In EDA we observed that in the dataset there are 1736 outliers so before applying the algorithm we will remove these outliers from our dataset and we will then divide this data set into training and testing data set which is as shown below.

```
dataset_without_outliers <- dataset[-c(outliers_row),]

set.seed(43)
randomized__withoutOutliers=dataset_without_outliers[sample(1:nrow(dataset_without_outliers),nrow(dataset_without_outliers)),]
tridx_withoutOutliers=sample(1:nrow(dataset_without_outliers),0.7*nrow(dataset_without_outliers),replace=F)
trainingDataset_withoutOutliers = randomized__withoutOutliers[tridx_withoutOutliers,]
testingDataset_withoutOutliers = randomized__withoutOutliers[-tridx_withoutOutliers,]
```

We also observed from the Pearson's Coefficient Graph calculated in the EDA that the Feature variable V2 is highly correlated with V3, V4, V5 and V6. And V14 is highly correlated with V6, V12 is highly correlated with V8.
Theoretically we can remove the feature variables V3, V4, V5, V6 and V12 and we will have V2 and V8 to cover up for them. But instead of blindly removing the feature variables I will run the algorithms in which only the outliers are removed and in the other I will remove the correlated feature variables and see if the effect is substantial. If from the accuracy we observe that removing correlated feature variables improves the accuracy of the algorithm we will remove it or else if the removal of feature algorithms reduces the accuracy of the algorithms, we will keep the variables as is.

The first algorithm that we will run is LDA, LDA works best when the Feature variables are continuous and not categorical. If the feature variables had some categorical Feature variables, we would have chosen Logistic Regression.

LDA assumes that each input variable has the same variance and in QDA each class uses its own estimate of variance (or covariance when there are multiple input variables) hence performance of QDA should be better than LDA. We will run LDA and QDA both to see if QDA accuracy is better than LDA.

Let's start with LDA with the dataset where outliers are removed and another with outliers and correlated coefficient both removed.

LDA Learning Phase

```
> ##############################################################################
> #                                    LDA                                     #
> ##############################################################################
>
> # Fit the model - Learning Phase
> lda.model.withoutOutliers <- lda(Letter~., data = trainingDataset_withoutOutliers)
> # Make predictions for training data set
> predictions.withoutOutliers <- lda.model.withoutOutliers %>% predict(trainingDataset_withoutOutliers)
> # Train rate - Learning
> lda.withoutOutliers.rate<- mean(predictions.withoutOutliers$class == trainingDataset_withoutOutliers$Letter)
>
> print(paste("LDA Train Rate =",round(lda.withoutOutliers.rate*100, 4)))
[1] "LDA Train Rate = 72.1918"
```

Learning Phase Confusion Matrix

```
> # Learnign Phase - Confusion Matrix
> lda_table_training <- table(list(predicted=predictions.withoutOutliers$class, observed=trainingDataset_withoutOutliers$Letter))
> lda_cfm_training <- confusionMatrix(lda_table_training)
>
> print("Learning Phase Confusion Matrix")
[1] "Learning Phase Confusion Matrix"
Confusion Matrix and Statistics

          observed
predicted   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
        1 444   0   0   0   0   0   6   4   0   0   0   2   2   1   9   0  13   0   9   0   0   0   0   0   0   1
        2   1 418   1  26  28  30  23   9  13   8  11  17   0   0   1  13  20  44  45   6   0   3   0  13   0   1
        3   0   0 371   0  53   0  80   1   1   0   3  10   0   0   2   0   0   0   0   0   2   0   0   0   0   0
        4   0  11   0 417   2   3   2  28   3   5   3   0   0  16  45   3   1  17   2   1   2   0   0   4   1   0
        5   0   2  17   0 260   2   5   0   0   0  26  18   0   0   0   0   2   1   1  16   0   0   0  10   0  23
        6   0   0   4   1   5 377   0   0   8  12   0   0   0   0   1  61   0   0  13  34   0   5   0   0  32   0
        7   1   4  20   4  31   3 276   2   2   0   1  24   0   0   5   6  20   1  20  17   0   0   0   1   0   5
        8   6  39   1  24   3  15  26 179   0   8  21   3  10  22  13   6   3  27   0  18  16  28  23   5   2   1
        9   0   1   0   2   1   0   0   0 434  50   1   1   0   0   0   1   0   0   1   1   0   0   0   4   0   0
        10 12   0   0   4   0   0   0   0  11 327   0   7   1   0   2   1   5   0   7   0   0   0   0   0   0  15
        11  3   2  33   1   8   1  17  18   0   0 337   8   3   7   1   5   6  20   0   7   1   4   2   4   0   0
        12  0   0   0   0   1   0   4   0   5   0   0 301   0   0   1   0   6   0  23   0   2   0   0   0   0   0
        13  4   0   1   3   0   0   1   3   0   0   1   0 385   7   2   0   0   0   0   0  15   2   5   0   2   0
        14  2   0   0   8   0   5   1  22   0   1  10   0  19 329   6   2   1   7   0   0  17   2  11   0   1   0
        15 16   4  14  15   0   0  10  63   1   7   0   0   1  12 407   9  61   0   2   0  35   0   2   0   4   0
        16  0   0   0   0   0  47   0   9   5   4   0   0   0   0   0 406   0   0   1   5   0   1   0   0   0   0
        17  2   0   0   0   4  11  14  12   7  12   0   2   0   0   9  10 323   0   4   0   0   0   0  20  36  15
        18  0  47   1  15  10   2  16  24   0   0  44   2   3   1   2   1   1 373  18   1   0   1   1   6   0   2
        19  5  10  16   4  25  18  23   0  28  22   0  10   0   0   0   0  37   0 296  13   0   0   0  35  20  49
        20  1   0   4   0   1  10   0   1   0   0   0   0   0   0   1   2   0   0   0 369   0   4   0   0  44   1
        21  6   0   6   1   3   1   0  10   0   0  23   0   2   1   5   1   0   0   0   0 360   2   4  17   1   0
        22  0   0   0   0   0   0   0   5   0   0   1   0   0   4   0   1   1   0   1   1   0 439   1   0  74   0
        23  4   0   1   0   0   4   3   4   0   0   1   0   5  21  12   6   1   0   0   0   5  12 398   0   0   0
        24  5   5   0  22  25   6   4   5   5  12  20  11   0   0   1   0   3  11  26   8   0   1   0 402   1   4
        25  4   0   0   0   0   1   0   0   8   0   0   0   0   0   0   7   0   0   5   5   0   9   0   6 307   0
        26  0   0   0   0  17   1   0   0   1   1   0   0   0   0   0   0   0   0  55   8   0   0   0   4   0 294

Overall Statistics

               Accuracy : 0.7219
                 95% CI : (0.7141, 0.7297)
    No Information Rate : 0.0428
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7107

 Mcnemar's Test P-Value : NA


Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
Sensitivity           0.86047  0.76980  0.75714  0.76234  0.54507  0.70205  0.54012  0.44862  0.81579   0.69723   0.66998   0.72356
Specificity           0.99617  0.97443  0.98764  0.98782  0.99001  0.98563  0.98639  0.97416  0.99486   0.99472   0.98770   0.99660
Pos Pred Value        0.90428  0.57182  0.70937  0.73675  0.67885  0.68174  0.62302  0.35872  0.87324   0.83418   0.69057   0.87755
Neg Pred Value        0.99414  0.98963  0.99029  0.98936  0.98250  0.98692  0.98096  0.98209  0.99202   0.98854   0.98650   0.99076
Prevalence            0.04036  0.04247  0.03833  0.04279  0.03731  0.04201  0.03997  0.03121  0.04161   0.03669   0.03935   0.03254
Detection Rate        0.03473  0.03270  0.02902  0.03262  0.02034  0.02949  0.02159  0.01400  0.03395   0.02558   0.02636   0.02355
Detection Prevalence  0.03841  0.05718  0.04091  0.04427  0.02996  0.04326  0.03465  0.03903  0.03888   0.03066   0.03817   0.02683
Balanced Accuracy     0.92832  0.87211  0.87239  0.87508  0.76754  0.84384  0.76326  0.71139  0.90532   0.84598   0.82884   0.86008
                     Class: 13 Class: 14 Class: 15 Class: 16 Class: 17 Class: 18 Class: 19 Class: 20 Class: 21 Class: 22 Class: 23
Sensitivity            0.89327   0.78147   0.77524   0.75046   0.64087   0.74451   0.55955   0.72353   0.79121   0.85575   0.89038
Specificity            0.99628   0.99070   0.97912   0.99412   0.98713   0.98388   0.97430   0.99438   0.99327   0.99275   0.99360
Pos Pred Value         0.89327   0.74099   0.61388   0.84937   0.67152   0.65324   0.48445   0.84247   0.81264   0.83144   0.83438
Neg Pred Value         0.99628   0.99254   0.99026   0.98903   0.98529   0.98952   0.98086   0.98858   0.99230   0.99396   0.99602
Prevalence             0.03371   0.03293   0.04107   0.04232   0.03942   0.03919   0.04138   0.03989   0.03559   0.04013   0.03497
Detection Rate         0.03012   0.02574   0.03184   0.03176   0.02527   0.02918   0.02315   0.02886   0.02816   0.03434   0.03113
Detection Prevalence   0.03371   0.03473   0.05186   0.03739   0.03763   0.04467   0.04779   0.03426   0.03465   0.04130   0.03731
Balanced Accuracy      0.94477   0.88609   0.87718   0.87229   0.81400   0.86420   0.76692   0.85895   0.89224   0.92425   0.94199
                     Class: 24 Class: 25 Class: 26
Sensitivity            0.75706   0.58476   0.71533
Specificity            0.98572   0.99633   0.99297
Pos Pred Value         0.69671   0.87216   0.77165
Neg Pred Value         0.98943   0.98246   0.99057
Prevalence             0.04154   0.04107   0.03215
Detection Rate         0.03145   0.02401   0.02300
Detection Prevalence   0.04513   0.02753   0.02980
Balanced Accuracy      0.87139   0.79055   0.85415
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

Classification accuracy of Learning Phase

```
> # Accuracy of predictions with train data
> lda_acc_tr=round(lda_cfm_training$overall[["Accuracy"]],4)
> print(paste("Classification accuracy of learning phase =",lda_acc_tr*100))
[1] "Classification accuracy of learning phase = 72.19"
```

LDA Generalization Phase

```
> ##LDA - Generalization Phase
> lda.model.test.withoutOutliers <- lda(Letter~., data = testingDataset_withoutOutliers)
> # Make predictions for Test data set
> predictions.test.withoutOutliers <- lda.model.test.withoutOutliers %>% predict(testingDataset_withoutOutliers)
> # Test error
> lda.withoutOutliers.test.error <- mean(predictions.test.withoutOutliers$class == testingDataset_withoutOutliers$Letter)
>
> print(paste("LDA Test Rate =",round(lda.withoutOutliers.test.error*100, 4)))
[1] "LDA Test Rate = 72.281"
```

LDA Generalization Phase Confusion Matrix

```
> # Confusion Matrix
> lda_table_testing <- table(list(predicted=predictions.test.withoutOutliers$class, observed=testingDataset_withoutOutliers$Letter))
> lda_cfm_testing <- confusionMatrix(lda_table_testing)
> print("Generalization Phase Confusion Matrix")
[1] "Generalization Phase Confusion Matrix"
> lda_cfm_testing
Confusion Matrix and Statistics
```

	observed																								
predicted	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	191	0	0	0	0	0	3	1	0	0	0	0	2	0	3	0	4	0	3	0	0	0	0	0	0	1
2	0	150	1	9	13	16	11	4	6	0	2	12	0	0	0	4	7	12	18	2	0	0	0	4	0	0
3	0	0	165	0	31	0	40	0	0	0	4	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	0	4	0	196	0	1	1	7	2	2	0	0	0	7	14	2	1	10	2	0	0	0	0	2	1	0
5	0	1	9	0	123	0	1	0	1	0	5	11	0	0	0	0	5	0	3	3	0	0	0	4	0	10
6	0	2	3	0	1	142	0	1	3	5	0	0	0	0	0	16	0	0	8	8	0	2	0	0	6	2
7	0	2	10	0	20	5	129	3	0	0	1	15	0	0	7	8	13	1	5	8	0	0	0	0	0	1
8	0	12	1	10	2	4	8	81	0	0	9	1	7	3	6	2	1	16	0	3	8	18	10	1	2	0
9	0	0	0	1	0	2	0	0	180	19	0	0	0	0	0	0	0	0	1	2	0	0	0	3	0	1
10	3	0	0	1	0	0	0	0	3	121	0	2	0	0	2	0	0	0	1	0	0	0	0	1	0	4
11	4	0	21	0	3	0	10	2	0	0	155	2	2	2	0	1	6	9	0	4	2	1	1	6	0	0
12	0	0	0	0	0	0	0	1	7	0	0	116	0	0	1	0	2	0	4	0	0	0	0	0	0	0
13	3	0	1	1	0	0	1	0	0	0	1	0	171	3	2	0	0	0	0	0	6	0	7	0	0	0
14	1	0	0	4	0	3	0	16	0	1	0	0	1	150	3	1	0	2	0	0	4	0	3	0	0	0
15	6	2	5	5	0	0	5	24	0	5	0	0	0	2	163	4	30	0	1	1	14	0	0	0	2	0
16	0	0	0	0	0	18	0	1	0	2	0	0	0	1	0	174	0	0	0	5	0	1	0	1	0	0
17	0	0	0	0	4	5	9	4	2	5	0	1	0	0	2	2	152	0	0	0	0	0	0	11	20	5
18	1	14	0	4	3	0	7	14	1	0	18	1	7	0	4	0	0	186	8	1	0	0	0	1	0	1
19	5	13	3	5	6	4	12	0	7	9	0	2	0	0	0	1	7	0	85	6	0	0	0	11	7	20
20	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	165	0	3	0	2	17	0
21	0	0	2	0	3	0	0	8	0	0	7	0	1	1	2	0	0	0	0	2	146	1	0	10	2	0
22	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	1	0	0	192	1	0	50	0
23	0	1	1	0	0	0	2	0	0	0	0	0	3	13	4	2	0	0	0	0	1	7	184	0	0	0
24	4	7	0	11	13	3	1	5	4	4	7	4	0	0	1	0	0	6	12	4	0	1	0	180	1	2
25	5	0	0	0	0	1	0	1	3	0	0	0	0	0	0	9	0	0	0	0	1	2	0	1	118	0
26	0	0	0	1	2	0	0	0	3	1	0	0	0	0	0	0	3	0	24	2	0	0	0	2	0	146

Overall Statistics

```
Overall Statistics

               Accuracy : 0.7228
                 95% CI : (0.7108, 0.7346)
    No Information Rate : 0.0453
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7116

 Mcnemar's Test P-Value : NA
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

Statistics by Class

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
Sensitivity           0.85650  0.72115  0.74324  0.79032  0.54911  0.69268  0.53750  0.46552  0.81081   0.69540   0.73460   0.68235
Specificity           0.99677  0.97705  0.98498  0.98930  0.98992  0.98919  0.98111  0.97663  0.99448   0.99680   0.98558   0.99718
Pos Pred Value        0.91827  0.55351  0.67623  0.77778  0.69886  0.71357  0.56579  0.39512  0.86124   0.87681   0.67100   0.88550
Neg Pred Value        0.99393  0.98887  0.98911  0.99005  0.98096  0.98807  0.97887  0.98237  0.99203   0.99008   0.98933   0.98990
Prevalence            0.04069  0.03796  0.04051  0.04526  0.04088  0.03741  0.04380  0.03175  0.04051   0.03175   0.03850   0.03102
Detection Rate        0.03485  0.02737  0.03011  0.03577  0.02245  0.02591  0.02354  0.01478  0.03285   0.02208   0.02828   0.02117
Detection Prevalence  0.03796  0.04945  0.04453  0.04599  0.03212  0.03631  0.04161  0.03741  0.03814   0.02518   0.04215   0.02391
Balanced Accuracy     0.92663  0.84910  0.86411  0.88981  0.76951  0.84094  0.75930  0.72107  0.90265   0.84610   0.86009   0.83976
                     Class: 13 Class: 14 Class: 15 Class: 16 Class: 17 Class: 18 Class: 19 Class: 20 Class: 21 Class: 22 Class: 23
Sensitivity            0.88144   0.81967   0.76168   0.76652   0.65801   0.76860   0.48295   0.76389   0.79781   0.84211   0.89320
Specificity            0.99527   0.99264   0.97987   0.99448   0.98666   0.98377   0.97775   0.99525   0.99264   0.98953   0.99355
Pos Pred Value         0.87245   0.79365   0.60595   0.85714   0.68468   0.68635   0.41872   0.86842   0.78919   0.77733   0.84404
Neg Pred Value         0.99565   0.99376   0.99021   0.98996   0.98498   0.98925   0.98276   0.99036   0.99301   0.99312   0.99582
Prevalence             0.03540   0.03339   0.03905   0.04142   0.04215   0.04416   0.03212   0.03942   0.03339   0.04161   0.03759
Detection Rate         0.03120   0.02737   0.02974   0.03175   0.02774   0.03394   0.01551   0.03011   0.02664   0.03504   0.03358
Detection Prevalence   0.03577   0.03449   0.04909   0.03704   0.04051   0.04945   0.03704   0.03467   0.03376   0.04507   0.03978
Balanced Accuracy      0.93836   0.90615   0.87078   0.88050   0.82234   0.87618   0.73035   0.87957   0.89523   0.91582   0.94338
                     Class: 24 Class: 25 Class: 26
Sensitivity            0.75000   0.52212   0.75648
Specificity            0.98282   0.99562   0.99281
Pos Pred Value         0.66667   0.83688   0.79348
Neg Pred Value         0.98848   0.97977   0.99113
Prevalence             0.04380   0.04124   0.03522
Detection Rate         0.03285   0.02153   0.02664
Detection Prevalence   0.04927   0.02573   0.03358
Balanced Accuracy      0.86641   0.75887   0.87464
```

Classification Accuracy of the Generalization Phase

```
> # Accuracy of predictions with test data
> lda_acc_tst=round(lda_cfm_testing$overall[["Accuracy"]],4)
>
> print(paste("Classification accuracy of Generalization phase =",lda_acc_tst*100))
[1] "Classification accuracy of Generalization phase = 72.28"
```

Checking for over fitting

```
> ## LDA Checking for over fitting
> # Check for over-fitting. Criteria: Accuracy change from train to test > 25%
> lda_model_isOF=abs((lda_acc_tr-lda_acc_tst)/lda_acc_tr)
> if(lda_model_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fitting")
[1] "Model is not over-fitting"
```

Accuracy Drop

```
> lda_model_isOF=round(lda_model_isOF,4)
> print(paste("Accuracy drop from training data to test data is",lda_model_isOF*100,"%"))
[1] "Accuracy drop from training data to test data is 0.12 %"
```

LDA Model Performance Metrics

LDA - Learning Phase

```
> ### LDA Model Performance Metrics
> print("LDA Learning-Phase Performance Parameters:")
[1] "LDA Learning-Phase Performance Parameters:"
> lda_PM_tr= lda_cfm_training$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
> lda_PMavg_tr=round(apply(lda_PM_tr,2,mean),4)
> print("Macro Averages:")
[1] "Macro Averages:"
> t(lda_PMavg_tr)
     Balanced Accuracy Precision Sensitivity Specificity Recall
[1,]            0.8551    0.7349      0.7214      0.9889 0.7214
```

LDA Learning Phase AUC

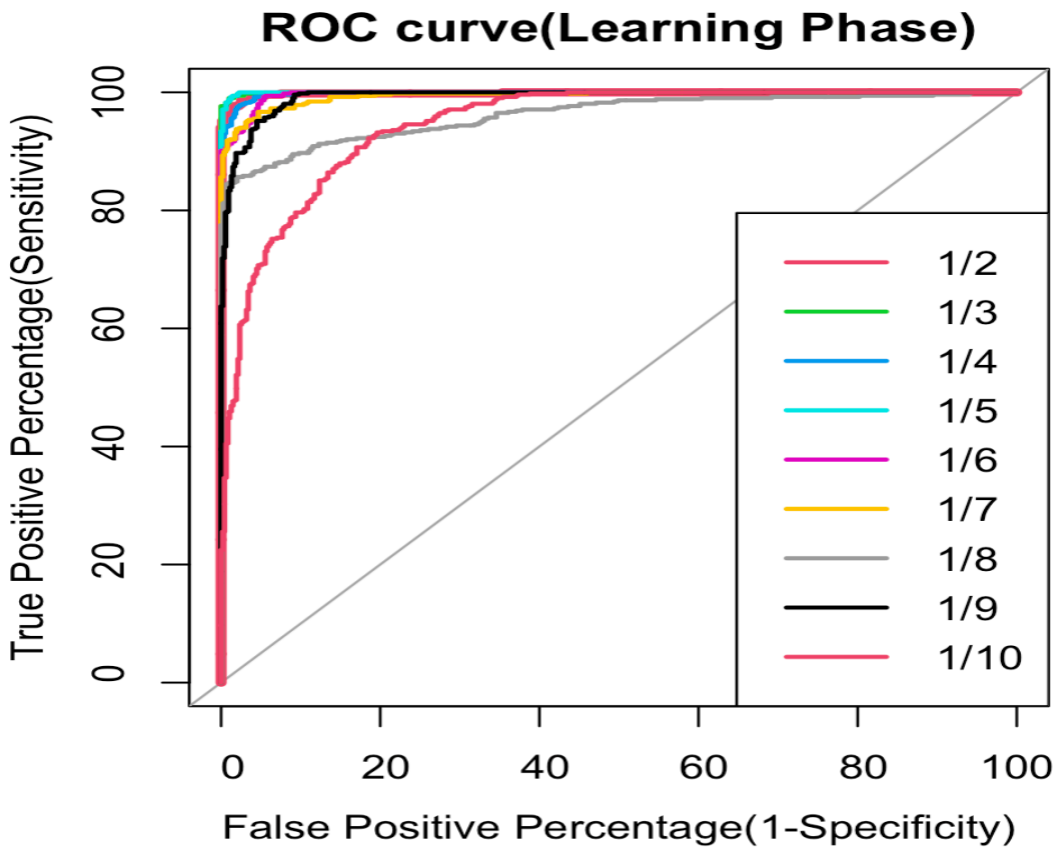
The AUC of the model is very close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> lda_prob_tr = predict(lda.model.withoutOutliers, trainingDataset_withoutOutliers[,-which(names(trainingDataset_withoutOutliers)=="Letter")],
 type="prob")
> lda_AUC_tr = multiclass.roc(trainingDataset_withoutOutliers[,which(names(trainingDataset_withoutOutliers)=="Letter")],lda_prob_tr$posterior,
 percent=TRUE)
> print(paste("LAD Learning-Phase AUC:",round(lda_AUC_tr$auc,4)))
[1] "LAD Learning-Phase AUC: 96.7069"
```

LDA Learning Phase ROC

As this is a multiclass problem, we can plot the N number of AUC ROC Curves for N number classes using the One vs ALL methodology. Here, we will plot the ROC using multiclass roc function provided by 'R' of 10 of the classes. The ROC plot for these classes is as shown below.

```
> lda_ROC_tr <- lda_AUC_tr[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(lda_ROC_tr[[ROC_num]][[2]], col=2, main="ROC curve(Learning Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentage(Sensitivity)",lwd=4)
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(lda_ROC_tr[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/9','1/10'), col=2:10, lwd=2)
```



LDA Generalization Phase

LDA Generalization Phase Performance Parameters

```
[1] "LDA Generalization-Phase Performance Parameters:"
> lda_PM_tst= lda_cfm_testing$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Spe
cificity", "Recall")]
>
> lda_PMavg_tst=round(apply(lda_PM_tst,2,mean),4)
>
> print("Macro Averages:")
[1] "Macro Averages:"
> t(lda_PMavg_tst)
     Balanced Accuracy Precision Sensitivity Specificity Recall
[1,]            0.8549    0.7342      0.7209      0.9889 0.7209
```

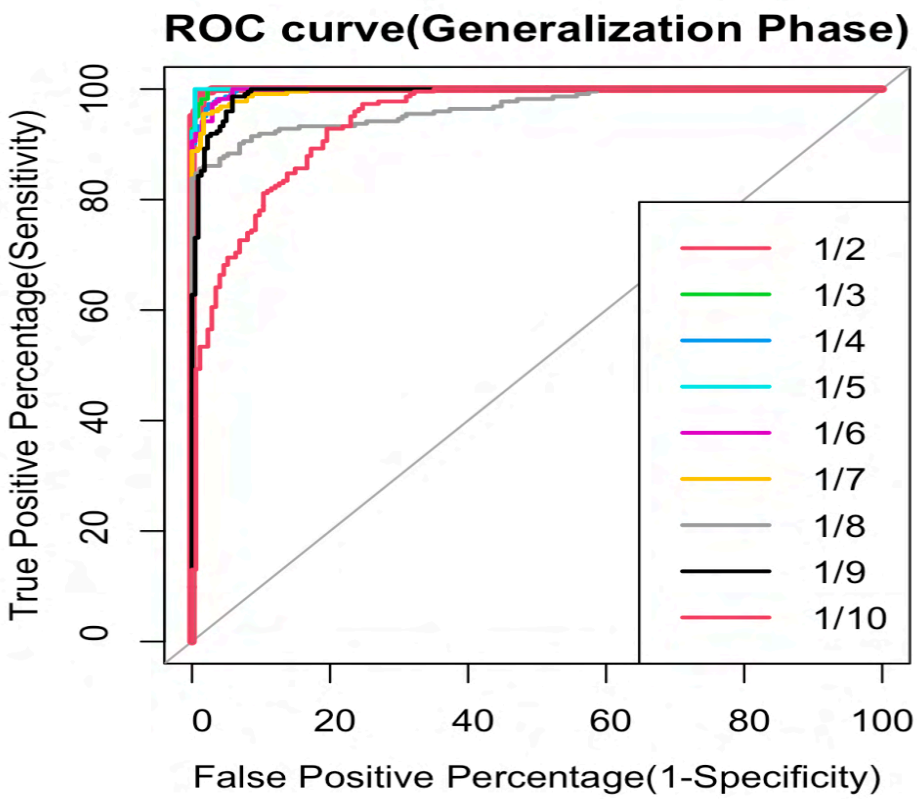
LDA Generalization Phase AUC

The AUC of the model is very close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> lda_prob_tst = predict(lda.model.test.withoutOutliers, testingDataset_withoutOutliers[,-which(names
(testingDataset_withoutOutliers)=="Letter")], type="prob")
> lda_prob_tst = predict(lda.model.test.withoutOutliers, testingDataset_withoutOutliers[,-which(names
(testingDataset_withoutOutliers)=="Letter")], type="prob")
>
> lda_AUC_tst = multiclass.roc(testingDataset_withoutOutliers[,which(names(testingDataset_withoutOutl
iers)=="Letter")],lda_prob_tst$posterior, percent=TRUE)
>
> print(paste("LAD Generalization-Phase AUC:",round(lda_AUC_tst$auc,4)))
[1] "LAD Generalization-Phase AUC: 96.9243"
```

Observation for Logistic-Regression

```
> lda_ROC_tst <- lda_AUC_tst[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(lda_ROC_tst[[ROC_num]][[2]], col=2, main="ROC curve(Generalization P
hase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", y
lab="True Positive Percentage(Sensitivity)",lwd=4)
>
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(lda_ROC_tst[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/
9','1/10'), col=2:10, lwd=2)
```



Generalization Confusion Matrix

Confusion Matrix and Statistics shows how many classes were observed correctly and how many classes were observed incorrectly.

