Credit Card Default

Table of Contents

ISLR packages provide several datasets. The description of datasets are available here: <http://cran.r-project.org/web/packages/ISLR/ISLR.pdf>

rm(list = ls()) # clear the workspace   
library(ISLR) # load ISLR data package  
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5 v purrr 0.3.4  
## v tibble 3.1.5 v dplyr 1.0.7  
## v tidyr 1.2.0 v stringr 1.4.0  
## v readr 2.0.2 v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(ggplot2)

# Data Preparation

Examine the dataset. It contains four variables, default, student,balance, and income.

Default<-as\_tibble(Default)  
Default

## # A tibble: 10,000 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No No 730. 44362.  
## 2 No Yes 817. 12106.  
## 3 No No 1074. 31767.  
## 4 No No 529. 35704.  
## 5 No No 786. 38463.  
## 6 No Yes 920. 7492.  
## 7 No No 826. 24905.  
## 8 No Yes 809. 17600.  
## 9 No No 1161. 37469.  
## 10 No No 0 29275.  
## # ... with 9,990 more rows

glimpse(Default)

## Rows: 10,000  
## Columns: 4  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, No, No, No~  
## $ student <fct> No, Yes, No, No, No, Yes, No, Yes, No, No, Yes, Yes, No, No, N~  
## $ balance <dbl> 729.5265, 817.1804, 1073.5492, 529.2506, 785.6559, 919.5885, 8~  
## $ income <dbl> 44361.625, 12106.135, 31767.139, 35704.494, 38463.496, 7491.55~

head(Default) # show the first six rows

## # A tibble: 6 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No No 730. 44362.  
## 2 No Yes 817. 12106.  
## 3 No No 1074. 31767.  
## 4 No No 529. 35704.  
## 5 No No 786. 38463.  
## 6 No Yes 920. 7492.

tail(Default) # show the last six rows

## # A tibble: 6 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No Yes 172. 14956.  
## 2 No No 712. 52992.  
## 3 No No 758. 19661.  
## 4 No No 845. 58636.  
## 5 No No 1569. 36669.  
## 6 No Yes 201. 16863.

names(Default) # variable names

## [1] "default" "student" "balance" "income"

nrow(Default) # the number of rows

## [1] 10000

ncol(Default) # the number of columns

## [1] 4

summary(Default) # basic summary statistics of the variables in Default dataset (default, student, balance, income)

## default student balance income   
## No :9667 No :7056 Min. : 0.0 Min. : 772   
## Yes: 333 Yes:2944 1st Qu.: 481.7 1st Qu.:21340   
## Median : 823.6 Median :34553   
## Mean : 835.4 Mean :33517   
## 3rd Qu.:1166.3 3rd Qu.:43808   
## Max. :2654.3 Max. :73554

# frequency table   
summary(Default$default) # summary of default variable

## No Yes   
## 9667 333

table(Default$default) # contingency table: frequency of each case (yes/no) in default variable

##   
## No Yes   
## 9667 333

table(Default$student) # contingency table: frequency of each case (yes/no) in student variable

##   
## No Yes   
## 7056 2944

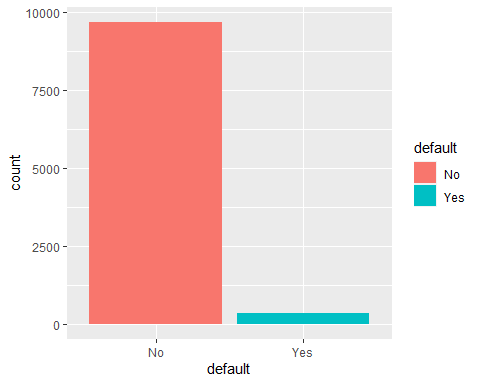
table(Default$default, Default$student) # cross-tabulation

##   
## No Yes  
## No 6850 2817  
## Yes 206 127

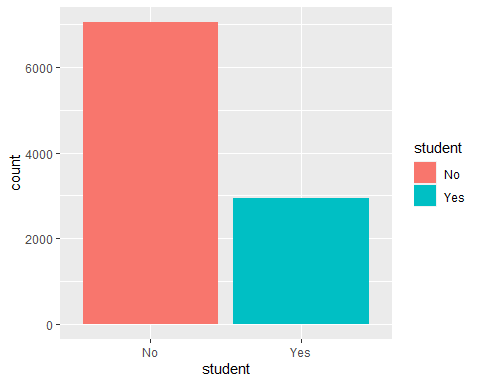
# Data Visualization

## Bar chart

Default %>%  
 ggplot(aes(x=default,fill=default)) +  
 geom\_bar()

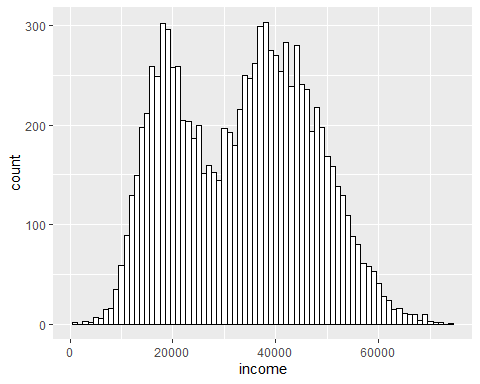


Default %>%  
 ggplot(aes(x=student,fill=student)) +  
 geom\_bar()

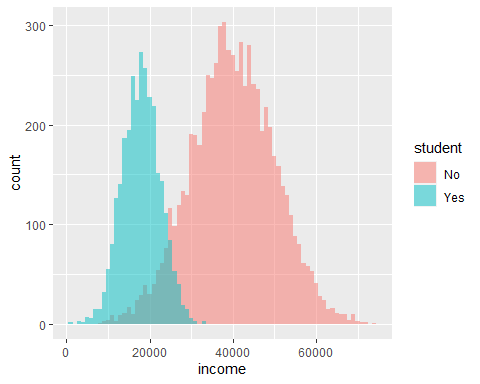


## Histograms

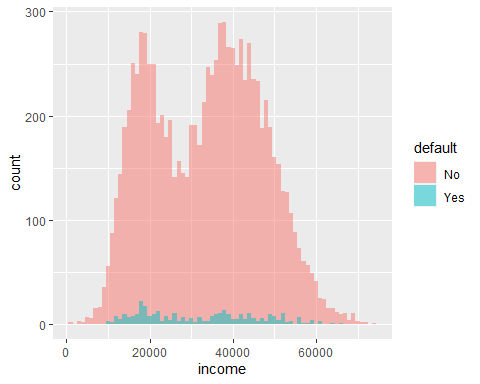
Default %>%  
 ggplot(aes(x=income)) +  
 geom\_histogram(binwidth=1000, colour="black",fill="white")



Default %>%  
 ggplot(aes(x=income,fill=student)) +  
 geom\_histogram(binwidth=1000,alpha=.5,position="identity")

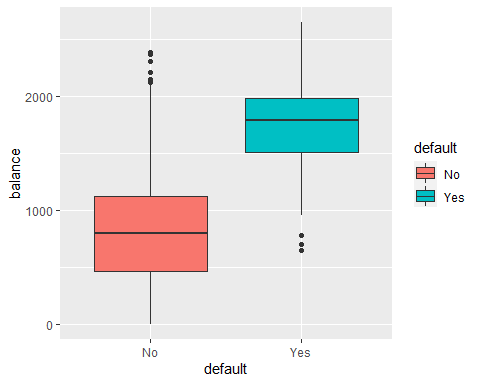


Default %>%  
 ggplot(aes(x=income,fill=default)) +  
 geom\_histogram(binwidth=1000,alpha=.5,position="identity")

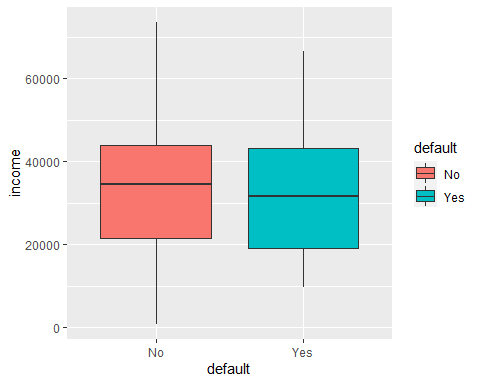


## Boxplots

ggplot(Default,aes(x=default,y=balance,fill=default))+geom\_boxplot()

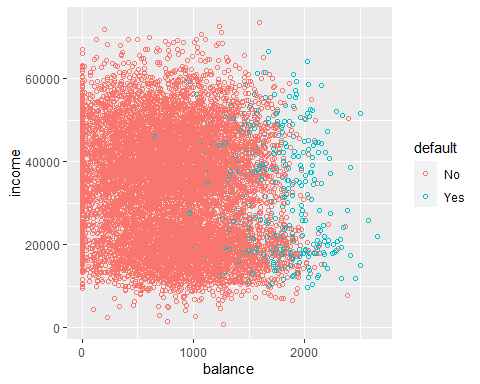


ggplot(Default,aes(x=default,y=income,fill=default))+geom\_boxplot()



## Scatter plots

Default %>%  
 ggplot(aes(x=balance,y=income,color=default)) +  
 geom\_point(shape=1)



We need *rpart* package for Classification Trees. More on rpart : <http://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf>

rpart.plot is a package for visualizing classification tree models.

