View on GitHub

stats-learning-notes

Notes from Introduction to Statistical Learning

Previous: Chapter 7 - Moving Beyond Linearity

Chapter 8 - Tree-Based Methods

Tree-based methods, also known as decision tree methods, involve stratifying or segmenting the predictor space into a number of simple regions. Predictions are then made using the mean or the mode of the training observations in the region to which the predictions belong. These methods are referred to as trees because the splitting rules used to segment the predictor space can be summarized in a tree.

Though tree-based methods are simple and useful for interpretation, they typically aren't competitive with the best supervised learning techniques. Because of this, approaches such as bagging, random forests, and boosting have been developed to produce multiple trees which are then combined to yield a since consensus prediction. Combining a large number of trees can often improve prediction accuracy at the cost of some loss in interpretation.

Regression Trees

Decision trees are typically drawn upside down with the leaves or terminal nodes at the bottom of the tree. The points in the tree where the predictor space is split are referred to as internal nodes. The segments of the tree that connect nodes are referred to as branches.

Regression trees are calculated in roughly two steps:

1. Divide the predictor space, $x_1,\ x_2,\ \ldots,\ x_p$ into J distinct and non-overlapping regions, $R_1,\ R_2,\ \ldots,\ R_J$.
2. For every observation that falls into the region R_j, make the same prediction, which is the mean value of the response values for the training observations in R_j.

To determine the appropriate regions, $R_1,\ R_2,\ \ldots,\ R_J$, it is preferable to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and ease of interpretation. Ideally the goal would be to find regions that minimize the residual sum of squares given by

$$\sum_{j=1}^{J}\sum_{i\in R_j}(y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations in the jth box. That said, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, a top-down, greedy approach known as recursive binary splitting is used.

Recursive binary splitting is considered top-down because it begins at the top of the tree where all observations belong to a single region, and successively splits the predictor space into two new branches. Recursive binary splitting is greedy because at each step in the process the best split is made relative to that particular step rather than looking ahead and picking a split that will result in a better split at some future step.

At each step the predictor X_j and the cutpoint s are selected such that splitting the predictor space into regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in the residual sum of squares. This means that at each step, all the predictors $X_1,\ X_2,\ \ldots,\ X_j$ and all possible values of the cutpoint s for each of the predictors are considered. The optimal predictor and cutpoint are selected such that the resulting tree has the lowest residual sum of squares compared to the other candidate predictors and cutpoints.

More specifically, for any j and s that define the half planes

$$R_1(j, s) = \{X|X_j < s\}$$

and

$$R_2(j, s) = \{X|X_j \geq s\},$$

the goal is to find the j and s that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for the training observations in the respective regions.

Only one region is split each iteration. The process concludes when some halting criteria are met.

Tree Pruning

The process outlined above often results in overly complex trees that overfit the data leading to poor test performance. A smaller tree often leads to lower variance and better interpretation at the cost of a little bias.

One option might be to predicate the halting condition on the reduction in the residual sum of squares, but this tends to be short sighted since a small reduction in RSS might be followed by a more dramatic reduction in RSS in a later iteration.

Because of this it is often more fruitful to grow a large tree, T_0, and then prune it down to a more optimal subtree.

Ideally, the selected subtree should have the lowest test error rate. However, this is impractical in practice since there a huge number of possible subtrees. Computationally it would be preferable to select a small set of subtrees for consideration.

Cost complexity pruning, also known as weakest link pruning, reduces the possibility space to a sequence of trees indexed by a non-negative tuning parameter, α.

For each value of α there corresponds a subtree, $T \subset T_0$, such that

$$\sum_{m=1}^{|T|} \sum_{i:X_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|,$$

where $|T|$ indicates the number of terminal nodes in the tree T, R_m is the predictor region corresponding to the mth terminal node and $\hat{y}_{R_m}$ is the predicted response associated with R_m (the mean of the training observations in R_m).

The tuning parameter α acts as a control on the trade-off between the subtree's complexity and its fit to the training data. When α is zero, then the subtree will equal T_0 since the training fit is unaltered. As α increases, the penalty for having more terminal nodes increases, resulting in a smaller subtree.

As α increases from zero, the pruning process proceeds in a nested and predictable manner which makes it possible to obtain the whole sequence of subtrees as a function of α easily.