```
> print("Generalization Confusion Matrix")
[1] "Generalization Confusion Matrix"
> lda_cfm_testing
Confusion Matrix and Statistics

          observed
predicted   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
        1 191   0   0   0   0   0   3   1   0   0   0   0   2   0   3
        2   0 150   1   9  13  16  11   4   6   0   2  12   0   0   0
        3   0   0 165   0  31   0  40   0   0   0   4   3   0   0   0
        4   0   4   0 196   0   1   1   7   2   2   0   0   0   7  14
        5   0   1   9   0 123   0   1   0   1   0   5  11   0   0   0
        6   0   2   3   0   1 142   0   1   3   5   0   0   0   0   0
        7   0   2  10   0  20   5 129   3   0   0   1  15   0   0   7
        8   0  12   1  10   2   4   8  81   0   0   9   1   7   3   6
        9   0   0   0   1   0   2   0   0 180  19   0   0   0   0   0
        10  3   0   0   1   0   0   0   0   3 121   0   2   0   0   2
        11  4   0  21   0   3   0  10   2   0   0 155   2   2   2   0
        12  0   0   0   0   0   0   0   1   7   0   0 116   0   0   1
        13  3   0   1   1   0   0   1   0   0   0   1   0 171   3   2
        14  1   0   0   4   0   3   0  16   0   1   0   0   1 150   3
        15  6   2   5   5   0   0   5  24   0   5   0   0   0   2 163
        16  0   0   0   0   0  18   0   1   0   2   0   0   0   1   0
        17  0   0   0   0   4   5   9   4   2   5   0   1   0   0   2
        18  1  14   0   4   3   0   7  14   1   0  18   1   7   0   4
        19  5  13   3   5   6   4  12   0   7   9   0   2   0   0   0
        20  0   0   0   0   0   1   0   0   0   0   0   0   0   1   0
        21  0   0   2   0   3   0   0   8   0   0   7   0   1   1   2
        22  0   0   0   0   0   0   0   1   0   0   2   0   0   0   0
```

```
       23   0   1   1   0   0   0   2   0   0   0   0   0   3  13   4
       24   4   7   0  11  13   3   1   5   4   4   7   4   0   0   1
       25   5   0   0   0   0   1   0   1   3   0   0   0   0   0   0
       26   0   0   0   1   2   0   0   0   3   1   0   0   0   0   0
         observed
predicted  16  17  18  19  20  21  22  23  24  25  26
       1    0   4   0   3   0   0   0   0   0   0   1
       2    4   7  12  18   2   0   0   0   4   0   0
       3    0   0   0   0   0   1   0   0   0   0   0
       4    2   1  10   2   0   0   0   0   2   1   0
       5    0   5   0   3   3   0   0   0   4   0  10
       6   16   0   0   8   8   0   2   0   0   6   2
       7    8  13   1   5   8   0   0   0   0   0   1
       8    2   1  16   0   3   8  18  10   1   2   0
       9    0   0   0   1   2   0   0   0   3   0   1
       10   0   0   0   1   0   0   0   0   1   0   4
       11   1   6   9   0   4   2   1   1   6   0   0
       12   0   2   0   4   0   0   0   0   0   0   0
       13   0   0   0   0   0   6   0   7   0   0   0
       14   1   0   2   0   0   4   0   3   0   0   0
       15   4  30   0   1   1  14   0   0   0   2   0
       16 174   0   0   0   5   0   1   0   1   0   0
       17   2 152   0   0   0   0   0   0  11  20   5
       18   0   0 186   8   1   0   0   0   1   0   1
       19   1   7   0  85   6   0   0   0  11   7  20
       20   1   0   0   0 165   0   3   0   2  17   0
       21   0   0   0   0   2 146   1   0  10   2   0
       22   0   0   0   1   0   0 192   1   0  50   0
       23   2   0   0   0   0   1   7 184   0   0   0
       24   0   0   6  12   4   0   1   0 180   1   2
       25   9   0   0   0   0   1   2   0   1 118   0
       26   0   3   0  24   2   0   0   0   2   0 146
```

Overall Statistics of the Generalization Phase

```
Overall Statistics

               Accuracy : 0.7228
                 95% CI : (0.7108, 0.7346)
    No Information Rate : 0.0453
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7116

 Mcnemar's Test P-Value : NA
```

As it is a multiclass classification, we get Statistics for each of the classes which as shown below:

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5
Sensitivity	0.85650	0.72115	0.74324	0.79032	0.54911
Specificity	0.99677	0.97705	0.98498	0.98930	0.98992
Pos Pred Value	0.91827	0.55351	0.67623	0.77778	0.69886
Neg Pred Value	0.99393	0.98887	0.98911	0.99005	0.98096
Prevalence	0.04069	0.03796	0.04051	0.04526	0.04088
Detection Rate	0.03485	0.02737	0.03011	0.03577	0.02245
Detection Prevalence	0.03796	0.04945	0.04453	0.04599	0.03212
Balanced Accuracy	0.92663	0.84910	0.86411	0.88981	0.76951

	Class: 6	Class: 7	Class: 8	Class: 9	Class: 10
Sensitivity	0.69268	0.53750	0.46552	0.81081	0.69540
Specificity	0.98919	0.98111	0.97663	0.99448	0.99680
Pos Pred Value	0.71357	0.56579	0.39512	0.86124	0.87681
Neg Pred Value	0.98807	0.97887	0.98237	0.99203	0.99008
Prevalence	0.03741	0.04380	0.03175	0.04051	0.03175
Detection Rate	0.02591	0.02354	0.01478	0.03285	0.02208
Detection Prevalence	0.03631	0.04161	0.03741	0.03814	0.02518
Balanced Accuracy	0.84094	0.75930	0.72107	0.90265	0.84610

	Class: 11	Class: 12	Class: 13	Class: 14	Class: 15
Sensitivity	0.73460	0.68235	0.88144	0.81967	0.76168
Specificity	0.98558	0.99718	0.99527	0.99264	0.97987
Pos Pred Value	0.67100	0.88550	0.87245	0.79365	0.60595
Neg Pred Value	0.98933	0.98990	0.99565	0.99376	0.99021
Prevalence	0.03850	0.03102	0.03540	0.03339	0.03905
Detection Rate	0.02828	0.02117	0.03120	0.02737	0.02974
Detection Prevalence	0.04215	0.02391	0.03577	0.03449	0.04909
Balanced Accuracy	0.86009	0.83976	0.93836	0.90615	0.87078

	Class: 16	Class: 17	Class: 18	Class: 19	Class: 20
Sensitivity	0.76652	0.65801	0.76860	0.48295	0.76389
Specificity	0.99448	0.98666	0.98377	0.97775	0.99525
Pos Pred Value	0.85714	0.68468	0.68635	0.41872	0.86842
Neg Pred Value	0.98996	0.98498	0.98925	0.98276	0.99036
Prevalence	0.04142	0.04215	0.04416	0.03212	0.03942
Detection Rate	0.03175	0.02774	0.03394	0.01551	0.03011
Detection Prevalence	0.03704	0.04051	0.04945	0.03704	0.03467
Balanced Accuracy	0.88050	0.82234	0.87618	0.73035	0.87957

	Class: 21	Class: 22	Class: 23	Class: 24	Class: 25
Sensitivity	0.79781	0.84211	0.89320	0.75000	0.52212
Specificity	0.99264	0.98953	0.99355	0.98282	0.99562
Pos Pred Value	0.78919	0.77733	0.84404	0.66667	0.83688
Neg Pred Value	0.99301	0.99312	0.99582	0.98848	0.97977
Prevalence	0.03339	0.04161	0.03759	0.04380	0.04124
Detection Rate	0.02664	0.03504	0.03358	0.03285	0.02153
Detection Prevalence	0.03376	0.04507	0.03978	0.04927	0.02573
Balanced Accuracy	0.89523	0.91582	0.94338	0.86641	0.75887

	Class: 26
Sensitivity	0.75648
Specificity	0.99281
Pos Pred Value	0.79348
Neg Pred Value	0.99113
Prevalence	0.03522
Detection Rate	0.02664
Detection Prevalence	0.03358
Balanced Accuracy	0.87464

```
> ## Variance Estimation
> # Variance Estimation for Learning Phase
> varEst_tridx=sample(1:nrow(trainingDataset_withoutOutliers), 0.9*nrow(trainingDataset_withou
tOutliers), replace=F)
> # Define training data for variance estimation
> varEst_trdf=trainingDataset_withoutOutliers[varEst_tridx,]
> varEst_tstdf=trainingDataset_withoutOutliers[-varEst_tridx,]
```

Variance Estimation

Variance helps us to understand how much the ML function can adjust depending on the given data set. Typically, a model with Models with high bias will have low variance & models with high variance will have a low bias

Let's see the process of doing variance estimation:
The variance estimation function was created to estimate the variance in the model using 30%,60% and 100% of the data.
The process includes partitioning the data into two subsets, one includes 90% and the other includes 10% of the total number of samples. The 10%subset is reserved for testing, while the 90% subset is placed inside a loop where, in each iteration, a percentage of samples are drawn and used to train a model, and the classification accuracy is computed using the reserved 10% subset. The variance of the accuracies obtained in each iteration would represent the variance estimation for the model. Since we will be using the same function in the subsequent sections, it includes four different model types; LDA, QDA, SVM, kNN and tree. The results of variance estimation for the LDA model, using 30%, 60% and 100% of the data, are shown in the output below.

First let's divide the data into two subsets, 90% and 10% of the data as shown below:

```
> ## Variance Estimation
> # Variance Estimation for Learning Phase
> varEst_tridx=sample(1:nrow(trainingDataset_withoutOutliers), 0.9*nrow(trainingDataset_withou
tOutliers), replace=F)
> # Define training data for variance estimation
> varEst_trdf=trainingDataset_withoutOutliers[varEst_tridx,]
> varEst_tstdf=trainingDataset_withoutOutliers[-varEst_tridx,]
```

Let's define the function variance estimation function for all the models:

```
+ varEst=function(trdf,tstdf,percent,type){
+   target_idx=which(colnames(trdf)=="Letter")
+   acc_varEstp=c(); # Initialize a variable to store the accuracies computed in the loop
+   for(i in 1:10){
+     varEstp_tridx=sample(1:nrow(trdf), percent/100*nrow(trdf), replace=F) # Take samples, pe
rcent% of the d
+     varEstp_trdf=trdf[varEstp_tridx,]
+     # For qda
+     if(type=="lda"){
+       lda_model_varEstp= lda(Letter~., data = varEstp_trdf) #Train a Logistic model
+       pred_varEstp= predict(lda_model_varEstp, tstdf[,-target_idx]) # Predict with varia
+       pred_varEstp = pred_varEstp$class
+     }
+     else if(type=="qda"){
+       lda_model_varEstp= qda(Letter~., data = varEstp_trdf) #Train a Logistic model
+       pred_varEstp= predict(lda_model_varEstp, tstdf[,-target_idx]) # Predict with varia
+       pred_varEstp = pred_varEstp$class
+     }
+     # For SVM
+     else if(type=="svm"){
+       svm_model_varEstp = svm(Letter~., varEstp_trdf)
+       pred_varEstp = predict(svm_model_varEstp, tstdf[,-target_idx])
+     }
+     # For Tree
+     else if(type=="tree"){
+       tree_model_varEstp = rpart(Letter~., varEstp_trdf, method = "class")
+       pred_varEstp = predict(tree_model_varEstp, tstdf[,-target_idx], type="class")
+     }
```

```
+     # For KNN
+     else if(type=="knn"){
+       trclass=factor(varEstp_trdf[,target_idx])
+       tstclass=factor(tstdf[,target_idx])
+       pred_varEstp=knn(varEstp_trdf[,-target_idx], tstdf[,-target_idx], trclass, k = 15, pro
b=TRUE)
+     }
+     else {
+       print("type should be 'lda' 'qda' 'tree' 'svm' or 'knn'")
+       return()
+     }
+     u_varEstp=union(pred_varEstp, tstdf[,target_idx]) # Avoids issues when number of classes
 are not equal
+     t_varEstp=table(factor(pred_varEstp, u_varEstp), factor(tstdf[,target_idx],u_varEstp))
+     mn_cfm_varEstp=confusionMatrix(t_varEstp) # Confusion Matrix
+     mn_acc_varEstp=mn_cfm_varEstp$overall[["Accuracy"]] # Accuracy of predictions
+     acc_varEstp=c(acc_varEstp,mn_acc_varEstp) # Store
+   }
+   mean_varEstp = signif(mean(acc_varEstp),4)
+   var_varEstp = signif(var(acc_varEstp),4)
+   varEstp = data.frame(mean_varEstp,var_varEstp)
+   names(varEstp) = c("Mean of Accuracies","Variance of Accuracies")
+   return(t(varEstp))
+ }
```

LDA Variance Estimation using 30%, 60% and 100% of the data.

```
> # Variance estimation using 30% of the data
> lda_varEst30=varEst(varEst_trdf, varEst_tstdf, 30, type="lda")
> # Variance estimation using 60% of the data
> lda_varEst60=varEst(varEst_trdf, varEst_tstdf, 60, type="lda")
> # Variance estimation using 100% of the data
> lda_varEst100=varEst(varEst_trdf, varEst_tstdf, 100, type="lda")
```

Variance Estimation using 30% of the data:

```
[1] "LDA Variance Estimation using 30% of data:"
> lda_varEst30
                            [,1]
Mean of Accuracies     7.196e-01
Variance of Accuracies 3.195e-05
```

Variance Estimation using 60% of the data:

```
> print("LDA Variance Estimation using 60% of data:")
[1] "LDA Variance Estimation using 60% of data:"
> lda_varEst60
                            [,1]
Mean of Accuracies     0.7263000
Variance of Accuracies 0.0000222
```

Variance Estimation using 100% of the data:

```
> print("LDA Variance Estimation using 100% of data:")
[1] "LDA Variance Estimation using 100% of data:"
> lda_varEst100
                          [,1]
Mean of Accuracies     0.7326
Variance of Accuracies 0.0000
```

LDA with Correlated Feature Variables Removed

We have removed feature variables V3, V4, V5, V6 and V16 which had high correlation

```
> dataset_without_outliers <- dataset[-c(outliers_row),]
> dataWithoutCorrelated_1 = dataset_without_outliers
>
> dataWithoutCorrelated_1 = dataWithoutCorrelated_1[,-which(names(dataWithoutCorrelated_1)=="V
3")]
> dataWithoutCorrelated_1 = dataWithoutCorrelated_1[,-which(names(dataWithoutCorrelated_1)=="V
4")]
> dataWithoutCorrelated_1 = dataWithoutCorrelated_1[,-which(names(dataWithoutCorrelated_1)=="V
5")]
> dataWithoutCorrelated_1 = dataWithoutCorrelated_1[,-which(names(dataWithoutCorrelated_1)=="V
6")]
> dataWithoutCorrelated_1 = dataWithoutCorrelated_1[,-which(names(dataWithoutCorrelated_1)=="V
12")]
>
> randomized__withoutCorrelation=dataWithoutCorrelated_1[sample(1:nrow(dataWithoutCorrelated_
1),nrow(dataWithoutCorrelated_1)),]
> tridx_withoutCorrelation = sample(1:nrow(dataWithoutCorrelated_1),0.7*nrow(dataWithoutCorrel
ated_1),replace=F)
> trainingDataset_withoutCorrelation = randomized__withoutCorrelation[tridx_withoutCorrelatio
n,]
> testingDataset_withoutCorrelation = randomized__withoutCorrelation[-tridx_withoutCorrelatio
n,]
```

LDA Learning Phase

```
> # Fit the model - Learning Phase
> lda.model.withoutOutliers <- lda(Letter~., data = trainingDataset_withoutCorrelation)
> # Make predictions for training data set
> predictions.withoutOutliers <- lda.model.withoutOutliers %>% predict(trainingDataset_without
Correlation)
> # Train error
> lda.withoutOutliers.error <- mean(predictions.withoutOutliers$class == trainingDataset_witho
utCorrelation$Letter)
>
> print(paste("LDA Train Rate =",round(lda.withoutOutliers.error*100, 4)))
[1] "LDA Train Rate = 68.6796"
```

The train Rate for LDA with correlated variables removed is 68.6796. And we observed that the train rate without removing correlated variables was 72.19, so from the train rate it doesn't look like removing the correlated variables has helped improving the training rate. We can observe the Generalization Phase Test Rate and the AUC to see if there is a need for removal of correlated variables for this data set.

```
> print("LDA Learning Phase Confusion Matrix")
[1] "LDA Learning Phase Confusion Matrix"
> lda_cfm_training <- confusionMatrix(lda_table_training)
> # Confusion Matrix
> lda_table_training <- table(list(predicted=predictions.withoutOutliers$class, observed=train
ingDataset_withoutCorrelation$Letter))
>
> print("LDA Learning Phase Confusion Matrix")
[1] "LDA Learning Phase Confusion Matrix"
> lda_cfm_training <- confusionMatrix(lda_table_training)
>
>
> print("Learning Phase Confusion Matrix")
[1] "Learning Phase Confusion Matrix"
> lda_cfm_training
```

```
Overall Statistics

               Accuracy : 0.6868
                 95% CI : (0.6787, 0.6948)
    No Information Rate : 0.0446
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6741

 Mcnemar's Test P-Value : NA
```

LDA Generalization Phase

```
> ##LDA - Generalization Phase
> lda.model.test.withoutOutliers <- lda(Letter~., data = testingDataset_withoutCorrelation)
> # Make predictions for training data set
> predictions.test.withoutOutliers <- lda.model.test.withoutOutliers %>% predict(testingDatase
t_withoutCorrelation)
> # Train error
> lda.withoutOutliers.test.error <- mean(predictions.test.withoutOutliers$class == testingData
set_withoutCorrelation$Letter)
>
> print(paste("LDA Test Rate =",round(lda.withoutOutliers.test.error*100, 4)))
[1] "LDA Test Rate = 68.6861"
> # Accuracy of predictions with train data
> lda_acc_tst=round(lda_cfm_testing$overall[["Accuracy"]],4)
>
> print(paste("Classification accuracy of Generalization phase =",lda_acc_tst))
[1] "Classification accuracy of Generalization phase = 0.6869"
```

LDA Learning Phase Test Rate is 68.6861, the test rate of LDA without Correlated Variables removed was 72.21. Here also we can see that the removing feature doesn't help in Generalization as well.

LDA with Predictor Interaction

For this dataset, let's look at the behavior of LDA with the Predictor Interaction. An interaction effect exists when the effect of an independent variable on a dependent variable change, depending on the value(s) of one or more other independent variables. As we know in the dataset the output variable Letter is dependent on one or more independent feature variables. This should improve the Accuracy of the Learning & Generalization Rate.

LDA with Predictor Interaction Learning Phase

```
> ###############################################################################
> #                      LDA Predictor Interaction                              #
> ###############################################################################
> # New fit with predictors interaction
> # this step takes time
> lda2.fit <- lda(Letter~.*.+.:.:., trainingDataset_withoutOutliers)
> #  Make prediction
> lda2.train.pred <- predict(lda2.fit, trainingDataset_withoutOutliers)
> lda2.train.rate <- mean(lda2.train.pred$class == trainingDataset_withoutOutliers$Letter)
> print(paste("LDA Predictor Interaction Learning Phase Train Rate: ",round(lda2.train.rate*100
4)))
[1] "LDA Predictor Interaction Learning Phase Train Rate:  91.7397"
```

LDA with Predictor Interaction Generalization Phase

```
> #  Make prediction
> lda2.test.pred <- predict(lda2.fit, testingDataset_withoutOutliers)
> lda2.test.class <- lda2.test.pred$class
> # Test error
> lda2.test.rate <- mean(lda2.test.class == testingDataset_withoutOutliers$Letter)
> print(paste("LDA Predictor Interaction Generalization Phase Test Rate: ",round(lda2.test.rate*
100, 4)))
[1] "LDA Predictor Interaction Generalization Phase Test Rate:  89.6533"
>
```

LDA with Predictor Interaction Learning Phase – Confusion Matrix

```
> # Confusion Matrix
> lda_table_training <- table(list(predicted=lda2.train.pred$class, observed=trainingDataset_withoutOutliers$Let
ter))
> lda_cfm_training <- confusionMatrix(lda_table_training)
> print("Learning Phase Confusion Matrix")
[1] "Learning Phase Confusion Matrix"
> lda_cfm_training
Confusion Matrix and Statistics
```

Mohit Shankar Pursnani

Letter Recognition Data Set

mp5578@nyu.edu

Assignment - 2

predicted \ observed	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	472	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	496	0	5	4	4	1	0	3	0	1	3	1	0	0	7	3	12	15	4
3	0	0	426	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	4	0	509	0	0	1	14	2	1	2	0	0	9	2	0	1	2	0	0
5	0	5	14	0	438	5	0	0	1	1	2	8	0	0	0	2	1	0	2	7
6	0	0	0	0	0	514	1	0	5	0	0	0	0	0	0	27	0	0	0	1
7	2	0	20	1	20	0	475	5	3	0	1	9	0	0	2	10	7	1	0	6
8	21	25	3	24	5	6	14	349	1	10	18	7	11	26	4	5	3	29	0	29
9	0	0	0	0	0	2	0	0	482	5	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	414	0	0	0	0	0	0	0	0	0	0
11	2	0	6	0	1	0	0	0	0	0	439	2	0	0	0	1	0	3	0	0
12	0	0	0	0	0	0	0	0	0	0	0	366	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	413	0	0	0	0	0	0	0
14	1	0	0	3	0	1	0	0	0	0	0	0	3	376	0	0	0	0	0	0
15	6	1	16	1	0	1	9	16	1	6	0	0	3	8	513	2	37	3	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	485	0	0	0	0
17	0	0	0	0	0	0	2	0	1	0	0	0	0	0	2	0	448	1	0	0
18	0	6	0	0	1	0	3	12	0	0	23	2	0	2	1	0	0	447	4	1
19	3	5	0	1	1	1	0	0	8	8	0	0	0	0	0	1	2	0	506	4
20	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	451
21	2	0	5	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	6	1	0	3	4	0	4	3	24	24	16	19	0	0	0	0	2	3	2	6
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
26	0	0	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

predicted \ observed	21	22	23	24	25	26
1	0	0	0	0	0	0
2	0	3	0	2	0	0
3	0	0	0	0	0	0
4	0	0	0	1	0	0
5	0	1	0	8	0	6
6	0	1	0	0	1	0
7	1	0	0	0	0	0
8	20	25	9	0	6	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	1	4	0	0
12	0	0	0	0	0	0
13	1	0	0	0	0	0
14	3	1	0	0	0	0
15	14	3	13	3	0	0
16	0	0	0	0	0	0
17	1	0	0	0	1	2
18	0	0	0	0	0	1
19	0	0	0	3	0	15
20	0	0	0	0	4	0
21	415	0	6	0	0	0
22	0	473	2	0	2	0
23	0	1	416	0	0	0
24	0	0	0	510	1	2
25	0	5	0	0	510	0
26	0	0	0	0	0	385

LDA with Predictor Interaction Learning Phase – Overall Statistics

```
Overall Statistics

               Accuracy : 0.9174
                 95% CI : (0.9125, 0.9221)
    No Information Rate : 0.0428
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9141

