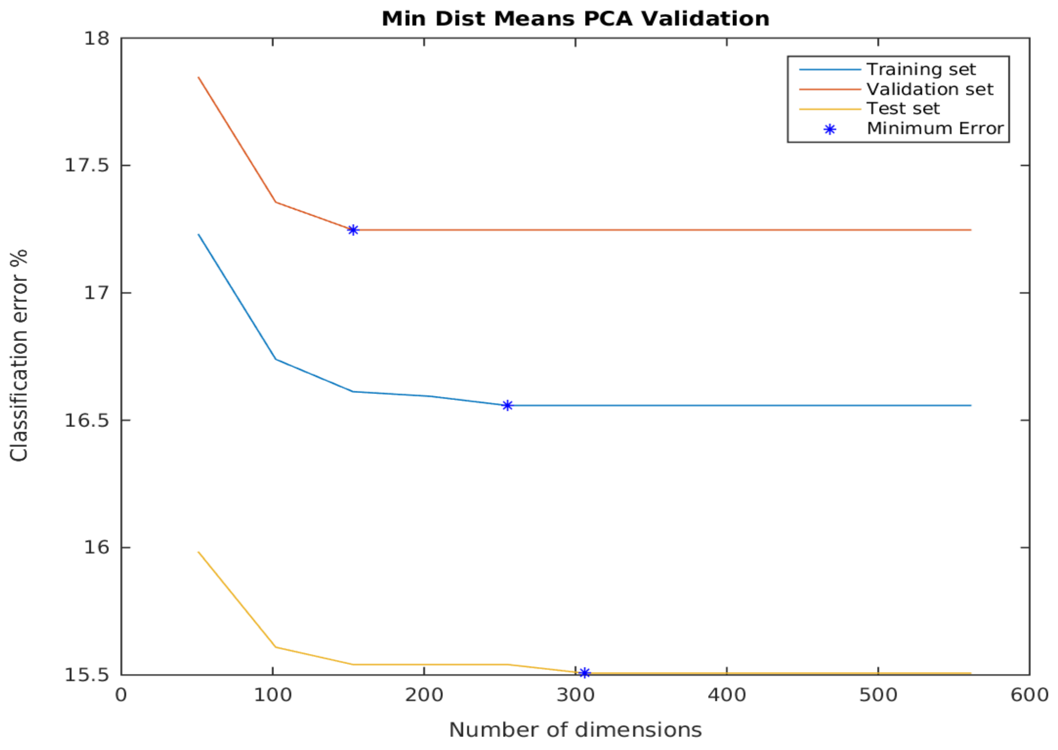
Where $\mathbf{m_k}$ is the mean of the training samples of class **k**
The multiclass classification is done using one-vs-all method given as below

$$g_k(\mathbf{x}) > g_j(\mathbf{x}), \forall j \neq i, \mathbf{x} \in S_k$$

The baseline performance of this classifier is as below
Training dataset = **83.29%** , Testing dataset= **84.52%**
The performance is much better that the random class assignment and it is a good benchmark for the next classifiers.

Dimensionality reduction

We know that our dataset has 561 features, and this huge number of features can result in over-fitting. So we can try dimensionality reduction techniques to see if they increase the performance. One of the dimensionality reduction technique is PCA(Principal Component Analysis). Below is the plot of number of dimensions vs classification error for minimum distance to class means classifier.



Min distance to class means PCA

Here the training dataset is randomly divided into *training(80%)* and *validation(20%)* dataset. The PCA is done from **51** dimensions to **561** in steps of **51** dimensions. The plot shows the classification error for training, validation and test data. Conventional validation plot does not include results from test data, but the performance of test data is included in plot to get comparative analysis with respect to training and validation data.
We can see from the plot that after **300** dimension, the test error remains the same. This means that we just need **300** diemensions for classification. Also in general the test and validation error increases as we increase the dimention, but in this case since the new dimention added is not significantly changing the class means, it is not affecting the classification.

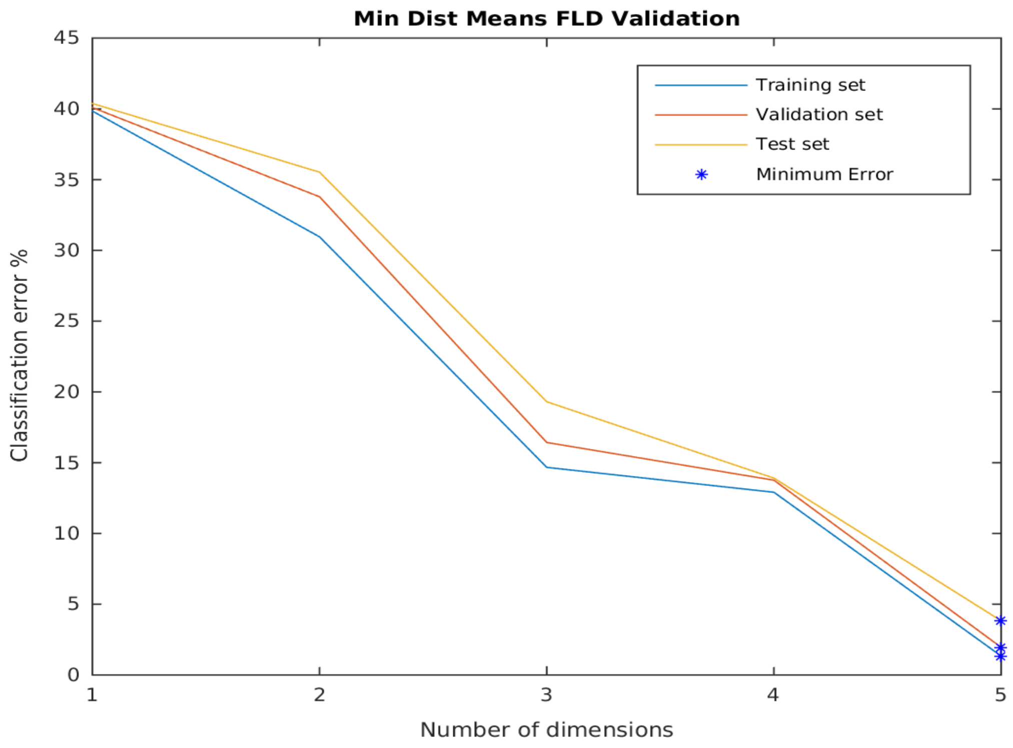
The minimum classification error for each dataset is as given below

Training	Validation	Test
16.55%	17.24%	15.50%

Another dimensionality reduction technique which uses class labels for mapping is Fisher mapping or LDA(Linear discriminant analysis).

> ” LDA is a mapping of the labeled dataset A onto an N-dimensional linear subspace such that it maximizes the the between scatter over the within scatter (also called the Fisher mapping]). Note that N should be less than the number of classes in A”

We should observe that LDA takes class labels into consideration while diemnsionality reduction. Hence it will perform better than PCA.
Below is the validation plot using LDA or FLD



Min distance to class means FLD

In general FLD performs best for N-1 dimentions, and we observe the same in this case for N-1 = 6-1 = 5.
The minimum classification error using FLD for each dataset is as given below

Training	Validation	Test
1.34%	1.95%	3.83%

We can see that FLD has done a very good job of dimensionality reduction and hence we are getting very good results when the dimension = 5 (number of class -1)

Criterion Function Methods

Perceptron Learning

The multiclass perceptron update for wrong classification is given as below

$$\mathbf{w}^k(i+1) = w^k(i) + \eta(i)\mathbf{x}^k$$

$$l = \arg\max_{j \neq k}\{g_j(\mathbf{x}^k)\}$$

$$\mathbf{w}^l(i+1) = w^l(i) + \eta(i)\mathbf{x}^k$$

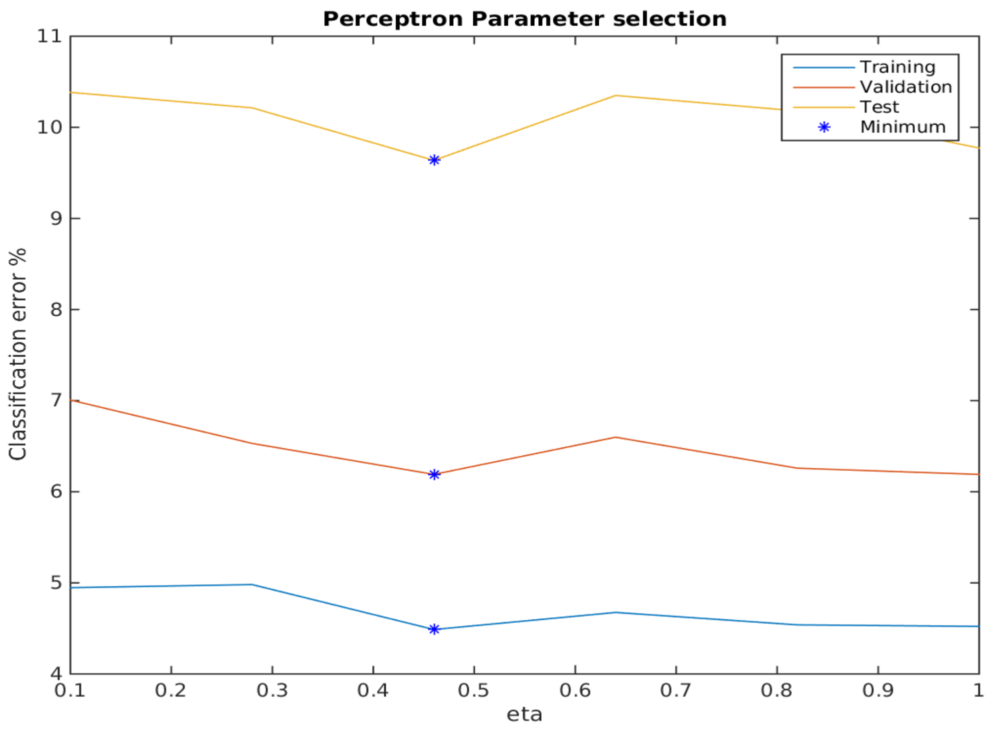
$$\mathbf{w}^m(i+1) = w^m(i) + \eta(i)\mathbf{x}^k, \forall m \neq l, k$$

where k is the class $\mathbf{x}$ belongs to and l is the class it is wrongly classified into.
And the multi-class perceptron update is based on maximal value method.

