Machine Learning Classification algorithms

Performance of classification models on election data

Karnam Yogesh, Yashraj Varma, SriRajitha, Kishan R, Akash Gupta

Preface

The sheer scale of the Indian General Elections, as well the unpredictability and excitement surrounding it, has motivated our group to study various factors affecting its outcomes. We have attempted to apply various classification models to study the Indian Election data, primarily in order to arrive at a fairly accurate predictive model that might tell us the election outcome for a particular candidate. We have seen from our analyses that, rather surprisingly, the Logit model helps us predict quite accurately whether a given candidate will win a seat in their constituency. The lowest test error rate was observed to be around 9%, which gives us a degree of confidence about the correctness of our model and predictions. In the subsequent sections, we will provide a detailed description of the dataset, the algorithms used and inferences.

Data Preparation

The dataset contains information about 2264 candidates across 543 constituencies with respect to the 2019 Lok Sabha elections. Information about the candidates was compiled by 'Prakrut Chauhan' from **myneta.info** portal, augmented from the Election commission database. Upon receiving this data, we found at that it required a rather tedious amount of cleaning and preparation before being fit for algorithmic inputs. These are some of the steps followed by us :

- **Missing and non-functional values** :- There were certain fields in which appropriate values were not given and were showing up as **NA**. In addition, we had instances of **NOTA** in some constituencies. There were many other case where fields were blank or had arbitrary symbols in them which indeed made it quite the task to clean this dataset. We ultimately dealt with the missing values by removing the corresponding rows since no clear way of estimating the respective values could be ascertained. Also, given that the number of our observations were quite large, losing them would not make a huge difference.
- **Removing certain attributes** :- The original dataset contained attributes wherein information regarding **Party Symbol**, **name**, **postal votes** was given. We felt that these would be useful and hence, discarded them.

- **Converting values to numeric type** :- Probably one of the biggest challenges here was convert the supposedly numerical values into actual numeric data types. For example, categories like **Value of Assets (in Rupees)** had numbers in them, but not in a numeric format. Therefore, various functions were applied across the dataset in **MS Excel** to convert these values into a numeric data type so that mathematical operations could be performed on them.
- **Forming levels/categories of qualitative variables** :- Another arduous task. We had attributes like **State** which mentioned the state that particular observation/candidate belonged to. Now when there are 29 states, it is not practical or feasible to form those many dummy variables! This is where we thought of a rather pragmatic approach to handling this. As per the wiki link - RStudio we successfully grouped all the states as per their **Geographical Regions** and replaced the appropriate 'state' entries with corresponding 'region' entries. Similarly, categories were created for **Education level of candidate** and **Party** as well. For example, instead of dealing with 40 different parties, we have made a broad categorization of **BJP, INC and OTHERS**.
- **Dealing with Percentages** :- There were instances of total votes cast, total votes cast by general population. These attributes seemed to be doubling up for essentially one count - **proportion of votes the candidate attained in his constituency**. So we effectively replaced these counts with two main attributes - **Fraction of votes gotten out of total votes** and **Total voters in constituency**.
- **Normalization of Data** :- We learned from various sources including our coursebook that for classification algorithms to be effective and error free, data should ideally be converted to a standard normal form. Essentially every column was converted to a standard normal distribution by applying the appropriate calculations. In the end all the numeric attributes had mean 0 and unit variance.
- **Splitting into Train and Test** :- As a final step, our final data was split into training and testing datasets.

Final Data Attributes

Before we proceed to our analyses and its ultimate purpose, we would take a moment to explain the various attributes/features of this dataset after the cleaning and preparation process had been completed.

- **Region** :- [Categorical predictor] Contains 5 levels as per geographical location of the appropriate *state*. This has 6 levels - **North, South, Northeast, Central, East, West**.
- **Party** :- [Categorical predictor] Contains 3 levels based on the party that the candidate belongs to - **bjp, inc, others**. The 'Others' category contains the various local/regional parties like - TDP, DMK, JDU, etc.
- **Gender** :- [Categorical predictor] Contains 2 levels based on the gender of the candidate

- **male, female**.

- **Criminal** :- [Numerical predictor] Contains the number of criminal cases against a particular candidate. **Trivia** - *Candidate having almost 200 criminal cases won his seat!*
- **Age** :- [Numerical predictor] Contains the numerical value of age of the particular candidate.
- **Category** :- [Categorical predictor] Contains 3 level based on the caste status of the particular candidate - **sc, st, general**.
- **Education** :- [Categorical predictor] Contains 4 broad categories based on highest level of education achieved by a candidate - **School, College, Doctorate, Others**.
- **Assets** :- [Numerical predictor] Contains a numerical value in **Rupees**, the value of the Assets owned by the candidate. For example - Rs. 3,56,492.
- **Debts** :- [Numerical predictor] Contains a numerical value in **Rupees**, the value of the Liabilities owed by the candidate. For example - Rs. 3,56,492.
- **Fraction** :- [Numerical predictor] Depicts the fraction of total votes gathered by the candidate out of total voters in their constituency.
- **Totalvotes** :- [Numerical predictor] Total number of electors in the constituency.
- **Winner** :- [Categorical Response] Whether a candidate won or lost the election - **yes, no**.
- ***NOTE*** :- All values are standardized to it might make sense to decipher them as per their face value.

Aim of our analyses

The prime motive here is to predict whether or not a given candidate will win his Lok Sabha constituency seat, given the requisite information about him. We shall now fit various classification models on our traiing data and then predict the response classes based on the test data.

Setting up R and dummy coding qualitative variables

```
electfile = read.csv("/Users/Akashgupta/Desktop/ML proj/Finaldataset_normalized.csv")
fix(electfile)
names(electfile)
```

```
##  [1] "Region"     "Winner"     "Party"      "Gender"     "Criminal"
##  [6] "Age"        "Category"   "Education"  "Assets"     "Debts"
## [11] "Fraction"   "Totalvotes"
```

```r
attach(electfile)
# relevel is to custom choose your own base class - class that is 0 under all cases
Gender <- relevel(Gender, ref = "FEMALE")  # FEMALE : base class - 0
Region <- relevel(Region, ref = "Central") # Central : base class - 0
Party <- relevel(Party, ref = "OTHERS") # OTHERS : base class - 0
Category <- relevel(Category, ref = "GENERAL")  # GENERAL : base class - 0
Education <- relevel(Education, ref = "Others") # Others : base class - 0
Winner<- relevel(Winner, ref = "NO") # NO : base class - 0
set.seed(2000)
elect_set <- sample(x = 1:nrow(electfile), size = round(nrow(electfile)*0.50), replace =
e_train <- electfile[elect_set, ]
e_test<- electfile[-elect_set, ]
x = model.matrix(Winner~.,data = electfile)
y = as.factor(electfile$Winner)
x_tr = x[elect_set,]
y_tr = y[elect_set]
x_te = x[-elect_set,]
y_te = y[-elect_set]
```

```r
head(e_train)
```

```
##        Region Winner  Party Gender Criminal    Age Category Education
## 597  Northern     NO    INC   MALE   -0.192 -0.199  GENERAL   College
## 1896 Northern     NO OTHERS   MALE   -0.192  0.813       SC   College
## 1202  Central     NO OTHERS   MALE   -0.192  0.391       ST    School
## 1054  Eastern    YES OTHERS   MALE    0.069  1.487  GENERAL   College
## 53   Northern     NO OTHERS   MALE   -0.192  0.223       SC    School
## 1423  Eastern     NO OTHERS FEMALE   -0.192 -0.452       SC    School
##       Assets  Debts Fraction Totalvotes
## 597  -0.200 -0.222   -0.189      1.192
## 1896 -0.251 -0.222   -1.126      0.077
## 1202 -0.285 -0.219   -1.023      0.914
## 1054 -0.174 -0.222    1.015     -0.685
## 53   -0.230 -0.221   -0.843      0.607
## 1423 -0.292 -0.194   -1.000      0.694
```

