Learning Classification and Regression Trees with R

Authored by

Mohammed looti

November 6, 2025

When data scientists attempt to model the relationship between a response variable and a set of predictors, standard approaches like multiple linear regression are highly effective, provided the underlying structure of the relationship is fundamentally linear. However, real-world data frequently exhibits complex, non-linear interactions and high dimensionality, conditions under which traditional linear models often fail to achieve satisfactory predictive accuracy.

To tackle these challenging scenarios, non-linear statistical learning techniques are indispensable. Among the most popular and highly interpretable methods are **Classification and Regression Trees** (CART). CART algorithms belong to the family of tree-based methods, constructing sophisticated decision trees by recursively partitioning the data space based on predictor variables. This iterative splitting process aims to accurately forecast the value of the response variable by creating homogeneous groups.

The specific type of CART model implemented depends entirely on the nature of the target variable: if the outcome is continuous (e.g., predicting salary or temperature), we build a **regression tree**; conversely, if the outcome is categorical or nominal (e.g., predicting survival or class membership), we construct a **classification tree**. This comprehensive tutorial provides a step-by-step guide to fitting, optimizing, and pruning both regression and classification trees using the powerful statistical programming language, R.

Example 1: Modeling Continuous Data with a Regression Tree in R

To demonstrate the robust construction of a **regression tree**, we will utilize the well-known **Hitters** dataset, which is conveniently packaged within the ISLR package. This dataset compiles detailed statistics for 263 professional baseball players. Our primary goal is to develop a predictive model for a player's **Salary** (a continuous numerical response variable) using two key predictors: the number of *home runs* (HmRun) and the total number of *years played* (Years).

The process of creating an effective tree model involves several critical stages, ranging from initial tree generation to precise pruning, ensuring the resulting model generalizes well to new data. We begin by preparing the R environment for this analysis.

Step 1: Installing and Loading Essential R Libraries

The first prerequisite for any CART analysis in R is ensuring that the necessary packages for data handling, model fitting, and visualization are successfully loaded into the current R session. We require the **ISLR** package for the data, the **rpart** package for fitting the decision tree model itself, and **rpart.plot** for generating clear visual representations of the final decision structure.

```
library(ISLR) # Contains the Hitters dataset
library(rpart) # Core package for fitting Classification and Regression Trees
```

library(rpart.plot) # Provides advanced plotting functionalities for trees

Step 2: Generating the Initial Overgrown Regression Tree

The initial phase in building a robust tree model involves intentionally creating an overly complex, or "overgrown," tree. This maximal tree is generated by allowing the splitting process to continue until the reduction in error is minimal. We achieve this control using the **cp** (complexity parameter) argument within the rpart package's control function. By setting the complexity parameter (cp) to a very small value (e.g., 0.0001), we instruct the algorithm to perform splits as long as the R-squared value increases, even slightly, resulting in a large, unpruned structure that may suffer from overfitting.

Once the maximal tree is built, we utilize the **printcp()** function. This function displays the complexity table, which is vital for the next step--identifying the optimal pruning point by examining how the cross-validated error changes with the number of splits.

 # Build the initial, maximal tree structure
tree <- rpart(Salary ~ Years + HmRun, data=Hitters, control=rpart.control(cp=.0001))

# View the complexity parameter table for pruning optimization
printcp(tree)

Variables actually used in tree construction:
HmRun Years

Root node error: 53319113/263 = 202734

n=263 (59 observations deleted due to missingness)

CP nsplit rel error xerror xstd
1 0.24674996 0 1.00000 1.00756 0.13890
2 0.10806932 1 0.75325 0.76438 0.12828
3 0.01865610 2 0.64518 0.70295 0.12769
4 0.01761100 3 0.62652 0.70339 0.12337
5 0.01747617 4 0.60891 0.70339 0.12337
6 0.01038188 5 0.59144 0.66629 0.11817
7 0.01038065 6 0.58106 0.65697 0.11687
8 0.00731045 8 0.56029 0.67177 0.11913
9 0.00714883 9 0.55298 0.67881 0.11960
10 0.00708618 10 0.54583 0.68034 0.11988
11 0.00516285 12 0.53166 0.68427 0.11997