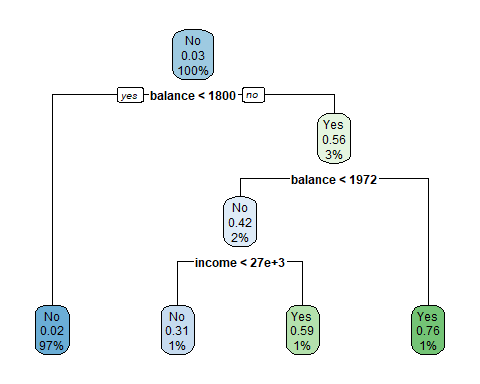
library(rpart)  
library(rpart.plot)

Let’s build a model.

ct\_model<-rpart(default~student+balance+income, # model formula  
 data=Default, # dataset  
 method="class", # "class" indicates a classification tree model   
 control=rpart.control(cp=0.03,maxdepth=4)) # tree control parameters.

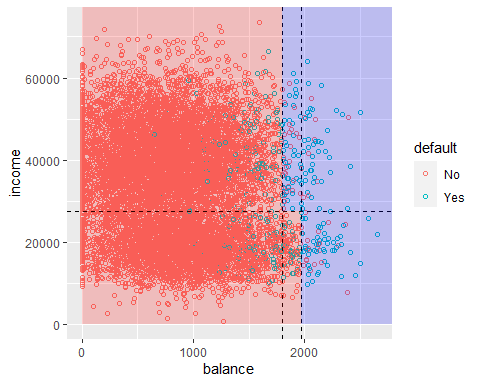
Next, let’s visualize the tree model.

rpart.plot(ct\_model) # tree plot



Here, I tried to visualize the classification tree model results on the scatterplot. It could be done, because the model used the two numeric variables.

Default %>%  
 ggplot(aes(x=balance,y=income,color=default)) +  
 geom\_point(shape=1)+  
 geom\_vline(xintercept=1800.002,linetype="dashed")+  
 geom\_vline(xintercept=1971.915,linetype="dashed")+  
 geom\_hline(yintercept=27401.2,linetype="dashed")+  
 annotate("rect",xmin=1800.002, xmax=1971.915, ymin=0, ymax=27401.2,fill="red",alpha=0.2)+  
 annotate("rect",xmin=1971.915, xmax=Inf, ymin=0, ymax=Inf,fill="blue",alpha=0.2)+  
 annotate("rect",xmin=0, xmax=1800.002, ymin=0, ymax=Inf,fill="red",alpha=0.2)+  
 annotate("rect",xmin=1800.002, xmax=1971.915, ymin=27401.2, ymax=Inf,fill="blue",alpha=0.2)



#print(ct\_model) # model results

#summary(ct\_model) # model result details

Get the predicted value - class membership (yes or no) –> using a cut-off of 50%.

ct\_pred\_class<-predict(ct\_model,type="class") # class membership (yes or no)   
head(ct\_pred\_class)

## 1 2 3 4 5 6   
## No No No No No No   
## Levels: No Yes

ct\_pred<-predict(ct\_model) # get the predicted values - class probabilities (default)  
head(ct\_pred)

## No Yes  
## 1 0.9823929 0.01760708  
## 2 0.9823929 0.01760708  
## 3 0.9823929 0.01760708  
## 4 0.9823929 0.01760708  
## 5 0.9823929 0.01760708  
## 6 0.9823929 0.01760708

Let’s create a new column in Default: save the predicted probability of default (yes) from the second column of dt\_pred.

Default$ct\_pred\_prob<-ct\_pred[,2]

Alternatively, you can specify a certain cut-off value to assign class membership. You can set the cut-off at 30%, 50%, 80%, or whatever you want.

Default$ct\_pred\_class<-ifelse(Default$ct\_pred\_prob>0.5,"Yes","No")

head(Default)

## # A tibble: 6 x 6  
## default student balance income ct\_pred\_prob ct\_pred\_class  
## <fct> <fct> <dbl> <dbl> <dbl> <chr>   
## 1 No No 730. 44362. 0.0176 No   
## 2 No Yes 817. 12106. 0.0176 No   
## 3 No No 1074. 31767. 0.0176 No   
## 4 No No 529. 35704. 0.0176 No   
## 5 No No 786. 38463. 0.0176 No   
## 6 No Yes 920. 7492. 0.0176 No

Default[253,] # get the information of 253th customer

## # A tibble: 1 x 6  
## default student balance income ct\_pred\_prob ct\_pred\_class  
## <fct> <fct> <dbl> <dbl> <dbl> <chr>   
## 1 No Yes 489. 15159. 0.0176 No

# show the customers whose predicted probability is greater than 70%  
Default%>%  
 filter(ct\_pred\_prob>0.7)

## # A tibble: 118 x 6  
## default student balance income ct\_pred\_prob ct\_pred\_class  
## <fct> <fct> <dbl> <dbl> <dbl> <chr>   
## 1 Yes Yes 2206. 14271. 0.763 Yes   
## 2 No Yes 2023. 18337. 0.763 Yes   
## 3 Yes No 1992. 42133. 0.763 Yes   
## 4 Yes No 1981. 28128. 0.763 Yes   
## 5 No Yes 2005. 27137. 0.763 Yes   
## 6 No No 2113. 34748. 0.763 Yes   
## 7 Yes No 2033. 44998. 0.763 Yes   
## 8 Yes No 2024. 51509. 0.763 Yes   
## 9 No Yes 1994. 14305. 0.763 Yes   
## 10 Yes No 2499. 51504. 0.763 Yes   
## # ... with 108 more rows

# sort customers by probability of default in descending order  
Default%>%  
 arrange(desc(ct\_pred\_prob))

## # A tibble: 10,000 x 6  
## default student balance income ct\_pred\_prob ct\_pred\_class  
## <fct> <fct> <dbl> <dbl> <dbl> <chr>   
## 1 Yes Yes 2206. 14271. 0.763 Yes   
## 2 No Yes 2023. 18337. 0.763 Yes   
## 3 Yes No 1992. 42133. 0.763 Yes   
## 4 Yes No 1981. 28128. 0.763 Yes   
## 5 No Yes 2005. 27137. 0.763 Yes   
## 6 No No 2113. 34748. 0.763 Yes   
## 7 Yes No 2033. 44998. 0.763 Yes   
## 8 Yes No 2024. 51509. 0.763 Yes   
## 9 No Yes 1994. 14305. 0.763 Yes   
## 10 Yes No 2499. 51504. 0.763 Yes   
## # ... with 9,990 more rows

## Random Forest

set.seed(1)  
#install.packages("randomForest")  
library(randomForest)

## randomForest 4.7-1

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

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

rf\_model<-randomForest(default~income+balance+student, # model formula  
 data=Default,ntree=500, cutoff=c(0.5,0.5))

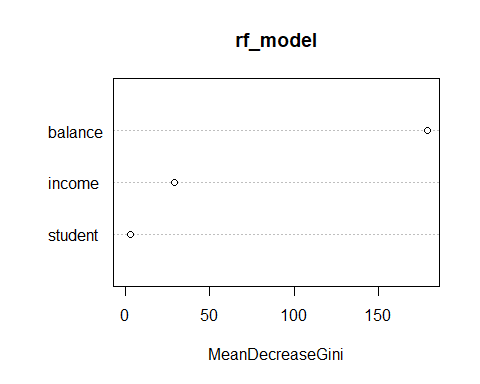
#print(rf\_model)  
head(rf\_model$votes) # indicates the % of trees that voted for each class

## No Yes  
## 1 1.0000000 0.000000000  
## 2 1.0000000 0.000000000  
## 3 0.9946809 0.005319149  
## 4 1.0000000 0.000000000  
## 5 1.0000000 0.000000000  
## 6 1.0000000 0.000000000

head(rf\_model$predicted) # the class favored by more trees (i.e. majority vote wins)

## 1 2 3 4 5 6   
## No No No No No No   
## Levels: No Yes

varImpPlot(rf\_model) # importance of variables



head(rf\_model$vote)

