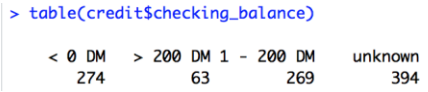
ALY 6050: Data Mining Applications – Week 3 Assignment

Data Exploration

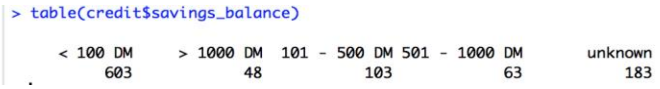
Risky bank loans data set (credit.csv) is obtained from a credit agency in Germany and consists of 1000 rows and 21 columns about examples of loans and information about applicants. Decision tree is a tree-like model that classifies future observation based on a set of decision rules. Banking industry uses decision trees to identify risky loans and evaluate future applicants based on factors that predict higher risk of default. To review this credit scoring we first read the credit.csv file and then install the necessary packages:

1. C.50: fits classification tree models using the C5.0 algorithm.
2. Gmodels: consists of various functions to manipulate tree models.

After applying str() we find that the file consists of information on 1000 loan applications with information of checking balance, savings balance, months of loan duration, length of employment, installment rate, residence history, property, age etc. in factor or integer data type.

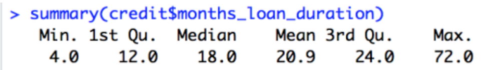


```
> table(credit$checking_balance)

    < 0 DM   > 200 DM 1 - 200 DM    unknown 
       274         63        269        394 
```

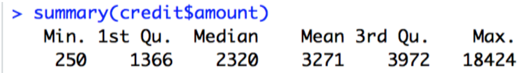


```
> table(credit$savings_balance)

     < 100 DM     > 1000 DM  101 - 500 DM 501 - 1000 DM       unknown 
          603            48           103            63           183 
```

*Fig: table() output for **factor** data types: checking_balance and savings_balance*



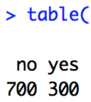
```
> summary(credit$months_loan_duration)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    4.0    12.0    18.0    20.9    24.0    72.0 
```



```
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    250    1366    2320    3271    3972   18424 
```

*Fig: table() output for **integer** data types: months_loan_duration and amount*

Table() for factor data type gives us the total number for checking and savings balance segregated in 4 levels of Deutsche Marks i.e 274 for <0DM, 63 for >200DM -1 etc. and 603 for <100DM, 48 for >1000DM etc. respectively. Similarly, we apply summary() to evaluate integer data type details which tell us for example, that the mean of loan months duration is 20.9 months and mean for the loan amount is $3271.



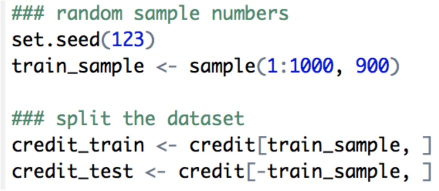
```
> table(credit$default)

 no yes
700 300
```

Fig: default or repayment status: yes-300 and no-700

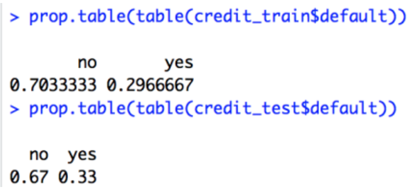
The **default** column is consisting of **1,2** is converted to **no, yes** respectively using the factor() function which is also our target feature for this analysis. It indicates the applicant's default status (yes or no) which means whether the applicant was able to pay back the loan with all interests or not.

Data Preparation

We split the data in 90% training and 10% testing to build a decision tree to evaluate the performance of the model which provides us with 100 new records to simulate applicants. We use the sample() function to randomly subset set of records and set.seed() ensures that the results can be replicated later if desired by selecting the same random sample.



```
### random sample numbers
set.seed(123)
train_sample <- sample(1:1000, 900)

### split the dataset
credit_train <- credit[train_sample, ]
credit_test <- credit[-train_sample, ]
```



```
> prop.table(table(credit_train$default))

       no       yes
0.7033333 0.2966667
> prop.table(table(credit_test$default))

  no  yes
0.67 0.33
```

Fig: Split in training and testing data

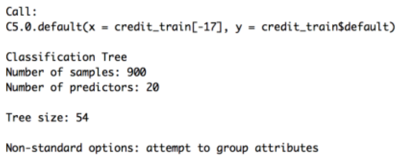
Fig: Characteristics of default in train and test data

train_sample selects 900 out of 1000 values randomly and this is assigned to credit_train data set while, the leftover 100 are assigned to credit_test data set. (-train_sample indicates excluding those 900 values i.e 100!)

