Notes on Statistical and Machine Learning

Trees-based Methods

Chapter: *13* **Prepared by:** *Chenxi Zhou*

This note is produced based on

- *Chapter 9, Additive Models, Trees, and Related Methods* in Hastie, Tibshirani, and Friedman (2009), and
- *Chapter 9, Recursive Partitioning and Tree-based Methods* in Izenman (2009).

I. Overview

1. **Overview:** Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.
2. **Recursive Binary Tree:** Consider the case where the response variable is continuous.
   (a) *Procedure:*
      i. First split the feature space into two regions, and model the response by the mean in each region. We choose the variable and split-point to achieve the best fit;
      ii. Then, one or both of these regions are further split into two more regions, and this process is continued, until some stopping rule is applied.
   (b) *Advantages:* The recursive tree is very easy to interpret. The feature space partition can be fully described by a single tree.

II. Regression Tree

1. **Setup:** Let data be $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i := (x_{i,1}, x_{i,2}, \cdots, x_{i,p})^\top \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, for all $i = 1, \cdots, n$.
2. **Computation Goal:** The algorithm needs to automatically decide on
   - the splitting variables and splitting points, and
   - what topology (shape) the tree should have.
3. **Constant Fitted Value Given a Partition:** Suppose
   (a) we have partitioned the feature space into M regions $R_1, R_2, \cdots, R_M$;

(b) we model the response as a *constant* c_m in each region:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{1}(\mathbf{x} \in R_m);$$

(c) we use the squared-error loss function $\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$ as the minimization criterion.

The best $\hat{c}_m$ is the average of y_i in region R_m:

$$\hat{c}_m = \frac{1}{\big|\{i \,|\, \mathbf{x}_i \in R_m\}\big|} \sum_{\{i \,|\, \mathbf{x}_i \in R_m\}} y_i. \tag{1}$$

4. Greedy Algorithm to Find the Best Binary Partition:

(a) *Why Greedy Algorithm:* Finding the best binary partition in terms of the minimum sum of squares is generally computationally infeasible.

(b) *Procedure:* Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{X \,|\, X_j \le s\}, \qquad \text{and} \qquad R_2(j, s) = \{X \,|\, X_j > s\}. \tag{2}$$

Seek the splitting variable j and split point s that solve

$$\min_{j,s}\left\{\min_{c_1} \sum_{\{i \,|\, \mathbf{x}_i \in R_1(j,s)\}} (y_i - c_1)^2 + \min_{c_2} \sum_{\{i \,|\, \mathbf{x}_i \in R_2(j,s)\}} (y_i - c_2)^2\right\}. \tag{3}$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = \frac{1}{\big|\{i \,|\, \mathbf{x}_i \in R_1(j,s)\}\big|} \sum_{\{i \,|\, \mathbf{x}_i \in R_1(j,s)\}} y_i,$$
$$\hat{c}_2 = \frac{1}{\big|\{i \,|\, \mathbf{x}_i \in R_2(j,s)\}\big|} \sum_{\{i \,|\, \mathbf{x}_i \in R_2(j,s)\}} y_i.$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible.

(c) *Complete procedure:* Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. This process is repeated on all of the resulting regions.

5. Notation: Let T be a tree.

(a) $|T|$ be the number of terminal nodes in T,

(b) $n_m := \big|\{i \,|\, \mathbf{x}_i \in R_m\}\big|$,

(c) $\hat{c}_m := \frac{1}{n_m} \sum_{\{i \,|\, \mathbf{x}_i \in R_m\}} y_i$, and

(d) $Q_m(T) := \frac{1}{n_m} \sum_{\{i \,|\, \mathbf{x}_i \in R_m\}} (y_i - \hat{c}_m)^2$.

6. **Subtree:** Let T be a tree. A tree $\widetilde{T}$ is said to be a *subtree* of T, denoted by $\widetilde{T} \subseteq T$, if $\widetilde{T}$ is any tree that can be obtained by pruning T, that is, collapsing any number of its internal (non-terminal) nodes.

7. **Tree Size:**

(a) *Overview:* Tree size is a tuning parameter in growing a regression tree, and governs the model complexity:

i. A very large tree might overfit the data, but

ii. a too small tree might *not* capture the important structure and lead to a large bias.

