Decision Tree Classification - 4 Algorithms

C. Perez

Project & Data Description

This code applies 4 decision tree algorithms (**C5.0, OneR, rpart, and randomForest**) to a diabetes dataset offered in the UCI machine learning repository with the aim to correctly classify the presence of diabetes given the presence of related conditions. The goal is to then improve on the models generated given the business objective, NOT improve on overall accuracy, and compare. The reason being that the presence of false negatives in trying to predict the presence of a condition outweighs the overall accuracy of correct classification. The dataset is made up of 520 observations of 17 features.

EDA

This section explores the diabetes data to find proportions of the target variable (class), split the data into appropriate training and test sets, and verify the proportions of both sets are representative of the whole.

```
# libraries
library(C50)
library(gmodels)
library(OneR)
library(tidyverse)
library(rJava)
library(RWeka)
library(rpart)
library(rpart.plot)
library(randomForest)

# load data -
# https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset.
diabetes <- read.csv("~/Documents/R/RProjects-Public/Machine-Learning-Data/diabetes_data.csv",
                     header = TRUE, stringsAsFactors = TRUE)

# cleaning
names(diabetes) <- str_to_lower(str_replace_all(names(diabetes),"\\.","_"))


# view the structure and get some proportions and info to start
str(diabetes)
```

```
'data.frame':   520 obs. of  17 variables:
 $ age               : int  40 58 41 45 60 55 57 66 67 70 ...
 $ gender            : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ polyuria          : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 2 2 2 1 ...
 $ polydipsia        : Factor w/ 2 levels "No","Yes": 2 1 1 1 2 2 2 2 2 2 ...
 $ sudden_weight_loss: Factor w/ 2 levels "No","Yes": 1 1 1 2 2 1 1 2 1 2 ...
```

```
 $ weakness          : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
 $ polyphagia        : Factor w/ 2 levels "No","Yes": 1 1 2 2 2 2 2 1 2 2 ...
 $ genital_thrush    : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 1 2 1 2 1 ...
 $ visual_blurring   : Factor w/ 2 levels "No","Yes": 1 2 1 1 2 2 1 2 1 2 ...
 $ itching           : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 1 2 2 2 ...
 $ irritability      : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 1 1 2 2 2 ...
 $ delayed_healing   : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 2 1 1 1 ...
 $ partial_paresis   : Factor w/ 2 levels "No","Yes": 1 2 1 1 2 1 2 2 2 1 ...
 $ muscle_stiffness  : Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 1 2 2 1 ...
 $ alopecia          : Factor w/ 2 levels "No","Yes": 2 2 2 1 2 2 1 1 1 2 ...
 $ obesity           : Factor w/ 2 levels "No","Yes": 2 1 1 1 2 2 1 1 2 1 ...
 $ class             : Factor w/ 2 levels "Negative","Positive": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
sum(is.na(diabetes))
```

```
[1] 0
```

```r
table(diabetes$class)
```

```
Negative Positive
     200      320
```

```r
prop.table(table(diabetes$class)) # approx. 61% positive 38% negative
```

```
 Negative  Positive
0.3846154 0.6153846
```

```r
# create some random samples (75/25 split)
set.seed(1230)
d_train_indices <- sample(nrow(diabetes), .75*nrow(diabetes))


# create train and test set
d_train <- diabetes[d_train_indices,]
d_test <- diabetes[-d_train_indices,]


# test the sample proportions to identify if it were a good split
prop.table(table(d_train$class))
```

```
 Negative  Positive
0.3871795 0.6128205
```

```r
prop.table(table(d_test$class))
```

```
 Negative  Positive
0.3769231 0.6230769
```

