# Machine Learing with Tree-Based Models in R

Ahern Nelson

Spring 2018

## 1 Introduction

In this paper we will investigate the tree-based models in R. We will start with classification trees, then regression trees, bagged/ensemble tree models, random forests, and finally gradient boosting machines. We will follow from lectures and guided excercises from datacamp(See [3]) and invoke our own methods as well. While the focus is mainly on the methods we will take a hands-on approach and explore these methods through applied examples on two real world datasets.

Tree based models are popular approach in machine learning in that they are easily interpreted and have high accuracy When applied appropriately. Tree based methods partition feature spaces into discrete regions and can be represented as a tree or by rectangular regions of a feature space. Consider the following in which we have a continuous target and two continuous features between zero and one.

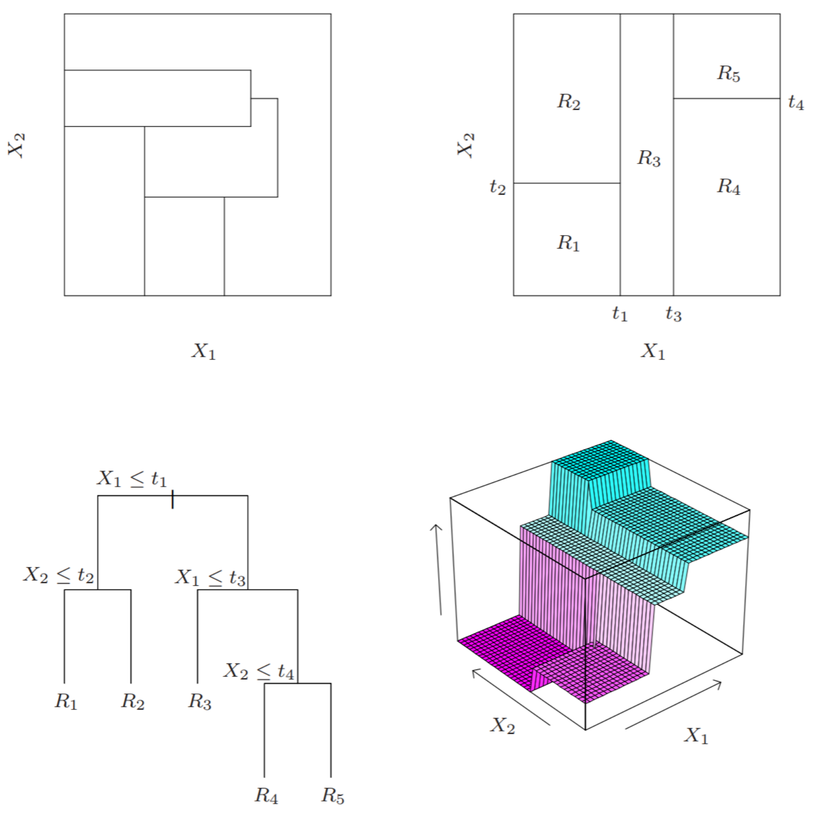
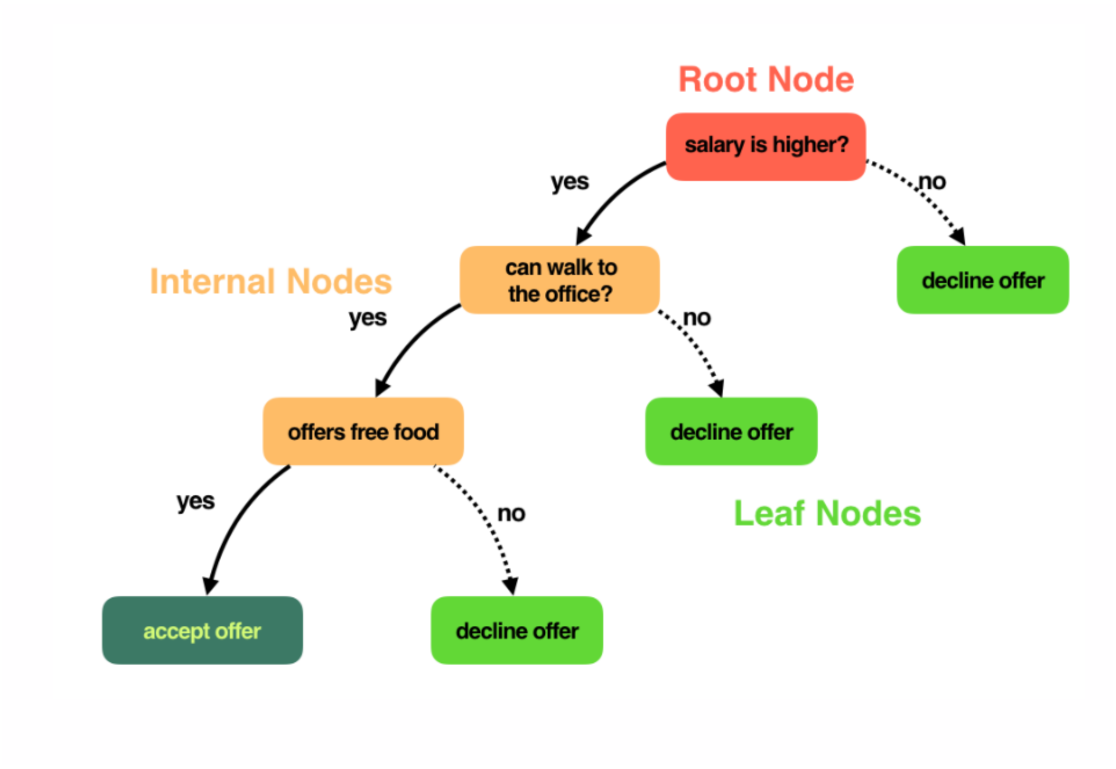