## No Yes  
## 1 1.0000000 0.000000000  
## 2 1.0000000 0.000000000  
## 3 0.9946809 0.005319149  
## 4 1.0000000 0.000000000  
## 5 1.0000000 0.000000000  
## 6 1.0000000 0.000000000

Default$rf\_vote<-predict(rf\_model,type="prob")[,2]  
head(Default)

## # A tibble: 6 x 7  
## default student balance income ct\_pred\_prob ct\_pred\_class rf\_vote  
## <fct> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 No No 730. 44362. 0.0176 No 0   
## 2 No Yes 817. 12106. 0.0176 No 0   
## 3 No No 1074. 31767. 0.0176 No 0.00532  
## 4 No No 529. 35704. 0.0176 No 0   
## 5 No No 786. 38463. 0.0176 No 0   
## 6 No Yes 920. 7492. 0.0176 No 0

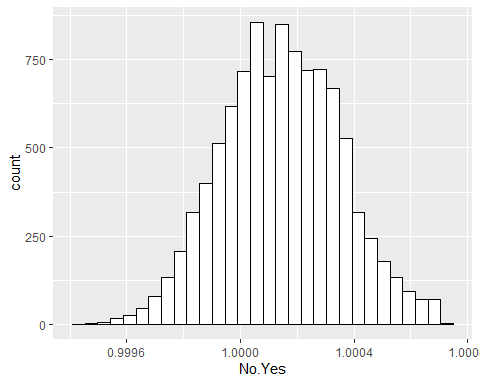
## Support Vector Machine (SVM)

library(e1071)  
model\_svm<-svm(formula= default ~ balance+income+student, # model formula   
 data=Default, # dataset  
 kernel="linear", # this is the form of the decision boundary. Let's start with a linear kernel.   
 cost=0.1) # there are paremeters that are used to tune the model   
model\_svm

##   
## Call:  
## svm(formula = default ~ balance + income + student, data = Default,   
## kernel = "linear", cost = 0.1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.1   
##   
## Number of Support Vectors: 672

dv<-data.frame(model\_svm$decision.values)  
  
ggplot(dv,aes(x=No.Yes)) +  
 geom\_histogram(colour="black",fill="white")

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



head(model\_svm$fitted)

## 1 2 3 4 5 6   
## No No No No No No   
## Levels: No Yes

table(model\_svm$fitted)

##   
## No Yes   
## 10000 0

predicted\_svm<-predict(model\_svm,Default,decision.values = TRUE)  
head(attr(predicted\_svm, "decision.values"))

## No/Yes  
## 1 1.0000632  
## 2 1.0004151  
## 3 0.9999982  
## 4 1.0001708  
## 5 1.0000694  
## 6 1.0003995

Default

## # A tibble: 10,000 x 7  
## default student balance income ct\_pred\_prob ct\_pred\_class rf\_vote  
## <fct> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 No No 730. 44362. 0.0176 No 0   
## 2 No Yes 817. 12106. 0.0176 No 0   
## 3 No No 1074. 31767. 0.0176 No 0.00532  
## 4 No No 529. 35704. 0.0176 No 0   
## 5 No No 786. 38463. 0.0176 No 0   
## 6 No Yes 920. 7492. 0.0176 No 0   
## 7 No No 826. 24905. 0.0176 No 0   
## 8 No Yes 809. 17600. 0.0176 No 0   
## 9 No No 1161. 37469. 0.0176 No 0   
## 10 No No 0 29275. 0.0176 No 0   
## # ... with 9,990 more rows

Default$svm\_pred\_class <- predict(model\_svm, Default)   
Default$svm\_dv<-c(attr(predicted\_svm, "decision.values"))  
  
Default

## # A tibble: 10,000 x 9  
## default student balance income ct\_pred\_prob ct\_pred\_class rf\_vote  
## <fct> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 No No 730. 44362. 0.0176 No 0   
## 2 No Yes 817. 12106. 0.0176 No 0   
## 3 No No 1074. 31767. 0.0176 No 0.00532  
## 4 No No 529. 35704. 0.0176 No 0   
## 5 No No 786. 38463. 0.0176 No 0   
## 6 No Yes 920. 7492. 0.0176 No 0   
## 7 No No 826. 24905. 0.0176 No 0   
## 8 No Yes 809. 17600. 0.0176 No 0   
## 9 No No 1161. 37469. 0.0176 No 0   
## 10 No No 0 29275. 0.0176 No 0   
## # ... with 9,990 more rows, and 2 more variables: svm\_pred\_class <fct>,  
## # svm\_dv <dbl>

## Logistic Regression

logit\_model<-glm(default~student+balance+income, # generalized linear models  
 family="binomial", # specifying error distribution  
 data=Default) # dataset  
summary(logit\_model)

##   
## Call:  
## glm(formula = default ~ student + balance + income, family = "binomial",   
## data = Default)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.4691 -0.1418 -0.0557 -0.0203 3.7383   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.087e+01 4.923e-01 -22.080 < 2e-16 \*\*\*  
## studentYes -6.468e-01 2.363e-01 -2.738 0.00619 \*\*   
## balance 5.737e-03 2.319e-04 24.738 < 2e-16 \*\*\*  
## income 3.033e-06 8.203e-06 0.370 0.71152   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2920.6 on 9999 degrees of freedom  
## Residual deviance: 1571.5 on 9996 degrees of freedom  
## AIC: 1579.5  
##   
## Number of Fisher Scoring iterations: 8

### Use of the model to predict

Default$log\_odd<-predict(logit\_model) # get predicted log odds (default)  
Default$logit\_pred\_prob<-predict(logit\_model,type="response") # get predicted probabilities  
glimpse(Default)

## Rows: 10,000  
## Columns: 11  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, No~  
## $ student <fct> No, Yes, No, No, No, Yes, No, Yes, No, No, Yes, Yes, N~  
## $ balance <dbl> 729.5265, 817.1804, 1073.5492, 529.2506, 785.6559, 919~  
## $ income <dbl> 44361.625, 12106.135, 31767.139, 35704.494, 38463.496,~  
## $ ct\_pred\_prob <dbl> 0.01760708, 0.01760708, 0.01760708, 0.01760708, 0.0176~  
## $ ct\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No", ~  
## $ rf\_vote <dbl> 0.000000000, 0.000000000, 0.005319149, 0.000000000, 0.~  
## $ svm\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, No~  
## $ svm\_dv <dbl> 1.0000632, 1.0004151, 0.9999982, 1.0001708, 1.0000694,~  
## $ log\_odd <dbl> -6.549544, -6.791338, -4.614261, -7.724689, -6.245449,~  
## $ logit\_pred\_prob <dbl> 1.428724e-03, 1.122204e-03, 9.812272e-03, 4.415893e-04~

Default%>%  
 select("default","student","log\_odd","logit\_pred\_prob")

## # A tibble: 10,000 x 4  
## default student log\_odd logit\_pred\_prob  
## <fct> <fct> <dbl> <dbl>  
## 1 No No -6.55 0.00143   
## 2 No Yes -6.79 0.00112   
## 3 No No -4.61 0.00981   
## 4 No No -7.72 0.000442   
## 5 No No -6.25 0.00194   
## 6 No Yes -6.22 0.00199   
## 7 No No -6.06 0.00233   
## 8 No Yes -6.82 0.00109   
## 9 No No -4.09 0.0164   
## 10 No No -10.8 0.0000208  
## # ... with 9,990 more rows

With the predicted probabilities, you can sort customers by the predicted probability of default in descending order.

Default%>%  
 arrange(desc(logit\_pred\_prob))%>%  
 select(default, student, balance, income, ct\_pred\_prob, ct\_pred\_class,logit\_pred\_prob)

## # A tibble: 10,000 x 7  
## default student balance income ct\_pred\_prob ct\_pred\_class logit\_pred\_prob  
## <fct> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 Yes Yes 2654. 21930. 0.763 Yes 0.978  
## 2 Yes No 2499. 51504. 0.763 Yes 0.974  
## 3 Yes Yes 2578. 25707. 0.763 Yes 0.966  
## 4 Yes No 2413. 38541. 0.763 Yes 0.957  
## 5 No No 2391. 50303. 0.763 Yes 0.953  
## 6 Yes Yes 2503. 14948. 0.763 Yes 0.947  
## 7 Yes No 2344. 51095. 0.763 Yes 0.939  
## 8 Yes Yes 2462. 11879. 0.763 Yes 0.933  
## 9 Yes No 2288. 52044. 0.763 Yes 0.918  
## 10 Yes Yes 2415. 17430. 0.763 Yes 0.916  
## # ... with 9,990 more rows

And use a different cut-off for class prediction using ifelse().