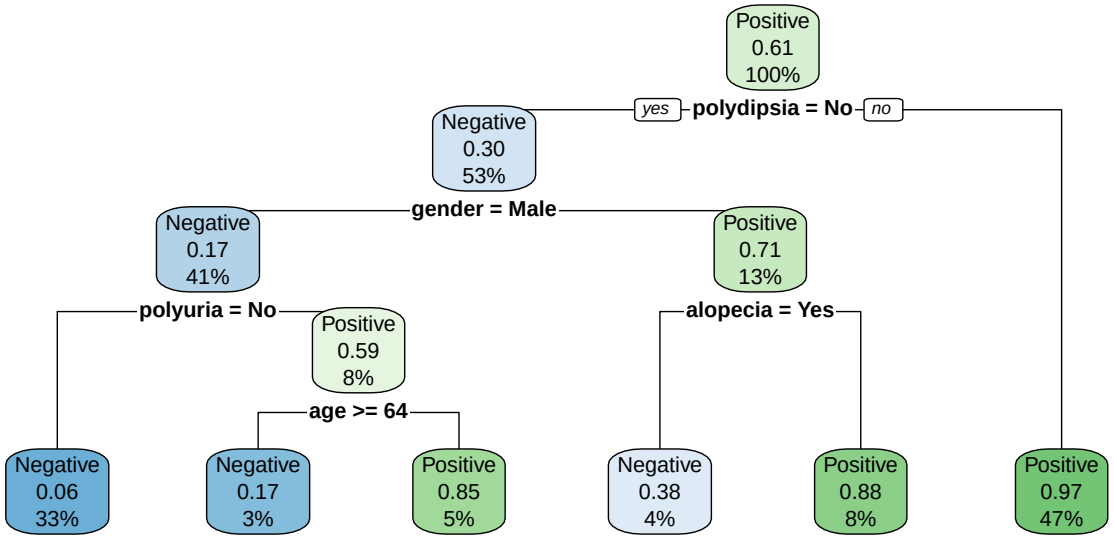
rpart

The following code implements the rpart algorithm by building a model to accurately classify the *class* variable given the other 16 features present in the data. The model is inspected and viewed (rpart.plot package) and then used to classify (predict) the outcome of the test set.

```
# create the model
rpart_model <- rpart(formula = class~., data = d_train, method = "class")

# view model
# summary(rpart_model)

# view plot of model
rpart_model
```

```
n= 390

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 390 151 Positive (0.38717949 0.61282051)
   2) polydipsia=No 208  62 Negative (0.70192308 0.29807692)
     4) gender=Male 159  27 Negative (0.83018868 0.16981132)
       8) polyuria=No 127   8 Negative (0.93700787 0.06299213) *
       9) polyuria=Yes 32  13 Positive (0.40625000 0.59375000)
        18) age>=63.5 12   2 Negative (0.83333333 0.16666667) *
        19) age< 63.5 20   3 Positive (0.15000000 0.85000000) *
     5) gender=Female 49  14 Positive (0.28571429 0.71428571)
      10) alopecia=Yes 16   6 Negative (0.62500000 0.37500000) *
      11) alopecia=No 33   4 Positive (0.12121212 0.87878788) *
   3) polydipsia=Yes 182   5 Positive (0.02747253 0.97252747) *
```

```
rpart.plot(rpart_model)
```



```r
#make a prediction
rpart_predictions <- predict(rpart_model, d_test[,-17], type = "class")
```

The accuracy is important to inspect from multiple perspectives. The first accuracy check provides a score on the match between the test class and the predicted class, and it is fairly good around 88%. Sometimes this accuracy should be compared when the data only has a few entries for one of the target classes, (i.e Positive or Negative). The accuracy for each case is computed and it shows that there is value in having the model as the target variable is not mainly of one type and the resulting accuracy drastically improves as a resul to fhaving the prediction.

