Credit Card Fraud Detection

Tahani Omer

12/6/2019

Inspired from Kaggle Dataset - <https://www.kaggle.com/mlg-ulb/creditcardfraud>

It is a binary classification problem; therefore, the following modeling method will be used is Decision Tree In this case, our model should be activated that means it should detect most of the frauds, but it should not miss classify the legitimate transaction as fraud, that means there should be few false negative and there should be few false positive

## Data Exploration

credit = read.csv("creditcard.csv",header = T,sep = ",")  
head(credit)

## Time V1 V2 V3 V4 V5 V6  
## 1 0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077 0.46238778  
## 2 0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -0.08236081  
## 3 1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813 1.80049938  
## 4 1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888 1.24720317  
## 5 2 -1.1582331 0.87773676 1.5487178 0.4030339 -0.40719338 0.09592146  
## 6 2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -0.02972755  
## V7 V8 V9 V10 V11 V12  
## 1 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -0.61780086  
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267 1.06523531  
## 3 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015 0.06608369  
## 4 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873 0.17822823  
## 5 0.59294075 -0.27053268 0.8177393 0.75307443 -0.8228429 0.53819555  
## 6 0.47620095 0.26031433 -0.5686714 -0.37140720 1.3412620 0.35989384  
## V13 V14 V15 V16 V17 V18  
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005 0.20797124 0.02579058  
## 2 0.4890950 -0.1437723 0.6355581 0.4639170 -0.11480466 -0.18336127  
## 3 0.7172927 -0.1659459 2.3458649 -2.8900832 1.10996938 -0.12135931  
## 4 0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279 1.96577500  
## 5 1.3458516 -1.1196698 0.1751211 -0.4514492 -0.23703324 -0.03819479  
## 6 -0.3580907 -0.1371337 0.5176168 0.4017259 -0.05813282 0.06865315  
## V19 V20 V21 V22 V23  
## 1 0.40399296 0.25141210 -0.018306778 0.277837576 -0.11047391  
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953 0.10128802  
## 3 -2.26185709 0.52497973 0.247998153 0.771679402 0.90941226  
## 4 -1.23262197 -0.20803778 -0.108300452 0.005273597 -0.19032052  
## 5 0.80348692 0.40854236 -0.009430697 0.798278495 -0.13745808  
## 6 -0.03319379 0.08496767 -0.208253515 -0.559824796 -0.02639767  
## V24 V25 V26 V27 V28 Amount Class  
## 1 0.06692808 0.1285394 -0.1891148 0.133558377 -0.02105305 149.62 0  
## 2 -0.33984648 0.1671704 0.1258945 -0.008983099 0.01472417 2.69 0  
## 3 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66 0  
## 4 -1.17557533 0.6473760 -0.2219288 0.062722849 0.06145763 123.50 0  
## 5 0.14126698 -0.2060096 0.5022922 0.219422230 0.21515315 69.99 0  
## 6 -0.37142658 -0.2327938 0.1059148 0.253844225 0.08108026 3.67 0

summary(credit)

## 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   
## Min. :0.000000   
## 1st Qu.:0.000000   
## Median :0.000000   
## Mean :0.001728   
## 3rd Qu.:0.000000   
## Max. :1.000000

str(credit)

## 'data.frame': 284807 obs. of 31 variables:  
## $ Time : num 0 0 1 1 2 2 4 7 7 9 ...  
## $ V1 : num -1.36 1.192 -1.358 -0.966 -1.158 ...  
## $ V2 : num -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...  
## $ V3 : num 2.536 0.166 1.773 1.793 1.549 ...  
## $ V4 : num 1.378 0.448 0.38 -0.863 0.403 ...  
## $ V5 : num -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...  
## $ V6 : num 0.4624 -0.0824 1.8005 1.2472 0.0959 ...  
## $ V7 : num 0.2396 -0.0788 0.7915 0.2376 0.5929 ...  
## $ V8 : num 0.0987 0.0851 0.2477 0.3774 -0.2705 ...  
## $ V9 : num 0.364 -0.255 -1.515 -1.387 0.818 ...  
## $ V10 : num 0.0908 -0.167 0.2076 -0.055 0.7531 ...  
## $ V11 : num -0.552 1.613 0.625 -0.226 -0.823 ...  
## $ V12 : num -0.6178 1.0652 0.0661 0.1782 0.5382 ...  
## $ V13 : num -0.991 0.489 0.717 0.508 1.346 ...  
## $ V14 : num -0.311 -0.144 -0.166 -0.288 -1.12 ...  
## $ V15 : num 1.468 0.636 2.346 -0.631 0.175 ...  
## $ V16 : num -0.47 0.464 -2.89 -1.06 -0.451 ...  
## $ V17 : num 0.208 -0.115 1.11 -0.684 -0.237 ...  
## $ V18 : num 0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...  
## $ V19 : num 0.404 -0.146 -2.262 -1.233 0.803 ...  
## $ V20 : num 0.2514 -0.0691 0.525 -0.208 0.4085 ...  
## $ V21 : num -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...  
## $ V22 : num 0.27784 -0.63867 0.77168 0.00527 0.79828 ...  
## $ V23 : num -0.11 0.101 0.909 -0.19 -0.137 ...  
## $ V24 : num 0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...  
## $ V25 : num 0.129 0.167 -0.328 0.647 -0.206 ...  
## $ V26 : num -0.189 0.126 -0.139 -0.222 0.502 ...  
## $ V27 : num 0.13356 -0.00898 -0.05535 0.06272 0.21942 ...  
## $ V28 : num -0.0211 0.0147 -0.0598 0.0615 0.2152 ...  
## $ Amount: num 149.62 2.69 378.66 123.5 69.99 ...  
## $ Class : int 0 0 0 0 0 0 0 0 0 0 ...