The optimal tree size should be adaptively chosen from data.

(b) *Cost-complexity Pruning:* Grow a large tree T_0, stopping the splitting process only when some minimum node size is reached. Then, prune this large tree using *cost-complexity pruning*

$$C_\alpha(T) := \sum_{m=1}^{|T|} n_m Q_m(T) + \alpha \cdot |T|. \tag{4}$$

For each $\alpha \geq 0$, find the subtree $T_\alpha \subset T_0$ that minimizes C_α.

(c) *Role of α:* The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data:

i. large values of α result in smaller trees T_α, and

ii. smaller values of α result in a larger tree.

The solution when $\alpha = 0$ is the full tree T_0.

(d) *Finding T_α That Minimizes C_α:*

- Uniqueness of the solution: For each $\alpha \geq 0$, there exists a *unique* smallest subtree T_α that minimizes C_α.
- Weakest link pruning:
  - Procedure: We successively collapse the internal node that produces the *smallest* per-node increase in $\sum_m n_m Q_m(T)$, and continue until we produce the single-node (root) tree.
  - Validity: This procedure gives a (finite) sequence of subtrees. It can be shown that this sequence must contain T_α.

(e) *Choosing α by Cross-validation:* The optimal choice of $\alpha \geq 0$ can be achieved by five- or ten-fold cross-validation. That is, we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T_{\hat{\alpha}}$.

III. Classification Tree

1. **Setup:** Let data be $\{(\mathbf{x}_i, g_i)\}_{i=1}^n$, where $\mathbf{x}_i := (x_{i,1}, x_{i,2}, \cdots, x_{i,p})^\top \in \mathbb{R}^p$ and $g_i \in \mathcal{W} := \{1, 2, \cdots, W\}$, for all $i = 1, \cdots, n$.

2. **Classification Rule:** In a node m that represents a region R_m with n_m observations, let

$$\hat{p}_{m,w} := \frac{1}{n_m} \sum_{\{i \,|\, \mathbf{x}_i \in R_m\}} \mathbb{1}(g_i = w), \tag{5}$$

which is the proportion of observations in Class w in node m. We classify observations in node m to the class

$$w^*(m) := \underset{w \in \mathcal{W}}{\arg\max} \ \hat{p}_{m,w},$$

the majority class in node m.

3. **Splitting Criterion:** In the classification tree, we use the following criteria for searching for splitting variables and splitting values:

   (a) *Misclassification error:*

$$\frac{1}{n_m} \sum_{\{i \,|\, \mathbf{x}_i \in R_m\}} \mathbb{1}(y_i \neq w^*(m)) = 1 - \hat{p}_{m,w^*(m)}; \tag{6}$$

   (b) *Gini index:*

$$\sum_{w \neq w'} \hat{p}_{m,w} \hat{p}_{m,w'} = \sum_{w=1}^{W} \hat{p}_{m,w}(1 - \hat{p}_{m,w}). \tag{7}$$

   (c) *Cross-entropy or deviance:*

$$-\sum_{w=1}^{W} \hat{p}_{m,w} \log \hat{p}_{m,w}. \tag{8}$$

Remark 1. If $K = 2$ so that we only have two classes, if p is the proportion in the second class, the misclassification error, the Gini index, and the cross-entropy become

$$1 - \max(p, 1-p), \qquad 2p(1-p), \qquad -p \log p - (1-p)\log(1-p),$$

respectively.

Remark 2. The cross-entropy and the Gini index are differentiable but the misclassification error is not. Thus, the former two are more amenable to numerical optimization.

Remark 3. The cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification error.

4. Interpretations of Gini Index:

(a) Rather than classify observations to the majority class in the node, we could classify them to Class w with probability $\hat{p}_{m,w}$. Then, the expected training error rate of this rule in the node is $\sum_{w \neq w'} \hat{p}_{m,w}\hat{p}_{m,w'}$, which is the Gini index.

(b) If we code each observation as 1 for Class w and zero otherwise, the *variance* over the node of this 0-1 response is $\hat{p}_{m,w}(1-\hat{p}_{m,w})$. Summing over W different classes again gives the Gini index.

