A Project On:

Classification of Stars, Galaxies, and Quasars. Sloan Digital Sky Survey DR17

Rinkle Agarwal
Abdul Rashid Zakaria

MA 5790
April 2022

Abstract

This report details predictive models built using data from the DR17 release of the Sloan Digital Sky Survey using linear and non-linear classifiers. The dataset will be first preprocessed to remove identifiers used in collecting the data, remove non-zero variances, and solve outliers within the distribution. Predictive models to be deployed will include a penalized multinomial regression model, linear discriminant model, partial least squares discriminant analysis model, a penalized logistic regression model implemented using 'glmnet' in R-studio, sparse linear discriminant analysis, mixture discriminant analysis model, neural networks, flexible discriminant analysis model, a support vector machine, K-nearest neighbor model for classification and finally a naïve Bayes model. The predictive models will be built using the dataset with predictors providing information on the spectral properties of the classes, galaxies, quasars, and stars. The top two models resulting from the training set will be then used to predict the unseen testing data to evaluate their performances.

Table of Contents

1 Background

The classification of stars based on their spectral features is known as stellar classification in astronomy. One of the most basic classification schemes in astronomy is that of galaxies, quasars, and stars. As more powerful telescopes were created, the early cataloging of stars and their distribution in the sky led to the knowledge that they make up our own galaxy, and after the finding that Andromeda was a separate galaxy from our own, several galaxies began to be examined. This dataset will use spectral properties to classify stars, galaxies, and quasars.

2 Variable Definitions and Structure

The data consists of 100,000 observations of space taken by the SDSS (Sloan Digital Sky Survey). Each observation is featured by 17 attribute columns and 1 response class column which identifies it to be a star, galaxy, or a quasar based on their several spectral characteristics. Below is a list of all the variables and their descriptions:

Variable Name	Description
1. obj_ID	ID to uniquely identify objects
2. alpha	Right Ascension angle
3. delta	Declination angle
4. u	Ultraviolet filter in the photometric system
5. g	Green filter in the photometric system
6. r	Red filter in the photometric system
7. i	Near Infrared filter in the photometric system
8. z	Infrared filter in the photometric system
9. run_ID	Run identification to specify image scan
10. rerun_ID	Rerun identification to describe image
11. cam_col	Camera column to identify run
12. field_ID	Field identification number
13. spec_obj_ID	Unique ID for spectroscopic objects
14. class	Object class (star, galaxy or quasar)
15. redshift	Wavelength shift
16. plate	Unique identification for object plate
17. MJD	Date when object was recorded
18. Fiber_ID	Fiber identification for each object

All the predictor variables are continuous, and the response variable is categorical. Based on the distributions of the variables, viable pre-processing techniques will be applied to each before proceeding to the model building phase. After that, both linear and non-linear classification models will be deployed to predict the class of the object.

3 Pre-processing

a. Dummy Variables

Since all the predictors are continuous variables, there was no need to create and add dummy variables to the dataset.

b. Deleting Identifiers:

Certain predictors in the data did not include relevant information that would help the model make predictions and were used to only describe certain aspects of the data. Hence, they needed to be removed from the data. The variables were obj_ID, run_ID, rerun_ID, cam_col, field_ID, spec_obj_ID, fiber_id, MJD and plate. After deleting these variables, there were a total of 8 predictor variables left.

c. Missing Values:

There were no missing values identified in the data. So, no processing steps to handle such a scenario were required.

d. Correlation:

A correlation plot to explore the relationship between the remaining continuous variables was determined. The figure below shows the correlation plot between the predictors and suggests that there are some strong correlations between the variables g, z, and u. However, we decided not to remove these predictors but instead to deploy models that can handle strong correlations.

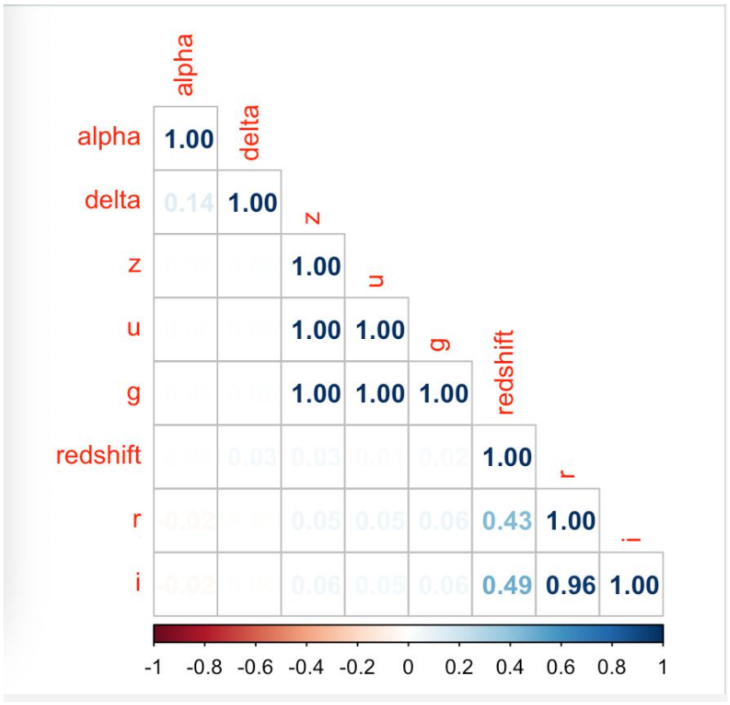


Fig.1 Correlation plot

e. Exploring Feature Distribution:

It is necessary to explore and analyse the distribution of the data before moving forward with the models and making necessary transformations. We explored the distributions of the variables using histograms and boxplots to identify skewness and outliers in the data.

i. Skewness

Firstly, we explored the distribution of the variables using histograms and determined a certain degree of skewness in a few of them. The diagram below shows the distribution of all the continuous variables. It can be observed that the variables u, g, z, and redshift are highly skewed.

