PREDICTIVE ANALYTICS

Northeastern University

ALY6020, SPRING 2020

MODULE 3 PROJECT ASSIGNMENT

WEEK 3: CLASSIFICATION USING DECSION TREE AND RULES

SUBMITTED BY: SHIVANI ADSAR

NUID: 001399374

SUBMITTED TO: PROF. NA YU

DATE: 04/25/2020

Introduction

The assignment provides practical experience in understanding the working of decision trees using the concept of divide and conquer strategy. We have used the "Credit" and "Mushrooms" datasets to implement the decision tree algorithm. A decision tree is a supervised machine learning algorithm that is used for classification and regression analysis, which works by splitting from the root node and then comparing values recursively.

Analysis

Credit Dataset

- The Credit Dataset has been taken from the Kaggle. This dataset has 1000 instances. The data consists of collection of the details of credit card applications.[1]
- In order to perform the predictions, we have used the Decision Tree algorithm.

Importing the Dataset

- The Credit dataset has been imported initially into a dataframe. Using the "str" function, the structure of the dataset has been analysed into 1000 observations and 21 variables.

```
> credit <- read.csv("C:/Users/Shivani Adsar/OneDrive/Desktop/Northeastern University/Predictive Analytics/Module 3/credit.csv")
> str(credit)
'data.frame':   1000 obs. of  21 variables:
 $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
 $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history      : Factor w/ 5 levels "critical","delayed",..: 1 5 1 5 2 5 5 5 5 1 ...
 $ purpose             : Factor w/ 10 levels "business","car (new)",..: 8 8 5 6 2 5 6 3 8 2 ...
 $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
 $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
 $ employment_length   : Factor w/ 5 levels "> 7 yrs","0 - 1 yrs",..: 1 3 4 4 3 3 1 3 4 5 ...
 $ installment_rate    : int  4 2 2 2 3 2 3 2 2 4 ...
 $ personal_status     : Factor w/ 4 levels "divorced male",..: 4 2 4 4 4 4 4 4 1 3 ...
 $ other_debtors       : Factor w/ 3 levels "Co-applicant",..: 3 3 3 2 3 3 3 3 3 3 ...
 $ residence_history   : int  4 2 3 4 4 4 4 2 4 2 ...
 $ property            : Factor w/ 4 levels "building society savings",..: 3 3 3 1 4 4 1 2 3 2 ...
 $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
 $ installment_plan    : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ housing             : Factor w/ 3 levels "for free","own",..: 2 2 2 1 1 1 2 3 2 2 ...
 $ existing_credits    : int  2 1 1 1 2 1 1 1 1 2 ...
 $ default             : int  1 2 1 1 2 1 1 1 1 2 ...
 $ dependents          : int  1 1 2 2 2 1 1 1 1 1 ...
```

Fig.1: Structure of the dataset

The "str" function is used to view the structure of the dataset for understanding the variables for further analysis.

- We have used the table() function to understand the data for the users savings and checking balance.[1]

```
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.2: Output of table() function

Since the dataset has the loan data of Germany, the currency is Deutsche Marks (DM). We can see the checking and savings accounts of users are categorical variables.

- The features like "months_long_duration" and "amount" are numeric. We have used the summary() function to view the data in the "months_loan_duration" and "amount" variables.

```
> 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.3: Numeric Features in dataset

We can observe that the loan amounts have a range between 250 DM to 18424 DM with a period from 4 months to 72 months, and the median amount being 2320 DM for a duration of 18 months.

- The "default" variable tells if the user was able to fulfil the payment terms.

```
> table(credit$default)

  1   2
700 300
```

Fig.4: The table() function to view default values

As we can see, 30% of the applicants have not met the payment terms while the rest 70% were able to meet the criteria.