 Mcnemar's Test P-Value : NA
```

LDA with Predictor Interaction Learning Phase – Statistics by Class

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7
Sensitivity	0.91473	0.91344	0.86939	0.93053	0.91824	0.95717	0.92955
Specificity	1.00000	0.99436	0.99992	0.99681	0.99488	0.99706	0.99283
Pos Pred Value	1.00000	0.87788	0.99766	0.92883	0.87425	0.93455	0.84369
Neg Pred Value	0.99643	0.99615	0.99482	0.99689	0.99682	0.99812	0.99705
Prevalence	0.04036	0.04247	0.03833	0.04279	0.03731	0.04201	0.03997
Detection Rate	0.03692	0.03880	0.03332	0.03982	0.03426	0.04021	0.03716
Detection Prevalence	0.03692	0.04420	0.03340	0.04287	0.03919	0.04302	0.04404
Balanced Accuracy	0.95736	0.95390	0.93465	0.96367	0.95656	0.97711	0.96119
	Class: 8	Class: 9	Class: 10	Class: 11	Class: 12	Class: 13	Class: 14
Sensitivity	0.87469	0.90602	0.88273	0.87276	0.87981	0.95824	0.89311
Specificity	0.97570	0.99943	1.00000	0.99837	1.00000	0.99992	0.99903
Pos Pred Value	0.53692	0.98569	1.00000	0.95643	1.00000	0.99758	0.96907
Neg Pred Value	0.99588	0.99593	0.99555	0.99481	0.99597	0.99854	0.99637
Prevalence	0.03121	0.04161	0.03669	0.03935	0.03254	0.03371	0.03293
Detection Rate	0.02730	0.03770	0.03238	0.03434	0.02863	0.03231	0.02941
Detection Prevalence	0.05084	0.03825	0.03238	0.03590	0.02863	0.03238	0.03035
Balanced Accuracy	0.92519	0.95272	0.94136	0.93557	0.93990	0.97908	0.94607
	Class: 15	Class: 16	Class: 17	Class: 18	Class: 19	Class: 20	
Sensitivity	0.97714	0.89649	0.88889	0.89222	0.95652	0.88431	
Specificity	0.98834	1.00000	0.99919	0.99544	0.99576	0.99943	
Pos Pred Value	0.78201	1.00000	0.97817	0.88867	0.90681	0.98472	
Neg Pred Value	0.99901	0.99545	0.99546	0.99560	0.99812	0.99521	
Prevalence	0.04107	0.04232	0.03942	0.03919	0.04138	0.03989	
Detection Rate	0.04013	0.03794	0.03504	0.03497	0.03958	0.03528	
Detection Prevalence	0.05131	0.03794	0.03583	0.03935	0.04365	0.03583	
Balanced Accuracy	0.98274	0.94824	0.94404	0.94383	0.97614	0.94187	
	Class: 21	Class: 22	Class: 23	Class: 24	Class: 25	Class: 26	
Sensitivity	0.91209	0.92203	0.93065	0.96045	0.97143	0.93674	
Specificity	0.99878	0.99967	0.99992	0.99021	0.99943	0.99968	
Pos Pred Value	0.96512	0.99161	0.99760	0.80952	0.98646	0.98972	
Neg Pred Value	0.99676	0.99675	0.99749	0.99827	0.99878	0.99790	
Prevalence	0.03559	0.04013	0.03497	0.04154	0.04107	0.03215	
Detection Rate	0.03246	0.03700	0.03254	0.03989	0.03989	0.03012	
Detection Prevalence	0.03364	0.03731	0.03262	0.04928	0.04044	0.03043	
Balanced Accuracy	0.95544	0.96085	0.96528	0.97533	0.98543	0.96821	

LDA with Predictor Interaction Learning Phase – Confusion Matrix

```
> # Generalization Phase Confusion Matrix
> lda_table_test <- table(list(predicted=lda2.test.class, observed=testingDataset_withoutOutliers$Letter))
> lda_cfm_test <- confusionMatrix(lda_table_test)
> print("Generalization Phase Confusion Matrix")
[1] "Generalization Phase Confusion Matrix"
> lda_cfm_test
Confusion Matrix and Statistics

Confusion Matrix and Statistics

          observed
predicted   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
        1 200   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
        2   0 189   0   1   2   6   0   2   0   0   0   2   0   1   0   4   0   6   5   4
        3   0   0 176   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
        4   0   1   0 229   0   0   1   6   0   1   0   0   0   1   1   1   0   1   0   0
        5   0   0   7   0 198   0   2   0   1   0   0   6   0   0   0   0   3   0   6   2
        6   0   0   0   0   1 194   0   0   1   1   0   0   0   0   0  16   0   0   0   1
        7   0   0  16   1  16   1 226   1   1   0   0   6   1   0   0   5   6   1   0   1
        8  13   9   4  10   0   3   5 147   0   6  11   3  12   9   3   4   2  19   2   7
        9   0   0   0   0   0   0   0   0 200   5   0   0   0   0   0   0   0   0   0   0
        10  0   0   0   0   0   0   0   0   1 143   0   0   0   0   0   0   0   0   0   0
        11  1   0   9   0   1   0   1   1   0   0 186   2   0   0   0   0   0   0   0   1
        12  0   0   0   0   0   0   0   0   0   0   0 138   0   0   0   0   0   0   0   0
        13  0   0   0   0   0   0   0   0   0   0   0   0 176   0   0   0   0   0   0   0
        14  1   0   0   0   0   0   0   1   0   1   0   0   2 168   0   0   0   0   0   0
        15  2   1   6   6   0   0   2   7   0   6   0   0   3   2 208   3  22   3   0   1
        16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 193   0   0   0   0
        17  0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   0 196   0   1   0
        18  0   1   0   1   0   0   0   6   0   1   7   0   0   2   0   0   0 210   4   0
        19  2   3   0   0   1   1   0   0   1   5   0   1   0   0   0   1   0   0 158   0
        20  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 194
        21  1   0   3   0   0   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0
        22  0   0   1   0   0   0   2   1   0   0   0   0   0   0   0   0   0   0   0   0
        23  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
        24  3   4   0   0   3   0   1   1  17   4   6  11   0   0   0   0   1   2   0   2
        25  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
        26  0   0   0   0   2   0   0   0   0   0   0   0   0   0   0   0   1   0   0   2

          observed
predicted  21  22  23  24  25  26
        1   0   0   0   0   0   0
        2   0   1   0   1   0   0
        3   0   0   0   0   0   0
        4   0   0   0   0   0   0
        5   0   0   0   2   0   2
        6   0   0   0   0   0   0
        7   1   0   0   0   0   0
        8   9  14   5   1   2   0
        9   0   0   0   0   0   0
        10  0   0   0   0   0   3
        11  0   0   0   3   0   0
        12  0   0   0   0   0   0
        13  0   0   0   0   0   0
        14  0   1   0   0   0   0
        15  7   1   9   1   2   0
        16  0   1   0   0   0   0
        17  0   0   0   0   1   3
        18  0   0   0   1   0   0
        19  0   0   0   0   0   7
        20  0   0   0   0   2   0
        21 166  0   0   0   0   0
        22  0 206   3   0   2   0
        23  0   1 189   0   0   0
        24  0   0   0 230   0   2
        25  0   3   0   0 217   0
        26  0   0   0   1   0 176
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

LDA with Predictor Interaction Learning Phase – Overall Statistics

```
Overall Statistics

               Accuracy : 0.8965
                 95% CI : (0.8882, 0.9045)
    No Information Rate : 0.0453
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8923

 Mcnemar's Test P-Value : NA
```

LDA with Predictor Interaction Learning Phase – Statistics by Class

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7
Sensitivity	0.89686	0.90865	0.79279	0.92339	0.88393	0.94634	0.94167
Specificity	1.00000	0.99336	0.99981	0.99752	0.99410	0.99621	0.98912
Pos Pred Value	1.00000	0.84375	0.99435	0.94628	0.86463	0.90654	0.79859
Neg Pred Value	0.99564	0.99639	0.99133	0.99637	0.99505	0.99791	0.99731
Prevalence	0.04069	0.03796	0.04051	0.04526	0.04088	0.03741	0.04380
Detection Rate	0.03650	0.03449	0.03212	0.04179	0.03613	0.03540	0.04124
Detection Prevalence	0.03650	0.04088	0.03230	0.04416	0.04179	0.03905	0.05164
Balanced Accuracy	0.94843	0.95101	0.89630	0.96045	0.93902	0.97127	0.96539
	Class: 8	Class: 9	Class: 10	Class: 11	Class: 12	Class: 13	Class: 14
Sensitivity	0.84483	0.90090	0.82184	0.88152	0.81176	0.90722	0.91803
Specificity	0.97116	0.99905	0.99925	0.99639	1.00000	1.00000	0.99887
Pos Pred Value	0.49000	0.97561	0.97279	0.90732	1.00000	1.00000	0.96552
Neg Pred Value	0.99479	0.99583	0.99419	0.99526	0.99401	0.99661	0.99717
Prevalence	0.03175	0.04051	0.03175	0.03850	0.03102	0.03540	0.03339
Detection Rate	0.02682	0.03650	0.02609	0.03394	0.02518	0.03212	0.03066
Detection Prevalence	0.05474	0.03741	0.02682	0.03741	0.02518	0.03212	0.03175
Balanced Accuracy	0.90800	0.94997	0.91054	0.93896	0.90588	0.95361	0.95845
	Class: 15	Class: 16	Class: 17	Class: 18	Class: 19	Class: 20	
Sensitivity	0.97196	0.85022	0.84848	0.86777	0.89773	0.89815	
Specificity	0.98405	0.99981	0.99867	0.99561	0.99585	0.99962	
Pos Pred Value	0.71233	0.99485	0.96552	0.90129	0.87778	0.98980	
Neg Pred Value	0.99884	0.99357	0.99337	0.99390	0.99660	0.99584	
Prevalence	0.03905	0.04142	0.04215	0.04416	0.03212	0.03942	
Detection Rate	0.03796	0.03522	0.03577	0.03832	0.02883	0.03540	
Detection Prevalence	0.05328	0.03540	0.03704	0.04252	0.03285	0.03577	
Balanced Accuracy	0.97801	0.92501	0.92358	0.93169	0.94679	0.94888	
	Class: 21	Class: 22	Class: 23	Class: 24	Class: 25	Class: 26	
Sensitivity	0.90710	0.90351	0.91748	0.95833	0.96018	0.91192	
Specificity	0.99887	0.99829	0.99962	0.98912	0.99924	0.99887	
Pos Pred Value	0.96512	0.95814	0.98953	0.80139	0.98190	0.96703	
Neg Pred Value	0.99680	0.99582	0.99679	0.99807	0.99829	0.99679	
Prevalence	0.03339	0.04161	0.03759	0.04380	0.04124	0.03522	
Detection Rate	0.03029	0.03759	0.03449	0.04197	0.03960	0.03212	
Detection Prevalence	0.03139	0.03923	0.03485	0.05237	0.04033	0.03321	
Balanced Accuracy	0.95299	0.95090	0.95855	0.97373	0.97971	0.95539	

LDA with Predictor Interaction – Check for Over Fitting

```
> ## LDA Checking for over fitting
> # Check for over-fitting. Criteria: Accuracy change from train to test > 25%
> lda_model_isOF=abs((lda2.train.rate-lda2.test.rate)/lda2.train.rate)
> if(lda_model_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fitting")
[1] "Model is not over-fitting"
```

LDA with Predictor Interaction Learning Phase – Classification Accuracy

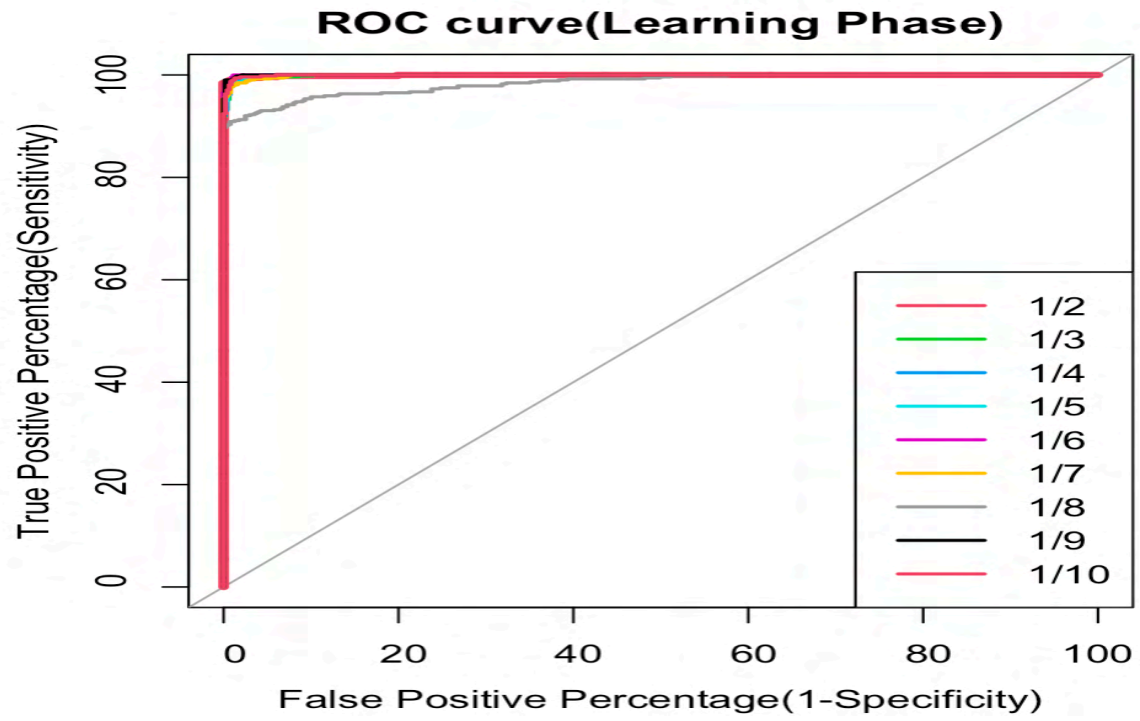
```
> # Accuracy of predictions with train data
> lda2_acc_tr=round(lda_cfm_training$overall[["Accuracy"]],4)
> print(paste("Classification accuracy of learning phase =",round(lda2_acc_tr*100, 4)))
[1] "Classification accuracy of learning phase = 91.74"
```

LDA with Predictor Interaction Generalization Phase – Classification Accuracy

```
> # Accuracy of predictions with test data
> lda2_acc_test=round(lda_cfm_test$overall[["Accuracy"]],4)
> print(paste("Classification accuracy of Generalization phase =",round(lda2_acc_test*100, 4)))
[1] "Classification accuracy of Generalization phase = 89.65"
>
```

LDA with Predictor Interaction Learning Phase – AUC & ROC

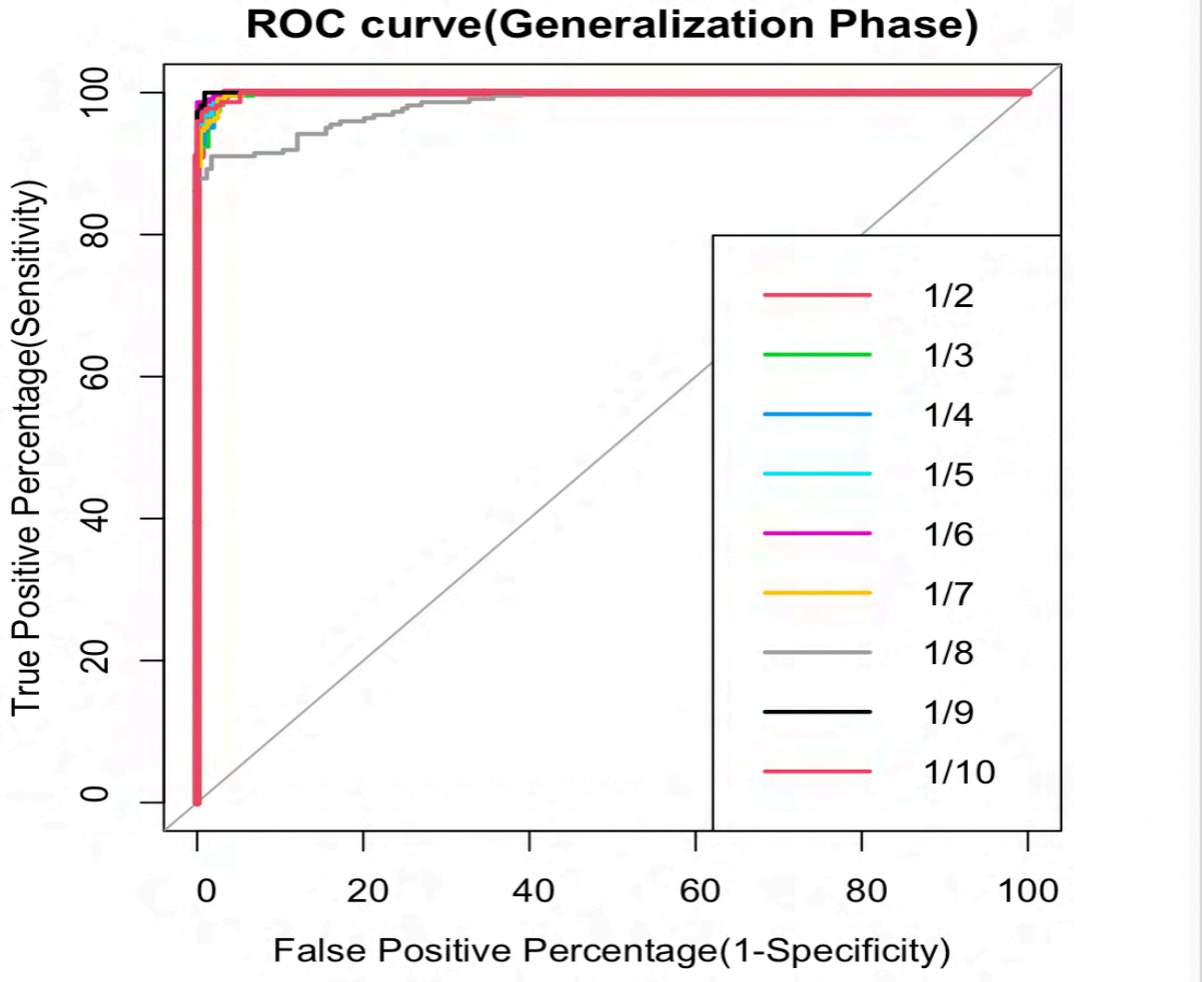
```
> lda2_AUC_train = multiclass.roc(trainingDataset_withoutOutliers[,which(names(trainingDataset_withoutOutliers)=
="Letter")],lda2.train.pred$posterior, percent=TRUE)
> print(paste("LDA Learning-Phase AUC:",round(lda2_AUC_train$auc,4)))
[1] "LDA Learning-Phase AUC: 99.8284"
> lda2_ROC_train <- lda2_AUC_train[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(lda2_ROC_train[[ROC_num]][[2]], col=2, main="ROC curve(Learning Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentage(Sen
sitivity)",lwd=4)
>
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(lda2_ROC_train[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/9','1/10'), col=2:10, lwd=2)
```



LDA with Predictor Interaction Generalization Phase – AUC & ROC

```
> lda2_AUC_test = multiclass.roc(testingDataset_withoutOutliers[,which(names(testingDataset_withoutOutl
iers)=="Letter")],lda2.test.pred$posterior, percent=TRUE)
> print(paste("LDA Generalization-Phase AUC:",round(lda2_AUC_test$auc,4)))
[1] "LDA Generalization-Phase AUC: 99.7248"
```

```
> lda2_ROC_test <- lda2_AUC_test[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(lda2_ROC_test[[ROC_num]][[2]], col=2, main="ROC curve(Generalization Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Perce
ntage(Sensitivity)",lwd=4)
>
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(lda2_ROC_test[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/9','1/10'), col=2:10, lwd
=2)
```



QDA

As stated earlier, performance of QDA should be better than LDA, let's apply QDA and see the behavior with this dataset.

```
> ##############################################################################
> #                                    QDA                                     #
> ##############################################################################
> # Fit the model - Learning Phase
> qda.model.withoutOutliers <- qda(Letter~., data = trainingDataset_withoutOutliers)
> # Make predictions for training data set
> qda.predictions.withoutOutliers <- qda.model.withoutOutliers %>% predict(trainingDataset_wit
houtOutliers)
> # Train Rate
> qda.withoutOutliers.rate <- mean(qda.predictions.withoutOutliers$class == trainingDataset_wi
thoutOutliers$Letter)
> print(paste("QDA Train Rate: ",round(qda.withoutOutliers.rate*100,4)))
[1] "QDA Train Rate:  89.9953"
```

QDA Train Rate for the data set is 89.9953.

QDA Learning Phase – Confusion Matrix

```
> # Confusion Matrix
> qda_table_training <- table(list(predicted=qda.predictions.withoutOutliers$class, observed=t
rainingDataset_withoutOutliers$Letter))
> qda_cfm_training <- confusionMatrix(qda_table_training)
> print("Learning Phase Confusion Matrix")
[1] "Learning Phase Confusion Matrix"
> qda_cfm_training
Confusion Matrix and Statistics
```

	observed																		
predicted	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	489	0	0	0	0	0	0	0	0	1	0	0	2	0	3	0	9	1	1
2	0	497	0	2	4	4	4	9	1	0	3	3	3	1	0	5	1	10	5
3	0	0	447	0	3	0	16	0	0	0	9	1	0	0	0	1	0	0	3
4	1	9	0	510	0	2	3	19	15	3	4	1	0	7	6	1	1	5	0
5	0	5	10	0	425	9	0	0	1	0	1	5	1	1	1	0	1	0	9
6	0	5	0	0	2	482	5	3	9	2	0	0	0	0	0	23	0	0	7
7	0	1	14	0	16	1	449	7	0	0	0	4	1	0	2	5	2	0	0
8	4	8	0	2	4	6	2	276	1	3	12	7	7	9	3	0	0	4	1
9	0	1	0	0	0	4	0	0	463	3	0	0	0	0	0	1	0	0	6
10	0	0	0	0	0	2	0	1	17	437	0	0	0	0	0	0	0	0	2
11	0	0	2	2	2	0	1	19	4	0	432	3	1	2	1	3	0	7	0
12	0	0	6	0	5	0	3	0	1	1	2	371	0	0	1	1	1	0	1
13	2	0	0	5	0	1	0	1	0	0	0	0	408	2	1	0	0	0	0
14	1	0	1	3	0	7	0	10	0	2	0	0	4	383	1	0	0	5	0
15	0	1	9	2	0	0	8	18	0	2	0	0	0	9	478	0	39	1	0
16	0	0	0	1	0	8	0	2	1	0	0	0	0	0	0	493	5	0	0
17	0	0	0	1	8	0	4	1	0	3	0	3	0	0	14	1	438	1	1
18	0	10	0	6	0	0	3	18	0	2	23	1	1	6	0	3	3	460	0
19	0	3	0	3	2	1	8	0	12	6	2	11	0	0	0	0	3	2	475
20	0	0	0	5	0	8	0	0	3	1	0	0	1	0	0	0	0	1	4
21	8	0	1	4	0	1	0	3	0	0	0	0	1	0	3	0	0	3	0
22	1	2	0	0	0	0	4	1	0	0	1	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	1	1	0	0	1	0	1	1	10	0	1	0	0
24	0	0	0	0	1	0	0	5	1	0	13	6	0	0	1	0	0	0	0
25	10	1	0	0	0	1	0	4	0	0	0	0	0	0	0	4	0	0	0
26	0	0	0	1	5	0	0	1	3	3	0	0	0	0	0	0	0	0	14

predicted \ observed	20	21	22	23	24	25	26
1	0	0	0	0	0	1	2
2	0	0	8	1	0	0	0
3	0	0	0	0	1	0	0
4	2	0	1	0	2	0	0
5	7	0	3	0	2	0	2
6	12	0	1	0	1	7	0
7	9	0	3	0	0	0	0
8	5	2	5	4	14	5	2
9	0	0	0	0	6	0	0
10	0	0	0	0	2	0	0
11	4	0	1	0	13	0	0
12	0	0	0	0	4	0	0
13	0	6	1	2	0	1	0
14	0	0	1	1	0	0	0
15	0	2	0	4	0	0	0
16	0	0	0	0	0	7	0
17	3	0	0	0	0	0	3
18	2	0	2	0	3	0	0
19	1	0	0	0	8	3	6
20	450	0	2	0	4	8	2
21	9	436	0	8	3	2	0
22	0	0	475	1	0	41	0
23	0	9	5	426	0	0	0
24	1	0	0	0	464	1	2
25	4	0	5	0	0	449	0
26	1	0	0	0	4	0	392

QDA Learning Phase Accuracy

Classification accuracy of the QDA in Learning Phase is 0.9

```
> # Accuracy of predictions with train data
> qda_acc_tr=round(qda_cfm_training$overall[["Accuracy"]],4)
> print(paste("Classification accuracy of learning phase =",qda_acc_tr))
[1] "Classification accuracy of learning phase = 0.9"
```

QDA Generalization Phase

```
> ##QDA - Generalization Phase
> qda.model.test.withoutOutliers <- qda(Letter~., data = testingDataset_withoutOutliers)
> # Make predictions for training data set
> qda.predictions.test.withoutOutliers <- qda.model.test.withoutOutliers %>% predict(testingDa
taset_withoutOutliers)
> # Train error
> qda.withoutOutliers.test.error <- mean(qda.predictions.test.withoutOutliers$class == testing
Dataset_withoutOutliers$Letter)
> print(paste("QDA Test Rate: ",round(qda.withoutOutliers.test.error*100,4)))
[1] "QDA Test Rate:  91.8796"
>
```

```
> # Confusion Matrix
> qda_table_testing <- table(list(predicted=qda.predictions.test.withoutOutliers$class, observ
ed=testingDataset_withoutOutliers$Letter))
> qda_cfm_testing <- confusionMatrix(qda_table_testing)
> print("QDA Generalization Phase Confusion Matrix")
[1] "QDA Generalization Phase Confusion Matrix"
> qda_cfm_testing
Confusion Matrix and Statistics

          observed
predicted   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
        1 215   0   0   0   0   0   0   0   0   0   0   0   1   0   3   0   1   0   0
        2   0 191   0   0   2   1   1   3   0   0   1   1   3   0   0   0   0   4   3
        3   0   0 207   0   1   0   5   1   1   0   7   0   0   0   2   0   0   1   0
        4   0   1   0 232   0   2   1   4   1   2   0   0   0   2   1   2   0   1   0
        5   0   1   4   0 198   1   1   0   1   0   2   1   0   0   0   0   0   0   6
        6   0   1   0   0   2 188   1   0   4   1   0   0   0   1   0   3   0   0   3
        7   0   0   3   0   7   1 217   3   0   0   0   3   0   0   1   4   0   0   0
        8   0   2   0   3   1   0   0 135   0   1   7   4   4   3   1   1   1   3   0
        9   0   1   0   0   0   1   0   0 200   1   0   0   0   0   0   0   0   0   2
        10  1   0   0   0   0   0   0   0   3 164   0   0   0   0   0   0   0   0   0
        11  0   0   3   0   0   0   1   3   1   0 180   0   1   0   0   0   0   0   0
        12  0   0   1   1   6   0   3   1   1   0   0 158   0   0   0   0   2   0   1
        13  1   0   0   2   0   0   1   1   0   0   0   0 182   3   0   0   0   0   0
        14  0   0   0   1   0   2   0   4   0   0   0   0   1 168   0   0   0   0   0
        15  0   0   4   3   0   0   1   8   0   1   0   0   0   3 195   0  10   0   0
        16  1   0   0   0   0   5   1   0   0   0   0   0   0   0   0 214   0   0   0
        17  0   0   0   0   2   0   2   0   0   0   0   2   0   0   6   2 215   0   0
        18  0   5   0   2   1   0   2   7   2   0  11   0   2   2   2   0   0 231   1
        19  0   2   0   3   1   0   2   0   5   2   0   1   0   0   0   0   1   0 155
        20  0   0   0   1   0   2   0   0   0   0   0   0   0   0   0   1   0   0   2
        21  2   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0
        22  0   3   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   2   0
        23  0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0
        24  0   1   0   0   0   0   0   1   2   0   2   0   0   0   0   0   0   0   0
        25  3   0   0   0   0   2   0   3   0   1   0   0   0   1   0   0   1   0   0
        26  0   0   0   0   3   0   0   0   1   0   0   0   0   0   0   0   0   0   3
          observed
predicted  20  21  22  23  24  25  26
        1   1   0   0   0   0   1   0
        2   0   0   1   0   1   0   0
        3   0   0   0   0   0   0   0
        4   0   0   1   0   1   0   0
        5   2   0   0   0   0   0   0
        6   2   0   0   0   0   3   0
        7   2   0   2   0   0   0   0
        8   3   0   1   0   1   1   0
        9   0   0   0   0   2   0   0
        10  1   0   0   0   1   1   0
        11  3   1   0   1   4   0   0
        12  1   0   0   0   0   0   0
        13  0   1   1   6   0   0   0
        14  0   0   0   1   0   0   0
        15  0   1   0   1   0   0   0
        16  1   0   1   0   0   3   0
        17  0   0   0   0   1   0   2
        18  0   0   3   0   0   0   0
        19  1   0   0   0   1   2   3
        20 197  0   0   0   1   4   3
        21  1 178   1   1   0   1   0
        22  0   0 212   0   0  12   0
        23  0   2   2 195   0   0   0
        24  0   0   0   0 225   0   0
        25  1   0   3   1   1 198   0
        26  0   0   0   0   1   0 185
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

Overall Statistics of QDA Generalization Phase