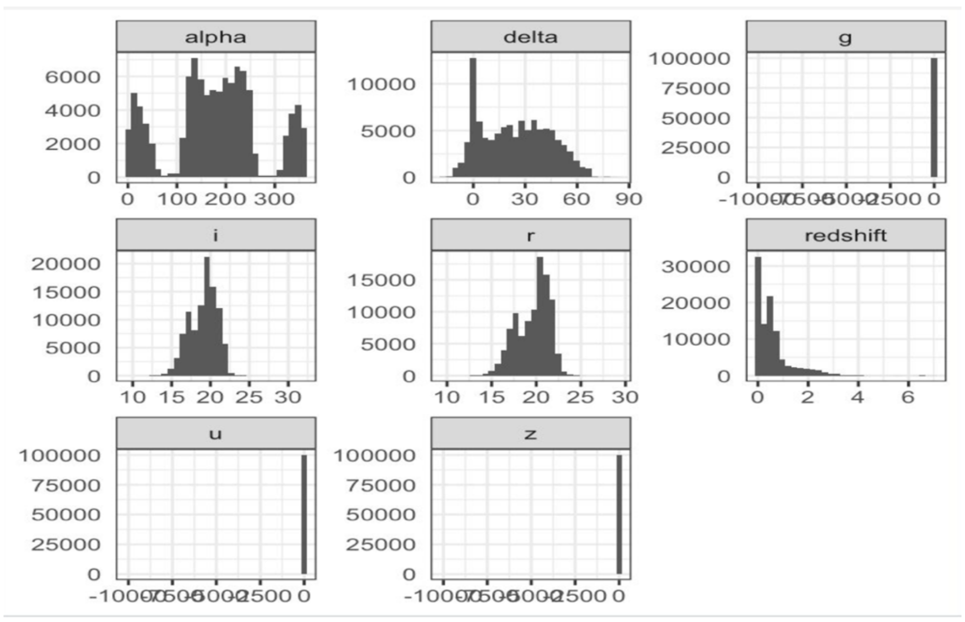


Fig. 2 Histograms of the continuous variables

Before applying the traditional approach to solve the issue of skewness, we decided to dig deep into the statistics of these variables to determine any underlying causes of the skewness. For the variables u, g and z there was inconsistent data that was making the data skewed. So we identified those values and removed them from the data. Finally, we applied Box-Cox transformation on the variables to address any additional skewness. The diagram below shows the distribution of the variables after removing the inconsistent data and Box-Cox transformation.

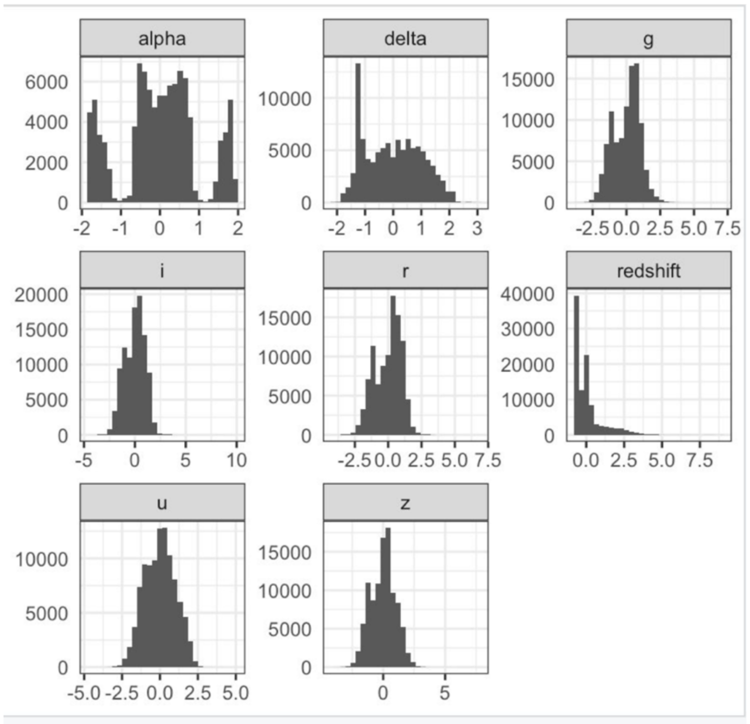


Fig. 3 Histograms of the continuous variables after Box-Cox

ii. Outliers

The next step was to determine the presence of outliers in the data. We used boxplots to see the distribution of the variables and determine if any outliers existed in the data. The diagram below shows the boxplots of all the continuous variables.

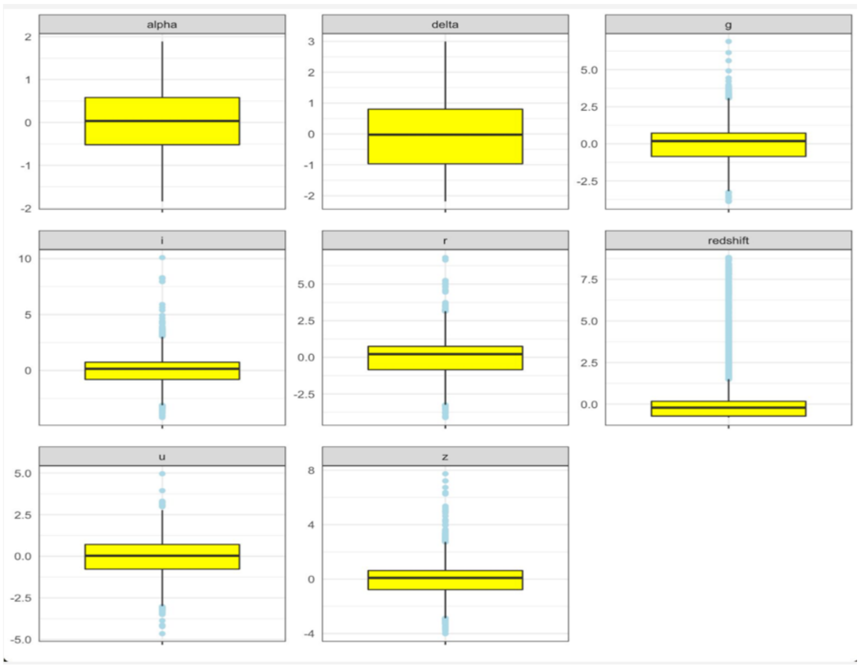


Fig. 4 Boxplots of the continuous variables

We can observe that a large amount of outliers exist in the data and they had to be removed in order to create effective prediction models. We used Spatial Sign transformation to resolve this issue. The boxplots below show the distribution of the variables after the Spatial Sign transformation was applied . We can observe that all the variables, except redshift, contained zero to little outliers afterwards.