Default$logit\_pred\_class<-ifelse(Default$logit\_pred\_prob>0.5,"Yes","No")  
Default

## # A tibble: 10,000 x 12  
## default student balance income ct\_pred\_prob ct\_pred\_class rf\_vote  
## <fct> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 No No 730. 44362. 0.0176 No 0   
## 2 No Yes 817. 12106. 0.0176 No 0   
## 3 No No 1074. 31767. 0.0176 No 0.00532  
## 4 No No 529. 35704. 0.0176 No 0   
## 5 No No 786. 38463. 0.0176 No 0   
## 6 No Yes 920. 7492. 0.0176 No 0   
## 7 No No 826. 24905. 0.0176 No 0   
## 8 No Yes 809. 17600. 0.0176 No 0   
## 9 No No 1161. 37469. 0.0176 No 0   
## 10 No No 0 29275. 0.0176 No 0   
## # ... with 9,990 more rows, and 5 more variables: svm\_pred\_class <fct>,  
## # svm\_dv <dbl>, log\_odd <dbl>, logit\_pred\_prob <dbl>, logit\_pred\_class <chr>

.

Let’s try to find the best set of parameters through model validation.

## Classification Tree Model

### train/test split

Let’s select 2,000 indices out of 10,000 (20% of dataset). Using these indices, we will create a test and a training dataset.

set.seed(1) # set a random seed   
index <- sample(10000, 2000) # random selection of indices.

Check index. It contains 2,000 random numbers.

