Project3 R code

Xiangyu Zeng

January 15, 2018

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.3
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.3
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.4.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.4.3
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.4.3
```

```
##
## Attaching package: 'gplots'
```

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

```
library(DMwR)
```

```
## Warning: package 'DMwR' was built under R version 3.4.3
```

```
## Loading required package: grid
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.4.3
```

```
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 3.4.3
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.3
```

```
library(rpart)
library(randomForest)
library(ggplot2)
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.4.3
```

```
## Rattle: A free graphical interface for data science with R.
## XXXX 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.4.3
```

```
library(RColorBrewer)
library(ROSE)
```

```
## Warning: package 'ROSE' was built under R version 3.4.3
```

```
## Loaded ROSE 0.0-3
```

```
df = fread("creditcard.csv")
```

```
##
Read 21.1% of 284807 rows
Read 38.6% of 284807 rows
Read 56.2% of 284807 rows
Read 73.7% of 284807 rows
Read 87.8% of 284807 rows
Read 98.3% of 284807 rows
Read 284807 rows and 31 (of 31) columns from 0.140 GB file in 00:00:09
```

```
names(df)
```

```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"
##  [8] "V7"     "V8"     "V9"     "V10"    "V11"    "V12"    "V13"
## [15] "V14"    "V15"    "V16"    "V17"    "V18"    "V19"    "V20"
## [22] "V21"    "V22"    "V23"    "V24"    "V25"    "V26"    "V27"
## [29] "V28"    "Amount" "Class"
```

```
head(df)
```

```
##    Time         V1          V2        V3         V4          V5
## 1:    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077
## 2:    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765
## 3:    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813
## 4:    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888
## 5:    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338
## 6:    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688
##            V6          V7          V8         V9         V10        V11
## 1:  0.46238778  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995
## 2: -0.08236081 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267
## 3:  1.80049938  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015
## 4:  1.24720317  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873
## 5:  0.09592146  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429
## 6: -0.02972755  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620
##           V12        V13        V14        V15        V16         V17
## 1: -0.61780086 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124
## 2:  1.06523531  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466
## 3:  0.06608369  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938
## 4:  0.17822823  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279
## 5:  0.53819555  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324
## 6:  0.35989384 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282
##           V18         V19         V20          V21          V22
## 1:  0.02579058  0.40399296  0.25141210 -0.018306778  0.277837576
## 2: -0.18336127 -0.14578304 -0.06908314 -0.225775248 -0.638671953
## 3: -0.12135931 -2.26185710  0.52497973  0.247998153  0.771679402
## 4:  1.96577500 -1.23262197 -0.20803778 -0.108300452  0.005273597
## 5: -0.03819479  0.80348692  0.40854236 -0.009430697  0.798278495
## 6:  0.06865315 -0.03319379  0.08496767 -0.208253515 -0.559824796
##           V23         V24        V25        V26          V27         V28
## 1: -0.11047391  0.06692807  0.1285394 -0.1891148  0.133558377 -0.02105305
## 2:  0.10128802 -0.33984648  0.1671704  0.1258945 -0.008983099  0.01472417
## 3:  0.90941226 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184
## 4: -0.19032052 -1.17557533  0.6473760 -0.2219288  0.062722849  0.06145763
## 5: -0.13745808  0.14126698 -0.2060096  0.5022922  0.219422230  0.21515315
## 6: -0.02639767 -0.37142658 -0.2327938  0.1059148  0.253844225  0.08108026
##    Amount Class
## 1: 149.62     0
## 2:   2.69     0
## 3: 378.66     0
## 4: 123.50     0
## 5:  69.99     0
## 6:   3.67     0
```

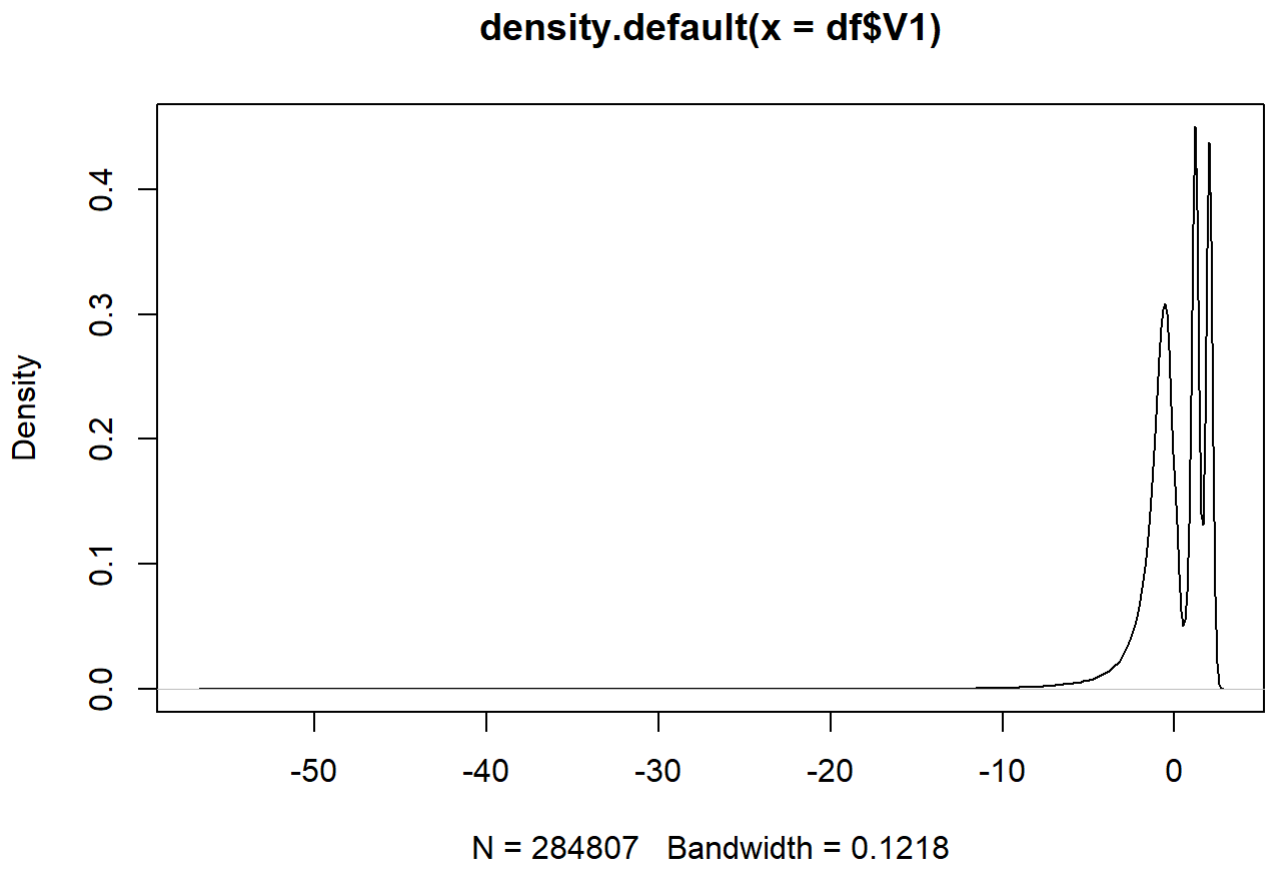
```
summary(df)
```

