EE559 Project Report

Ahmet Can Ozbek
USC ID:8401860152

Individual Project
Prior or Parallel Work: None

Abstract

In this project, my aim is to correctly classify Human Activity Recognitions. (The dataset I used is the fifth suggested dataset.). For the default system I used random assignment to classes and minimum distance to means classifier. Then I used three different categories of classifiers. These categories are: distribution-free classification, statistical classification, and support vector machine or neural networks. For distribution-free classification I used Perceptron and MSE, for statistical classification I used Linear Bayes Normal Classifier, Quadratic Bayes Normal Classifier, Parzen widow method and kNN, and for the last part I used SVM. Training, validation and test sets are randomly chosen from the whole dataset with keeping the same ratios of classes as it is in the whole dataset. After comparing many different classifiers, I got the best results from Parzen Window method, kNN and SVM classifiers.

Dataset:

The name of the dataset I used is: Wearable Computing: Classification of Body Postures and Movements (PUC-Rio) Data Set.
There are 165,633 samples and 5 classes which are 'sitting', 'standing', 'walking', 'standingup', 'sittingdown'.

The number of features are 18, however one feature is the name of the test subjects, therefore this feature is cancelled out. After that, one of the samples have a missing feature therefore this sample is also discarded, so we are left with 165,632 samples and 17 features. One of the features is a categorical feature which is the gender of the test subject. So this categorical feature is replaced with a binary vector which has length 2. For 'male', it is replaced with [0 1] and female is replaced with [1 0]. As a result, we have 165,632 samples, 5 classes and 18 features.

For normalization, MATLAB's zscore() function is used. This function makes the samples have zero mean and standard deviation one. For SVM classification, features are scaled between 0 and 1.

Methods:

For this project I used Matlab and its toolboxes. I used classification toolbox, PRTools, and libSVM.

I subsampled three different sets which are training set, validation set and test set. Training set has 6000 samples, validation and test sets have 2000 samples. These samples are taken from the whole dataset randomly but in each set the ratio of classes is preserved to the ratio of the classes in the larger dataset.
A function that takes training, validation ,and test sets randomly but keeping the class ratios is coded up by me.
For default systems, I performed classifications with Random Assignments to Classes with priors obtained by class frequencies and minimum distance to means classifier. (Codes for these systems are written by me).
For distribution-free classification, Perceptron and MSE is used for these multiclass() function is used.
For statistical classification, Linear Bayes Normal Classifier, Quadratic Bayes Normal Classifier used. (functions used: ldc(),qdc(), parzenc(), knnc())
In the last part SVM is used. (svmtrain(), svmpredict() functions are used).

clevalf() function is used to analyze the error rate of the classifier depending on the number of dimensions.

For feature expansion quadratic mapping(Phi Machine) is used. The code of this mapping is impelemented by me and can me found in the matlab code section of this report. The name of the function is myPhiMachine()

1)Default Systems

1.1)Random Assignment to Classes

For default systems, I assigned random labels to classes using priors. Priors are obtained using the frequency of the classes over the whole number of samples in the dataset.

The correct rate for Random Assignment with priors is: **25.492 %**

1.2) Minimum Distance to Class Means Classifier

I chose minimum distance to class means classifier as my baseline system: I performed this classifier with and without normalizing the data. Normalizing is done with MATLAB's zscore() function which makes the samples have zero mean and standard deviation one.
The results for this system is:

*Without Normalizing
Correct Rate in Training Set: 63.1667%
Correct Rate in Validation Set: 61.75%
Correct Rate in Test Set: 63.2%
*With Normalizing
Correct Rate in Training Set: 69.6%
Correct Rate in Validation Set: 69.2%
Correct Rate in Test Set: 70.2%

So here we observe that normalizing the data gives us better results. Therefore throughout the project all datasets used are normalized with zscore() function.