**FIGURE 9.2.** *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

This model estimates the region, $R_m$, in which our target lives. Trees are comprised of nodes and branches. A tree begins with what is called a root node and will branch off into child nodes. The last nodes in a tree, *i.e.*, nodes with no children, are called leaf nodes, and the rest are called internal nodes. A special and well known tree is a something called a binary tree which has at most two children per each node. Binary trees will be used thoroughly in what we called a decision tree. Decision trees are incredibly intuitive and function similar to the way a flow-chart would.



A classification tree is merely a decision tree that performs a classification or labeling task. Let us consider an example problem. The German Credit Dataset contains data on loan applications, namely whether the applicant defaulted on the loan and feature inputs describing income, loan duration, age of applicant, and how long the applicant has occupied their residence. We wish to construct a tree model in which we can predict if a certain loan applicant will default on their loan.

## 2 Classification Trees: Credit Dataset

We will begin our use of classification trees on the German Credit Dataset. The dataset has been preloaded and can obtained from [2]. Note that we will use a simplified subset of data (522 observations from 100). Source code for reproducibility is available on request.

Let's take a look at our data.

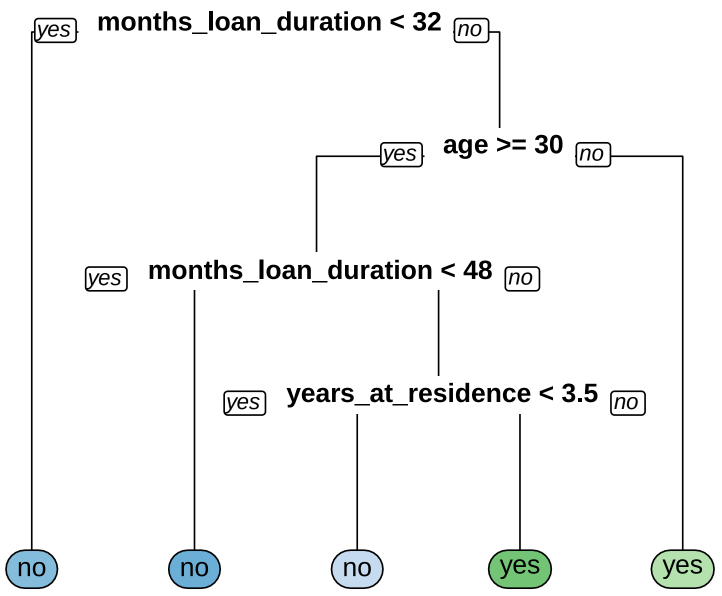
```
# Look at data
str(creditsub)

## 'data.frame': 522 obs. of  5 variables:
##  $ months_loan_duration: int  48 48 24 24 9 11 18 24 24 12 ...
##  $ percent_of_income   : int  2 4 3 3 3 4 4 4 3 2 ...
##  $ years_at_residence  : int  4 3 2 3 2 4 4 3 1 4 ...
##  $ age                 : int  31 38 26 38 28 35 26 34 32 34 ...
##  $ default             : Factor w/ 2 levels "no","yes": 1 1 1 1 2 1 1 1 1 1 ...
```

We fit a simple classification tree model using the following code:

```
library(rpart)

## Warning:  package 'rpart' was built under R version 3.4.4

# Create the model
credit_model <- rpart(formula = default ~ .,
                      data = creditsub,
                      method = "class")
```

We can view the resulting model below

```
library(rpart.plot)

## Warning:  package 'rpart.plot' was built under R version 3.4.4

rpart.plot(x = credit_model , yesno = 2, type = 0, extra = 0)
```



Notice, the model finds strong boundaries with respect to age and whether an applicant defaults. The tree model is easy to interpret and provides useful information. However, it should be noted that our model is naive at this point as we assume that the data here generalizes to a larger population. We will continue to improve our model as we continue our study of classification trees.