```
Overall Statistics

               Accuracy : 0.9188
                 95% CI : (0.9112, 0.9259)
    No Information Rate : 0.0453
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9155

 Mcnemar's Test P-Value : NA
```

Generalization Phase Statistics by Class

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7
Sensitivity	0.96413	0.91827	0.93243	0.93548	0.88393	0.91707	0.90417
Specificity	0.99867	0.99602	0.99658	0.99637	0.99639	0.99602	0.99504
Pos Pred Value	0.96847	0.90094	0.92000	0.92430	0.91244	0.89952	0.89300
Neg Pred Value	0.99848	0.99677	0.99715	0.99694	0.99506	0.99677	0.99561
Prevalence	0.04069	0.03796	0.04051	0.04526	0.04088	0.03741	0.04380
Detection Rate	0.03923	0.03485	0.03777	0.04234	0.03613	0.03431	0.03960
Detection Prevalence	0.04051	0.03869	0.04106	0.04580	0.03960	0.03814	0.04434
Balanced Accuracy	0.98140	0.95714	0.96450	0.96593	0.94016	0.95655	0.94960

	Class: 8	Class: 9	Class: 10	Class: 11	Class: 12	Class: 13
Sensitivity	0.77586	0.90090	0.94253	0.85308	0.92941	0.93814
Specificity	0.99303	0.99867	0.99868	0.99658	0.99680	0.99697
Pos Pred Value	0.78488	0.96618	0.95906	0.90909	0.90286	0.91919
Neg Pred Value	0.99265	0.99583	0.99812	0.99413	0.99774	0.99773
Prevalence	0.03175	0.04051	0.03175	0.03850	0.03102	0.03540
Detection Rate	0.02464	0.03650	0.02993	0.03285	0.02883	0.03321
Detection Prevalence	0.03139	0.03777	0.03120	0.03613	0.03193	0.03613
Balanced Accuracy	0.88444	0.94978	0.97060	0.92483	0.96311	0.96756

	Class: 14	Class: 15	Class: 16	Class: 17	Class: 18	Class: 19
Sensitivity	0.91803	0.91121	0.94273	0.93074	0.95455	0.88068
Specificity	0.99830	0.99392	0.99772	0.99676	0.99236	0.99548
Pos Pred Value	0.94915	0.85903	0.94690	0.92672	0.85240	0.86592
Neg Pred Value	0.99717	0.99638	0.99753	0.99695	0.99789	0.99604
Prevalence	0.03339	0.03905	0.04142	0.04215	0.04416	0.03212
Detection Rate	0.03066	0.03558	0.03905	0.03923	0.04215	0.02828
Detection Prevalence	0.03230	0.04142	0.04124	0.04234	0.04945	0.03266
Balanced Accuracy	0.95817	0.95257	0.97022	0.96375	0.97345	0.93808

	Class: 20	Class: 21	Class: 22	Class: 23	Class: 24	Class: 25
Sensitivity	0.91204	0.97268	0.92982	0.94660	0.93750	0.87611
Specificity	0.99734	0.99849	0.99657	0.99867	0.99885	0.99676
Pos Pred Value	0.93365	0.95699	0.92174	0.96535	0.97403	0.92093
Neg Pred Value	0.99639	0.99906	0.99695	0.99792	0.99714	0.99468
Prevalence	0.03942	0.03339	0.04161	0.03759	0.04380	0.04124
Detection Rate	0.03595	0.03248	0.03869	0.03558	0.04106	0.03613
Detection Prevalence	0.03850	0.03394	0.04197	0.03686	0.04215	0.03923
Balanced Accuracy	0.95469	0.98558	0.96320	0.97264	0.96818	0.93644

	Class: 26
Sensitivity	0.95855
Specificity	0.99849
Pos Pred Value	0.95855
Neg Pred Value	0.99849
Prevalence	0.03522
Detection Rate	0.03376
Detection Prevalence	0.03522
Balanced Accuracy	0.97852

Classification Accuracy of the Generalization Phase

```
> # Accuracy of predictions with train data
> qda_acc_tst=round(qda_cfm_testing$overall[["Accuracy"]],4)
> print(paste("Classification accuracy of Generalization phase =",qda_acc_tst))
[1] "Classification accuracy of Generalization phase = 0.9188"
```

Generalization Phase - Check for Over-Fitting

```
> ## QDA Checking for over fitting
> # Check for over-fitting. Criteria: Accuracy change from train to test > 25%
> qda_model_isOF=abs((qda_acc_tr-qda_acc_tst)/qda_acc_tr)
> qda_model_isOF=round(qda_model_isOF,4)
> print(paste("Accuracy drop from training data to test data is",qda_model_isOF*100,"%"))
[1] "Accuracy drop from training data to test data is 2.09 %"
> if(lda_model_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fitting")
[1] "Model is not over-fitting"
```

QDA Model Performance Metrics

```
> ### QDA Model Performance Metrics
> print("QDA Learning-Phase Performance Parameters:")
[1] "QDA Learning-Phase Performance Parameters:"
> qda_PM_tr= qda_cfm_training$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "R
ecall")]
>
> qda_PMavg_tr=round(apply(qda_PM_tr,2,mean),4)
>
> print("Macro Averages:")
[1] "Macro Averages:"
> t(qda_PMavg_tr)
     Balanced Accuracy Precision Sensitivity Specificity Recall
[1,]            0.9477    0.9006      0.8994       0.996 0.8994
```

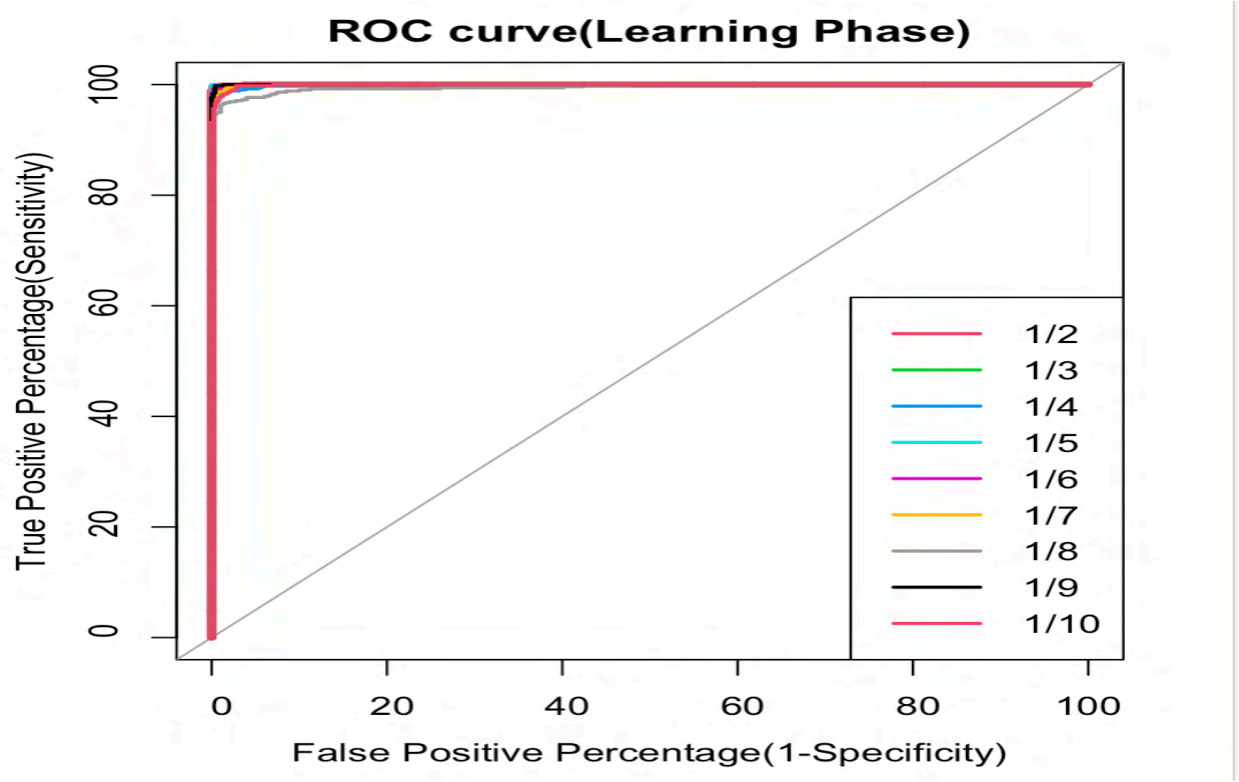
QDA Learning Phase – AUC

The AUC of the model is very close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> qda_prob_tr = predict(qda.model.withoutOutliers, trainingDataset_withoutOutliers[,-which(names(trainingDa
taset_withoutOutliers)=="Letter")], type="prob")
> qda_AUC_tr = multiclass.roc(trainingDataset_withoutOutliers[,which(names(trainingDataset_withoutOutliers)
=="Letter")],qda_prob_tr$posterior, percent=TRUE)
> print(paste("QDA Learning-Phase AUC:",round(qda_AUC_tr$auc,4)))
[1] "QDA Learning-Phase AUC: 99.7348"
```

QDA Learning Phase - ROC

```
> qda_ROC_tr <- qda_AUC_tr[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(qda_ROC_tr[[ROC_num]][[2]], col=2, main="ROC curve(Learning Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentag
e(Sensitivity)",lwd=4)
> qda_ROC_tr <- qda_AUC_tr[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(qda_ROC_tr[[ROC_num]][[2]], col=2, main="ROC curve(Learning Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentag
e(Sensitivity)",lwd=4)
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(qda_ROC_tr[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/9','1/10'), col=2:10, lwd=2)
```



QDA Generalization Phase

```
> ##QDA - Generalization Phase
> print("QDA Generalization-Phase Performance Parameters:")
[1] "QDA Generalization-Phase Performance Parameters:"
> qda_PM_tst= qda_cfm_testing$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "R
ecall")]
> qda_PMavg_tst=round(apply(qda_PM_tst,2,mean),4)
> print("Macro Averages:")
[1] "Macro Averages:"
> t(qda_PMavg_tst)
     Balanced Accuracy Precision Sensitivity Specificity Recall
[1,]            0.9573    0.9189      0.9179      0.9968 0.9179
```

QDA Generalization Phase – AUC

The AUC of the model is very close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> qda_prob_tst = predict(qda.model.test.withoutOutliers, testingDataset_withoutOutliers[,-which(names(testi
ngDataset_withoutOutliers)=="Letter")], type="prob")
> qda_AUC_tst = multiclass.roc(testingDataset_withoutOutliers[,which(names(testingDataset_withoutOutliers)=
="Letter")],qda_prob_tst$posterior, percent=TRUE)
> print(paste("QDA Generalization-Phase AUC:",round(qda_AUC_tst$auc,4)))
[1] "QDA Generalization-Phase AUC: 99.8023"
```

QDA Generalization Phase – ROC

```
> qda_ROC_tst <- qda_AUC_tst[['rocs']]
> ROC_num=paste("1/",as.character(2),sep="")
> par(pty = "s")
> plot.roc(qda_ROC_tst[[ROC_num]][[2]], col=2, main="ROC curve(Generalization Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentag
e(Sensitivity)",lwd=4)
> for(i in 3:10) {
+   ROC_num=paste("1/",as.character(i),sep="")
+   lines.roc(qda_ROC_tst[[ROC_num]][[2]],col=i)
+ }
> legend("bottomright", legend=c('1/2','1/3','1/4','1/5','1/6','1/7','1/8','1/9','1/10'), col=2:10, lwd=2)
```

QDA Generalization Phase – Confirmation Matrix

```
> qda_cfm_testing
Confusion Matrix and Statistics

          observed
predicted   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22
        1 215   0   0   0   0   0   0   0   0   0   0   0   1   0   3   0   1   0   0   1   0   0
        2   0 191   0   0   2   1   1   3   0   0   1   1   3   0   0   0   0   4   3   0   0   1
        3   0   0 207   0   1   0   5   1   1   0   7   0   0   0   2   0   0   1   0   0   0   0
        4   0   1   0 232   0   2   1   4   1   2   0   0   0   2   1   2   0   1   0   0   0   1
        5   0   1   4   0 198   1   1   0   1   0   2   1   0   0   0   0   0   0   6   2   0   0
        6   0   1   0   0   2 188   1   0   4   1   0   0   0   1   0   3   0   0   3   2   0   0
        7   0   0   3   0   7   1 217   3   0   0   0   3   0   0   1   4   0   0   0   2   0   2
        8   0   2   0   3   1   0   0 135   0   1   7   4   4   3   1   1   1   3   0   3   0   1
        9   0   1   0   0   0   1   0   0 200   1   0   0   0   0   0   0   0   0   2   0   0   0
        10  1   0   0   0   0   0   0   0   3 164   0   0   0   0   0   0   0   0   0   1   0   0
        11  0   0   3   0   0   0   1   3   1   0 180   0   1   0   0   0   0   0   0   3   1   0
        12  0   0   1   1   6   0   3   1   1   0   0 158   0   0   0   0   2   0   1   1   0   0
        13  1   0   0   2   0   0   1   1   0   0   0   0 182   3   0   0   0   0   0   0   1   1
        14  0   0   0   1   0   2   0   4   0   0   0   0   1 168   0   0   0   0   0   0   0   0
        15  0   0   4   3   0   0   1   8   0   1   0   0   0   3 195   0  10   0   0   0   1   0
        16  1   0   0   0   0   5   1   0   0   0   0   0   0   0   0 214   0   0   0   1   0   1
        17  0   0   0   0   2   0   2   0   0   0   0   2   0   0   6   2 215   0   0   0   0   0
        18  0   5   0   2   1   0   2   7   2   0  11   0   2   2   2   0   0 231   1   0   0   3
        19  0   2   0   3   1   0   2   0   5   2   0   1   0   0   0   0   1   0 155   1   0   0
        20  0   0   0   1   0   2   0   0   0   0   0   0   0   0   0   1   0   0   2 197   0   0
        21  2   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   1 178   1
        22  0   3   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   2   0   0   0 212
        23  0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   2   2
        24  0   1   0   0   0   0   0   1   2   0   2   0   0   0   0   0   0   0   0   0   0   0
        25  3   0   0   0   0   2   0   3   0   1   0   0   0   1   0   0   1   0   0   1   0   3
        26  0   0   0   0   3   0   0   0   1   0   0   0   0   0   0   0   0   0   3   0   0   0

          observed
predicted  23  24  25  26
        1   0   0   1   0
        2   0   1   0   0
        3   0   0   0   0
        4   0   1   0   0
        5   0   0   0   0
        6   0   0   3   0
        7   0   0   0   0
        8   0   1   1   0
        9   0   2   0   0
        10  0   1   1   0
        11  1   4   0   0
        12  0   0   0   0
        13  6   0   0   0
        14  1   0   0   0
        15  1   0   0   0
        16  0   0   3   0
        17  0   1   0   2
        18  0   0   0   0
        19  0   1   2   3
        20  0   1   4   3
        21  1   0   1   0
        22  0   0  12   0
        23 195  0   0   0
        24  0 225  0   0
        25  1   1 198   0
        26  0   1   0 185
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

QDA Generalization Phase – Overall Statistics

```
Overall Statistics

               Accuracy : 0.9188
                 95% CI : (0.9112, 0.9259)
    No Information Rate : 0.0453
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9155

 Mcnemar's Test P-Value : NA
```

QDA Generalization Phase – Statistics by Class

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity           0.96413  0.91827  0.93243  0.93548  0.88393  0.91707  0.90417  0.77586
Specificity           0.99867  0.99602  0.99658  0.99637  0.99639  0.99602  0.99504  0.99303
Pos Pred Value        0.96847  0.90094  0.92000  0.92430  0.91244  0.89952  0.89300  0.78488
Neg Pred Value        0.99848  0.99677  0.99715  0.99694  0.99506  0.99677  0.99561  0.99265
Prevalence            0.04069  0.03796  0.04051  0.04526  0.04088  0.03741  0.04380  0.03175
Detection Rate        0.03923  0.03485  0.03777  0.04234  0.03613  0.03431  0.03960  0.02464
Detection Prevalence  0.04051  0.03869  0.04106  0.04580  0.03960  0.03814  0.04434  0.03139
Balanced Accuracy     0.98140  0.95714  0.96450  0.96593  0.94016  0.95655  0.94960  0.88444
                     Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
Sensitivity           0.90090   0.94253   0.85308   0.92941   0.93814   0.91803   0.91121   0.94273
Specificity           0.99867   0.99868   0.99658   0.99680   0.99697   0.99830   0.99392   0.99772
Pos Pred Value        0.96618   0.95906   0.90909   0.90286   0.91919   0.94915   0.85903   0.94690
Neg Pred Value        0.99583   0.99812   0.99413   0.99774   0.99773   0.99717   0.99638   0.99753
Prevalence            0.04051   0.03175   0.03850   0.03102   0.03540   0.03339   0.03905   0.04142
Detection Rate        0.03650   0.02993   0.03285   0.02883   0.03321   0.03066   0.03558   0.03905
Detection Prevalence  0.03777   0.03120   0.03613   0.03193   0.03613   0.03230   0.04142   0.04124
Balanced Accuracy     0.94978   0.97060   0.92483   0.96311   0.96756   0.95817   0.95257   0.97022
                     Class: 17 Class: 18 Class: 19 Class: 20 Class: 21 Class: 22 Class: 23 Class: 24
Sensitivity            0.93074   0.95455   0.88068   0.91204   0.97268   0.92982   0.94660   0.93750
Specificity            0.99676   0.99236   0.99548   0.99734   0.99849   0.99657   0.99867   0.99885
Pos Pred Value         0.92672   0.85240   0.86592   0.93365   0.95699   0.92174   0.96535   0.97403
Neg Pred Value         0.99695   0.99789   0.99604   0.99639   0.99906   0.99695   0.99792   0.99714
Prevalence             0.04215   0.04416   0.03212   0.03942   0.03339   0.04161   0.03759   0.04380
Detection Rate         0.03923   0.04215   0.02828   0.03595   0.03248   0.03869   0.03558   0.04106
Detection Prevalence   0.04234   0.04945   0.03266   0.03850   0.03394   0.04197   0.03686   0.04215
Balanced Accuracy      0.96375   0.97345   0.93808   0.95469   0.98558   0.96320   0.97264   0.96818
                     Class: 25 Class: 26
Sensitivity            0.87611   0.95855
Specificity            0.99676   0.99849
Pos Pred Value         0.92093   0.95855
Neg Pred Value         0.99468   0.99849
Prevalence             0.04124   0.03522
Detection Rate         0.03613   0.03376
Detection Prevalence   0.03923   0.03522
Balanced Accuracy      0.93644   0.97852
```

QDA Generalization Phase – Variance Estimation

For variance estimation divided the dataset into two subsets, one includes 90% and the other includes 10% of the total number of samples.

```
> ## Variance Estimation
> # Variance Estimation for Learning Phase
> varEst_tridx=sample(1:nrow(trainingDataset_withoutOutliers), 0.9*nrow(trainingDataset_withoutOutliers), r
eplace=F)
> # Define training data for variance estimation
> varEst_trdf=trainingDataset_withoutOutliers[varEst_tridx,]
> varEst_tstdf=trainingDataset_withoutOutliers[-varEst_tridx,]
```

QDA Variance Estimation using 30%, 60% and 100% of the data.

```
> # Define training data for variance estimation
> varEst_trdf=trainingDataset_withoutOutliers[varEst_tridx,]
> varEst_tstdf=trainingDataset_withoutOutliers[-varEst_tridx,]
> # Variance estimation using 30% of the data
> qda_varEst30=varEst(varEst_trdf, varEst_tstdf, 30, type="qda")
> # Variance estimation using 60% of the data
> qda_varEst60=varEst(varEst_trdf, varEst_tstdf, 60, type="qda")
> # Variance estimation using 100% of the data
> qda_varEst100=varEst(varEst_trdf, varEst_tstdf, 100, type="qda")
```

QDA Variance Estimation using 30% of the data

```
> print("QDA Variance Estimation using 30% of data:")
[1] "QDA Variance Estimation using 30% of data:"
> qda_varEst30
                            [,1]
Mean of Accuracies     0.8625000
Variance of Accuracies 0.0000506
```

QDA Variance Estimation using 60% of the data

```
> print("QDA Variance Estimation using 60% of data:")
[1] "QDA Variance Estimation using 60% of data:"
> qda_varEst60
                            [,1]
Mean of Accuracies     0.8767000
Variance of Accuracies 0.0000174
```

QDA Variance Estimation using 100% of the data

```
> print("QDA Variance Estimation using 100% of data:")
[1] "QDA Variance Estimation using 100% of data:"
> qda_varEst100
                          [,1]
Mean of Accuracies     0.8843
Variance of Accuracies 0.0000
```

Overall, the performance of QDA is better than LDA. The next algorithm that we will work with is SVM.

SVM

SVM usually doesn't perform well when we have large data set because the required training time is higher. So here we run SVM to create the model the time it took to generate the model was high. SVM works relatively well when there is a clear margin of separation between classes. But, SVM is not affected by outliers and its accuracy with the predicting the correct data is better than most of the algorithms. The accuracy of the this dataset with SVM should be higher, let's see the same.

SVM Learning Phase

```
> ###############################################################################
> #                                     SVM                                     #
> ###############################################################################
> # Fit SVM
> svm.fit <- svm(Letter~., trainingDataset_withoutOutliers)
> # Predict using train data (Learning Phase)
> svm_pred_tr=predict(svm.fit, trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutlie
rs)=="Letter")])
```

SVM Learning Train Rate

```
> svm.train.rate <- mean(svm_pred_tr == trainingDataset_withoutOutliers$Letter)
> print(paste("SVM Train Rate: ",round(svm.train.rate*100,4)))
[1] "SVM Train Rate:  95.862"
```

SVM Learning Phase Confusion Matrix

```
> # Confusion Matrix for train data
> svm_cfm_tr=confusionMatrix(table(trainingDataset_withoutOutliers$Letter,svm_pred_tr))
> print("SVM Learning Phase Confusion Matrix")
[1] "SVM Learning Phase Confusion Matrix"
> svm_cfm_tr
Confusion Matrix and Statistics
```

	svm_pred_tr																							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	509	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
2	0	529	0	3	0	0	0	2	0	0	0	0	0	1	0	0	0	5	1	1	0	1	0	0
3	0	0	464	0	6	0	9	1	0	0	1	0	0	0	8	0	0	1	0	0	0	0	0	0
4	0	4	0	526	0	0	0	4	0	0	0	0	0	8	0	0	0	4	0	0	1	0	0	0
5	0	3	2	0	453	0	13	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
6	0	4	0	0	2	523	0	1	1	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0
7	0	0	1	4	0	1	494	1	0	0	0	0	0	0	4	0	0	3	0	0	1	1	1	0
8	0	4	0	16	1	1	5	338	0	0	7	0	0	0	2	0	3	20	0	0	2	0	0	0
9	0	1	1	1	0	1	0	0	500	23	1	0	0	0	0	1	0	0	2	0	0	0	0	1
10	0	0	0	2	0	1	0	2	4	455	0	0	0	1	2	0	0	0	1	0	0	0	0	0
11	0	0	0	2	1	0	0	5	0	0	469	0	0	0	0	0	0	17	0	0	2	0	0	7
12	0	1	1	0	7	0	7	3	0	0	1	383	0	0	0	0	0	4	1	0	0	0	0	8
13	1	7	0	0	0	0	1	1	0	0	0	0	417	2	0	0	0	0	0	0	0	1	1	0
14	0	1	0	3	0	0	0	2	0	0	0	0	3	403	6	0	0	2	0	0	0	1	0	0
15	0	0	0	4	0	0	0	1	0	0	0	0	0	0	508	0	2	2	0	0	2	0	6	0
16	0	4	0	1	3	22	3	1	0	0	1	1	0	0	1	499	2	1	0	0	0	0	0	0
17	1	2	0	1	1	0	2	0	0	0	0	0	0	0	8	0	487	1	0	0	0	0	0	0
18	1	9	0	4	0	0	0	4	0	0	8	0	0	5	0	0	1	469	0	0	0	0	0	0
19	0	3	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	523	0	0	0	0	0
20	0	1	0	2	0	3	2	3	0	0	0	0	0	0	0	0	0	1	0	495	0	0	0	3
21	0	0	0	0	0	0	0	0	0	0	1	0	2	0	3	0	0	0	0	0	445	2	2	0
22	0	12	0	0	0	3	0	0	0	0	0	0	1	3	0	0	0	1	0	0	0	490	1	0
23	0	0	0	0	0	0	0	1	0	0	0	0	2	1	1	0	0	0	0	0	2	0	439	0
24	0	0	0	3	4	1	0	0	2	0	1	0	0	0	0	0	2	1	0	0	0	0	0	517
25	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
26	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	3	1	5	0	0	0	0	0