Comments:

from str()function and Kaggle wedpage, we see that “Time” and “Amount” are the actual variables, and there are 28 variables which are the principal components of the actual variables, plus one class that is 31 variables in total

## Data cleaning

#Check for missing values  
sum(is.na(credit))

## [1] 0

## Data pre-processing

Comments: - Not needed becuase our model is discrete model(built in featur selection and discretizations) - which only partitions data(invariant to scale)

attach(credit)  
table(Class) # our data is imbalanced(1 in almost 600 transactions are fraud)

## Class  
## 0 1   
## 284315 492

Comments:

first try out without any balacing techniques(if not satisfied we will go into one undersampling or oversampling techniques)

## Decision Tree

library(tree) # to construct classification tree  
head( Class )

## [1] 0 0 0 0 0 0

Comments:

Class is a binary variable

fraud = ifelse( Class<=0.5,"No","Yes" )

Comments: we used ifels()function to create a variable, called fraud, which takes on a value of Yes if the class variable exceeds 0.5, and takes on a value of No otherwise.

credit = data.frame(credit,fraud) #to merge fraud with the rest of the credit data.  
head(credit)

## Time V1 V2 V3 V4 V5 V6  
## 1 0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077 0.46238778  
## 2 0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -0.08236081  
## 3 1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813 1.80049938  
## 4 1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888 1.24720317  
## 5 2 -1.1582331 0.87773676 1.5487178 0.4030339 -0.40719338 0.09592146  
## 6 2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -0.02972755  
## V7 V8 V9 V10 V11 V12  
## 1 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -0.61780086  
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267 1.06523531  
## 3 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015 0.06608369  
## 4 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873 0.17822823  
## 5 0.59294075 -0.27053268 0.8177393 0.75307443 -0.8228429 0.53819555  
## 6 0.47620095 0.26031433 -0.5686714 -0.37140720 1.3412620 0.35989384  
## V13 V14 V15 V16 V17 V18  
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005 0.20797124 0.02579058  
## 2 0.4890950 -0.1437723 0.6355581 0.4639170 -0.11480466 -0.18336127  
## 3 0.7172927 -0.1659459 2.3458649 -2.8900832 1.10996938 -0.12135931  
## 4 0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279 1.96577500  
## 5 1.3458516 -1.1196698 0.1751211 -0.4514492 -0.23703324 -0.03819479  
## 6 -0.3580907 -0.1371337 0.5176168 0.4017259 -0.05813282 0.06865315  
## V19 V20 V21 V22 V23  
## 1 0.40399296 0.25141210 -0.018306778 0.277837576 -0.11047391  
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953 0.10128802  
## 3 -2.26185709 0.52497973 0.247998153 0.771679402 0.90941226  
## 4 -1.23262197 -0.20803778 -0.108300452 0.005273597 -0.19032052  
## 5 0.80348692 0.40854236 -0.009430697 0.798278495 -0.13745808  
## 6 -0.03319379 0.08496767 -0.208253515 -0.559824796 -0.02639767  
## V24 V25 V26 V27 V28 Amount Class  
## 1 0.06692808 0.1285394 -0.1891148 0.133558377 -0.02105305 149.62 0  
## 2 -0.33984648 0.1671704 0.1258945 -0.008983099 0.01472417 2.69 0  
## 3 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66 0  
## 4 -1.17557533 0.6473760 -0.2219288 0.062722849 0.06145763 123.50 0  
## 5 0.14126698 -0.2060096 0.5022922 0.219422230 0.21515315 69.99 0  
## 6 -0.37142658 -0.2327938 0.1059148 0.253844225 0.08108026 3.67 0  
## fraud  
## 1 No  
## 2 No  
## 3 No  
## 4 No  
## 5 No  
## 6 No