We will have to work on improving the model in case the "default" value increases because the bank will not be making enough profit and would be running out of investments. Hence, the default value has to be reduced.[1]

Analysis on Dataset

- In order to perform analysis, we would split the data into training and testing datasets. This simulation will help in considering all the applicants.
- Sometimes, banks have data sorted in their databases and hence it is not preferred to use the same sequence in train and test as this can give biased results. Hence, we would be using random data in our train and test datasets.

```
> set.seed(12345)
> credit_rand <- credit[order(runif(1000)), ]
```

Fig.5: Random data generation

In order to generate random data we have used the set.seed() function and runif() function.

- The set.seed() uses a predefined sequence to generate random numbers, we have used 12345 as an arbitrary sequence for generating identical results. Also, the runif() command generates 1000 random numbers.[1]
- We have validated and compared with the raw dataset and the randomized dataset, if the random sequence has been generated.

```
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18424
> summary(credit_rand$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18424
> head(credit$amount)
[1] 1169 5951 2096 7882 4870 9055
> head(credit_rand$amount)
[1] 1199 2576 1103 4020 1501 1568
```

Fig.6: Comparing data for random generation

We can observe that, the data generated by the raw dataset and randomized data is different. Hence, the random data has been generated.

Splitting of Datasets

- Now, we would be splitting our data into train and test datasets.

```
> credit_train <- credit_rand[1:900, ]
> credit_test  <- credit_rand[901:1000, ]
```

We have split 90% of data into Train and 10% in Test dataset.

Fig.7: Splitting of Data

- As observed earlier, our dataset consists of 30% of defaulted loans. We will check if the 30% of the defaulted loans are present in both Train and Test datasets.

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

        1         2
0.7022222 0.2977778
> prop.table(table(credit_test$default))

   1    2
0.68 0.32
```

As we can interpret, approximately, 30% of the data in both Train and Test is defaulted.

Fig.8: Checking Defaulted Loans in Split Data

Model Building

- In order to train our decision tree model, we have used the C5.0 algorithm which uses entropy for finding out if the data is homogenous and is easier to deploy.
- The 17th column in our dataset is the "default" variable which is an independent variable and has been excluded from training dataset. This would be used as a target variable for classification.

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

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

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

Tree size: 57

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

We can note that the model has been trained by excluding the "default" feature which is a class feature. Also, our model contains the C5.0 decision object, has 20 predictors and 900 samples. The tree has a size of 67 which means it has been through 67 decisions.

Fig.9: Training Model

- We have used the summary() function to view the decisions.

```
> summary(credit_model)

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


C5.0 [Release 2.07 GPL Edition]        Sat Apr 25 11:29:36 2020
-------------------------------

Class specified by attribute `outcome'

Read 900 cases (21 attributes) from undefined.data

Decision tree:

checking_balance = unknown: 1 (358/44)
checking_balance in {< 0 DM,> 200 DM,1 - 200 DM}:
:...foreign_worker = no:
    :...installment_plan in {none,stores}: 1 (17/1)
    :   installment_plan = bank:
    :   :...residence_history <= 3: 2 (2)
    :       residence_history > 3: 1 (2)
```

We can observe from the first few lines, that,

If the checking balance is "unknown", then classify as not default. Otherwise, if checking balance is less than 0 DM, between 1 and 200 DM then, classify as default etc.

Fig.10: Decision Rules

- The summary function also displays the confusion matrix for the model, which helps in understanding the incorrectly classified fields in training data.

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

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

           57  127(14.1%)   <<


          (a)   (b)    <-classified as
         ----  ----
          590    42    (a): class 1
           85   183    (b): class 2
```

Fig.11: Confusion Matrix for Train Data

As observed in the confusion matrix, the errors show 127 out of 900 of the training data having error of 14.1%. A total of actual 42 no values were incorrectly classified as yes, false positives. Whereas, 85 yes values were incorrectly classified as no, false negatives.

- Since, decision trees have the tendency to overfit the model and that is why we need to evaluate decision trees on training and testing data.

```
> 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'))
```

Fig.12: Model Performance Evaluation

The above function creates a vector of predicted values which are further compared with the actual values. The proportions, prop.c and prop.r has been set to “False” to remove row and column percentages from the table.
The library “gmodels” is used for the CrossTable() function which displays the confusion matrix.