div style="margin-bottom:50px;">

Logit model

We will not attempt to fit the logit model using the glm function. Within this section we will be fitting a glm model using the **Ridge regression** technique, the **Lasso regression** technique and **K-folds** cross validation. We would then compare the results of the different types of model fit on the criteria of **test error rate** and **ROC curves** obtained from each method. It turns out through this analysis that the Lasso gives out a more accurate model.

The Graph given below tells us the best model as per the Lasso Regression. We can see that for a **lambda** value of 4 our model is the best performing one.

```
x1 = model.matrix(Winner~Region+Party+Gender+Criminal+Age+Category+Education+
                    Assets+Debts+Fraction+Totalvotes, data = e_train)
y1 = e_train$Winner
x2 = model.matrix(Winner~Region+Party+Gender+Criminal+Age+Category+Education+
                     Assets+Debts+Fraction+Totalvotes, data = e_test)
y2 = e_test$Winner

library(glmnet)
```

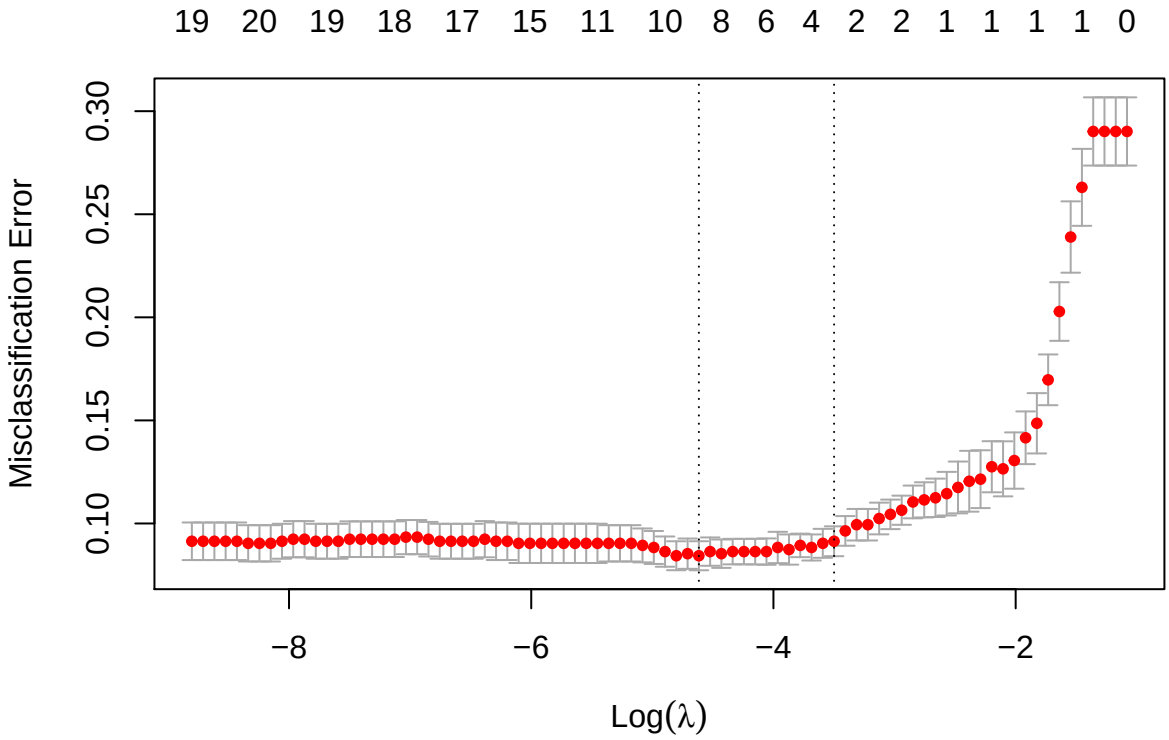
```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

```
fit = glmnet(x1, y1, family = "binomial")
cvfit = cv.glmnet(x1, y1, family = "binomial", type.measure = "class")
coef(cvfit, s = "lambda.min")
```

```
## 22 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)         -2.52925039
## (Intercept)          .
## RegionEastern        .
## RegionNorth Eastern  .
## RegionNorthern       0.68390470
## RegionSouthern       .
## RegionWestern        .
## PartyINC            -1.15510352
## PartyOTHERS         -0.21037377
## GenderMALE           .
## Criminal             0.24167434
## Age                  .
## CategorySC           .
## CategoryST           .
## EducationCollege     .
## EducationDoctorate   .
## EducationOthers      .
## EducationSchool      .
## Assets               0.03018030
## Debts                0.03691479
## Fraction             3.37528068
## Totalvotes           0.15643366
```

```
plot(cvfit)
```



The **test error rate** for the GLM model with lasso and k-folds cross validation is as follows :-

```
mean(preds123 != y2)
```

```
## [1] 0.09447236
```

The test error rate using **Ridge** on the GLM logit has now been computed. As we can see the test error rate using lasso was marginally better since it effectively elimates some variables.

```
grid=10^seq(10,-2,length=100)
y1 = ifelse(y=="YES",1,0)
ridge.mod2=glmnet(x_tr,y_tr,alpha=0,lambda=grid,family = "binomial")
cv.out1=cv.glmnet(x_tr,y_tr,alpha=0,family = "binomial")
ridge.pred1=predict(ridge.mod2,s=cv.out1$lambda.min,newx=x_te,type = "class")
ridge.pred2=predict(ridge.mod2,s=cv.out1$lambda.min,newx=x_te,type = "response")
mean(ridge.pred1 != y_te)
```

```
## [1] 0.09547739
```

```
mean(ridge.pred1 != y_te)
```

```
## [1] 0.09547739
```

We can now compare the **ROC curves** of both the techniques to get a clear visual representation of model performance.

