Yonsei University, Department of Applied Statistics

Chapter 8 Tree-Based Methods

12기 YBIGTA 신보현

February 28, 2019

해당 챕터에서는 회귀와 분류를 위해나 tree-based 방법을 소개한다. 이것은 예측 변수 공간을 많은 간단한 지역으로 stratifying 또는 segmenting하는 것을 포함한다. 이렇게 예측변수 공간을 구분짓기 위해 나누는 규칙이 나무로 요약될 수 있기 때문에 이러한 접근법을 의사결정 나무 방법이라고 부른다.
tree-based 방법은 간단하고 해석하기 용이하다. 하지만 다른 지도학습과 비교하였을 때는 그렇게 성능이 좋지 않다. 따라서 이 챕터에서는 bagging, random forests을 소개한다. 이들은 하나의 예측을 하기 위해 합쳐지는 여러 개의 나무들을 생산하는 접근법이다. 많은 수의 나무를 합쳤을 때, 해석에서의 조금의 손실에 대비해서종종 꽤 정확한 예측을 할 수 있음을 살펴볼 것이다.

8.1 The Basics of Decision Trees

의사결정 나무는 회귀와 분류 문제에 모두 적용될 수 있다. 회귀에서의 의사결정 나무부터 살펴보자.

8.1.1 Regression Trees

Hitters 데이터에서 야구 선수의 Salary을 Years, Hits에 기반하여 예측하는 상황을 생각해보자. 먼저 결측치가 있는 행을 제거하고 Salary 변수에 대해 log 변환을 시행한다.

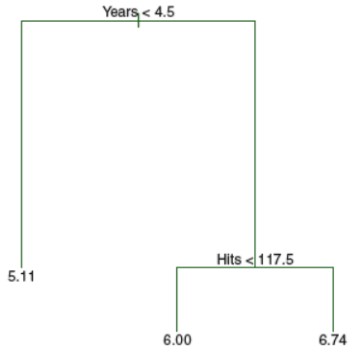


FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

그림 8.1은 해당 데이터에 회귀 나무(regression tree)을 적합한 결과이다. 나무의 위에서부터 시작해서 어떤 기준에 따라 데이터를 나누고 있다. 가장 처음에는 years<4.5에 해당하는 데이터들의 Salary 평균은 5.11이다. 이 값은 log 변환을 한 값이므로, 실제로 years<4.5에 해당하는 선수들의 Salary 평균은 $e^{5.11}$ 이다. years>4.5에 해당하는 선수들은 다시 Hits에 의해서 나뉜다. 전체적으로 이 나무는 공간을 세 개로 나눈다.$R_1 = \{years < 4.5\}, R_2 = \{years > 4.5, Hits < 117.5\}, R_3 = \{years > 4.5, Hits > 117.5\}$

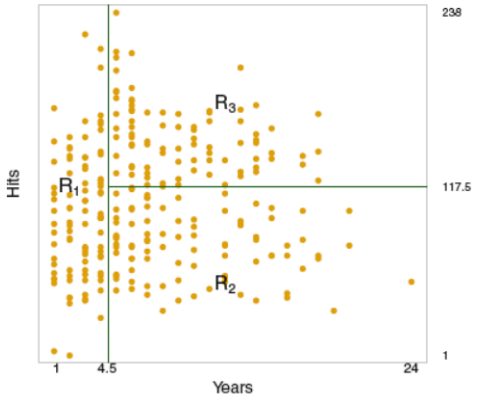


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

나무와의 유사성으로 살펴보면, R_1, R_2, R_3은 나무의 terminal nodes 또는 leaves으로 알려져 있다. 예측 공간이 나뉘는 점은 internal nodes라고 불린다. 그림 8.1에서 internal nodes는 years<4.5와 Hits<117.5이다. node를 연결하는 나무의 부분은 branches라고 부른다.
그림 8.1은 다음과 같이 해석할 수 있다. Saraly을 결정하는데 years가 가장 중요한 요인이고 따라서 경험이 적은 선수는 돈을 적게 번다. 선수의 경험이 적다는 전제 하에는, 선수들 간에 Salary 차이는 그렇게 크지 않다. 하지만 5년 이상의 경험이 있는 선수들은 Hits의 수가 그들의 Salary에 영향을 미쳤는데, 더 많은 Hits를 기록한 선수가 더 높은 Salary을 가졌다. 그림 8.1에 있는 회귀 나무는 진정한 관계에 대해서 지나치게 단순화한 그림이지만 해석하기 쉽고 좋은 그래프적인 표현이 있다는 장점이 있다.
이제 회귀 나무를 어떻게 생성하는지 알아보자.