Classification and decision trees have the advantage that the model process, interpretation, and visualization are easy to digest and explain. Further, decisions trees do not require normalization or scaling of numeric features and can handle categorical input feature variables without creating indicator variables. We will also see that decision tress handle missing data in an elegant form using standard R libraries. Since we are merely partitioning the feature space, trees are quite robust with respect to outliers meaning that they require very little data preparation. Trees also model non-linearities in the data without explicit specification from the modeler. Lastly, the training process is relatively short.

Unfortunately, and expectedly trees come with an array of disadvantages. Very large and deep trees are difficult to interpret. Tree models have high variance that greatly impacts model performance, a slight change in data can render models meaningless. Of course, all of this implies that trees are extremely susceptible to overfitting.

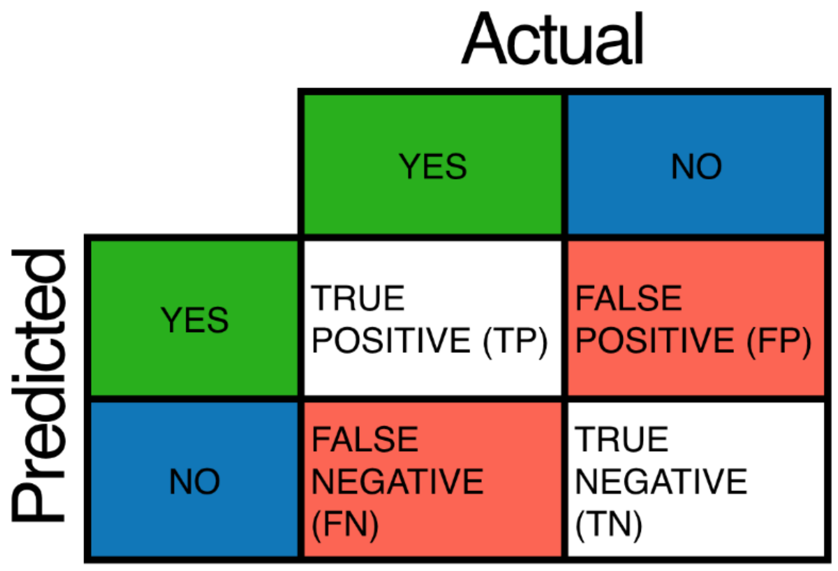
Now, we previously mentioned that model displayed above was quite naive in that it only considered the data we gave it, and another more complex data set might seriously depart from our believed model. To counter this, we split the data into training and testing sets, something we did in our supervised learning paper. The process here is identical. We will use a the standard $80-20$ train-test split. This is done with the following code. Note we will als use the full credit dataset, with no sub setting.

```r
credit <- read.csv("credit.csv")
#### Train test #############
# Total number of rows in the credit data frame
n <- nrow(credit)

# Number of rows for the training set (80% of the dataset)
n_train <- round(.8 * n)

# Create a vector of indices which is an 80% random sample
set.seed(123)
train_indices <- sample(1:n, n_train)

# Subset the credit data frame to training indices only
credit_train <- credit[train_indices, ]

# Exclude the training indices to create the test set
credit_test <- credit[-train_indices, ]
```

The next step is to evalute a model. We first build a model on the training set and then predict on that model using our test data. Since we know the correct labels of our test set we will be able to measure the accuracy of our model. We deffine accuracy as:

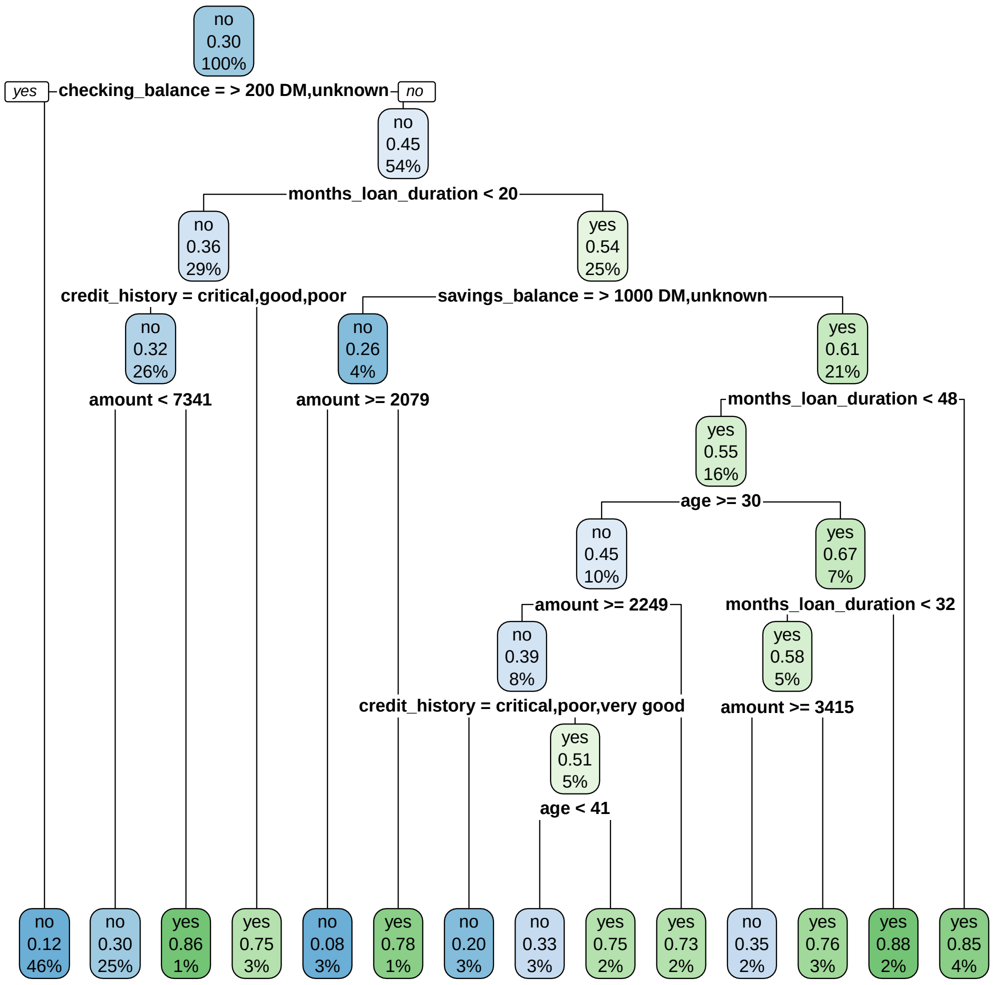
$$accuracy = \frac{\text{of correct predictions}}{\text{of total data points}}$$

Another useful way to evalute model performance is the through something know as the confuson matrix. The confusion matrix is a table that displays how our model labels things correctly or incorrectly.



We will use the *caret* package in R to investigate the confusion matrix. This package will provide us with some other useful diagnostics that we will discuss shortly. In the mean time, let us begin our modeling.

```r
##### Model ###################
# Train the model (to predict 'default')
credit_model <- rpart(formula = default ~ .,
                      data = credit_train,
                      method = "class")

# Look at the model output
rpart.plot(credit_model)
```



Note here that we have the option to print our model, but in it generally preferable to plot it. However, we also note the complexity of the model has increased significantly now that we are considering all of the feature inputs of the original data. Model interpretability is preserved here, but it should be clear that models with many feature inputs can become hard to interpret. We evaluate our model

```
##### Evaluate ##################
library(caret)

## Warning:  package 'caret' was built under R version 3.4.4
## Loading required package:  lattice
## Loading required package:  ggplot2

class_prediction <- predict(object = credit_model,
                             newdata = credit_test,
                             type = "class")
confusionMatrix(data = class_prediction,
                reference = credit_test$default)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  125  46
##        yes  13  16
##
##                Accuracy : 0.705
##                  95% CI : (0.6366, 0.7672)
##     No Information Rate : 0.69
##     P-Value [Acc > NIR] : 0.3543
##
##                   Kappa : 0.192
##  Mcnemar's Test P-Value : 3.099e-05
##
##             Sensitivity : 0.9058
##             Specificity : 0.2581
##          Pos Pred Value : 0.7310
##          Neg Pred Value : 0.5517
##              Prevalence : 0.6900
##          Detection Rate : 0.6250
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.5819
##
##        'Positive' Class : no
##
```

The confusion matrix gives a clear idea of what our model misclassifies. We would prefer for all the values to live in the diagonal of the matrix, but such a goal is not tangible. Also note that the confusion matrix provides us with a confidence interval about the accuracy estimate. Uncertainty is an important detail that is too often ignored in machine learning models. We will not discuss the other statistics reported here, but they offer important perspectives, particularly adjusted accuracy.