This was known prior to by viewing the proportion of each class in the target variable. If a proportion is around 95+%, then the additonal accuracy should be checked. For the case where there is a fair split (i.e one target class not too dominant) then the first accuracy check is generally sufficient.

```r
# get accuracy
mean(rpart_predictions == d_test$class)
```

```
[1] 0.8846154
```

```r
# compare this to predicting every result positive or every result negative
mean(d_test$class == "Positive")
```

```
[1] 0.6230769
```

```r
mean(d_test$class == "Negative")
```

```
[1] 0.3769231
```

```r
# cross tabulate to identify the types of errors and discuss
CrossTable(d_test$class, rpart_predictions, prop.chisq = FALSE, prop.c = FALSE,
           prop.r = FALSE, dnn = c("Actual", "Predicted"))
```

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


Total Observations in Table:  130


             | Predicted
      Actual |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |        43 |         6 |        49 |
             |     0.331 |     0.046 |           |
-------------|-----------|-----------|-----------|
    Positive |         9 |        72 |        81 |
```

```
             |     0.069 |     0.554 |           |
-------------|-----------|-----------|-----------|
Column Total |        52 |        78 |       130 |
-------------|-----------|-----------|-----------|
```

Although the model was pretty accurate, there are a high number of false negatives, as can be seen in the cross-table above, which is dangerous. Improving this model would be reducing the likelihood of a false negative regardless of whether overall accuracy increase or decreases because it can be quite costly to misdiagnose someone this way. As shown below, adding a cost matrix to the rpart function allows for a penalty to be applied to producing false negatives opposed to producing false positives.

```
# How to reduce the likelihood of a false negative?
# make a prediction with a loss matrix penalizing false negatives
rpart_model_improved <- rpart(formula = class~., data = d_train, method = "class",
                              parms = list(loss=matrix(c(0,1,2,0), byrow=TRUE, nrow=2)))

rpart_predictions_improved <- predict(rpart_model_improved, d_test[,-17], type = "class")

# get new accuracy
mean(rpart_predictions_improved == d_test$class)
```

```
[1] 0.9
```

```
# cross tabulate to identify the types of errors and discuss
CrossTable(d_test$class, rpart_predictions_improved, prop.chisq = FALSE, prop.c = FALSE,
           prop.r = FALSE, dnn = c("Actual", "Predicted"))
```

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


Total Observations in Table:  130


             | Predicted
      Actual |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |        42 |         7 |        49 |
             |     0.323 |     0.054 |           |
-------------|-----------|-----------|-----------|
    Positive |         6 |        75 |        81 |
             |     0.046 |     0.577 |           |
-------------|-----------|-----------|-----------|
Column Total |        48 |        82 |       130 |
-------------|-----------|-----------|-----------|
```

The amount of false negatives dropped by roughly 33% and overall accuracy actually increased. The improvement in this case is sufficient in that it reduced the costly errors, however the model can be further improved via trying to eliminate them entirely.