The proportion of yes and no in training and testing datasets are 0.70 and 0.67 for no whereas, 0.29 and 0.33 for yes respectively which shows that the data splitting was even.

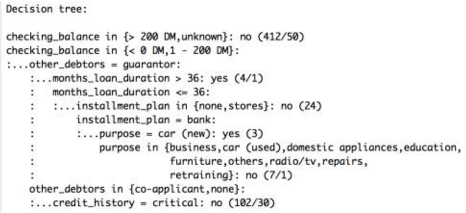
a. Credit Model

We implement a decision tree (credit_model) on train data by applying C5.0 function and passing parameters credit_train data excluding 17th column or default as x and credit_train data default column (target feature) as y.



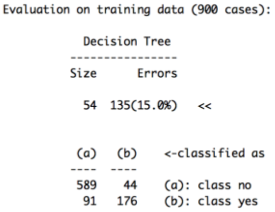
```
Call:
C5.0.default(x = credit_train[-17], y = credit_train$default)

Classification Tree
Number of samples: 900
Number of predictors: 20

Tree size: 54

Non-standard options: attempt to group attributes
```

Fig: credit model



```
Decision tree:

checking_balance in {> 200 DM,unknown}: no (412/50)
checking_balance in {< 0 DM,1 - 200 DM}:
:...other_debtors = guarantor:
    :...months_loan_duration > 36: yes (4/1)
    :   months_loan_duration <= 36:
    :   :...installment_plan in {none,stores}: no (24)
    :       installment_plan = bank:
    :       :...purpose = car (new): yes (3)
    :           purpose in {business,car (used),domestic appliances,education,
    :                       furniture,others,radio/tv,repairs,
    :                       retraining}: no (7/1)
    other_debtors in {co-applicant,none}:
    :...credit_history = critical: no (102/30)
```

Fig: Summary of credit model

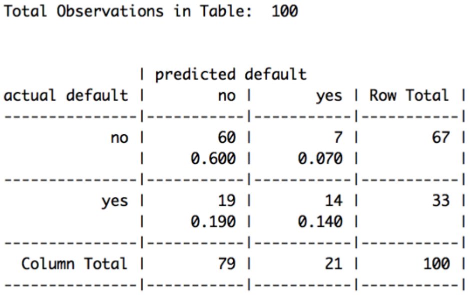


```
Evaluation on training data (900 cases):

	    Decision Tree
	  ----------------
	  Size      Errors

	    54  135(15.0%)   <<


	   (a)   (b)    <-classified as
	  ----  ----
	   589    44    (a): class no
	    91   176    (b): class yes
```

Fig: Confusion matrix

The function call C5.0 generated a credit_model classification resulting in a big tree of size 54 with number of samples = 900 and number of features or predictors = 20.

After applying the summary() function on the credit model we get a decision tree characteristics for example, if the checking_balance >200DM will be classified as **not** likely to default. The number 412/50 shows that 412 examples reached to that decision however, 50 were classified incorrectly as likely to default. Further, for checking_balance < 0DM and 1-200DM, the tree was split up further based on other factors such months_loan_duration, installment_plan, purpose etc until 54 steps. The evaluation of credit model displays a confusion matrix which summarizes the model's incorrectly classified records in the training data. A total of 135 values out of 900 were incorrectly classified resulting to 15% error rate. Here, 44 actual 'no' values were classified as 'yes' and 91 actual 'yes' values were classified as 'no'.