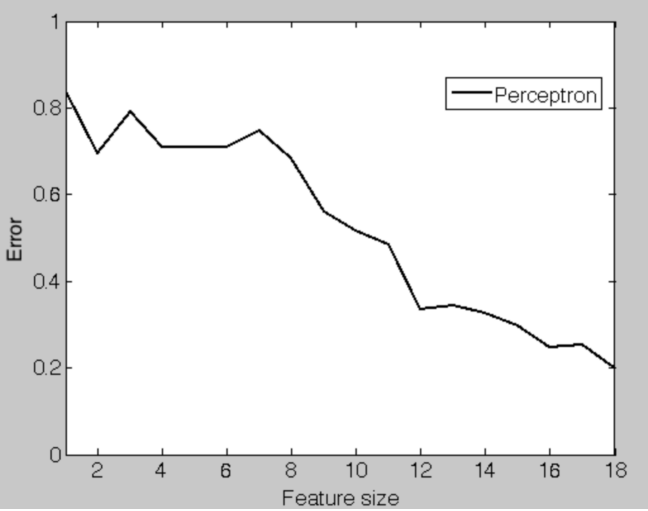
2)Distribution-free Classifiers

2.1)Perceptron

For perceptron algorithm, I used 10000 maximum iterations and one vs. all method

As my dataset is real data sensors, it is a moderately complicated dataset, so it is not likely to be linearly separable. We can also deduct this assumption from the fact that perceptron algorithm does not converge after 10000 iterations. As the dataset is not linearly separable, the perceptron algorithm does not converge after 10000 iterations. As a result, perceptron does not give us good results. The results for perceptron algorithm is:

Perceptron Correct Rate on Training Set:0.5925
Perceptron Correct Rate on Validation Set:0.57
Perceptron Correct Rate on Test Set:0.582

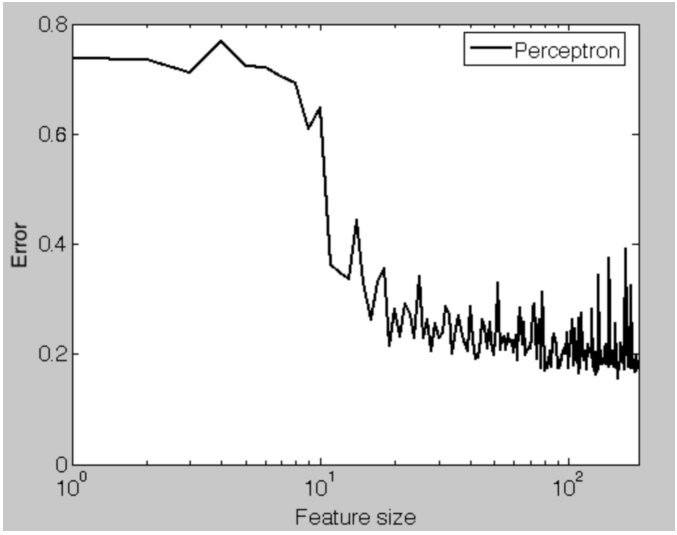


Here, we are also assessing the effect of number of dimensions for Perceptron algorithm. The plot above shows us the error rate versus the number of dimensions. From the plot, we can see that as we use more features for Perceptron, we get a better result and performance. This makes sense because, as there are more features, we have more information to use to classify the samples

To do more evaluation on the effect of feature dimensionality for the performance, I used Phi Machine(quadratic mapping) on my dataset for feature expansion. As I have 18 features originally, my new mapped dataset has 18x21/2 + 1 = 190 features. If we apply perceptron with the same parameters, we get a better result. The results for datasets are:

Perceptron Correct Rate on Training Set:0.70217
Perceptron Correct Rate on Validation Set:0.572
Perceptron Correct Rate on Test Set:0.576

The fact that we got better results after applying feature expansion is an expected result, because our original dataset is not linearly separable however by feature expansion, we are making our dataset tend toward more linearly separable in higher dimensions. As Perceptron algorithm is expected to perform better for more linearly separable datasets, it makes sense that we are getting better performance with expanded feature space.



This plot, again shows us the error rate versus the number of dimensions. This time we expanded our feature space and we have 190 dimensions. Similarly for our original feature space (unexpanded), we get the same trend which means that using more dimensions give better results, as we are using more information.