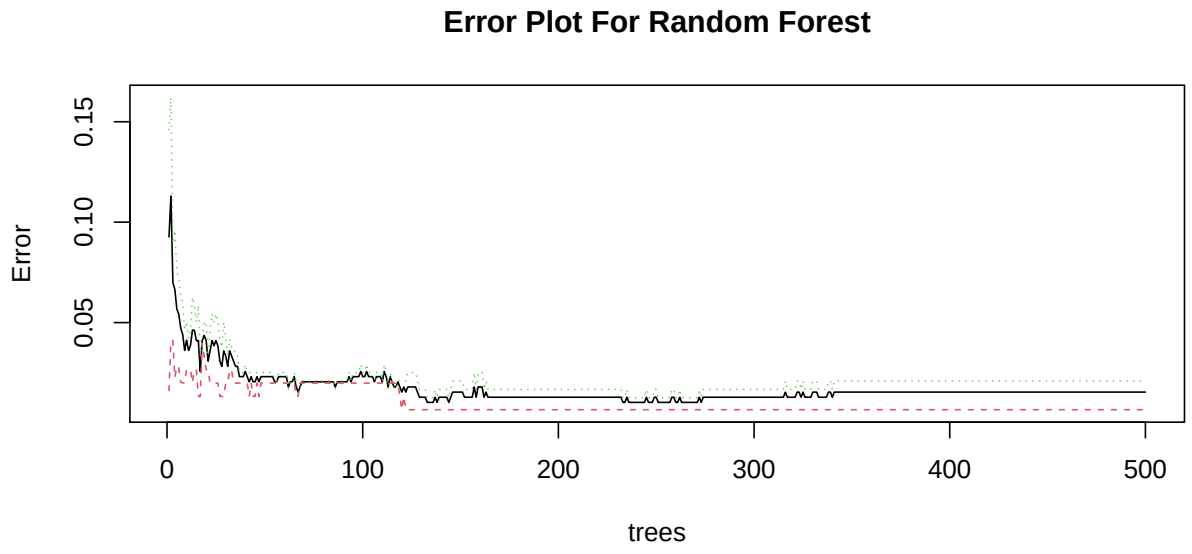
Random Forests

The randomForest function essentially creates multiple trees using subsets of the data in order to aggregate class output and conducts a vote to choose the class for the target variable based on this vote. The number of features used is generally sqrt(p), so in this case 4. The same process was repeated for the random forest, excluding the secondary accuracy checks, and is provided below.

```
# create the model
rf_model <- randomForest(class~., data = d_train)

# inspect model and error plot of model
rf_model
```

```
Call:
 randomForest(formula = class ~ ., data = d_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 1.54%
Confusion matrix:
         Negative Positive class.error
Negative      150        1 0.006622517
Positive        5      234 0.020920502
```

```
plot(rf_model, main = str_to_title("error plot for random forest"))
```



```r
#make a prediction
rf_predictions <- predict(rf_model, d_test[,-17], type = "class")

# get accuracy
mean(rf_predictions == d_test$class)
```

```
[1] 0.9538462
```

```r
#compare this to predicting every result positive or every result negative
#mean(rf_predictions == "Positive")
#mean(rf_predictions == "Negative")

# cross tabulate to identify the types of errors and discuss
CrossTable(d_test$class, rf_predictions, prop.chisq = FALSE, prop.c = FALSE,
           prop.r = FALSE, dnn = c("Actual", "Predicted"))
```

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


Total Observations in Table:  130


             | Predicted
      Actual |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |        47 |         2 |        49 |
             |     0.362 |     0.015 |           |
-------------|-----------|-----------|-----------|
    Positive |         4 |        77 |        81 |
             |     0.031 |     0.592 |           |
-------------|-----------|-----------|-----------|
Column Total |        51 |        79 |       130 |
-------------|-----------|-----------|-----------|
```

Although this model is very accurate, there are a few false negatives, as can be seen in the cross-table above, which is still dangerous. So the following makes use of the *cutoff* parameter to reduce the likelihood of false negatives. The results are very good and provides a better solution than the rpart function as we can successfully eliminate the false negatives.

```r
# How to reduce the likelihood of a false negative?
# make a prediction with a cutoff parameter base don ROC curve
rf_model_improved <- randomForest(formula = class~., data = d_train, type = "class",
                                  cutoff = c(.80,.20))
```

```r
rf_predictions_improved <- predict(rf_model_improved, d_test[,-17], type = "class")

# get new accuracy
mean(rf_predictions_improved == d_test$class)
```

```
[1] 0.9769231
```

```r
# cross tabulate to identify the types of errors and discuss
CrossTable(d_test$class, rf_predictions_improved, prop.chisq = FALSE, prop.c = FALSE,
           prop.r = FALSE, dnn = c("Actual", "Predicted"))
```

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


Total Observations in Table:  130


             | Predicted
      Actual |  Negative |  Positive | Row Total |
-------------|-----------|-----------|-----------|
    Negative |        46 |         3 |        49 |
             |     0.354 |     0.023 |           |
-------------|-----------|-----------|-----------|
    Positive |         0 |        81 |        81 |
             |     0.000 |     0.623 |           |
-------------|-----------|-----------|-----------|
Column Total |        46 |        84 |       130 |
-------------|-----------|-----------|-----------|
```