IV. Some Practical Issues

1. Dealing With Categorical Variables:

(a) *Difficulty:* When splitting a predictor having q possible unordered values, there are $2^q - 1$ possible partitions of the q values into two groups. Computation can be prohibitively expensive for large q.

(b) *Special case for 0-1 outcome:* With the 0-1 outcome, we can simplify the computation as below:

i. Order the predictor classes according to the proportion falling in outcome Class 1;

ii. Split this predictor as if it were an ordered predictor.

This procedure gives the optimal split in terms of cross-entropy or Gini index.

This procedure does not *work for multi-category outcomes.*

(c) *Comments:* The partitioning algorithm tends to favor categorical predictors with too large q — the number of partitions grows exponentially in q, and the more choices we have, the more likely we can find a good one for the data at hand. This can lead to severe <u>overfitting</u> if q is large, and such variables should be avoided.

2. Asymmetric Loss Matrix:

(a) *Setup:* In classification problems, the consequences of misclassifying observations are more serious in some classes than others. Define a $W \times W$ loss matrix $\mathbf{L}$, with $\mathbf{L}_{w,w'}$ being the loss incurred for classifying an observation in Class w to Class w'. Typically, $\mathbf{L}_{w,w} = 0$, i.e., *no* loss is incurred for correct classifications.

(b) *Incorporating Asymmetric Loss Matrix into Fitting Classification Tree:*

- For <u>two-class</u> problems, weight the observations in Class w by $\mathbf{L}_{w,w'}$.
  Remark 1. This works for the multi-class problems only if $\mathbf{L}_{w,w'}$ does *not* depend on w' as a function of w.
  Remark 2. The effect of observation weighting is to alter the prior probability on the classes.
- For <u>multi-class</u> problems, modify the Gini index to

$$\sum_{w \neq w'} \mathbf{L}_{w,w'}\hat{p}_{m,w}\hat{p}_{m,w'}.$$

This is the expected loss incurred by the randomized rule.

3. **Imputation of Missing Values in Predictors:**

   (a) *Motivation:* Suppose data have some missing values in predictors. One approach is to discard any observation with missing values, which may lead to serious depletion of the training dataset.

   (b) *Imputation strategies:*

   - Impute the missing values with the mean/median of that predictor over the non-missing values.
   - For categorical predictors, simply make a new category for "missing".
   - Constructing *surrogate variables*:
     i. When considering a predictor for a split, we use only the *observations* for which that predictor is *not* missing.
     ii. Having chosen the best primary predictor and splitting point, form a list of surrogate predictors and splitting points.

   When sending observations down the tree, we use the surrogate splits in order, if the primary splitting predictor is missing.
   Remark. Surrogate splits explores the *correlations* between predictors to try and alleviate the effect of the missing data. The higher the correlation between the missing predictor and the other predictors, the smaller the loss of information due to the missing value.

4. **From Binary Splits to Multiway Splits:** Rather than splitting each node into two groups at each stage, we can consider *multiway* splits into more than two groups. However, this is *not* a good strategy, since the multiway splits fragment the data too quickly, leaving *insufficient* data at the next level down. Hence, we would want to use such splits only when needed.

5. **Linear Combination Splits:** Rather than restricting splits to be of the form $X_j \leq s$, one can allow splits along linear combinations of the form $\sum_j a_j X_j \leq s$, where the weights a_j's and split point s are optimized to minimize the relevant criterion.

   (a) *Advantage:* This can improve the predictive power of the tree;

   (b) *Disadvantages:*

   i. This approach hurts interpretability;
   ii. Computationally, the discreteness of the split point search precludes the use of a smooth optimization for the weights.

6. **Instability of Trees:**

   (a) *Issue:* Trees have high variance. Often a small change in the data can result in a very different series of splits.

   (b) *Major Cause:* The major reason for this instability is the *hierarchical nature* of the process: the effect of an error in the top split is propagated down to all of the splits below it.
   (c) *Solution:* Bagging averages many trees and may reduce the variance.

7. **Lack of Smoothness:** Trees lack smoothness of the prediction surface. This can degrade performance in the *regression setting*, where we would normally expect the underlying function to be smooth.

8. **Difficulty in Capturing Additive Structure:** Trees have difficulty in modeling *additive structure*, such as $Y = c_1 \mathbb{1}(X_1 < t_1) + c_2 \mathbb{1}(X_2 < t_2) + \varepsilon$, where ε is the zero-mean random noise. This difficult is mainly due to the binary tree structure.

