CreditCard Status Classification using C5.0 Algorithm

```r
#libraries
library(C50)
```

```
## Warning: package 'C50' was built under R version 3.4.3
```

```r
## data reading
data = read.csv('data/creditcard.csv')
print(head(data))
```

```
##   b X30.83    X0 u g w v X1.25 t t.1 X01 f g.1 X00202  X0.1 X.
## 1 a  58.67 4.460 u g q h  3.04 t   t   6 f   g  00043   560  +
## 2 a  24.50 0.500 u g q h  1.50 t   f   0 f   g  00280   824  +
## 3 b  27.83 1.540 u g w v  3.75 t   t   5 t   g  00100     3  +
## 4 b  20.17 5.625 u g w v  1.71 t   f   0 f   s  00120     0  +
## 5 b  32.08 4.000 u g m v  2.50 t   f   0 t   g  00360     0  +
## 6 b  33.17 1.040 u g r h  6.50 t   f   0 t   g  00164 31285  +
```

```r
## data processing [The dataset contains categorical varibles, need categorical to numerical conversion]

#Categorial Variables to Numeric
must_convert<-sapply(data,is.factor)       # logical vector telling if a variable needs to be displayed
M2<-sapply(data[,must_convert],unclass)    # data.frame of all categorical variables now displayed as n
credit_card<-cbind(data[,!must_convert],M2)
```

```r
##Partitioning the data into traning and test by shuffing
#https://stackoverflow.com/questions/17200114/how-to-split-data-into-training-testing-sets-using-sample
set.seed(101) # Set Seed so that same sample can be reproduced in future also
# Now Selecting 75% of data as sample from total 'n' rows of the data
sample <- sample.int(n = nrow(credit_card), size = floor(.75*nrow(credit_card)), replace = F)
train <- credit_card[sample, ]
test  <- credit_card[-sample, ]

column_names = c("X0","X1.25","X01","X0.1","b","X30.83", "u", "g","w","v","t","t.1","f","g.1","X00202")
```

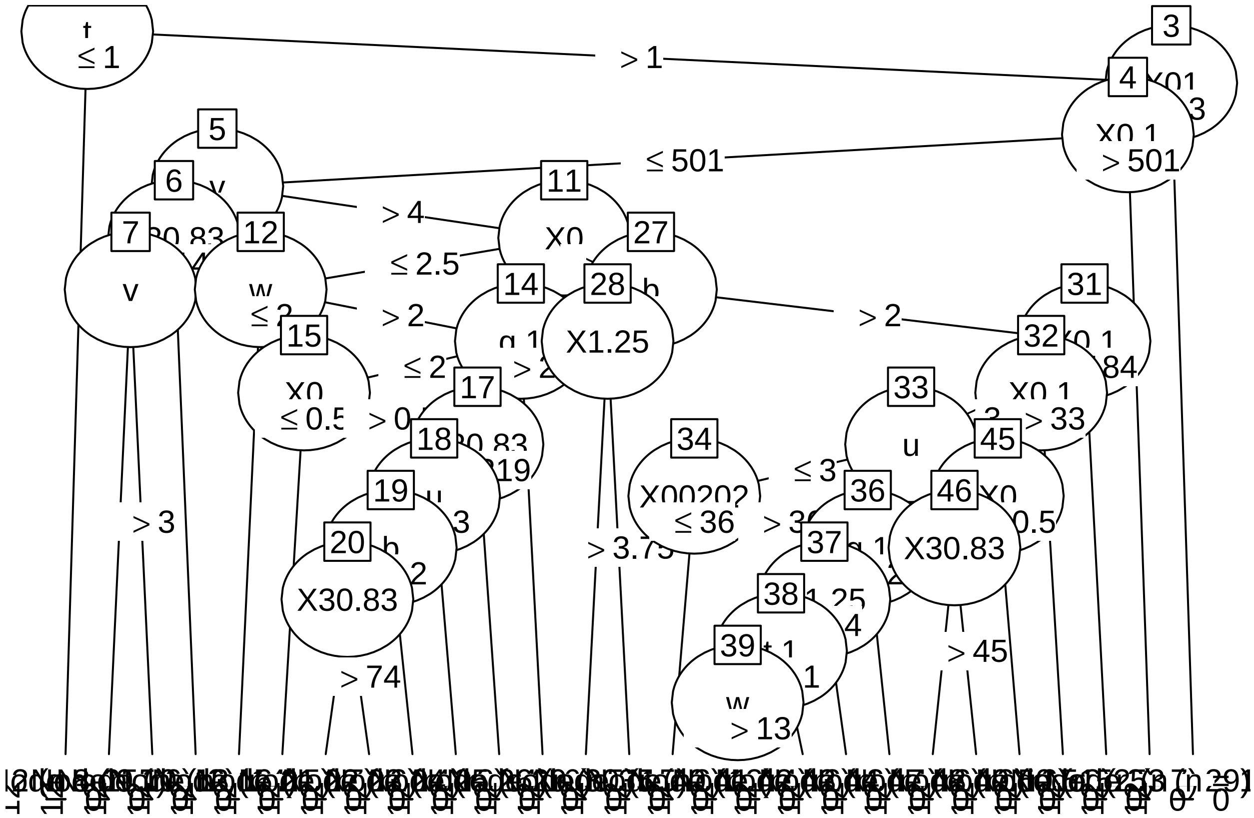
```r
##Classification using c5.0 algorithm
tree_mod <- C5.0(x =train[column_names], y = as.factor(train$X.))
