"Classification using Decision tree with implementation of C5.0 and Learner Rule Algorithm"

ALY6020 Week 3 Assignment for Predictive Analytics

Submitted to:

Dr. Michael Prokle
College of Professional Studies
Northeastern University, Boston

Submitted by:

Anupam Maheshwari
Academic Term: Summer Quarter 2018
Northeastern University, MA

Abstract

The Paper talk in brief about the implementation of Decision tree algorithm with C5.0 and learners rule. The assignment is split into two parts with part-A using Credit analysis and part-B for poisonous Mushrooms rule classification in a dataset. The data is retrieved from UCI machine learning repository. The paper includes discussion on decision tree methodology, best splitting criteria, entropy, information gains, pruning, strengths, and weakness.

The world is relying heavily on classification models today, be it for finance industry or food industry. Decision tree is a very strong algorithm which has marked its foot print as a standard algorithm for data classification methodology. We can use this algorithm in situations as simple as on deciding if to go to market by setting decision making feature rules like weather conditions, available budget or in the complicate market situations like deciding on the approval of loan by checking the credit history of client.

The best thing about the decision tree classification is that if covers and include all the possible case scenarios associated with a case. Note: This can be a disadvantage as well for the complex dataset as tree over grows in size and prediction is not real time leading to loss of utility of the algorithm itself.

Decision trees

As the name suggests decision trees generates model in the tree structure and comprises of links between nodes which are basically the decision rules. The branches of each node are the decision results and can or can't be split ahead. The tree ends on the leaf node or the terminal node which holds the final decision.

As the decision tree is essentially a flowchart which transparently help decision making. There are many versions and algorithms which implements decision tree functionality but this assignment talks majorly about C5.0 and Learners Rule.

The C5.0 decision tree algorithm

C5.0 is one of the industry standardised algorithm for implementation of the decision tree. The single threaded implementation of C5.0 is available for public and the algorithm performance. Note: Intel Math kernel library may be used to enhance the performance.

The C5.0 takes following criteria into the consideration

- *Choosing the best split –* The first task of the implementation is to evaluate and decide on what feature to split for classification. This also depends on the purity of data. If data is heavily heterogenous it is considered as impure while a data with a single feature is called pure.

C5.0 uses entropy for measuring purity. Entropy uses 0 to indicate pure data while 1 indicates heterogeneous data. The difference of resulting value of entropy before and after the split of feature gives the information gain and assist on deciding if the chosen feature is perfect fit for classification.

- *Pruning the decision tree*- This is one of the limiting factor of decision tree. If the complexity of data increases heavily than it needs to be addressed and we generally have Pre-pruning and Post-pruning method to deal with the situation.

Part A

Credit Dataset- "The dataset contains information on loans obtained from a credit agency in Germany. The credit dataset includes 1,000 examples on loans, plus a set of numeric and nominal features indicating the characteristics of the loan and the loan applicant. A class variable indicates whether the loan went into default".

Step 1 is all about downloading the data and analysing it for the useful feature. I particularly found following attributes of real importance it I associate it with our problem statement :

• Checking Balance

• Savings Balance

• Loan Duration

• Loan Amount

• Default Applicants

Step 2 is about preparing dataset for training and testing while exploring the features in parallel. This starts with reading data in a variable from the CSV file. The next step is exploring some of the attributes like.

```
> 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
```

Figure 1: Result for credit attribute analysis

The default variable indicates whether the loan applicant was unable to meet the agreed payment terms and went into default. A total of 30 percent of the loans went into default:

```
> table(credit$default)

  1   2
700 300
```

Figure 2: Result for checking percentage of defaulted loans

Note: Here Default- 2 ,Non-Default- 1

Data is split in a training dataset to build the decision tree and a test dataset to evaluate the performance of the model on new data. The 90 percent goes for the training and 10 percent remains for testing and evaluation.
The data is checked heavily for consistency throughout the process and the identical credit data store is maintained. Functions like summary and table property is used to assert that.

Step 3 is about training a model on the data. C5 model is installed and imported for implementation in the program. The following image highlights major features of C5 package

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 (by default, 1)
- `costs` is an optional matrix specifying costs associated with 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 likely 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)
```

Figure 3: C5.0 package description

Note: With the updated dataset the 17th column in credit_train is the class variable, default, so we need to exclude it from the training data frame as an independent variable, but supply it as the target factor vector for classification

```
> credit_model

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

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

Tree size: 57

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

Figure 4: Credit model summary and tree depth

The summary of the model is later created which produces confusion matrix and represents the incorrectly identified record majorly. The tree size shows the depth of the decision tree which is 57 decision deep.