A validation set or cross-validation can be used to select a value of α. The selected value can then be used on the full training data set to produce a subtree corresponding to α. This process is summarized below.

1. Use recursive binary splitting to grow a large tree from the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree to obtain a sequence of best subtrees as a function of α.
3. Use k-fold cross validation to choose α. Divide the training observations into k folds. For each $k = 1,\ k = 2,\ \ldots,\ k = K$:
    1. Repeat steps 1 and 2 on all but the kth fold of the training data.
    2. Evaluate the mean squared prediction error on the data in the left-out kth fold as a function of α. Average the results for each value of α and pick α to minimize the average error.
4. Return the subtree from step 2 that corresponds to the chosen value of α.

Classification Trees

A classification tree is similar to a regression tree, however it is used to predict a qualitative response. For a classification tree, predictions are made based on the notion that each observation belongs to the most commonly occurring class of the training observations in the region to which the observation belongs.

In interpreting a classification tree, the class proportions within a particular terminal node region are often also of interest.

Growing a classification tree is similar to growing a regression tree, however residual sum of squares cannot be used as a criteria for making binary splits and instead classification error rate is used. Classification error rate for a given region is defined as the fraction of training observations in that region that do not belong to the most common class. Formally,

$$E = 1 - max_k(\hat{p}_{mk})$$

where $\hat{p}_{mk}$ represents the proportion of the training observations in the mth region that are from the kth class.

In practice, it turns out that classification error rate is not sensitive enough for tree growing and other criteria are preferable.

The Gini index is a measure of the total variance across the K classes defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where, again, $\hat{p}_{mk}$ represents the proportion of the training observations in the mth region that are from the kth class.

The Gini index can be viewed as a measure of region purity as a small value indicates the region contains mostly observations from a single class.

An alternative to the Gini index is cross-entropy, defined as

$$D = \sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Since $\hat{p}_{mk}$ must always be between zero and one it reasons that $\hat{p}_{mk} \log \hat{p}_{mk} \geq 0$. Like the Gini index, cross-entropy will take on a small value if the mth region is pure.

When growing a tree, the Gini index or cross-entropy are typically used to evaluate the quality of a particular split since both methods are more sensitive to node purity than classification error rate is.

When pruning a classification tree, any of the three measures can be used, though classification error rate tends to be the preferred method if the goal of the pruned tree is prediction accuracy.

Compared to linear models, decision trees will tend to do better in scenarios where the relationship between the response and the predictors is non-linear and complex. In scenarios where the relationship is well approximated by a linear model, an approach such as linear regression will tend to better exploit the linear structure and outperform decision trees.

Advantages and Disadvantages of Trees

Advantages:

- Trees are easy to explain; even easier than linear regression.
- Trees, arguably, more closely mirror human decision-making processes than regression or classification.
- Trees can be displayed graphically and are easily interpreted by non-experts.
- Trees don't require dummy variables to model qualitative variables.

Disadvantages:

- Interpretability comes at a cost; trees don't typically have the same predictive accuracy as other regression and classification methods.
- Trees can lack robustness. Small changes in the data can cause large changes in the final estimated tree.

Aggregating many decision trees using methods such as bagging, random forests, and boosting can substantially improve predictive performance and help mitigate some of the disadvantages of decision trees.

Bootstrap aggregation, or bagging, is a general purpose procedure for reducing the variance of statistical learning methods that is particularly useful for decision trees.

Given a set of n independent observations, $Z_1,\ Z_2,\ \ldots,\ Z_n$, each with a variance of σ^2, the variance of the mean $\bar{Z}$ of the observations is given by $\frac{\sigma^2}{n}$. In simple terms, averaging a set of observations reduces variance. This means that taking many training sets from the population, building a separate predictive model from each, and then averaging the predictions of each model can reduce variance.

More formally, bagging aims to reduce variance by calculating $\hat{f}^{*1}(x),\ \hat{f}^{*2},\ \ldots,\ \hat{f}^{*B}$ using B separate training sets created using bootstrap resampling, and averaging the results of the functions to obtain a single, low-variance statistical learning model given by

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

Bagging can improve predictions for many regression methods, but it's especially useful for decision trees.

Bagging is applied to regression trees by constructing B regression trees using B bootstrapped training sets and averaging the resulting predictions. The constructed trees are grown deep and are not pruned. This means each individual tree has high variance, but low bias. Averaging the results of the B trees reduces the variance.