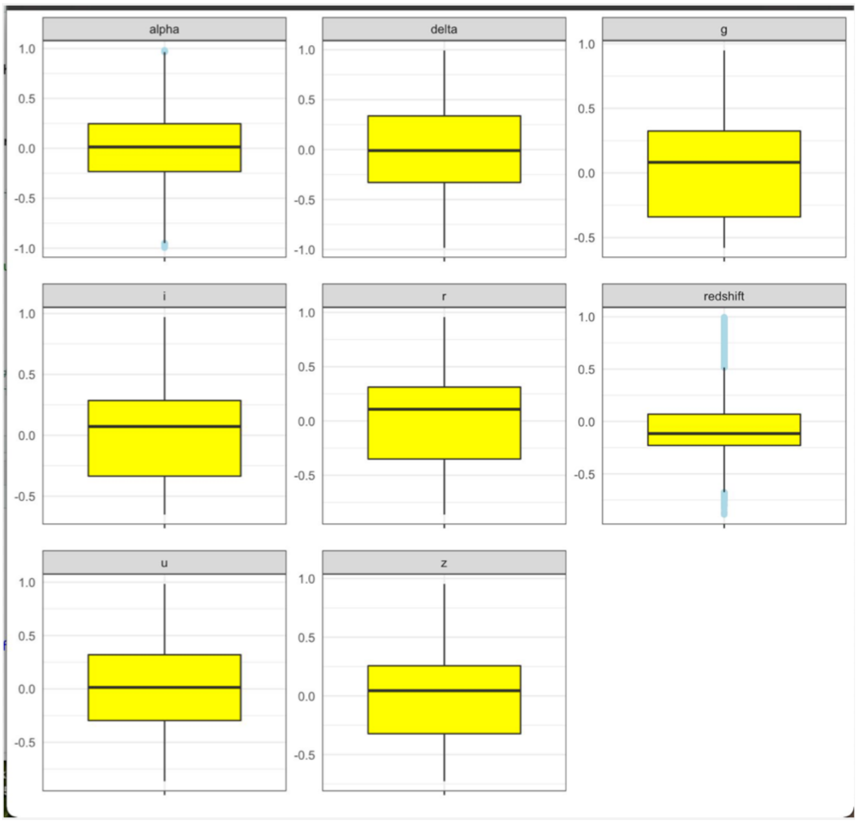


Fig. 5 Boxplots of the continuous variables after Spatial Sign

4 Data Spending

a) Data Splitting

After the pre-processing was done, we were left with a total of 99,999 samples and 8 predictor variables. The next step was to determine the data spending technique. The bar plot below shows the distribution of the response variable. It can be observed that it is highly imbalanced. So, the optimal spitting technique would be stratified random sampling. We used 70% of the data as training set and the remaining 30% as the testing set.

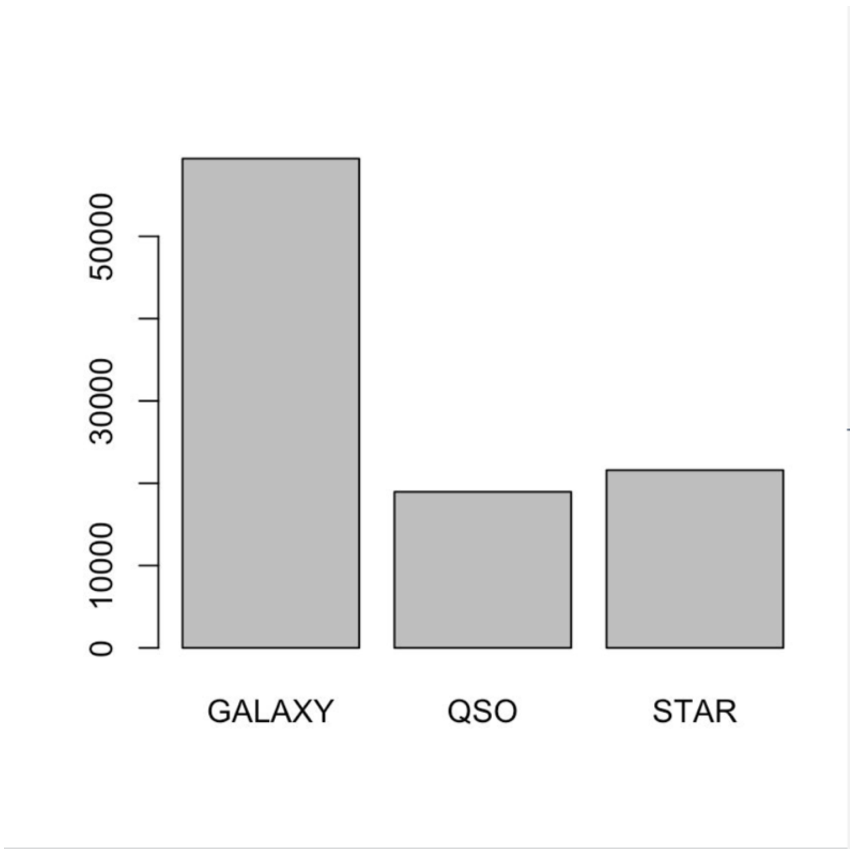
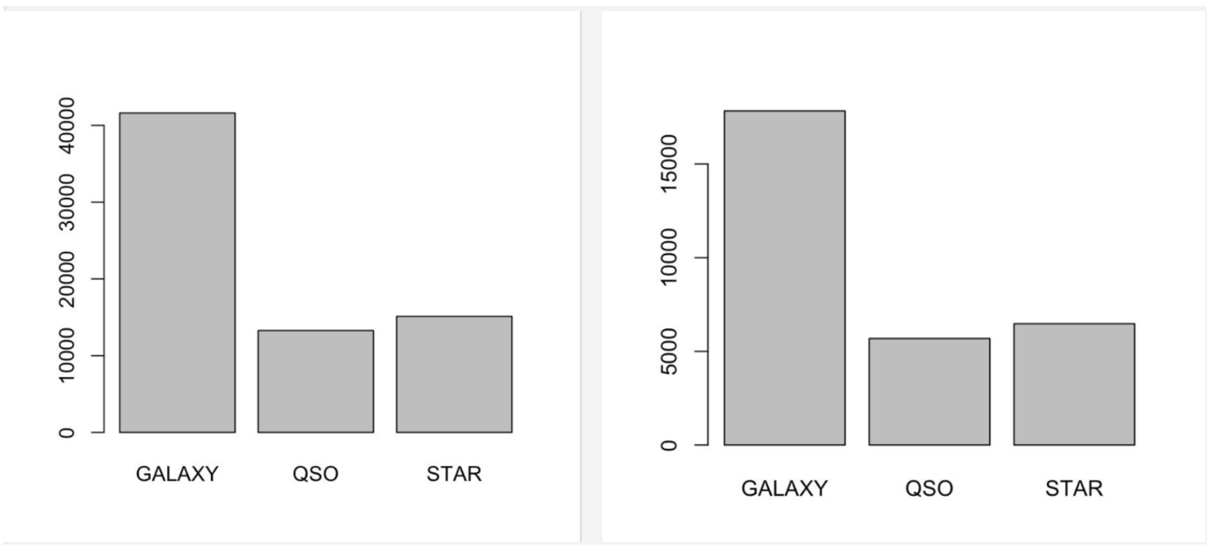


Fig. 6 Distribution of the response variable

After splitting we have, 70,000 samples in the training set and 29,999 in the testing set. The bar plots below show the distribution of the response classes in the training and the testing sets. We can see the frequency of all the three classes i.e. Galaxy, Star and Quasar is approximately the same in both the sets.