```
##       Time              V1                  V2
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855
##  Median : 84692   Median :  0.01811   Median :  0.06549
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773
##        V3                 V4                 V5
##  Min.   :-48.3256   Min.   :-5.68317   Min.   :-113.74331
##  1st Qu.: -0.8904   1st Qu.:-0.84864   1st Qu.:  -0.69160
##  Median :  0.1799   Median :-0.01985   Median :  -0.05434
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :   0.00000
##  3rd Qu.:  1.0272   3rd Qu.: 0.74334   3rd Qu.:   0.61193
##  Max.   :  9.3826   Max.   :16.87534   Max.   :  34.80167
##        V6                 V7                 V8
##  Min.   :-26.1605   Min.   :-43.5572   Min.   :-73.21672
##  1st Qu.: -0.7683   1st Qu.: -0.5541   1st Qu.: -0.20863
##  Median : -0.2742   Median :  0.0401   Median :  0.02236
##  Mean   :  0.0000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.:  0.3986   3rd Qu.:  0.5704   3rd Qu.:  0.32735
##  Max.   : 73.3016   Max.   :120.5895   Max.   : 20.00721
##        V9                 V10                V11
##  Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                V13                V14
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256
##  Median :  0.1400   Median :-0.01357   Median :  0.0506
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268
##       V15                V16                V17
##  Min.   :-4.49894   Min.   :-14.12985   Min.   :-25.16280
##  1st Qu.:-0.58288   1st Qu.: -0.46804   1st Qu.: -0.48375
##  Median : 0.04807   Median :  0.06641   Median : -0.06568
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.64882   3rd Qu.:  0.52330   3rd Qu.:  0.39968
##  Max.   : 8.87774   Max.   : 17.31511   Max.   :  9.25353
##       V18                V19                V20
##  Min.   :-9.498746   Min.   :-7.213527   Min.   :-54.49772
##  1st Qu.:-0.498850   1st Qu.:-0.456299   1st Qu.: -0.21172
##  Median :-0.003636   Median : 0.003735   Median : -0.06248
##  Mean   : 0.000000   Mean   : 0.000000   Mean   :  0.00000
##  3rd Qu.: 0.500807   3rd Qu.: 0.458949   3rd Qu.:  0.13304
##  Max.   : 5.041069   Max.   : 5.591971   Max.   : 39.42090
##       V21                V22                 V23
##  Min.   :-34.83038   Min.   :-10.933144   Min.   :-44.80774
##  1st Qu.: -0.22839   1st Qu.: -0.542350   1st Qu.: -0.16185
##  Median : -0.02945   Median :  0.006782   Median : -0.01119
```

```
##  Mean   :  0.00000   Mean   :  0.000000   Mean   :  0.00000
##  3rd Qu.:  0.18638   3rd Qu.:  0.528554   3rd Qu.:  0.14764
##  Max.   : 27.20284   Max.   : 10.503090   Max.   : 22.52841
##       V24                V25                 V26
##  Min.   :-2.83663   Min.   :-10.29540   Min.   :-2.60455
##  1st Qu.:-0.35459   1st Qu.: -0.31715   1st Qu.:-0.32698
##  Median : 0.04098   Median :  0.01659   Median :-0.05214
##  Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.: 0.43953   3rd Qu.:  0.35072   3rd Qu.: 0.24095
##  Max.   : 4.58455   Max.   :  7.51959   Max.   : 3.51735
##       V27                  V28                Amount
##  Min.   :-22.565679   Min.   :-15.43008   Min.   :    0.00
##  1st Qu.: -0.070840   1st Qu.: -0.05296   1st Qu.:    5.60
##  Median :  0.001342   Median :  0.01124   Median :   22.00
##  Mean   :  0.000000   Mean   :  0.00000   Mean   :   88.35
##  3rd Qu.:  0.091045   3rd Qu.:  0.07828   3rd Qu.:   77.17
##  Max.   : 31.612198   Max.   : 33.84781   Max.   :25691.16
##     Class
##  Length:284807
##  Class :character
##  Mode  :character
##
##
##
```