index

## [1] 1017 8004 4775 9725 8462 4050 8789 1301 8522 1799 9941 8229 1129 8921  
## [15] 878 8677 7845 5922 6526 5071 2900 7075 4650 9638 2159 3476 1948 2580  
## [29] 1530 7289 4633 6519 4344 1222 2858 5400 9888 526 1069 5522 8214 5838  
## [43] 5862 6337 5491 9320 7494 9175 1791 9831 4939 465 7976 9392 3863 9326  
## [57] 8276 1895 3101 3990 6877 6764 1328 8749 8479 6473 6995 9714 858 6763  
## [71] 9182 316 733 6273 4907 5051 2330 9359 6429 3992 1706 501 6655 536  
## [85] 6881 7507 7514 3747 5163 29 1942 4934 1820 2281 9509 1966 8561 4182  
## [99] 5603 5742 8547 5939 6956 9265 8553 9532 9458 6902 2866 8943 219 8327  
## [113] 532 3123 9104 6611 2178 5936 7233 5932 4455 2153 1148 1101 1242 1218  
## [127] 4115 8465 418 6563 867 4499 3821 9803 5897 818 4730 664 719 500  
## [141] 3045 6816 6373 8615 8159 8613 9181 3598 3700 5136 9346 8696 785 6789  
## [155] 7131 8053 1572 4401 1833 2461 4225 309 7892 4078 7146 8633 3189 470  
## [169] 1360 1822 1790 8541 8066 3144 9086 4686 8666 9855 5048 455 6500 7793  
## [183] 3306 6217 8207 4158 4486 9251 5987 9988 5447 6809 8174 4096 6887 4084  
## [197] 2110 1172 797 9966 9788 4072 5057 7452 5388 2762 8802 1265 7812 9903  
## [211] 2265 2156 6415 5390 9891 7151 5458 5729 1760 5704 5685 4664 4075 5474  
## [225] 7525 4212 3070 5353 7439 9655 9413 462 6443 4049 6657 9690 9765 7888  
## [239] 2033 731 7039 2813 3805 4771 966 9383 6193 5591 5793 8483 4749 8439  
## [253] 2482 6885 3388 3418 2722 2884 56 7193 934 7640 8109 4590 7883 3552  
## [267] 5123 6803 4257 127 717 9644 7715 7447 676 9969 5795 148 121 8361  
## [281] 8790 1491 2815 1115 6308 3616 3551 6525 436 2137 8137 533 9573 2752  
## [295] 5592 9038 5191 6459 8411 9050 3565 570 6976 1926 5666 1813 7681 2833  
## [309] 7885 5694 7981 5161 7560 8473 4063 7918 9864 724 1354 8684 6164 3255  
## [323] 9779 5199 8706 6017 9445 2016 132 6186 5094 1659 6068 6438 9613 2312  
## [337] 6321 5040 1839 3505 1838 5259 8512 4211 6595 9434 6990 6303 2167 2543  
## [351] 5914 8301 3760 6663 2587 8945 9926 7935 393 8840 7046 7923 4208 5147  
## [365] 4318 6410 4254 9746 2778 9673 7343 6103 7334 9331 4436 6236 5772 99  
## [379] 7684 7983 3200 3822 8090 6989 5763 7805 1190 1459 5117 1776 116 7360  
## [393] 9618 3157 1749 1326 1847 9424 1018 9858 383 8920 771 5207 484 4599  
## [407] 6449 3887 2563 58 4801 2021 9736 2170 1803 1124 313 6790 822 6625  
## [421] 5480 5131 3382 9607 81 3328 3126 2194 5235 5663 2796 6472 2775 7174  
## [435] 1401 8627 5605 7961 3360 8426 1779 7656 4278 8253 4606 1089 8841 1861  
## [449] 9890 7374 7893 9568 2992 3797 6079 1177 8209 1057 3706 6440 7123 2257  
## [463] 1525 6420 5809 2079 9847 8987 7009 8765 3037 6485 3558 4568 7842 2351  
## [477] 7666 3651 6266 1871 5034 5005 8060 363 8816 2573 9459 9647 3329 2373  
## [491] 702 1378 959 2273 9874 8589 5031 9791 744 564 9796 2290 3482 5327  
## [505] 5287 87 4313 5633 860 5620 7817 6134 2446 628 5368 1104 1493 4062  
## [519] 3926 4784 2811 1473 2572 4364 9545 8863 4933 2569 9410 4745 2551 128  
## [533] 7211 1435 102 7255 4066 3457 453 4585 6138 9321 8146 6050 815 8766  
## [547] 644 7238 8240 8151 3437 7120 9366 2608 4358 1590 7368 3352 9027 2899  
## [561] 9734 1821 9531 7917 8793 1759 8910 2107 1019 2020 8735 7757 9592 296  
## [575] 5144 5853 764 5566 8314 3047 8690 1043 376 4562 6101 919 9876 1487  
## [589] 6109 5999 5661 7025 854 8717 9954 108 9965 3862 3018 2285 8588 144  
## [603] 3733 5254 9739 8444 2069 4963 133 9792 386 1802 8482 8962 5756 7500  
## [617] 9094 2158 4297 487 7379 3254 1421 908 3363 247 8054 8649 9183 6320  
## [631] 8775 647 2732 492 3807 3586 1620 7348 3129 5435 553 1864 1443 5238  
## [645] 7445 7200 1031 4330 2531 3417 9447 1587 9737 1651 3402 6818 5776 5195  
## [659] 7512 1630 7998 8864 5527 2136 9477 5148 8343 1524 8978 3946 827 4876  
## [673] 2616 2684 8825 6167 8968 7419 1131 4333 1890 5334 4404 2997 7637 6398  
## [687] 9846 8378 4407 2019 2548 5750 1967 4279 9667 8539 4490 6557 8781 5305  
## [701] 6583 2161 5630 4210 8692 1216 2395 8423 129 2424 3145 2792 314 4324  
## [715] 8881 8101 823 8164 407 5024 9741 280 3579 517 3793 3590 8861 851  
## [729] 9914 3465 9367 6314 7652 793 703 324 8427 3641 3390 2106 787 8068  
## [743] 1373 1592 660 5006 1038 6574 9405 64 2131 9112 626 8498 6850 5607  
## [757] 5502 4848 2506 6293 8097 8567 5920 284 6940 1863 1950 8094 8608 8168  
## [771] 3210 4759 4973 4023 9777 9935 7173 7555 9924 9491 4206 3729 6849 4400  
## [785] 5512 7891 384 6776 5330 9884 5846 5943 9243 8644 573 1611 5165 8345  
## [799] 1700 3781 479 1922 7914 8544 8389 4083 3298 9904 5863 4186 4064 5671  
## [813] 2664 8971 8827 249 9740 7239 101 7965 3665 520 5670 8313 9339 3415  
## [827] 7435 650 617 4676 1846 4223 3680 7568 1428 7300 4105 4944 5361 3338  
## [841] 5864 9101 9228 5337 4754 8778 812 2480 5956 338 7809 5845 6512 8398  
## [855] 8634 4777 8147 2267 6919 8782 9594 7310 1372 6063 6081 9009 2823 220  
## [869] 7095 6558 5413 6584 4572 2214 4637 8744 3663 6581 8714 611 7671 217  
## [883] 9290 8654 4819 6239 7309 3631 6515 6531 7826 271 1299 9808 2871 6799  
## [897] 7821 4171 7304 5957 4893 4733 27 8753 6448 4578 4385 9167 2937 4292  
## [911] 3766 7228 7261 9839 8443 5061 9080 3115 2223 5912 9398 1598 8405 5575  
## [925] 5934 8693 1088 4721 2761 9051 3572 1071 6348 9816 503 2541 5466 2439  
## [939] 5857 2864 979 3152 9210 2284 610 9162 8682 7574 6226 2116 656 9998  
## [953] 4327 3702 6693 5135 5176 7486 1252 2826 9325 2391 2678 9319 413 8395  
## [967] 8671 6660 7785 7987 8331 1259 5189 7740 4901 4103 9677 4895 7799 1654  
## [981] 427 9276 8131 582 8768 8045 1817 225 6247 5578 8377 2841 2426 4609  
## [995] 5770 8437 3679 3842 2928 450 6008 665 1670 8806 1090 5281 5253 3715  
## [1009] 4836 6357 4589 4596 2403 8231 7008 1552 721 5761 9156 7545 8578 5473  
## [1023] 4896 6883 5767 6417 9630 327 6750 1205 3237 1768 4038 6903 5049 2296  
## [1037] 6834 9565 7658 3316 6160 838 7808 2490 4474 4703 5396 4649 4630 3977  
## [1051] 556 4346 306 945 2772 4826 1279 8032 8941 4286 8416 9053 7040 9702  
## [1065] 4904 8818 2765 2240 4588 534 2423 6265 9660 1526 2389 5749 401 6987  
## [1079] 4780 3929 34 6387 3728 9610 6093 2751 7986 347 3984 1302 8763 4866  
## [1093] 3580 5564 7895 5727 9384 5418 68 4618 9284 5184 8365 391 2350 505  
## [1107] 4475 3983 9845 3553 935 1316 5240 7565 8926 3936 5157 2408 90 2600  
## [1121] 2499 1689 5101 1954 3387 4142 1511 773 3375 2145 40 41 5291 4239  
## [1135] 113 3890 6744 2255 3158 6262 8325 7004 5735 1641 4527 9621 7663 5394  
## [1149] 5470 4612 9695 3159 6196 2705 6480 1830 6963 1581 5958 5007 2904 4778  
## [1163] 667 3435 4811 4611 5260 1778 2889 5971 9606 9340 7758 2002 4298 9127  
## [1177] 8117 1792 7298 7598 6896 759 1560 6211 6251 1258 9774 4107 6238 5948  
## [1191] 2617 3920 699 8532 3792 3426 9942 7182 3814 6384 5650 2235 3993 1376  
## [1205] 9790 3845 7871 3956 4953 1098 5794 9357 2934 2538 4724 1000 5728 6335  
## [1219] 6035 9863 9834 7945 619 5192 1144 2182 1636 2282 2185 7613 4258 3531  
## [1233] 6461 6762 9656 6592 5307 9875 3561 2129 8501 1965 1742 310 5814 7662  
## [1247] 616 1092 4086 6060 2211 4119 8942 8739 9414 756 7721 1818 1782 1198  
## [1261] 7993 6968 8096 5993 6264 8432 1619 7571 4773 6958 3309 5468 1929 6128  
## [1275] 649 2602 8598 965 5265 54 4634 3257 8916 9002 5245 6586 2595 3266  
## [1289] 830 9345 569 441 1983 7 9448 229 5316 2930 9171 8243 9688 5581  
## [1303] 2301 8705 1362 6284 2409 4110 1951 2124 6430 672 2766 2379 1996 6488  
## [1317] 8656 1961 1998 8075 335 7263 5788 9421 9799 1106 4150 3818 5836 2744  
## [1331] 3051 44 7628 6623 6275 1786 1935 9141 2380 7436 6571 5257 6791 3804  
## [1345] 5797 3235 5318 9198 7995 3327 5138 9254 8524 1936 1949 2840 8857 3879  
## [1359] 7936 5686 2356 8977 8451 7523 406 5765 5186 4804 3324 697 3915 3447  
## [1373] 2924 2076 2837 1154 1748 9200 692 4451 6970 4894 1035 7218 3661 5093  
## [1387] 4315 7680 3731 1457 5003 3058 2044 2789 7621 3442 7421 9657 8730 7974  
## [1401] 8413 3245 7471 5226 9248 498 3143 1221 2962 9262 9753 3100 3318 9920  
## [1415] 3229 5807 1806 7216 3039 4713 5268 5949 9640 2003 9225 4632 8587 4807  
## [1429] 6537 1213 3765 1855 7109 6207 5562 9513 5843 9095 3625 9772 6616 1610  
## [1443] 5889 463 7903 2788 8866 1603 8517 263 3768 1386 9873 9220 2382 4419  
## [1457] 9099 8689 8133 3434 7696 9264 36 990 9947 9281 6058 1984 8322 5653  
## [1471] 8724 1828 5841 8407 6514 2645 2378 3212 9299 6313 5089 1718 8275 9238  
## [1485] 7303 5411 7400 1229 9149 6383 761 4507 7837 3414 4736 2419 1186 5963  
## [1499] 2206 7679 1692 168 5940 8425 9012 8014 9523 1732 615 5533 7114 1368  
## [1513] 4418 2571 967 3381 6696 881 8976 5158 8531 7738 8762 7464 2336 9597  
## [1527] 1781 7503 4860 9268 259 8653 5438 5227 5941 2567 9824 4965 396 7375  
## [1541] 5239 1389 6344 6522 6432 3750 7739 8890 8518 6851 2410 6305 7410 7697  
## [1555] 3873 6061 4888 1920 3698 9971 6589 6237 8333 9184 9322 7933 8548 6939  
## [1569] 6003 5479 1600 2436 2437 3119 4742 2919 7451 2432 5677 9350 8119 1274  
## [1583] 3840 4654 7033 2109 7814 2258 4660 2486 2652 149 7117 7267 3204 9266  
## [1597] 3970 7381 6311 1907 185 7332 9219 5651 1937 1116 5014 9206 9973 1731  
## [1611] 7101 2686 4689 7566 5406 1370 3852 977 7276 8139 4176 38 7912 5621  
## [1625] 7154 6298 6444 2921 4433 170 7116 6052 334 1656 1424 5617 6394 8655  
## [1639] 7880 3031 2261 6684 9120 3052 2429 3489 9146 2025 9807 7462 9960 1677  
## [1653] 4506 3290 5560 2008 9293 9007 9070 1831 9335 6649 1568 7245 2148 8516  
## [1667] 1697 4125 8586 2053 2238 3446 6889 8878 6112 9090 6487 8507 7425 8612  
## [1681] 741 2985 6386 4471 746 7000 9375 2058 3431 9297 6878 4178 3726 5350  
## [1695] 8640 7015 7413 2370 5310 9089 9232 5196 7150 476 8858 1865 6983 2401  
## [1709] 8853 889 6257 9294 1693 4723 226 8514 3184 8015 9380 8337 97 4852  
## [1723] 7894 448 4240 842 1579 1078 9123 4482 5632 5100 9044 1065 7386 6969  
## [1737] 8550 2784 7647 2164 1297 7387 8093 7415 2530 8984 2674 3922 524 4758  
## [1751] 7763 6156 3154 4037 7176 5030 1801 4166 3127 8767 9716 7710 2445 5082  
## [1765] 1026 3543 3165 1577 3575 8672 1845 46 5917 8219 726 2725 2892 1023  
## [1779] 8710 3954 5819 4477 7230 7449 2844 283 457 1275 333 4692 7124 7752  
## [1793] 6362 392 4288 1726 1158 728 5830 1853 5045 7626 2102 2377 2638 3876  
## [1807] 4007 9632 7119 3203 2213 4547 2797 3669 7539 1076 1261 7776 2790 1488  
## [1821] 5931 7736 6745 1953 3439 2397 6954 6869 3034 456 3503 5764 4430 3219  
## [1835] 251 6505 1022 3104 7184 5893 9191 668 2367 5262 796 8048 6900 2637  
## [1849] 5530 1495 4799 5970 6719 8986 8940 3913 8347 2374 1130 9363 7085 3347  
## [1863] 4045 7256 5609 4058 1349 2472 6882 682 6926 9669 5676 6242 2335 3530  
## [1877] 241 5456 6587 3384 871 5918 1164 9418 4360 2708 4111 2673 7473 6681  
## [1891] 1663 5540 9580 4615 5249 4863 4917 7140 874 2323 6839 826 7266 2359  
## [1905] 4428 3075 9163 63 1245 4003 6573 6997 2504 4146 4487 2619 1974 8171  
## [1919] 6307 740 326 5628 3752 6051 925 80 5044 4051 778 5321 7544 4625  
## [1933] 9151 7534 2413 1133 4458 2967 112 6208 5433 2644 4222 7306 214 2880  
## [1947] 5272 2568 9031 7083 4899 3522 3193 8596 2842 1193 9143 1874 4485 931  
## [1961] 3596 4740 2781 4753 4669 8435 8788 9701 7265 1567 952 6646 3690 4550  
## [1975] 8088 5667 5058 8124 9169 3017 4290 8200 2857 5367 9406 2094 3025 369  
## [1989] 5319 9399 6091 1678 9776 4303 1517 637 6310 7034 6539 1391