Training Set **Testing Set**

Fig. 7 Distribution of the response variable in the training and testing sets

b) Data Resampling

Since our dataset is large, we decided to use 10-fold cross validation to evaluate the performance of the models. This resampling technique is useful in identifying over-fitting issues and provides acceptable variance with low bias. Additionally, it balances out the classes in the response variables of unbalanced datasets.

5 Model Fitting

All of the data was centred and scaled prior to training the models. In training all models, 10-fold cross-validation was used for resampling. Additionally, Kappa was chosen as the metric for selecting the best models. Using the Kappa value from the training set, the two best models were chosen. and the final model was chosen using predictions on the test data set. The R-code for model fitting and testing is given in Appendix A.

a. Linear Models

i. Penalized Multinomial Regression

The optimal model selected a decay value of 0 which is shown in the tuning plot below. The best kappa value resulted to be 0.77.

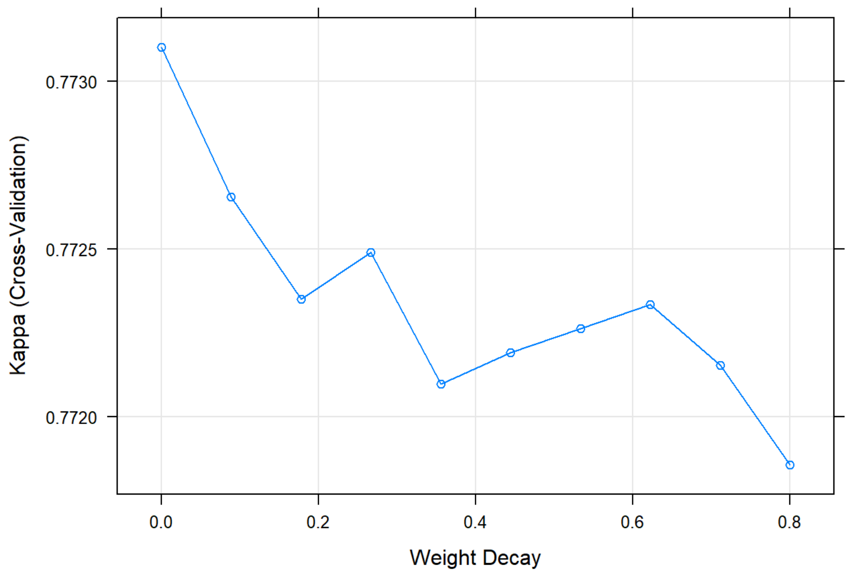


Fig. 8 Tuning parameter plot

ii. Linear Discriminant Analysis

The model has no tuning parameter. The kappa value came out to be 0.71.

iii. Partial Least Squares Discriminant Analysis

The optimal number of components was 6 which gave a kappa value of 0.67 as shown below.

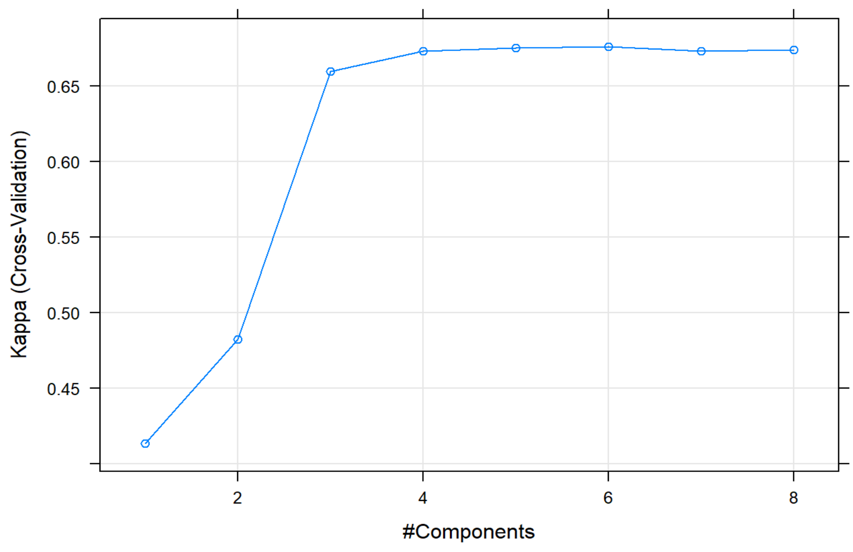


Fig. 9 Tuning parameter plot

iv. Penalized Logistic Regression

The optimal tuning parameters were alpha =0 .6 and lambda = 0 which gave a kappa value of 0.77 as observed below.

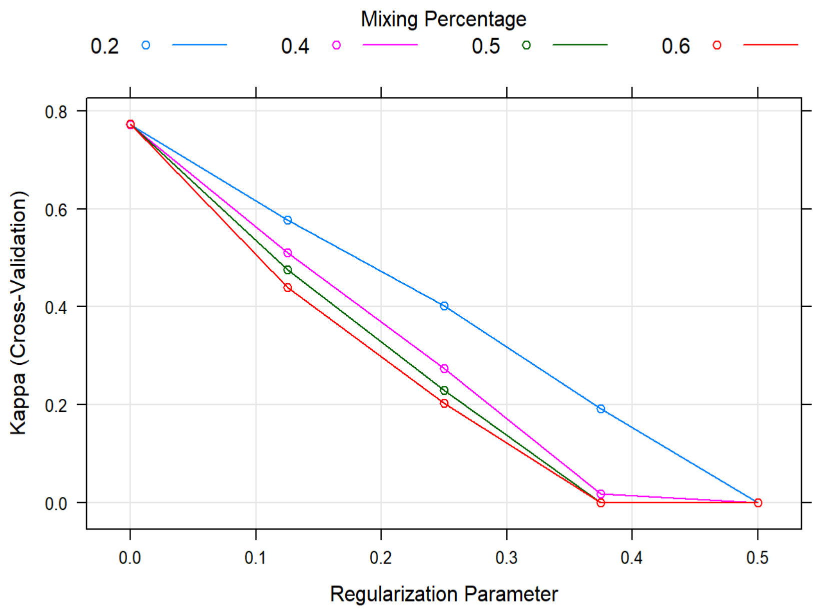


Fig. 10 Tuning parameter plot

v. Spatial Linear Discriminant Analysis

The optimal tuning parameters were Numvars = 3 and lambda = 0.2 with a kappa value of 0.71. The figure below shows the tuning parameter plot for the model.