V. PRIM: Bump Huning

1. **Patient Rule Induction Method (PRIM):** The *patient rule induction method (PRIM)*, similar to the tree model described earlier, finds boxes in which the response average is high. Hence, it looks for maximum in the target function, known as *bump hunting*.

2. **Applicability:**
   (a) PRIM is primarily designed for *regression* (quantitative response variable);
   (b) A two-class outcome can be handled simply by coding the response variable as 0 and 1;
   (c) There is no simple way to deal with $W > 2$ classes simultaneously: one approach is to run PRIM separately for each class versus a baseline class.

3. **Procedure:**
   (a) *Top-down Compression:*
      i. Starting with a box containing all of the data, the box is compressed along one face by a small amount, and the observations then falling outside the box are peeled off;
      ii. Then, the process is repeated, stopping when the current box contains some minimum number of data points.

      Choice of Face: The face chosen for compression at each stage is the one resulting in the largest box mean, after the compression is performed.
   (b) *Bottom-up Pasting:* After the top-down sequence is computed, PRIM reverses the process, expanding along any edge, if such an expansion *increases* the box mean. This procedure is called *pasting*.

      Remark. Since the top-down procedure is greedy at each step, such an expansion is often possible.

The top-down compression and bottom-up pasting process is repeated several times, producing a sequence of boxes B_1, B_2, $\cdots$, B_k. Each box contains different numbers of observations and is defined by a set of rules involving a subset of predictors like

$$a_1 \leq X_1 \leq a_2 \qquad \text{and} \qquad b_1 \leq X_2 \leq b_2.$$

Cross-validation can be used to choose the optimal box size.

Algorithm for PRIM is outlined in Algorithm 1.

Algorithm 1 Patient Rule Induction Method (PRIM)

```
1: Start with all of the training data, and a maximal box containing all of the data;
2: Consider shrinking the box by compressing one face, so as to *peel off* the proportion
   $\alpha$ of observations having either the highest values of a predictor $X_j$, or the lowest.
   Choose the peeling that produces the highest response mean in the remaining box
   (Typically $\alpha = 0.05$ or $0.10$.);
3: Repeat Step 2 until some pre-specified minimal number of observations remain in
   the box;
4: Expand the box along any face, as long as the resulting box mean increases;
5: Steps 1-4 give a sequence of boxes, with different numbers of observations in each
   box. Use cross-validation to choose a member of the sequence. Call the box $B_1$;
6: Remove the data in box $B_1$ from the dataset and repeat Steps 2-5 to obtain a
   second box, and continue to get as many boxes as desired.
```

4. **Comparison with Tree-based Methods:** The main difference is that boxes in PRIM are *not* described by binary trees. Consequences are the following:

   (a) interpretation of the collection of rules in PRIM is more difficult, but

   (b) interpretation of individual rule is often simpler.

VI. MARS: Multivariate Adaptive Regression Splines

1. **Overview:** MARS is an adaptive procedure for regression. It can be viewed as

   - a generalization of stepwise linear regression, or
   - a modification of the CART method to improve the performance of the latter in the regression setting.

2. **Basis Functions in MARS:** Basis functions in MARS are piecewise linear basis functions of the form $(x-t)_+$ and $(t-x)_+$, where $(x)_+ := \max\{x, 0\}$. More explicitly, we have

$$(x-t)_+ = \begin{cases} x-t, & \text{if } x \geq t, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$(t-x)_+ = \begin{cases} t-x, & \text{if } x \le t, \\ 0, & \text{otherwise,} \end{cases}$$

The entire collection of basis functions consists of the piecewise linear basis functions for each input variable and at each distinct value of the corresponding variable, i.e.,

$$\mathcal{C} := \big\{(X_j - t)_+, (t - X_j)_+\big\}_{t \in \{x_{1,j}, x_{2,j}, \cdots, x_{n,j}\}, j \in \{1,2,\cdots,p\}}.$$

Remark 1. We call the pair of functions $(X_j - t)_+$ and $(t - X_j)_+$ the *reflected pair*.

Remark 2. If all of the input values are distinct, there are $2np$ basis functions altogether.