```
set.seed(1003)
```

```
# Look at distribution
plot(density(x= df$V1))
```

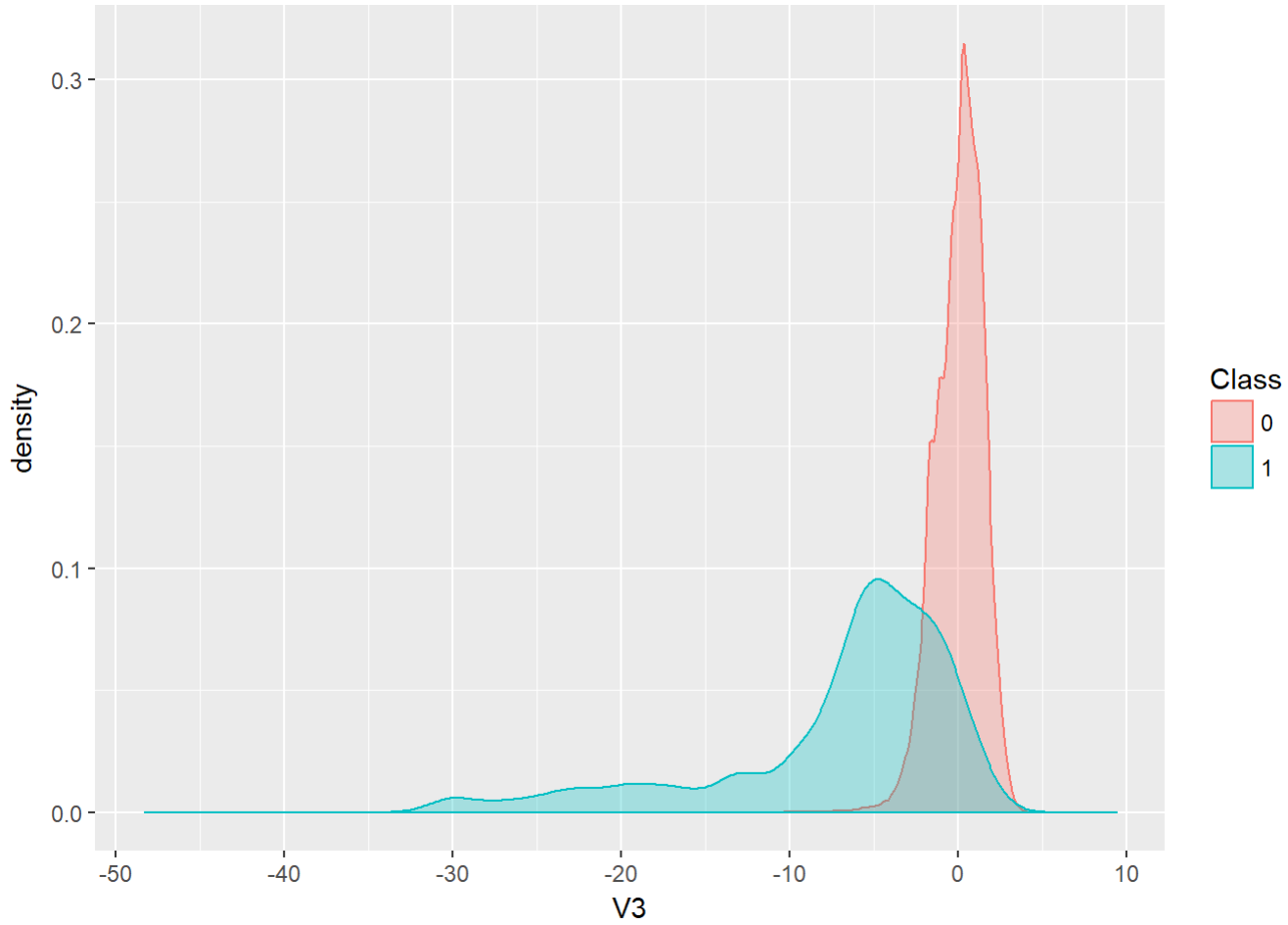