Step 4 is about implementing the decision tree model on the test data set using predict function. The function creates predicted class values, and by using the CrossTable() function in the gmodels package we retrieved the comparison between actual and tested vectors.

```
> 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 |         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 |
---------------|-----------|-----------|-----------|
```

Figure 5: CrossTable for predicted vs actual observations

Note: The reported error rate of the classification is 35% while the accuracy rate is 65%

The error percentage is huge, and the Step 5 involves process to improve the accuracy rate. The process of boosting is used which includes:

Including number of weak learners and associating them for better decision making. The trial parameter is included in C5 function to stop adding decision tree if it is not making accuracy difference. On implementing the methodology, I got following result:

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


Total Observations in Table:  100


               | predicted default
actual default |         1 |         2 | Row Total |
---------------|-----------|-----------|-----------|
             1 |        68 |         0 |        68 |
               |     0.680 |     0.000 |           |
---------------|-----------|-----------|-----------|
             2 |         0 |        32 |        32 |
               |     0.000 |     0.320 |           |
---------------|-----------|-----------|-----------|
  Column Total |        68 |        32 |       100 |
---------------|-----------|-----------|-----------|
```

Figure 6: CrossTable for predicted vs actual observation after boosting

The result is 3% more accurate which is good sign of improving efficiency of algorithm.

Rule learners

The technique is used in alignment with decision tree for generating knowledge gain to perform prediction and classification of any dataset. The application of learner rule is heavily done in the areas like :

- Stock market prediction
- Hardware health testing and life expectancy

On the other hand, rule learners offer some distinct advantages over trees for some tasks. Unlike a tree, which must be applied from top-to-bottom, rules are facts that stand alone. The result of a rule learner is often more parsimonious, direct, and easier to understand than a decision tree built on the same data.

"Rule learners are generally applied to problems where the features are primarily or entirely nominal. They do well at identifying rare events, even if the rare event occurs only for a very specific interaction among features".

Part B problem include implementation of :

- One Rule algorithm- "It turns out that very simple association rules, involving just one attribute in the condition part, often work disgustingly well in practice with real-world data. Suppose in the weather data, you wish to be able to predict the value of play. The idea of the OneR (one-attribute-rule) algorithm is to find the one attribute to use that makes fewest prediction errors".

- Ripper algorithm- "RIPPER is an inductive rule learner. This algorithm generated a detection model composed of resource rules that was built to detect future examples of malicious executables. This algorithm used libBFD information as features. RIPPER is a rule-based learner that builds a set of rules that identify the classes while minimizing the amount of error. The error is defined by the number of training examples misclassified by the rules. An inductive algorithm learns what a malicious executable is given a set of training".

Part-B

Dataset- "This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy".

The step 1 involves collecting data for the problem of identifying rules for distinguishing between poisonous and edible mushrooms.

Step 2 prepares and format data by removing the useless attribute veil type in mushroom as I don't change throughout the dataset. I also checked for the split percentage of data because excess of split needs to be addressed before the knowledge gain and rules creation.

Step 3 is training data model with oneR function in Weka package. Note: there might be some issues while installing Weka package because its dependence on Javar package.

1R classification rule syntax

using the `OneR()` function in the `RWeka` package

Building the classifier:

```
m <- OneR(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a 1R model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `OneR()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- OneR(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

Figure 7: 1R package description

"Using the formula type ~ ., we will allow our OneR() rule learner to consider all possible features in the mushroom data when constructing its rules to predict type"

```
> mushroom_1R <- OneR(type ~ ., data = mushrooms)
> mushroom_1R
odor:
        a       -> e
        c       -> p
        f       -> p
        l       -> e
        m       -> p
        n       -> e
        p       -> p
        s       -> p
        y       -> p
(8004/8124 instances correct)
```

Figure 8: oneR learning

Step 4 involves testing the performance, the algorithm correctly classify 98.5% of the data. The best thing about the classification is that it didn't classify any of the poisonous mushroom as edible.

```
> summary(mushroom_1R)

=== Summary ===

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  %
Total Number of Instances             8124

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = e
  120 3796 |    b = p
```

Figure 9: Summary of oneR

Step 5 involves developing a better classifier using Ripper rule learning algorithm. Using JRIP() function instead of oneR() we are getting great learner rules.

RIPPER classification rule syntax

using the `JRip()` function in the `RWeka` package

Building the classifier:

```
m <- JRip(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

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

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `JRip()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- JRip(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

Figure 10: RIPPER package description

Implementation of RIPPER shows 100% result which is a great boost in the performance and hence clearly distinguish mushrooms. We can see it generate 9 rules for the same dataset with last rule classifying only the edible mushroom in the if-else style kind of statement.

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

(odor = f) => type=p (2160.0/0.0)
(gill_size = n) and (gill_color = b) => type=p (1152.0/0.0)
(gill_size = n) and (odor = p) => type=p (256.0/0.0)
(odor = c) => type=p (192.0/0.0)
(spore_print_color = r) => type=p (72.0/0.0)
(stalk_surface_below_ring = y) and (stalk_surface_above_ring = k) => type=p (68.0/0.0)
(habitat = l) and (cap_color = w) => type=p (8.0/0.0)
(stalk_color_above_ring = y) => type=p (8.0/0.0)
 => type=e (4208.0/0.0)

Number of Rules : 9
```

Figure 11: Rules generated by RIPPER

Conclusion and Insights

- I can conclude that RIPPER algorithm is able to process highly complex data with top notch accuracy level with multiple classification rules.
- Decision tree can be used for all kind of predicting attribute.
- Decision trees bring a particular set of biases to the task that a rule learner avoids by identifying the rules directly and hence we need them.

References and Citations

- Machine Learning With R: Brett Lantz, Retrieved 29th July, 2018
- http://www.soc.napier.ac.uk/~peter/vldb/dm/node8.html
- http://www.fsl.cs.stonybrook.edu/docs/binaryeval/node5.html
- https://rstudio-pubs-static.s3.amazonaws.com/272331_2152a9341c6242a9b4f5fd8dad03e008.html
- http://www.rpubs.com/jasonchanhku/loans
- http://archive.ics.uci.edu/ml/datasets.html)