The following codes will select the data instances in the rows in index and save them as test. The rest will be saved as training.

test <- Default[index,c("default","student","balance","income")] # save 20% as a test dataset  
#test1 <- Default[index, ]  
training <-Default[-index,c("default","student","balance","income")] # save the rest as a training set  
  
test

## # A tibble: 2,000 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No No 939. 45519.  
## 2 No Yes 398. 22711.  
## 3 Yes No 1512. 53507.  
## 4 No No 301. 51540.  
## 5 No No 878. 29562.  
## 6 Yes No 1673. 49310.  
## 7 No No 310. 37697.  
## 8 No No 1272. 44896.  
## 9 No No 887. 41641.  
## 10 No No 231. 32799.  
## # ... with 1,990 more rows

training

## # A tibble: 8,000 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No No 730. 44362.  
## 2 No Yes 817. 12106.  
## 3 No No 1074. 31767.  
## 4 No No 529. 35704.  
## 5 No No 786. 38463.  
## 6 No Yes 920. 7492.  
## 7 No Yes 809. 17600.  
## 8 No No 1161. 37469.  
## 9 No No 0 29275.  
## 10 No Yes 0 21871.  
## # ... with 7,990 more rows

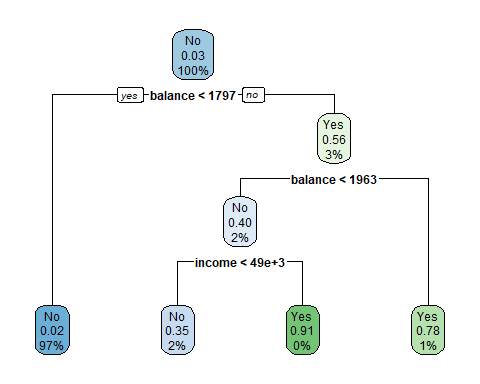
Alternatively,

test<-Default%>%  
 filter(row\_number() %in% index)%>%  
 select("default","student","balance","income")  
  
training<-Default%>%  
 select("default","student","balance","income")%>%  
 setdiff(test)

### build a model with a training set

Build a model using the training set. This part is the same as before except we use training instead of the entire dataset.

training\_model<-rpart(default~student+balance+income,  
 data=training,   
 method="class",   
 control=rpart.control(cp=0.03))  
  
rpart.plot(training\_model)



This model does not look much different from what we had. But we now have 20% of dataset that were not used when we built this model.

### predicting probabilities/class labels for test data

Now, the model will be evaluated on a test data. For this, we apply the model to the test dataset and get the predicted values. It can be done by providing a dataset name with a model to predict() function.

test$ct\_pred\_prob<-predict(training\_model,test)[,2]  
head(test)

## # A tibble: 6 x 5  
## default student balance income ct\_pred\_prob  
## <fct> <fct> <dbl> <dbl> <dbl>  
## 1 No No 826. 24905. 0.0174  
## 2 No No 643. 41474. 0.0174  
## 3 No No 616. 39376. 0.0174  
## 4 No No 914. 46907. 0.0174  
## 5 No Yes 1500. 13191. 0.0174  
## 6 No No 351. 35087. 0.0174

test$ct\_pred\_class<-predict(training\_model,test,type="class")  
test

## # A tibble: 2,000 x 6  
## default student balance income ct\_pred\_prob ct\_pred\_class  
## <fct> <fct> <dbl> <dbl> <dbl> <fct>   
## 1 No No 826. 24905. 0.0174 No   
## 2 No No 643. 41474. 0.0174 No   
## 3 No No 616. 39376. 0.0174 No   
## 4 No No 914. 46907. 0.0174 No   
## 5 No Yes 1500. 13191. 0.0174 No   
## 6 No No 351. 35087. 0.0174 No   
## 7 No No 653. 39490. 0.0174 No   
## 8 No No 872. 41788. 0.0174 No   
## 9 No Yes 221. 16873. 0.0174 No   
## 10 No No 409. 54207. 0.0174 No   
## # ... with 1,990 more rows

Check the accuracy of the model by comparing the the actual values (default) and the predicted values (ct\_pred\_class).

table(test$default==test$ct\_pred\_class)

##   
## FALSE TRUE   
## 63 1937

1947/2000

## [1] 0.9735

sum(predict(training\_model, test, type="class")==test$default)/nrow(test)

## [1] 0.9685

sum(predict(training\_model, training, type="class")==training$default)/nrow(training)

## [1] 0.975

###### holdout validation (Back to slide)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

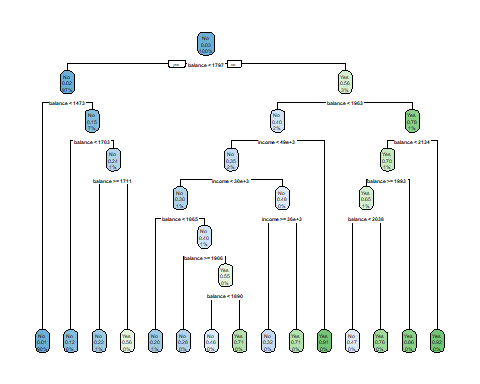
You may also want to check a confusion table.

table(test$ct\_pred\_class,test$default, dnn=c("predicted","actual")) # confusion table on test data

## actual  
## predicted No Yes  
## No 1916 49  
## Yes 14 21

### k-fold Cross-validation

set.seed(1) # set a random seed   
full\_tree<-rpart(default~student+balance+income,  
 data=training,   
 method="class",  
 control=rpart.control(cp=0)) #complexity increases as cp value ~ zero  
  
rpart.plot(full\_tree)



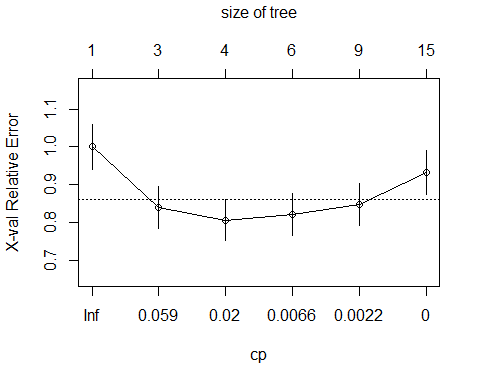
xerror is cross-validated relative error rates, and xstd is its standard deviation.

printcp(full\_tree) # xerror, xstd - cross validation results

##   
## Classification tree:  
## rpart(formula = default ~ student + balance + income, data = training,   
## method = "class", control = rpart.control(cp = 0))  
##   
## Variables actually used in tree construction:  
## [1] balance income   
##   
## Root node error: 263/8000 = 0.032875  
##   
## n= 8000   
##   
## CP nsplit rel error xerror xstd  
## 1 0.1026616 0 1.00000 1.00000 0.060641  
## 2 0.0342205 2 0.79468 0.84030 0.055739  
## 3 0.0114068 3 0.76046 0.80608 0.054624  
## 4 0.0038023 5 0.73764 0.82129 0.055122  
## 5 0.0012674 8 0.72624 0.84791 0.055983  
## 6 0.0000000 14 0.71863 0.93156 0.058597

Using plotcp(), you can check how the cross-validation error rate changes as the complexity of the model increases. In this chart, x-axis is model complexity, and y-axis is xerror rate (from cross-validation). The bars indicate standard deviation.

plotcp(full\_tree)

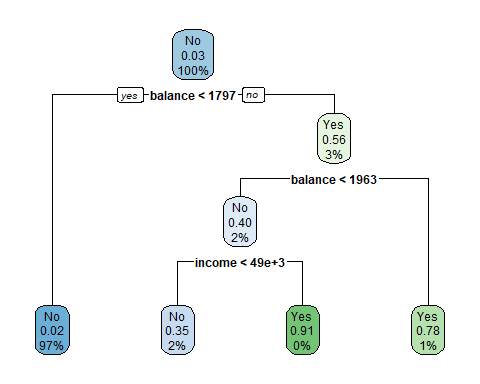


We may choose the cp value that minimizes cross-validation errors. However, it may not be always the best option. As you can see, the error rate with 4 splits is within the rage of standard deviation of the minimum error rate. You may want to choose the one with 4 splits for the ease of interpretation.