1. 예측 변수 공간(predictor space, 즉 $X_1, \cdots X_p$의 가능한 값들)을 서로 겹치지 않는 지역, $R_1, R_2, \cdots R_J$로 나눈다.

2. R_j에 속하는 모든 관측치에 대해서 동일한 예측을 하는데 즉, R_j에 속하는 훈련 데이터들의 평균 값으로 예측을 한다.

예를 들어서, 1단계에서 지역을 R_1, R_2로 나누고 각 지역에 속하는 관측치의 평균이 각각 10, 20이라면 R_1에 속하는 모든 관측치에 대해서는 그에 해당하는 반응변수가 10이라고 예측을 하는 것이다.
이제 1단계를 더 상세하게 보자. 어떻게 J개의 지역을 만들까? 이론적으로 어떤 모양도 될 수가 있지만 해석의 용이성을 위해 예측변수 공간을 고차원의 직사각형, 혹은 박스 모양으로 나눈다. 즉, SSE를 최소로 하는 $R_1, \cdots, R_J$를 찾는 것이다. 여기서 SSE는 아래와 같이 정의된다.

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

여기서 $\hat{y}_{R_j}$는 j 번째 상자에 있는 훈련 관측치들의 평균 값이다. 계산상의 이유로, 예측변수 공간을 J개의 상자로 쪼개는 모든 경우의 수를 고려할 수는 없다. 이러한 이유로, top-down 접근법, 즉 recursive binary splitting이라고

알려진 욕심 많은(greedy) 접근법을 사용한다. 이러한 접근법은 top-down인데 이유는 나무의 위에서부터(모든 관측치가 하나의 지역에 포함되어 있는 시점) 시작하고 연속적으로 예측공간 변수를 나누기 때문이다. 각각의 분리(split)는 새로운 두 개의 branches을 통해서 표시된다. 이러한 접근법은 greedy한데 그 이유는 나무를 만드는 각 단계마다 이후 단계에서 좋은 결과를 가져올 것으로 기대되는 분리보다는 그 특정한 단계에서 가장 좋은 분리가 만들어지기 때문이다.
recursive binary splitting을 실행하기 위해서 우선 한 예측변수 X_j를 선택하고 어떤 cutpoint인 s을 정하여 예측변수 공간을 $\{X \mid X_j < s\}, \{X \mid X_j \geq s\}$로 나누는데, 가능한 가장 많은 SSE의 감소를 가져오는 s를 선택한다. 즉, 모든 예측 변수 $X_1, \cdots, X_p$와 각 예측변수에 대한 모든 가능한 cutpoint인 s값을 고려하는데 이 때 가장 작은 SSE를 만드는 s을 선택하는 것이다. 모든 값 j, s에 대해서 다음의 두 지역을 정의하고

$$R_1(j,s) = \{X \mid X_j < s\} \; and \; R_2(j,s) = \{X \mid X_j \geq s\}$$

다음의 식을 최소화하는 j, s을 선택한다.

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

여기서 $\hat{y}_{R_1}$는 $R_1(j,s)$에 속하는 훈련 관측치들의 평균 값이다. 예측 변수의 개수가 그렇게 많지 않으면 위 식을 최소화하는 j, s를 찾는데 그렇게 오래 걸리지는 않는다.
다음으로 SSE를 최소화하는 가장 좋은 예측 변수와 cutpoint을 찾으며 이러한 과정을 반복한다. 하지만 이 때에는 전체 예측 변수 공간을 나누는 것이 아니라 이전에 나뉘어진 지역 중 하나만 나눈다. 이제 세 개의 지역이 있다. 다시, 이 세개 중 한 개의 지역을 SSE를 최소화하며 나눈다. 이 과정은 멈추는 기준이 있기 전까지 계속된다. 예를 들어, 어떤 지역도 5개 이상의 관측치를 가지지 않을 때까지 이러한 과정을 반복하는 것이다.
$R_1, \cdots, R_J$가 생성이 되었다면 주어진 테스트 관측치를 그 관측치가 속하는 지역의 평균 값을 이용하여 예측하게 된다.