The baseline performance of correct classification with 561 features with $\eta = 0.1$ and number of iterations = 100 is as given below.
Training dataset = **94.96%** , Test dataset = **90.36%**
We can see that the baseline performance is pretty good compared to Minimum distance to means classifier which is expected.

Perceptron Parameter selection

Here we fix the number of iterations to 100 and run the perceptron for different values of η . The plot is as below

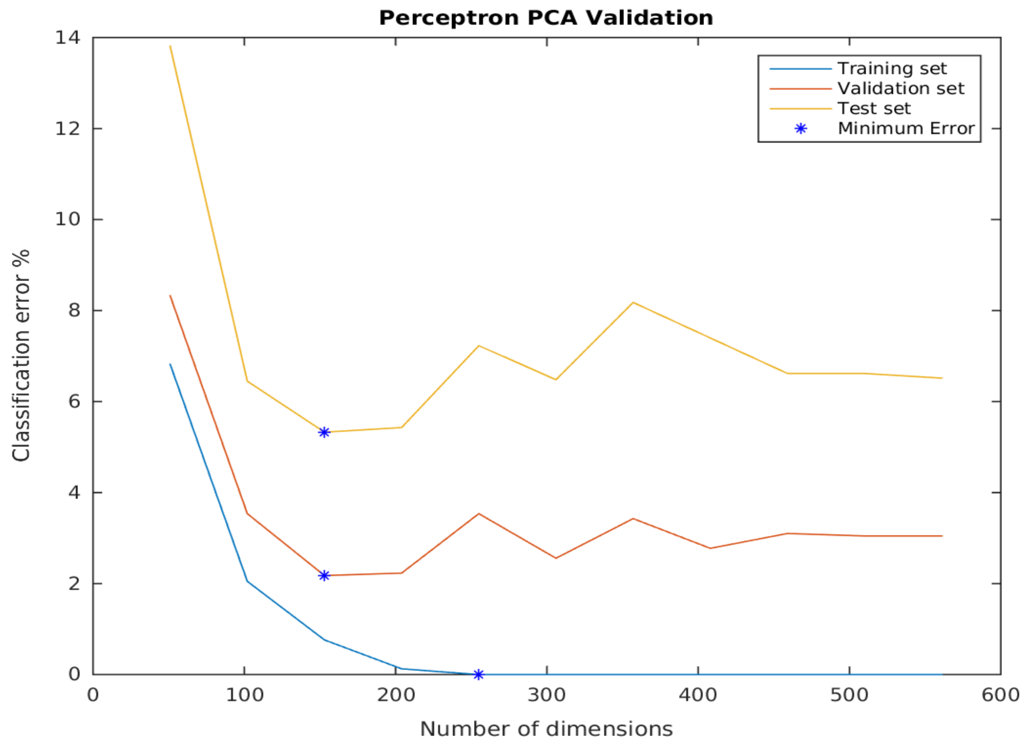


Perceptron Parameter Selection

From the graph we can observer that for $\eta = 0.46$ we get the least error in classification for 100 iterations. Hence we choose this η for the next experiments.

Dimensionality reduction

Below is the PCA validation plot of Perceptron



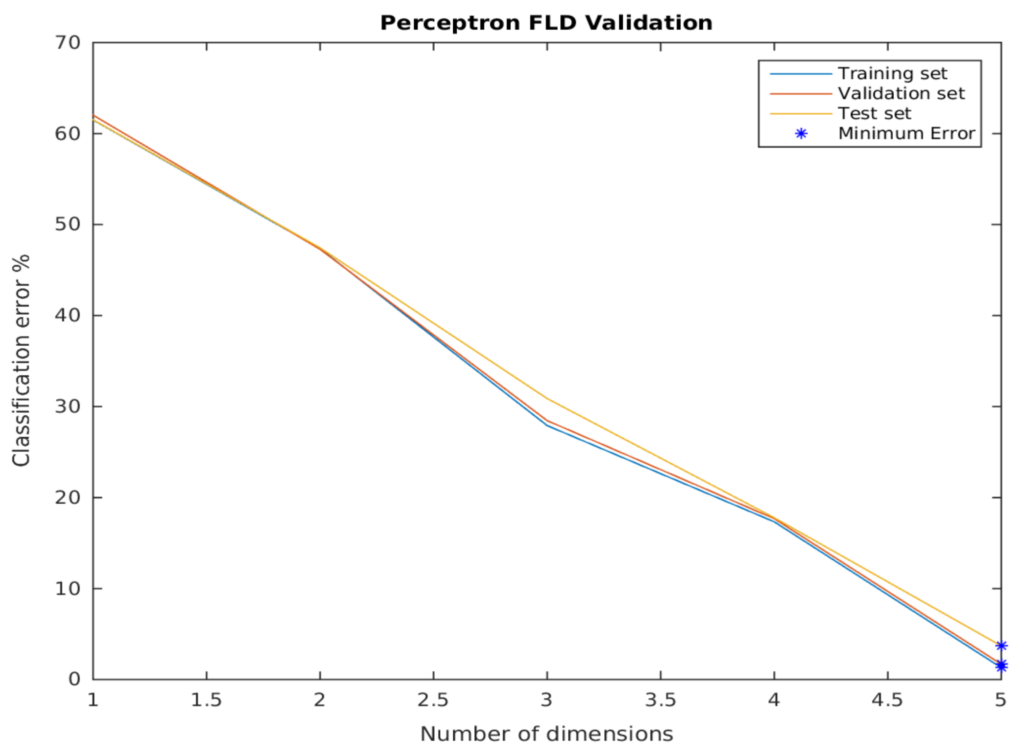
Perceptron PCA Validation

We can see the classic validation curve above, where training and validation curves increase while training decreases as we increase the dimentions after the minimum test and validation error is acheived. This is the clear case of overfitting of data due to higher dimentions and the ideal dimentions for this case is **153**

The minimum classification error for each dataset is as given below

Training	**Validation**	**Test**
0%	2.17%	5.32%

Below is the FLD validation plot of Perceptron



Perceptron FLD Validation

The minimum classification error for each dataset is as given below

Training	Validation	Test
1.30%	1.68%	3.69%

We get better results with FLD than PCA because FLD uses class labels for dimensionality reduction and hence tries to increase the variance along the direction perpendicular to the class separating boundary.

Least Square Method

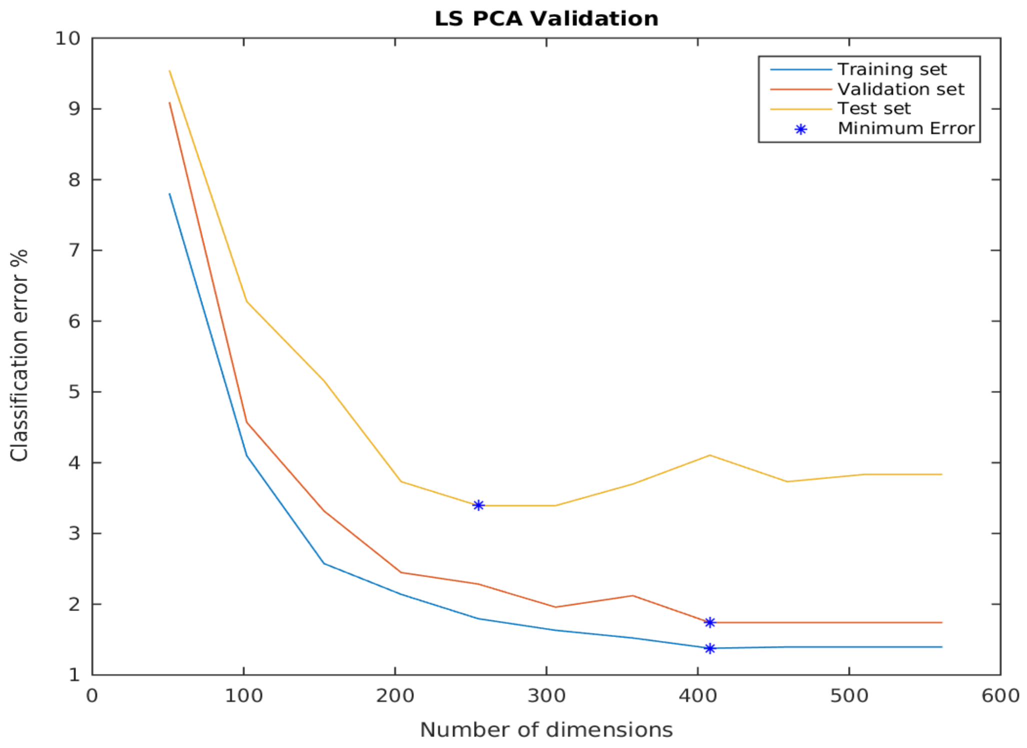
Least square or pseudo inverse learning is given by

$$\mathbf{w} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{b}$$

where $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T$ is called the pseudo inverse of $\mathbf{x}$ and $g(\mathbf{x}) = \mathbf{x}^T\mathbf{x} > 0$ for correct classification.
For multiclass classification, maximal value method is used.
The baseline correct classification % of LS method is as given below
Training dataset = **98.62%** , Testing dataset = **96.20%**