In the case of classification trees, a similar approach can be taken, however instead of averaging the predictions, the prediction is determined by the most commonly occurring class among the B predictions or the mode value.

The number of trees, B, is not a critical parameter with bagging. Picking a large value for B will not lead to overfitting. Typically, a value of B is chosen to ensure the variance and error rate of settled down.

Out of Bag Error Estimation

It can be shown that, on average, each bagging tree makes use of around two-thirds of the training observations. The remaining third of the observations that are not used to fit a given bagged tree are referred to as the out-of-bag observations. An approximation of the test error of the bagged model can be obtained by taking each of the out-of-bag observations, evaluating the $\frac{B}{3}$ predictions from those trees that did not use the given out-of-bag prediction, taking the mean/mode of those predictions, and comparing it to the value predicted by the bagged model, yielding the out-of-bag error. When B is sufficiently large, out-of-bag error is nearly equivalent to leave-one-out cross validation.

Variable Importance Measures

Bagging improves prediction accuracy at the expense of interpretability. Averaging the amount that the residual sum of squares decreased for a given predictor over all B trees can be useful as a metric on the importance of the given predictor. Similarly, the decrease in the Gini index can be used as a metric on the importance of the given predictor for classification models.

Random Forests

Random forests are similar to bagged trees, however, random forests introduce a randomized process that helps decorrelate trees.

During the random forest tree construction process, each time a split in a tree is considered, a random sample of m predictors is chosen from the full set of p predictors to be used as candidates for making the split. Only the randomly selected m predictors can be considered for splitting the tree in that iteration. A fresh sample of m predictors is considered at each split. Typically $m \approx \sqrt{p}$ meaning that the number of predictors considered at each split is approximately equal to the square root of the total number of predictors, p. This means that at each split only a minority of the available predictors are considered. This process helps mitigate the strength of very strong predictors, allowing more variation in the bagged trees, which ultimately helps reduce correlation between trees and better reduces variance. In the presence of an overly strong predictor, bagging may not outperform a single tree. A random forest would tend to do better in such a scenario.

On average, $\frac{p-m}{p}$ of the splits in a random forest will not even consider the strong predictor which causes the resulting trees to be less correlated. This process is a kind of decorrelation process.

As with bagging, random forests will not overfit as B is increased, so a value of B should be selected that allows the error rate to settle down.

Boosting

Boosting works similarly to bagging, however, where as bagging builds each tree independent of the other trees, boosting trees are grown using information from previously grown trees. Boosting also differs in that it does not use bootstrap sampling. Instead, each tree is fit to a modified version of the original data set. Like bagging, boosting combines a large number of decision trees, $\hat{f}^{*1}, \hat{f}^{*2}, \ldots, \hat{f}^{*B}$.

Each new tree added to a boosting model is fit to the residuals of the model instead of the response, Y.

Each new decision tree is then added to the fitted function to update the residuals. Each of the trees can be small with just a few terminal nodes, determined by the tuning parameter, d.

By repeatedly adding small trees based on the residuals, $\hat{f}$ will slowly improve in areas where it does not perform well.

Boosting has three tuning parameters:

- B, the number of trees. Unlike bagging and random forests, boosting can overfit if B is too large, although overfitting tends to occur slowly if at all. Cross validation can be used to select a value for B.
- λ, the shrinkage parameter, a small positive number that controls the rate at which the boosting model learns. Typical values are 0.01 or 0.001, depending on the problem. Very small values of λ can require a very large value for B in order to achieve good performance.
- d, the number of splits in each tree, which controls the complexity of the boosted model. Often $d = 1$ works well, in which case each tree is a stump consisting of one split. This approach yields an additive model since each involves only a single variable. In general terms, d is the interaction depth of the model and it controls the interaction order of the model since d splits can involve at most d variables.

With boosting, because each tree takes into account the trees that came before it, smaller trees are often sufficient. Smaller trees also aid interpretability.

An algorithm for boosting regression trees:

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1,\ b = 2,\ \ldots,\ b = B,$ repeat:
    1. Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r)
    2. Update $\hat{f}$ by adding a shrunken version of the new tree:

$$\hat{f}(x) \Leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

    3. Update the residuals:

$$r_i \Leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Next: Chapter 9 - Support Vector Machines

stats-learning-notes maintained by tdg5

Published with GitHub Pages