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

Total Observations in Table:  100

                | predicted default
actual default |         1 |         2 | Row Total |
---------------|-----------|-----------|-----------|
             1 |        54 |        14 |        68 |
               |     0.540 |     0.140 |           |
---------------|-----------|-----------|-----------|
             2 |        11 |        21 |        32 |
               |     0.110 |     0.210 |           |
---------------|-----------|-----------|-----------|
  Column Total |        65 |        35 |       100 |
---------------|-----------|-----------|-----------|
```

Fig.12: Confusion Matrix for Test Data

We can observe that, out of 100 test loan applications, model correctly predicted 54 that did not default and 21 that did default. The accuracy can be seen to be 65% and error as 35%.We note that the model predicted almost 50% of the 32 loan defaults correctly in test data, and this can be a potential problem for a bank. Hence, needs to be improved.

Improving Model Performance

- A way to improve the C5.0 algorithm’s accuracy is by introducing the adaptive boosting in which maximum decision trees are built and best class is selected.

- We have used “trials” which will use different decision trees in the algorithm. If the trials stop improving the accuracy, the algorithm will stop adding trials.

```
> credit_boost10 <- C5.0(credit_train[-17], factor(credit_train$default), trials = 10)
> credit_boost10

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

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

Number of boosting iterations: 10
Average tree size: 47.3

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

The size of the tree has reduced within 10 iterations.

Fig.13: Adaptive boosting using Trials

- We will execute the summary function on credit_boost to view the performance on training model.

```
   (a)   (b)    <-classified as
  ----  ----
   629     3    (a): class 1
    27   241    (b): class 2
```

We can see that the classifier has made 30 errors out of 900 samples, which accounts to 3.4% of error rate. It is seen that the model has improved accuracy due to boosting as compared to previous model having error rate of 14.1%.

Fig.14: Summary on Training Model

- We can performed prediction on the test data to view the improved confusion matrix.

```
> credit_boost_pred10 <- predict(credit_boost10, credit_test)
> CrossTable(credit_test$default, credit_boost_pred10,prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default',
  'predicted default'))
```

Fig.15: Boosting on Test Data

```
Total Observations in Table:  100


                | predicted default
actual default |         1 |         2 | Row Total |
---------------|-----------|-----------|-----------|
             1 |        63 |         5 |        68 |
               |     0.630 |     0.050 |           |
---------------|-----------|-----------|-----------|
             2 |        16 |        16 |        32 |
               |     0.160 |     0.160 |           |
---------------|-----------|-----------|-----------|
  Column Total |        79 |        21 |       100 |
---------------|-----------|-----------|-----------|
```

We can observe that the error rate has reduced from 27% to 23%, however the algorithm needs more improvement as it is predicting 45% wrong defaults.

Fig.16: Confusion Matrix for Boosting

- A bank can get in a loss if it allocates loans to applicants that have a high probability to default. Hence we need to reduce the number of errors.
- We can use the C5.0 algorithm to assign penalty to the errors to avoid them from occurring again. The penalties can be assigned to the cost matrix, which will train the algorithm by specifying the cost of error with respect to others.

```
> error_cost <- matrix(c(0, 1, 4, 0), nrow = 2)
> error_cost
     [,1] [,2]
[1,]    0    4
[2,]    1    0
```

Fig.17: Cost Matrix

Note: The 1 shows no and 2 indicates a yes defaulted value. The rows show predicted values and columns indicate actual values. It can be seen that "false negative" has a cost of 4 while "false positive" has a cost of 1.

- We will apply the cost parameter to the C5.0 function in algorithm.