Dimensionality reduction

PCA validation for LS method is as below



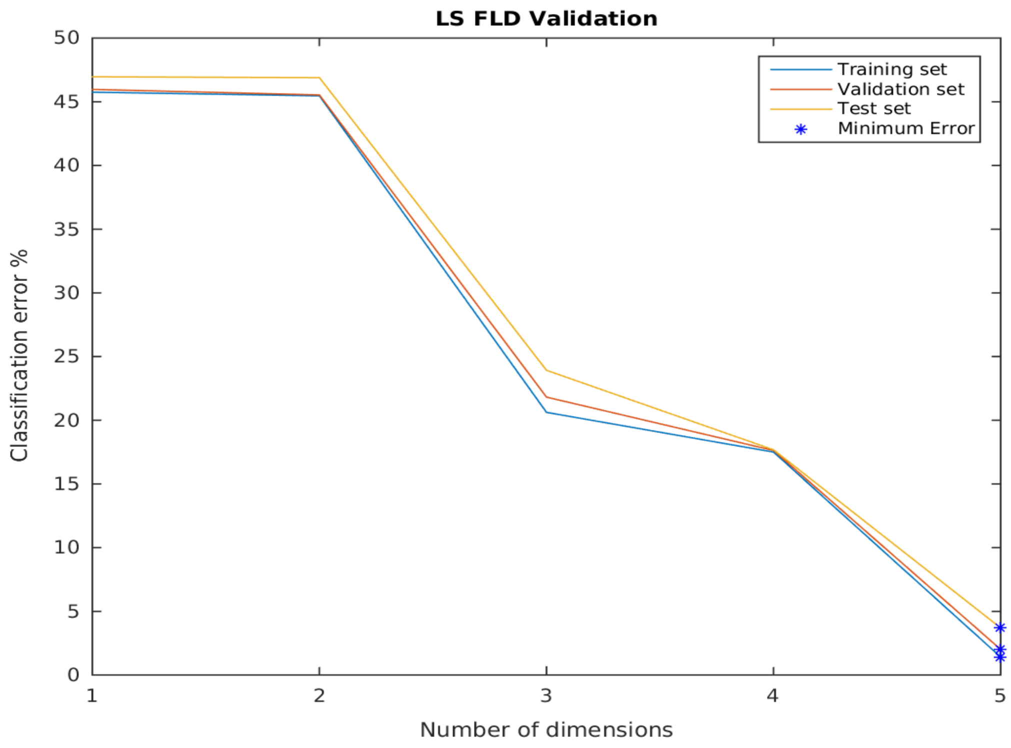
LS PCA Validation

The minimum classification error for each dataset is as given below

Training	**Validation**	**Test**
1.38%	1.74%	3.39%

We can see that we get minimum classification error on test data using **255** dimensions.

FLD validation for LS method is as below



LS FLD Validation

The minimum classification error for each dataset is as given below

Training	Validation	Test
1.41%	2.01%	3.69%

SVM classification

The svm algorithm **C-Support Vector Classification** used in this project is defined as below

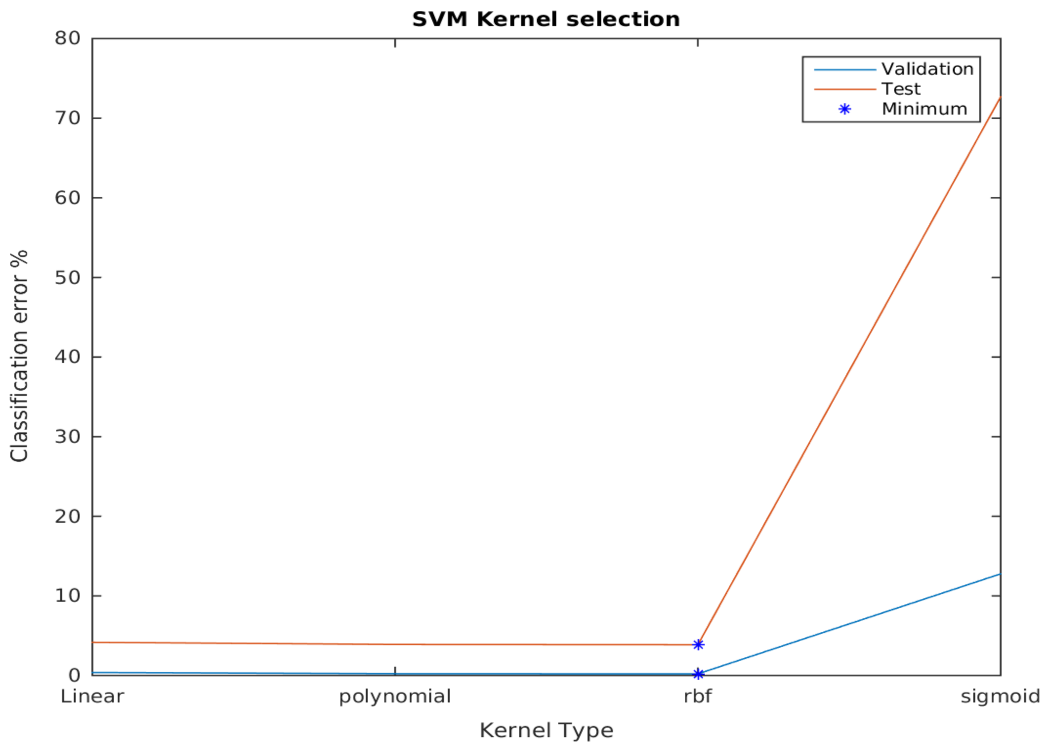
> “Given training vectors $\mathbf{x}_i \in R^n, i = 1, \ldots, l,$ in two classes, and an indicator vector
> $\mathbf{y} \in R^l$
> such that $y_i \in \{1, -1\}$, C-SVC (Boser et al., 1992; Cortes and Vapnik, 1995) solves the following primal optimization problem”
>
> $$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - \mathbf{e}^T\alpha$$
>
> Subject to
>
> $$\mathbf{y}^T\alpha = 0,$$
>
> $$0 < \alpha_i < C, i = 1, \ldots, l,$$
>
> where $\mathbf{e} = [1, \ldots, 1]^T$ is the vector of all ones, Q is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is the kernel

> function.
>
> $$\mathbf{w} = \sum_{i=1}^{l} y_i \alpha_i \phi(\mathbf{x}_i)$$

The baseline correct classification of SVM classifier with 561 feature using **linear** kernel function and **cost c=1** is as below
Training dataset = **95.55%** , Testing dataset = **94.02%**

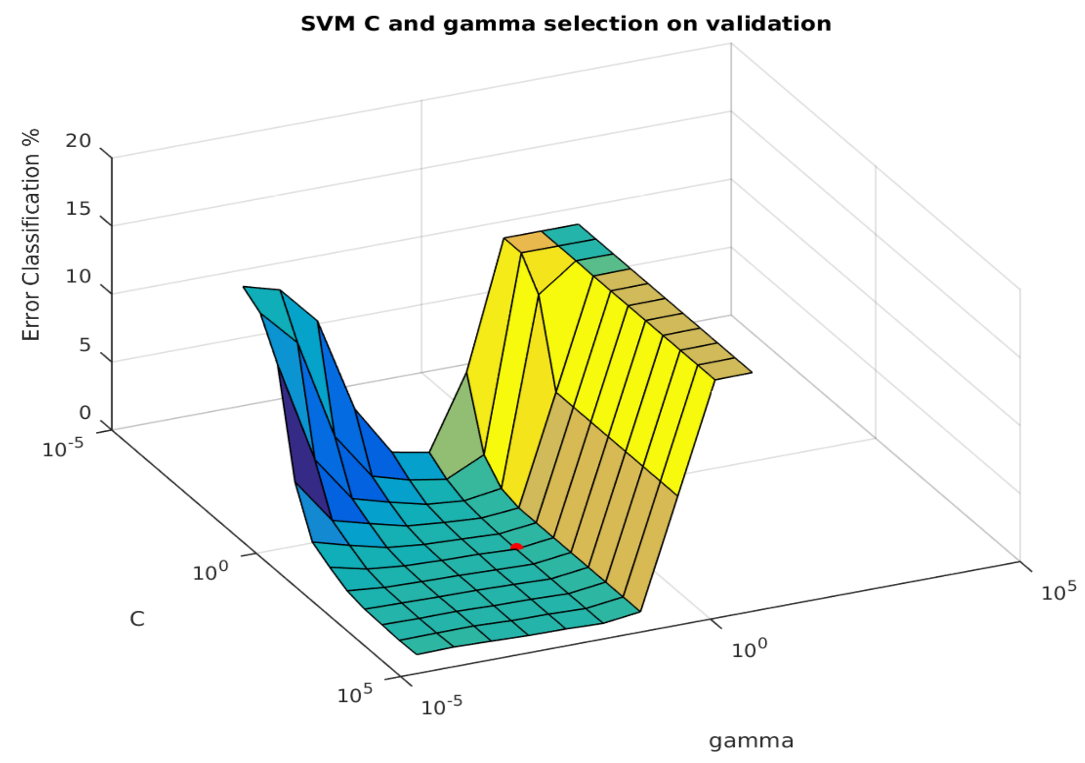
SVM Parameter selection

The performance of different kernels for $C = 1000$ is as below



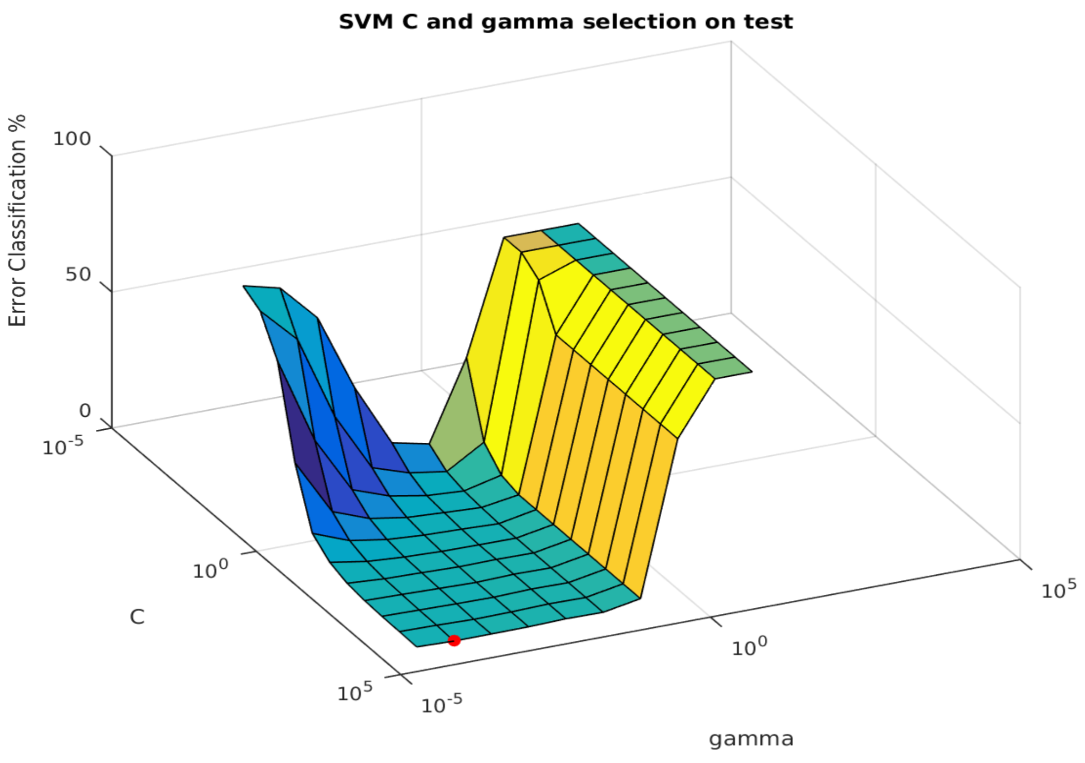
SVM kernel Selection

From the figure we can see that we get minimum error for RBF kernel.
After deciding on RBF kernel, we need to find the best C and *gamma* parameters. This is done by performing SVM on grid of C and *gamma* and finding the parameters resulting minimum classification error.
The classification error for validation set on grid of C and gamma is as below



