The University of Akron

Management Department

Advanced Data Analytics 6500-663

Business Report

Identifying risky bank loans using C5.0 decision tree algorithm

By Kirill Samaray

Introduction

To reduce any possible financial losses in the financial and banking sectors of the economy, it is crucial to identify risky loans. The latter is defined as the ones having a high likelihood of default which may in turn result in financial losses to the financial institution. With the rise of Big Data, the application of data-driven technologies and machine learning has been widely incorporated into the routine operations of companies worldwide to help with identifying and mitigating risks.

One of the algorithm examples was proven to be effective in detecting risky loans using the decision tree approach – C5.0. It is a supervised machine learning algorithm that creates a decision tree to identify combinations based on their attributes. It is widely used in different areas including finance because it is quite simple to integrate, it shows a decent accuracy, and the results of this algorithm are easy to interpret.

The main goal of this report is to study and elaborate on the C5.0 decision tree algorithm, utilize it with real-life data, and properly identify risky loans. The findings of this report can be used by financial institutions to facilitate their loan portfolios and mitigate current risks associated with loans.

Problem Statement

The economy is currently experiencing another financial crisis. Banks and other related institutions face increased risks associated with inflation, mortgages, bank loans, rate hikes, and volatility. One of the main risks lies within the borrowers being incapable of paying their debt and consequently defaulting on their loans. This in turn can lead to a domino effect because the banks may also not be capable of recovering from such losses and therefore go bankrupt. It could have an overarching effect on the economy and eventually deepen the financial crisis. Therefore, it’s of paramount importance to identify the loans that could have a high likelihood of default and prevent any losses associated with them.

Traditionally, the risk level was measured by people with the help of a simple credit history score, income levels of an individual, and other similar data. Although, this process can be very time-consuming and carry a significant level of human error. People are not always able to cover all important factors that could help in identifying the risks. Thus, a more precise and efficient approach is needed.

Using data-integrated solutions and machine learning technology could be viewed as a solution to the problem although choosing an appropriate algorithm might be a challenging task. One of the most well-known algorithms for this purpose is the C5.0 algorithm. Even though its effectiveness was tested numerous times the performance of the algorithm must be continuously monitored considering the nature of the data.

Objectives

The primary objectives of this report are:

- To evaluate the efficiency of the C5.0 decision tree algorithm in identifying risky bank loans.
- To identify the most important features that contribute to the identification of risky loans using the C5.0 decision tree algorithm.
- How to balance and penalize type I and type II errors.
- To provide recommendations for the use of the C5.0 decision tree algorithm in identifying and managing the risks associated with loan portfolios.

To achieve these objectives, the report will use a dataset of bank loans and their associated outcomes to train and test the C5.0 decision tree algorithm. The report will also conduct an analysis to identify the most critical factors that contribute to the identification of risky loans. The report will also show how to prune the resulting tree and how to improve the model performance using adaptive boosting. The findings of this report can provide valuable insights into the use of machine learning algorithms for loan classification and help banks and financial institutions to better manage their loan portfolios.

Methodology

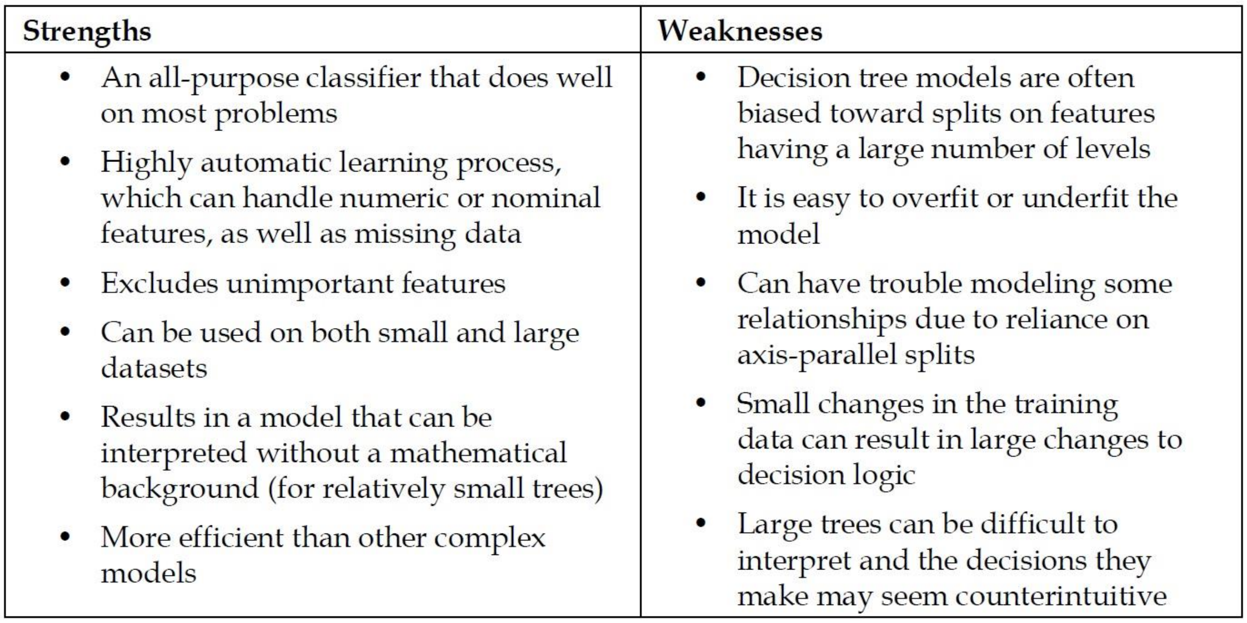
The methodology used in this report is based on the analysis of the publicly available data set "credit" and a review of the existing literature: Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8), Jason Brownlee, Information Gain, and Mutual Information for Machine Learning, October 16, 2019, Probability, Quinlan R (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Anshul Saini — Master the AdaBoost Algorithm: Guide to Implementing & Understanding AdaBoost, Published On September 15, 2021, and Last Modified On March 3rd, 2023

The following report is split into the sections:

- Step 1. Overview of the model.
- Step 2. Collecting data.
- Step 3. Exploring and preparing data.
- Step 4. Model training.
- Step 5. Model Evaluation.
- Step 6. Model Optimization.

Step 1. Overview of the model

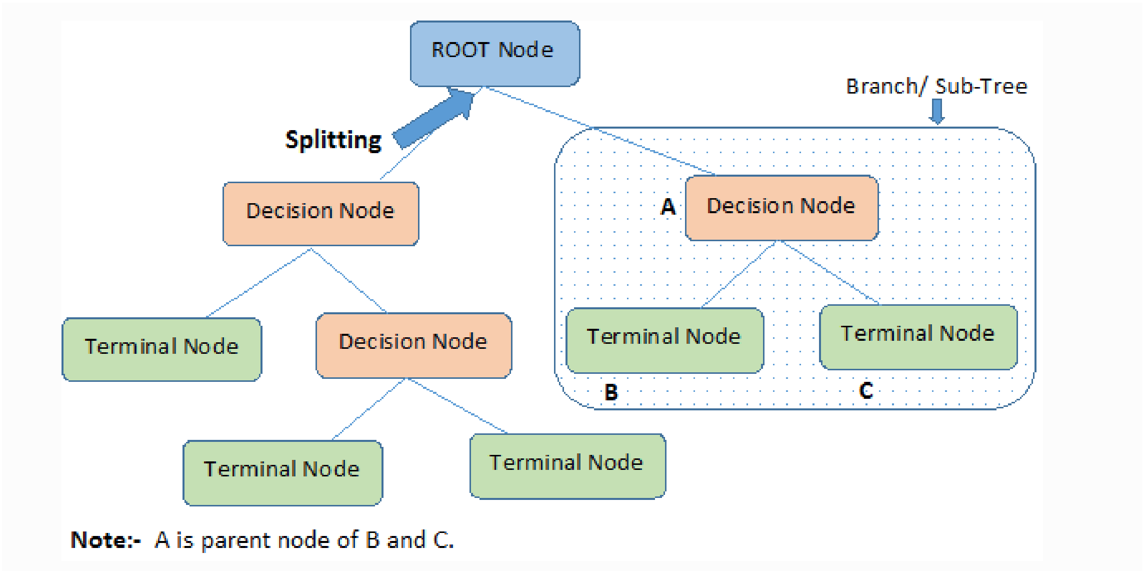
The C5.0 algorithm was created by J. Ross Quinlan as a modification of the existing C4.5 algorithm. There are also other available options included in R's packages such as J48, although the differences between them are minor and C5.0 has become the most popular one due to its simplicity and reliability over the year. Here is the list of its advantages and disadvantages:



Strengths	Weaknesses
• An all-purpose classifier that does well on most problems • Highly automatic learning process, which can handle numeric or nominal features, as well as missing data • Excludes unimportant features • Can be used on both small and large datasets • Results in a model that can be interpreted without a mathematical background (for relatively small trees) • More efficient than other complex models	• Decision tree models are often biased toward splits on features having a large number of levels • It is easy to overfit or underfit the model • Can have trouble modeling some relationships due to reliance on axis-parallel splits • Small changes in the training data can result in large changes to decision logic • Large trees can be difficult to interpret and the decisions they make may seem counterintuitive

(Lantz, 2015)

The C5.0 decision tree algorithm is a supervised machine learning algorithm used for classification tasks. It creates a decision tree to classify cases based on their features. It uses a divide-and-conquer (recursive partiotioning) approach to build the decision tree. It starts with the entire dataset and repeatedly splits the data into smaller subsets based on the information processed. The decision tree is a hierarchical structure that consists of nodes that represent attributes. It all starts with the root node and then proceeds through decision nodes which offer a choice and a subsequent decision to be made. Every new decision splits the data again and creates new branches with the potential outcomes of a new decision. In case a final decision is reached, there are also terminal nodes, which represent a terminal point for the tree and the ability to make a final decision based on information gained throughout the process. The model is widely used in many spheres such as diagnosing medical conditions, marketing studies, and credit scoring models.



(Chauhan, 2022)

The algorithm selects the attribute that maximizes the information gain, which measures how well the attribute separates the instances into different classes. **Information gain** is a concept used in the C5.0 decision tree algorithm to select the best attribute for splitting the data. It measures the reduction in entropy or impurity of the data that results from splitting the data on a particular attribute (Brownlee, 2019).

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

(Lantz, 2015)

Entropy is a measure of the randomness or uncertainty of the data. In the context of decision trees, entropy is used to measure the impurity of a node. A node with high entropy has a more random distribution of classes, while a node with low entropy has a more homogeneous distribution of classes. The goal of the C5.0 algorithm is to minimize the entropy of the nodes in the decision tree by selecting the best attribute for splitting the data.

$$\text{Entropy}(S) = \sum_{i=1}^{n} w_i \, \text{Entropy}(P_i)$$

(Lantz, 2015)

The information gain of an attribute is calculated as the difference between the entropy of the parent node and the entropy of the child nodes after splitting on the attribute. The attribute with the highest information gain is selected as the splitting criterion. Thus, the information gain measures how much information about the class labels is gained by splitting the data on a particular attribute. If an attribute has a high information gain, it means that splitting the data on that attribute results in a significant reduction in entropy and a better separation of the classes. If an attribute has a low information gain, it means that splitting the data on that attribute does not result in a significant reduction in entropy and may not be an effective splitting criterion (Brownlee, 2019).

It is also possible to prune the decision tree to reduce overfitting and improve its generalization performance. The tree can continue to grow unless one of the following conditions is met:

- All (or nearly all) of the examples at the node have the same class
- There are no remaining features to distinguish among the examples
- The tree has grown to a predefined size limit (Lantz, 2015)

Although there is also a drawback for a tree to grow indefinitely. In case it does grow to a very large extent, lots of decisions within the model may be very specific, contain too much information noise, and the model may become overfitted on the training data set. This is when the **pruning** process comes into place which means reducing the number of branches to generalize it better. There are several ways how pruning can be carried out:

Pre-pruning is a technique that stops the tree construction process before it reaches its full complexity. It involves setting a certain level when the algorithm should stop such as the minimum number of instances in a node or minimum information gain, for example. If the stopping criterion is met, the algorithm stops splitting the node and assigns the majority class of the instances in the node as the class label. Pre-pruning reduces the complexity of the decision tree and prevents overfitting, but it may result in underfitting if the tree is not deep enough to capture the underlying patterns in the data.

Post-pruning is a technique that prunes or trims the decision tree after it is built. It involves evaluating the performance of the tree on a validation set or using a statistical test to determine which branches or nodes should be pruned. The algorithm evaluates the impact of pruning a node on the accuracy of the tree and removes the node if it improves the accuracy or does not significantly affect it. Post-pruning can be more effective than pre-pruning as it allows the tree to grow to its full depth and capture the underlying patterns in the data, while still preventing overfitting.

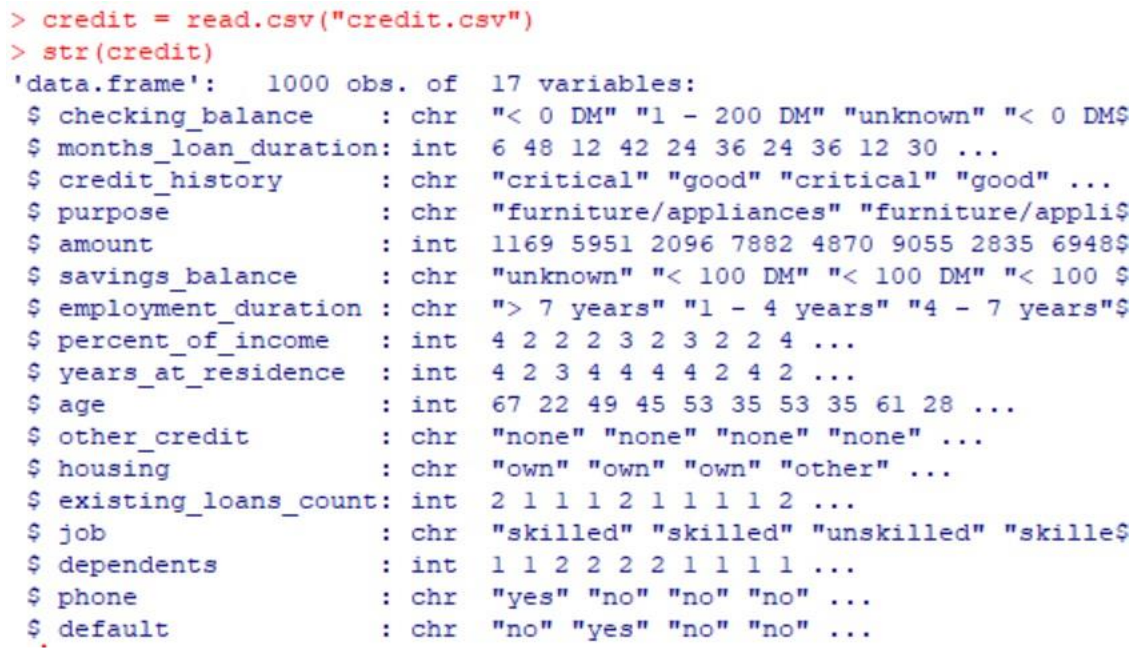
The C5.0 algorithm is capable of making various decisions automatically using the defaults integrated into the system. It incorporates post-pruning which enables it to build a large tree first and then eliminate the branches and nodes with little or no effect on the classification errors. In certain scenarios, it may move the branches up the tree or replace them with simpler decisions which are called **subtree raising and subtree replacement**. Adjusting the model so that it neither overfits nor underfits is usually one of the most challenging tasks although considering the importance of the data output, trying different pruning options may significantly improve the performance.

Step 2. Collecting Data

Since we were working with bank loans and our primary goal was to identify whether an individual loan would default due to some critical factors we had to use the information with the existing data on loans and whether they went into default. The data set used in this project is called credit.csv and it contains information on loans obtained from a credit agency. It consists of 1,000 examples of loans with some additional information indicating the features of the loan.