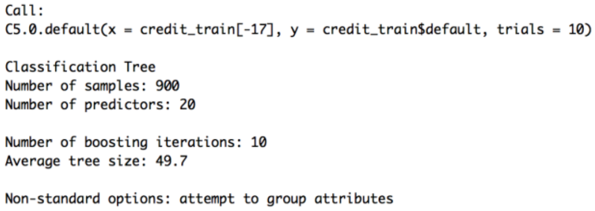


```
Total Observations in Table:  100


               | predicted default 
actual default |        no |       yes | Row Total | 
---------------|-----------|-----------|-----------|
            no |        60 |         7 |        67 | 
               |     0.600 |     0.070 |           | 
---------------|-----------|-----------|-----------|
           yes |        19 |        14 |        33 | 
               |     0.190 |     0.140 |           | 
---------------|-----------|-----------|-----------|
  Column Total |        79 |        21 |       100 | 
---------------|-----------|-----------|-----------|
```

Predicting credit_test evaluation, our model correctly predicted 60 as not defaulted and 14 as defaulted resulting in an accuracy of 74% and an error rate of 26%. However, only 14 were correctly predicted default out of 33.

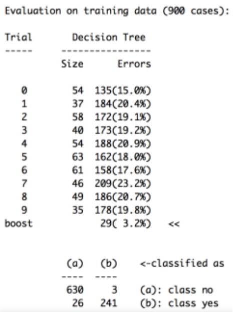
b. Improving model performance

We now implement a decision tree (credit_boost10) model on test data by applying the C5.0 function and passing credit_train excluding 17th column as x and credit_train default column as y along with an extra parameter trails indicating the number of separate decision trees to use in the boosted team. The trials parameter sets an upper limit where the algorithm stops adding trees if it recognizes that additional trials do not seem to be improving the accuracy.



```
Call:
C5.0.default(x = credit_train[-17], y = credit_train$default, trials = 10)

Classification Tree
Number of samples: 900
Number of predictors: 20

Number of boosting iterations: 10
Average tree size: 49.7

Non-standard options: attempt to group attributes
```

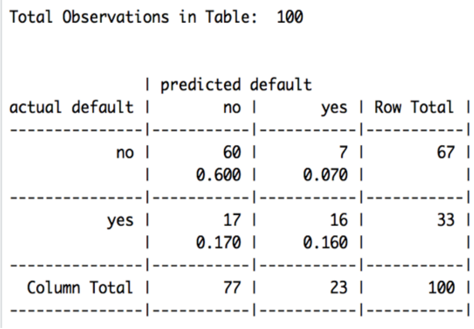
Fig: credit boost



```
Evaluation on training data (900 cases):

Trial       Decision Tree
-----     ----------------
           Size      Errors

   0         54  135(15.0%)
   1         37  184(20.4%)
   2         58  172(19.1%)
   3         40  173(19.2%)
   4         54  188(20.9%)
   5         63  162(18.0%)
   6         61  158(17.6%)
   7         46  209(23.2%)
   8         49  186(20.7%)
   9         35  178(19.8%)
boost             29( 3.2%)   <<


           (a)   (b)    <-classified as
          ----  ----
           630     3    (a): class no
            26   241    (b): class yes
```

Fig: Summary of credit boost and confusion matrix

After calling the credit_boost function we notice that for 900 samples and 20 predictors or features, in 10 iterations the average tree size has shrunk to 49.7. The summary() function on credit_boost results into a confusion matrix showing that 3 values were defaulted as 'yes' instead of 'no' and 26 values were defaulted as 'no' instead of 'yes'. So, a total of 29 values were incorrectly classified among 900 samples making the total error rate 3% only. Overall, improvement in error was noted as 12% which is compared to before and after boosting.



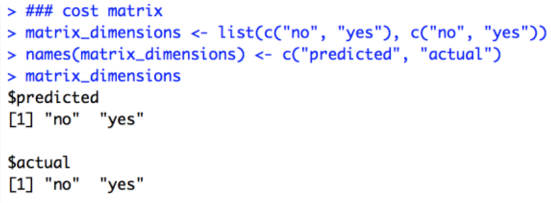
```
Total Observations in Table:  100


               | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        60 |         7 |        67 |
               |     0.600 |     0.070 |           |
---------------|-----------|-----------|-----------|
           yes |        17 |        16 |        33 |
               |     0.170 |     0.160 |           |
---------------|-----------|-----------|-----------|
  Column Total |        77 |        23 |       100 |
---------------|-----------|-----------|-----------|
```