SVM C and gamma selection on validation

The classification error for test set on grid of C and gamma is as below



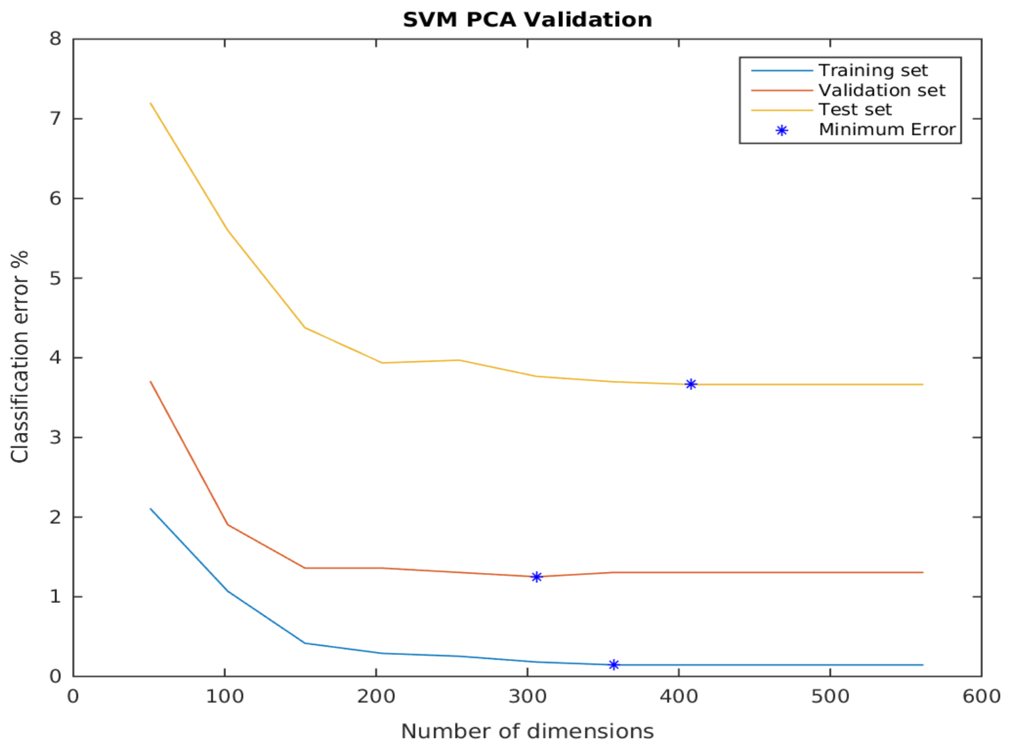
SVM C and gamma selection on test

The minimum error is marked with a **red point** on the plot.
The minimum classification error of **3.32%** occurs for C **= 32768** and ***gamma* = 1.22e-04**
The minimum classification error for each dataset is as given below

Validation	Test
0.08%	3.32%

Dimensionality reduction

PCA validation for SVM method is as below



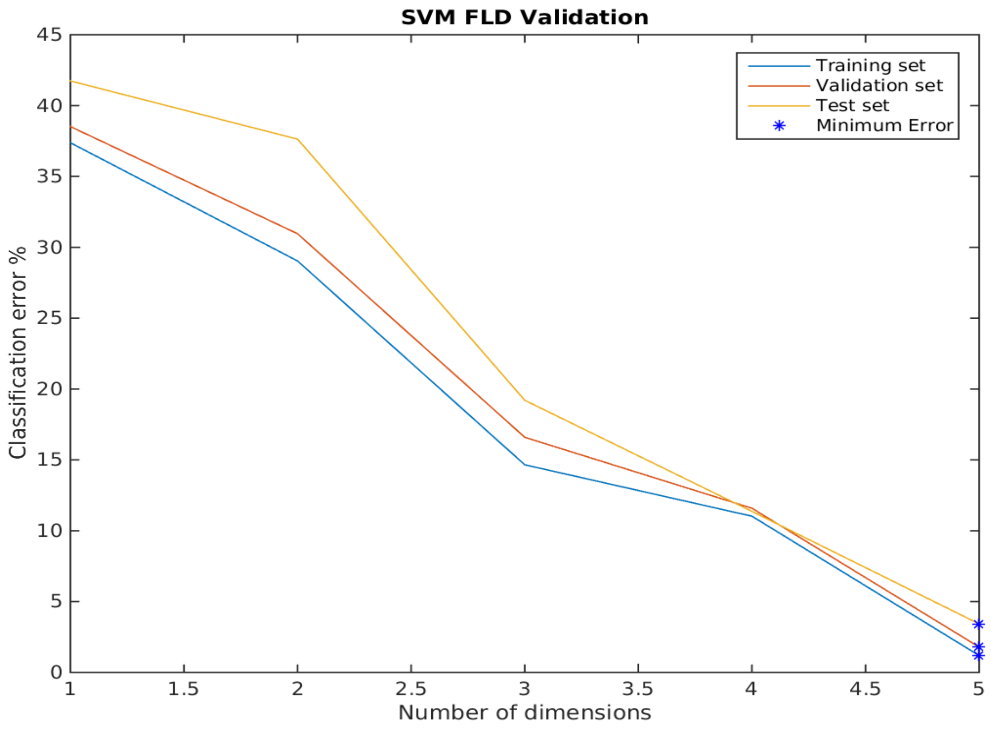
SVM PCA Validation

The minimum classification error for each dataset is as given below

Training	Validation	Test
0.14%	1.25%	3.66%

We can see that we get minimum classification error on test data using **408** dimensions.

FLD validation for SVM method is as below



SVM FLD Validation

The minimum classification error for each dataset is as given below

Training	Validation	Test
1.19%	1.79%	3.42%

Statistical Classification

Nonparametric classification

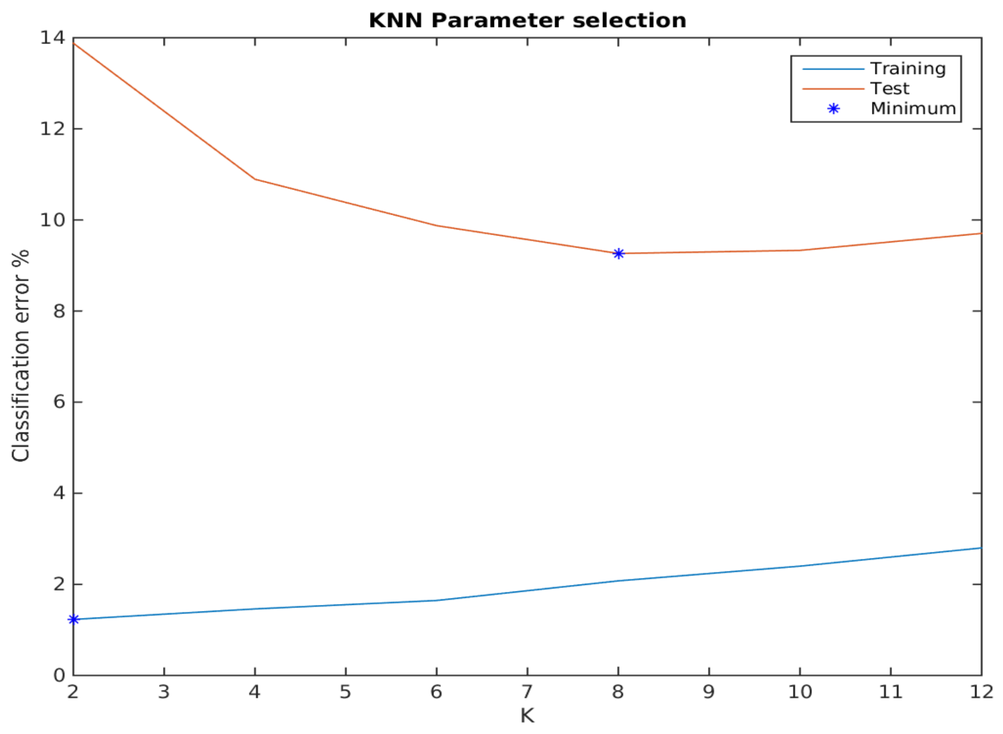
KNN classification

> “In kNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.”