2.2)MSE

The results for MSE classifier:

MSE Correct Rate on Training Set:0.6225
MSE Correct Rate on Validation Set:0.6275
MSE Correct Rate on Test Set:0.709

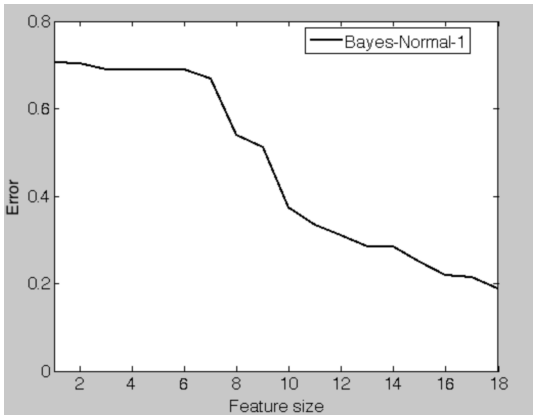
3)Statistical Classification

Here, we have the confusion matrices and plot of error rate depending on the dimensionality for the following classifiers obtained by PRTools.

3.1)LDC(Linear Bayes Normal Classifier)

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	611	0	0	0	0	611
sittingdown	7	81	29	21	5	143
standing	0	5	517	1	49	572
standingup	15	23	34	71	7	150
walking	1	2	153	10	358	524
Totals	634	111	733	103	419	2000

LDC validationSet correct rate: 0.819



True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	610	1	0	1	0	612
sittingdown	15	80	29	15	4	143
standing	0	5	521	1	45	572
standingup	22	18	34	70	5	149
walking	0	5	137	16	366	524
Totals	647	109	721	103	420	2000

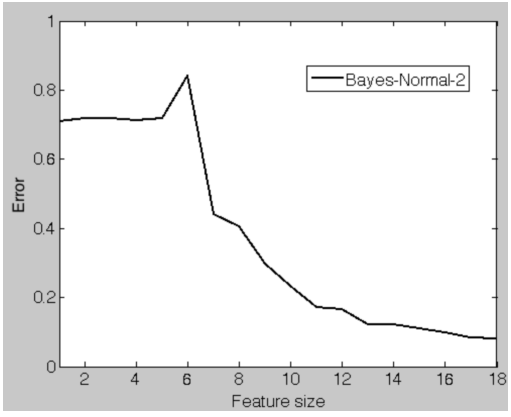
LDC testSet correct rate: 0.8235

By comparing the results of linear bayes and quadratic bayes, we see that quadratic bayes performs better. This stems from the fact that the data is not linearly separable and classifiers having linear properties perform worse that non-linear ones.

3.2)QDC(Quadratic Bayes Normal Classifier)

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	584	16	0	11	0	611
sittingdown	2	127	2	11	1	143
standing	0	7	548	8	9	572
standingup	0	14	15	116	5	150
walking	0	9	40	6	469	524
Totals	586	173	605	152	484	2000

QDC validationSet correct rate: 0.922



True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	599	5	0	8	0	612
sittingdown	3	125	2	12	1	143
standing	0	6	545	10	11	572
standingup	1	20	7	109	12	149
walking	0	7	31	7	479	524
Totals	603	163	585	146	503	2000

QDC testSet correct rate: 0.9285

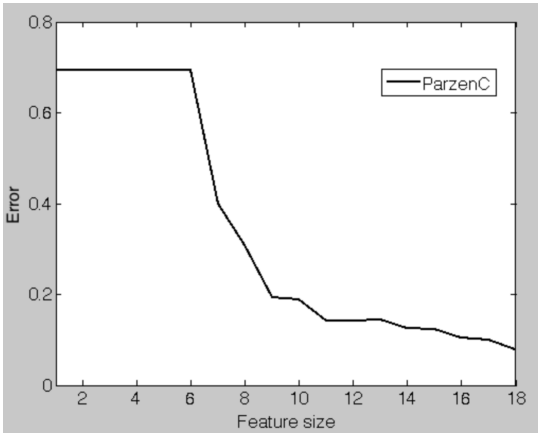
3.3)Parzen Window Method

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	611	0	0	0	0	611
sittingdown	1	128	9	3	2	143
standing	0	1	570	1	0	572
standingup	1	4	26	119	0	150
walking	0	6	64	1	453	524
Totals	613	139	669	124	455	2000