```
library(ROCR)
```

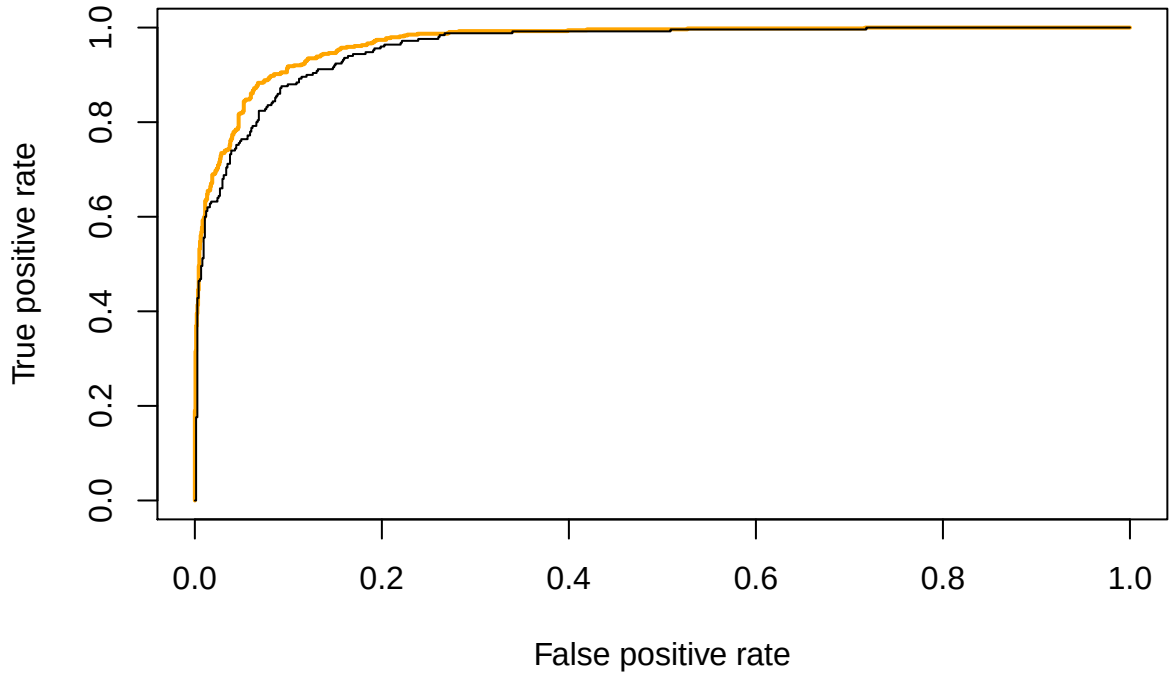
```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```
pred1 = prediction(glm.fit666$fitted.values, Winner)
pred11 = prediction(ridge.pred2, y_te)
roc22 = performance(pred11, measure="tpr",x.measure="fpr")
roc2 = performance(pred1, measure="tpr",x.measure="fpr")
```

```
plot(roc2,col="orange", lwd=2)
plot(roc22, add=TRUE)
```



LDA model

Now we shall move on to the **linear discriminant analysis** classifier. We will compare two versions of this model - the lda model with and without **Leave out one cross validation**. Our best lda model will be the one with the lowest **test error rate**. error rate for the lda model with cross validation is given as :-

```
library(MASS)
#####-------CV lda---------#########
ldamod1 <- lda(Winner~.,data = e_test,CV = TRUE)
ldamod2 <- lda(Winner~.,data = e_train, CV = FALSE)
predlda = predict(ldamod2, e_test, type = "class")
```

```
mean(ldamod1$class != e_test$Winner)
```

```
## [1] 0.1075377
```

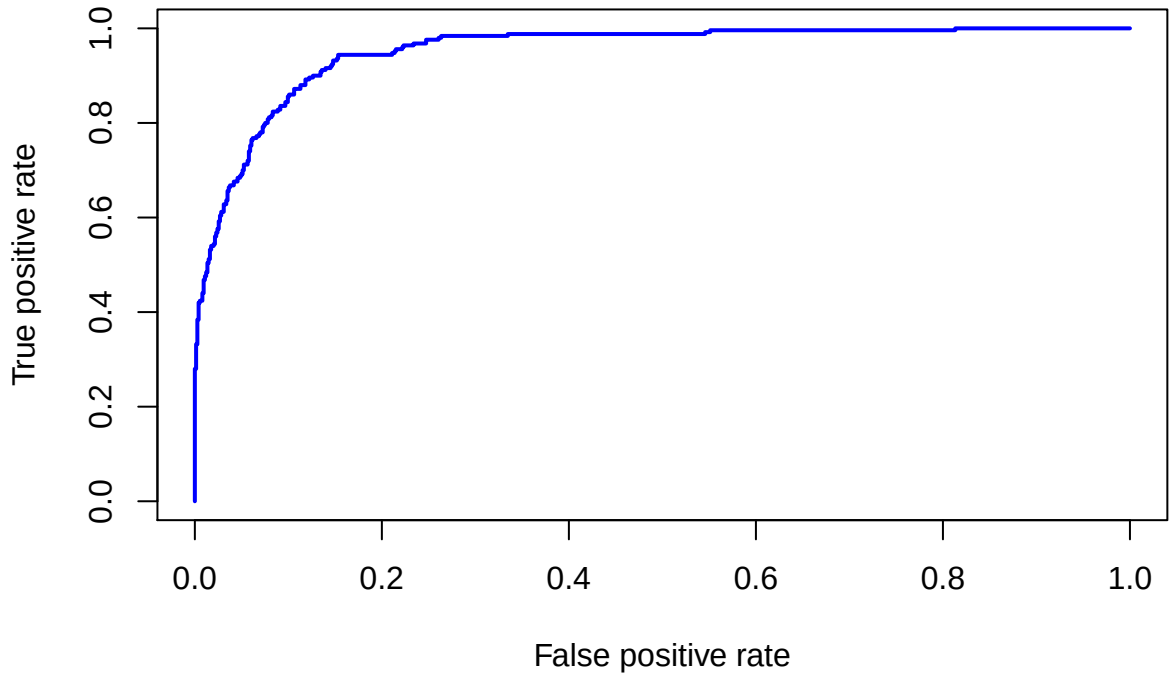
Again, it is clearly seen that the **test error rate** computed by the model after performing cross validation is better and hence we would select this model. Finally, the ROC curve is plotted.

```
mean(predlda$class != e_test$Winner)
```

```
## [1] 0.118593
```

We can now compute the **ROC curve** for the final lda model as follows :-

```
plot(roc3,col="blue", lwd=2)
```



A brief look into Bootstrapping :-

We are aware of the fact that bootstrapping helps us to estimate the true standard errors of our model. It deos so by taking repeated samples out of the original data. In this demonstration we have shown how bootstrapping estimates the standard errors of the coefficients of our **GLM logit regression** by repeatedly sampling 1000 times with replacement.