## Split data set into train and test data

Now we will creat a training and testing set, in order to grow the tree on the training set, and evaluate its perfomance on the test set.

set.seed(2)  
train = sample(nrow(credit), nrow(credit) \* 0.80)   
#to split the observations into a training and a testing set  
  
tree.credit = tree( fraud~.- Class ,credit, subset = train)

Comments:

* we ﬁt a classiﬁcation tree in order to predict fraud using all variables but Class
* we build the tree using the training set

summary(tree.credit)

##   
## Classification tree:  
## tree(formula = fraud ~ . - Class, data = credit, subset = train)  
## Variables actually used in tree construction:  
## [1] "V17" "V10" "V14"  
## Number of terminal nodes: 6   
## Residual mean deviance: 0.005895 = 1343 / 227800   
## Misclassification error rate: 0.0005793 = 132 / 227845

# the training error rate is 5%

Comments:

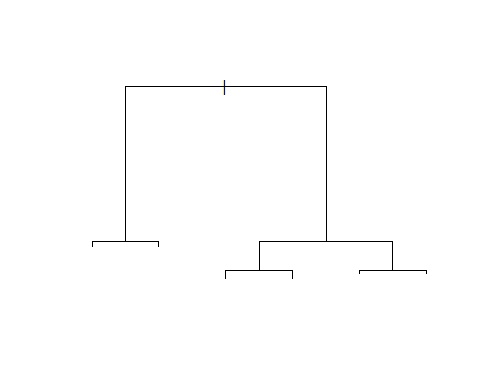
summary() function lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate.

tree.credit

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 227845 5595.00 No ( 0.9983410 0.0016590 )   
## 2) V17 < -2.5113 372 437.00 Yes ( 0.2741935 0.7258065 )   
## 4) V10 < -1.37876 320 293.70 Yes ( 0.1718750 0.8281250 ) \*  
## 5) V10 > -1.37876 52 32.92 No ( 0.9038462 0.0961538 ) \*  
## 3) V17 > -2.5113 227473 1869.00 No ( 0.9995252 0.0004748 )   
## 6) V14 < -4.2192 310 295.90 No ( 0.8161290 0.1838710 )   
## 12) V10 < -1.85864 68 74.20 Yes ( 0.2352941 0.7647059 ) \*  
## 13) V10 > -1.85864 242 48.69 No ( 0.9793388 0.0206612 ) \*  
## 7) V14 > -4.2192 227163 959.00 No ( 0.9997755 0.0002245 )   
## 14) V17 < 1.4182 218617 611.40 No ( 0.9998582 0.0001418 ) \*  
## 15) V17 > 1.4182 8546 282.30 No ( 0.9976597 0.0023403 ) \*

Comments: when we just type tree.credit, R prints output corresponding to each branch of the tree, displays the split criterion, the number of observations in that branch, the deviance, the overall prediction for the branch (Yes or No), and the fraction of observations in that branch that take on values of Yes and No.

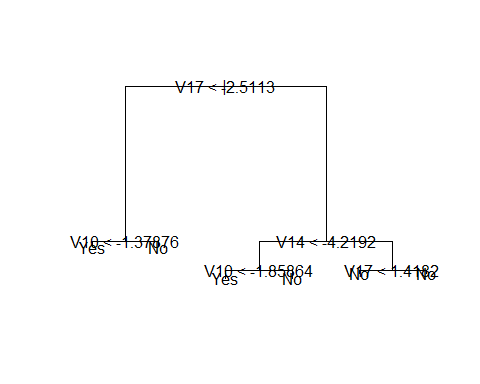
plot(tree.credit) #to display the tree structure



text(tree.credit,plot.new = plot(tree.credit),pretty = 0) # to display the node labels

## Warning in text.default(xy$x[ind], xy$y[ind] + 0.5 \* charht, rows[ind], :  
## "plot.new" is not a graphical parameter

## Warning in text.default(xy$x[leaves], xy$y[leaves] - 0.5 \* charht, labels =  
## stat, : "plot.new" is not a graphical parameter



# evaluate its performance on the test data  
  
tree.pred = predict(tree.credit,credit[-train,],type = "class")  
  
# type="class" instructs R to return the actual class prediction

with(credit[-train,],table(tree.pred,fraud))

## fraud  
## tree.pred No Yes  
## No 56830 28  
## Yes 18 86

Comments:

* (56830+86)/56962 #0.9991924
* This approach leads to correct predictions for around 99.92% of the locations in the test data set.