Parzen validationSet correct rate: 0.9405

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	611	0	0	1	0	612
sittingdown	5	121	12	3	2	143
standing	1	1	568	1	1	572
standingup	1	8	26	113	1	149
walking	0	3	44	0	477	524
Totals	618	133	650	118	481	2000

Parzen testSet correct rate: 0.945



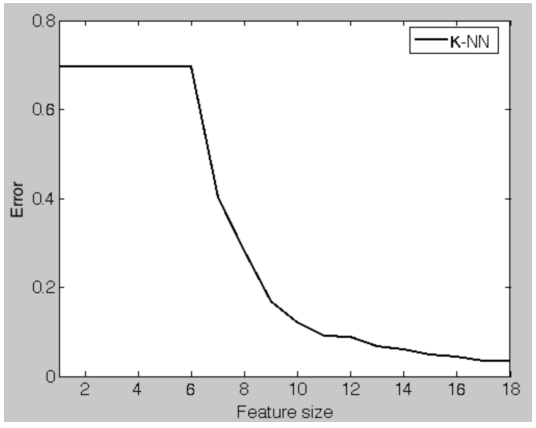
3.4)k-Nearest Neighbor method

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	611	0	0	0	0	611
sittingdown	0	135	0	6	2	143
standing	0	1	569	2	0	572
standingup	0	3	7	139	1	150
walking	0	5	25	2	492	524
Totals	611	144	601	149	495	2000

kNN validationSet correct rate: 0.973

True Labels	Estimated Labels: sittin	sittin	standi	standi	walkin	Totals
sitting	611	0	0	1	0	612
sittingdown	1	134	2	3	3	143
standing	0	0	569	2	1	572
standingup	0	9	6	133	1	149
walking	0	3	19	0	502	524
Totals	612	146	596	139	507	2000

kNN testSet correct rate: 0.9745



4)Support Vector Machine(SVM)

Before performing SVM all features are scaled into [0 1] range.

For SVM, I performed classifications with different Kernel functions and compered the results of them. In addition, I changed the parameters of these different Kernel functions to see how the performance results change according to the parameters of Kernels. SVM is trained on the training set and results are obtained on the test set.

The Kernel functions used and the parameters changed are:

- Linear (Parameters: C)
- Polynomial (Parameters: gamma, degree)
- Radial Basis Function (Parameters: C, gamma)

-Linear Kernel Function Results:

C:	50	500	5000	10000
Correct Rate:	83.1%	89.55%	92.7%	93.35%

Here, we observe that as we increase the parameter C, we get better results. The C parameter prevents misclassifying each training example with the trade-off of margin-length. Therefore we can say that, we get less misclassified samples if C is greater.

Polynomial Kernel Function Results:

degree\gamma	10	50	100	500	1000	2500	5000
degree=2	0.963	0.958	0.956	0.956	0.9595	0.954	0.9455
degree=3	0.9655	0.967	0.9665	0.9515	0.9575	0.961	0.927
degree=4	0.965	0.9625	0.9625	0.953	0.953	0.9505	0.9505

Here, we are changing the parameters degree and gamma of polynomial Kernel function. As we can observe, changing these parameters don't have substantial effect on the performance.

Radial Basis Kernel Function Results:

C\gamma	10	50	100	500	1000	2500	5000
C=50	0.973	0.974	0.967	0.9115	0.874	0.81	0.757
C=500	0.973	0.972	0.967	0.9115	0.874	0.81	0.757
C=5000	0.9755	0.972	0.967	0.9115	0.874	0.81	0.757

Here we observe that changing the parameter C does not have much effect on the performance rate, however changing the gamma function has effect. If we increase the gamma parameter, we get worse results in terms of correctly classifying the test set. This is due to over-fitting. If the gamma parameter is large, it fits over the training set almost perfectly and this causes problems on test data which is unknown.