12 0.00445345 13 0.52650 0.68994 0.11996
13 0.00406069 14 0.52205 0.68988 0.11940
14 0.00264728 15 0.51799 0.68874 0.11916
15 0.00196586 16 0.51534 0.68638 0.12043
16 0.00016686 17 0.51337 0.67577 0.11635
17 0.00010000 18 0.51321 0.67576 0.11615
n=263 (59 observations deleted due to missingness)

Step 3: Pruning the Tree for Optimal Generalization

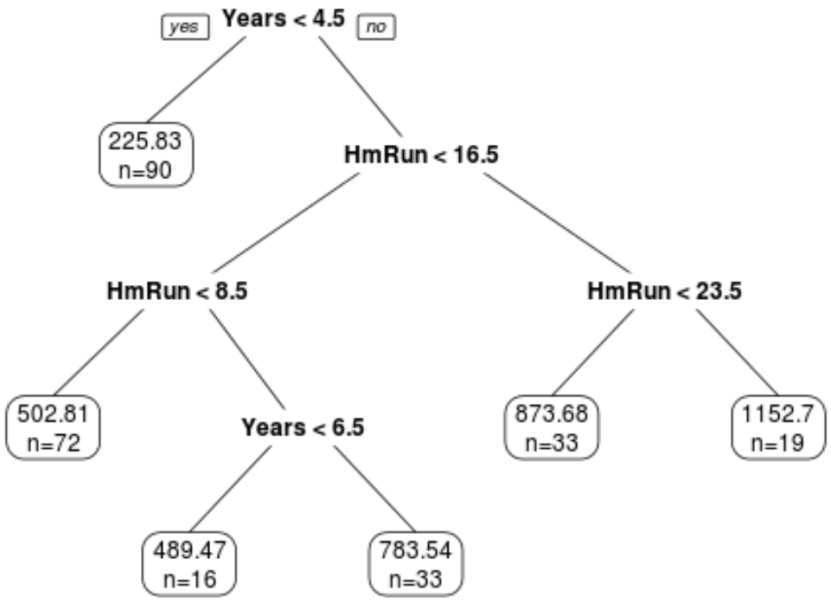
Overfitting is a significant risk when using complex models, and tree pruning is the necessary technique used to mitigate this risk. Our goal is to simplify the large tree (generated in Step 2) by identifying the optimal complexity parameter (cp) value that minimizes the estimated error on unseen data. This estimate is provided by the **xerror** column in the complexity table, which is calculated using internal cross-validation.

By selecting the cp value corresponding to the minimum **xerror**, we ensure the resulting model is simpler, more interpretable, and significantly more robust for future predictions. The **prune()** function in R then applies this optimal cp value, yielding our final, generalized **regression tree** structure.

 # Identify the best complexity parameter (cp) value that minimizes cross-validated error (xerror)
best <- tree$cptable),"CP"]

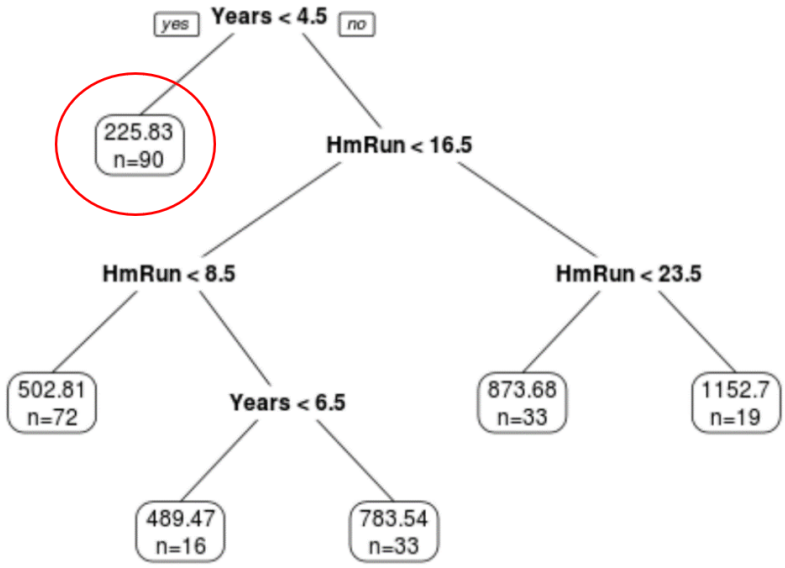
# Produce the final, pruned tree based on the calculated best cp value
pruned_tree <- prune(tree, cp=best)