We also want to consider the splitting criteria in tress. We want pure regions, the criteria presented in the tree are from decision boundaries. These boundaries are formed by splitting criteria of the data into homogenous subsets. The more homogeneous the subset the purer the region and the stronger the model. To assess the purity of our models splitting we want to measure the impurity. We do this with a common metric known as the Gini index. To keep it simple we will not go into detail on these indexes. The lower the index the purer each region is and the stronger the model.

```
##### split criteria #########
# Train a gini-based model
credit_model1 <- rpart(formula = default ~ .,
                       data = credit_train,
                       method = "class",
                       parms = list(split = "gini"))

# Train an information-based model
credit_model2 <- rpart(formula = default ~ .,
                       data = credit_train,
                       method = "class",
                       parms = list(split = "information"))

# Generate predictions on the validation set using the gini model
pred1 <- predict(object = credit_model1,
                 newdata = credit_test,
                 type = "class")

# Generate predictions on the validation set using the information model
pred2 <- predict(object = credit_model2,
                 newdata = credit_test,
                 type = "class")

# Compare classification error
ModelMetrics::ce(actual = credit_test$default,
   predicted = pred1)

## [1] 0.295

ModelMetrics::ce(actual = credit_test$default,
   predicted = pred2)
```

```
## [1] 0.275
```

## 3 Regression Trees and Grade Dataset

Another advantage of trees is there generality with respect to both features and targets. We can construct similar decision tress for a continuous target. These trees are called regression trees. The differences will be largely in how form decision boundaries, no longer will the focus be entropy but rather variance. We will continue to use the *rpart* package in R to train our models.

Previously, we split our data into training and test sets to assess model performance. Here the process is slightly more complex, in that we can divide our data into an arbitrary number of subsets. For pedagogical reasons we will only divide our data into three subsets: a training set, a validation set, and a test set. The validation set is used much like to test set to evaluate the performance of a model on unseen data. The difference is that the purpose of a validation set is to tune the hyperparameters of a model or select the best model from a list of candidate models. The test set, however is only used to once at the very end to of the modeling process.

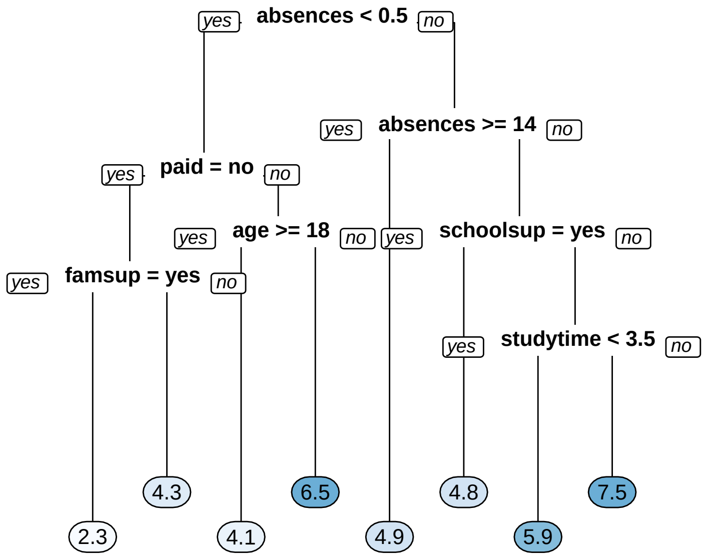
We will put these methods into practice by analyzing the Student Performance Dataset(See [1]). The goal is to predict a student final grade based on the feature inputs: sex, age, address, study time, whether they receive educational support, extra paid classes, and absences. The final grade is a continuous target between 0 and 20. Let's take a look:

```
grade <- read.csv("grade.csv")
# Look/explore the data
str(grade)

## 'data.frame': 395 obs. of  8 variables:
##  $ final_grade: num  3 3 5 7.5 5 7.5 5.5 3 9.5 7.5 ...
##  $ age        : int  18 17 15 15 16 16 16 17 15 15 ...
##  $ address    : Factor w/ 2 levels "R","U": 2 2 2 2 2 2 2 2 2 2 ...
##  $ studytime  : int  2 2 2 3 2 2 2 2 2 2 ...
##  $ schoolsup  : Factor w/ 2 levels "no","yes": 2 1 2 1 1 1 1 2 1 1 ...
##  $ famsup     : Factor w/ 2 levels "no","yes": 1 2 1 2 2 2 1 2 2 2 ...
##  $ paid       : Factor w/ 2 levels "no","yes": 1 1 2 2 2 2 1 1 2 2 ...
##  $ absences   : int  6 4 10 2 4 10 0 6 0 0 ...
```