Final Result,Final System and Conclusion:

The best results are obtained from kNN and SVM classifiers. The best classifier is SVM with Radial Basis Kernel Function which have parameters: C=5000 and gamma = 10 and the result it gave is 97.55% which is my best correct rate.

For perceptron, Linear Bayes Normal Classifier, and SVM with linear Kernel Function, we got worse results compared to other classifiers. This indicates that, classifiers that have linear property performs worse because our dataset does not have a distribution similar to a dataset that is linearly separable.

Project Codes

The function to get random training,validation, test from the original data sets keeping the class ratios (my implementation):

```
function[trainingSet, validationSet, testSet, classTraining,
classValidation, classTest] =
getSets(howManyTraining,howManyValid,howManyTest,
featureMatrix,classLabels)

total = howManyTraining + howManyValid + howManyTest;
howManyClass = size(unique(classLabels),1);
howManyFeatures = size(featureMatrix,1);
classes = unique(classLabels);
%getting ratios
r = zeros(howManyClass,1);
trainIndices = [];
validIndices = [];
testIndices = [];
for i=1:howManyClass
    r(i) = sum(strcmp(classLabels,classes{i})) / howManyFeatures;
    temp =
randsample(find(strcmp(classLabels,classes{i})==1),round(total *
r(i)));
    [y] = mySplit(temp,[round(howManyTraining*r(i))
round(howManyValid*r(i)) round(total * r(i))-
(round(howManyTraining*r(i))+round(howManyValid*r(i)))]);
    newTrainIndices = y{1};
    newValidIndices = y{2};
    newTestIndices = y{3};
    trainIndices = [trainIndices newTrainIndices'];
    validIndices = [validIndices newValidIndices'];
    testIndices = [testIndices newTestIndices'];

end
%training
trainingSet = featureMatrix(trainIndices,:); classTraining =
classLabels(trainIndices,:);
validationSet = featureMatrix(validIndices,:); classValidation =
classLabels(validIndices,:);
testSet = featureMatrix(testIndices,:); classTest =
classLabels(testIndices,:);

end

function[y] = mySplit(input,lengths)
start=0; endp=0;
howManyPieces = max(size(lengths));
y = [];

for i=1:howManyPieces
    start = endp + 1;
    endp = start + lengths(i) -1;
    y{i} = input(start:endp);
end

end
```

The function to turn categorical featues into binary vectors(my implementation)::

```
function [binaryFeatureMatrix] =
turnCategoricalFeatureToBinary(categoricalFeature)

numberOfFeatures = size(categoricalFeature,1);
howManyCategory = size(unique(categoricalFeature),1);
categories = unique(categoricalFeature);

binaryFeatureMatrix = zeros(numberOfFeatures,howManyCategory);

    for i=1:howManyCategory
         binaryFeatureMatrix(:,i) =
strcmp(categoricalFeature,categories(i));
    end
end
```

The function to expand the feature space according to quadratic mapping(Phi Machine) (my implementation)::

```
function[newMatrix] = myPhiMachine(featureMatrix)

[howManySamples, howManyFeatures] = size(featureMatrix);
counter = 0;
for i=1:howManyFeatures
    for j=i:howManyFeatures
        counter = counter + 1;
        newMatrix(:,counter) = featureMatrix(:,i) .*
featureMatrix(:,j);
    end
end

newMatrix = [ones(howManySamples,1) featureMatrix(:,1:howManyFeatures)
newMatrix];
end
```