min\_xerror<-full\_tree$cptable[which.min(full\_tree$cptable[,"xerror"]),]  
min\_xerror

## CP nsplit rel error xerror xstd   
## 0.01140684 3.00000000 0.76045627 0.80608365 0.05462358

# prune tree with minimum cp value  
min\_xerror\_tree<-prune(full\_tree, cp=min\_xerror[1])  
rpart.plot(min\_xerror\_tree)



Let’s consider mim\_xerror\_tree as the best pruned tree, and get the prediction.

bp\_tree<-min\_xerror\_tree  
test$ct\_bp\_pred\_prob<-predict(bp\_tree,test)[,2]  
test$ct\_bp\_pred\_class=ifelse(test$ct\_bp\_pred\_prob>0.5,"Yes","No")  
  
head(test)

## # A tibble: 6 x 8  
## default student balance income ct\_pred\_prob ct\_pred\_class ct\_bp\_pred\_prob  
## <fct> <fct> <dbl> <dbl> <dbl> <fct> <dbl>  
## 1 No No 826. 24905. 0.0174 No 0.0174  
## 2 No No 643. 41474. 0.0174 No 0.0174  
## 3 No No 616. 39376. 0.0174 No 0.0174  
## 4 No No 914. 46907. 0.0174 No 0.0174  
## 5 No Yes 1500. 13191. 0.0174 No 0.0174  
## 6 No No 351. 35087. 0.0174 No 0.0174  
## # ... with 1 more variable: ct\_bp\_pred\_class <chr>

table(test$ct\_bp\_pred\_class!=test$default) # error rate

##   
## FALSE TRUE   
## 1937 63

53/2000

## [1] 0.0265

1947/2000

## [1] 0.9735

sum(predict(bp\_tree, test, type="class")!=test$default)/nrow(test)

## [1] 0.0315

table(test$ct\_bp\_pred\_class,test$default, dnn=c("predicted","actual")) # confusion table on test data

## actual  
## predicted No Yes  
## No 1916 49  
## Yes 14 21

## Random Forest

### hold-out validation vs. OOB errors

Following a similar process, we can validate the performance of a random forest model.

set.seed(1)  
rf\_training\_model<-randomForest(default~income+balance+student, # model formula  
 data=training, # use a training dataset for building a model  
 ntree=500,   
 cutoff=c(0.5,0.5),   
 mtry=2, # number of variables considered in each split (useful when lots of variables)  
 importance=TRUE)  
rf\_training\_model

##   
## Call:  
## randomForest(formula = default ~ income + balance + student, data = training, ntree = 500, cutoff = c(0.5, 0.5), mtry = 2, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 2.96%  
## Confusion matrix:  
## No Yes class.error  
## No 7671 66 0.008530438  
## Yes 171 92 0.650190114

sum(predict(rf\_training\_model,test,type="class")!=test$default)/nrow(test)

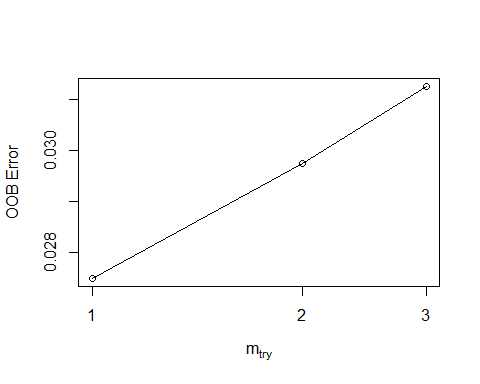
## [1] 0.033

### hyperparameter tuning for Random Forest

The tuneRF() function has an argument, ntreeTry that defaults to 50 trees. Set nTreeTry = 500 to train a random forest model of the same size as you previously did. After tuning the forest, this function will plot model performance (OOB error) as a function of the mtry values that were evaluated.

# Execute the tuning process  
set.seed(1)   
res <- tuneRF(x = training%>%select(-default),  
 y = training$default,mtryStart=2,  
 ntreeTry = 500)

## mtry = 2 OOB error = 2.97%   
## Searching left ...  
## mtry = 1 OOB error = 2.75%   
## 0.07563025 0.05   
## Searching right ...  
## mtry = 3 OOB error = 3.12%   
## -0.1363636 0.05



rf\_best\_model<-randomForest(default~income+balance+student, # model formula  
 data=training, # use a training dataset for building a model  
 ntree=500,   
 cutoff=c(0.5,0.5),   
 mtry=1,  
 importance=TRUE)  
rf\_best\_model

##   
## Call:  
## randomForest(formula = default ~ income + balance + student, data = training, ntree = 500, cutoff = c(0.5, 0.5), mtry = 1, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 1  
##   
## OOB estimate of error rate: 2.7%  
## Confusion matrix:  
## No Yes class.error  
## No 7711 26 0.003360476  
## Yes 190 73 0.722433460

test$rf\_pred\_prob<-predict(rf\_best\_model,test,type="prob")[,2] #use a test dataset for model evaluation  
test$rf\_pred\_class<-predict(rf\_best\_model,test,type="class")  
glimpse(test)

## Rows: 2,000  
## Columns: 10  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ student <fct> No, No, No, No, Yes, No, No, No, Yes, No, Yes, Yes, N~  
## $ balance <dbl> 825.5133, 642.9997, 615.7043, 913.5872, 1499.7247, 35~  
## $ income <dbl> 24905.23, 41473.51, 39376.39, 46907.23, 13190.65, 350~  
## $ ct\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ ct\_bp\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_bp\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~  
## $ rf\_pred\_prob <dbl> 0.000, 0.000, 0.000, 0.000, 0.004, 0.000, 0.002, 0.00~  
## $ rf\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~

table(test$default==test$rf\_pred\_class)

##   
## FALSE TRUE   
## 61 1939

56/2000

## [1] 0.028

1944/2000

## [1] 0.972

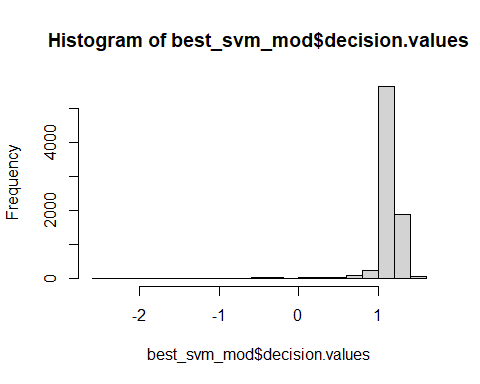
## SVM

We can tune SVM models using tune function. Set a range of search values for the parameter. It builds an SVM model for each possible combination of parameter values and evaluate accuracy. It will return the parameter combination that yields the best accuracy.

svm\_tune <- tune(svm, # find a best set of parameters for the svm model   
 default~student+balance+income,   
 data = training,  
 kernel="radial",   
 ranges = list(cost = 10^(-5:0))) # specifying the ranges of parameters   
 # in the penalty function to be examined  
 # you may wish to increase the search space   
   
  
print(svm\_tune) # best parameters for the model

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1  
##   
## - best performance: 0.02825

best\_svm\_mod <- svm\_tune$best.model # best model is saved in best.model  
  
hist(best\_svm\_mod$decision.values)



#sum(best\_svm\_mod$decision.values < 0)  
table(best\_svm\_mod$fitted)

##   
## No Yes   
## 7937 63

table(training$default==best\_svm\_mod$fitted)

##   
## FALSE TRUE   
## 222 7778

224/8000

## [1] 0.028

test$svm\_pred\_class <- predict(best\_svm\_mod, test) # save the predicted class by the svm model  
test$svm\_dv<-as.numeric(attr(predict(best\_svm\_mod, test, decision.values = TRUE),"decision.values"))  
glimpse(test)

## Rows: 2,000  
## Columns: 12  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ student <fct> No, No, No, No, Yes, No, No, No, Yes, No, Yes, Yes, N~  
## $ balance <dbl> 825.5133, 642.9997, 615.7043, 913.5872, 1499.7247, 35~  
## $ income <dbl> 24905.23, 41473.51, 39376.39, 46907.23, 13190.65, 350~  
## $ ct\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ ct\_bp\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_bp\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~  
## $ rf\_pred\_prob <dbl> 0.000, 0.000, 0.000, 0.000, 0.004, 0.000, 0.002, 0.00~  
## $ rf\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_dv <dbl> 1.134960, 1.167183, 1.195762, 1.012859, 1.105100, 1.2~

table(test$default==test$svm\_pred\_class)