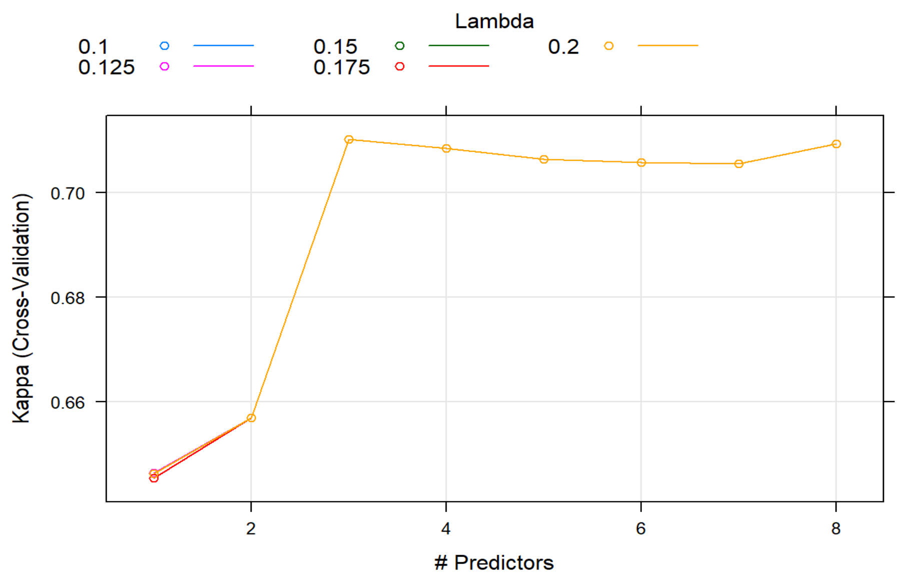


Fig. 11 Tuning parameter plot

A summary of the linear models, the best tuning parameters, and the results from predictions on the training dataset is given in the table below.

MODEL	OPTIMUM TUNING PARAMETER(S)	KAPPA VALUE
Penalized Multinomial Regression	Decay = 0	0.7731030
Linear Discriminant Analysis	No tuning parameter	0.7092975
Partial Least Squares Discriminant Analysis	ncomp=6	0.6763567
Penalized Logistic Regression	alpha = 0.6 and lambda = 0	0.7724555
Sparse Linear Discriminant Analysis	Num vars = 3 and lambda = 0.2	0.7100993

b. Non-Linear Models

i. Non-linear discriminant analysis

The optimum number of subclasses were thirty-three which gave a kappa value of 0.80 as shown below.

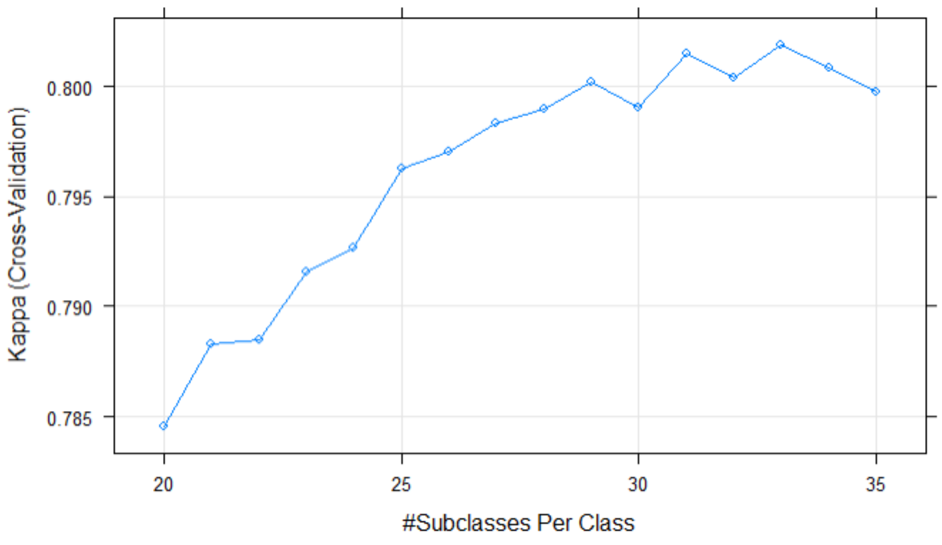


Fig. 12 Tuning parameter plot

ii. Neural Network

The optimal tuning parameters were size = 8 and decay = 0 with a kappa value of 0.89 as shown below.

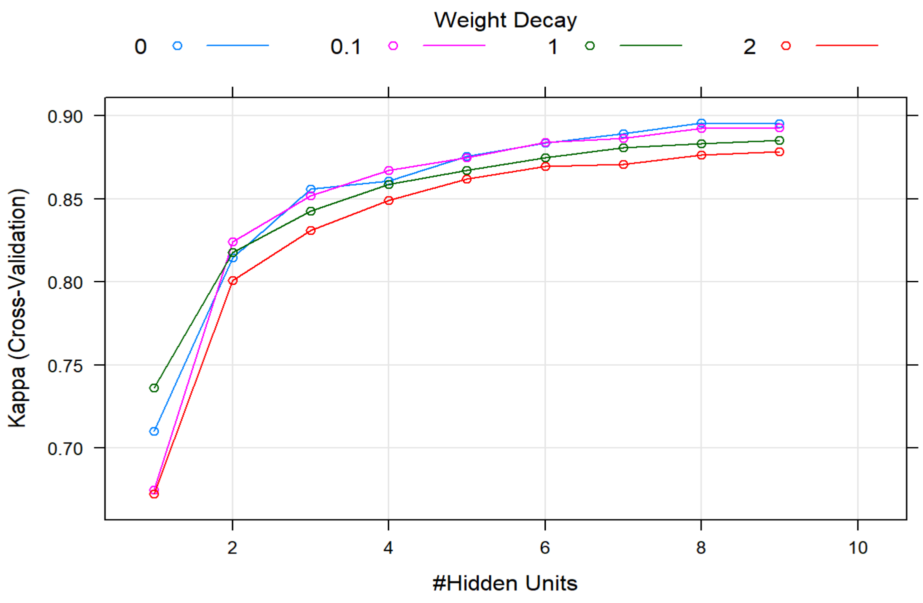


Fig. 13 Tuning parameter plot

iii. Flexible Discriminant Analysis

The optimal tuning parameters were degree = 2 and nprune = 19. Th best kappa value was 0.79 which is shown below.