Default Systems(Random assignment with class priors and Minimum distance to class means classifier):

```
%-----Default Systems-----
%[trainingSet, validationSet, testSet, classTraining, classValidation,
classTest] = getSets(howManyTraining,howManyValid,howManyTest,
featureMatrix,class1);
clear; load('sets.mat');
disp('***Default Systems***');
%*****Random assignment to classes*****
load('dataSet5.mat');
classes = unique(class1);
howManyClass = max(size(unique(class1)));
howManySamples = max(size(class1,1));
for i=1:howManyClass
    r(i) = sum(strcmp(class1,classes(i)));
end
%randomly assign class labels with priors
randomAssignedLabels = randsample(howManyClass,howManySamples,true,r);
correctRate = mean(convert2NumLabels(class1) == randomAssignedLabels);
```

```
disp(['***Random Assignment To Classes Correct Rate: '
num2str(correctRate)]);

%*****Baseline System: Minimum distance to means classifier*****
% trainingSet = myPhiMachine(trainingSet);
% validationSet = myPhiMachine(validationSet);
% testSet = myPhiMachine(testSet);
disp('*****Baseline System: Minimum distance to means
classifier*****');
%Without normalizing
disp('*Without Normalizing');
disp(['Correct Rate in Training Set: ' num2str(100-
minimimDistanceToMeansClassifier(trainingSet,classTraining,trainingSet,
classTraining)) '%']);
disp(['Correct Rate in Validation Set: ' num2str(100-
minimimDistanceToMeansClassifier(trainingSet,classTraining,validationSe
t,classValidation)) '%']);
disp(['Correct Rate in Test Set: ' num2str(100-
minimimDistanceToMeansClassifier(trainingSet,classTraining,testSet,clas
sTest)) '%']);
%With normalizing
disp('*With Normalizing');
disp(['Correct Rate in Training Set: ' num2str(100-
minimimDistanceToMeansClassifier(zscore(trainingSet),classTraining,zsco
re(trainingSet),classTraining)) '%']);
disp(['Correxct Rate in Validation Set: ' num2str(100-
minimimDistanceToMeansClassifier(zscore(trainingSet),classTraining,zsco
re(validationSet),classValidation)) '%']);
disp(['Correct Rate in Test Set: ' num2str(100-
minimimDistanceToMeansClassifier(zscore(trainingSet),classTraining,zsco
re(testSet),classTest)) '%']);
%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('');
%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Distribution-Free Classifiers(Perceptron and MSE):

```
%-----(i)Distribution Free Classification-----
%[trainingSet, validationSet, testSet, classTraining, classValidation,
classTest] = getSets(howManyTraining,howManyValid,howManyTest,
featureMatrix,class1);
 clear; load('sets.mat');
trainingSet = myPhiMachine(trainingSet);
validationSet = myPhiMachine(validationSet);
testSet = myPhiMachine(testSet);
 %*****Perceptron*****
 disp('*****Perceptron*****');
 [train_targetsP, ~] = multiclass(zscore(trainingSet)',
double(convert2NumLabels(classTraining))', zscore(trainingSet)',
'[''OAA'', 0, ''Perceptron'', [10000]]');
 validation_error_Percept_ova =
mean(convert2NumLabels(classTraining)'==train_targetsP);
 disp(['Perceptron Correct Rate on Training Set:'
num2str(validation_error_Percept_ova)]);

 [validation_targetsP, ~] = multiclass(zscore(trainingSet'),
double(convert2NumLabels(classTraining))', zscore(validationSet'),
'[''OAA'', 0, ''Perceptron'', [10000]]');
```

```
 validation_error_Percept_ova =
mean(convert2NumLabels(classValidation)'==validation_targetsP);
 disp(['Perceptron Correct Rate on Validation Set:'
num2str(validation_error_Percept_ova)]);

 [test_targets, ~] = multiclass(zscore(trainingSet'),
convert2NumLabels(classTraining)', zscore(testSet'), '[''OAA'', 0,
''Perceptron'', [10000]]');
 test_error_Percept_ova =
mean(convert2NumLabels(classTest)'==test_targets);
 disp(['Perceptron Correct Rate on Test Set:'
num2str(test_error_Percept_ova)]);

 %Normalizing
 trainingSet = zscore(trainingSet);
 validationSet = zscore(validationSet);
 testSet = zscore(testSet);
 %Construct datasets
 trainingData = dataset(trainingSet,classTraining);
 validationData = dataset(validationSet,classValidation);
 testData = dataset(testSet,classTest);
 %Analyze Dimensionality
 ErrorTrainPerceptron = clevalf(trainingData,perlc,[],[],1);
 figure();
 plote(ErrorTrainPerceptron);