Step 3. Exploring and Preparing Data

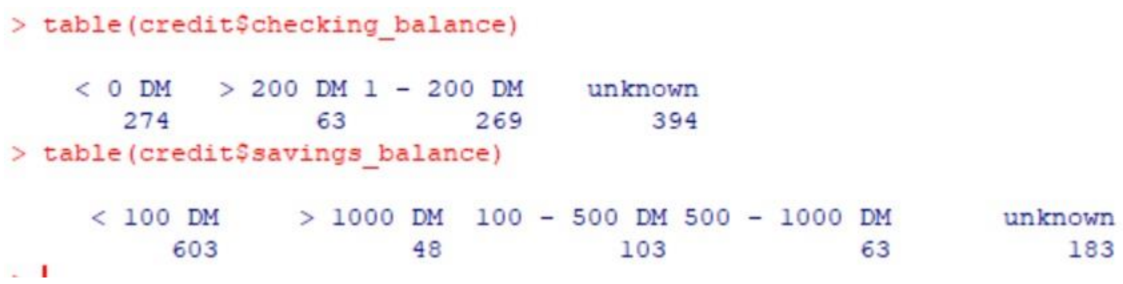
Considering the data in the data set has been already preprocessed, it significantly simplified the process for us. We imported the data using the read.csv() function and ignored stringsAsFactors features since most of the features in the data set were nominal. The value of TRUE was kept as default in this case.



```
> credit = read.csv("credit.csv")
> str(credit)
'data.frame':   1000 obs. of  17 variables:
 $ checking_balance    : chr  "< 0 DM" "1 - 200 DM" "unknown" "< 0 DM$
 $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history      : chr  "critical" "good" "critical" "good" ...
 $ purpose             : chr  "furniture/appliances" "furniture/appli$
 $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948$
 $ savings_balance     : chr  "unknown" "< 100 DM" "< 100 DM" "< 100 $
 $ employment_duration : chr  "> 7 years" "1 - 4 years" "4 - 7 years"$
 $ percent_of_income   : int  4 2 2 2 3 2 3 2 2 4 ...
 $ years_at_residence  : int  4 2 3 4 4 4 4 2 4 2 ...
 $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
 $ other_credit        : chr  "none" "none" "none" "none" ...
 $ housing             : chr  "own" "own" "own" "other" ...
 $ existing_loans_count: int  2 1 1 1 2 1 1 1 1 2 ...
 $ job                 : chr  "skilled" "skilled" "unskilled" "skille$
 $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
 $ phone               : chr  "yes" "no" "no" "no" ...
 $ default             : chr  "no" "yes" "no" "no" ...
```

Looking at the structure of the data with the help of the str() function we could thoroughly examine observations and 17 different features such as phone, job, savings_balance, etc. Some of the features were of integer type and some of character one although for the training model, we will need to convert some of the data into a factor type.

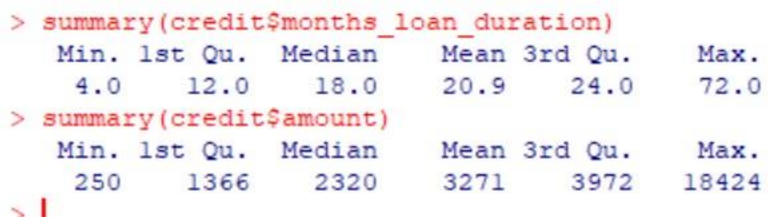
Some of the features were more likely to predict the default such as checking balance and savings balance information. We used the table() function to examine it further and noticed it was recorded in a categorical format:



```
> table(credit$checking_balance)

    < 0 DM   > 200 DM 1 - 200 DM    unknown 
       274         63        269        394 
> table(credit$savings_balance)

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

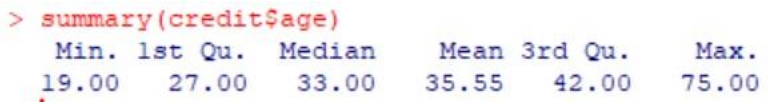
DM letters stand for the German currency Deutsche Marks indicating the number of accounts with less/more than 0, 100, 200, 500, 1000, and unknown respectively.

On the contrary, some of the features were shown in a numeric format such as duration and amount of credit requested:



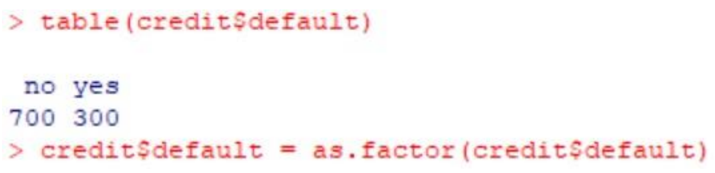
```
> summary(credit$months_loan_duration)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    4.0    12.0    18.0    20.9    24.0    72.0 
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    250    1366    2320    3271    3972   18424 
```

Here we could see that the number of loans was varying from 250 to 18424 DM and the median being 2320 DM. The loan duration ranged from 4 to 72 months and a median of 18 months.



```
> summary(credit$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  19.00   27.00   33.00   35.55   42.00   75.00 
```

As with the age, the minimum age of a person who had a loan was 19 and the maximum was 75. Then we checked the last feature of whether a person went into default or not by checking the default feature in the data set using the table() function. As it is shown below, 300 applicants went into default and 700 did not. Then we had to convert this feature into a factor to process it through the model training:

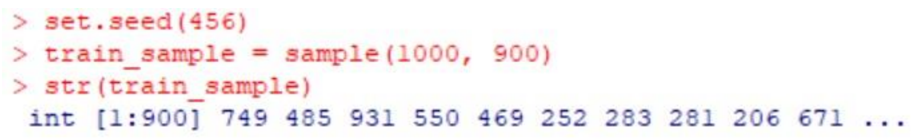


```
> table(credit$default)

 no yes 
700 300 
> credit$default = as.factor(credit$default)
```

This number is preferable to be kept at lower levels since for a bank a high number of default loans means an additional financial burden. The idea of our project was to make sure that the algorithm would help the bank to identify the risk levels of some individuals and refuse to offer them a credit line.

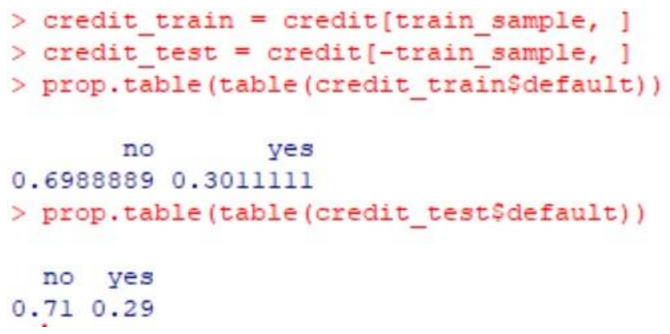
The next step was to separate the data set into 2 parts – training and testing data sets. The first one was used for training the algorithm and constructing a decision tree and the latter one to test the model and check the performance of results. We used 90 percent of observations for training and 10 percent for testing. Whenever the data is randomly stored in data sets, the first 90 percent of data can be simply extracted for training purposes. However, in the case of this data set, the data was not stored randomly thus a different approach was required. Otherwise, we could run into a risk of the result being skewed considering the smallest loans placed in the beginning and the largest ones by the end of the list.

Another approach included using the sample() function and set.seed() function. The first one simply provides a sample of random numbers from a desired data set. The second one sets a seed value that enables to repeat of the randomization process if needed at further stages with an identical outcome. Hence, we set a seed value at 456 and with the help of the sample() function selected 900 random values from the sequence of integers ranging from 1 to 1000. As the result, with the help of the str() function we could check a vector of 900 random integers:



```
> set.seed(456)
> train_sample = sample(1000, 900)
> str(train_sample)
 int [1:900] 749 485 931 550 469 252 283 281 206 671 ...
```

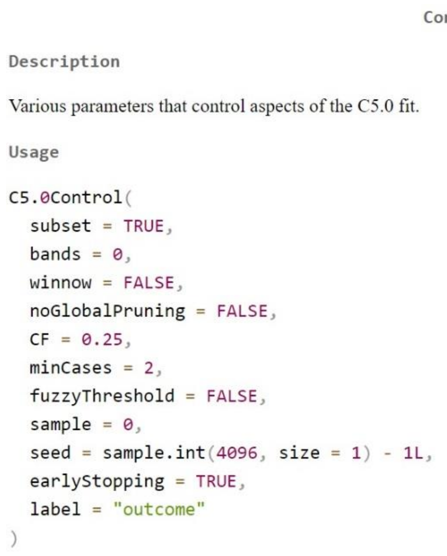
The next step was splitting the data into 90 percent of training and 10 percent testing data sets. The dash operator was used to select the records not included in specified rows so the testing data set would not include any data from the training sample. After splitting we could see that nearly 70 percent of data included the loans that did not go into default and around 30 percent indicated defaulted ones for both train and test data sets. It showed that the split went well considering the information on defaulted loans we obtained earlier.