- The R Code for using the cost parameter in the algorithm is as follows:

```
> 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'))
```

Fig.18: Applying cost parameter to the algorithm

```
Total Observations in Table:  100

                | predicted default
actual default |         1 |         2 | Row Total |
---------------|-----------|-----------|-----------|
             1 |        38 |        30 |        68 |
               |     0.380 |     0.300 |           |
---------------|-----------|-----------|-----------|
             2 |         5 |        27 |        32 |
               |     0.050 |     0.270 |           |
---------------|-----------|-----------|-----------|
  Column Total |        43 |        57 |       100 |
---------------|-----------|-----------|-----------|
```

Fig.19: Confusion Matrix for Cost Function

It can be observed that this model predicts more incorrect results of 32% than the boosting algorithm that predicted 23%. This model has incorrectly classified only 25% of defaults, whereas the previous algorithms wrongly classified 50% of defaults incorrectly.

Mushrooms Dataset

- The Mushrooms Dataset has been taken from the UCI Machine Learning Repository. This dataset has 8124 instances. The data consists of collection of samples of 23 species of mushrooms.[2]
- In order to perform the predictions, we have used the Decision Tree algorithm.

<u>Importing the Dataset</u>

- Initially, the data has been imported and analysed to see the distribution of samples.

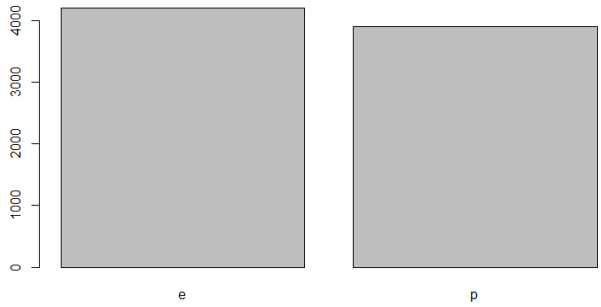
```
> table(mushrooms$class)

   e    p
4208 3916
```

Fig.20: Importing Data

We can observe that 4208 of the total samples are edible while 3916 are poisonous. We can see that the data is split equally with 50% in each sample.

- The below plot shows the distribution of edible and poisonous samples in the data.



The samples in edible and poisonous mushrooms are approximately equally distributed.

Fig.21: Plot for Samples of Mushrooms

Performing Analysis and Evaluation

- We have installed RWeka packages used for performing regressions, classifications and visualisations while rJava packages which are used for R to Java connectivity.[2]

- The usage of OneR for prediction features of mushrooms.

The OneR () rule considers all the features in mushrooms data for predictions, hence class ~.

```
install.packages("RWeka")
library(RWeka)
install.packages("rJava")
library(rJava)
install.packages("OneR")
library(OneR)
mushroom_1R <- OneR(class ~ ., data = mushrooms)
```

Fig.22: OneR function

- The OneR() learner has created the following rules:

```
Call:
OneR.formula(formula = class ~ ., data = mushrooms)

Rules:
If odor = a then class = e
If odor = c then class = p
If odor = f then class = p
If odor = l then class = e
If odor = m then class = p
If odor = n then class = e
If odor = p then class = p
If odor = s then class = p
If odor = y then class = p

Accuracy:
8004 of 8124 instances classified correctly (98.52%)
```

We see that the odor feature was selected for rule generation, and has classified into poisonous and edible for species like almond, anise, fishy, foul etc. We can note that, 8004 out of 8124 samples were correctly classified as 98.52%.

Fig.23: Output of OneR()

- We have analysed the summary of the model for better understanding.

```
Correctly Classified Instances        8004   98.5229 %
Incorrectly Classified Instances       120    1.4771 %
Kappa statistic                          0.9704
Mean absolute error                      0.0148
Root mean squared error                  0.1215
Relative absolute error                  2.958  %
Root relative squared error             24.323  %
Coverage of cases (0.95 level)          98.5229 %
Mean rel. region size (0.95 level)      50      %
Total Number of Instances             8124

=== Confusion Matrix ===
    a    b   <-- classified as
 4208    0 |    a = edible
  120 3796 |    b = poisonous
```

Fig.24: Output of Summary() function

The summary function shows different statistics. The 120 value show mushrooms that were actually edible but classified as poisonous. Whereas, 0 poisonous mushrooms were classified as edible.