Plot the optimized tree structure for visualization
prp(pruned_tree,
faclen=0, # Use full names for factor labels
extra=1, # Display the number of observations for each terminal node
roundint=F, # Prevent rounding of continuous values in output
digits=5) # Display results with five decimal places



The resulting pruned tree features six terminal nodes, each representing a distinct segment of players defined by the combination of *Years* and *HmRun*. For a **regression tree**, each terminal node provides two crucial pieces of information: the predicted value of the response variable (salary, in thousands of dollars) for players falling into that category, and the count of observations from the original dataset belonging to that specific node.

For instance, by examining the far-left terminal node, we can determine that for players with less than 4.5 years of experience, the model predicts an average salary of **$225.83k**, a prediction based on 90 observations in the dataset. This immediate interpretability is a core strength of tree-based methods.



Step 4: Utilizing the Model for Prediction

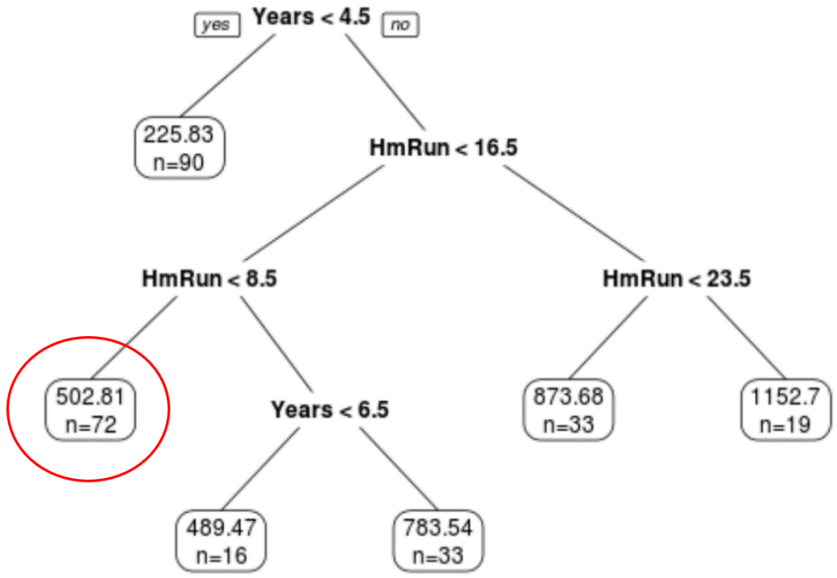
With the optimized model in hand, the pruned tree is ready to forecast the salary of a new player based on their performance metrics. The prediction process involves simply tracing the new observation through the hierarchy of splits until a terminal node is reached.

Consider a hypothetical player who has 7 years of experience and has hit 4 average home runs. By navigating the decision path established by the tree (Years > 4.5, HmRun < 7.5, Years < 9.5), the model determines that the predicted salary for this player is precisely **$502.81k**. This prediction can be programmatically confirmed using R's dedicated **predict()** function, ensuring accuracy and consistency.

 # Define the new player's characteristics in a data frame
new <- data.frame(Years=7, HmRun=4)

# Use the pruned tree model to predict the salary for this new observation
predict(pruned_tree, newdata=new)

502.8079



Example 2: Modeling Categorical Data with a Classification Tree in R

Moving beyond continuous outcomes, our second example focuses on predicting a binary categorical outcome using a **classification tree**. We will utilize the **ptitanic** dataset, included within the **rpart.plot** package, which records essential information regarding the passengers aboard the Titanic. Our analytical goal is to construct a model that predicts whether a passenger **survived** (the categorical response variable) based on the predictor variables: *class* (pclass), *sex*, and *age*.