##   
## FALSE TRUE   
## 60 1940

56/2000

## [1] 0.028

## Logit regression

### Hold-out validation

logit\_training\_model<-glm(default~student+balance+income,family="binomial",data=training)  
summary(logit\_training\_model)

##   
## Call:  
## glm(formula = default ~ student + balance + income, family = "binomial",   
## data = training)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1617 -0.1412 -0.0545 -0.0195 3.7829   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.072e+01 5.622e-01 -19.072 <2e-16 \*\*\*  
## studentYes -6.461e-01 2.657e-01 -2.432 0.015 \*   
## balance 5.784e-03 2.647e-04 21.854 <2e-16 \*\*\*  
## income -4.445e-06 9.487e-06 -0.469 0.639   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2313.6 on 7999 degrees of freedom  
## Residual deviance: 1247.6 on 7996 degrees of freedom  
## AIC: 1255.6  
##   
## Number of Fisher Scoring iterations: 8

test$logit\_pred\_prob<-predict(logit\_training\_model, test, type="response") #get predicted probability  
test$logit\_pred\_class<-ifelse(test$logit\_pred\_prob>0.5,"Yes","No") #get predicted class  
glimpse(test)

## Rows: 2,000  
## Columns: 14  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ student <fct> No, No, No, No, Yes, No, No, No, Yes, No, Yes, Yes, N~  
## $ balance <dbl> 825.5133, 642.9997, 615.7043, 913.5872, 1499.7247, 35~  
## $ income <dbl> 24905.23, 41473.51, 39376.39, 46907.23, 13190.65, 350~  
## $ ct\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ ct\_bp\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_bp\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~  
## $ rf\_pred\_prob <dbl> 0.000, 0.000, 0.000, 0.000, 0.004, 0.000, 0.002, 0.00~  
## $ rf\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_dv <dbl> 1.134960, 1.167183, 1.195762, 1.012859, 1.105100, 1.2~  
## $ logit\_pred\_prob <dbl> 2.332228e-03, 7.550376e-04, 6.508659e-04, 3.515907e-0~  
## $ logit\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~

table(test$default==test$logit\_pred\_class)

##   
## FALSE TRUE   
## 57 1943

57/2000

## [1] 0.0285

#sum(test$logit\_pred\_class!=test$default)/nrow(test)

### step-wise regression

To improve the logit regression performance, it’s important to understand which variables should be included. Step-wise regression helps us select variables - assess model performance by adding variable and compare.

# Specify a null model with no predictors  
null\_model <- glm(default~1, data = training, family = "binomial")   
  
# Specify the full model using all of the potential predictors  
full\_model <- glm(default~student+balance+income, data = training, family = "binomial")  
  
# Use a forward stepwise algorithm to build a parsimonious model  
forward\_model <- step(null\_model, scope = list(lower = null\_model, upper = full\_model), direction = "forward")

## Start: AIC=2315.57  
## default ~ 1  
##   
## Df Deviance AIC  
## + balance 1 1259.5 1263.5  
## + student 1 2292.9 2296.9  
## + income 1 2299.7 2303.7  
## <none> 2313.6 2315.6  
##   
## Step: AIC=1263.54  
## default ~ balance  
##   
## Df Deviance AIC  
## + student 1 1247.8 1253.8  
## + income 1 1253.5 1259.5  
## <none> 1259.5 1263.5  
##   
## Step: AIC=1253.84  
## default ~ balance + student  
##   
## Df Deviance AIC  
## <none> 1247.8 1253.8  
## + income 1 1247.6 1255.6

summary(forward\_model)

##   
## Call:  
## glm(formula = default ~ balance + student, family = "binomial",   
## data = training)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1738 -0.1413 -0.0546 -0.0195 3.7753   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.090e+01 4.220e-01 -25.826 < 2e-16 \*\*\*  
## balance 5.784e-03 2.647e-04 21.850 < 2e-16 \*\*\*  
## studentYes -5.474e-01 1.630e-01 -3.359 0.000781 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2313.6 on 7999 degrees of freedom  
## Residual deviance: 1247.8 on 7997 degrees of freedom  
## AIC: 1253.8  
##   
## Number of Fisher Scoring iterations: 8

# Use a forward stepwise algorithm to build a parsimonious model  
backward\_model <- step(full\_model, scope = list(lower = null\_model, upper = full\_model), direction = "backward")

## Start: AIC=1255.62  
## default ~ student + balance + income  
##   
## Df Deviance AIC  
## - income 1 1247.8 1253.8  
## <none> 1247.6 1255.6  
## - student 1 1253.5 1259.5  
## - balance 1 2292.8 2298.8  
##   
## Step: AIC=1253.84  
## default ~ student + balance  
##   
## Df Deviance AIC  
## <none> 1247.8 1253.8  
## - student 1 1259.5 1263.5  
## - balance 1 2292.9 2296.9

summary(backward\_model)

##   
## Call:  
## glm(formula = default ~ student + balance, family = "binomial",   
## data = training)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1738 -0.1413 -0.0546 -0.0195 3.7753   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.090e+01 4.220e-01 -25.826 < 2e-16 \*\*\*  
## studentYes -5.474e-01 1.630e-01 -3.359 0.000781 \*\*\*  
## balance 5.784e-03 2.647e-04 21.850 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2313.6 on 7999 degrees of freedom  
## Residual deviance: 1247.8 on 7997 degrees of freedom  
## AIC: 1253.8  
##   
## Number of Fisher Scoring iterations: 8

logit\_best\_model<-glm(default~student+balance,family="binomial",data=training)  
summary(logit\_best\_model)

##   
## Call:  
## glm(formula = default ~ student + balance, family = "binomial",   
## data = training)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1738 -0.1413 -0.0546 -0.0195 3.7753   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.090e+01 4.220e-01 -25.826 < 2e-16 \*\*\*  
## studentYes -5.474e-01 1.630e-01 -3.359 0.000781 \*\*\*  
## balance 5.784e-03 2.647e-04 21.850 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2313.6 on 7999 degrees of freedom  
## Residual deviance: 1247.8 on 7997 degrees of freedom  
## AIC: 1253.8  
##   
## Number of Fisher Scoring iterations: 8

test$logit\_pred\_prob<-predict(logit\_best\_model,test,type="response")  
test$logit\_pred\_class<-ifelse(test$logit\_pred\_prob>0.5,"Yes","No")   
glimpse(test)

## Rows: 2,000  
## Columns: 14  
## $ default <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ student <fct> No, No, No, No, Yes, No, No, No, Yes, No, Yes, Yes, N~  
## $ balance <dbl> 825.5133, 642.9997, 615.7043, 913.5872, 1499.7247, 35~  
## $ income <dbl> 24905.23, 41473.51, 39376.39, 46907.23, 13190.65, 350~  
## $ ct\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ ct\_bp\_pred\_prob <dbl> 0.01737452, 0.01737452, 0.01737452, 0.01737452, 0.017~  
## $ ct\_bp\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~  
## $ rf\_pred\_prob <dbl> 0.000, 0.000, 0.000, 0.000, 0.004, 0.000, 0.002, 0.00~  
## $ rf\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_pred\_class <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~  
## $ svm\_dv <dbl> 1.134960, 1.167183, 1.195762, 1.012859, 1.105100, 1.2~  
## $ logit\_pred\_prob <dbl> 2.184175e-03, 7.611338e-04, 6.500509e-04, 3.629822e-0~  
## $ logit\_pred\_class <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",~

table(test$default==test$logit\_pred\_class)

##   
## FALSE TRUE   
## 57 1943

57/2000

## [1] 0.0285

## Performance Visualization with ROC

library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

ct\_roc<-roc(test$default,test$ct\_pred\_prob,auc=TRUE) #two arguments for roc function: true value, numeric result from model

## Setting levels: control = No, case = Yes

## Setting direction: controls < cases

rf\_roc<-roc(test$default,test$rf\_pred\_prob,auc=TRUE)

## Setting levels: control = No, case = Yes  
## Setting direction: controls < cases

logit\_roc<-roc(test$default,test$logit\_pred\_prob,auc=TRUE)

## Setting levels: control = No, case = Yes  
## Setting direction: controls < cases

svm\_roc<-roc(test$default,test$svm\_dv,auc=TRUE)

## Setting levels: control = No, case = Yes

## Setting direction: controls > cases

plot(ct\_roc,print.auc=TRUE,col="blue")  
plot(rf\_roc,print.auc=TRUE,print.auc.y=.4,col="green", add=TRUE)  
plot(logit\_roc,print.auc=TRUE,print.auc.y=.3, col="red",add=TRUE)  
plot(svm\_roc,print.auc=TRUE,print.auc.y=.2, col="black",add=TRUE)