The baseline performance of correct classification with default number of neighbors **K** is as below
Training dataset = **100%** , Test dataset = **87.85%**

KNN Parameter selection

The classification error for different values of number of neighbors **K** is as below

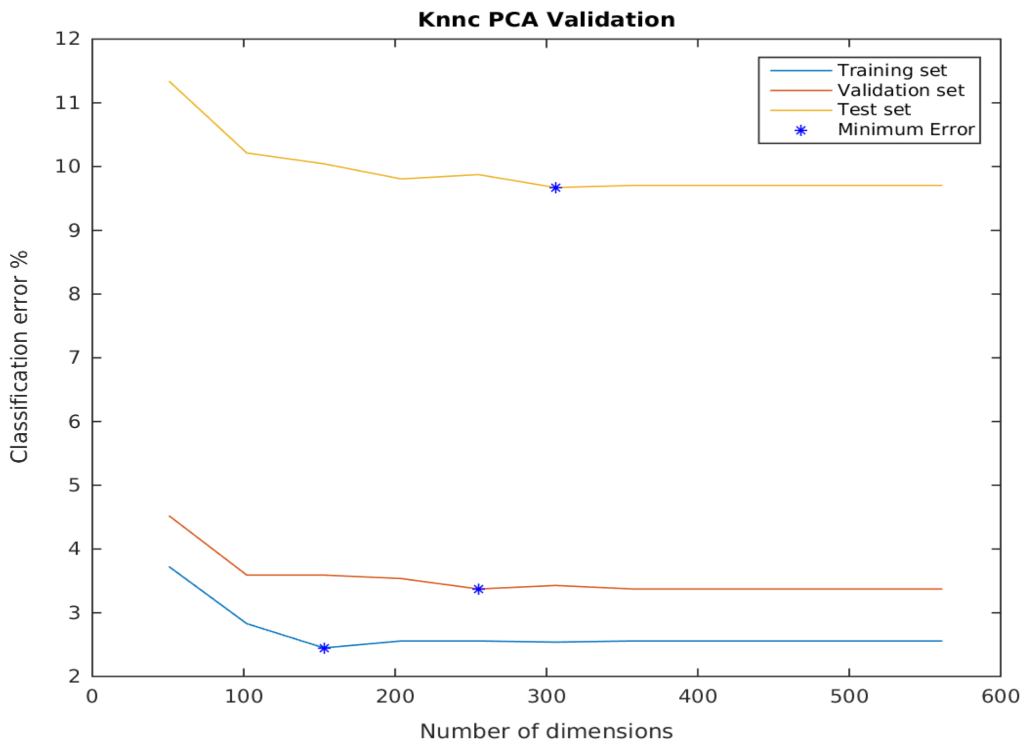


KNN Parameter selection

From the plot we can see that minimum test error of **9.26%** occurs for **K=8** . Hence we will use K=8 for further classification.

Dimensionality reduction

The minimum classification error for **PCA** validation is as below

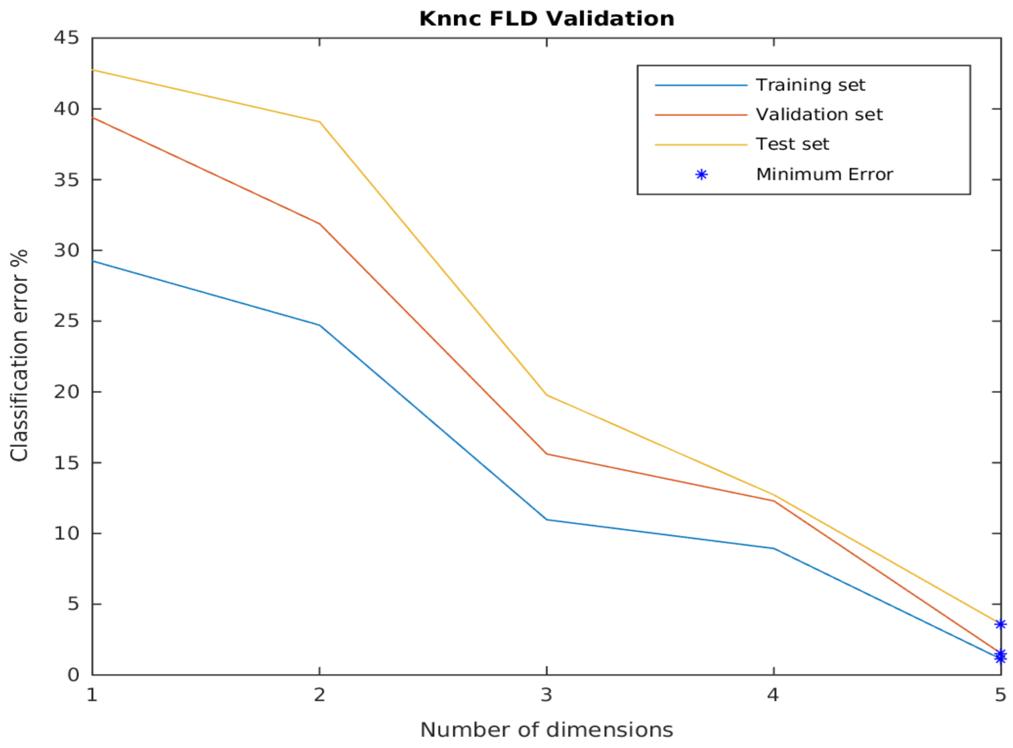


KNN PCA Validation

Training	Validation	Test
2.44%	3.37%	9.20%

The minimum test classification error occurs at **306** dimensions

The minimum classification error for **FLD** validation is as below



KNN FLD Validation

Training	Validation	Test
1.12%	1.52%	3.59%

Naive Bayes Classfier

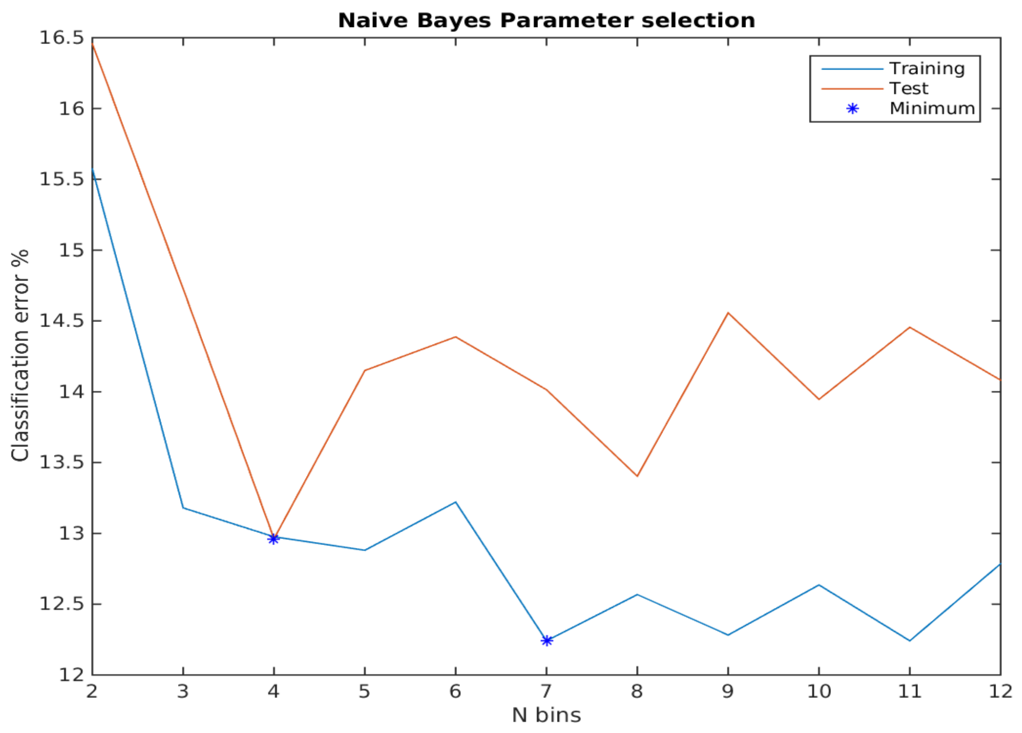
> “The Naive Bayes Classifier estimates for every class and every feature separately. Total class densities are constructed by assuming independency and consequently multiplying the separate feature densities.”
>
> “The default version divides each axis into N bins, counts the number of training examples for each of the classes in each of the bins, and classifies the object to the class that gives maximum posterior probability.”

The baseline performance of correct classification with default number of bins **N=10** is as below

Training dataset = **87.36%** , Test dataset = **86.05%**

Naive Bayes Parameter selection

The classification error for different values of number of bins **N** is as below

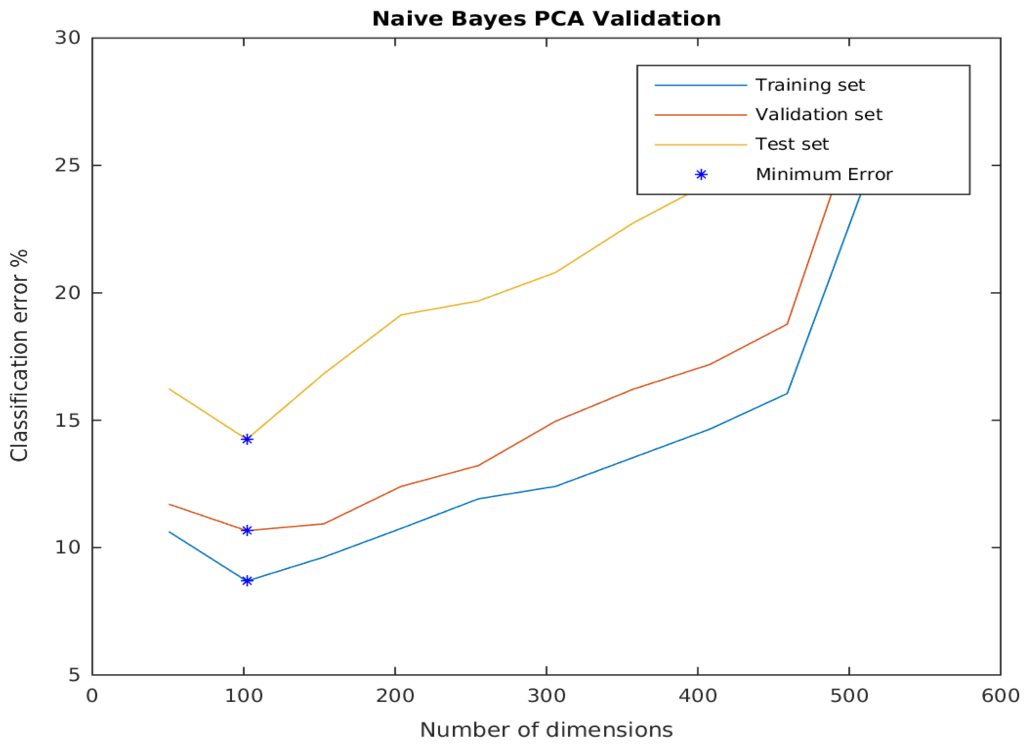


Naive Bayes Parameter Selection

The minimum test classification error of **12.96%** occurs on **N=4** bins. We will be using this value of N for future classification.

