Classification and Regression Trees

Paulo S. A. Sousa

2022-04-18

Overview

- Trees are a flexible data-driven method that can be used for both:
  - Classification (called *classification tree*)
  - And regression (called *regression tree*).
- Trees are easy to interpret:
  - Trees are based on separating records into subgroups by creating splits on pre dictors.
  - These splits create logical rules that are transparent and easily understandable:
    - For example, "IF Age < 55 AND Education > 12 THEN class = 1."

Overview

- Trees require large amounts of data.
  - However, once constructed, they are computationally cheap to deploy, even on large samples.
- Trees are highly automated.
- Trees are robust to outliers.
- Trees are able to handle missing values.

Classification Trees

Two key ideas underlie classification trees are:

1. *Recursive partitioning* of the space of the predictor variables.

2. *Pruning*, using validation data.

Recursive Partitioning

Let us denote the outcome variable by Y and the input (predictor) variables by $X_1, X_2, \ldots, X_p$.

- Recursive partitioning divides up the p-dimensional space of the X predictor variables into non-overlapping multidimensional rectangles.
- This division is accomplished recursively (i.e., operating on the results of prior divisions).
  - First, one of the predictor variables is selected, say X_i , and a value of X_i , say s_i, is chosen to split the p-dimensional space into two parts:
  - one part that contains all the points with $X_i < s_i$ and the other with all the points with $X_i \geq s_i$.
  - Then, one of these two parts is divided in a similar manner by again choosing a predictor variable (it could be X_i or another variable) and a split value for that variable. This results in three (multidimensional) rectangular regions.
  - This process is continued so that we get smaller and smaller rectangular regions.

Recursive Partitioning

- The idea is to divide the entire X-space up into rectangles such that each rectangle is as *homogeneous* or "pure" as possible.
  - By pure, we mean containing records that belong to just one class.

 Is that goal always possible?

Measures of Impurity

- The two most popular measures are the *Gini index* and an *entropy measure*.
- Denote the m classes of the outcome variable by $k = 1, 2, \ldots, m$.
- The Gini impurity index for a rectangle A is defined by

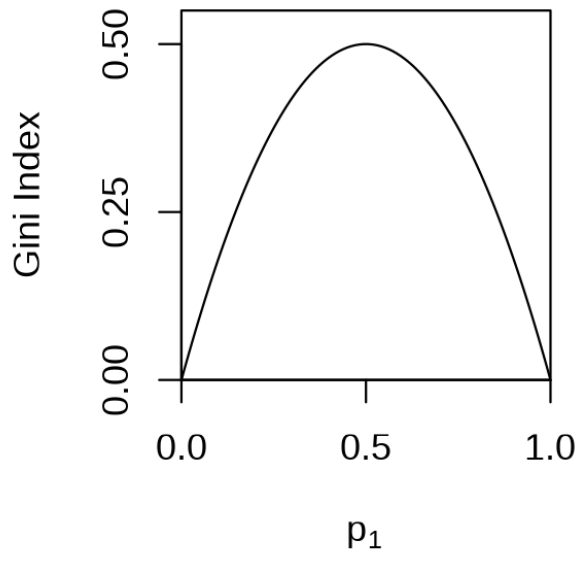
$$I(A) = 1 - \sum_{k=1}^{m} p_k^2,$$

  where p_k is the proportion of records in rectangle A that belong to class k.

  - This measure takes values between 0 (when all the records belong to the same class) and $(m-1)/m$ (when all m classes are equally represented).

Measures of Impurity

- The figure below shows the values of the Gini index for a two-class case as a function of p_k.
- It can be seen that the impurity measure is at its peak when $p_k = 0.5$ (i.e., when the rectangle contains 50% of each of the two classes).



Measures of Impurity

- A second impurity measure is the entropy measure.
- The entropy for a rectangle A is defined by

$$\text{entropy}\,(A) = -\sum_{k=1}^{m} p_k \log_2(p_k).$$

  - This measure ranges between 0 (most pure, all records belong to the same class) and $\log_2(m)$ (when all m classes are represented equally).
- In the two-class case, the entropy measure is maximized (like the Gini index) at $p_k = 0.5$.

Tree Structure

- We have two types of nodes in a tree: decision nodes and terminal nodes.
- Nodes that have successors are called *decision nodes*.
- Nodes that have no successors are called *terminal nodes* or *leaves* of the tree.