```
library(boot)
names(electfile)
```

```
##  [1] "Region"     "Winner"     "Party"      "Gender"     "Criminal"
##  [6] "Age"        "Category"   "Education"  "Assets"     "Debts"
```

```
## [11] "Fraction"   "Totalvotes"
```

```
boot.fn=function(data,index)
  return(coef(glm(Winner~Fraction+Assets+Party+Region+Gender+Age+Totalvotes+Category+Cri

set.seed(1)
boot.fn(electfile,sample(1991,1991,replace=T))
```

```
##         (Intercept)            Fraction              Assets
##         -3.72220374          4.48786190          0.04063082
##            PartyINC         PartyOTHERS       RegionEastern
##         -1.73560292         -0.78154377          0.60172655
## RegionNorth Eastern      RegionNorthern      RegionSouthern
##          0.09121284          1.61536405          0.48652046
##       RegionWestern          GenderMALE                 Age
##          0.48166276         -0.24060386         -0.08420534
##          Totalvotes          CategorySC          CategoryST
##          0.08568215         -0.03159250          0.72996981
##            Criminal               Debts
##          0.04918964          0.12198323
```

```
boot(electfile,boot.fn ,R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = electfile, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##         original       bias    std. error
## t1*  -3.71833661 -0.101643029  0.57455480
## t2*   4.44519136  0.128528026  0.35823816
## t3*   0.03333389  0.002719927  0.09830624
## t4*  -1.53787616 -0.037112606  0.27188772
## t5*  -0.99305820 -0.012578558  0.22276629
## t6*   0.59596027  0.003248728  0.42332066
## t7*  -0.07956020  0.013288320  0.62714681
## t8*   1.51562490  0.035360346  0.39324239
## t9*   0.50277835 -0.009070719  0.41335691
## t10*  0.69825876  0.025366786  0.55518645
## t11*  0.08227062 -0.011967583  0.25507507
## t12* -0.03896513  0.005118363  0.10593964
## t13*  0.17949352 -0.001125645  0.13016687
```

```
## t14* -0.19333002  0.011848047  0.24267972
## t15* -0.17872630  0.005493717  0.31901765
## t16*  0.04609457  0.083266891  0.18968944
## t17*  0.02767393  0.003214895  0.08214852
```

```
newglmmodel = glm(Winner~Fraction+Assets+Party+Region+Gender+Age+Totalvotes+Category+Cri
summary(newglmmodel)$coefficients
```

```
##                          Estimate Std. Error    z value     Pr(>|z|)
## (Intercept)         -3.71833661 0.57858714 -6.4265802 1.305066e-10
## Fraction             4.44519136 0.28268175 15.7250738 1.018320e-55
## Assets               0.03333389 0.08963390  0.3718893 7.099753e-01
## PartyINC            -1.53787616 0.26869124 -5.7235813 1.043016e-08
## PartyOTHERS         -0.99305820 0.21656249 -4.5855504 4.527921e-06
## RegionEastern        0.59596027 0.45673471  1.3048281 1.919514e-01
## RegionNorth Eastern -0.07956020 0.61016784 -0.1303907 8.962573e-01
## RegionNorthern       1.51562490 0.45124992  3.3587261 7.830263e-04
## RegionSouthern       0.50277835 0.46234794  1.0874458 2.768398e-01
## RegionWestern        0.69825876 0.68879761  1.0137357 3.107089e-01
## GenderMALE           0.08227062 0.26642001  0.3088005 7.574733e-01
## Age                 -0.03896513 0.09802158 -0.3975158 6.909871e-01
## Totalvotes           0.17949352 0.10585870  1.6955953 8.996254e-02
## CategorySC          -0.19333002 0.24432008 -0.7912981 4.287701e-01
## CategoryST          -0.17872630 0.31255004 -0.5718326 5.674354e-01
## Criminal             0.04609457 0.06303039  0.7313071 4.645916e-01
## Debts                0.02767393 0.09653422  0.2866748 7.743613e-01
```

The above table corresponds to **bootstrapped** coefficient standard errors and the table below shows the standard errors resulting from a regular glm fit. We can compare the two estimates. The bootstrapped version usually gives us more accurate results regarding the degree of uncertainty in our model.

Decision Trees and Boosting models

Here we shall attempt to fit a **pruned** decision tree after performing k-fold cross valiation. We will then compare the fit of the pruned tree with that of the **boosted** decision tree, wherein we will fit multiple layers of trees such that the model learns slowly and reduces our error rate. The decision tree fit and error rate are given below :-

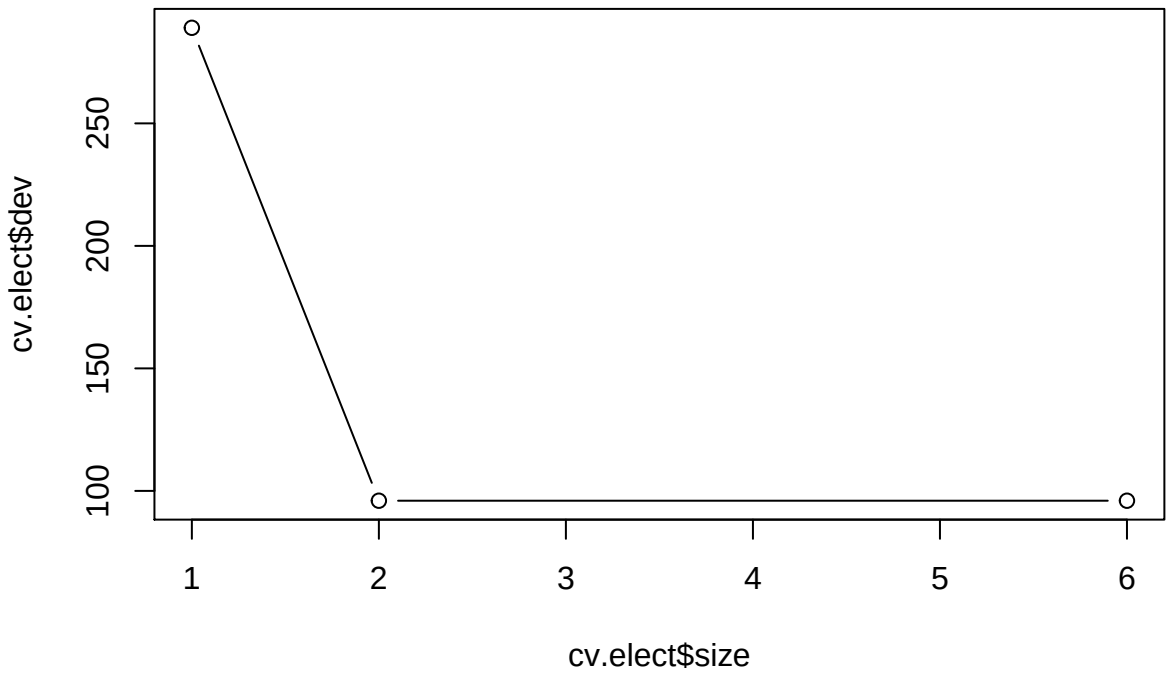
```
library(tree)
tree.elect =tree(Winner~Region+Party+Gender+Criminal+Age+Category+Education+
                   Assets+Debts+Fraction+Totalvotes, e_train)
## predicting with tree ##
tree.pred=predict(tree.elect,e_test,type="class")
## confusion matrix ##
table(tree.pred ,e_test$Winner)
```

```
##
## tree.pred  NO YES
##       NO  656  26
##       YES  89 224
```

```
## test error rate ##
mean(tree.pred != e_test$Winner)
```

```
## [1] 0.1155779
```

```
## cross validation ##
cv.elect =cv.tree(tree.elect ,FUN=prune.misclass)
plot(cv.elect$size ,cv.elect$dev ,type="b")
```