Remark 3. Each basis function is univariate and depends only on a single X_j. It is considered as a function over the entire input space $\mathbb{R}^p$.

3. **Model Specification:** The model-building strategy is like a forward stepwise linear regression, but instead of using the original inputs, we are allowed to use functions from the set $\mathcal{C}$ *and* their products. Thus, the model has the form

$$f(\mathbf{x}) = \beta_0 + \sum_{\ell=1}^{M} \beta_\ell h_\ell(\mathbf{x}), \tag{9}$$

   where each h_ℓ either belongs to $\mathcal{C}$, or is a product of two or more functions in $\mathcal{C}$.

4. **Model Fitting:** Given a choice for the h_ℓ, the coefficients $\boldsymbol{\beta}_\ell$ are estimated by minimizing the residual sum-of-squares.

5. **Choosing Basis Functions h_ℓ's:** Let $\mathcal{M}$ denote the set of basis functions that are already in the model.

   We start with only the constant function $h_0(\mathbf{x}) = 1$, and all functions in the set $\mathcal{C}$ are candidate functions. Hence, $\mathcal{M} = \{h_0\}$ at the very beginning.

   At each stage, we consider all products of a function h_ℓ in the model set $\mathcal{M}$ with one of the reflected pairs in $\mathcal{C}$ that has *not* appeared in $\mathcal{M}$ as a new basis function pair. We add the following term to the model $\mathcal{M}$

$$\hat{\beta}_{M+1} h_\ell(\mathbf{x})(X_j - t)_+ + \hat{\beta}_{M+2} h_\ell(\mathbf{x})(t - X_j)_+, \tag{10}$$

   that produces the largest decrease in training error. In (10), $h_\ell \in \mathcal{M}$, and $M = |\mathcal{M}|$, the number of basis functions in the model prior to adding new terms, and $\hat{\beta}_{M+1}$ and $\hat{\beta}_{M+2}$ are coefficients estimated by least squares, along with all the other coefficients in the model.

   Then, this process is continued until the model set $\mathcal{M}$ contains some pre-specified maximum number of terms.

   Remark. Each input variable can appear *at most once* in a product. This prevents the formation of higher-order powers of an input, which increase or decrease too sharply near the boundaries of the feature space.

6. **Avoiding Overfitting:** At the end of the fitting process, we have a large model of the form (9). This model typically overfits the data, and so a *backward deletion procedure* is necessary.

   The term whose removal causes the smallest increase in residual squared error is deleted from the model at each stage, producing an estimated best model $\hat{f}_\lambda$ of each size (number of terms) λ.

   From computational consideration, we use generalized cross-validation to choose the best number of λ, where the *generalized cross-validation* is defined as

$$\mathrm{GCV}(\lambda) := \frac{\sum_{i=1}^{n}(y_i - \hat{f}_\lambda(\mathbf{x}_i))^2}{(1 - M(\lambda)/n)^2}. \tag{11}$$

   In (11), $M(\lambda)$ is the effective number of parameters in the model, which accounts for both

   - the number of terms in the models, and
   - the number of parameters used in selecting the optimal positions of the knots.

   Example. If there are r linearly independent basis functions in the model, and K knots were selected in the forward process, then

$$M(\lambda) = r + cK,$$

   where $c = 3$ in MARS.

   When the model is restricted to be *additive*, we should choose $c = 2$.

7. **Comments on MARS:**

   (a) Functions of form (9) can operate locally and are zero over part of their range. In particular, note that the product $h_\ell(\mathbf{x})(X_j - t)_+$ appearing in (10) is nonzero only when both $h_\ell(\mathbf{x})$ and $(X_j - t)_+$ are nonzero.

   As a result, the regression surface is built up parsimoniously, using nonzero components locally.

   (b) Exploiting the piecewise linear basis functions, computation in MARS is very efficient.

   (c) The forward modeling strategy in MARS is hierarchical, in the sense that multi-way products are built up from products involving terms already in the model. The philosophy here is that a high-order interaction will likely *only* exist if some of its lower-order terms exist as well.

   (d) MARS procedure sets an upper limit on the order of interaction. This can aid in the interpretation of the final model.

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. Vol. 1. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Izenman, Alan J (Mar. 2009). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. en. Springer Science & Business Media.