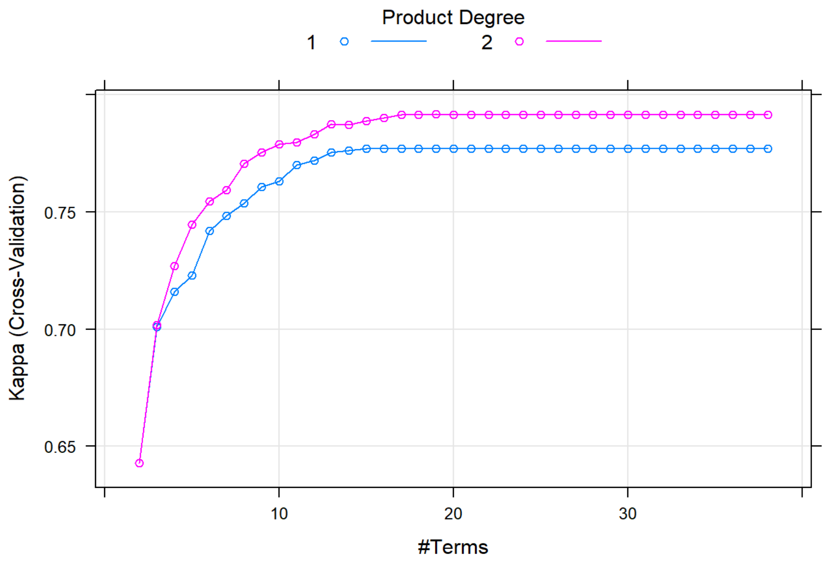


Fig. 14 Tuning parameter plot

iv. Support Vector Machine

The optimal tuning parameter, sigma = 0.0350143 and C = 1024 gave a kappa value of 0.90 as shown below .

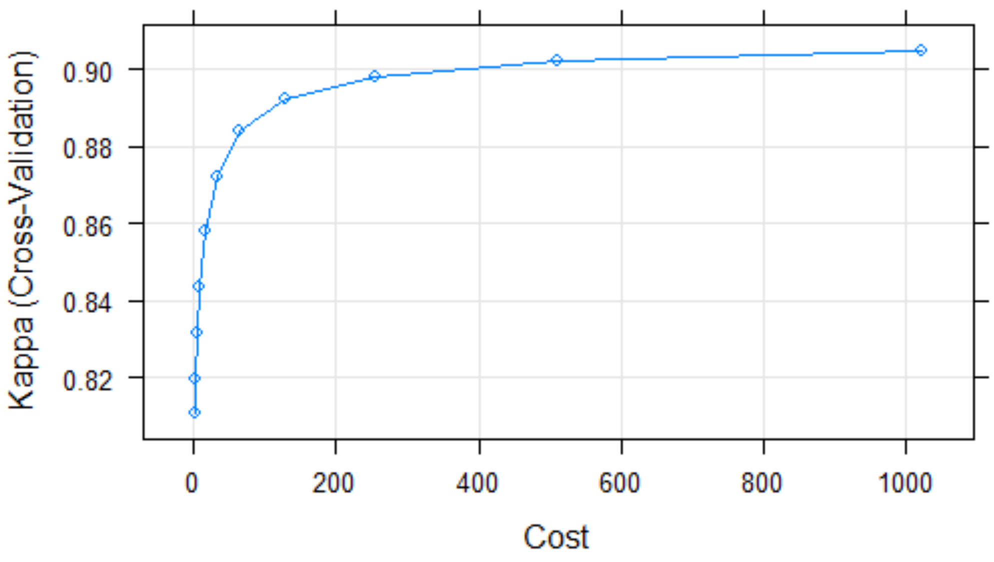


Fig. 15 Tuning parameter plot

Extending the cost beyond C = 2^{10}, greatly increased the computational time, the last model trained took more than two days without any significant increase in the Kappa metric. Hence, the final model cost was limited to 1024. Moreover, increasing the cost ultimately results in overfitting of the model.

v. K-Nearest Neighbours

The optimal tuning parameter k=5 gave a kappa value of 0.87.

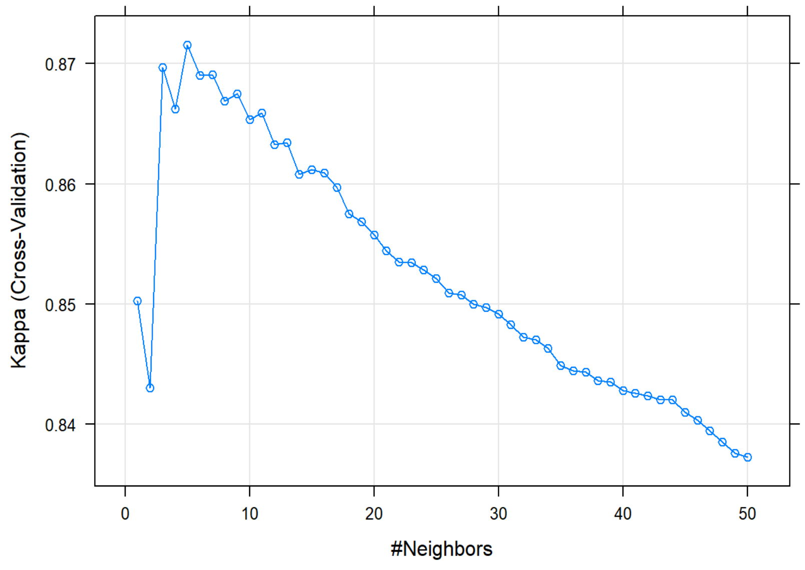


Fig. 16 Tuning parameter plot

vi. Naïve Bayes Model

There is no tuning parameter for the model The best kappa value resulted out to be 0.66.

A summary of the non-linear models, the best tuning parameters, and the results from predictions on the training dataset is given in the table below.

MODEL	OPTIMUM TUNING PARAMETER(S)	KAPPA VALUE
Non-Linear Discriminant Analysis	subclasses = 33	0.8018798
Neural Networks	**size = 8 and decay = 0**	**0.8953934**
Flexible Discriminant Analysis	degree = 2 and nprune = 19	0.7916896
Support Vector Machines	**sigma = 1 and C = 8**	**0.9049372**
K-Nearest Neighbours	k = 5	0.8715700
Naive Bayes	No tuning parameter	0.6644922

6 Best Models Summary

The best two models were the support vector machine and the neural network models. The models were then used to predict the response of the test data set. The Kappa metric was used to choose the best model. A summary of the results is given below.

MODEL	KAPPA VALUE
Neural Networks	0.8937
Support Vector Machines (SVM)	**0.9109**

It can be observed that SVM performed slightly better than Neural Network model and hence was selected as the best model. The confusion Matrix from the SVM is given below. The model's important variables used in the prediction the different classes indicates 'redshift', 'z' and 'I' as the most important predictors for all the classes.