Fig: test data evaluation matrix

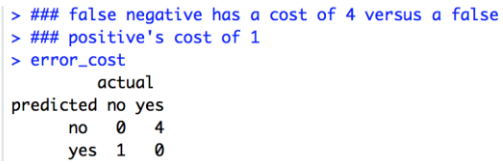
Predicting the effect of boosting on credit_test data, we can see that the total error rate has dropped from 26% before boosting to 24% after boosting which means there 2% of reduce in error rate. Therefore, 7 and 17 values were incorrectly predicted as 'yes' and 'no' respectively.

c. Error Rate

Approving loans of people who are most likely to default can be highly expensive for the banks therefore, to reduce these defaulters the key is to reduce the false negatives by rejecting the borderline applicants that are most likely to be defaulters. We use the C5.0 algorithm to assign penalties for avoiding such costly mistakes.



```
> ### cost matrix
> matrix_dimensions <- list(c("no", "yes"), c("no", "yes"))
> names(matrix_dimensions) <- c("predicted", "actual")
> matrix_dimensions
$predicted
[1] "no"  "yes"

$actual
[1] "no"  "yes"
```

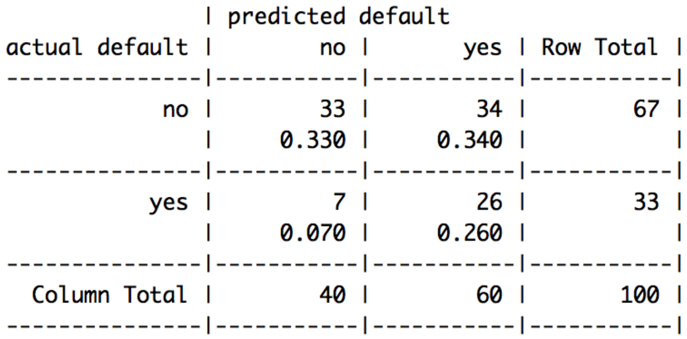


```
> ### false negative has a cost of 4 versus a false
> ### positive's cost of 1
> error_cost
         actual
predicted no yes
      no   0   4
      yes  1   0
```

Fig: Cost Matrix

Fig: Error Cost

We formulate a 2x2 cost matrix with two vectors each consisting of 'no' and 'yes' and we label each one as predicted and actual to avoid confusions. Then, we assign the penalties as 0,1,4,0 for points in the matrix (1,1), (1,2), (2,1) and (2,2) respectively. There is no cost assigned when the algorithm classifies a no or yes correctly, but a false negative has a cost of 4 versus a false positive's cost of 1 implying that a loan default costs 4 times as much as a missed opportunity.



```
               | predicted default 
actual default |        no |       yes | Row Total | 
---------------|-----------|-----------|-----------|
            no |        33 |        34 |        67 | 
               |     0.330 |     0.340 |           | 
---------------|-----------|-----------|-----------|
           yes |         7 |        26 |        33 | 
               |     0.070 |     0.260 |           | 
---------------|-----------|-----------|-----------|
  Column Total |        40 |        60 |       100 | 
---------------|-----------|-----------|-----------|
```

Fig: test data evaluation matrix

Compared to the boosted model, this model makes has an error rate of 41% which is 15% more. The false negatives were reduced at the expense of increase in false positives which is not wise.

d. Plots:

We use the ggplot() function to make histogram plots between credit and savings balance, checking balance, loan duration and loan amount.