- On the basis of above statistics, we can say that the model is classifying edible mushrooms as poisonous and vice-versa which may lead to problems for the consumers. Hence, we need to improve our model performance.

Improving Model Accuracy

- In order to improve the accuracy, we have trained the model by JRip() which is a rule based learner similar to OneR.[3]

```
> mushroom_JRip <- JRip(class ~ ., data = mushrooms)
> mushroom_JRip
JRIP rules:
===========

(odor = f) => class=p (2160.0/0.0)
(gill.size = n) and (gill.color = b) => class=p (1152.0/0.0)
(gill.size = n) and (odor = p) => class=p (256.0/0.0)
(odor = c) => class=p (192.0/0.0)
(spore.print.color = r) => class=p (72.0/0.0)
(stalk.surface.below.ring = y) and (stalk.surface.above.ring = k) => class=p (68.0/0.0)
(habitat = l) and (cap.color = w) => class=p (8.0/0.0)
(stalk.color.above.ring = y) => class=p (8.0/0.0)
 => class=e (4208.0/0.0)

Number of Rules : 9
```

Fig.25: JRip Classifier

We can see that the rule shows, it is a poisonous mushroom if, odor is foul, the gill size is narrow and buff and odor is pungent, otherwise, it is edible mushroom.

- We have plotted the decision tree on “odor” and “gill size” using the ctree() function.

```
> mushrooms_ctreeall <- ctree(class ~ odor + gill.size , data = mushrooms)
> plot(mushrooms_ctreeall)
```

Fig.25: Ctree() function for plotting decision tree

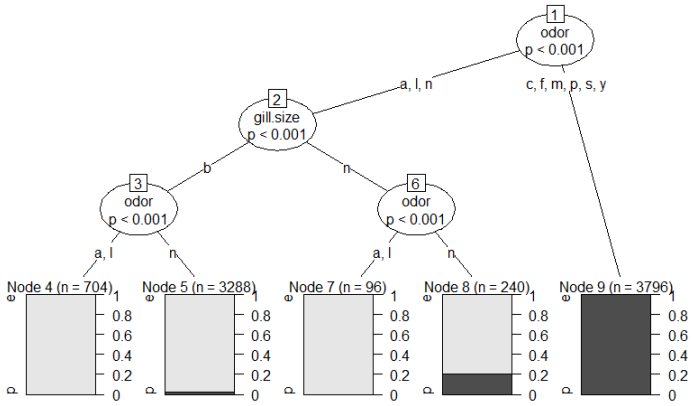


Fig.26: Decision Tree

It is observed from the Decision Tree that, if the gill size is narrow and has a buff color, the mushroom is poisonous. If the mushroom smells like anise or almond, it is considered to be edible.

Conclusion

- On the basis of Decision Tree analysis performed on “Credit” dataset, it can be observed that, the accuracy of model improves with error rate reduction from 27% to 23% after implementing boosting algorithm. Also, the cost function is effective for certain datasets, as for our model it has incorrectly classified only 25% of defaults, whereas the previous algorithms wrongly classified 50% of defaults incorrectly.
- After performing analysis on the “Mushrooms” dataset, it was observed, that the JRip() classifier has helped in improving accuracy, and predictions. It can be noted that, the rule shows, it is a poisonous mushroom if, odor is foul, the gill size is narrow and buff and odor is pungent, otherwise, it is edible mushroom.

References

[1] Kumar, S. (2018, February 27). Credit approval. Retrieved from https://www.kaggle.com/shravan3273/credit-approval

[2] Uci. (2016, December 1). Mushroom Classification. Retrieved from https://www.kaggle.com/uciml/mushroom-classification

[3] Wingate, J. (2019, August 20). Mushroom Classification with OneR and JRip in R: Mushroom Dataset. Retrieved from https://www.engineeringbigdata.com/mushroom-classification-oner-jrip-r/