Confusion Matrix Table

		Observations		
		Galaxy	Quasar	Star
Predictions	Galaxy	17084	404	332
	Quasar	235	5271	10
	Star	514	13	6135

Confusion Matrix for the SVM Model

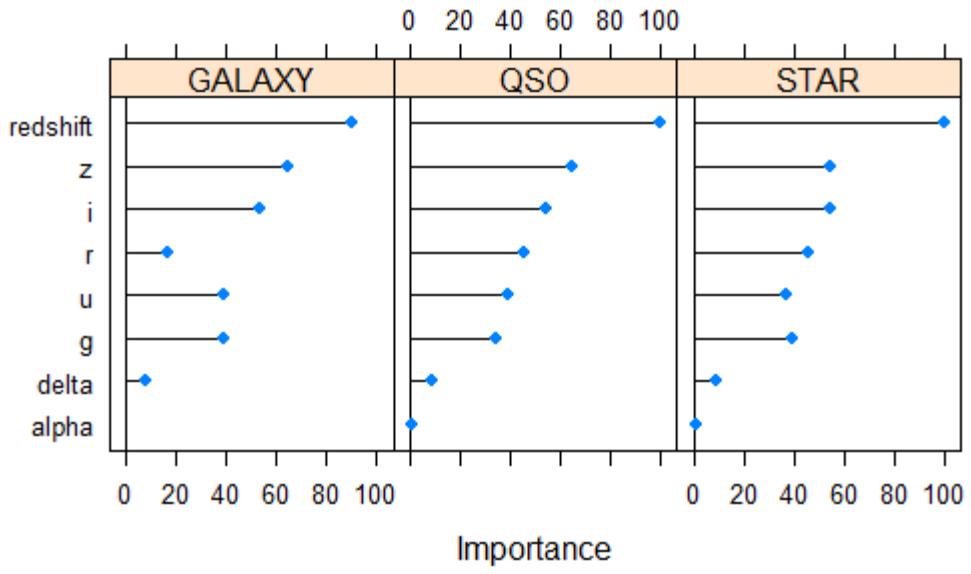


Fig. 17 Important variables for the SVM Model

7 Conclusion

In conclusion, SVM and Neural Network performed the best on the training set with similar kappa values of 0.90 and 0.89 respectively. Hence, we decided to move forward with these two models to make predictions on the testing data. Again, both the models performed similarly with SVM giving a kappa value of 0.91 and Neural Network giving a kappa value of 0.89. Since SVM provided a higher kappa value it was chosen as the best model for our dataset. However, both the models were computationally expensive.