```
   svm_pred_tr
     25  26
1     4   0
2     0   0
3     0   0
4     0   0
5     0   4
6     0   0
7     0   0
8     0   0
9     0   0
10    0   1
11    0   0
12    0   0
13    0   0
14    0   0
15    0   0
16    2   0
17    1   0
18    0   0
19    0   0
20    0   0
21    0   0
22    2   0
23    1   0
24    0   0
25  520   0
26    0 400
```

SVM Learning Phase – Overall Metrics

```
Overall Statistics

               Accuracy : 0.9586
                 95% CI : (0.955, 0.962)
    No Information Rate : 0.0458
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9569

 Mcnemar's Test P-Value : NA
```

SVM Learning Phase – Statistics by Class

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity           0.99414  0.90427  0.98723  0.91958  0.94375  0.93560  0.92164  0.91105
Specificity           0.99943  0.99885  0.99789  0.99828  0.99805  0.99885  0.99861  0.99509
Pos Pred Value        0.98643  0.97422  0.94694  0.96161  0.94969  0.97393  0.96673  0.84712
Neg Pred Value        0.99976  0.99543  0.99951  0.99624  0.99781  0.99706  0.99658  0.99734
Prevalence            0.04005  0.04576  0.03676  0.04474  0.03755  0.04373  0.04193  0.02902
Detection Rate        0.03982  0.04138  0.03630  0.04115  0.03543  0.04091  0.03864  0.02644
Detection Prevalence  0.04036  0.04247  0.03833  0.04279  0.03731  0.04201  0.03997  0.03121
Balanced Accuracy     0.99679  0.95156  0.99256  0.95893  0.97090  0.96723  0.96013  0.95307
                     Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
Sensitivity           0.98619   0.95188   0.95714   0.99740   0.97658   0.95047   0.93554   0.99800
Specificity           0.99739   0.99886   0.99723   0.99734   0.99887   0.99854   0.99861   0.99658
Pos Pred Value        0.93985   0.97015   0.93241   0.92067   0.96752   0.95724   0.96762   0.92237
Neg Pred Value        0.99943   0.99813   0.99829   0.99992   0.99919   0.99830   0.99714   0.99992
Prevalence            0.03966   0.03739   0.03833   0.03004   0.03340   0.03317   0.04247   0.03911
Detection Rate        0.03911   0.03559   0.03669   0.02996   0.03262   0.03152   0.03974   0.03903
Detection Prevalence  0.04161   0.03669   0.03935   0.03254   0.03371   0.03293   0.04107   0.04232
Balanced Accuracy     0.99179   0.97537   0.97719   0.99737   0.98772   0.97451   0.96708   0.99729
                     Class: 17 Class: 18 Class: 19 Class: 20 Class: 21 Class: 22 Class: 23 Class: 24
Sensitivity           0.97012   0.87828   0.98124   0.98410   0.97588   0.98790   0.97556   0.96276
Specificity           0.99862   0.99739   0.99951   0.99878   0.99919   0.99813   0.99935   0.99886
Pos Pred Value        0.96627   0.93613   0.98866   0.97059   0.97802   0.95517   0.98210   0.97363
Neg Pred Value        0.99878   0.99471   0.99918   0.99935   0.99911   0.99951   0.99911   0.99837
Prevalence            0.03927   0.04177   0.04169   0.03935   0.03567   0.03880   0.03520   0.04201
Detection Rate        0.03809   0.03669   0.04091   0.03872   0.03481   0.03833   0.03434   0.04044
Detection Prevalence  0.03942   0.03919   0.04138   0.03989   0.03559   0.04013   0.03497   0.04154
Balanced Accuracy     0.98437   0.93783   0.99037   0.99144   0.98753   0.99302   0.98745   0.98081
                     Class: 25 Class: 26
Sensitivity           0.98113   0.98765
Specificity           0.99959   0.99911
Pos Pred Value        0.99048   0.97324
Neg Pred Value        0.99918   0.99960
Prevalence            0.04146   0.03168
Detection Rate        0.04068   0.03129
Detection Prevalence  0.04107   0.03215
Balanced Accuracy     0.99036   0.99338
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

SVM Learning Phase Accuracy

```
> svm_acc_tr=round(svm_cfm_tr$overall[["Accuracy"]],4)
> print(paste("SVM Learning Phase Accuracy =",round(svm_acc_tr*100,4)))
[1] "SVM Learning Phase Accuracy = 95.86"
```

SVM Generalization Phase

SVM Generalization Phase Test Rate

```
> ### SVM Generalization Phase
> # Predict using test data (Generalization Phase)
> svm_pred_tst=predict(svm.fit, testingDataset_withoutOutliers[, -which(names(testingDataset_withoutOutlier
s)=="Letter")])
> svm.pred.test.rate <- mean(svm_pred_tst == testingDataset_withoutOutliers$Letter)
>
> print(paste("SVM Test Rate: ",round(svm.pred.test.rate*100,4)))
[1] "SVM Test Rate:  94.0146"
```

SVM Generalization – Confirmation Matrix

```
> # Confusion Matrix for test data
> svm_cfm_testing=confusionMatrix(table(testingDataset_withoutOutliers$Letter,svm_pred_tst))
> print("SVM Learning Phase Confusion Matrix")
[1] "SVM Learning Phase Confusion Matrix"
> svm_cfm_testing
Confusion Matrix and Statistics
```

	svm_pred_tst																							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	219	0	0	0	0	0	0	0	0	0	1	0	2	1	0	0	0	0	0	0	0	0	0	0
2	0	196	0	4	0	0	0	2	0	0	0	0	0	0	0	0	0	1	3	0	0	0	0	2
3	0	0	206	0	5	0	4	1	0	0	0	0	0	0	3	0	0	1	0	0	0	2	0	0
4	0	3	0	231	0	0	0	5	0	0	0	0	0	2	5	0	0	1	0	0	0	0	0	0
5	0	1	0	0	207	0	12	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	203	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	1	0	5	4	0	225	0	0	0	1	0	0	0	1	0	1	1	0	0	0	0	1	0
8	0	4	1	5	0	0	5	145	0	0	3	0	0	0	1	0	1	8	0	0	1	0	0	0
9	0	0	1	2	0	1	0	0	209	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	0	0	1	1	1	0	1	7	158	0	0	0	1	1	0	0	0	2	0	0	0	0	0
11	0	0	2	0	0	0	0	3	0	0	195	0	1	0	0	0	0	6	0	0	0	0	0	4
12	0	0	1	0	6	0	5	2	0	0	0	154	0	0	0	0	0	1	1	0	0	0	0	0
13	2	5	0	0	0	0	1	2	0	0	0	0	177	2	2	0	0	0	0	0	0	1	2	0
14	1	2	0	1	0	0	0	2	0	0	0	0	1	172	3	0	0	1	0	0	0	0	0	0
15	0	1	1	2	0	0	0	0	0	0	0	0	0	0	202	0	4	0	0	0	0	0	4	0
16	0	1	0	2	1	12	0	3	0	0	0	0	0	0	2	204	0	0	0	1	0	0	0	0
17	0	0	0	0	1	0	0	1	0	0	0	0	0	0	2	0	226	0	0	0	0	0	0	0
18	0	8	0	4	0	0	0	3	0	0	1	0	0	3	0	0	0	221	0	0	0	0	1	1
19	1	3	0	0	4	1	0	1	0	0	0	0	0	0	0	0	0	1	161	0	0	0	0	1
20	0	1	0	0	1	2	1	0	0	0	2	0	0	0	0	0	0	0	1	205	1	0	0	1
21	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	179	0	1	0
22	1	7	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	1	0	0	0	214	1	0
23	0	1	0	0	0	0	1	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	201	0
24	0	1	0	1	1	2	0	0	0	0	2	0	0	0	0	0	1	1	0	1	0	0	0	230
25	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
26	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0

	svm_pred_tst	
	25	26
1	0	0
2	0	0
3	0	0
4	0	1
5	0	2
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	1	0
17	0	1
18	0	0
19	0	3
20	1	0
21	0	0
22	1	0
23	0	0
24	0	0
25	224	0
26	0	188

SVM Generalization – Overall Statistics

```
Overall Statistics

               Accuracy : 0.9401
                 95% CI : (0.9335, 0.9463)
    No Information Rate : 0.0471
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9377

 Mcnemar's Test P-Value : NA
```

SVM Generalization – Statistics by Class

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity           0.96903  0.83404  0.97170  0.89535  0.89224  0.91031  0.88583  0.82857
Specificity           0.99924  0.99771  0.99696  0.99674  0.99676  0.99962  0.99713  0.99453
Pos Pred Value        0.98206  0.94231  0.92793  0.93145  0.92411  0.99024  0.93750  0.83333
Neg Pred Value        0.99867  0.99260  0.99886  0.99484  0.99524  0.99621  0.99447  0.99435
Prevalence            0.04124  0.04288  0.03869  0.04708  0.04234  0.04069  0.04635  0.03193
Detection Rate        0.03996  0.03577  0.03759  0.04215  0.03777  0.03704  0.04106  0.02646
Detection Prevalence  0.04069  0.03796  0.04051  0.04526  0.04088  0.03741  0.04380  0.03175
Balanced Accuracy     0.98413  0.91588  0.98433  0.94605  0.94450  0.95497  0.94148  0.91155
                     Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
Sensitivity           0.96759   0.94611   0.94203   1.00000   0.97790   0.93989   0.90179   1.00000
Specificity           0.99753   0.99699   0.99697   0.99700   0.99679   0.99792   0.99772   0.99564
Pos Pred Value        0.94144   0.90805   0.92417   0.90588   0.91237   0.93989   0.94393   0.89868
Neg Pred Value        0.99867   0.99830   0.99772   1.00000   0.99924   0.99792   0.99582   1.00000
Prevalence            0.03942   0.03047   0.03777   0.02810   0.03303   0.03339   0.04088   0.03723
Detection Rate        0.03814   0.02883   0.03558   0.02810   0.03230   0.03139   0.03686   0.03723
Detection Prevalence  0.04051   0.03175   0.03850   0.03102   0.03540   0.03339   0.03905   0.04142
Balanced Accuracy     0.98256   0.97155   0.96950   0.99850   0.98735   0.96891   0.94975   0.99782
                     Class: 17 Class: 18 Class: 19 Class: 20 Class: 21 Class: 22 Class: 23 Class: 24
Sensitivity           0.96170   0.90574   0.94152   0.98558   0.98352   0.98618   0.95261   0.96234
Specificity           0.99905   0.99599   0.99717   0.99791   0.99924   0.99734   0.99905   0.99809
Pos Pred Value        0.97835   0.91322   0.91477   0.94907   0.97814   0.93860   0.97573   0.95833
Neg Pred Value        0.99829   0.99561   0.99811   0.99943   0.99943   0.99943   0.99810   0.99828
Prevalence            0.04288   0.04453   0.03120   0.03796   0.03321   0.03960   0.03850   0.04361
Detection Rate        0.04124   0.04033   0.02938   0.03741   0.03266   0.03905   0.03668   0.04197
Detection Prevalence  0.04215   0.04416   0.03212   0.03942   0.03339   0.04161   0.03759   0.04380
Balanced Accuracy     0.98037   0.95086   0.96935   0.99175   0.99138   0.99176   0.97583   0.98022
                     Class: 25 Class: 26
Sensitivity           0.98678   0.96410
Specificity           0.99962   0.99905
Pos Pred Value        0.99115   0.97409
Neg Pred Value        0.99943   0.99868
Prevalence            0.04142   0.03558
Detection Rate        0.04088   0.03431
Detection Prevalence  0.04124   0.03522
Balanced Accuracy     0.99320   0.98158
```

SVM Generalization Phase Accuracy

```
> # Accuracy
> svm_acc_tst=round(svm_cfm_testing$overall[["Accuracy"]],4)
> print(paste("SVM Generalization Phase Accuracy =",round(svm_acc_tst*100,4)))
[1] "SVM Generalization Phase Accuracy = 94.01"
```

SVM check for Overfitting

```
> # Check for over-fitting. Criteria: Accuracy change from train to test > 25%
> svm_model_isOF=abs((svm_acc_tr-svm_acc_tst)/svm_acc_tr)
> svm_model_isOF=round(svm_model_isOF,4)
> print(paste("Accuracy drop from training data to test data is",svm_model_isOF*100,"%"))
[1] "Accuracy drop from training data to test data is 1.93 %"
> if(svm_model_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fitting")
[1] "Model is not over-fitting"
```

SVM Model Performance Metrics

```
> ##############################################################################
> #                    SVM Model Performance Metrics                           #
> ##############################################################################
> svm_PM_tr = svm_cfm_tr$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recal
l")]
> print("SVM Learning-Phase Performance Parameters:")
[1] "SVM Learning-Phase Performance Parameters:"
> svm_PM_tr
          Balanced Accuracy Precision Sensitivity Specificity    Recall
Class: 1          0.9967851 0.9864341   0.9941406   0.9994296 0.9941406
Class: 2          0.9515629 0.9742173   0.9042735   0.9988524 0.9042735
Class: 3          0.9925613 0.9469388   0.9872340   0.9978886 0.9872340
Class: 4          0.9589304 0.9616088   0.9195804   0.9982804 0.9195804
Class: 5          0.9708997 0.9496855   0.9437500   0.9980494 0.9437500
Class: 6          0.9672270 0.9739292   0.9355993   0.9988548 0.9355993
Class: 7          0.9601269 0.9667319   0.9216418   0.9986120 0.9216418
Class: 8          0.9530685 0.8471178   0.9110512   0.9950858 0.9110512
Class: 9          0.9917934 0.9398496   0.9861933   0.9973935 0.9861933
Class: 10         0.9753726 0.9701493   0.9518828   0.9988623 0.9518828
Class: 11         0.9771886 0.9324056   0.9571429   0.9972344 0.9571429
Class: 12         0.9973673 0.9206731   0.9973958   0.9973387 0.9973958
Class: 13         0.9877239 0.9675174   0.9765808   0.9988670 0.9765808
Class: 14         0.9745077 0.9572447   0.9504717   0.9985437 0.9504717
Class: 15         0.9670773 0.9676190   0.9355433   0.9986112 0.9355433
Class: 16         0.9972905 0.9223660   0.9980000   0.9965809 0.9980000
Class: 17         0.9843677 0.9662698   0.9701195   0.9986159 0.9701195
Class: 18         0.9378325 0.9361277   0.8782772   0.9973878 0.8782772
Class: 19         0.9903743 0.9886578   0.9812383   0.9995102 0.9812383
Class: 20         0.9914370 0.9705882   0.9840954   0.9987786 0.9840954
Class: 21         0.9875330 0.9780220   0.9758772   0.9991888 0.9758772
Class: 22         0.9930157 0.9551657   0.9879032   0.9981283 0.9879032
Class: 23         0.9874535 0.9821029   0.9755556   0.9993514 0.9755556
Class: 24         0.9808065 0.9736347   0.9627561   0.9988569 0.9627561
Class: 25         0.9903620 0.9904762   0.9811321   0.9995920 0.9811321
Class: 26         0.9933829 0.9732360   0.9876543   0.9991114 0.9876543
```

SVM Learning Phase Performance Metrics

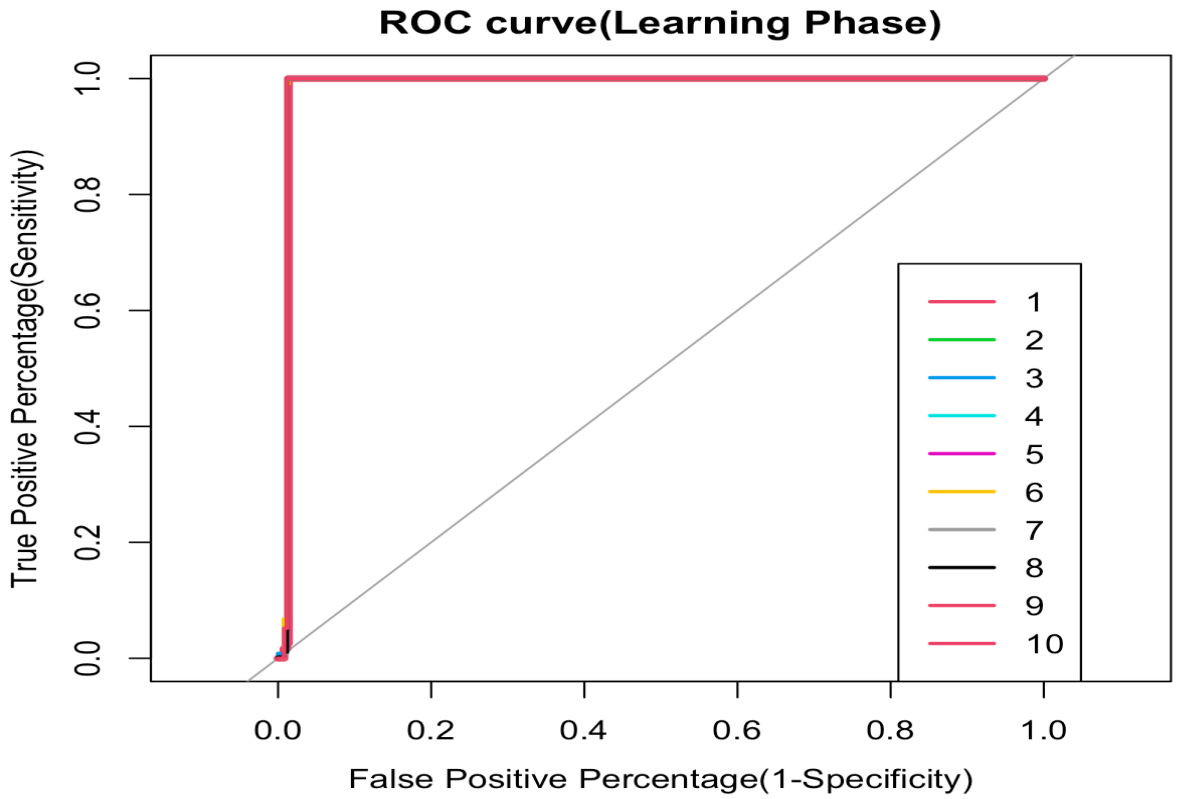
```
> svm_PM_avg_tr=round(apply(svm_PM_tr,2,mean),4)
> print("Macro Averages:")
[1] "Macro Averages:"
> t(svm_PM_avg_tr)
     Balanced Accuracy Precision Sensitivity Specificity Recall
[1,]            0.9791    0.9576      0.9598      0.9983 0.9598
```

SVM Learning Phase – AUC & ROC

The AUC of the model is close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> svm_prob_tr = predict(svm.fit,
+                       trainingDataset_withoutOutliers[,-which(names(trainingDataset_withoutOutliers)=="Le
tter")], )
> svm_AUC_tr= multiclass.roc(trainingDataset_withoutOutliers$Letter, as.numeric(svm_prob_tr))
> print(paste("SVM Learning-Phase AUC:", round(svm_AUC_tr$auc, 4)))
[1] "SVM Learning-Phase AUC: 0.975"
```

```
> svm_AUC_tr <- svm_AUC_tr$rocs
> plot.roc(svm_AUC_tr[[1]], col=2, main="ROC curve(Generalization Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage", ylab="True Positive Percentage",lwd=4)
> for(i in 2:10){
+   lines.roc(svm_AUC_tr[[i]],col=i)
+ }
> legend("bottomright", legend=c('1', '2','3','4','5','6','7','8','9','10'), col=2:10, lwd=2)
```



SVM Generalization – Performance Metrics

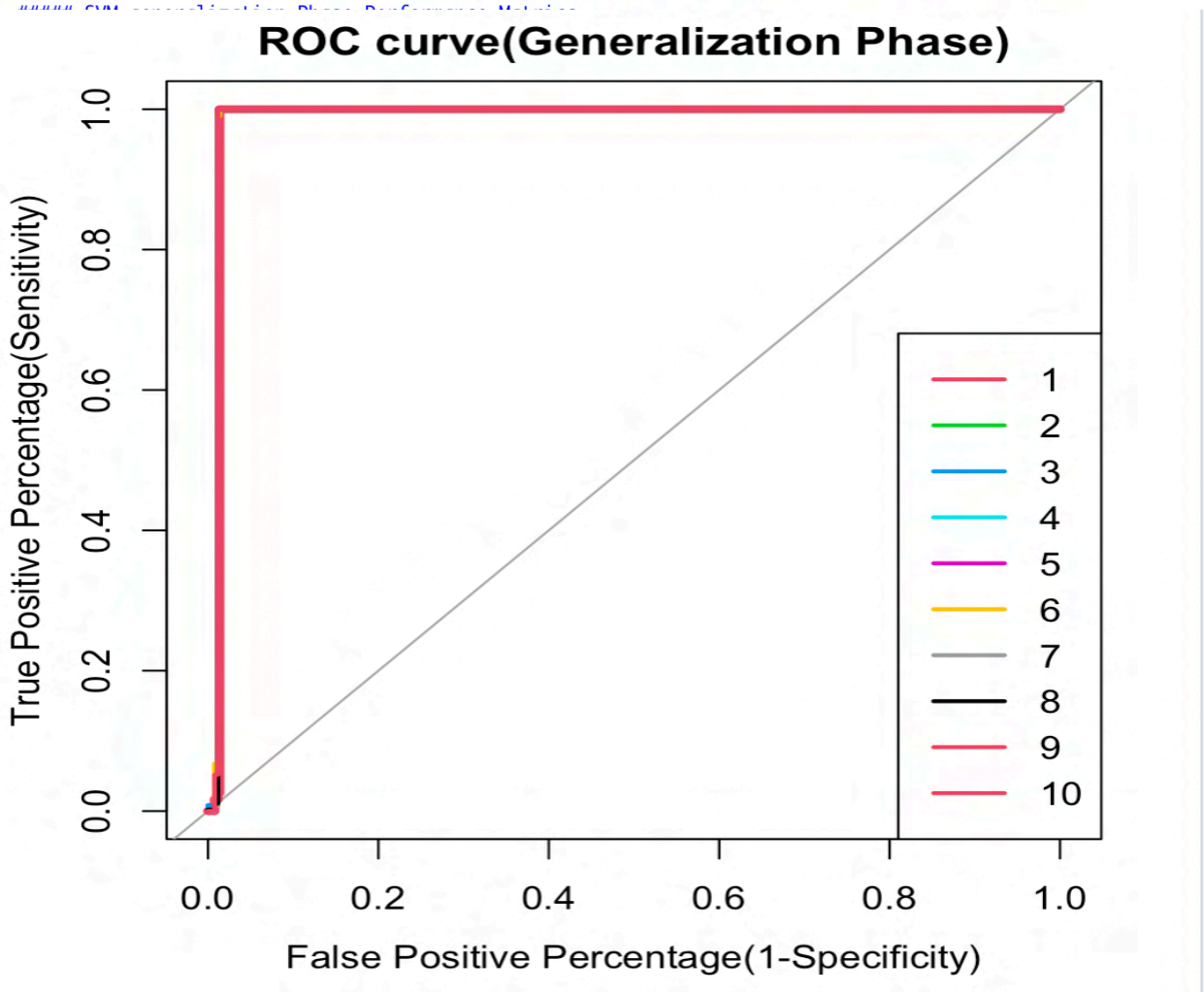
```
> ##### SVM generalization-Phase Performance Matrics
> svm_PM_tst=svm_cfm_testing$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Re
call")]
> print("SVM Generalization-Phase Performance Parameters:")
[1] "SVM Generalization-Phase Performance Parameters:"
> svm_PM_tst
          Balanced Accuracy Precision Sensitivity Specificity    Recall
Class: 1          0.9841326 0.9820628   0.9690265   0.9992387 0.9690265
Class: 2          0.9158773 0.9423077   0.8340426   0.9977121 0.8340426
Class: 3          0.9843305 0.9279279   0.9716981   0.9969628 0.9716981
Class: 4          0.9460467 0.9314516   0.8953488   0.9967445 0.8953488
Class: 5          0.9445010 0.9241071   0.8922414   0.9967607 0.8922414
Class: 6          0.9549667 0.9902439   0.9103139   0.9996196 0.9103139
Class: 7          0.9414783 0.9375000   0.8858268   0.9971297 0.8858268
Class: 8          0.9115524 0.8333333   0.8285714   0.9945335 0.8285714
Class: 9          0.9825615 0.9414414   0.9675926   0.9975304 0.9675926
Class: 10         0.9715482 0.9080460   0.9461078   0.9969885 0.9461078
Class: 11         0.9694973 0.9241706   0.9420290   0.9969657 0.9420290
Class: 12         0.9984979 0.9058824   1.0000000   0.9969959 1.0000000
Class: 13         0.9873462 0.9123711   0.9779006   0.9967918 0.9779006
Class: 14         0.9689070 0.9398907   0.9398907   0.9979234 0.9398907
Class: 15         0.9497513 0.9439252   0.9017857   0.9977169 0.9017857
Class: 16         0.9978203 0.8986784   1.0000000   0.9956406 1.0000000
Class: 17         0.9803744 0.9783550   0.9617021   0.9990467 0.9617021
Class: 18         0.9508635 0.9132231   0.9057377   0.9959893 0.9057377
Class: 19         0.9693475 0.9147727   0.9415205   0.9971746 0.9415205
Class: 20         0.9917452 0.9490741   0.9855769   0.9979135 0.9855769
Class: 21         0.9913807 0.9781421   0.9835165   0.9992450 0.9835165
Class: 22         0.9917575 0.9385965   0.9861751   0.9973399 0.9861751
Class: 23         0.9758288 0.9757282   0.9526066   0.9990511 0.9526066
Class: 24         0.9802175 0.9583333   0.9623431   0.9980920 0.9623431
Class: 25         0.9932017 0.9911504   0.9867841   0.9996193 0.9867841
Class: 26         0.9815782 0.9740933   0.9641026   0.9990539 0.9641026
```

```
> svm_PM_avg_tst=round(apply(svm_PM_tst,2,mean),4)
> svm_PM_avg_tst
Balanced Accuracy         Precision       Sensitivity       Specificity            Recall
           0.9698            0.9390            0.9420            0.9976            0.9420
>
```

SVM Generalization Phase – AUC & ROC

The AUC of the model is close to 100% i.e., 1, which shows that the model is very capable of distinguishing classes.