The modeling process is almost identical with the addition of our validation set,

```r
# Randomly assign rows to ids (1/2/3 represents train/valid/test)
# This will generate a vector of ids of length equal to the number of rows
# The train/valid/test split will be approximately 70% / 15% / 15%
set.seed(1)
assignment <- sample(1:3, size = nrow(grade), prob = c(.7,.15,.15), replace = TRUE)

# Create a train, validation and tests from the original data frame
grade_train <- grade[assignment == 1, ]
grade_valid <- grade[assignment == 2, ]
grade_test <- grade[assignment == 3, ]
# Train the model
grade_model <- rpart(formula = final_grade ~ .,
                     data = grade_train,
                     method = "anova")
```

```r
# Plot the tree model
rpart.plot(x = grade_model, yesno = 2, type = 0, extra = 0)
```



Now that we've trained a regression tree we need evaluate the performance of our model. Previously, we used accuracy to evaluate the performance our classification trees, but that metric does not exist for regression. Instead we should use a metric such as SSE, MSE, etc. Here we will use a new metric common in regression and tree modeling known as Root Mean Square

Error (RMSE):

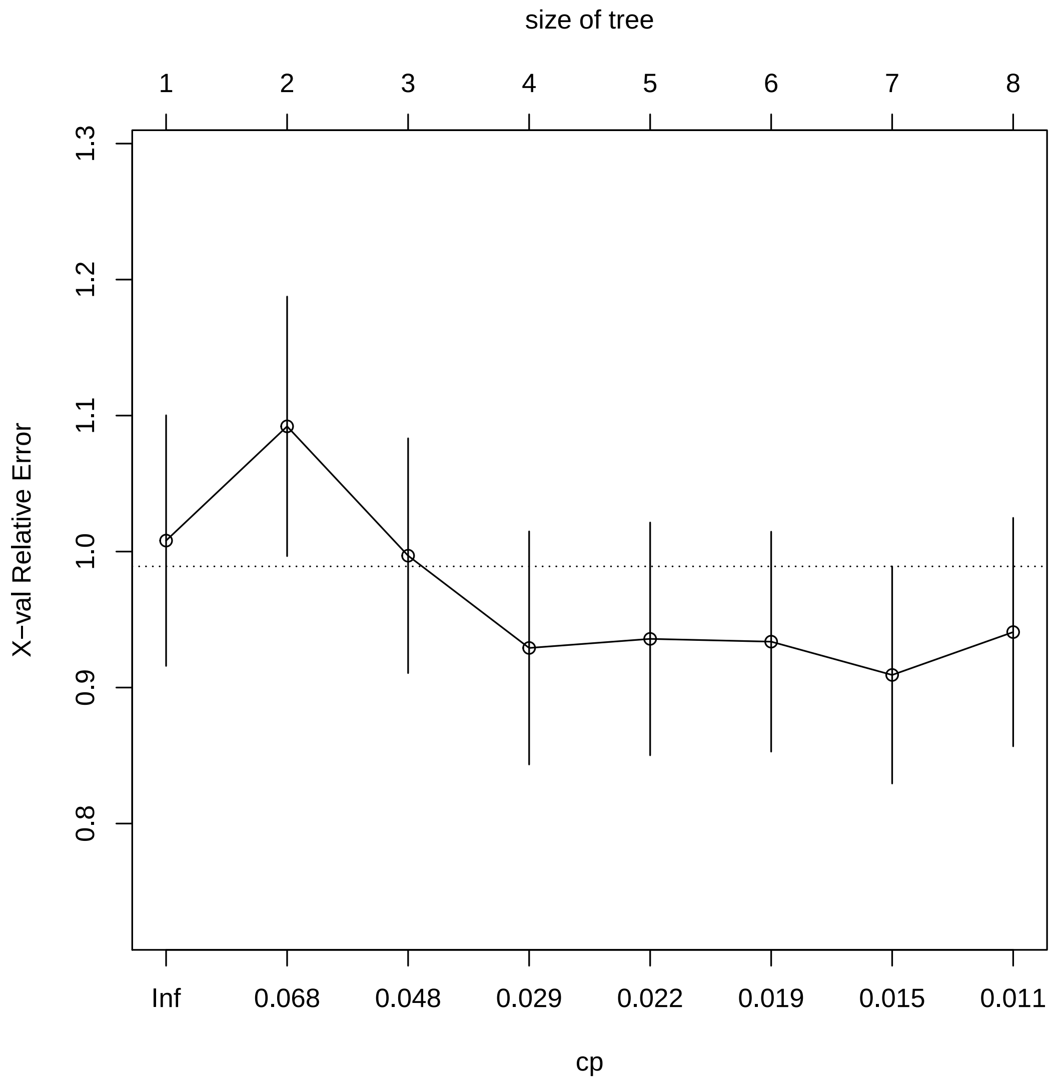
$$RMSE = \sqrt{\frac{1}{n}\sum(actual - predicted)^2}$$