```
## pruning
prune.elect=prune.misclass(tree.elect,best=6)
summary(prune.elect)
```

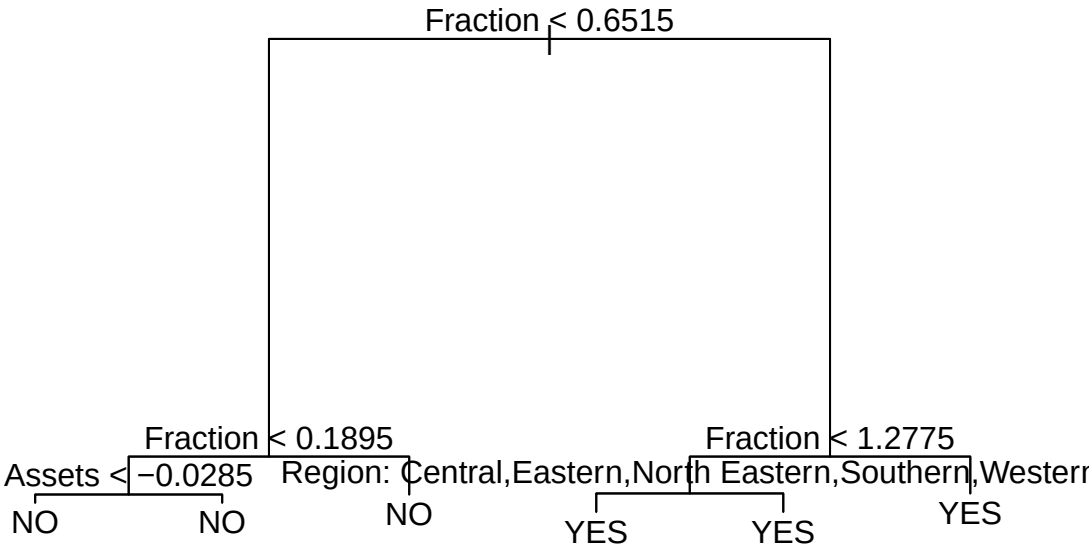
```
##
## Classification tree:
## tree(formula = Winner ~ Region + Party + Gender + Criminal +
##     Age + Category + Education + Assets + Debts + Fraction +
##     Totalvotes, data = e_train)
## Variables actually used in tree construction:
## [1] "Fraction" "Assets"   "Region"
## Number of terminal nodes:  6
## Residual mean deviance:  0.3997 = 395.7 / 990
## Misclassification error rate: 0.09438 = 94 / 996
```

```
## predicting from cross validated, pruned tree ##
tree.pred1=predict(prune.elect,e_test,type="class")
## confusion ##
table(tree.pred1 ,e_test$Winner)
```

```
##
## tree.pred1  NO YES
##        NO  656  26
##        YES  89 224
```

```
## test error ##
```

```
plot(prune.elect)
text(prune.elect,pretty=0)
```




```
mean(tree.pred1 != e_test$Winner)
```

```
## [1] 0.1155779
```

We shall now have a look at how boosting reduces our error rate thereby resolving to a more accurate model. We can see that the error rate of prediction is an improvement over the decision tree model given in the previous section.

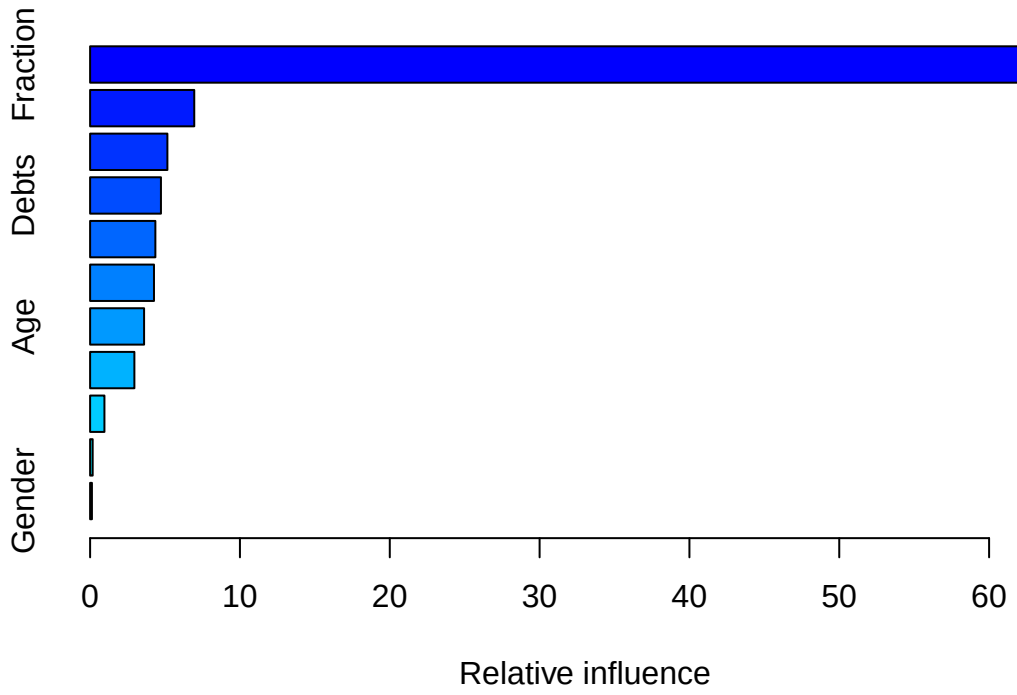
```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
boost.elect=gbm((unclass(Winner)-1)~Region+Party+Gender+Criminal+Age+Category+Education+
                  "bernoulli",n.trees=5000, interaction.depth=4)

yhat.elect11=predict(boost.elect,newdata=e_test, n.trees=5000, type = "response")
predbinaries <- as.factor(ifelse(yhat.elect11>0.5,"YES","NO"))
```