```
> print(paste("SVM Learning-Phase AUC:", round(svm_AUC_tr$auc, 4)))
[1] "SVM Learning-Phase AUC: 0.975"
>
> svm_AUC_tr <- svm_AUC_tr$rocs
> plot.roc(svm_AUC_tr[[1]], col=2, main="ROC curve(Generalization Phase)",
+          legacy.axes=TRUE, xlab="False Positive Percentage(1-Specificity)", ylab="True Positive Percentag
e(Sensitivity)",lwd=4)
> for(i in 2:10){
+   lines.roc(svm_AUC_tr[[i]],col=i)
+ }
> legend("bottomright", legend=c('1', '2','3','4','5','6','7','8','9','10'), col=2:10, lwd=2)
```



Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

SVM Generalization Phase – Performance Metrics

```
> ##### SVM generalization-Phase Performance Matrics
> svm_PM_tst=svm_cfm_testing$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Re
call")]
> print("SVM Generalization-Phase Performance Parameters:")
[1] "SVM Generalization-Phase Performance Parameters:"
> svm_PM_tst
          Balanced Accuracy Precision Sensitivity Specificity    Recall
Class: 1          0.9841326 0.9820628   0.9690265   0.9992387 0.9690265
Class: 2          0.9158773 0.9423077   0.8340426   0.9977121 0.8340426
Class: 3          0.9843305 0.9279279   0.9716981   0.9969628 0.9716981
Class: 4          0.9460467 0.9314516   0.8953488   0.9967445 0.8953488
Class: 5          0.9445010 0.9241071   0.8922414   0.9967607 0.8922414
Class: 6          0.9549667 0.9902439   0.9103139   0.9996196 0.9103139
Class: 7          0.9414783 0.9375000   0.8858268   0.9971297 0.8858268
Class: 8          0.9115524 0.8333333   0.8285714   0.9945335 0.8285714
Class: 9          0.9825615 0.9414414   0.9675926   0.9975304 0.9675926
Class: 10         0.9715482 0.9080460   0.9461078   0.9969885 0.9461078
Class: 11         0.9694973 0.9241706   0.9420290   0.9969657 0.9420290
Class: 12         0.9984979 0.9058824   1.0000000   0.9969959 1.0000000
Class: 13         0.9873462 0.9123711   0.9779006   0.9967918 0.9779006
Class: 14         0.9689070 0.9398907   0.9398907   0.9979234 0.9398907
Class: 15         0.9497513 0.9439252   0.9017857   0.9977169 0.9017857
Class: 16         0.9978203 0.8986784   1.0000000   0.9956406 1.0000000
Class: 17         0.9803744 0.9783550   0.9617021   0.9990467 0.9617021
Class: 18         0.9508635 0.9132231   0.9057377   0.9959893 0.9057377
Class: 19         0.9693475 0.9147727   0.9415205   0.9971746 0.9415205
Class: 20         0.9917452 0.9490741   0.9855769   0.9979135 0.9855769
Class: 21         0.9913807 0.9781421   0.9835165   0.9992450 0.9835165
Class: 22         0.9917575 0.9385965   0.9861751   0.9973399 0.9861751
Class: 23         0.9758288 0.9757282   0.9526066   0.9990511 0.9526066
Class: 24         0.9802175 0.9583333   0.9623431   0.9980920 0.9623431
> svm_PM_avg_tst=round(apply(svm_PM_tst,2,mean),4)
> svm_PM_avg_tst
Balanced Accuracy         Precision       Sensitivity       Specificity            Recall
           0.9698            0.9390            0.9420            0.9976            0.9420
```

SVM Generalization Phase – Confusion Matrix

```
> svm_cfm_testing
Confusion Matrix and Statistics

          svm_pred_tst
             1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
  1  219   0   0   0   0   0   0   0   0   0   1   0   2   1   0   0   0   0   0   0   0   0   0   0
  2    0 196   0   4   0   0   0   2   0   0   0   0   0   0   0   0   0   1   3   0   0   0   0   2
  3    0   0 206   0   5   0   4   1   0   0   0   0   0   3   0   0   1   0   0   2   0   0   0
  4    0   3   0 231   0   0   0   5   0   0   0   0   0   2   5   0   0   1   0   0   0   0   0   0
  5    0   1   0   0 207   0  12   0   0   0   1   0   0   0   0   0   0   0   1   0   0   0   0   0
  6    0   0   0   0   0 203   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0
  7    0   1   0   5   4   0 225   0   0   0   1   0   0   0   1   0   1   1   0   0   0   0   1   0
  8    0   4   1   5   0   0   5 145   0   0   3   0   0   0   1   0   1   8   0   0   1   0   0   0
  9    0   0   1   2   0   1   0   0 209   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  10   1   0   0   1   1   1   0   1   7 158   0   0   0   1   1   0   0   0   2   0   0   0   0   0
  11   0   0   2   0   0   0   0   3   0   0 195   0   1   0   0   0   0   6   0   0   0   0   0   4
  12   0   0   1   0   6   0   5   2   0   0   0 154   0   0   0   0   0   1   1   0   0   0   0   0
  13   2   5   0   0   0   0   1   2   0   0   0   0 177   2   2   0   0   0   0   0   0   1   2   0
  14   1   2   0   1   0   0   0   2   0   0   0   0   1 172   3   0   0   1   0   0   0   0   0   0
  15   0   1   1   2   0   0   0   0   0   0   0   0   0   0 202   0   4   0   0   0   0   0   4   0
  16   0   1   0   2   1  12   0   3   0   0   0   0   0   0   2 204   0   0   0   1   0   0   0   0
  17   0   0   0   0   1   0   0   1   0   0   0   0   0   0   0   2   0 226   0   0   0   0   0   0
  18   0   8   0   4   0   0   0   3   0   0   1   0   0   3   0   0   0   0 221   0   0   0   0   1   1
  19   1   3   0   0   4   1   0   1   0   0   0   0   0   0   0   0   0   1 161   0   0   0   0   1
  20   0   1   0   0   1   2   1   0   0   0   2   0   0   0   0   0   0   0   1 205   1   0   0   1
  21   0   0   0   0   0   0   0   2   0   0   1   0   0   0   0   0   0   0   0   0 179   0   1   0
  22   1   7   0   0   0   0   0   1   0   0   0   0   0   2   0   0   0   1   0   0   0 214   1   0
  23   0   1   0   0   0   0   1   1   0   0   0   0   0   0   2   0   0   0   0   0   0   0 201   0
  24   0   1   0   1   1   2   0   0   0   0   2   0   0   0   0   0   1   1   0   1   0   0   0 230
  25   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
  26   0   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   2   0   1   0   0   0   0   0
```

	svm_pred_tst	
	25	26
1	0	0
2	0	0
3	0	0
4	0	1
5	0	2
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	1	0
17	0	1
18	0	0
19	0	3
20	1	0
21	0	0
22	1	0
23	0	0
24	0	0
25	224	0
26	0	188

SVM Generalization Phase – Overall Statistics

```
Overall Statistics

               Accuracy : 0.9401
                 95% CI : (0.9335, 0.9463)
    No Information Rate : 0.0471
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9377

 Mcnemar's Test P-Value : NA
```

SVM Generalization Phase – Statistics by Class

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8
Sensitivity	0.96903	0.83404	0.97170	0.89535	0.89224	0.91031	0.88583	0.82857
Specificity	0.99924	0.99771	0.99696	0.99674	0.99676	0.99962	0.99713	0.99453
Pos Pred Value	0.98206	0.94231	0.92793	0.93145	0.92411	0.99024	0.93750	0.83333
Neg Pred Value	0.99867	0.99260	0.99886	0.99484	0.99524	0.99621	0.99447	0.99435
Prevalence	0.04124	0.04288	0.03869	0.04708	0.04234	0.04069	0.04635	0.03193
Detection Rate	0.03996	0.03577	0.03759	0.04215	0.03777	0.03704	0.04106	0.02646
Detection Prevalence	0.04069	0.03796	0.04051	0.04526	0.04088	0.03741	0.04380	0.03175
Balanced Accuracy	0.98413	0.91588	0.98433	0.94605	0.94450	0.95497	0.94148	0.91155

	Class: 9	Class: 10	Class: 11	Class: 12	Class: 13	Class: 14	Class: 15	Class: 16
Sensitivity	0.96759	0.94611	0.94203	1.00000	0.97790	0.93989	0.90179	1.00000
Specificity	0.99753	0.99699	0.99697	0.99700	0.99679	0.99792	0.99772	0.99564
Pos Pred Value	0.94144	0.90805	0.92417	0.90588	0.91237	0.93989	0.94393	0.89868
Neg Pred Value	0.99867	0.99830	0.99772	1.00000	0.99924	0.99792	0.99582	1.00000
Prevalence	0.03942	0.03047	0.03777	0.02810	0.03303	0.03339	0.04088	0.03723
Detection Rate	0.03814	0.02883	0.03558	0.02810	0.03230	0.03139	0.03686	0.03723
Detection Prevalence	0.04051	0.03175	0.03850	0.03102	0.03540	0.03339	0.03905	0.04142
Balanced Accuracy	0.98256	0.97155	0.96950	0.99850	0.98735	0.96891	0.94975	0.99782

	Class: 17	Class: 18	Class: 19	Class: 20	Class: 21	Class: 22	Class: 23	Class: 24
Sensitivity	0.96170	0.90574	0.94152	0.98558	0.98352	0.98618	0.95261	0.96234
Specificity	0.99905	0.99599	0.99717	0.99791	0.99924	0.99734	0.99905	0.99809
Pos Pred Value	0.97835	0.91322	0.91477	0.94907	0.97814	0.93860	0.97573	0.95833
Neg Pred Value	0.99829	0.99561	0.99811	0.99943	0.99943	0.99943	0.99810	0.99828
Prevalence	0.04288	0.04453	0.03120	0.03796	0.03321	0.03960	0.03850	0.04361
Detection Rate	0.04124	0.04033	0.02938	0.03741	0.03266	0.03905	0.03668	0.04197
Detection Prevalence	0.04215	0.04416	0.03212	0.03942	0.03339	0.04161	0.03759	0.04380
Balanced Accuracy	0.98037	0.95086	0.96935	0.99175	0.99138	0.99176	0.97583	0.98022

Class: 25 Class: 26

```
                     Class: 25 Class: 26
Sensitivity            0.98678   0.96410
Specificity            0.99962   0.99905
Pos Pred Value         0.99115   0.97409
Neg Pred Value         0.99943   0.99868
Prevalence             0.04142   0.03558
Detection Rate         0.04088   0.03431
Detection Prevalence   0.04124   0.03522
Balanced Accuracy      0.99320   0.98158
```

SVM Variance Estimation

Variance Estimation with 30%, 60% and 100% of the dataset.

```
> svm_varEst30 = varEst(varEst_trdf, varEst_tstdf, 30, type="svm")
> svm_varEst60 = varEst(varEst_trdf, varEst_tstdf, 60, type="svm")
> svm_varEst100 = varEst(varEst_trdf, varEst_tstdf, 100, type="svm")
```

SVM Variance Estimation using 30% of the data

```
> print("SVM Variance Estimation using 30% of data:")
[1] "SVM Variance Estimation using 30% of data:"
> svm_varEst30
                            [,1]
Mean of Accuracies     8.821e-01
Variance of Accuracies 4.168e-05
```

SVM Variance Estimation using 60% of the data

```
> print("SVM Variance Estimation using 60% of data:")
[1] "SVM Variance Estimation using 60% of data:"
> svm_varEst60
                            [,1]
Mean of Accuracies     9.281e-01
Variance of Accuracies 1.473e-05
```

SVM Variance Estimation using 100% of the data

```
> print("SVM Variance Estimation using 100% of data:")
[1] "SVM Variance Estimation using 100% of data:"
> svm_varEst100
                            [,1]
Mean of Accuracies     9.535e-01
Variance of Accuracies 9.849e-07
```

SVM Without Correlated Feature Variables

SVM Without Correlated Feature Variables Learning Phase

```
> ##############################################################################
> #               SVM Without Correlated Variables                             #
> ##############################################################################
> # Fit SVM
> svm.withoutCorrelation.fit <- svm(Letter~., trainingDataset_withoutCorrelation)
> # Predict using train data (Learning Phase)
> svm_pred_withoutcorrelation_tr=predict(svm.withoutCorrelation.fit, trainingDataset_withoutCorrelation
[, -which(names(trainingDataset_withoutCorrelation)=="Letter")])
> svm.train.rate <- mean(svm_pred_withoutcorrelation_tr == trainingDataset_withoutCorrelation$Letter)
> print(paste("SVM Learning Phase: ",round(svm.train.rate*100, 4)))
[1] "SVM Learning Phase:  93.6405"
```

SVM Without Correlated Feature Variables Generalization Phase

```
> # Fit SVM
> svm.withoutCorrelation.fit.test <- svm(Letter~., testingDataset_withoutCorrelation)
> # Predict using train data (Generalization Phase)
> svm_pred_withoutcorrelation_test=predict(svm.withoutCorrelation.fit.test, testingDataset_withoutCorre
lation[, -which(names(trainingDataset_withoutCorrelation)=="Letter")])
> svm.test.rate <- mean(svm_pred_withoutcorrelation_test == testingDataset_withoutCorrelation$Letter)
> print(paste("SVM Generalization Phase: ",round(svm.test.rate*100, 4)))
[1] "SVM Generalization Phase:  92.8467"
```

SVM Without Correlation – Check for Overfitting

```
> # Confusion Matrix for train data
> svm_cfm2_tr=confusionMatrix(table(trainingDataset_withoutCorrelation$Letter,svm_pred_withoutcorrelati
on_tr))
> svm_acc2_tr=round(svm_cfm2_tr$overall[["Accuracy"]],4)
> print(paste("SVM Learning Phase Accuracy =",round(svm_acc2_tr*100,4)))
[1] "SVM Learning Phase Accuracy = 93.64"
>
> # Confusion Matrix for test data
> svm_cfm2_tst=confusionMatrix(table(testingDataset_withoutCorrelation$Letter,svm_pred_withoutcorrelati
on_test))
> svm_acc2_tst=round(svm_cfm2_tst$overall[["Accuracy"]],4)
> print(paste("SVM Generalization Phase Accuracy =",round(svm_acc2_tst*100,4)))
[1] "SVM Generalization Phase Accuracy = 92.85"
>
> svm_model2_isOF=abs((svm_acc2_tr-svm_acc2_tst)/svm_acc2_tr)
> svm_model2_isOF=round(svm_model2_isOF,4)
> print(paste("Accuracy drop from training data to test data is",svm_model2_isOF*100,"%"))
[1] "Accuracy drop from training data to test data is 0.84 %"
> if(svm_model2_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fitting")
[1] "Model is not over-fitting"
```

Removing Correlated Feature Variables does not improve accuracy any further. Let's look now how Tree algorithm behaves with the data set.

Decision Tree

For a decision tree algorithm calculation can become complex when there are many class labels. Generally, Decision Tree algorithm gives low prediction accuracy for a dataset as compared to other machine learning algorithms. Decision-tree learners can create over-complex trees that do not generalize the data well.

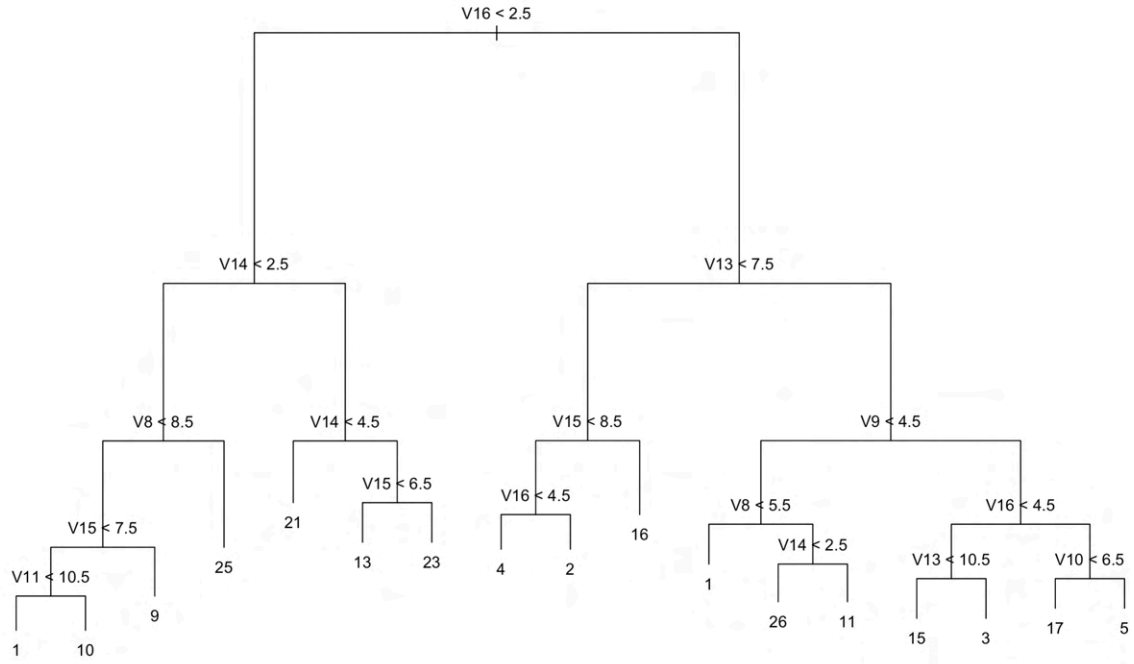
For our dataset we have 26 class labels so the calculation will be complex and the prediction will also be badly affected, let's see the prediction by running the algorithm.

```
> ###############################################################################
> #                                    TREE                                     #
> ###############################################################################
> tree_model = tree(Letter~., data = trainingDataset_withoutOutliers)
> summary(tree_model)

Classification tree:
tree(formula = Letter ~ ., data = trainingDataset_withoutOutliers)
Variables actually used in tree construction:
[1] "V16" "V14" "V8"  "V15" "V11" "V13" "V9"  "V10"
Number of terminal nodes:  17
Residual mean deviance:  3.738 = 47720 / 12770
Misclassification error rate: 0.6039 = 7720 / 12784
> # Make prediction
> tree_pred_tr=predict(tree_model, trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutliers)=="Lett
er")], type="class")
```

Plot of the Decision Tree Learning Model

```
> plot(tree_model)
> text(tree_model,pretty=0)
```



Tree Learning Phase – Train Rate

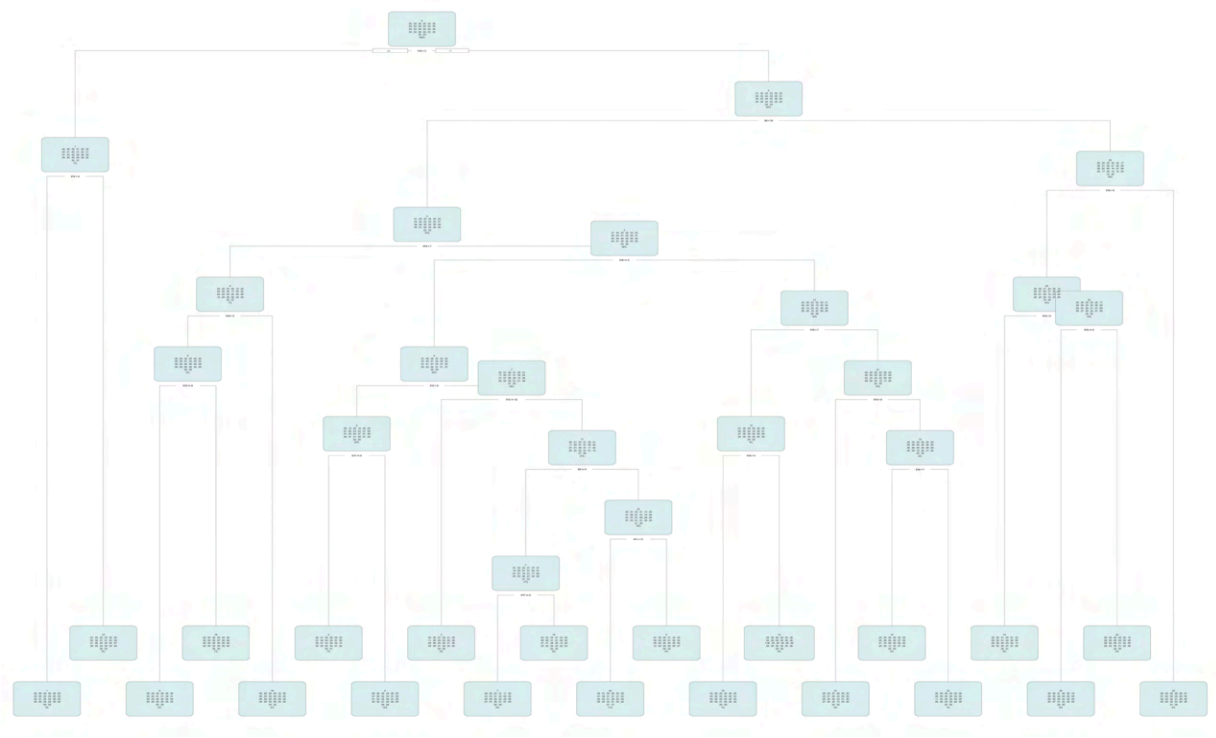
```
> # Train rate
> tree.train.rate <- mean(tree_pred_tr == trainingDataset_withoutOutliers$Letter)
> print(paste("Tree Train Rate: ",round(tree.train.rate*100,4)))
[1] "Tree Train Rate:  39.612"
```

Let's use Rpart library to generate the tree model. Rpart helps in more verbosity so we will have more information about the model in the tree.