The methodology applied to classification trees closely parallels that used for regression trees. The core steps--initial model development, optimization via pruning, and final prediction--remain the same, though the interpretation of the terminal nodes shifts from predicted values to probabilities or counts of class membership.

Step 1: Loading Required Packages for Classification

For the classification analysis, we primarily require the **rpart** package for constructing the decision tree and **rpart.plot** for visualization. The **ptitanic** dataset is loaded automatically when the **rpart.plot** package is attached, simplifying the data preparation step.

library(rpart) # Core package for fitting classification models
library(rpart.plot) # For plotting the resultant tree structure

Step 2: Building and Evaluating the Initial Classification Tree

As before, we initiate the process by constructing an intentionally large classification tree. Setting a minimal value for the complexity parameter (cp) ensures that the model explores nearly all possible splits in the data, providing a comprehensive starting point before optimization. This maximal tree is essential because it allows us to evaluate the trade-off between complexity and error reduction across the entire model space.

Reviewing the complexity table generated by **printcp()** is paramount. This output tracks the relative error reduction achieved at each split and, crucially, provides the estimated cross-validation error (xerror), which guides the subsequent pruning phase to prevent overfitting.

 # Build the initial, maximal classification tree
tree <- rpart(survived~pclass+sex+age, data=ptitanic, control=rpart.control(cp=.0001))

# View the complexity parameter table
printcp(tree)

Variables actually used in tree construction:
age pclass sex

Root node error: 500/1309 = 0.38197

n= 1309

CP nsplit rel error xerror xstd
1 0.4240 0 1.000 1.000 0.035158
2 0.0140 1 0.576 0.576 0.029976
3 0.0095 3 0.548 0.578 0.030013
4 0.0070 7 0.510 0.552 0.029517
5 0.0050 9 0.496 0.528 0.029035
6 0.0025 11 0.486 0.532 0.029117
7 0.0020 19 0.464 0.536 0.029198
8 0.0001 22 0.458 0.528 0.029035

Step 3: Pruning the Classification Tree for Optimal Performance

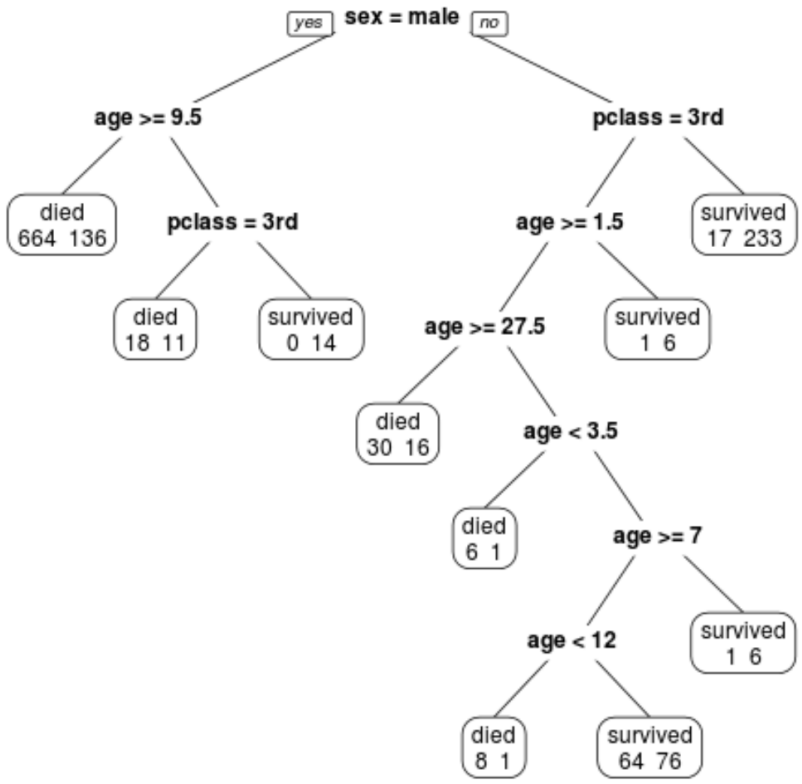
Pruning is the essential optimization step where we balance model complexity against predictive power. We identify the complexity parameter (cp) value that corresponds to the minimum **xerror** in the complexity table. This value represents the tree size that is expected to generalize best to new data, minimizing the risk associated with an overly complicated structure.