```r
summary(boost.elect)
```



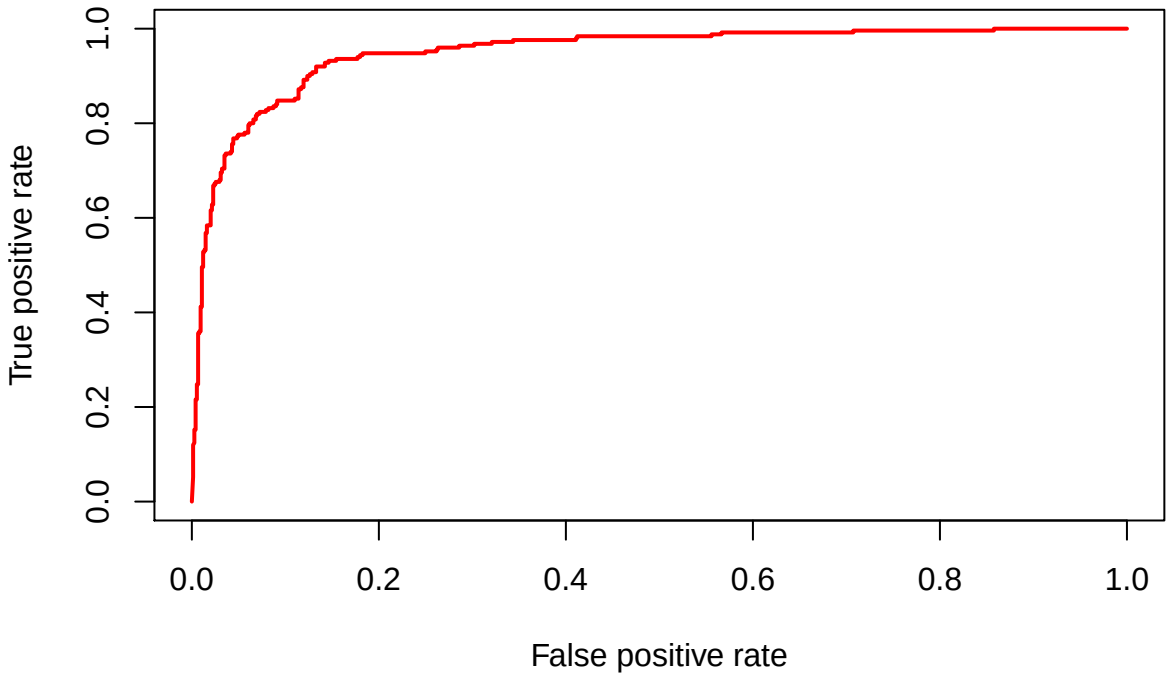

```
##                   var    rel.inf
## Fraction     Fraction 66.7409310
## Totalvotes Totalvotes  6.9507774
## Assets         Assets  5.1589513
## Debts           Debts  4.7282220
## Region         Region  4.3549247
## Party           Party  4.2636329
## Age               Age  3.6020443
## Criminal     Criminal  2.9541069
## Education   Education  0.9552134
## Category     Category  0.1703002
## Gender         Gender  0.1208958
```

```r
mean(predbinaries!=e_test$Winner)
```

```
## [1] 0.09849246
```

It is clearly seen from the diagram that boosting has identified the most important features for us. It appears as thought the attribute **Fraction** accounts for most of the variation in the model and that is the most significant internal node where the trees are performing splits. Consequently, even **Assets** and **Party** seem to be important features about a candidate that have an effect on whether he will win the election or not. We can plot the ROC curve now as well.

```r
plot(roc4,col="red", lwd=2)
```

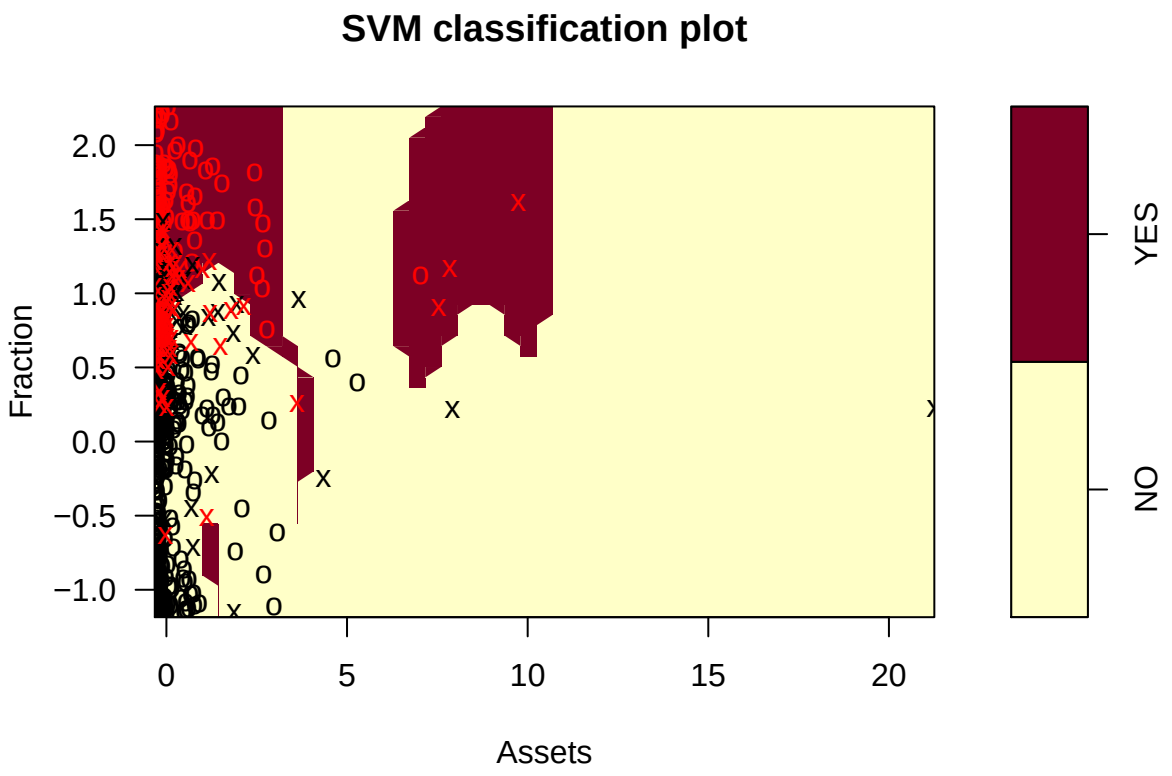


SVM

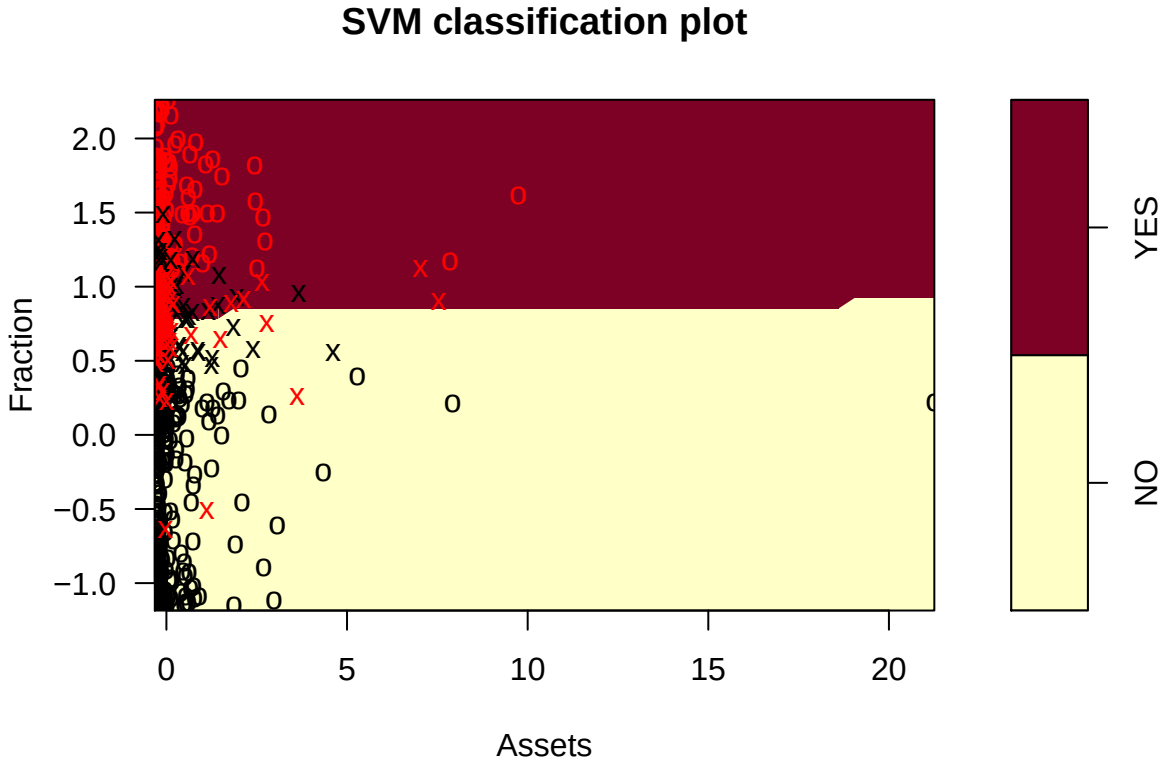
Now we come to our final model. The SVM is computed using a **linear** kernel first, which essentially makes it into a **support vector classifier**. After this we will compute SVM using the **radial** kernel. It is noted that the linear kernel gives us a higher error rate than the radial one, albeit not with a significant difference. The plots of the two classifiers are shown below for given values of **gamma** and **cost**.