```
> credit_train = credit[train_sample, ]
> credit_test = credit[-train_sample, ]
> prop.table(table(credit_train$default))

       no       yes 
0.6988889 0.3011111 
> prop.table(table(credit_test$default))

  no  yes 
0.71 0.29 
```

Step 4. Model training

As described earlier, the C5.0 algorithm was chosen to create a decision tree model. We installed the package in R and loaded it into the R session. The algorithms can be fine-tuned using the specifications mentioned on the help page:



Control for C5.0 Models

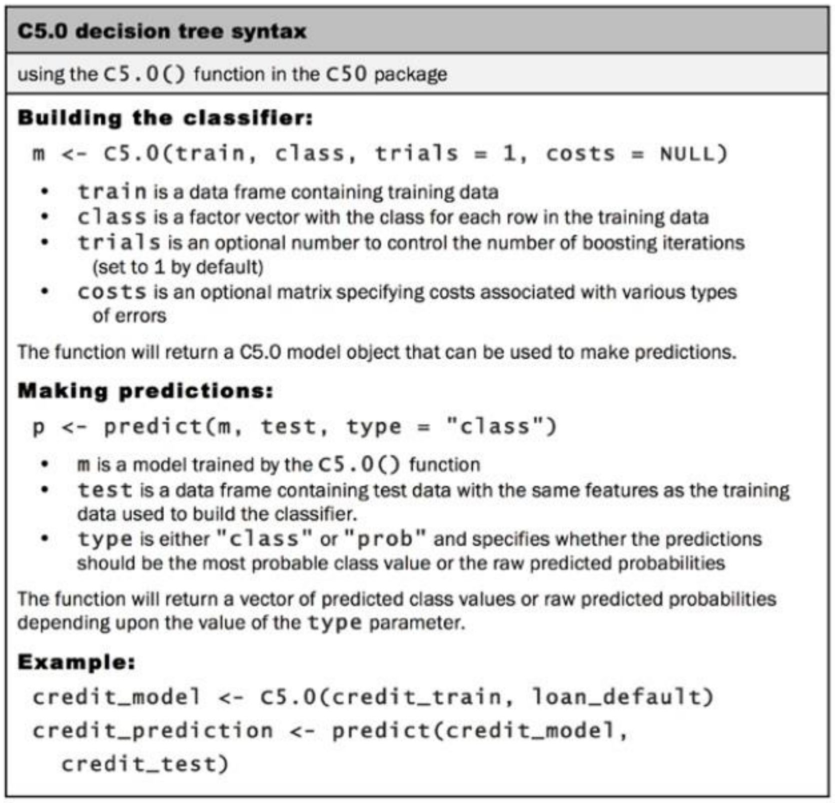
Description

Various parameters that control aspects of the C5.0 fit.

Usage

```
C5.0Control(
  subset = TRUE,
  bands = 0,
  winnow = FALSE,
  noGlobalPruning = FALSE,
  CF = 0.25,
  minCases = 2,
  fuzzyThreshold = FALSE,
  sample = 0,
  seed = sample.int(4096, size = 1) - 1L,
  earlyStopping = TRUE,
  label = "outcome"
)
```

(Quinlan R, 1993)



C5.0 decision tree syntax

using the `C5.0()` function in the `C50` package

Building the classifier:

```
m <- C5.0(train, class, trials = 1, costs = NULL)
```

- `train` is a data frame containing training data
- `class` is a factor vector with the class for each row in the training data
- `trials` is an optional number to control the number of boosting iterations (set to 1 by default)
- `costs` is an optional matrix specifying costs associated with various types of errors

The function will return a C5.0 model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

- `m` is a model trained by the `C5.0()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.
- `type` is either `"class"` or `"prob"` and specifies whether the predictions should be the most probable class value or the raw predicted probabilities

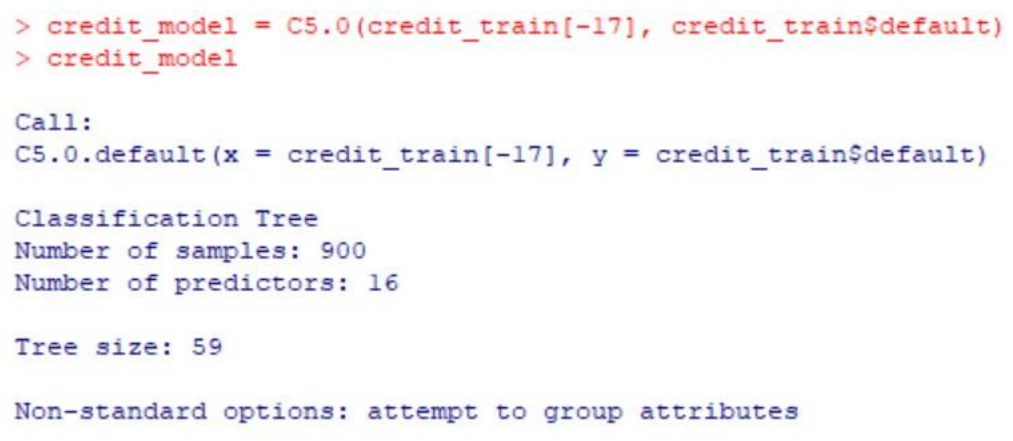
The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

Example:

```
credit_model <- C5.0(credit_train, loan_default)
credit_prediction <- predict(credit_model,
  credit_test)
```

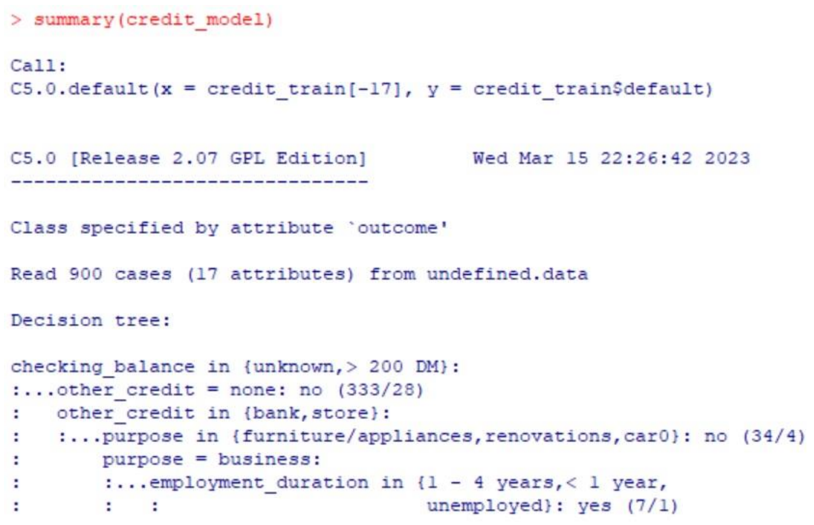
(Lantz, 2015)

As shown above, the first data frame the model is using makes up the training data, however, the default variable must be removed from this data frame and used as a factor vector with the class for each row in the training data. The other settings were used as default ones:

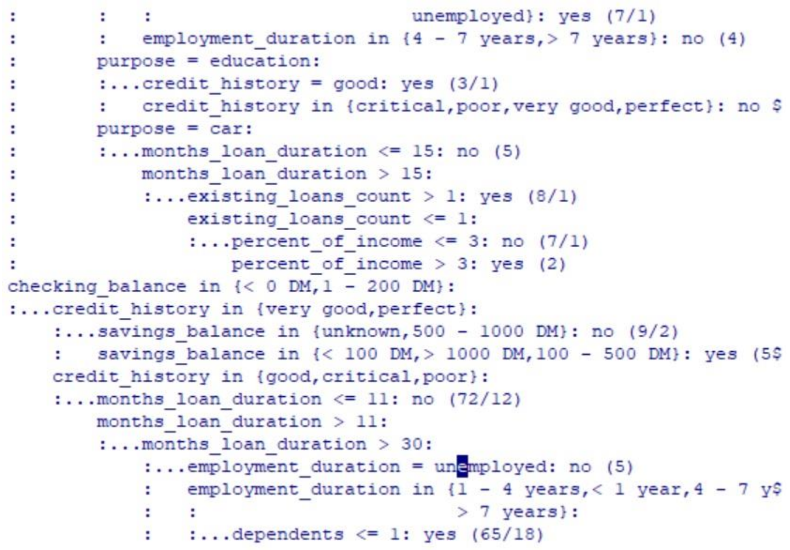