R-code

```
#Loading libraries
library(tidyverse)
library(ggplot2)
library(dplyr)
library(moments)
library(e1071)
library(caret)
library(kableExtra)
library(corrplot)
library(knitr)
library(lattice)
library(psych)
library(car)
library(kernlab)
library(caret)
library(doParallel)

#set the number of processors
cl <- makePSOCKcluster(15)
registerDoParallel(cl)



#Loading dataset
df <- read.csv("star_classification.csv")
#df
nrow(df)

#check for missing data
sum(is.na(df))

#creating predictor df and response df and dropping unwanted predictors
```

```
predictors<-df%>%select(c(-obj_ID, -class, -rerun_ID,-run_ID, -cam_col,-field_ID, -spec_ob
j_ID, -fiber_ID, -MJD, -plate))

response<-df%>%select(c(class))


#predictors

#response


filteredPredictors<-df%>%select(c(-obj_ID, -class, -rerun_ID,-run_ID, -cam_col,-field_ID, -
spec_obj_ID, -fiber_ID, -MJD, -plate))


#checking correlation

correlations = cor(predictors)

corrplot(correlations, order = "hclust", method = "number", type = "lower")


highCorr <- findCorrelation(correlations, cutoff = .95)

length(highCorr)


#plotting histograms to see distribution

filteredPredictors %>%

  gather() %>%

  ggplot(aes(x = value))+

  geom_histogram() +

  facet_wrap(~ key, scales = "free") +

  labs(x = NULL, y = NULL) +

  theme_bw() +

  theme(axis.ticks.y=element_blank())


#checking skewness values

skewValues <- apply(filteredPredictors, 2, skewness)

#skewValues


#determining bad data

#describe(filteredPredictors$u)
```

```
#describe(filteredPredictors$g)
#describe(filteredPredictors$z)

#filteredPredictors[which(filteredPredictors$u <0), ]
#filteredPredictors[which(filteredPredictors$g <0), ]
#filteredPredictors[which(filteredPredictors$z <0), ]

#dropping bad observation
filteredPredictors<-filteredPredictors%>%slice(-c(79544))
response<-response%>%slice(-c(79544))

#checking skewness values
skewValues <- apply(filteredPredictors, 2, skewness)
#skewValues

#pre-processing transformations
trans <- preProcess(filteredPredictors, method = c("BoxCox"))
#trans

# Apply the transformation:
transformedPred<- predict(trans, filteredPredictors)
#transformedPred

#checking skewness after transformation
skewValues <- apply(transformedPred, 2, skewness)
#skewValues

#plotting histograms to see distribution
transformedPred %>%
  gather() %>%
  ggplot(aes(x = value))+
  geom_histogram() +
  facet_wrap(~ key, scales = "free") +
  labs(x = NULL, y = NULL) +
  theme_bw() +
  theme(axis.ticks.y=element_blank())

#checking for outliers via box plots

transformedPred %>%
  gather() %>%
  ggplot(aes(x ="", y = value))+
  geom_boxplot(outlier.colour = "lightblue", fill="yellow") +
  facet_wrap(~ key, scales = "free") +
  labs(x = NULL, y = NULL) +
  theme_bw() +
  theme(axis.ticks.y=element_blank())

#removing outliers by SpatialSign
#pre-processing transformations
trans <- preProcess(transformedPred, method = c("spatialSign"))
#trans

# Apply the transformation:
transformedPred<- predict(trans, transformedPred)
#transformedPred

#checking for outliers after transformation
transformedPred %>%
  gather() %>%
  ggplot(aes(x ="", y = value))+
  geom_boxplot(outlier.colour = "lightblue", fill="yellow") +
  facet_wrap(~ key, scales = "free") +
  labs(x = NULL, y = NULL) +
  theme_bw() +
  theme(axis.ticks.y=element_blank())
```

```
##################################################
#checking distribution of classes to decide on splitting method
barplot(table(response$class))

set.seed(980)

#stratifed random sampling
trainingRows <- createDataPartition(response$class, p = .70, list= FALSE)
nrow(trainingRows)

#creating training and testing data

trainPredictors <- transformedPred [trainingRows, ]
trainClasses <- response[trainingRows]

str(trainClasses)

# Do the same for the test set using negative integers.
testPredictors <- transformedPred[-trainingRows, ]
testClasses <- response[-trainingRows]

str(trainPredictors)
str(testPredictors)
str(trainClasses)
str(testClasses)

nrow(trainPredictors)
nrow(testPredictors)

#summary of data
```

```
summary(trainPredictors)
sum(is.na(trainPredictors))

#checking frequency distribution of training and test classes
barplot(table(response$class))
barplot(table(trainClasses))
barplot(table(testClasses))

#model building

#1.Logistic Regression
set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)
lrGrid <- expand.grid(.decay = seq(0, .8, length = 10))  ## use sequence till .8

set.seed(980)
lrFit <- train(x=trainPredictors,
            y = trainClasses,
            method = "multinom",
            metric = "Kappa",
            trControl = ctrl,
            tuneGrid = lrGrid)

lrFit
plot(lrFit)

predictiedLR<-predict(lrFit, testPredictors)
confusionMatrix(data = predictiedLR,
             reference = as.factor(testClasses))

#2.Linear Discriminant Analysis
library(MASS)
```

```
set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
ldaFit <- train(x = trainPredictors,
              y = trainClasses,
              method = "lda",
              metric = "Kappa",
              preProc = c("center", "scale"),
              trControl = ctrl)
ldaFit

predictedLDA <- predict(ldaFit, testPredictors)
confusionMatrix(data = predictedLDA,
              reference = as.factor(testClasses))


#3.Partial Least Squares Discriminant
set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
plsFit<- train(x = trainPredictors,
             y = trainClasses,
             method = "pls",
             tuneGrid = expand.grid(.ncomp = 1:8),
             preProc = c("center","scale"),
             metric = "Kappa",
             trControl = ctrl)
plsFit
plot(plsFit)

predictedPLS <-predict(plsFit, testPredictors)
```

```
confusionMatrix(data = predictedPLS,
            reference = as.factor(testClasses))



#4. Penalized Logistic Regression Model
plgGrid <- expand.grid(.alpha = c(.2, .4, .5, .6),
                 .lambda = seq(.0, .5, length = 5))   ##change tuning values to get a curve
set.seed(980)
ctrl <- trainControl(method = "cv", number = 15, summaryFunction = defaultSummary)

set.seed(980)
plgFit <- train(x=trainPredictors,
            y =trainClasses,
            method = "glmnet",
            tuneGrid = plgGrid,
            preProc = c("center", "scale"),
            metric = "Kappa",
            trControl = ctrl)

plgFit
plot(plgFit)
predictedPLG <-  predict(plgFit, testPredictors)
confusionMatrix(data = predictedPLG,
            reference = as.factor(testClasses))



#5.Sparse LDA
library(sparseLDA)
ldaGrid <- expand.grid(.lambda = seq(.1, .2, length = 5),
                 .NumVars = c(1:8))
set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)
```

```
pldaFit <- train(x=trainPredictors,
                 y =trainClasses,
                 method = "sparseLDA",
                 importance = TRUE,
                 tuneGrid = ldaGrid,
                 preProc = c("center", "scale"),
                 metric = "Kappa",
                 trControl = ctrl)

pldaFit
plot(pldaFit)

predictedPLDA <-  predict(pldaFit, testPredictors)
confusionMatrix(data = predictedPLDA,
                reference = as.factor(testClasses))



#6. Nonlinear Discriminant Analysis
library(mda)
mdaGrid <- expand.grid(.subclasses = 20:35)

set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
mdaFit <- train(x=trainPredictors,
                y =trainClasses,
                method = "mda",
                tuneGrid = mdaGrid,
                metric = "Kappa",
                trControl = ctrl)

mdaFit
```

```
plot(mdaFit)

predictedMDA <-  predict(mdaFit, testPredictors)
confusionMatrix(data = predictedMDA,
              reference = as.factor(testClasses))



#7.Neural Networks
library(nnet)

nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (8 + 1) + (maxSize+1)*2) ## 8 is the number of predictors

set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
nnetFit <- train(x=trainPredictors,
               y =trainClasses,
               method = "nnet",
               metric = "Kappa",
               preProc = c("center", "scale"),
               tuneGrid = nnetGrid,
               trace = FALSE,
               maxit = 2000,
               MaxNWts = numWts,
               trControl = ctrl)
nnetFit
plot(nnetFit)

predictedNNET <-  predict(nnetFit, testPredictors)
confusionMatrix(data = predictedNNET,
          reference = as.factor(testClasses))


#8. Flexible Discriminant Analysis

fdaGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
fdaFit <- train(x=trainPredictors,
          y =trainClasses,
          method = "fda",
          tuneGrid = fdaGrid,
          metric = "Kappa",
          trControl = ctrl)

fdaFit
plot(fdaFit)

predictedFDA <-  predict(fdaFit, testPredictors)
confusionMatrix(data = predictedFDA,
          reference = as.factor(testClasses))



#9. Support Vector Machines

 sigmaRangeReduced <- sigest(as.matrix(trainPredictors[,1:8]))

 svmGrid<- expand.grid(.sigma = sigmaRangeReduced[1],
                       .C = 2^(seq(0, 10)))
 set.seed(980)
 ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

 set.seed(980)
 svmFit2 <- train(x=trainPredictors,
              y =as.factor(trainClasses),
              method = "svmRadial",
              tuneGrid = svmGrid,
              preProc = c("center", "scale"),
              metric = "Kappa",
              trControl = ctrl)

svmFit2
 plot(svmFit2)

predictedSVM2<-  predict(svmFit2, testPredictors)
 confusionMatrix(data = predictedSVM,
             reference = as.factor(testClasses))



SVMvar2 <-  varImp(svmFit2)
SVMvar2



plot(SVMvar2)



#10. K-Nearest Neighbors ##
knnGrid<- data.frame(.k = 1:50)

set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)
```

```
set.seed(980)
knnFit <- train(x=trainPredictors,
            y =trainClasses,
            method = "knn",
            metric = "Kappa",
            preProc = c("center", "scale"),
            tuneGrid = knnGrid,
            trControl = ctrl)
knnFit
plot(knnFit)

predictedKNN<-  predict(knnFit, testPredictors)
confusionMatrix(data = predictedKNN,
            reference = as.factor(testClasses))


#11 Naive Bayes

nbGrid<- data.frame(.fL = 2,.usekernel = TRUE,.adjust = TRUE)

set.seed(980)
ctrl <- trainControl(method = "cv", number = 10, summaryFunction = defaultSummary)

set.seed(980)
nbFit <- train(x=trainPredictors,
            y =trainClasses,
            method = "nb",
            metric = "Kappa",
            tuneGrid = nbGrid,
            trControl = ctrl)
nbFit

predictedNB<-  predict(nbFit, testPredictors)
```

```
confusionMatrix(data = predictedNB,
          reference = as.factor(testClasses))
```