## Pruning

Comments: pruning the tree might lead to improved results

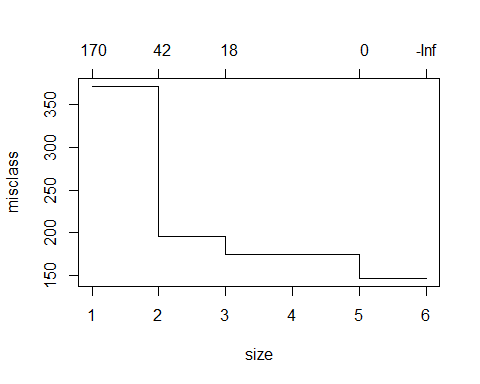
cv.credit = cv.tree(tree.credit,FUN = prune.misclass)

Comments: cv.tree()function will performs cross-validation in order to determine the optimal level of tree complexity

cv.credit #will reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost-complexity parameter used

## $size  
## [1] 6 5 3 2 1  
##   
## $dev  
## [1] 147 147 175 196 371  
##   
## $k  
## [1] -Inf 0 18 42 168  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

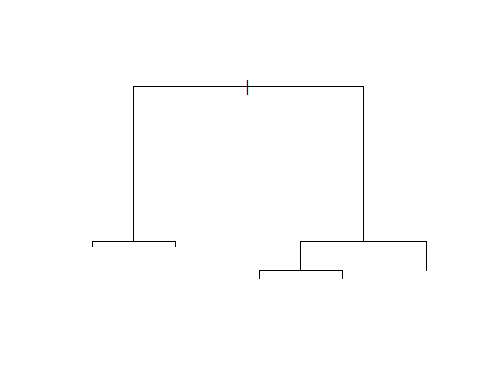
# plot the error rate as a function of both size and k  
plot(cv.credit)



Comments: now we will apply the prune.misclass() function in order to prune the tree to prune.obtain the five-node tree.

prune.credit = prune.misclass(tree.credit,best =5 )

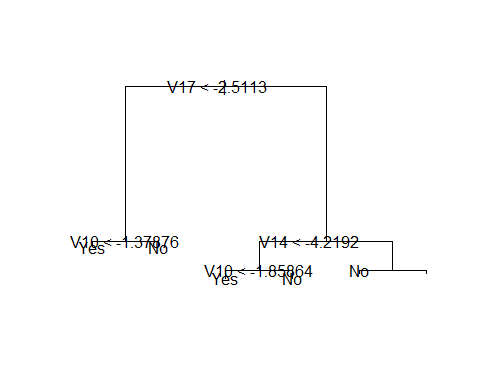
plot(prune.credit)



text(prune.credit,plot.new = plot(tree.credit),pretty = 0)

## Warning in text.default(xy$x[ind], xy$y[ind] + 0.5 \* charht, rows[ind], :  
## "plot.new" is not a graphical parameter

## Warning in text.default(xy$x[leaves], xy$y[leaves] - 0.5 \* charht, labels =  
## stat, : "plot.new" is not a graphical parameter



Comments: Once again, we will apply the predict() function

prune.pred = predict(prune.credit,credit[-train,],type = "class")

with(credit[-train,],table(prune.pred,fraud))

## fraud  
## prune.pred No Yes  
## No 56830 28  
## Yes 18 86

Comments:

pruning process produced a more interpretable tree, but it has not improved the classiﬁcation accuracy

library(caret) # for accuracy measures

## Loading required package: lattice

## Loading required package: ggplot2

fraud = as.factor(fraud)  
confusionMatrix(tree.pred,fraud[-train])

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 56830 28  
## Yes 18 86  
##   
## Accuracy : 0.9992   
## 95% CI : (0.9989, 0.9994)  
## No Information Rate : 0.998   
## P-Value [Acc > NIR] : 3.715e-13   
##   
## Kappa : 0.7886   
##   
## Mcnemar's Test P-Value : 0.1845   
##   
## Sensitivity : 0.9997   
## Specificity : 0.7544   
## Pos Pred Value : 0.9995   
## Neg Pred Value : 0.8269   
## Prevalence : 0.9980   
## Detection Rate : 0.9977   
## Detection Prevalence : 0.9982   
## Balanced Accuracy : 0.8770   
##   
## 'Positive' Class : No   
##

Comments:

The decision tree gives us a great accuracy, we are almost an 99% accurace

The model has learned, what the best questions to ask about the input data are, and can respond with a prediction for credit card fraud class

At this point,I tried decision tree and I had prune the tree, but it has not improved the classiﬁcation accuracy so, I will try different models in the following step.