```
> credit_model = C5.0(credit_train[-17], credit_train$default)
> credit_model

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

Classification Tree
Number of samples: 900
Number of predictors: 16

Tree size: 59

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

In the picture above we can see features of the decision tree which include the function call, 16 labeled predictors which determine all the features excluding the default feature, 900 samples, and 59 decision-size trees. In order to study some of the decisions we used the summary() function:



```
> summary(credit_model)

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


C5.0 [Release 2.07 GPL Edition]         Wed Mar 15 22:26:42 2023
-------------------------------

Class specified by attribute `outcome'

Read 900 cases (17 attributes) from undefined.data

Decision tree:

checking_balance in {unknown,> 200 DM}:
:...other_credit = none: no (333/28)
:   other_credit in {bank,store}:
:   :...purpose in {furniture/appliances,renovations,car0}: no (34/4)
:       purpose = business:
:       :...employment_duration in {1 - 4 years,< 1 year,
:       :   :                       unemployed}: yes (7/1)
```



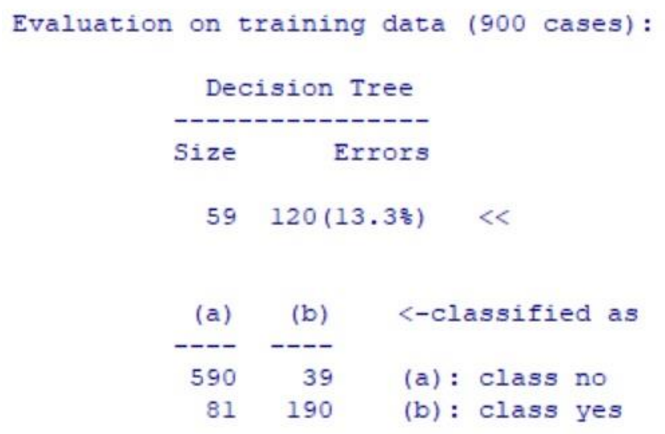
```
:       :   :                        unemployed}: yes (7/1)
:       :   employment_duration in {4 - 7 years,> 7 years}: no (4)
:       purpose = education:
:       :...credit_history = good: yes (3/1)
:       :   credit_history in {critical,poor,very good,perfect}: no $
:       purpose = car:
:       :...months_loan_duration <= 15: no (5)
:           months_loan_duration > 15:
:           :...existing_loans_count > 1: yes (8/1)
:               existing_loans_count <= 1:
:               :...percent_of_income <= 3: no (7/1)
:                   percent_of_income > 3: yes (2)
checking_balance in {< 0 DM,1 - 200 DM}:
:...credit_history in {very good,perfect}:
    :...savings_balance in {unknown,500 - 1000 DM}: no (9/2)
    :   savings_balance in {< 100 DM,> 1000 DM,100 - 500 DM}: yes (5$
    credit_history in {good,critical,poor}:
    :...months_loan_duration <= 11: no (72/12)
        months_loan_duration > 11:
        :...months_loan_duration > 30:
            :...employment_duration = unemployed: no (5)
            :   employment_duration in {1 - 4 years,< 1 year,4 - 7 y$
            :   :                      > 7 years}:
            :   :...dependents <= 1: yes (65/18)
```

The first section shows some of the decisions made by the algorithm and can be easily interpreted by a random reader. For example:

- If the checking balance is unknown or more than 200 DM and other credit is none then classify as "not likely to default". Here we have 333 examples reaching the decision and 28 cases incorrectly classified by the model.
- Otherwise, if other credit is in a bank or store and the purpose is furniture/appliances or renovations or car then classify as "not likely to default". In this case, 34 examples were meeting the criteria, and 4 were incorrectly classified cases.
- Otherwise, if the purpose is business and employment duration is from 1 to 4 years or less than 1 year or the person is unemployed then classify as "likely to default". 7 cases were reaching the decision and 1 case if improper classification.

It is a good practice to examine the decisions thoroughly since a lot of them may make a lot of sense, although in certain cases decisions may contradict common sense. For instance, the fact that a person is unemployed or employed only for less than 1 year indeed makes him more vulnerable to default. On the other hand, if a checking balance is sufficient enough and a person does not have any other credit he seems to have a more robust profile.

There is also a confusion matrix shown in the summary which indicates the error rate of the model:

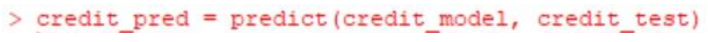


```
Evaluation on training data (900 cases):

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

            59  120(13.3%)   <<


           (a)   (b)    <-classified as
          ----  ----
           590    39    (a): class no
            81   190    (b): class yes
```

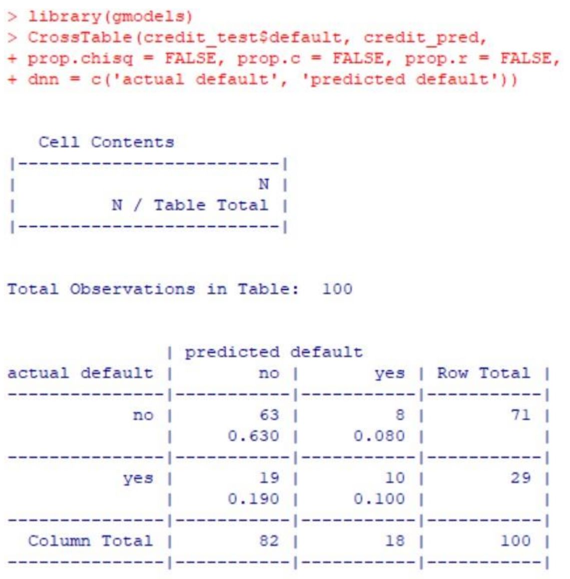
As per the picture above, out of the 900 samples, there were 120 cases misclassified which constituted a 13.3% error rate. Out of the 120 cases, there were 39 false positive cases and 81 false negative ones. It is a usual case for decision trees to overfit the model to the training data set so it is crucial to examine the performance of the algorithm on test data.

Step 5. Model Evaluation

To evaluate the performance of the model we used to predict() function and applied it to the test data. This function created a vector of predicted class values which we compared to the actual class values with the help of the gmodels package and CrossTable() function. We can leave the column row and column percentages in the table by setting prop.c and prop.r parameters to TRUE or remove them with FALSE. Prop.t percentage can also indicate the proportion of records in the cell out of the total number of records.



```
> credit_pred = predict(credit_model, credit_test)
```