```
#look at relationship with y level

#look at errors
#find missing values
sum(is.na(df))
```

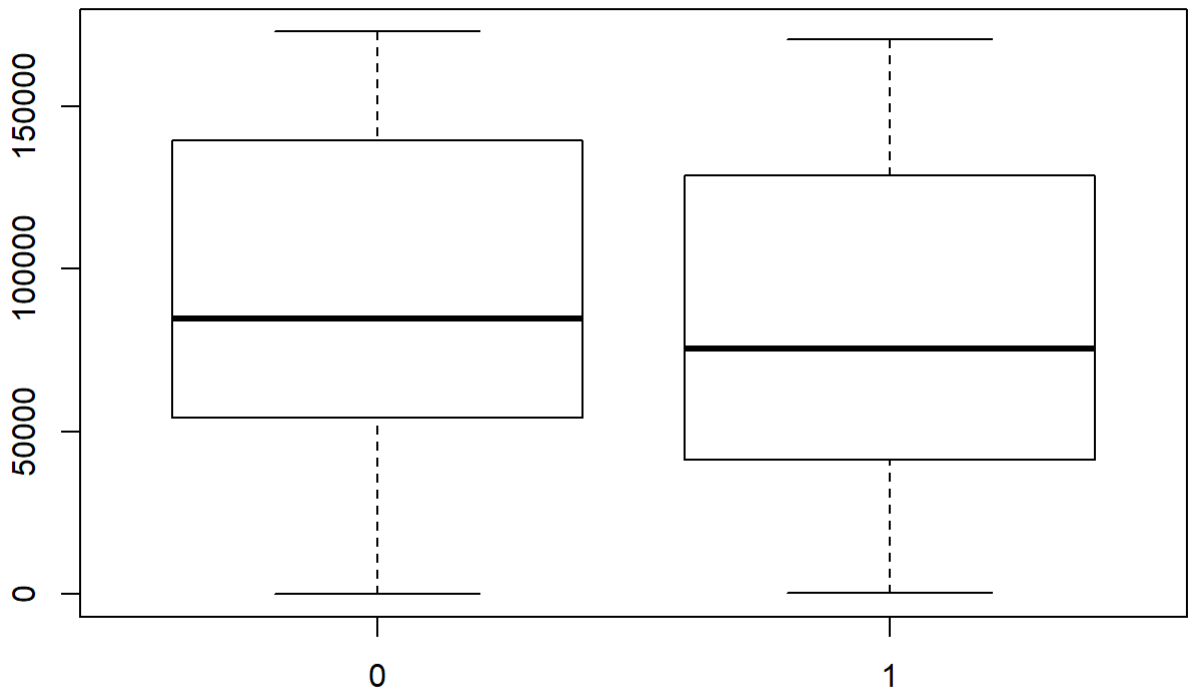
```
## [1] 0
```

```
#prepossessing: deal with it later
#class0: fruad
#class1:not fruad

#PCA is already done/ find there is no missing values
#save steps in dealing with data set
```

```
#Exploatory data analysis
ggplot(df, aes(x=V3)) + geom_density(aes(group=Class, colour=Class, fill=Class), alpha=0.3)
```




```
#Boxplot
boxplot(df$Time~df$Class)
```



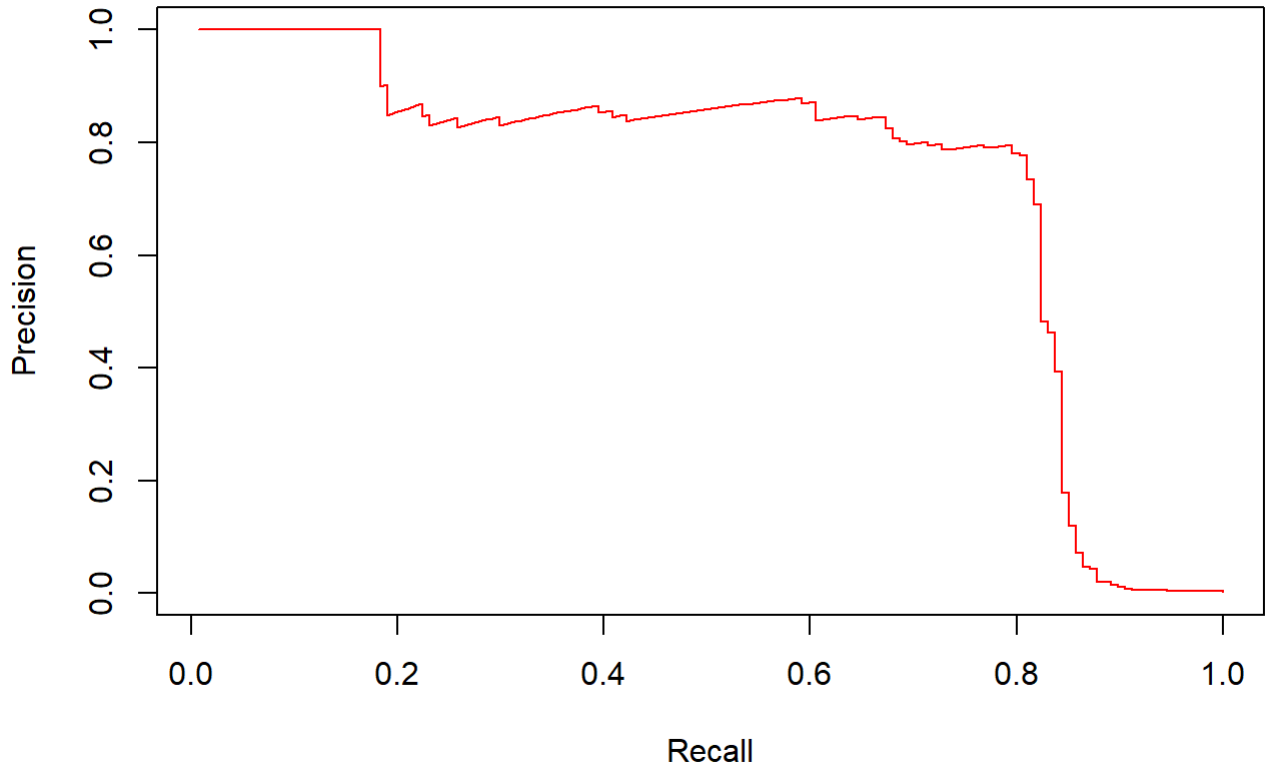