 %*****MSE*****
 disp('*****MSE*****');
 [training_targets, ~] = multiclass(zscore(trainingSet)',
double(convert2NumLabels(classTraining))', zscore(trainingSet)',
'[''OAA'', 0, ''LS'', []]');
 validation_error_MSE_ova =
mean(convert2NumLabels(classTraining)'==training_targets);
 disp(['MSE Correct Rate on Training Set:'
num2str(validation_error_MSE_ova)]);

  [validation_targets, ~] = multiclass(zscore(trainingSet)',
double(convert2NumLabels(classTraining))', zscore(validationSet)',
'[''OAA'', 0, ''LS'', []]');
 validation_error_MSE_ova =
mean(convert2NumLabels(classValidation)'==validation_targets);
 disp(['MSE Correct Rate on Validation Set:'
num2str(validation_error_MSE_ova)]);

 [test_targets, ~] = multiclass(zscore(trainingSet)',
double(convert2NumLabels(classTraining))', zscore(testSet)', '[''OAA'',
0, ''LS'', []]');
 test_error_MSE_ova =
mean(convert2NumLabels(classTest)'==test_targets);
 disp(['MSE Correct Rate on Test Set:' num2str(test_error_MSE_ova)]);
```

Statistical Classification(Linear Bayes, Quadratic Bayes, Parzen Window, kNN):

```
%-----(ii)Statistical Classification-----
%[trainingSet, validationSet, testSet, classTraining, classValidation,
classTest] = getSets(howManyTraining,howManyValid,howManyTest,
```

```
featureMatrix,class1);
clear; load('sets.mat');
%*****LDC, QDC*****
disp('*****LDC, QDC*****');
%Normalizing
trainingSet = zscore(trainingSet);
validationSet = zscore(validationSet);
testSet = zscore(testSet);
%Construct datasets
trainingData = dataset(trainingSet,classTraining);
validationData = dataset(validationSet,classValidation);
testData = dataset(testSet,classTest);

%LDC Linear Bayes Normal Classifier
%validation data
trueValidationLabels = getlab(validationData);
w = ldc(trainingData);
predictedValidationLabels = validationData * w * labeld;
confmat(trueValidationLabels,predictedValidationLabels);
disp(['LDC validationSet correct rate: '
num2str(mean(strcmp(cellstr(predictedValidationLabels),cellstr(trueVali
dationLabels))))]);
%test data
trueTestLabels = getlab(testData);
w = ldc(trainingData);
predictedTestLabels = testData * w * labeld;
confmat(trueTestLabels,predictedTestLabels);
disp(['LDC testSet correct rate: '
num2str(mean(strcmp(cellstr(predictedTestLabels),cellstr(trueTestLabels
))))]);

%QDC Quadratic Bayes Normal Classifier
%validation data
trueValidationLabels = getlab(validationData);
w = qdc(trainingData);
predictedValidationLabels = validationData * w * labeld;
confmat(trueValidationLabels,predictedValidationLabels);
disp(['QDC validationSet correct rate: '
num2str(mean(strcmp(cellstr(predictedValidationLabels),cellstr(trueVali
dationLabels))))]);
%test data
trueTestLabels = getlab(testData);
w = qdc(trainingData);
predictedTestLabels = testData * w * labeld;
confmat(trueTestLabels,predictedTestLabels);
disp(['QDC testSet correct rate: '
num2str(mean(strcmp(cellstr(predictedTestLabels),cellstr(trueTestLabels
))))]);

 %ldc and qdc
 trainingErrorPlot_ldc = clevalf(trainingData,ldc,[],[],1);
 figure();
 plote(trainingErrorPlot_ldc);