$RMSE$ measures the approximate mean error or deviation of the model. We will use the $Metrics$ package in R to compute $RMSE$

```
###### Evaluate ###############
library(Metrics)

## Warning:  package 'Metrics' was built under R version 3.4.4

# Generate predictions on a test set
pred <- predict(object = grade_model,   # model object
                newdata = grade_test)  # test dataset

# Compute the RMSE
rmse(actual = grade_test$final_grade,
     predicted = pred)

## [1] 2.278249
```

We previously mentioned model hyperparameters. Hyperparameters are hidden parameters that can be tweaked to improve model performance. The parameters we will look at are the *minsplit*, the complexity parameter *cp*, and the *maxdepth*. The *minsplit* parameter is the minimum number of data points needed to attempt a split before it creates a leaf node. The complexity parameter is a scalar that serves a penalty parameter to control tree size the larger this value the harsher the penalty. The *maxdepth* limits the number maximum number of nodes between a the root node and any leaf node. The *rpart* package stores the 10-fold cross validation in a table that allows us to analyze potential *cp* values. We will discuss cross validation in the next section.

```
#### Hyperparms and control #################
# Plot the "CP Table"
plotcp(grade_model)
```

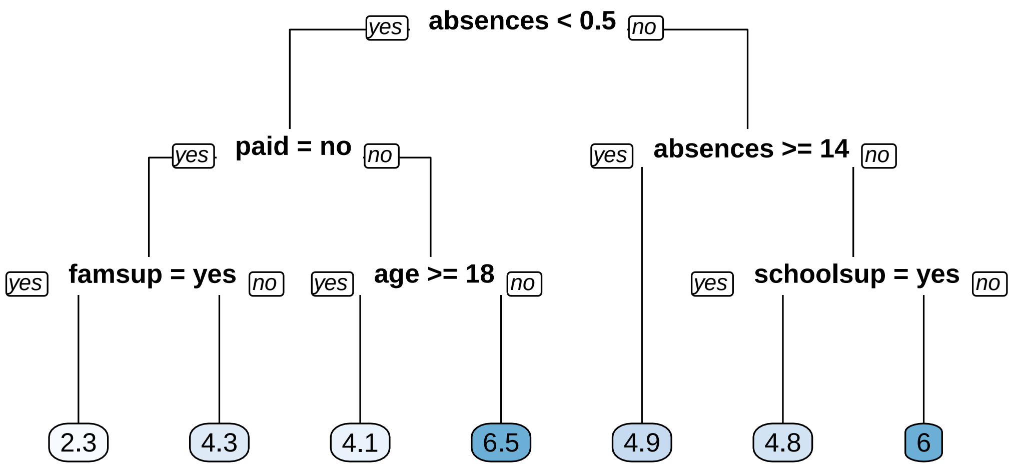


```
# Print the "CP Table"
print(grade_model$cptable)
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.06839852      0 1.0000000 1.0080595 0.09215642
## 2 0.06726713      1 0.9316015 1.0920667 0.09543723
## 3 0.03462630      2 0.8643344 0.9969520 0.08632297
## 4 0.02508343      3 0.8297080 0.9291298 0.08571411
## 5 0.01995676      4 0.8046246 0.9357838 0.08560120
## 6 0.01817661      5 0.7846679 0.9337462 0.08087153
## 7 0.01203879      6 0.7664912 0.9092646 0.07982862
## 8 0.01000000      7 0.7544525 0.9407895 0.08399125
```

```
# Retreive optimal cp value based on cross-validated error
```

```
opt_index <- which.min(grade_model$cptable[, "xerror"])
cp_opt <- grade_model$cptable[opt_index, "CP"]

# Prune the model (to optimized cp value)
grade_model_opt <- prune(tree = grade_model,
                         cp = cp_opt)

# Plot the optimized model
rpart.plot(x = grade_model_opt, yesno = 2, type = 0, extra = 0)
```