```
#because data is relatively clean, decide to run robust models
#also need to perform some transformations.
#Log is not as good as boxcox

#from the plot can see the data is heavity skewed to the left:
#decide to perform a box-cox transformation

#### Data pre-processing
## 'normalize' the data
transform_columns <- c("V","Amount")
transformed_column     <- df[ ,grepl(paste(transform_columns, collapse = "|"),names(df)),with =
FALSE]
transformed_column_processed <- predict(preProcess(transformed_column, method = c("BoxCox","scal
e")),transformed_column)
```

```
## Warning in is.na(lam): is.na() applied to non-(list or vector) of type
## 'NULL'
```

```
#Create the new dataframe
df_new <- data.table(cbind(transformed_column_processed,Class = df$Class))
df_new[,Class:=as.factor(Class)]
set.seed(1003)
```

```
#### Training and Test dataset
training_index <- createDataPartition(df_new$Class, p=0.7,list=FALSE)
training <- df_new[training_index,]
test<- df_new[-training_index,]
```

```
### Logistic regression
logit <- glm(Class ~ ., data = training, family = "binomial")
logit_pred <- predict(logit, test, type = "response")

logit_prediction <- prediction(logit_pred,test$Class)
logit_recall <- performance(logit_prediction,"prec","rec")
logit_roc <- performance(logit_prediction,"tpr","fpr")
logit_auc <- performance(logit_prediction,"auc")
plot(logit_recall,col='red')
```




```
logit_auc
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.9609662
##
##
## Slot "alpha.values":
## list()
```

```
### Random forest (Too long to get the running results)
#rf.model <- randomForest(Class ~ ., data = training,ntree = 2000, nodesize = 20)
#rf_pred <- predict(rf.model, test,type="prob")

#rf_prediction <- prediction(rf_pred[,2],test$Class)
#rf_recall <- performance(rf_prediction,"prec","rec")
#rf_roc <- performance(rf_prediction,"tpr","fpr")
#rf_auc <- performance(rf_prediction,"auc")
#plot(rf_recall, add = TRUE, col = 'blue')
```

```
#auprc <- function(pr_curve) {
 #x <- as.numeric(unlist(pr_curve@x.values))
 #y <- as.numeric(unlist(pr_curve@y.values))
 #y[is.nan(y)] <- 1
 #id <- order(x)
 #result <- sum(diff(x[id])*rollmean(y[id],2))
 #return(result)
#}

#bagging tree
#auprc_results <- data.frame(logit=auprc(logit_recall)
 #                          , rf = auprc(rf_recall)
  #                         , tb = auprc(tb_recall))

#ctrl <- trainControl(method = "cv", number = 10)

#tb_model <- train(Class ~ ., data = train_smote, method = "treebag",
   #              trControl = ctrl)

#tb_pred <- predict(tb_model$finalModel, test, type = "prob")

#tb_prediction <- prediction(tb_pred[,2],test$Class)
#tb_recall <- performance(logit_prediction,"prec","rec")
#tb_roc <- performance(logit_prediction,"tpr","fpr")
#tb_auc <- performance(logit_prediction,"auc")
#plot(tb_recall, add = TRUE, col = 'green')
```

```
## naive Bayes
df$Class <- factor(df$Class, levels = c("1", "0"))
set.seed(1234)
dataSplit <- sample(2, nrow(df), replace = TRUE, prob = c(0.7, 0.3))
trainSplit<- df[dataSplit==1,]
testSplit <- df[dataSplit==2,]

library(e1071)
# create a simple naive bayes model
nb.model <- naiveBayes(Class ~ ., data = trainSplit)

# make predictions - test data
nb.pred <- predict(nb.model, testSplit, type = "class")

# create a naive bayes confusion matrix
table(nb.pred, testSplit$Class)
```

```
##
## nb.pred     1     0
##       1   130  1877
##       0    20 83293
```

```
# performance metrics
confusionMatrix(nb.pred, testSplit$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     1     0
##          1   130  1877
##          0    20 83293
##
##                Accuracy : 0.9778
##                  95% CI : (0.9768, 0.9787)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1177
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.866667
##             Specificity : 0.977962
##          Pos Pred Value : 0.064773
##          Neg Pred Value : 0.999760
##              Prevalence : 0.001758
##          Detection Rate : 0.001524
##    Detection Prevalence : 0.023523
##       Balanced Accuracy : 0.922314
##
##        'Positive' Class : 1
##
```

```
# data balancing
# used SMOTE to generate additional new positive class observations and attained an almost
# equal split -  1968 (47%) and otherwise 2214 (53%)
# I chose k=5, but I think there are better approaches to chosing k
# so that the model does not overfit during learning

set.seed(1234)
new.data <- SMOTE(Class ~ ., df, perc.over = 300, perc.under=150, k = 5)
table(new.data$Class)
```

```
##
##    1    0
## 1968 2214
```

```
prop.table(table(new.data$Class))
```

```
##
##         1         0
## 0.4705882 0.5294118
```

```
# randomly split the data
set.seed(1234)
bal <- sample(2, nrow(new.data), replace = TRUE, prob = c(0.8, 0.2))
bal.train <- new.data[bal==1,]
bal.test <- new.data[bal==2,]