print(summary(tree_mod))
```

```
##
## Call:
## C5.0.default(x = train[column_names], y = as.factor(train$X.))
##
##
## C5.0 [Release 2.07 GPL Edition]      Wed Apr 18 16:33:46 2018
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 516 cases (16 attributes) from undefined.data
##
## Decision tree:
##
## t <= 1: 1 (255/18)
```

```
## t > 1:
## :...X01 > 3: 2 (104/3)
##     X01 <= 3:
##     :...X0.1 > 501: 2 (29/1)
##         X0.1 <= 501:
##         :...v <= 4:
##             :...X30.83 > 147: 1 (14/2)
##             :   X30.83 <= 147:
##             :   :...v <= 3: 2 (6/1)
##             :       v > 3: 1 (2)
##             v > 4:
##             :...X0 <= 2.5:
##                 :...w <= 2: 1 (5)
##                 :   w > 2:
##                 :   :...g.1 > 2: 1 (5/1)
##                 :       g.1 <= 2:
##                 :       :...X0 <= 0.5: 2 (7)
##                 :           X0 > 0.5:
##                 :           :...X30.83 > 219: 1 (6)
##                 :               X30.83 <= 219:
##                 :               :...u > 3: 1 (4/1)
##                 :                   u <= 3:
##                 :                   :...b > 2: 2 (8/1)
##                 :                       b <= 2:
##                 :                       :...X30.83 <= 74: 2 (2)
##                 :                           X30.83 > 74: 1 (4)
##                 X0 > 2.5:
##                 :...b <= 2:
##                     :...X1.25 <= 3.75: 2 (15)
##                     :   X1.25 > 3.75: 1 (3/1)
##                     b > 2:
##                     :...X0.1 > 184: 2 (7)
##                         X0.1 <= 184:
##                         :...X0.1 > 33: 1 (3)
##                             X0.1 <= 33:
##                             :...u > 3:
##                                 :...X0 > 10.5: 1 (2)
##                                 :   X0 <= 10.5:
##                                 :   :...X30.83 <= 45: 1 (2)
##                                 :       X30.83 > 45: 2 (6)
##                                 u <= 3:
##                                 :...X00202 <= 36: 2 (12)
##                                     X00202 > 36:
##                                     :...g.1 > 2: 2 (2)
##                                         g.1 <= 2:
##                                         :...X1.25 > 4: 2 (4)
##                                             X1.25 <= 4:
##                                             :...t.1 > 1: 1 (3)
##                                                 t.1 <= 1:
##                                                 :...w <= 13: 2 (4/1)
##                                                     w > 13: 1 (2)
##
##
## Evaluation on training data (516 cases):
```

```
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##      27   30( 5.8%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     287     7    (a): class 1
##      23   199    (b): class 2
##
##
##  Attribute usage:
##
##  100.00% t
##   50.58% X01
##   30.43% X0.1
##   24.81% v
##   20.54% X0
##   15.31% b
##   10.66% u
##   10.47% X30.83
##    9.88% g.1
##    9.11% w
##    6.01% X1.25
##    5.23% X00202
##    1.74% t.1
##
##
## Time: 0.0 secs
```

```
plot(tree_mod)
```



```r
##Prediction
predicted=predict(tree_mod, newdata = test[, column_names])

#print(output_labels)
cm = as.matrix(table(Actual = test$X., Predicted = predicted)) # create the confusion matrix
print(cm)
```

```
##       Predicted
## Actual  1  2
##      1 78 11
##      2 17 67
```

```r
##Evaluting the model performace
 n = sum(cm) # number of instances
 nc = nrow(cm) # number of classes
 diag = diag(cm) # number of correctly classified instances per class
 rowsums = apply(cm, 1, sum) # number of instances per class
 colsums = apply(cm, 2, sum) # number of predictions per class
 p = rowsums / n # distribution of instances over the actual classes
 q = colsums / n # distribution of instances over the predicted classes

#Accuracy
 accuracy = sum(diag) / n
 print("Accuracy: ")
```

```
## [1] "Accuracy: "
```

```r
 print(accuracy)
```

```
## [1] 0.8381503
```

```r
#Per-class Precision, Recall, and F-1
precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)
evalution_stat=data.frame(precision, recall, f1)
print("Evalution Statistics")
```

```
## [1] "Evalution Statistics"
```

```r
print(evalution_stat)
```

```
##   precision    recall        f1
## 1 0.8210526 0.8764045 0.8478261
## 2 0.8589744 0.7976190 0.8271605
```

```r
##Macro-averaged Metrics
macroPrecision = mean(precision)
macroRecall = mean(recall)
macroF1 = mean(f1)
evalution_stat=data.frame(macroPrecision, macroRecall, macroF1)
print(macroPrecision)
```

```
## [1] 0.8400135
```

```r
print("Macro Evalution Statistics")
```

```
## [1] "Macro Evalution Statistics"
```

```r
print(evalution_stat)
```

```
##   macroPrecision macroRecall   macroF1
## 1      0.8400135   0.8370118 0.8374933
```