Avoiding Overfitting

- The overall error is expected to decrease as the number of terminal nodes grows until the point of overfitting.
- Moreover, for the training data the overall error decreases more and more until it is zero.

Stopping Tree Growth: Conditional Inference Trees

One can think of different criteria for stopping the tree growth before it starts overfitting the data:

- Tree depth (i.e., number of splits);
- Minimum number of records in a terminal node;
- And minimum reduction in impurity.
- In R's `rpart`, for example, we can control the depth of the tree with the complexity parameter (CP).
    - The problem is that it is not simple to determine what is a good stopping point using such rules.

Stopping Tree Growth: Conditional Inference Trees

Based on the idea of recursive partitioning, some rules have been developed to prevent the tree from growing excessively and overfitting the training data:

- One popular method called CHAID (chi-squared automatic interaction detection) is a recursive partitioning method that predates classification and regression tree (CART) procedures by several years and is widely used in database marketing applications to this day.
- It uses a well-known statistical test (the chi-square test for independence) to assess whether splitting a node improves the purity by a statistically significant amount.
    - In particular, at each node, we split on the predictor with the strongest association with the outcome variable. The strength of association is measured by the p-value of a chi-squared test of independence.
    - If for the best predictor the test does not show a significant improvement, the split is not carried out, and the tree is terminated.
    - This method is more suitable for categorical predictors, but it can be adapted to continuous predictors by binning the continuous values into categorical bins.

Stopping Tree Growth: Conditional Inference Trees

- A more general class of trees based on this idea is called *conditional inference* trees (see Hothorn et al., 2006).
  - It is suitable for both numerical and categorical outcome and predictor variables.

Pruning the Tree

- An alternative popular solution that has proven to be more successful than stopping tree growth is pruning the full-grown tree.
- A very large tree is likely to overfit the training data, and therefore the weakest branches, which hardly reduce the error rate, should be removed.
- Pruning consists of successively selecting a decision node and redesignating it as a terminal node [lopping off the branches extending beyond that decision node (its subtree) and thereby reducing the size of the tree].
- The pruning process trades off misclassification error in the validation dataset against the number of decision nodes in the pruned tree to arrive at a tree that captures the patterns -- but not the noise -- in the training data.

Cross-Validation

- Pruning the tree with the validation data solves the problem of overfitting,
  - But it does not address the problem of instability.
- The solution is to avoid relying on just one partition of the data into training and validation:
  - Rather, we do so repeatedly using cross-validation, then pool the results.
  - And use them to learn how deep to grow the original tree.
- We introduce a parameter that can measure, and control, how deep we grow the tree.
- We will register this parameter value for each minimum-error tree in the cross-validation process, take an average, then apply that average to limit tree growth to this optimal depth when working with new data.

Cross-Validation

- The cost complexity (CC) of a tree is equal to its misclassification error (based on the training data) plus a penalty factor for the size of the tree:
  - For a tree T that has $L(T)$ terminal nodes, the cost complexity can be written as

    $$CC(T) = \text{err}(T) + \alpha L(T),$$

    where $\text{err}(T)$ is the fraction of training records that are misclassified by tree T and α is a penalty factor for tree size:
  - When $\alpha = 0$, there is no penalty for having too many nodes in a tree, and this yields a tree using the cost complexity criterion that is the full-grown unpruned tree.
  - When we increase α to a very large value the penalty cost component swamps the misclassification error component of the cost complexity criterion, and the result is simply the tree with the fewest terminal nodes: namely, the tree with one node.
  - So there is a range of trees, from tiny to large, corresponding to a range of α, from large to small.

Cross-Validation

Returning to the cross-validation process, we can now associate a value of α with the minimum error tree developed in each iteration of that process. Here is a simple version of the algorithm:

Here is a simple version of the algorithm:

1. Partition the data into training and validation sets.
2. Grow the tree with the training data.
3. Prune it successively, step by step, recording CP (using the training data) at each step.
4. Register the CP that corresponds to the minimum error on the validation data.
5. Repartition the data into training and validation, and repeat the growing, pruning and CP recording process.
6. Do this again and again, and average the CP's that reflect minimum error for each tree.
7. Go back to the original data, or future data, and grow a tree, stopping at this optimum CP value.

Cross-Validation

- Typically, cross-validation is done such that the partitions (also called "folds") used for validation are non-overlapping.
- R automatically does this cross- validation process to select CP and build a default pruned tree, but the user can, instead, specify an alternate value of CP (e.g., if you wanted to see what a deeper tree looked like).