 trainingErrorPlot_qdc  = clevalf(trainingData,qdc,[],[],1);
 figure();
 plote(trainingErrorPlot_qdc);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('%%%%%%%%%%%%%%%');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*****Parzen*****
%validation data
trueValidationLabels = getlab(validationData);
w = parzenc(trainingData);
predictedValidationLabels = validationData * w * labeld;
confmat(trueValidationLabels,predictedValidationLabels);
disp(['Parzen validationSet correct rate: '
num2str(mean(strcmp(cellstr(predictedValidationLabels),cellstr(trueVali
dationLabels))))]);
%test data
trueTestLabels = getlab(testData);
w = parzenc(trainingData);
predictedTestLabels = testData * w * labeld;
confmat(trueTestLabels,predictedTestLabels);
disp(['Parzen testSet correct rate: '
num2str(mean(strcmp(cellstr(predictedTestLabels),cellstr(trueTestLabels
))))]);
%*****kNN*****
%validation data
trueValidationLabels = getlab(validationData);
w = knnc(trainingData);
predictedValidationLabels = validationData * w * labeld;
confmat(trueValidationLabels,predictedValidationLabels);
disp(['kNN validationSet correct rate: '
num2str(mean(strcmp(cellstr(predictedValidationLabels),cellstr(trueVali
dationLabels))))]);
%test data
trueTestLabels = getlab(testData);
w = knnc(trainingData);
predictedTestLabels = testData * w * labeld;
confmat(trueTestLabels,predictedTestLabels);
disp(['kNN testSet correct rate: '
num2str(mean(strcmp(cellstr(predictedTestLabels),cellstr(trueTestLabels
))))]);


%parzen and knn
trainingErrorPlot_parzen = clevalf(trainingData,parzenc,[],[],1);
figure();
plote(trainingErrorPlot_parzen);

trainingErrorPlot_knn  = clevalf(trainingData,knnc,[],[],1);
figure();
plote(trainingErrorPlot_knn);
```

SVM Classifier:

```
%WITH LOOPS
[scaledTraining, scaledValid, scaledTest] =
myScaleSets(trainingSet,validationSet,testSet);
%for linear(-t 0)
c = [50,500,5000,10000];
resultLinearMatrix = zeros(size(c,2),1);
for i=1:size(c,2)
```

```
        model =
svmtrain(convert2NumLabels(classTraining),scaledTraining,['-t 1' ' -c '
num2str(c(i)) ' -q' ]);
        [predicted_label, accuracy, decision_values] =
svmpredict(convert2NumLabels(classTest), scaledValid, model);
        resultLinearMatrix(i) = mean(predicted_label ==
convert2NumLabels(classTest));
end

%for Polynomial(-t 1)
c = [50,500,5000];
gamma = [10,50,100,500,1000,2500,5000];
degree = [2,3,4];
resultMatrixPoly = zeros(size(c,2),size(gamma,2));
for i=1:size(degree,2)
    for j=1:size(gamma,2)
        model =
svmtrain(convert2NumLabels(classTraining),scaledTraining,['-t 1 ' ' -c
100 ' ' -g ' num2str(gamma(j)) ' -d ' num2str(degree(i)) ' -q' ]);
        [predicted_label, accuracy, decision_values] =
svmpredict(convert2NumLabels(classTest), scaledValid, model);
        resultMatrixPoly(i,j) = mean(predicted_label ==
convert2NumLabels(classTest));
    end
end

%for Gaussian (RBF) (-t 2)
c = [50,500,5000];
gamma = [10,50,100,500,1000,2500,5000];
resultMatrixRBF = zeros(size(c,2),size(gamma,2));
for i=1:size(c,2)
    for j=1:size(gamma,2)
        model =
svmtrain(convert2NumLabels(classTraining),scaledTraining,['-t 2' ' -c '
num2str(c(i)) ' -g ' num2str(gamma(j)) ' -q' ]);
        [predicted_label, accuracy, decision_values] =
svmpredict(convert2NumLabels(classTest), scaledValid, model);
        resultMatrixRBF(i,j) = mean(predicted_label ==
convert2NumLabels(classTest));
    end
end
```

The function that converts string class labels into numerical format(my implementation)

```
function[numClassLabels] = convert2NumLabels(classLabels)
%Converts different class labels in string format to numerical format
howManyClass = max(size(unique(classLabels)));
howManyFeatures = max(size(classLabels));
classes = unique(classLabels);
numClassLabels = zeros(howManyFeatures,1);

for i=1:howManyClass
    numClassLabels(strcmp(classes{i},classLabels)) = i;
end

end
```