Training a single model like we have done is good, but it is preferred to create set of models from which we can select the best one. This process is called model selection, and this frequently accomplished in tree models by a method called Grid Search. A grid refers to the set of hyperparameter combinations that result in several models. The goal of a grid search is to try a larger number of hyperparameter settings and construct models from each. We then select the model with the best performance measured by a chosen metric such as $RSME$.

Let's look at a simple example in R. First we provide a sequence of possible hyperparameter values.

```
# Establish a list of possible values for minsplit and maxdepth
minsplit <- seq(1, 4, 1)
maxdepth <- seq(1, 6, 1)
```

Then we create a grid to search over

```
# Create a data frame containing all combinations
hyper_grid <- expand.grid(minsplit = minsplit, maxdepth = maxdepth)

# Check out the grid
head(hyper_grid)

##   minsplit maxdepth
## 1        1        1
## 2        2        1
## 3        3        1
## 4        4        1
## 5        1        2
## 6        2        2

# Number of potential models in the grid
num_models <- nrow(hyper_grid)
```

Then we will implement the grid search by training models on each possible combination of parameters.

```
# Create an empty list to store models
grade_models <- list()

# Write a loop over the rows of hyper_grid to train the grid of models
for (i in 1:num_models) {

  # Get minsplit, maxdepth values at row i
  minsplit <- hyper_grid$minsplit[i]
  maxdepth <- hyper_grid$maxdepth[i]

  # Train a model and store in the list
  grade_models[[i]] <- rpart(formula = final_grade ~ .,
                             data = grade_train,
                             method = "anova",
                             minsplit = minsplit,
                             maxdepth = maxdepth)
}
  # Number of potential models in the grid
  num_models <- length(grade_models)
```

Next, we evaluate the models and select the "best" one according the $RMSE$ criteria on the validation set

```
 # Create an empty vector to store RMSE values
  rmse_values <- c()

  # Write a loop over the models to compute validation RMSE
  for (i in 1:num_models) {

    # Retreive the i^th model from the list
    model <- grade_models[[i]]

    # Generate predictions on grade_valid
    pred <- predict(object = grade_models[[i]],
                    newdata = grade_valid)

    # Compute validation RMSE and add to the
    rmse_values[i] <- rmse(actual = grade_valid$final_grade,
                           predicted = pred)
  }

  # Identify the model with smallest validation set RMSE
  best_model <- grade_models[[which.min(rmse_values)]]
```

We can look at the selected hyperparameters and evaluate the model on the test set.

```
 best_model$control

## $minsplit
## [1] 2
##
## $minbucket
## [1] 1
##
```

```
## $cp
## [1] 0.01
##
## $maxcompete
## [1] 4
##
## $maxsurrogate
## [1] 5
##
## $usesurrogate
## [1] 2
##
## $surrogatestyle
## [1] 0
##
## $maxdepth
## [1] 1
##
## $xval
## [1] 10

  # Compute test set RMSE on best_model
  pred <- predict(object = best_model,
                  newdata = grade_test)
  rmse(actual = grade_test$final_grade,
       predicted = pred)

## [1] 2.124109
```

```
oldMods <- list(best_model, grade_model, grade_model_opt)
vals <- c()
for(i in 1:length(oldMods)){
    pred <- predict(object = oldMods[[i]],
                  newdata = grade_test)
  vals[i] <-rmse(actual = grade_test$final_grade,
                 predicted = pred)
}
vals

## [1] 2.124109 2.278249 2.286792
```

As you can see the resulting model reduces $RSME$ compared to the first two models. We will continue working with the credit dataset in the next paper and improve our modeling approaches by exploring new tree modeling methods. Namely, bagged models, random forests, and gradient boosting machines.

## References

[1] P. Cortez and A. Silva. *Using Data Mining to Predict Secondary School Student Performance*. 2008. URL: `http://archive.ics.uci.edu/ml`.

[2] Dua Dheeru and Efi Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[3] E. LeDell and G. Queiroz. *Machine Learning with Tree-Based Models in R*. URL: `https://www.datacamp.com`.