Dimensionality Reduction

The minimum classification error for **PCA** validation is as below

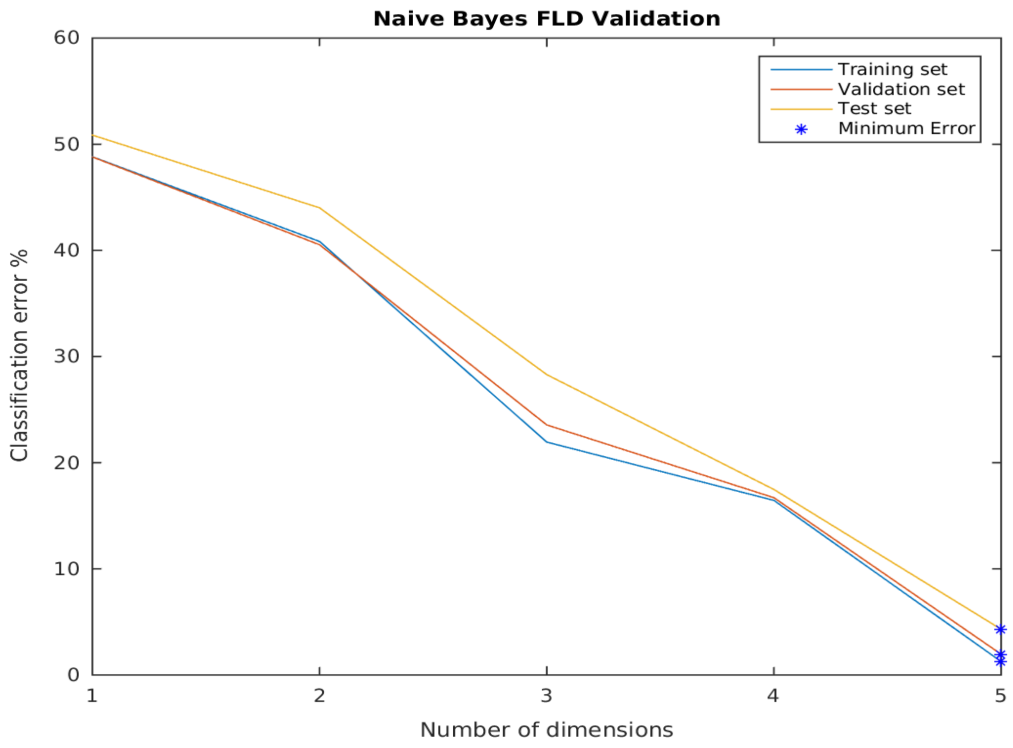


Naive Bayes PCA Validation

Training	Validation	Test
8.68%	10.66%	14.25%

The minimum test classification error occurs at **102** dimensions. Also we can see that the test classification error is more than the previous parameter selection value because here we are dividing training data into training and validation and this affects the training.

The minimum classification error for **FLD** validation is as below



Naive Bayes FLD Validation

Training	Validation	Test
1.26%	1.95%	4.30%

Parametric classification

Linear Bayes Normal Classifier (LDC)

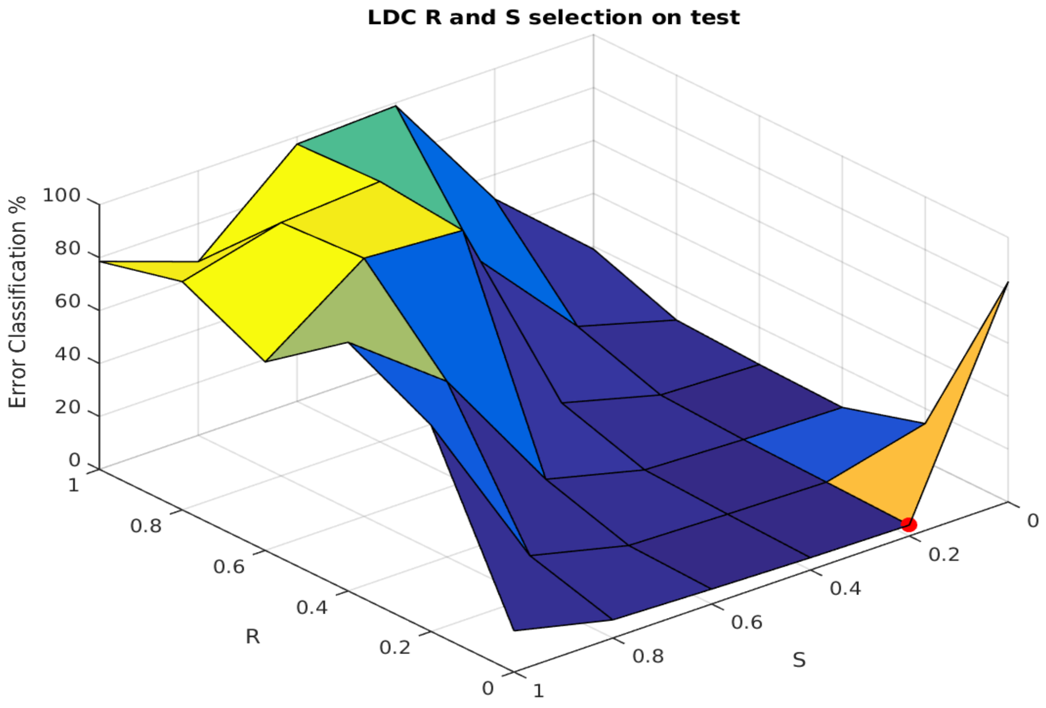
> “Computation of the linear classifier between the classes of the dataset A by assuming normal densities with equal covariance matrices. The joint covariance matrix is the weighted (by a priori probabilities) average of the class covariance matrices. R and S (0 <= R,S <= 1) are regularization parameters used for finding the covariance matrix G by”
>
> $$G = (1 - R - S) * G + R * diag(diag(G)) + S * mean(diag(G)) * eye(size(G, 1))$$
>
> “This covariance matrix is then decomposed as”
>
> $$G = W * W' + sigma^2 * eye(K)$$
>
> “where W is a K x M matrix containing the M leading principal components and $sigma^2$ is the mean of the K-M smallest eigenvalues. The use of soft labels is supported. The classification A*W is computed by NORMAL_MAP.”

The baseline performance of correct classification with default regularization parameters **R** and **S** is as below

Training dataset = **83.22%** , Test dataset = **83.06%**

LDC Parameter selection

The classification error for the grid of **R** and **S** is as below

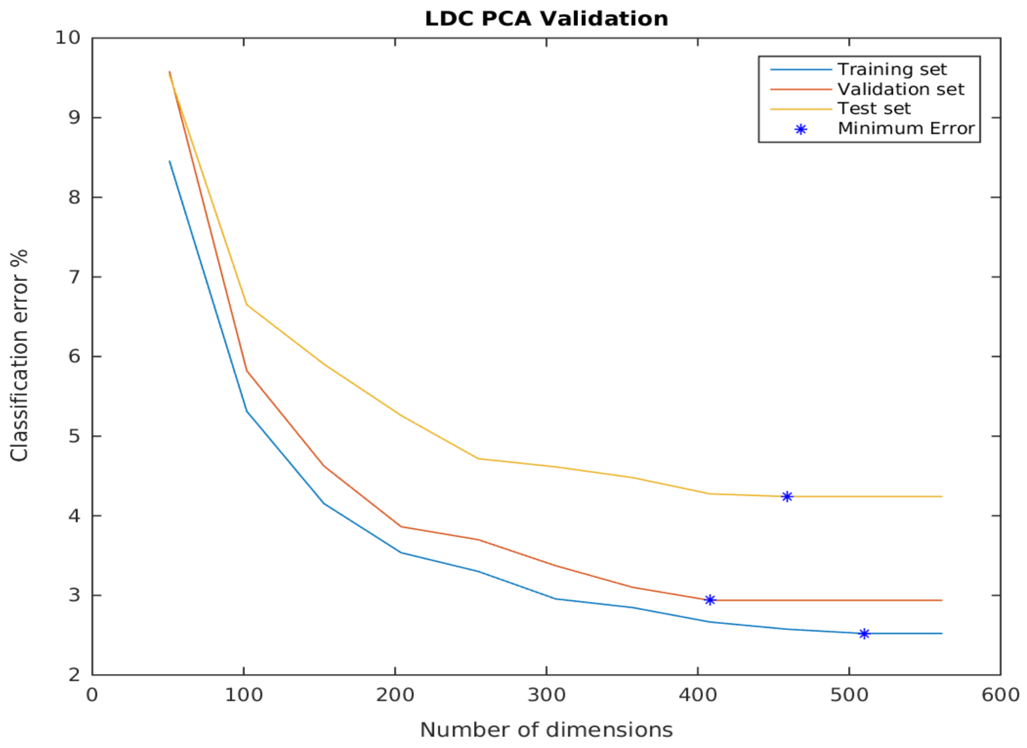


LDC R and S selection on test

From the plot we can see that minimum test error of **4.10%** marked with **red dot** occurs for **R=0** and **S=0.2** . Hence we will use these values for further classification.

Dimensionality Reduction

The minimum classification error for **PCA** validation is as below

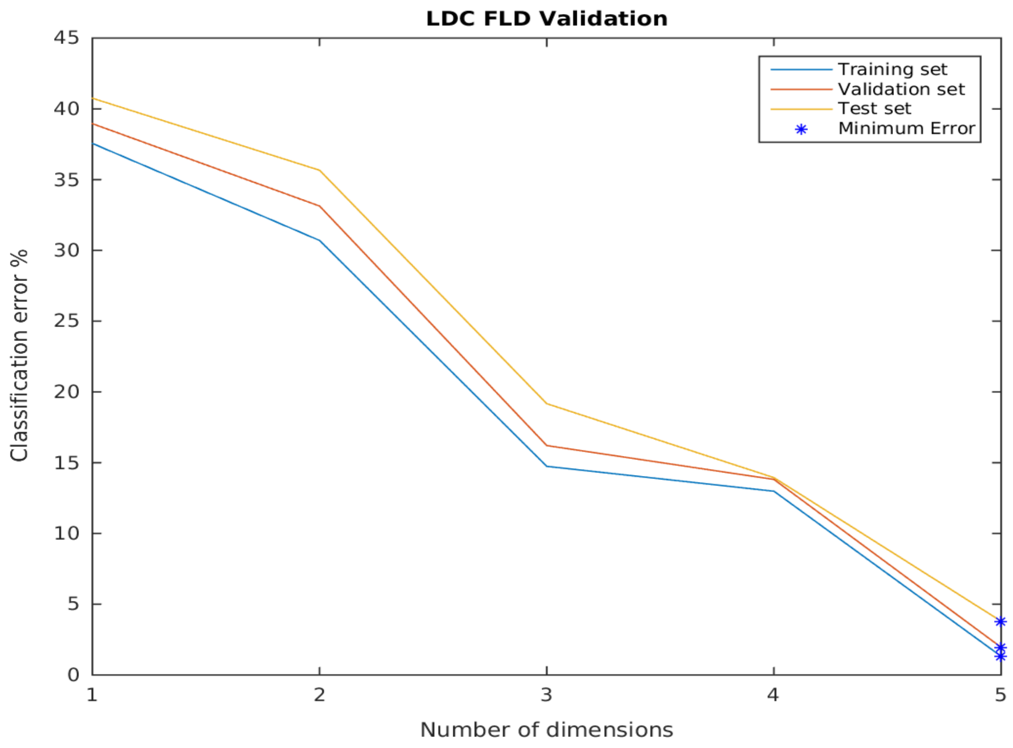


LDC PCA Validation

Training	Validation	Test
2.52%	2.93%	4.24%

The minimum test classification error occurs at **459** dimensions. Also we can see that the test classification error is more than the previous parameter selection value because here we are dividing training data into training and validation and this affects the training.