The application of the **prune()** function with this optimal cp value yields the final, simplified classification model. This final structure can then be plotted for clear visualization of the survival decision rules.

 # Identify the best cp value that minimizes the cross-validated error
best <- tree$cptable),"CP"]

# Generate the final, pruned classification tree
pruned_tree <- prune(tree, cp=best)

Plot the optimized tree structure
prp(pruned_tree,
faclen=0, # Use full names for factor labels
extra=1, # Display number of observations for each terminal node
roundint=F, # Do not round continuous output values
digits=5) # Display five decimal places in output



Interpreting the Classification Tree Results

The final pruned **classification tree**, which consists of 10 terminal nodes, must be interpreted

differently from its regression counterpart. Instead of predicting a continuous value (like salary), the terminal nodes here display the breakdown of the categorical response variable. Specifically, they show the counts of passengers who died versus those who survived within the specific subset of predictors defined by that node.

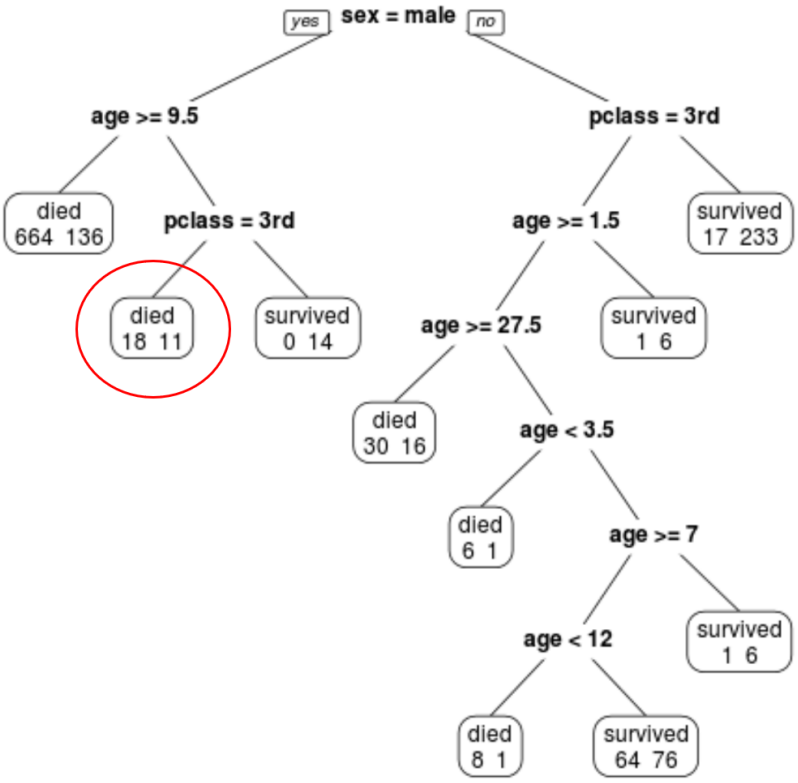
For example, upon examining the leftmost terminal node, we can infer that the specific combination of predictor variables defining this group resulted in 664 passengers dying while 136 passengers survived. This visualization provides immediate insights into which demographic groups were most likely to survive the disaster.

Step 4: Making Survival Predictions

The optimized classification tree enables us to estimate the probability of survival for any new passenger, based on their class, age, and sex. This is achieved by tracing the path down the tree to the relevant terminal node and calculating the ratio of survivors to the total count in that node.

For instance, a male passenger who was in 1st class and 8 years old falls into a specific terminal node. Within this node, 11 passengers survived out of a total of 29 observations in the training

data. This leads to an estimated survival probability of 11/29, which is approximately 37.9%.



The complete R code utilized across both the **regression tree** and classification tree examples is available here for replication and further study.