# retained almost similar split prob in both test/train sets as the original data
dim(bal.train)
```

```
## [1] 3344   31
```

```
dim(bal.test)
```

```
## [1] 838  31
```

```
prop.table(table(bal.train$Class))
```

```
##
##         1         0
## 0.4635167 0.5364833
```

```
prop.table(table(bal.test$Class))
```

```
##
##         1         0
## 0.4988067 0.5011933
```

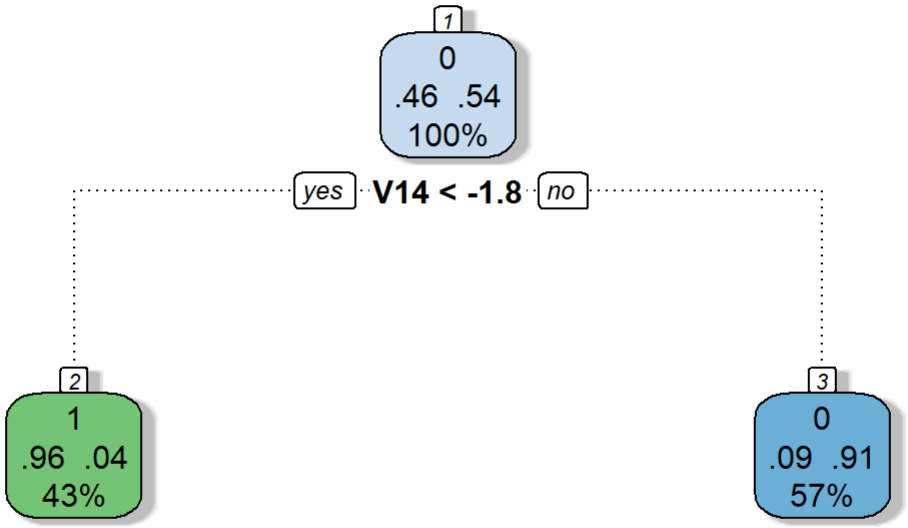
```
# decision tree

# fit the tree on balanced training set and validate with test
# I also use 10-fold cross validation to compare results

# Base Accuracy 93.4%
# AUC ROC Curve = 93.5%
# 10-fold Cross Validation Accuracy = 94.5%

set.seed(529)
bal.tree <- rpart(Class ~ ., data = bal.train)
# summary
summary(bal.tree)
```

```
## Call:
## rpart(formula = Class ~ ., data = bal.train)
##   n= 3344
##
##         CP nsplit rel error    xerror        xstd
## 1 0.843871      0  1.000000 1.0000000 0.018604253
## 2 0.010000      1  0.156129 0.1567742 0.009684767
##
## Variable importance
## V14 V10 V11 V12 V17  V3
##  20  17  16  16  16  15
##
## Node number 1: 3344 observations,    complexity param=0.843871
##   predicted class=0  expected loss=0.4635167  P(node) =1
##     class counts:  1550  1794
##    probabilities: 0.464 0.536
##   left son=2 (1432 obs) right son=3 (1912 obs)
##   Primary splits:
##       V14 < -1.803944 to the left,  improve=1218.3580, (0 missing)
##       V10 < -1.877758 to the left,  improve=1135.5890, (0 missing)
##       V12 < -2.378749 to the left,  improve=1037.3310, (0 missing)
##       V11 < 1.779057  to the right, improve=1031.7730, (0 missing)
##       V4  < 1.623406  to the right, improve= 990.1044, (0 missing)
##   Surrogate splits:
##       V10 < -1.868591 to the left,  agree=0.942, adj=0.864, (0 split)
##       V11 < 1.779057  to the right, agree=0.920, adj=0.813, (0 split)
##       V12 < -2.565602 to the left,  agree=0.912, adj=0.795, (0 split)
##       V17 < -1.303995 to the left,  agree=0.908, adj=0.785, (0 split)
##       V3  < -2.27274  to the left,  agree=0.890, adj=0.742, (0 split)
##
## Node number 2: 1432 observations
##   predicted class=1  expected loss=0.04329609  P(node) =0.4282297
##     class counts:  1370    62
##    probabilities: 0.957 0.043
##
## Node number 3: 1912 observations
##   predicted class=0  expected loss=0.09414226  P(node) =0.5717703
##     class counts:   180  1732
##    probabilities: 0.094 0.906
```

```
# model performance on test data - class
pred.tree <- predict(bal.tree, bal.test, type = "class")

# performance metrics
confusionMatrix(pred.tree, bal.test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   0
##          1 369  10
##          0  49 410
##
##                Accuracy : 0.9296
##                  95% CI : (0.9101, 0.946)
##     No Information Rate : 0.5012
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8592
##  Mcnemar's Test P-Value : 7.53e-07
##
##             Sensitivity : 0.8828
##             Specificity : 0.9762
##          Pos Pred Value : 0.9736
##          Neg Pred Value : 0.8932
##              Prevalence : 0.4988
##          Detection Rate : 0.4403
##    Detection Prevalence : 0.4523
##       Balanced Accuracy : 0.9295
##
##        'Positive' Class : 1
##
```