Best-Pruned Tree

- Aiming at model parsimony, we can incorporate the sampling error, which might cause this minimum to vary if we had a different sample.
- This enhancement uses the estimated standard error of the cross-validation error (`xstd`) to prune the tree even further; we can add one standard error to the minimum `xerror`.
- This is sometimes called the *Best-Pruned Tree*.

Classification Rules from Trees

- Classification trees provide easily understandable.
- Each terminal node is equivalent to a classification rule.

Regression Trees

- When the outcome variable is numerical, our problem is a regression one.
- Likewise in classification, many splits are attempted,
  - And for each, we measure "impurity" in each branch of the resulting tree.
  - The tree procedure then selects the split that minimizes the sum of such measures.

Measuring Impurity

- An usual impurity measure is the sum of the squared deviations from the mean of the terminal node.
- This is equivalent to the sum of the squared errors, because the mean of the terminal node is exactly the prediction.

Evaluating Performance

As stated above, predictions are obtained by averaging the outcome values in the nodes. We therefore have the usual definition of predictions and errors. The predictive performance of regression trees can be measured in the same way that other predictive methods are evaluated (e.g., linear regression), using summary measures such as RMSE.

Random Forests

Random forests work as follows:

1. Draw multiple random samples, with replacement, from the data (this sampling approach is called the *bootstrap*).
2. Using a random subset of predictors at each stage, fit a classification (or regression) tree to each sample (and thus obtain a "forest").
3. Combine the predictions/classifications from the individual trees to obtain improved predictions:
    - Use voting for classification and averaging for regression.

- Random forests can produce "variable importance" scores, which quantify the relative contribution of the different predictors.

Boosted Trees

The second type of multitree improvement is *boosted trees*.

Here a sequence of trees is fitted, so that each tree concentrates on misclassified records from the previous tree.

1. Fit a single tree.
2. Draw a sample that gives higher selection probabilities to misclassified records.
3. Fit a tree to the new sample.
4. Repeat Steps 2 and 3 multiple times.
5. Use weighted voting to classify records, with heavier weight for later trees.

Advantages and Weaknesses of a Tree

- Tree methods are good ready to use classifiers and predictors.
- They are also useful for variable selection, with the most important predictors usually showing up at the top of the tree.
- Trees are also intrinsically robust to outliers, since the choice of a split depends on the ordering of values and not on the absolute magnitudes of these values.
    - However, they are sensitive to changes in the data, and even a slight change can cause very different splits!
- Unlike models that assume a particular relationship between the outcome and predictors (e.g., a linear relationship such as in linear regression), classification and regression trees are nonlinear and non-parametric.
    - This allows for a wide range of relationships between the predictors and the outcome variable.
    - But this can also be a weakness: Since the splits are done on one predictor at a time, rather than on combinations of predictors, the tree is likely to miss relationships between predictors, in particular linear structures like those in linear or logistic regression models.

Advantages and Weaknesses of a Tree

- Classification trees are useful classifiers in cases where horizontal and vertical splitting of the predictor space adequately divides the classes. But consider, for instance, a dataset with two predictors and two classes, where separation between the two classes is most obviously achieved by using a diagonal line:
  - In such cases, a classification tree is expected to have lower performance than methods such as discriminant analysis.
    - One way to improve performance is to create new predictors that are derived from existing predictors, which can capture hypothesized relationships between predictors (similar to interactions in regression models).
    - Random forests are another solution in such situations.
- Classification trees is that they require a large dataset in order to construct a good classifier:
  - From a computational aspect, trees can be relatively expensive to grow, because of the multiple sorting involved in computing all possible splits on every variable.
  - Pruning the data using the validation sets adds further computation time.

Advantages and Weaknesses of a Tree

- Although trees are useful for variable selection, one challenge is that they "favor" predictors with many potential split points.
  - This includes categorical predictors with many categories and numerical predictors with many different values. Such predictors have a higher chance of appearing in a tree.
  - One simplistic solution is to combine multiple categories into a smaller set and bin numerical predictors with many values.
  - Alternatively, some special algorithms avoid this problem by using a different splitting criterion.
- An appealing feature of trees is that they handle missing data without having to impute values or delete records with missing values.
- Finally, a very important practical advantage of trees is the transparent rules that they generate.
  - Such transparency is often useful in managerial applications, though this advantage is lost in the ensemble versions of trees (random forests, boosted trees).