```
library(e1071)
###--new data consiting of only three columns - two predictors - one response--##
newdata1 = data.frame(Winner,Fraction,Assets)
train1 = sample(1991, 996)

###----svm works with numeric values, so recoded response----###
y1 = newdata1[train1,"Winner"]
y1 <- ifelse(y1 == "YES",1,0)
x1 <- as.matrix(newdata1[train1,-which(names(newdata1)=="Winner")])
#####--------fitting svm on training data---############
sv.fit <- svm(Winner~Fraction+Assets,data=newdata1[train1,],
              kernel="radial",gamma=1, cost=100)
plot(sv.fit,newdata1[train1,])
```




```
sv.fit3 <- svm(Winner~Fraction+Assets,data=newdata1[train1,],
               kernel="linear",gamma=1, cost=100)
plot(sv.fit3,newdata1[train1,])
```



```r
#####-----tuning svm with CV to get best model----#########
tune.out=tune(svm, Winner~Fraction+Assets, data=newdata1[train1,], kernel="radial",
              ranges=list(cost=c(0.1,1,10,100,1000),
                          gamma=c(0.5,1,2,3,4) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     2
##
## - best performance: 0.09033333
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.09838384 0.03459881
## 2  1e+00   0.5 0.09333333 0.03855694
## 3  1e+01   0.5 0.09433333 0.03709263
## 4  1e+02   0.5 0.09232323 0.03666087
## 5  1e+03   0.5 0.09534343 0.03710830
## 6  1e-01   1.0 0.09736364 0.03527294
## 7  1e+00   1.0 0.09432323 0.03884369
## 8  1e+01   1.0 0.09133333 0.03747678
## 9  1e+02   1.0 0.09633333 0.03911441
## 10 1e+03   1.0 0.09936364 0.04009314
## 11 1e-01   2.0 0.09836364 0.03673389
## 12 1e+00   2.0 0.09033333 0.03554251
## 13 1e+01   2.0 0.09233333 0.03846513
## 14 1e+02   2.0 0.09635354 0.03772963
## 15 1e+03   2.0 0.10240404 0.03615797
## 16 1e-01   3.0 0.10137374 0.03811496
## 17 1e+00   3.0 0.09034343 0.03770326
## 18 1e+01   3.0 0.09636364 0.04005407
## 19 1e+02   3.0 0.09736364 0.03768492
## 20 1e+03   3.0 0.09736364 0.03588485
## 21 1e-01   4.0 0.10339394 0.03499731
## 22 1e+00   4.0 0.09034343 0.03857723
## 23 1e+01   4.0 0.09735354 0.03677960
## 24 1e+02   4.0 0.09738384 0.03592450
## 25 1e+03   4.0 0.10139394 0.03382093
```

```
tune.out1=tune(svm, Winner~Fraction+Assets, data=newdata1[train1,], kernel="linear",
               ranges=list(cost=c(0.1,1,10,100,1000),
                           gamma=c(0.5,1,2,3,4) ))
summary(tune.out1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.1044242
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.1044242 0.03517327
## 2  1e+00   0.5 0.1054141 0.03818482
## 3  1e+01   0.5 0.1054141 0.03818482
## 4  1e+02   0.5 0.1054141 0.03818482
## 5  1e+03   0.5 0.1054141 0.03818482
## 6  1e-01   1.0 0.1044242 0.03517327
## 7  1e+00   1.0 0.1054141 0.03818482
## 8  1e+01   1.0 0.1054141 0.03818482
## 9  1e+02   1.0 0.1054141 0.03818482
## 10 1e+03   1.0 0.1054141 0.03818482
## 11 1e-01   2.0 0.1044242 0.03517327
## 12 1e+00   2.0 0.1054141 0.03818482
## 13 1e+01   2.0 0.1054141 0.03818482
## 14 1e+02   2.0 0.1054141 0.03818482
## 15 1e+03   2.0 0.1054141 0.03818482
## 16 1e-01   3.0 0.1044242 0.03517327
## 17 1e+00   3.0 0.1054141 0.03818482
## 18 1e+01   3.0 0.1054141 0.03818482
## 19 1e+02   3.0 0.1054141 0.03818482
## 20 1e+03   3.0 0.1054141 0.03818482
## 21 1e-01   4.0 0.1044242 0.03517327
## 22 1e+00   4.0 0.1054141 0.03818482
## 23 1e+01   4.0 0.1054141 0.03818482
## 24 1e+02   4.0 0.1054141 0.03818482
## 25 1e+03   4.0 0.1054141 0.03818482
```

```
######---predicting with best model and creating confusion matrix----####
preds=predict(tune.out$best.model,newdata = newdata1[-train1,])
table(preds,newdata1[-train1,"Winner"])
```

```
##
## preds  NO YES
##   NO  643  47
##   YES  78 227
```

```
mean(preds != newdata1[-train1,"Winner"])
```

```
## [1] 0.1256281
```

```
preds11=predict(tune.out1$best.model,newdata = newdata1[-train1,])
table(preds11,newdata1[-train1,"Winner"])
```

```
##
## preds11  NO YES
##     NO  656  45
##     YES  65 229
```

```
mean(preds11 != newdata1[-train1,"Winner"])
```

```
## [1] 0.1105528
```

```
#####------fitting best svm on whole data------#########
sv.fit1 <- svm(Winner~Fraction+Assets,data=newdata1,
               kernel="radial",gamma=2, cost=1, decision.values = T)
sv.fit2 <- svm(Winner~Fraction+Assets,data=newdata1,
               kernel="linear",gamma=0.5, cost=1, decision.values = T)

fitted=attributes(predict(sv.fit1,newdata1,
                          decision.values=TRUE))$decision.values

fitted1=attributes(predict(sv.fit2,newdata1,
                          decision.values=TRUE))$decision.values

####-----alternate code for ROC---USE THIS-----$$$$$$$
library(ROCR)
pred0 = prediction(fitted1,Winner)
roc0 = performance(pred0, measure="tpr", x.measure="fpr")
pred = prediction(fitted,Winner)
roc9 = performance(pred, measure="tpr", x.measure="fpr")
```

We will now take models and tune them by checking the model performance under various values of cost and lambda such that the best model for both **linear** as well as **radial** kernels can be computed. After that, we can compare their error rates. It turns out that the **radial**

classifier performed better that the linear one. The error rates are compared below :-