```
> # Using rpart (more verbosity)
> tree_model1 = rpart(Letter~., data = trainingDataset_withoutOutliers, method = 'class')
> rpart.plot(tree_model1, box.palette = "azure2")
```

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2



Tree Learning Phase Model

```
> printcp(tree_model1)

Classification tree:
rpart(formula = Letter ~ ., data = trainingDataset_withoutOutliers,
    method = "class")

Variables actually used in tree construction:
 [1] V10 V11 V12 V13 V14 V15 V16 V17 V8  V9

Root node error: 12237/12784 = 0.95721

n= 12784

         CP nsplit rel error  xerror      xstd
1  0.033832      0   1.00000 1.00490 0.0017687
2  0.033219      1   0.96617 0.96045 0.0025159
3  0.031094      3   0.89973 0.90128 0.0031798
4  0.024434      5   0.83754 0.84040 0.0036647
5  0.024107      6   0.81311 0.80273 0.0038979
6  0.023208      7   0.78900 0.78107 0.0040134
7  0.020920      9   0.74258 0.72796 0.0042469
8  0.020593     10   0.72166 0.68579 0.0043879
9  0.020021     13   0.65988 0.64763 0.0044850
10 0.016589     14   0.63986 0.62523 0.0045294
11 0.012094     15   0.62327 0.60448 0.0045624
12 0.011359     16   0.61118 0.59770 0.0045715
13 0.010869     17   0.59982 0.59157 0.0045791
14 0.010460     19   0.57808 0.57972 0.0045919
15 0.010000     20   0.56762 0.56672 0.0046032
```

Tree Learning Phase – Prediction

```
> tree_pred1_tr=predict(tree_model1, trainingDataset_withoutOutliers[, -which(names(trainingDat
aset_withoutOutliers)=="Letter")],type="class")
>
```

```
> tree_cfm_tr=confusionMatrix(table(trainingDataset_withoutOutliers$Letter,tree_pred1_tr))
> print("Tree Learning Phase Confusion Matrix")
[1] "Tree Learning Phase Confusion Matrix"
> tree_cfm_tr
Confusion Matrix and Statistics
```

	tree_pred1_tr																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	402	0	16	8	3	0	2	0	0	4	0	16	0	0	22	0	0	0	0	0	0
2	0	383	0	98	2	0	19	0	0	0	0	0	0	0	24	1	0	0	0	0	2
3	0	2	264	4	0	0	33	0	28	0	0	0	0	0	35	7	0	0	0	4	50
4	0	75	0	393	5	0	5	0	0	0	0	13	0	1	14	0	0	0	0	0	34
5	0	1	4	0	163	0	109	0	0	0	0	0	0	0	35	5	0	0	88	2	0
6	0	23	0	89	0	0	22	0	0	0	0	0	0	0	2	154	0	0	0	129	23
7	1	11	44	23	1	0	314	0	0	0	0	2	0	0	94	2	0	0	0	0	0
8	0	39	0	79	2	0	26	0	0	0	0	11	0	0	69	13	0	0	0	0	51
9	0	40	0	33	2	0	12	0	412	8	0	3	0	0	6	7	0	0	0	0	1
10	0	39	0	48	3	0	1	0	5	300	0	6	0	1	11	15	0	0	0	0	26
11	0	9	35	10	10	0	46	0	0	0	0	14	0	25	3	0	0	0	0	0	72
12	0	3	25	0	24	0	22	0	3	20	0	272	0	0	1	0	0	0	0	0	2
13	0	2	0	20	1	0	3	0	0	0	0	7	329	5	17	0	0	0	0	0	0
14	3	8	0	57	0	0	2	0	0	0	0	26	0	250	9	16	0	0	0	0	1
15	0	26	0	81	0	0	17	0	0	0	0	0	0	0	339	2	0	0	0	0	46
16	0	42	0	134	0	0	20	0	1	0	0	0	0	0	0	297	0	0	0	15	24
17	0	18	3	20	25	0	192	0	0	0	0	0	0	0	176	15	0	0	0	0	4
18	2	139	0	75	0	0	84	0	0	0	0	33	0	0	5	86	0	0	0	0	1
19	0	32	0	130	56	0	14	0	0	0	0	38	0	0	62	3	0	0	122	0	4
20	0	2	0	28	2	0	8	0	2	0	0	0	0	0	25	32	0	0	0	342	0
21	0	0	3	21	0	0	4	0	0	1	0	0	1	43	75	0	0	0	0	6	272
22	0	4	0	7	0	0	7	0	0	0	0	0	0	10	18	4	0	0	0	16	116
23	0	6	0	8	0	0	11	0	0	0	0	0	0	24	23	9	0	0	0	0	1
24	0	23	0	83	35	0	0	0	0	0	0	21	0	0	34	0	0	0	0	3	2
25	0	0	0	93	0	0	0	0	2	0	0	0	0	0	28	12	0	0	0	202	29
26	0	7	0	98	137	0	2	0	0	0	0	9	0	0	71	8	0	0	67	1	0

	tree_pred1_tr				
	22	23	24	25	26
1	0	0	43	0	0
2	0	0	14	0	0
3	0	0	63	0	0
4	0	1	6	0	0
5	0	0	70	0	0
6	91	2	2	0	0
7	0	3	16	0	0
8	0	0	109	0	0
9	0	0	8	0	0
10	0	0	14	0	0
11	0	0	279	0	0
12	0	0	44	0	0
13	0	24	23	0	0
14	11	17	21	0	0
15	0	4	10	0	0
16	4	3	1	0	0
17	11	0	40	0	0
18	0	0	76	0	0
19	0	0	68	0	0
20	19	1	49	0	0
21	0	8	21	0	0
22	305	7	19	0	0
23	14	349	2	0	0
24	0	0	330	0	0
25	144	2	13	0	0
26	0	0	11	0	0

```
Overall Statistics

               Accuracy : 0.4567
                 95% CI : (0.448, 0.4653)
    No Information Rate : 0.1283
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4342

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7
Sensitivity           0.98529  0.41006  0.67005  0.23963  0.34607       NA  0.32205
Specificity           0.99079  0.98650  0.98176  0.98618  0.97450  0.95799  0.98332
Pos Pred Value        0.77907  0.70534  0.53878  0.71846  0.34172       NA  0.61448
Neg Pred Value        0.99951  0.95499  0.98943  0.89810  0.97497       NA  0.94614
Prevalence            0.03191  0.07306  0.03082  0.12829  0.03684  0.00000  0.07627
Detection Rate        0.03145  0.02996  0.02065  0.03074  0.01275  0.00000  0.02456
Detection Prevalence  0.04036  0.04247  0.03833  0.04279  0.03731  0.04201  0.03997
Balanced Accuracy     0.98804  0.69828  0.82591  0.61291  0.66029       NA  0.65268
                     Class: 8 Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14
Sensitivity                NA  0.90949   0.90090        NA   0.57749   0.99697   0.69638
Specificity           0.96879  0.99027   0.98643   0.96065   0.98831   0.99181   0.98624
Pos Pred Value             NA  0.77444   0.63966        NA   0.65385   0.76334   0.59382
Neg Pred Value             NA  0.99665   0.99732        NA   0.98391   0.99992   0.99118
Prevalence            0.00000  0.03543   0.02605   0.00000   0.03684   0.02581   0.02808
Detection Rate        0.00000  0.03223   0.02347   0.00000   0.02128   0.02574   0.01956
Detection Prevalence  0.03121  0.04161   0.03669   0.03935   0.03254   0.03371   0.03293
Balanced Accuracy          NA  0.94988   0.94366        NA   0.78290   0.99439   0.84131
                     Class: 15 Class: 16 Class: 17 Class: 18 Class: 19 Class: 20
Sensitivity            0.28297   0.43169        NA        NA  0.440433   0.47500
Specificity            0.98395   0.97983   0.96058   0.96081  0.967458   0.98607
Pos Pred Value         0.64571   0.54898        NA        NA  0.230624   0.67059
Neg Pred Value         0.92993   0.96806        NA        NA  0.987352   0.96920
Prevalence             0.09371   0.05382   0.00000   0.00000  0.021668   0.05632
Detection Rate         0.02652   0.02323   0.00000   0.00000  0.009543   0.02675
Detection Prevalence   0.04107   0.04232   0.03942   0.03919  0.041380   0.03989
Balanced Accuracy      0.63346   0.70576        NA        NA  0.703946   0.73054
                     Class: 21 Class: 22 Class: 23 Class: 24 Class: 25 Class: 26
Sensitivity            0.35742   0.50918   0.82898   0.24408        NA        NA
Specificity            0.98478   0.98293   0.99207   0.98242   0.95893   0.96785
Pos Pred Value         0.59780   0.59454   0.78076   0.62147        NA        NA
Neg Pred Value         0.96034   0.97604   0.99416   0.91659        NA        NA
Prevalence             0.05953   0.04686   0.03293   0.10576   0.00000   0.00000
Detection Rate         0.02128   0.02386   0.02730   0.02581   0.00000   0.00000
Detection Prevalence   0.03559   0.04013   0.03497   0.04154   0.04107   0.03215
```

```
> tree_acc_tr=round(tree_cfm_tr$overall[["Accuracy"]],4) # Accuracy
> print(paste("Tree Learning Phase Accuracy =",round(tree_acc_tr*100, 4)))
[1] "Tree Learning Phase Accuracy = 45.67"
```

Tree Generalization Phase

```
> ################################################################################
> #                          Tree Generalization Phase                           #
> ################################################################################
>
> tree_pred1_tst= predict(tree_model1, testingDataset_withoutOutliers[, -which(names(testingDat
aset_withoutOutliers)=="Letter")], type = 'class')
>
> tree_cfm1_tst=confusionMatrix(table(testingDataset_withoutOutliers[, which(names(testingDatas
et_withoutOutliers)=="Letter")],tree_pred1_tst)) # Confusion Matrix
>
> print("Tree Generalization-Phase Confusion Matrix")
[1] "Tree Generalization-Phase Confusion Matrix"
> tree_cfm1_tst
> tree_cfm1_tst
Confusion Matrix and Statistics
```

	tree_pred1_tst																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	181	0	0	7	1	0	0	0	0	2	0	5	0	0	11	0	0	0	0	0	0
2	0	154	0	31	0	0	10	0	0	0	0	0	0	0	4	0	0	0	0	0	1
3	0	0	116	1	1	0	17	0	12	0	0	0	0	0	22	4	0	0	0	3	14
4	0	33	0	173	0	0	1	0	0	0	0	3	0	0	9	0	0	0	0	0	27
5	0	0	3	0	76	0	49	0	0	0	0	0	0	0	18	2	0	0	45	1	0
6	0	13	0	24	0	0	9	0	0	0	0	0	0	0	3	56	0	0	0	55	9
7	0	5	25	11	0	0	145	0	0	0	0	1	0	0	38	1	0	0	0	0	0
8	0	11	0	32	0	0	14	0	0	0	0	2	0	0	36	5	0	0	0	1	24
9	0	13	0	11	1	0	2	0	180	3	0	3	0	0	0	2	0	0	0	0	1
10	0	8	0	18	2	0	1	0	5	112	0	2	0	0	4	4	0	0	0	0	15
11	0	6	26	1	3	0	17	0	0	0	0	1	0	10	1	0	0	0	0	0	26
12	0	2	10	0	11	0	12	0	1	6	0	101	0	0	0	0	0	0	0	0	3
13	0	0	0	12	3	0	6	0	0	0	0	2	139	3	14	0	0	0	0	0	0
14	0	1	0	20	0	0	0	0	0	0	0	11	1	123	5	3	0	0	0	0	0
15	0	8	0	34	0	0	11	0	0	0	0	0	0	0	134	1	0	0	0	0	21
16	0	18	0	52	0	0	10	0	0	0	0	0	0	0	2	132	0	0	0	7	6
17	0	8	3	3	11	0	103	0	0	0	0	0	0	0	73	4	0	0	0	1	3
18	3	68	0	34	0	0	36	0	0	0	0	15	0	0	2	42	0	0	0	0	1
19	0	8	0	63	9	0	4	0	0	0	0	10	0	0	20	4	0	0	36	0	1
20	0	1	0	10	2	0	4	0	0	0	0	0	0	0	8	16	0	0	0	154	0
21	0	0	1	5	0	0	3	0	0	0	0	0	3	12	27	0	0	0	0	3	122
22	0	1	0	4	0	0	4	0	0	0	0	0	0	3	8	4	0	0	0	10	67
23	0	2	0	1	0	0	6	0	0	0	0	0	0	12	12	0	0	0	0	0	1
24	0	15	0	44	14	0	0	0	0	0	0	7	0	0	13	0	0	0	0	0	5
25	0	0	0	38	0	0	0	0	0	0	0	0	0	0	12	7	0	0	0	85	10
26	0	3	0	47	60	0	0	0	0	0	0	3	0	0	44	8	0	0	23	1	0

	tree_pred1_tst				
	22	23	24	25	26
1	0	0	16	0	0
2	0	0	8	0	0
3	0	0	32	0	0
4	0	0	2	0	0
5	0	0	30	0	0
6	36	0	0	0	0
7	0	4	10	0	0
8	0	0	49	0	0
9	0	0	6	0	0
10	0	0	3	0	0
11	0	1	119	0	0
12	0	0	24	0	0
13	0	7	8	0	0
14	4	11	4	0	0
15	0	0	5	0	0
16	0	0	0	0	0
17	3	0	19	0	0
18	0	1	40	0	0
19	0	0	21	0	0
20	8	0	13	0	0
21	0	1	6	0	0
22	114	4	9	0	0
23	10	162	0	0	0
24	0	0	142	0	0
25	62	3	9	0	0
26	0	0	4	0	0

Mohit Shankar Pursnani **Letter Recognition Data Set** mp5578@nyu.edu

Assignment - 2

Tree Generalization Phase – Overall Statistics

```
Overall Statistics

               Accuracy : 0.4555
                 95% CI : (0.4422, 0.4688)
    No Information Rate : 0.1234
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4328

 Mcnemar's Test P-Value : NA
```

Tree Generalization Phase – Statistics by Class

Statistics by Class:

	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7
Sensitivity	0.98370	0.40741	0.63043	0.25592	0.39175	NA	0.31250
Specificity	0.99207	0.98942	0.97998	0.98439	0.97200	0.96259	0.98106
Pos Pred Value	0.81166	0.74038	0.52252	0.69758	0.33929	NA	0.60417
Neg Pred Value	0.99943	0.95751	0.98707	0.90386	0.97755	NA	0.93912
Prevalence	0.03358	0.06898	0.03358	0.12336	0.03540	0.00000	0.08467
Detection Rate	0.03303	0.02810	0.02117	0.03157	0.01387	0.00000	0.02646
Detection Prevalence	0.04069	0.03796	0.04051	0.04526	0.04088	0.03741	0.04380
Balanced Accuracy	0.98788	0.69841	0.80521	0.62015	0.68188	NA	0.64678

	Class: 8	Class: 9	Class: 10	Class: 11	Class: 12	Class: 13	Class: 14
Sensitivity	NA	0.90909	0.91057	NA	0.60843	0.97203	0.75460
Specificity	0.96825	0.99205	0.98843	0.9615	0.98702	0.98969	0.98872
Pos Pred Value	NA	0.81081	0.64368	NA	0.59412	0.71649	0.67213
Neg Pred Value	NA	0.99658	0.99793	NA	0.98776	0.99924	0.99245
Prevalence	0.00000	0.03613	0.02245	0.0000	0.03029	0.02609	0.02974
Detection Rate	0.00000	0.03285	0.02044	0.0000	0.01843	0.02536	0.02245
Detection Prevalence	0.03175	0.04051	0.03175	0.0385	0.03102	0.03540	0.03339
Balanced Accuracy	NA	0.95057	0.94950	NA	0.79772	0.98086	0.87166

	Class: 15	Class: 16	Class: 17	Class: 18	Class: 19	Class: 20
Sensitivity	0.25769	0.44746	NA	NA	0.346154	0.47975
Specificity	0.98387	0.98168	0.95785	0.95584	0.973958	0.98798
Pos Pred Value	0.62617	0.58150	NA	NA	0.204545	0.71296
Neg Pred Value	0.92670	0.96897	NA	NA	0.987179	0.96828
Prevalence	0.09489	0.05383	0.00000	0.00000	0.018978	0.05858
Detection Rate	0.02445	0.02409	0.00000	0.00000	0.006569	0.02810
Detection Prevalence	0.03905	0.04142	0.04215	0.04416	0.032117	0.03942
Balanced Accuracy	0.62078	0.71457	NA	NA	0.660056	0.73387

	Class: 21	Class: 22	Class: 23	Class: 24	Class: 25	Class: 26
Sensitivity	0.34174	0.48101	0.83505	0.24525	NA	NA
Specificity	0.98809	0.97826	0.99168	0.98000	0.95876	0.96478
Pos Pred Value	0.66667	0.50000	0.78641	0.59167	NA	NA
Neg Pred Value	0.95564	0.97658	0.99393	0.91660	NA	NA
Prevalence	0.06515	0.04325	0.03540	0.10566	0.00000	0.00000
Detection Rate	0.02226	0.02080	0.02956	0.02591	0.00000	0.00000
Detection Prevalence	0.03339	0.04161	0.03759	0.04380	0.04124	0.03522
Balanced Accuracy	0.66491	0.72963	0.91336	0.61263	NA	NA

Tree Generalization Phase - Accuracy

```
> tree_acc1_tst=round(tree_cfm1_tst$overall[["Accuracy"]],4) # Accuracy
> print(paste("Tree Generalization-Phase Accuracy =",round(tree_acc1_tst*100, 4)))
[1] "Tree Generalization-Phase Accuracy = 45.55"
>
```

Check for overfitting

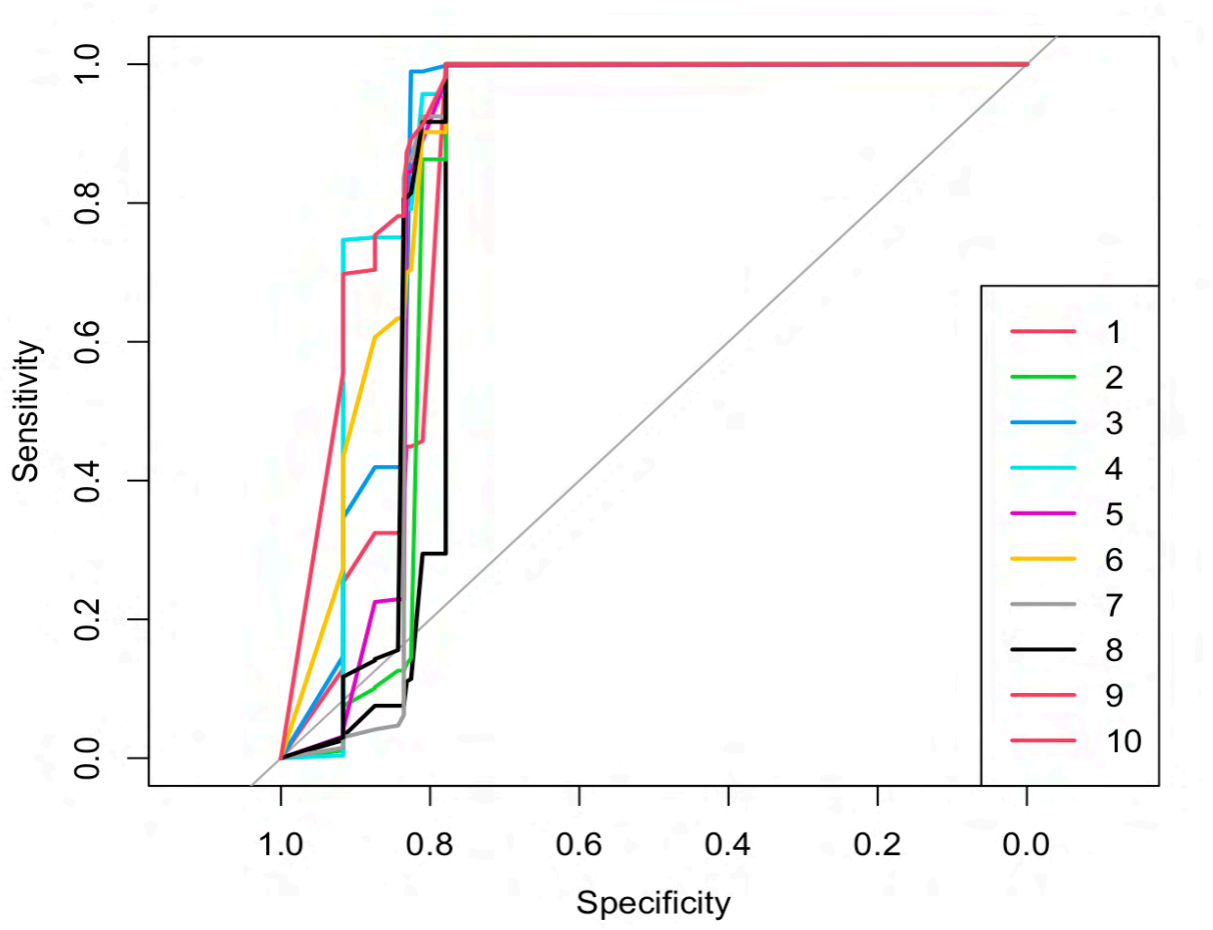
```
> ## Checking for over-fitting
> # Check for over-fitting. Criteria: Accuracy change from train to test > 25%
> tree_model1_isOF=abs((tree_acc_tr-tree_acc1_tst)/tree_acc_tr)
> tree_model1_isOF=round(tree_model1_isOF,4)
> print(paste("Accuracy drop from training data to test data is",tree_model1_isOF*100,"%"))
[1] "Accuracy drop from training data to test data is 0.26 %"
>
> if(tree_model1_isOF>0.25) print("Model is over-fitting") else print("Model is not over-fittin
g")
[1] "Model is not over-fitting"
```

Tree Generalization Phase – Performance Metrics

```
> ################################################################################
> #                    Tree Model Performance Metrics                            #
> ################################################################################
> #Tree Learning Phase
> tree_PM_tr = tree_cfm_tr$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Specifi
city", "Recall")]
> print("Tree Learning-Phase Performance Parameters:")
[1] "Tree Learning-Phase Performance Parameters:"
> tree_PM_tr
         Balanced Accuracy Precision Sensitivity Specificity    Recall
Class: 1         0.9880414 0.7790698   0.9852941   0.9907886 0.9852941
Class: 2         0.6982811 0.7053407   0.4100642   0.9864979 0.4100642
Class: 3         0.8259051 0.5387755   0.6700508   0.9817595 0.6700508
Class: 4         0.6129075 0.7184644   0.2396341   0.9861809 0.2396341
Class: 5         0.6602853 0.3417191   0.3460722   0.9744985 0.3460722
Class: 6                NA 0.0000000          NA   0.9579944        NA
Class: 7         0.6526845 0.6144814   0.3220513   0.9833178 0.3220513
Class: 8                NA 0.0000000          NA   0.9687891        NA
Class: 9         0.9498804 0.7744361   0.9094923   0.9902684 0.9094923
Class: 10        0.9436638 0.6396588   0.9009009   0.9864268 0.9009009
Class: 11               NA 0.0000000          NA   0.9606539        NA
Class: 12        0.7828999 0.6538462   0.5774947   0.9883050 0.5774947
Class: 13        0.9943898 0.7633411   0.9969697   0.9918099 0.9969697
Class: 14        0.8413081 0.5938242   0.6963788   0.9862374 0.6963788
Class: 15        0.6334589 0.6457143   0.2829716   0.9839461 0.2829716
Class: 16        0.7057570 0.5489834   0.4316860   0.9798280 0.4316860
Class: 17               NA 0.0000000          NA   0.9605757        NA
Class: 18               NA 0.0000000          NA   0.9608104        NA
Class: 19        0.7039457 0.2306238   0.4404332   0.9674582 0.4404332
Class: 20        0.7305371 0.6705882   0.4750000   0.9860743 0.4750000
Class: 21        0.6711018 0.5978022   0.3574244   0.9847792 0.3574244
Class: 22        0.7460559 0.5945419   0.5091820   0.9829298 0.5091820
Class: 23        0.9105259 0.7807606   0.8289786   0.9920731 0.8289786
Class: 24        0.6132503 0.6214689   0.2440828   0.9824178 0.2440828
Class: 25               NA 0.0000000          NA   0.9589330        NA
Class: 26               NA 0.0000000          NA   0.9678504        NA
```

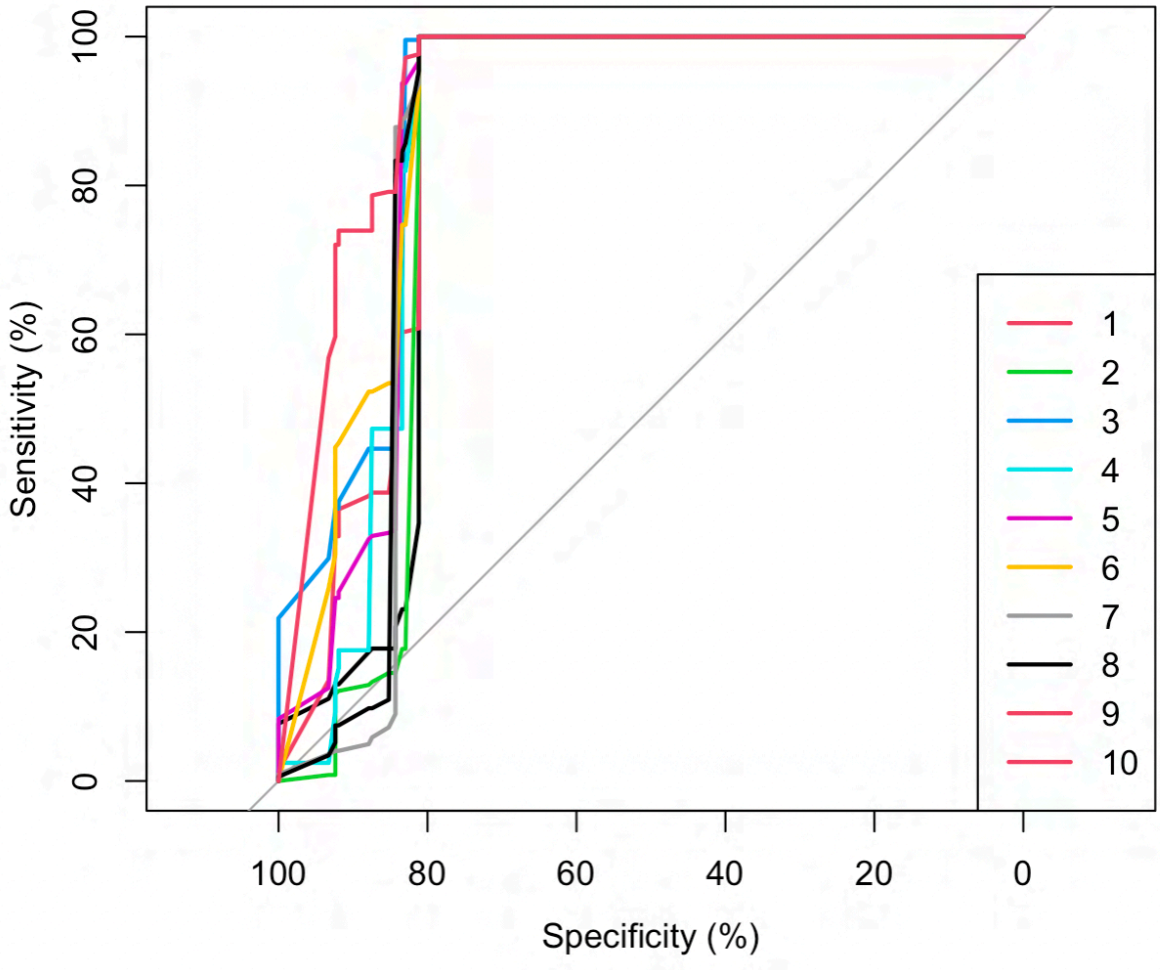
Tree Learning Phase – AUC & ROC

```
> tree_model2 = rpart(Letter~., data = trainingDataset_withoutOutliers, method = 'class')
> tree_class2_tr = predict(tree_model2, trainingDataset_withoutOutliers[, -which(names(training
Dataset_withoutOutliers)=="Letter")], type='class')
> tree_AUC2_tr = multiclass.roc(trainingDataset_withoutOutliers$Letter, as.numeric(tree_class2_
tr))
> print(paste("tree Learning-Phase AUC:",round(tree_AUC2_tr$auc,4)))
[1] "tree Learning-Phase AUC: 0.7305"
>
> tree_AUC2_rocs_tr = tree_AUC2_tr$rocs
> plot.roc(tree_AUC2_rocs_tr[[1]])
>
> for(i in 2:10){
+   lines.roc(tree_AUC2_rocs_tr[[i]],col=i)
+ }
> legend("bottomright", legend=c('1', '2','3','4','5','6','7','8','9','10'), col=2:10, lwd=2)
```



Tree Generalization Phase – AUC & ROC