The minimum classification error for **FLD** validation is as below



LDC FLD Validation

Training	Validation	Test
1.32%	1.95%	3.80%

Conclusion

The best classifier for the dataset is **C-SVC SVM classifier** with RBF kernel and C=32768 and gamma=1.22e-04 with correct classification of **96.68%** . The Ad-hoc method of *Minimum Distance to Class Means classifier* gives *84.52%* correct classification, which means that the data is close to linearly separable but may not be 100% linearly separable. With this hint we can use more sophesticated linear classifiers like **Perceptron** and **LS** to get better results which is proven in this case with **94.68%** and **96.61%** respectively. This also means that we can get still better results using **SVM** , because SVM decision boundary depends only on the support vectors and not on all the training vectors. This is proven by doing rigorous parameter selection using grid search and we get the maximum correct classification of **96.68%** .
Also we can observe that the peroformance of **Statistical classifiers** is highly dependent on their parameters. The maximum difference in baseline performance and best performance is in **LDC** , which is from **83.06%** to **95.90%** . Hence parameter selection is critical, whereas in **Discriminant free classifiers** , parameter selection is useful to get final few percentage of improvement.

References

- http://prtools.org/
- http://www.37steps.com/prtools/

- http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf
- http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- http://www.cad.zju.edu.cn/home/dengcai/Data/code/PCA.m
- http://www.cad.zju.edu.cn/home/dengcai/Data/code/mySVD.m
- http://en.wikipedia.org/wiki/Principal_component_analysis
- http://en.wikipedia.org/wiki/Linear_discriminant_analysis
- http://www.37steps.com/prhtml/prtools.html
- http://en.wikipedia.org/wiki/Naive_Bayes_classifier
- http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Appendix [Code]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% PROJECT.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
addpath('libsvm-3.20/matlab');
addpath('prtools');

f_train = dlmread('../Project/UCI HAR Dataset/train/X_train.txt');
l_train = dlmread('../Project/UCI HAR Dataset/train/y_train.txt');
f_test = dlmread('../Project/UCI HAR Dataset/test/X_test.txt');
l_test = dlmread('../Project/UCI HAR Dataset/test/y_test.txt');

num_features = size(f_train,1);
num_dim = size(f_train,2);
num_classes = size(unique(l_train),1);

% Random class assignment with no priors
l_pred = unidrnd(ones(num_features,1)*num_classes);
100 - 100 * sum(l_pred ~= l_train)/num_features

% Random class assignment with priors
h = histogram(l_train,'Normalization','probability');
xlabel('Class label');
ylabel('PDF');
title('PDF of class labels');
print('prior_pdf.png','-dpng');

pdf = h.Values;
v = unidrnd(ones(num_features*2,1)*num_classes);
u = rand(1,num_features*2);
m = max(pdf);
r = m.*u < pdf(v);
l_pred = v(r==1);
l_pred = l_pred(1:num_features);
```

```
histogram(l_pred,'Normalization','probability');
100 - 100 * sum(l_pred ~= l_train)/num_features

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% MIN DIST TO MEANS.M %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function min_dist_means(f_train,l_train,f_test,l_test)

mean = min_dist_means_train(l_train,f_train);

l_pred = min_dist_means_pred(l_train,f_train,mean);

fprintf('\n Percentage Correct classification Training = %f\n',100*sum(l
_pred == l_train)/length(l_train));

l_pred = min_dist_means_pred(l_test,f_test,mean);

fprintf('\n Percentage Correct classification Test = %f\n',100*sum(l_pre
d == l_test)/length(l_test));

[train_error, validation_error, test_error] = validation('Min Dist Means
 PCA', f_train, l_train, f_test, l_test, @min_dist_means_train, @min_dis
t_means_pred, 1);

[train_error, validation_error, test_error] = validation('Min Dist Means
 FLD', f_train, l_train, f_test, l_test, @min_dist_means_train, @min_dis
t_means_pred, 0);

function mean = min_dist_means_train(l_train,f_train,varargin)
num_classes = length(unique(l_train));
num_dim = size(f_train,2);
mean = zeros(num_classes,num_dim);

for i=1:num_classes
    train_cur = l_train ==  i;
    mean(i,:) = sum(f_train(train_cur,:),1)/sum(train_cur);
end

function l_pred = min_dist_means_pred(l_test,f_test,mean)

num_features = size(f_test,1);
num_dim = size(f_test,2);
num_classes = length(unique(l_test));

d = pdist2(f_test,mean);
```

```
[M,l_pred] = min(d,[],2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% PERCEPTRON.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function perceptron(f_train,l_train,f_test,l_test)

perceptron_train_1 = @(l_train, f_train, varargin) perlc(prdataset(f_tra
in,l_train),100);
perceptron_pred_1 = @(l_test, f_test, w) labeld(prdataset(f_test,l_test)
,w);

w = perceptron_train_1(l_train,f_train);

l_pred = perceptron_pred_1(l_train,f_train,w);

fprintf('\n Percentage Correct classification Training = %f\n',100*sum(l
_pred == l_train)/length(l_train));

l_pred = perceptron_pred_1(l_test,f_test,w);

fprintf('\n Percentage Correct classification Test = %f\n',100*sum(l_pre
d == l_test)/length(l_test));

[train_error, validation_error, test_error] = validation('Perceptron PCA
', f_train, l_train, f_test, l_test, perceptron_train_1, perceptron_pred
_1, 1 );

[train_error, validation_error, test_error] = validation('Perceptron FLD
', f_train, l_train, f_test, l_test, perceptron_train_1, perceptron_pred
_1, 0 );


% Parameter estimation
P = cvpartition(size(f_train,1),'Holdout',0.20);
f_tr_pr = prdataset(f_train(P.training,:),l_train(P.training));
f_va_pr = prdataset(f_train(P.test,:),l_train(P.test));
f_te_pr = prdataset(f_test,l_test);
f = {f_tr_pr,f_va_pr,f_te_pr};

eta = 0.1:0.18:1;
e = zeros(size(eta,2),3);
```

```
for k = 1:size(eta,2)
    w = perlc(f_tr_pr,100,eta(k));
    e(k,:) = f*w*testc*100;
end

plot(eta,e);hold on
[M idx] = min(e);
plot(eta(idx),M,'b*');
legend('Training','Validation','Test','Minimum');
hold off;
xlabel('eta'); ylabel('Classification error %');
title('Perceptron Parameter selection');
print('Perceptron Parameter Selection.png','-dpng');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% LS.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function LS(f_train,l_train,f_test,l_test)

%LS_train = @(l, f, varargin) pinv(f'*f)*f'*l;
%LS_pred = @(l, f, a) round(f*a);

LS_train = @(l_train, f_train, varargin) fisherc(prdataset(f_train,l_tra
in));
LS_pred = @(l_test, f_test, w) labeld(prdataset(f_test,l_test),w);

f_tr_pr = prdataset(f_train,l_train);
f_te_pr = prdataset(f_test,l_test);
f = {f_tr_pr, f_te_pr};

a = LS_train(l_train,f_train,'-q');
c = 100 - f*a*testc*100

[train_error, validation_error, test_error] = validation('LS PCA', f_tra
in, l_train, f_test, l_test, LS_train, LS_pred, 1 );
[train_error, validation_error, test_error] = validation('LS FLD', f_tra
in, l_train, f_test, l_test, LS_train, LS_pred, 0 );

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% SVM.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function svm(f_train,l_train,f_test,l_test)
```

```
model = svmtrain(l_train,f_train,'-q');
[l_pred, accuracy, decision_values] = svmpredict(l_train,f_train,model);

[l_pred, accuracy, decision_values] = svmpredict(l_test,f_test,model);

[train_error, validation_error, test_error] = validation('SVM PCA', f_tr
ain, l_train, f_test, l_test, @svmtrain, @svmpredict, 1 );
[train_error, validation_error, test_error] = validation('SVM FLD', f_tr
ain, l_train, f_test, l_test, @svmtrain, @svmpredict, 0 );



kernel_t = [0 1 2 3];
kernel_t_name = {'Linear','polynomial', 'rbf', 'sigmoid'};
validation_error_kernel = zeros(size(kernel_t,2),1);
test_error_kernel = validation_error_kernel;

P = cvpartition(size(f_train,1),'Holdout',0.20);
% For kernel testing
for k = 1:size(kernel_t,2)
    params = ['-q -c 1000 -g 0.01 -t ',num2str(kernel_t(k))];
    model = svmtrain(l_train(P.training),f_train(P.training,:),params);
    [l_pred, accuracy, decision_values] = svmpredict(l_train(P.test),f_t
rain(P.test,:),model);
    validation_error_kernel(k) = 100*sum((l_pred ~= l_train(P.test)))/si
ze(P.test,1);
    [l_pred, accuracy, decision_values] = svmpredict(l_test,f_test,model
);
    test_error_kernel(k) = 100*sum((l_pred ~= l_test))/size(l_test,1);
end

figure;
plot(kernel_t, validation_error_kernel);
hold on
plot(kernel_t, test_error_kernel);
hold on;
[M1, idx1] = min(validation_error_kernel);
[M2, idx2] = min(test_error_kernel);
plot(kernel_t([idx1 idx2]),[M1 M2],'b*');

set(gca, 'XTick',kernel_t, 'XTickLabel',kernel_t_name);
xlabel('Kernel Type');
ylabel('Classification error %');
title('SVM Kernel selection');
legend('Validation','Test','Minimum');
print('SVM Kernel Selection.png','-dpng');
```