```
mean(preds != newdata1[-train1,"Winner"])
```

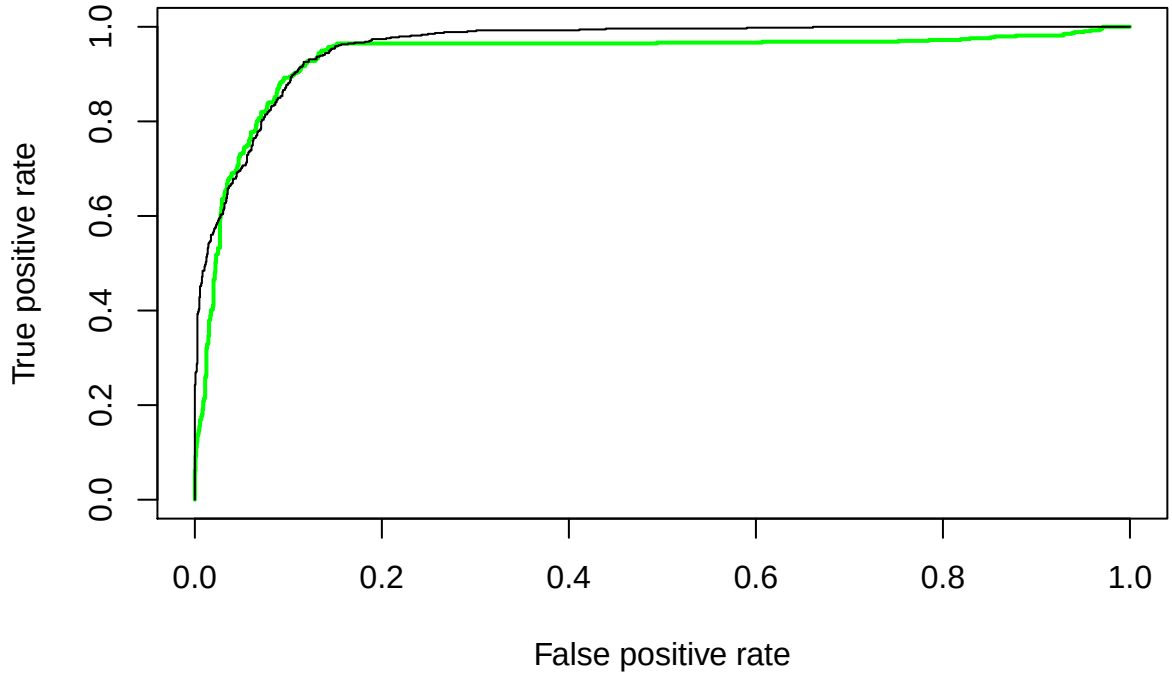
```
## [1] 0.1256281
```

```
mean(preds11 != newdata1[-train1,"Winner"])
```

```
## [1] 0.1105528
```

The **ROC curves** for both the classifiers are shown below. We can see that they are quite close to one another indicating a very similiar ratio of **sensitivity** and **specificity**. They are shown below :-

```
plot(roc9,col="green", lwd=2)
plot(roc0,add=TRUE)
```



Comparing all the models

Area under ROC curve for GLM with lasso :-

```
auc(rr)
```

```
## Area under the curve: 0.9696
```

Area under ROC curve for LDA with CV :-

```
auc(rr3)
```

```
## Area under the curve: 0.9522
```

Area under ROC curve for Decision Tree with Boosting :-

```
auc(rr5)
```

```
## Area under the curve: 0.9497
```

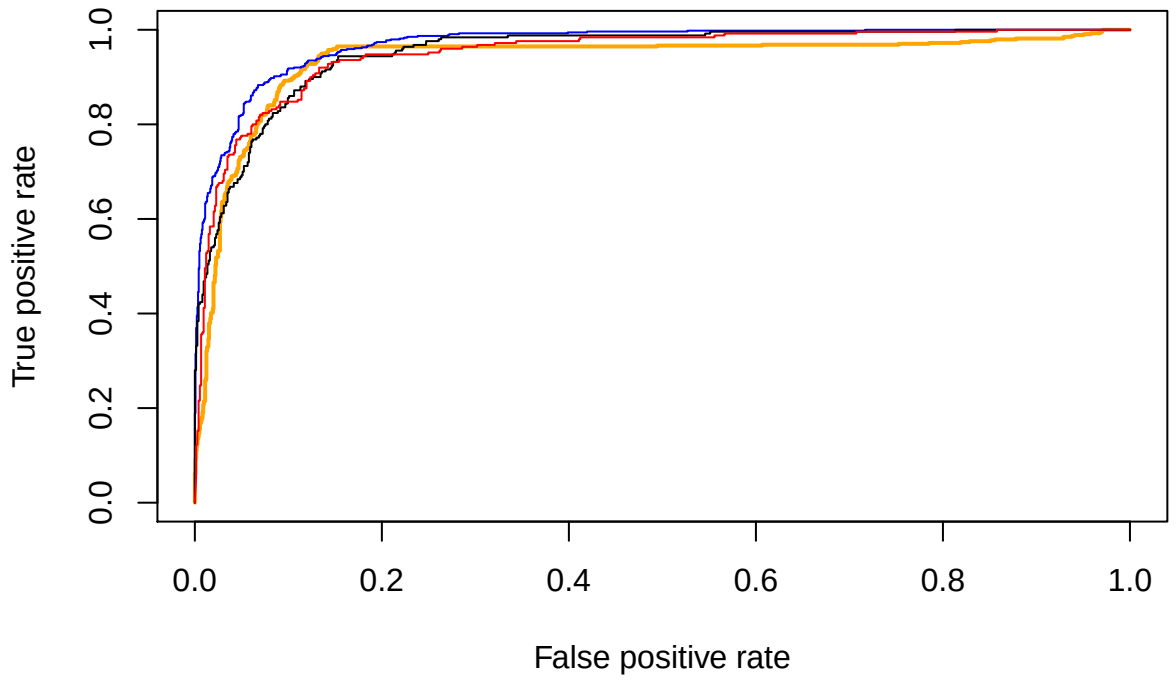
Area under ROC curve for SVM with radial kernel :-

```
auc(rr6)
```

```
## Area under the curve: 0.9363
```

As a final step to getting closer to our best model out of all the classification models we have applied so far, we can, in addition to looking at the **AUC** for the various models - the area under the curve of the ROC curves - we can also look at all the ROC plots together to see which plot **hugs the top left corner the most**.

```
plot(roc9,col="orange", lwd=2) #svm
plot(roc2, col = "blue",add=TRUE) #logit
plot(roc3,add = TRUE) #lda
plot(roc4,col="red", add=TRUE) #dtboost
```



From the above results we can conclude that it is infact, rather surprisingly, the **Logit model** with **Lasso and 10-folds** that turned out to be the best model with an **AUC** of almost **97%** and a **ROC** curve that hugs the left corner most tightly out of all the curves. LDA and boosted decision trees are a close second. We can hence conclude that when it comes to predicting **election results**, the **Logit** model with lasso turns out to be the most accurate in classifying potential candidates as Winners or losers.