```
fancyRpartPlot(bal.tree)
```



Rattle 2018-Jan-16 21:46:41 zengxiangyu

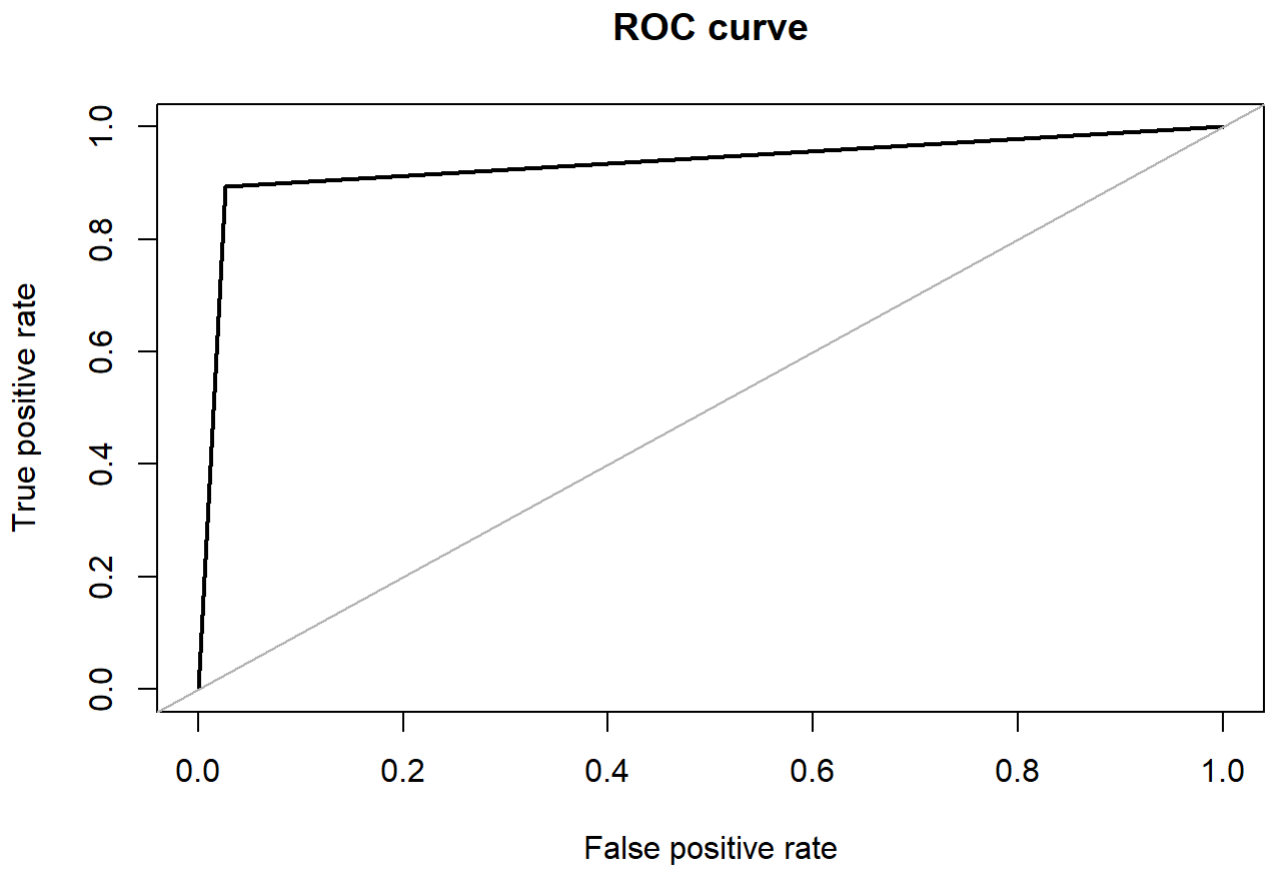
```
# variable importance
bal.tree$variable.importance
```

```
##       V14       V10       V11       V12       V17        V3
## 1218.3580 1052.4503  990.3413  969.0711  956.3089  904.4096
```

```
# from package ROSE we get precision/recall and f-measure
accuracy.meas(pred.tree, bal.test$Class)
```

```
##
## Call:
## accuracy.meas(response = pred.tree, predicted = bal.test$Class)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.548
## recall: 1.000
## F: 0.354
```

```
roc.curve(pred.tree, bal.test$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.933
```

```
# 10-fold cross validation
set.seed(529)
t.control <- trainControl(method = "cv", number = 10, savePredictions = TRUE)
cv.tree <- train(Class ~ ., data = new.data, trControl = t.control, method = "rpart", tuneLength
=5)
cv.tree.pred <- predict(cv.tree, new.data)
# confusion matrix
confusionMatrix(cv.tree.pred, new.data$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1 1795   79
##          0  173 2135
##
##                Accuracy : 0.9397
##                  95% CI : (0.9321, 0.9468)
##     No Information Rate : 0.5294
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8787
##  Mcnemar's Test P-Value : 4.672e-09
##
##             Sensitivity : 0.9121
##             Specificity : 0.9643
##          Pos Pred Value : 0.9578
##          Neg Pred Value : 0.9250
##              Prevalence : 0.4706
##          Detection Rate : 0.4292
##    Detection Prevalence : 0.4481
##       Balanced Accuracy : 0.9382
##
##        'Positive' Class : 1
##
```

```
# Random forest

set.seed(1234)
model.rf <- randomForest(Class ~ ., data = bal.train, ntree = 1000, importance = TRUE)
pred.rf <- predict(model.rf, bal.test, type = "class")
# confusion matrix
confusionMatrix(table(pred.rf, bal.test$Class))
```