```
c_power = -5:2:15;
C = 2.^c_power;
gamma_power = -15: 2: 3;
gamma = 2.^gamma_power;
test_error_kernel = zeros(size(C,2),size(gamma,2));
validation_error_kernel = test_error_kernel;

P = cvpartition(size(f_train,1),'Holdout',0.20);

% For C ang gamma testing
for i = 1:size(C,2)
    for j = 1:size(gamma,2)
        params = ['-q -t 2  -c ',num2str(C(i)),' -g ',num2str(gamma(j))]
;
        model = svmtrain(l_train(P.training,:),f_train(P.training,:),par
ams);
        [l_pred, accuracy, decision_values] = svmpredict(l_train(P.test,
:),f_train(P.test,:),model);
        validation_error_kernel(i,j) = 100*sum((l_pred ~= l_train(P.test
,:)))/size(P.test,1);
        [l_pred, accuracy, decision_values] = svmpredict(l_test,f_test,m
odel);
        test_error_kernel(i,j) = 100*sum((l_pred ~= l_test))/size(l_test
,1);
    end
end

[Y,X] = meshgrid(gamma,C);

[M idx1] = min(validation_error_kernel);
[M idx2] = min(M);
min_c = X(idx1(idx2),idx2)
min_g = Y(idx1(idx2),idx2)
M

surf(X,Y,validation_error_kernel, gradient(validation_error_kernel)); ho
ld on;
scatter3(min_c,min_g,M,500,'r.');
view(65, 45);
ax = gca;
ax.XScale = 'log';
ax.YScale = 'log';
xlabel('C');
ylabel('gamma');
```

```
zlabel('Error Classification %');
title('SVM C and gamma selection on validation');
print('SVM C and gamma selection on validation.png','-dpng');

figure;
[M idx1] = min(test_error_kernel);
[M idx2] = min(M);
min_c = X(idx1(idx2),idx2)
min_g = Y(idx1(idx2),idx2)
M

surf(X,Y,test_error_kernel,gradient(test_error_kernel)); hold on;
scatter3(min_c,min_g,M,500,'r.');
view(65, 45);
ax = gca;
ax.XScale = 'log';
ax.YScale = 'log';
xlabel('C');
ylabel('gamma');
zlabel('Error Classification %');
title('SVM C and gamma selection on test');
print('SVM C and gamma selection on test.png','-dpng');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% KNN.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

knnc_train = @(l_train, f_train, varargin) knnc(prdataset(f_train,l_trai
n),8);
knnc_pred = @(l_test, f_test, w) labeld(prdataset(f_test,l_test),w);

W = knnc(f_tr_pr);
100 - f_te_pr*W*testc*100

[train_error, validation_error, test_error] = validation('Knnc PCA', f_t
rain, l_train, f_test, l_test, knnc_train, knnc_pred, 1 );
[train_error, validation_error, test_error] = validation('Knnc FLD', f_t
rain, l_train, f_test, l_test, knnc_train, knnc_pred, 0 );


% KNN parameter selection
K = 2:2:12;
n = size(K,2);
f_tr_pr = prdataset(f_train,l_train);
f_te_pr = prdataset(f_test,l_test);
```

```
f = {f_tr_pr(1:6500,:),f_te_pr};

e = zeros(n,2);
for i=1:n
    w = knnc(f_tr_pr,K(i));
    e(i,:) = f*w*testc*100;
end

plot(K,e);hold on
[M idx] = min(e);
plot(K(idx),M,'b*');
legend('Training','Test','Minimum');
hold off;
xlabel('K'); ylabel('Classification error %');
title('KNN Parameter selection');
print('KNN Parameter Selection.png','-dpng');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% NAIVE BAYES.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

naivebc_train = @(l_train, f_train, varargin) naivebc(prdataset(f_train,
l_train),4);
naivebc_pred = @(l_test, f_test, w) labeld(prdataset(f_test,l_test),w);

W = naivebc(f_tr_pr);
100 - f_tr_pr*W*testc*100
100 - f_te_pr*W*testc*100

[train_error, validation_error, test_error] = validation('Naive Bayes PC
A', f_train, l_train, f_test, l_test, naivebc_train, naivebc_pred, 1 );
[train_error, validation_error, test_error] = validation('Naive Bayes FL
D', f_train, l_train, f_test, l_test, naivebc_train, naivebc_pred, 0 );


% Naive bayes parameter selection
K = 2:1:12;
n = size(K,2);
f_tr_pr = prdataset(f_train,l_train);
f_te_pr = prdataset(f_test,l_test);
f = {f_tr_pr,f_te_pr};

e = zeros(n,2);
for i=1:n
    w = naivebc(f_tr_pr,K(i));
    e(i,:) = f*w*testc*100;
end

plot(K,e);hold on
[M idx] = min(e)
plot(K(idx),M,'b*');
legend('Training','Test','Minimum');
hold off;
xlabel('N bins'); ylabel('Classification error %');
title('Naive Bayes Parameter selection');
print('Naive Bayes Parameter Selection.png','-dpng');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% LDC STATISTICAL.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ldc_train = @(l_train, f_train, varargin) ldc(prdataset(f_train,l_train)
,0,0.2);
ldc_pred = @(l_test, f_test, w) labeld(prdataset(f_test,l_test),w);

W = ldc(f_tr_pr);
f_tr_pr*W*testc*100
f_te_pr*W*testc*100

[train_error, validation_error, test_error] = validation('LDC PCA', f_tr
ain, l_train, f_test, l_test, ldc_train, ldc_pred, 1 );
[train_error, validation_error, test_error] = validation('LDC FLD', f_tr
ain, l_train, f_test, l_test, ldc_train, ldc_pred, 0 );


% Naive bayes parameter selection
R = 0:.2:1;
S = 0:.2:1;
n = size(R,2);
f_tr_pr = prdataset(f_train,l_train);
f_te_pr = prdataset(f_test,l_test);

e_test = zeros(n,n);
for i=1:n
    for j=1:n
        w = ldc(f_tr_pr,R(i),S(j));
         e_test(i,j) = f_te_pr*w*testc*100;
    end
end
[Y,X] = meshgrid(S,R);
```

```
[M idx1] = min(e_test);
[M idx2] = min(M);
min_R = X(idx1(idx2),idx2)
min_S = Y(idx1(idx2),idx2)
M

surf(X,Y,e_test); hold on;
scatter3(min_R,min_S,M,800,'r.');
view(230, 45);
xlabel('R');
ylabel('S');
zlabel('Error Classification %');
title('LDC R and S selection on test');
print('LDC R and S selection on test.png','-dpng');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% VALIDATION.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [train_error, validation_error, test_error] = validation(classi
fier,
 f_train, l_train, f_test, l_test, func_train, func_pred, pca_or_fld)
  t_start = tic;
  addpath('libsvm-3.20/matlab');
  num_features = size(f_train,1);
  num_classes = size(unique(l_train),1);
  num_dim = size(f_train,2);

  if pca_or_fld ==1
      num_steps = 11;
  else
      num_steps = num_classes - 1;
  end

  step = floor(num_dim/num_steps);
  train_error = zeros(num_steps,1);
  validation_error = zeros(num_steps,1);
  test_error = zeros(num_steps,1);

  idx = 1:num_features;

  f_validation = f_train(mod(idx,4)==0,:);
  l_validation = l_train(mod(idx,4)==0);
  f_train_s = f_train(mod(idx,4)~=0,:);
  l_train_s = l_train(mod(idx,4)~=0);

  for i = 1:num_steps
```

```
    n_f =  step * i;
    if pca_or_fld ==1
        options.ReducedDim = n_f;
        [e,~] = PCA(f_train,options);
        f_train_new = f_train_s * e;
        f_validation_new = f_validation * e;
        f_test_new = f_test * e;
    else
        f_tr_pr = prdataset(f_train_s,l_train_s);
        f_va_pr = prdataset(f_validation,l_validation);
        f_te_pr = prdataset(f_test,l_test);
        W = fisherm(f_tr_pr,i);
        f_train_new = f_tr_pr*W;
        f_validation_new = f_va_pr*W;
        f_test_new = f_te_pr*W;
        f_train_new = f_train_new.data;
        f_validation_new = f_validation_new.data;
        f_test_new = f_test_new.data;
    end
%{
    f_train_new = f_train_new(:,1:n_f);
    f_validation_new = f_validation_new(:,1:n_f);
    f_test_new = f_test_new(:,1:n_f);
%}
    model = func_train(l_train_s, f_train_new,'-q -c 32768 -g 1.22e-04')
  ;

    l_pred = func_pred( l_train_s,f_train_new,model);
    train_error(i) = 100*sum((l_pred ~= l_train_s))/size(l_train_s,1);

    l_pred = func_pred(l_validation,f_validation_new,model);
    validation_error(i) = 100*sum((l_pred ~= l_validation))/size(l_valid
ation,1);

    l_pred = func_pred( l_test,f_test_new,model);
    test_error(i) = 100*sum((l_pred ~= l_test))/size(l_test,1);

  end

  if pca_or_fld ==1
      x = step:step:num_dim;
  else
      x = 1:num_steps;
  end
  figure;
  [M, idx] = min(train_error);
  h1 = plot(x,train_error);
  hold on
  plot(x(idx),M,'b*');
  hold on
  M
  [M, idx] = min(validation_error);
  h2 = plot(x,validation_error);
  hold on
  plot(x(idx),M,'b*');
  hold on
  M
  [M, idx] = min(test_error);
  h3 = plot(x,test_error);
  hold on
  h4 = plot(x(idx),M,'b*');
  hold off
  legend([h1,h2,h3,h4],'Training set','Validation set','Test set','Minim
um Error');
  M
  title(strcat(classifier,' Validation'));
  xlabel('Number of dimensions');
  ylabel('Classification error %');
  print(strcat(classifier,'_validation.png'),'-dpng');

  toc(t_start)
```