```
> ## Tree Generalization Phase - AUC & ROC
> tree_PM2_tst = tree_cfm1_tst$byClass[,c("Balanced Accuracy", "Precision", "Sensitivity", "Spe
cificity", "Recall")]
> tree_test_model2 = rpart(Letter~., data = testingDataset_withoutOutliers, method = 'class')
> tree_prob2_tst=predict(tree_test_model2, testingDataset_withoutOutliers[, -which(names(testin
gDataset_withoutOutliers)=="Letter")], type = 'class')
> tree_AUC2_tst=multiclass.roc(testingDataset_withoutOutliers$Letter, as.numeric(tree_prob2_ts
t), percent = TRUE)
> print(paste("tree Generalization-Phase AUC:",round(tree_AUC2_tst$auc,4)))
[1] "tree Generalization-Phase AUC: 71.0016"
> tree_AUC2_rocs_tst <- tree_AUC2_tst$rocs
> plot.roc(tree_AUC2_rocs_tst[[1]])
> for(i in 2:10){
+   lines.roc(tree_AUC2_rocs_tst[[i]],col=i)
+ }
> legend("bottomright", legend=c('1', '2','3','4','5','6','7','8','9','10'), col=2:10, lwd=2)
```



Tree – Variance Estimation

```
> ## Tree Variance Estimation
> tree_varEst30=varEst(varEst_trdf, varEst_tstdf, 30, type="tree") # Variance estimation using
 30% of the data
> tree_varEst60=varEst(varEst_trdf, varEst_tstdf, 60, type="tree") # Variance estimation using
 60% of the data
> tree_varEst100=varEst(varEst_trdf, varEst_tstdf, 100, type="tree") # Variance estimation usin
g 100% of the data
```

Variance Estimation using 30% of the data

```
> print("Tree Variance Estimation using 30% of data:")
[1] "Tree Variance Estimation using 30% of data:"
> tree_varEst30
                            [,1]
Mean of Accuracies     0.4541000
Variance of Accuracies 0.0003805
```

Variance Estimation using 60% of the data

```
> print("Tree Variance Estimation using 60% of data:")
[1] "Tree Variance Estimation using 60% of data:"
> tree_varEst60
                            [,1]
Mean of Accuracies     4.515e-01
Variance of Accuracies 8.372e-05
```

Variance Estimation using 100% of the data

```
> print("Tree Variance Estimation using 100% of data:")
[1] "Tree Variance Estimation using 100% of data:"
> tree_varEst100
                          [,1]
Mean of Accuracies     0.4519
Variance of Accuracies 0.0000
```

Decision Tree algorithm are immune to multi-collinearity by nature. When they decide to split, the tree will choose only one of the perfectly correlated features. So, removing correlated feature variables will not be of much use here.

KNN

KNN Learning Phase

```
> ###############################################################################
> #                                    KNN                                      #
> ###############################################################################
> label <- trainingDataset_withoutOutliers$Letter
> # Explore different K values
> error.rate <- numeric(10)
> for(i in 1:10){
+   knn.pred <- knn(trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutliers)=="Lette
r")],
+                   testingDataset_withoutOutliers[, -which(names(testingDataset_withoutOutliers)=="Letter")],
label, k = i)
+   error.rate[i] <- 1-mean(knn.pred == testingDataset_withoutOutliers$Letter)
+ }
```

Mohit Shankar Pursnani

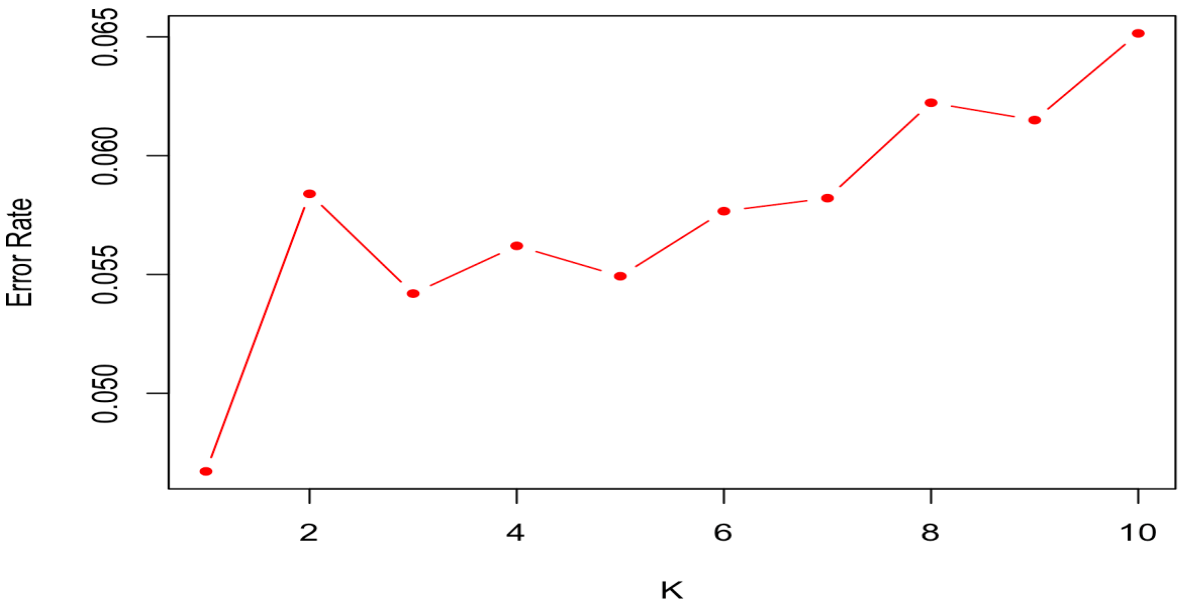
Letter Recognition Data Set

Assignment - 2

mp5578@nyu.edu

PLOT Error Rate for different values of K

```
> # Plot error rates
> plot(1:10, error.rate,"b", pch = 20, col = "red", xlab = "K", ylab = "Error Rate")
```



Make Prediction with k with 1

```
> # Make prediction with K = 1
> prediction <- knn(trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutliers)=="Letter")],
+                   testingDataset_withoutOutliers[, -which(names(testingDataset_withoutOutliers)=="Letter")],label, k =
 1)
> table(prediction,testingDataset_withoutOutliers$Letter)
```

```
> # Make prediction with K = 1
> prediction <- knn(trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutlier
s)=="Letter")],
+                   testingDataset_withoutOutliers[, -which(names(testingDataset_withoutOutliers)
=="Letter")],label, k = 1)
> table(prediction,testingDataset_withoutOutliers$Letter)
```

prediction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	222	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
2	1	195	0	1	0	1	0	5	0	1	1	0	1	0	0	0	0	6	1	1
3	0	0	214	0	2	0	2	0	0	0	0	0	0	0	1	0	0	0	0	1
4	0	2	0	235	0	0	2	3	0	0	0	0	0	2	3	3	0	1	0	0
5	0	0	1	0	212	0	2	0	0	0	1	2	0	0	0	0	0	0	1	0
6	0	0	1	0	0	195	0	0	0	0	0	0	0	0	0	16	0	0	0	1
7	0	0	2	0	4	0	229	0	0	0	1	1	1	0	0	0	1	0	0	0
8	0	4	0	4	0	0	0	155	0	0	10	0	1	1	0	0	0	3	0	0
9	0	0	0	0	0	0	0	0	215	4	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	0	7	167	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	1	0	1	5	0	0	187	0	0	0	0	0	0	3	0	0
12	0	0	1	0	0	0	0	0	0	0	0	165	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	186	2	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	176	0	1	0	4	0	0
15	0	0	0	2	0	0	3	4	0	0	0	0	0	0	205	0	0	0	0	0
16	0	0	0	0	1	7	0	0	0	0	0	0	0	0	0	203	0	0	0	0
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	5	1	228	0	0	0
18	0	3	1	5	0	0	0	1	0	0	3	2	0	1	0	1	1	225	1	0
19	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	170	0
20	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	207
21	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
22	0	2	0	0	0	0	1	0	0	0	0	0	4	0	0	0	0	0	0	0
23	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
24	0	0	0	0	2	0	0	0	0	0	8	0	0	0	0	0	0	0	0	1
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
26	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

prediction	21	22	23	24	25	26
1	0	0	0	0	0	0
2	0	5	0	1	0	0
3	0	0	0	0	0	0
4	0	0	0	1	0	0
5	0	0	0	1	0	1
6	0	0	0	0	0	0
7	0	1	0	0	0	0
8	5	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	1
11	0	0	0	5	0	0
12	1	0	0	0	0	0
13	0	0	1	0	0	0
14	0	0	0	0	0	0
15	0	0	2	0	0	0
16	0	1	0	0	0	0
17	0	0	0	1	0	3
18	0	0	0	0	0	0
19	0	0	0	1	0	0
20	0	0	0	0	2	0
21	176	0	0	0	1	0
22	0	216	0	0	0	0
23	1	3	203	0	0	0
24	0	0	0	229	0	0
25	0	2	0	0	223	0
26	0	0	0	1	0	188

KNN Test Rate

KNN Test Rate is 95.4197.

```
> knn.test.rate <- mean(prediction==testingDataset_withoutOutliers$Letter)
> print(paste("KNN Test Rate: ",round(knn.test.rate*100,4)))
[1] "KNN Test Rate:  95.4197"
```

KNN With Cross Validation

KNN Test Rate is 96.0633.

```
> # Same with Cross Validation
> knn.cv.pred <- knn.cv(dataset_without_outliers[, -which(names(dataset_without_outliers)=="Letter")], dataset_without_ou
tliers$Letter, k = 1)
> knn.cv.test.rate <- mean(knn.cv.pred == dataset_without_outliers$Letter)
> print(paste("KNN Test Rate: ",round(knn.cv.test.rate*100,4)))
[1] "KNN Test Rate:  96.0633"
```

KNN Confusion Matrix

```
> trdf_knn=trainingDataset_withoutOutliers[, -which(names(trainingDataset_withoutOutliers)=="Letter")]
> tstdf_knn=testingDataset_withoutOutliers[, -which(names(testingDataset_withoutOutliers)=="Letter")]
> trclass_knn=factor(trainingDataset_withoutOutliers[, which(names(trainingDataset_withoutOutliers)=="Letter")])
> tstclass_knn=factor(testingDataset_withoutOutliers[, which(names(testingDataset_withoutOutliers)=="Letter")])
> knn_pred=knn(trdf_knn,tstdf_knn,trclass_knn, k = 1, prob=TRUE)
> # Predict using test data (Generalization Phase)
> # Confusion Matrix for test data
> knn_cfm_tst=confusionMatrix(table(tstclass_knn,knn_pred))
> knn_cfm_tst
Confusion Matrix and Statistics
```

tstclass_knn \ knn_pred	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	222	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	194	0	2	0	0	0	3	0	0	0	0	0	0	0	0	0	5	1	0	0	2	0	0	0
3	0	0	214	0	1	1	2	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	1	0	0
4	0	2	0	231	0	0	0	4	0	0	0	0	0	0	5	0	0	5	1	0	0	0	0	0	0
5	0	1	2	0	212	0	3	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	2	0
6	0	1	0	0	0	195	0	0	0	0	0	0	0	0	0	7	1	0	1	0	0	0	0	0	0
7	0	0	2	2	2	0	229	0	0	0	1	0	0	0	3	0	0	0	0	0	0	1	0	0	0
8	0	5	0	3	0	0	0	155	0	0	5	0	0	0	4	0	0	1	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	215	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	0	0	0	0	0	0	4	167	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
11	0	1	0	0	2	0	1	9	0	0	187	0	0	0	0	0	0	3	0	0	0	0	0	8	0
12	0	0	0	0	2	0	1	0	0	0	0	165	0	0	0	0	0	2	0	0	0	0	0	0	0
13	0	2	0	0	1	0	1	0	0	0	0	0	186	0	0	0	0	0	0	0	0	3	1	0	0
14	1	0	0	1	0	0	0	2	0	0	0	0	2	177	0	0	0	0	0	0	0	0	0	0	0
15	0	0	1	3	0	0	0	0	0	0	0	0	0	0	205	0	5	0	0	0	0	0	0	0	0
16	0	0	0	1	0	16	0	1	1	0	0	0	0	1	0	206	0	0	0	1	0	0	0	0	0
17	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	228	1	0	0	0	0	0	0	0
18	0	7	0	0	1	0	0	4	0	0	2	0	0	3	0	0	0	225	0	0	0	0	0	0	0
19	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	169	2	0	0	0	0	0
20	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	207	0	0	0	1	5
21	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	177	0	1	0	0
22	0	4	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	216	3	0	2
23	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	204	0	0
24	0	1	0	1	1	0	0	0	0	0	4	0	0	0	0	0	1	0	1	0	0	0	0	230	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	222
26	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0

tstclass_knn \ knn_pred	26
1	0
2	1
3	0
4	0
5	2
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	1
18	0
19	1
20	0
21	0
22	0
23	0
24	1
25	0
26	189

KNN – Overall Statistics

```
Overall Statistics

               Accuracy : 0.9538
                 95% CI : (0.9479, 0.9592)
    No Information Rate : 0.0445
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.952

 Mcnemar's Test P-Value : NA
```

KNN – Statistics by Class

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.99107  0.87387  0.97273  0.94672  0.95067  0.91549  0.95816  0.84699  0.97727   0.95429
Specificity           0.99981  0.99734  0.99848  0.99675  0.99772  0.99810  0.99790  0.99641  0.99867   0.99868
Pos Pred Value        0.99552  0.93269  0.96396  0.93145  0.94643  0.95122  0.95417  0.89080  0.96847   0.95977
Neg Pred Value        0.99962  0.99469  0.99886  0.99752  0.99791  0.99659  0.99809  0.99472  0.99905   0.99849
Prevalence            0.04088  0.04051  0.04015  0.04453  0.04069  0.03887  0.04361  0.03339  0.04015   0.03193
Detection Rate        0.04051  0.03540  0.03905  0.04215  0.03869  0.03558  0.04179  0.02828  0.03923   0.03047
Detection Prevalence  0.04069  0.03796  0.04051  0.04526  0.04088  0.03741  0.04380  0.03175  0.04051   0.03175
Balanced Accuracy     0.99544  0.93561  0.98560  0.97174  0.97419  0.95680  0.97803  0.92170  0.98797   0.97648
                     Class: 11 Class: 12 Class: 13 Class: 14 Class: 15 Class: 16 Class: 17 Class: 18 Class: 19
Sensitivity            0.93500   0.98802   0.98413   0.97253   0.94037   0.95814   0.95798   0.92213   0.97688
Specificity            0.99545   0.99906   0.99849   0.99887   0.99829   0.99601   0.99943   0.99675   0.99868
Pos Pred Value         0.88626   0.97059   0.95876   0.96721   0.95794   0.90749   0.98701   0.92975   0.96023
Neg Pred Value         0.99753   0.99962   0.99943   0.99906   0.99753   0.99829   0.99809   0.99637   0.99925
Prevalence             0.03650   0.03047   0.03449   0.03321   0.03978   0.03923   0.04343   0.04453   0.03157
Detection Rate         0.03412   0.03011   0.03394   0.03230   0.03741   0.03759   0.04161   0.04106   0.03084
Detection Prevalence   0.03850   0.03102   0.03540   0.03339   0.03905   0.04142   0.04215   0.04416   0.03212
Balanced Accuracy      0.96523   0.99354   0.99131   0.98570   0.96933   0.97708   0.97871   0.95944   0.98778
                     Class: 20 Class: 21 Class: 22 Class: 23 Class: 24 Class: 25 Class: 26
Sensitivity            0.96729   0.98333   0.97297   0.97143   0.95436   0.96943   0.96923
Specificity            0.99829   0.99887   0.99772   0.99962   0.99809   0.99924   0.99924
Pos Pred Value         0.95833   0.96721   0.94737   0.99029   0.95833   0.98230   0.97927
Neg Pred Value         0.99867   0.99943   0.99886   0.99886   0.99790   0.99867   0.99887
Prevalence             0.03905   0.03285   0.04051   0.03832   0.04398   0.04179   0.03558
Detection Rate         0.03777   0.03230   0.03942   0.03723   0.04197   0.04051   0.03449
Detection Prevalence   0.03942   0.03339   0.04161   0.03759   0.04380   0.04124   0.03522
Balanced Accuracy      0.98279   0.99110   0.98535   0.98552   0.97622   0.98434   0.98424
```

KNN Generalization Phase Accuracy

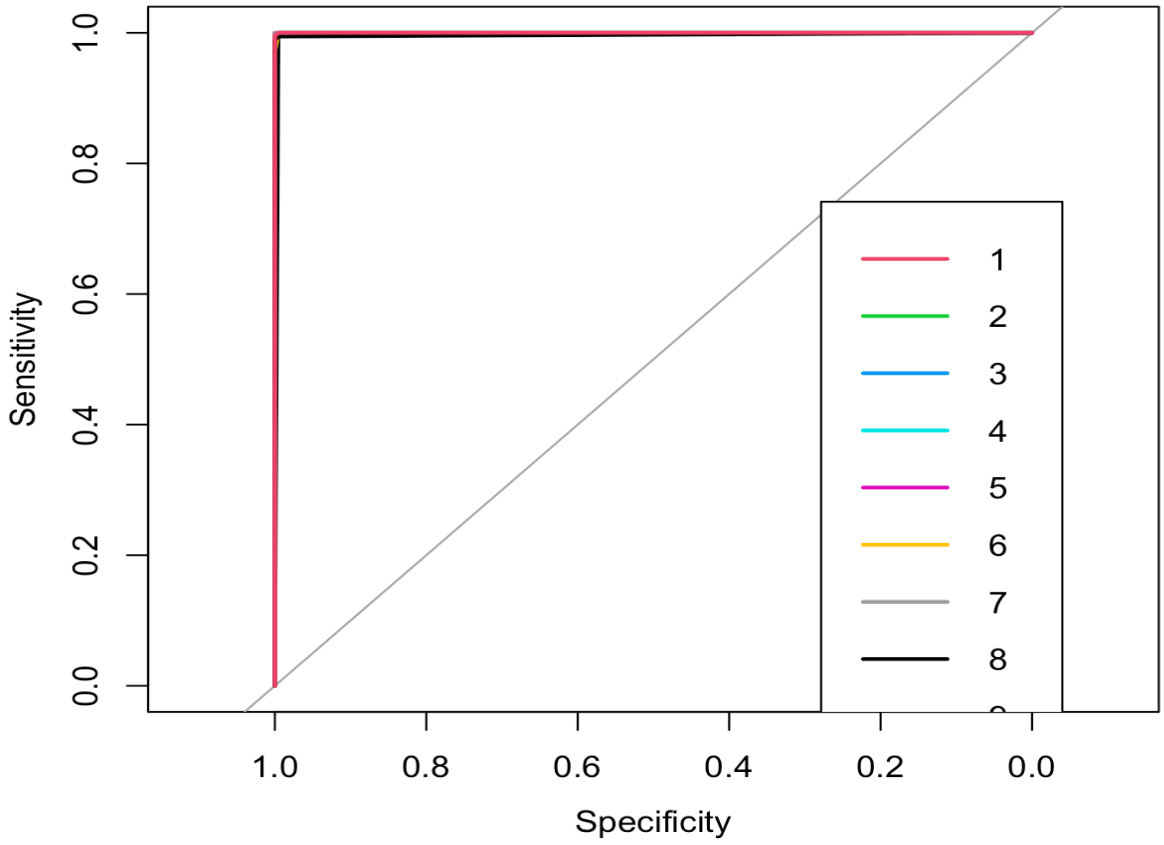
```
> knn_acc_tst=round(knn_cfm_tst$overall[["Accuracy"]],4)
> print(paste("kNN Generalization Phase Accuracy =",knn_acc_tst))
[1] "kNN Generalization Phase Accuracy = 0.9538"
```

KNN Generalization Phase Performance Parameters

```
> ###kNN Model Performance Metrics
> knn_PM_tst=knn_cfm_tst$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
> print("kNN Generalization-Phase Performance Parameters:")
[1] "kNN Generalization-Phase Performance Parameters:"
> knn_PM_tst
          Balanced Accuracy Precision Sensitivity Specificity    Recall
Class: 1          0.9954406 0.9955157   0.9910714   0.9998097 0.9910714
Class: 2          0.9356056 0.9326923   0.8738739   0.9973374 0.8738739
Class: 3          0.9856032 0.9639640   0.9727273   0.9984791 0.9727273
Class: 4          0.9717373 0.9314516   0.9467213   0.9967532 0.9467213
Class: 5          0.9741950 0.9464286   0.9506726   0.9977173 0.9506726
Class: 6          0.9567972 0.9512195   0.9154930   0.9981014 0.9154930
Class: 7          0.9780301 0.9541667   0.9581590   0.9979012 0.9581590
Class: 8          0.9217038 0.8908046   0.8469945   0.9964131 0.8469945
Class: 9          0.9879710 0.9684685   0.9772727   0.9986692 0.9772727
Class: 10         0.9764831 0.9597701   0.9542857   0.9986805 0.9542857
Class: 11         0.9652273 0.8862559   0.9350000   0.9954545 0.9350000
Class: 12         0.9935414 0.9705882   0.9880240   0.9990589 0.9880240
Class: 13         0.9913075 0.9587629   0.9841270   0.9984880 0.9841270
Class: 14         0.9856975 0.9672131   0.9725275   0.9988675 0.9725275
Class: 15         0.9693283 0.9579439   0.9403670   0.9982896 0.9403670
Class: 16         0.9770755 0.9074890   0.9581395   0.9960114 0.9581395
Class: 17         0.9787054 0.9870130   0.9579832   0.9994277 0.9579832
Class: 18         0.9594422 0.9297521   0.9221311   0.9967532 0.9221311
Class: 19         0.9877798 0.9602273   0.9768786   0.9986810 0.9768786
Class: 20         0.9827903 0.9583333   0.9672897   0.9982909 0.9672897
Class: 21         0.9911006 0.9672131   0.9833333   0.9988679 0.9833333
Class: 22         0.9853454 0.9473684   0.9729730   0.9977178 0.9729730
Class: 23         0.9855245 0.9902913   0.9714286   0.9996205 0.9714286
Class: 24         0.9762240 0.9583333   0.9543568   0.9980912 0.9543568
Class: 25         0.9843353 0.9823009   0.9694323   0.9992382 0.9694323
Class: 26         0.9842370 0.9792746   0.9692308   0.9992431 0.9692308
```

KNN Generalization – AUC & ROC

```
> knn_prob_tst=attr(knn_pred,"prob")
> knn_AUC_tst=multiclass.roc(tstclass_knn, as.ordered(knn_pred))
> print(paste("kNN Generalization-Phase AUC:",round(knn_AUC_tst$auc,4)))
[1] "kNN Generalization-Phase AUC: 0.9716"
> # ROC curves
> knn_ROC_tst=knn_AUC_tst$rocs
> plot.roc(knn_ROC_tst[[1]], col=1)
> for(i in 2:10){
+   lines.roc(knn_ROC_tst[[i]],col=i)
+ }
> legend("bottomright", legend=c('1', '2','3','4','5','6','7','8','9','10'), col=2:10, lwd=2)
```



KNN – Variance Estimation

```
> knn_varEst30=varEst(varEst_trdf, varEst_tstdf, 30, type="knn") # 30% of data
> knn_varEst60=varEst(varEst_trdf, varEst_tstdf, 60, type="knn") # 60% of data
> knn_varEst100=varEst(varEst_trdf, varEst_tstdf, 100, type="knn") # 100% of data
```

Variance Estimation using 30% of the data

```
> print("kNN Variance Estimation using 30% of data:")
[1] "kNN Variance Estimation using 30% of data:"
> knn_varEst30
                            [,1]
Mean of Accuracies     7.941e-01
Variance of Accuracies 9.483e-05
```

Variance Estimation using 60% of the data

```
> print("kNN Variance Estimation using 60% of data:")
[1] "kNN Variance Estimation using 60% of data:"
> knn_varEst60
                            [,1]
Mean of Accuracies     8.817e-01
Variance of Accuracies 5.516e-05
```

Variance Estimation using 100% of the data

```
> print("kNN Variance Estimation using 100% of data:")
[1] "kNN Variance Estimation using 100% of data:"
> knn_varEst100
                            [,1]
Mean of Accuracies     9.279e-01
Variance of Accuracies 6.086e-06
```

Overall

For this dataset we observed that all feature variables are equally important, removing the feature variables only reduced the performance of the model.
The accuracy of the models QDA, SVM and KNN is high as compared to other models and the AUC of QDA is 0.99 which is very close to 1, which shows that the model has a good measure of separability and its variance with 100% of the data is zero which shows that the model predicted value is very close or identical to the expected value. The variance estimates of SVM and kNN models are also very close to zero or zero if we round off the value to 4 decimal places. Hence there is very slight difference between these models in terms of Variance. As far as accuracy is concerned SVM and KNN performed better than QDA.
The decision tree algorithm performed the worst as with more class labels the prediction becomes more complex. Also, decision tree while building model used only some of the feature variables as Decision Tree is immune to multicollinearity, while building model it considers only few of the feature variables. But as we saw using all feature variables is important for this dataset hence tree accuracy is worst.
LDA accuracy was improved drastically when LDA model was built using predictor interaction. It showed us that the prediction of dependent variable in our data set is dependent on more than one feature variable.

Observation	LDA	QDA	SVM	Tree	kNN
Accuracy	0.7228	91.88	0.9401	0.4555	0.9545
Balanced Accuracy	0.8549	0.9573	0.9698	0.5631	0.83
Specificity	0.9889	0.9968	0.9420	0.4066	0.78
Precision	0.7342	0.9189	0.9390	0.4547	0.94
Recall	0.7209	0.9179	0.9420	0.4066	0.87
AUC	0.9692	0.9980	0.9636	0.7100	0.9716
Mean Variance Estimation using 30% of data	7.196e-01	0.8625	8.821e-01	0.4541	7.941e-01

Mean Variance Estimation using 60% of data	0.7263	0.8767	9.281e-01	4.515e-01	8.817e-01
Mean Variance Estimation using 100% of data	0.7326	0.8843	9.535e-01	0.4519	9.279e-01
Variance Estimation using 30% of data - Variance	3.001e-06	0.0000506	4.168e-05	0.0003805	9.483e-06
Variance Estimation using 60% of data - Variance	2.481e-06	0.0000174	1.473e-07	8.372e-05	5.516e-06
Variance Estimation using 100% of data - Variance	0.0000	0.0000	9.849e-07	0.0000	6.086e-06