```
## Confusion Matrix and Statistics
##
##
## pred.rf   1   0
##       1 403   0
##       0  15 420
##
##                Accuracy : 0.9821
##                  95% CI : (0.9706, 0.9899)
##     No Information Rate : 0.5012
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9642
##  Mcnemar's Test P-Value : 0.0003006
##
##             Sensitivity : 0.9641
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9655
##              Prevalence : 0.4988
##          Detection Rate : 0.4809
##    Detection Prevalence : 0.4809
##       Balanced Accuracy : 0.9821
##
##        'Positive' Class : 1
##
```

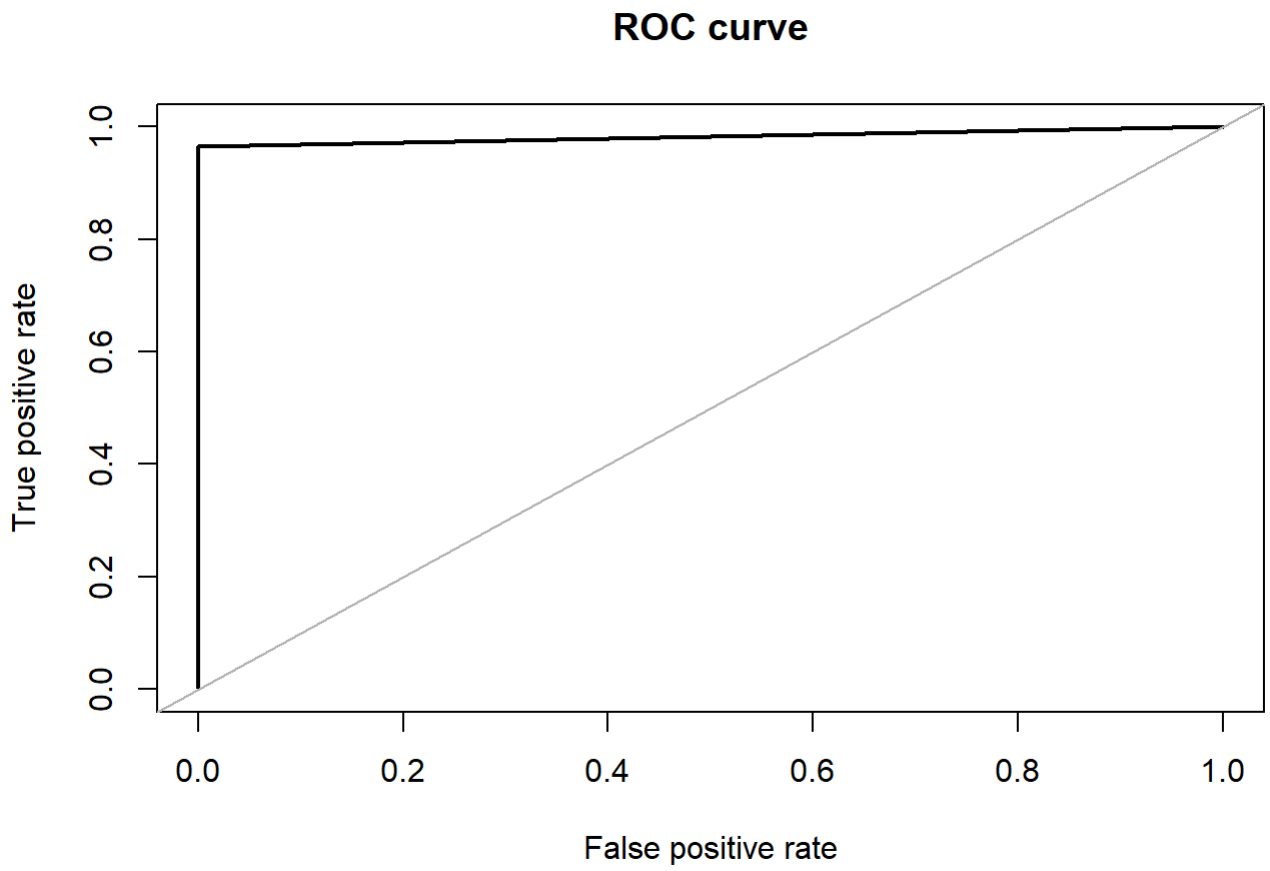
```
# variable importance
varImp(model.rf)
```

```
##               1        0
## Time   17.15128 17.15128
## V1     14.18302 14.18302
## V2     12.87521 12.87521
## V3     17.38823 17.38823
## V4     38.42271 38.42271
## V5     16.89217 16.89217
## V6     16.66281 16.66281
## V7     14.57964 14.57964
## V8     26.05350 26.05350
## V9     14.71085 14.71085
## V10    20.39057 20.39057
## V11    18.37038 18.37038
## V12    18.77494 18.77494
## V13    18.90429 18.90429
## V14    32.91277 32.91277
## V15    14.92389 14.92389
## V16    12.50612 12.50612
## V17    16.39265 16.39265
## V18    12.29486 12.29486
## V19    23.79910 23.79910
## V20    20.13913 20.13913
## V21    17.85923 17.85923
## V22    15.79072 15.79072
## V23    18.18132 18.18132
## V24    11.24040 11.24040
## V25    14.14559 14.14559
## V26    15.62040 15.62040
## V27    17.53227 17.53227
## V28    16.25277 16.25277
## Amount 23.44783 23.44783
```

```
# from package ROSE we get precision/recall and f-measure
accuracy.meas(pred.rf, bal.test$Class)
```

```
##
## Call:
## accuracy.meas(response = pred.rf, predicted = bal.test$Class)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.519
## recall: 1.000
## F: 0.342
```

```
roc.curve(pred.rf, bal.test$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.983
```