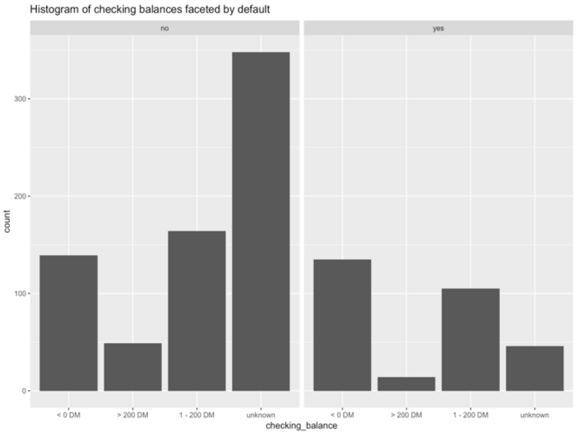


Fig: Histogram of checking balances by default

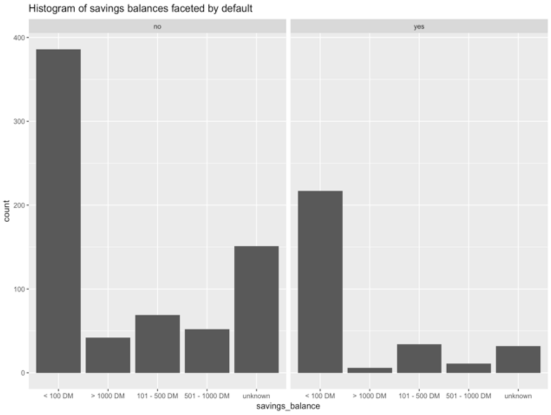


Fig: Histogram of savings balances by default

In histogram 1, number of people who have checking balance >200DM are least likely to be loan defaulters and <0DM are most likely to be defaulters. However, high number of unknown and 1-200DM are not likely to be defaulters. It is a statistical anomaly that people with unknown checking balances were least likely to default.

In histogram 2, it is clear that people with <100DM in their savings balance were most likely to be defaulters. However, it is strange to see that <100DM were also most likely to not be loan defaulters.

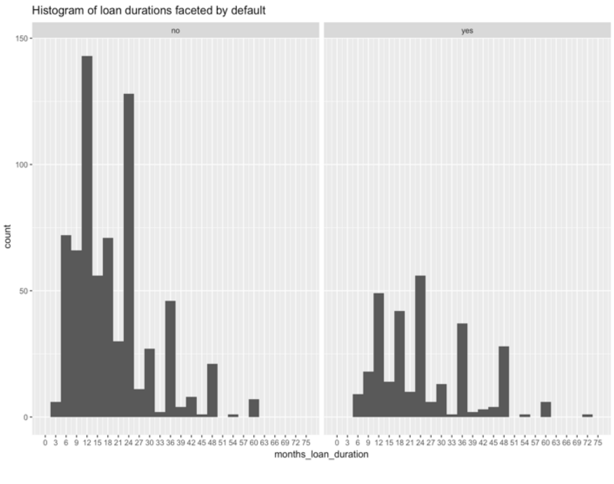


Fig: Histogram of loan durations by default

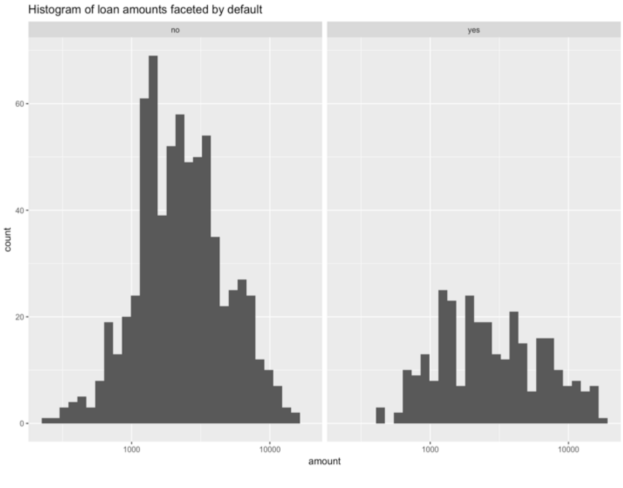


Fig: Histogram of loan amounts faceted by default

Histogram 3 of loan duration shows that for loan defaulters the loan repayment takes longer months than those who are not defaulters. The ones who are not defaulters are most likely to pay off their loans in upto 24 months however, defaulters may take 36 months.

Histogram 4 of loan amounts tells us that the ones who were not defaulters were allotted large amounts of loan after careful inspection of all parameters. On the other hand, the defaulters were comparatively allotted less amount of money after carefully inspecting their ability to pay back but still failed to repay back. Overall, most of the loans were carefully inspected and successfully repaid it back.

References:

1. https://cran.r-project.org/web/packages/C50/C50.pdf

2. https://www.rdocumentation.org/packages/gmodels/versions/2.0.7

3. Eckerson, W. W. (2012). *Secrets of analytical leaders*: Westfield, NJ: Technics Publication