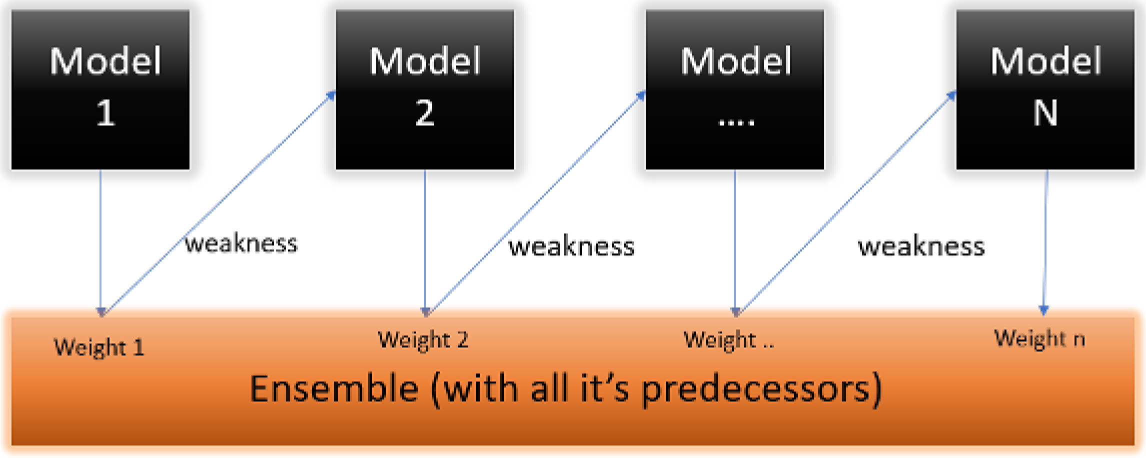
```
> library(gmodels)
> CrossTable(credit_test$default, credit_pred,
+ prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
+ dnn = c('actual default', 'predicted default'))


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


               | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        63 |         8 |        71 |
               |     0.630 |     0.080 |           |
---------------|-----------|-----------|-----------|
           yes |        19 |        10 |        29 |
               |     0.190 |     0.100 |           |
---------------|-----------|-----------|-----------|
  Column Total |        82 |        18 |       100 |
---------------|-----------|-----------|-----------|
```

As shown above, the model correctly predicted that 63 applicants didn't default and 10 applicants defaulted which brought us to 73% of accuracy and 27% of error rate respectively. In comparison with the training data, we can see a decline in the model's performance (13.3% error rate vs 27%). Actual default cases constituted 34.5% (10 out of 29 cases) which could significantly harm the banks more financially. In this case, a bank would accept the applicant who would later go into default with the bank's money.

Step 6. Model optimization.

Considering the error rate mentioned earlier, this model's results are insufficient to utilize the real data in the banking system. It could potentially lead to financial losses, especially in cases when the model could not identify the default cases properly. One of the options for model optimization is called **adaptive boosting**. The basic idea behind adaptive boosting is to combine multiple weak classifiers (i.e., classifiers that perform only slightly better than random guessing) to create a strong classifier.

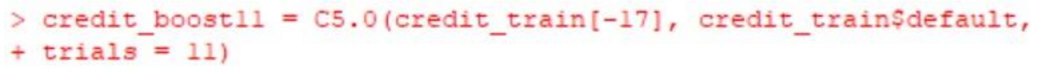
The algorithm works by iteratively training a sequence of decision trees, where each tree is trained on a weighted version of the original dataset. The weights assigned to each data point in the training set depend on whether the previous trees correctly classified them or not. Data points that were misclassified by previous trees are assigned higher weights so that subsequent trees pay more attention to them. This is the adaptive part of the algorithm, as it adjusts the weights of the training examples based on the errors made by the previous classifiers. During each iteration, the decision tree is trained on the weighted dataset, and then its error rate is calculated on the original dataset. The error rate is used to compute a weight for the tree, which is based on its performance. The better the tree performs, the higher the weight assigned to it. The final prediction is made by combining the predictions of all the individual trees, weighted by their respective weights.



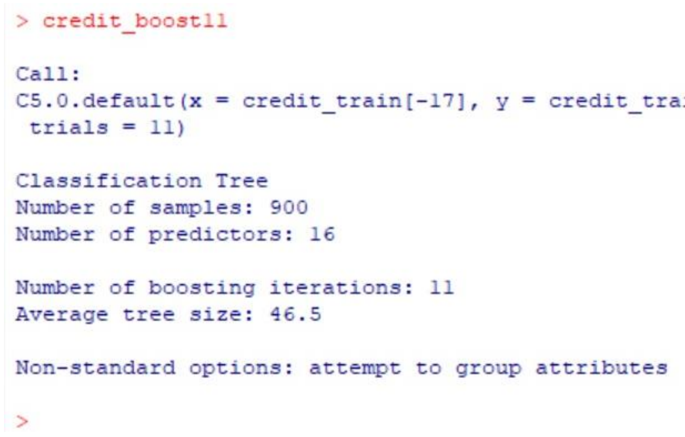
(Saini, 2021)

The main advantage of adaptive boosting is that it can improve the accuracy of decision tree models, even when they are prone to overfitting. This is because it uses a weighted sampling approach to focus on the most difficult examples in the dataset, which helps to reduce the generalization error of the final model. However, adaptive boosting can be sensitive to noisy data and outliers, which can negatively affect its performance (Saini, 2021).

As for the C5.0 algorithm, in the default configuration, we added the number of trials parameter which indicated the number of decision trees that were used. It works as an upper limit and in case the algorithm does not show any accuracy improvement with additional trials, the algorithm will stop. The standard number of trials, in this case, is usually 10 although we tried to utilize 11 trials:

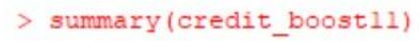


```
> credit_boost11 = C5.0(credit_train[-17], credit_train$default,
+ trials = 11)
```

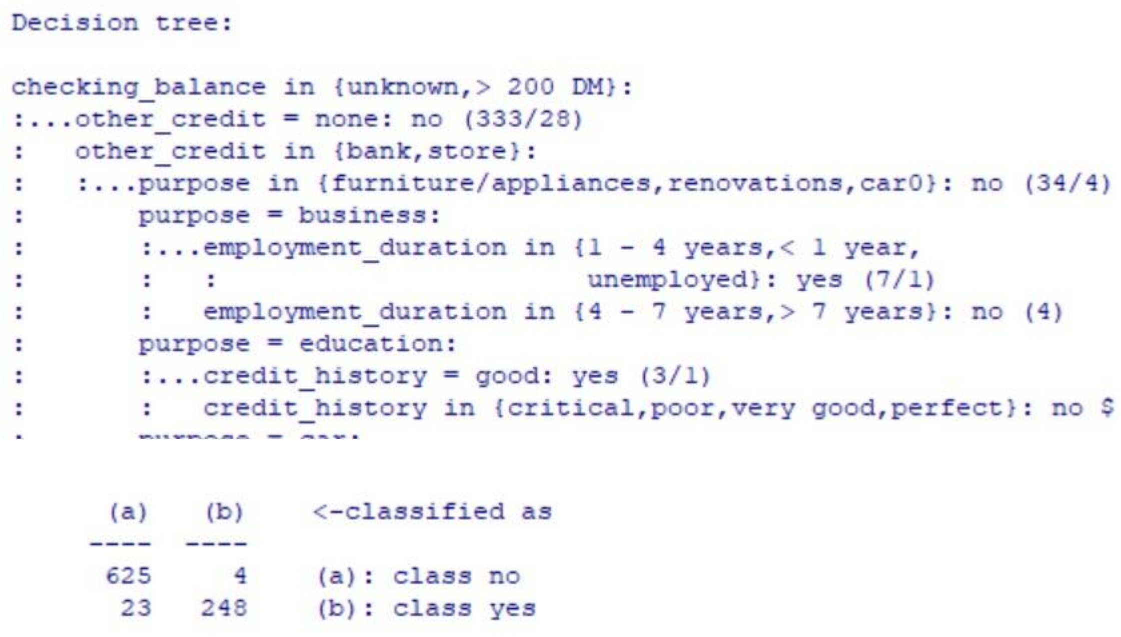


```
> credit_boost11

Call:
C5.0.default(x = credit_train[-17], y = credit_train$default,
 trials = 11)

Classification Tree
Number of samples: 900
Number of predictors: 16

Number of boosting iterations: 11
Average tree size: 46.5

Non-standard options: attempt to group attributes

>
```

By calling the prompt credit_boost11 we can see that the tree size has been reduced to 46.5 and after calling the summary() function we could thoroughly examine the new model’s performance:

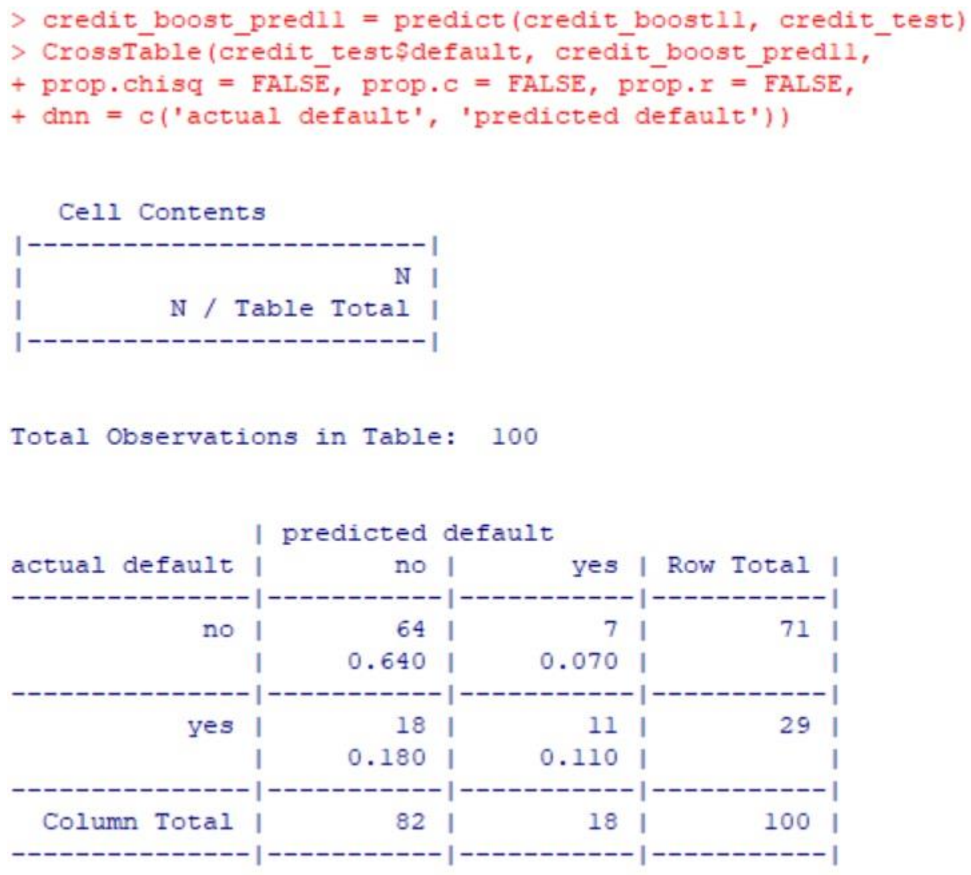


```
> summary(credit_boost11)
```



```
Decision tree:

checking_balance in {unknown,> 200 DM}:
:...other_credit = none: no (333/28)
:   other_credit in {bank,store}:
:   :...purpose in {furniture/appliances,renovations,car0}: no (34/4)
:       purpose = business:
:       :...employment_duration in {1 - 4 years,< 1 year,
:       :   :                           unemployed}: yes (7/1)
:       :   employment_duration in {4 - 7 years,> 7 years}: no (4)
:       purpose = education:
:       :...credit_history = good: yes (3/1)
:       :   credit_history in {critical,poor,very good,perfect}: no $
:       purpose = car:

       (a)   (b)    <-classified as
      ----  ----
       625     4    (a): class no
        23   248    (b): class yes
```

This time, the model predicted 27 cases wrong which constituted 3% of the error rate. It shows a considerable improvement of the model in percentage although a further look at the test data was required:



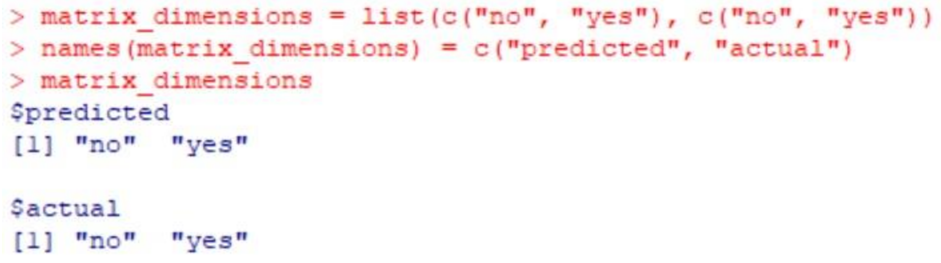
```
> credit_boost_pred11 = predict(credit_boost11, credit_test)
> CrossTable(credit_test$default, credit_boost_pred11,
+ prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
+ dnn = c('actual default', 'predicted default'))


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


               | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        64 |         7 |        71 |
               |     0.640 |     0.070 |           |
---------------|-----------|-----------|-----------|
           yes |        18 |        11 |        29 |
               |     0.180 |     0.110 |           |
---------------|-----------|-----------|-----------|
  Column Total |        82 |        18 |       100 |
---------------|-----------|-----------|-----------|
```

This time the total error rate was 25% (18 + 7) which turned into a 2% difference from the previous 27% of the model or a 7.4% improvement. As for Type II error prediction, it was improved by 1 showing 11 defaults were predicted out of 29 (38%). Should the training data set be larger, it could have possibly shown a better result on test data considering an algorithm would have had a better chance to train. Although adaptive boosting showed some improvement, the user should not utilize this function ubiquitously since it is sensitive to noise in the data and requires large computational power.

Providing a loan to a borrower who is likely to default can result in significant financial loss. To balance type I and type II errors, the cost matrix can be adjusted to give more weight to the type of error that is more costly for the specific problem at hand. For example, in a loan approval scenario, a false negative (rejecting a good loan application) could be more costly than a false positive (approving a risky loan application). In this case, the cost matrix can assign a higher penalty to false negatives to discourage the model from making this type of error.

The cost matrix can also be used to balance errors when the costs of type I and type II errors are relatively equal. For example, in a medical diagnosis scenario, both false positives (diagnosing a healthy patient as sick) and false negatives (diagnosing a sick patient as healthy) can have significant consequences. In this case, the cost matrix can be set to balance both types of errors by assigning equal penalties to each type of error.One approach to reducing the occurrence of false negatives is to reject the applicants that are highly likely to default. A bank can suffer financially more from defaults rather than from unearned interest from clients. The C5.0 algorithm offers a solution to this problem by allowing us to assign penalties to different types of errors. It reduces the costly mistakes and the cost matrix specifies the cost associated with each type of error to other predictions.

To create the cost matrix, we need to define the dimensions of the matrix. As both the predicted and actual values can only be either "yes" or "no", we need to create a 2 x 2 matrix using two vectors, each with two values. Additionally, it's important to name the matrix dimensions to avoid confusion at a later stages:

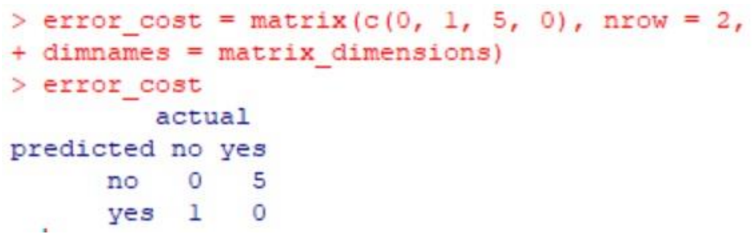


```
> matrix_dimensions = list(c("no", "yes"), c("no", "yes"))
> names(matrix_dimensions) = c("predicted", "actual")
> matrix_dimensions
$predicted
[1] "no"  "yes"

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

As discussed earlier, different types of errors should have different penalties. Defaults by common sense are more important for a bank thus more penalties should be assigned to it. There are four scenarios in the case of our model with R specifying the values for each one of them in the matrix:

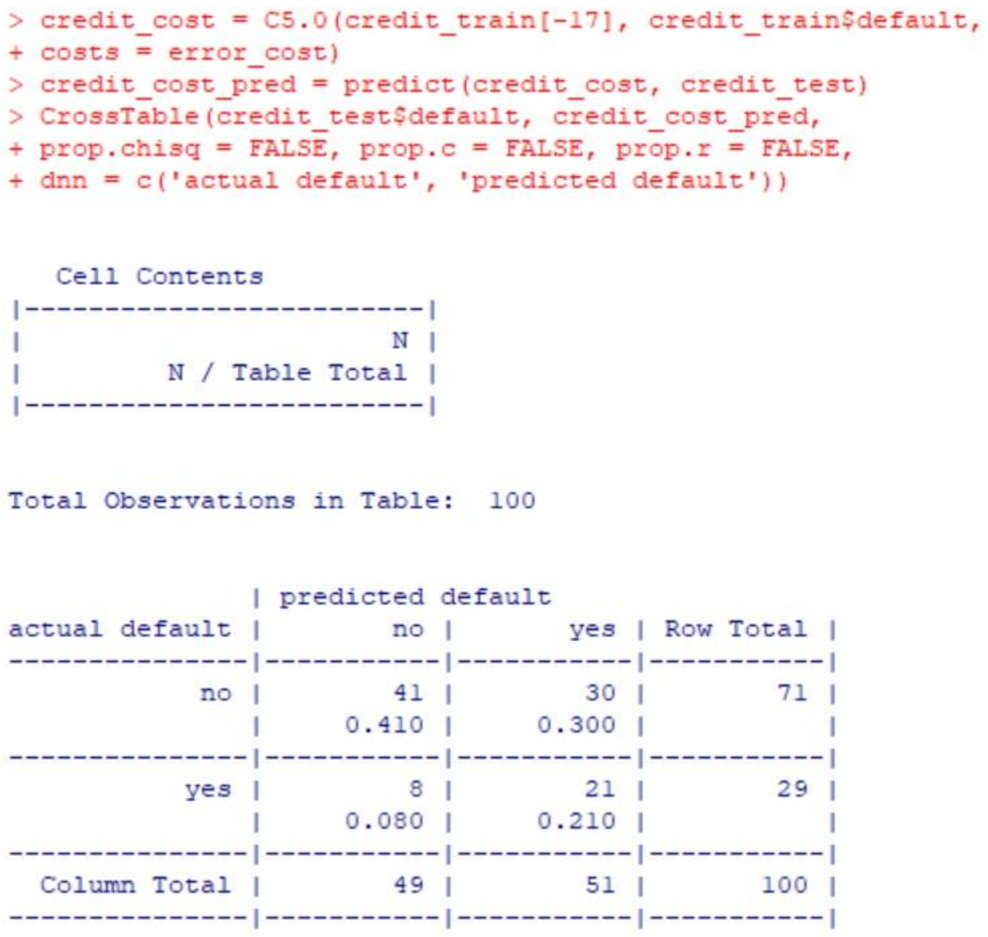
- The model predicted no default and Actually no default
- The model predicted default but Actually there was no default
- The model predicted no default and Actually there was no default
- The model predicted default and Actually there was a default

It is at the discretion of the bank staff to assign the penalty for each of the errors however in our case we assumed that a loan default costs usually five times as much as a missed client with actually no default. We marked the matrix accordingly:



```
> error_cost = matrix(c(0, 1, 5, 0), nrow = 2,
+ dimnames = matrix_dimensions)
> error_cost
         actual
predicted no yes
      no   0   5
      yes  1   0
```

We can see from the picture above that zeros indicate there was no cost for the algorithm when it does classify a loan correctly which makes sense in real-life scenarios. The same steps were conducted further on to apply our cost parameters to the decision tree:



```
> credit_cost = C5.0(credit_train[-17], credit_train$default,
+ costs = error_cost)
> credit_cost_pred = predict(credit_cost, credit_test)
> CrossTable(credit_test$default, credit_cost_pred,
+ prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
+ dnn = c('actual default', 'predicted default'))


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


               | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        41 |        30 |        71 |
               |     0.410 |     0.300 |           |
---------------|-----------|-----------|-----------|
           yes |         8 |        21 |        29 |
               |     0.080 |     0.210 |           |
---------------|-----------|-----------|-----------|
  Column Total |        49 |        51 |       100 |
---------------|-----------|-----------|-----------|
```

Here, there was a 38 percent error identified with this newly adjusted model which is much worse than a boosted one with only a 25 percent of error earlier. Nevertheless, the main idea was to reduce the false negatives (Type II error) which presumably impact the outcome more dramatically. With the latter model, we can see that 21 out of the 29 default cases were properly identified which makes it 72% although the previous boosted model showed only 38%. Considering the penalty values were assigned properly, this trade-off is favorable for a bank which could potentially save a lot of money.

Recommendations

Based on the results of our analysis using the C5.0 decision tree algorithm to identify risky bank loans, we recommend that:

- The bank adopts this method to help mitigate the risk of loan defaults only after carefully fine-tuning the parameters of the algorithm. Accuracy levels before fine-tuning are not appropriate for real-life data.
- Penalty values with cost matrices are carefully selected based on the bank's risk tolerance and industry standards.
- A larger data set is used for training to make sure the algorithm improves the efficiency with the test data set.
- Other approaches and algorithms are tested to compare the results since the decision tree may not be the most powerful one in this case or the bank loan task is too complicated for this model.

Overall, the adoption of the C5.0 algorithm will improve the accuracy of the bank's loan assessments, reduce the risk of loan defaults, and ultimately help to safeguard the bank's financial position.

Conclusion

In conclusion, the application of the C5.0 decision tree algorithm has demonstrated its relative efficacy in identifying risky bank loans. The algorithm demonstrated a moderate level of accuracy in predicting the risk level of loans, with an overall accuracy rate of 73% before boosting and 75% with boosting. This means that the bank can use this method to assess loan applications and identify high-risk applicants only after adjusting the parameters of the algorithm. Only then it offer a powerful tool to predict the risk level of loans, with a high level of accuracy, which can significantly reduce the risk of loan defaults and associated financial loss. The use of the C5.0 algorithm also offers the benefit of assigning penalties to different types of errors, which allows the bank to prioritize minimizing the risk of false negatives. By rejecting borderline applicants who may not meet the bank's lending criteria, the bank can significantly reduce the potential for defaults and associated financial loss. In our case, the cost matrix showed a 72% accuracy with false negatives.

In addition to its ability to improve the accuracy of loan assessments, the implementation of the C5.0 algorithm also offers an opportunity to improve the bank's operational efficiency. By automating the loan assessment process using this algorithm, the bank can streamline its operations, reduce manual errors, and increase productivity.

Coding Part

```
credit = read.csv("credit.csv")

str(credit)


table(credit$checking_balance)

table(credit$savings_balance)


summary(credit$months_loan_duration)

summary(credit$amount)


table(credit$default)


set.seed(456)

train_sample = sample(1000, 900)

str(train_sample)


credit_train = credit[train_sample, ]

credit_test = credit[-train_sample, ]


prop.table(table(credit_train$default))

prop.table(table(credit_test$default))



install.packages("C50")

library(C50)

credit_model = C5.0(credit_train[-17], credit_train$default)
```

```
credit_model

summary(credit_model)

credit_pred = predict(credit_model, credit_test)

library(gmodels)
CrossTable(credit_test$default, credit_pred,
prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
dnn = c('actual default', 'predicted default'))



credit_boost11 = C5.0(credit_train[-17], credit_train$default,
trials = 11)

credit_boost11

summary(credit_boost11)



credit_boost_pred11 = predict(credit_boost11, credit_test)
CrossTable(credit_test$default, credit_boost_pred11,
prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
dnn = c('actual default', 'predicted default'))



matrix_dimensions = list(c("no", "yes"), c("no", "yes"))
```

```
names(matrix_dimensions) = c("predicted", "actual")
matrix_dimensions

error_cost = matrix(c(0, 1, 5, 0), nrow = 2,
dimnames = matrix_dimensions)

error_cost

credit_cost = C5.0(credit_train[-17], credit_train$default,
costs = error_cost)
credit_cost_pred = predict(credit_cost, credit_test)
CrossTable(credit_test$default, credit_cost_pred,
prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
dnn = c('actual default', 'predicted default'))
```

References

1. Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8)
2. Jason Brownlee, Information Gain and Mutual Information for Machine Learning, October 16 2019, Probability
3. Quinlan R (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers
4. Anshul Saini — Master the AdaBoost Algorithm: Guide to Implementing & Understanding AdaBoost, Published On September 15, 2021 and Last Modified On March 3rd, 2023
5. Nagesh Singh Chauhan, KDnuggets on February 9, 2022 in Machine Learning