Tree Pruning

위에서 설명한 과정은 훈련 세트에 대해서는 좋은 예측을 할 수 있겠지만 생성되는 나무가 너무 복잡하여 데이터에 과적합 될 우려가 있다. 더 적은 수의 분리(splits)을 통한 작은 나무(더 적은 수의 지역을 가지는)는 조금의 bias을 비용으로 하여 더 적은 분산을 가지고 해석하기가 용이해진다. 대안으로, 각 단계에서 분리를 시행할 때, 높은 기준을 잡음으로써 분리가 쉽게 되지 않게 하는 것이다. 이러한 전략은 작은 나무를 만들 수는 있지만 나무를 만드는 것을 미리 끝내버려, 이후에 SSE의 큰 감소를 이끄는 분리를 할 수 없을 수도 있는 단점이 있다.
따라서 좀 더 나은 전략은, 매우 큰 나무 T_0을 만든 다음에 작은 나무(subtree)를 얻기 위해서 가지를 치는 것(prune)이다. 직관적으로, 우리는 가장 작은 테스트 오차 비율을 얻는 작은 나무를 얻는 것이 목표이다. 작은 나무가 주어지면, CV나 validation set approach을 통해서 그것의 테스트 오차를 추정할 수 있다. 하지만 모든 가능한 작은 나무에서 CV 오차를 측정하는 것은 너무 오래걸릴 것이다. 대신 작은 수의 작은 나무를 고르기 위한 방법이 필요하다.
weakest link pruning이라고도 알려진 Cost complexity pruning은 이것을 하는 방법을 제시한다. 모든 작은 나무를 고려하는 것이 아닌 어떤 tuning parameter인 α에 의해서 정해진 일부 작은 나무만을 고려한다.

α값에 대해서 아래 식을 최소로 하는 작은 나무 $T \subset T_0$를 찾는다.

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

여기서 $|T|$는 나무 T의 terinal nodes 개수이고 R_m은 m 번째 terminal node에 해당하는 직사각형, 그리고 $\hat{y}_{R_m}$은 R_m의 훈련 관측치의 평균이다. tuning parameter인 α는 작은 나무의 복잡도와 훈련 데이터에 대한 적합간의 trade-off을 조절한다. 만약 $\alpha = 0$이라면 작은 나무 T는 단순히 가장 큰 나무 T_0와 동일 할 것이다. 하지만 α가 증가할 수록 많은 수의 terminal nodes를 가지는 나무에 대한 비용(price)을 지불해야한다. 왜냐하면 α가 증가할 수록 penalty 느낌의 항인 $\alpha|T|$의 영향력이 커질 것이고, α가 클 때에는 많은 수의 terminal nodes, 즉 앞서 정의한 $|T|$가 큰 나무에 대해서는 최소화해야 하는 quantity가 커지므로 비용을 더 지불하는 것이다. 6단원에서 ridge와 lasso의 tuning parameter인 λ와 비슷한 느낌이다. 아래 알고리즘에 여태까지 설명한 내용들이 정리되어 있다.

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on 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 in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) 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.

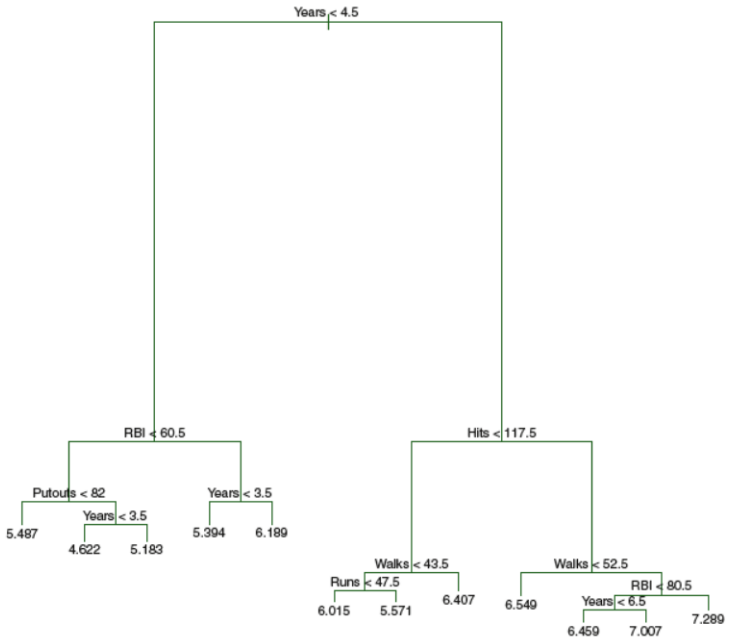


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

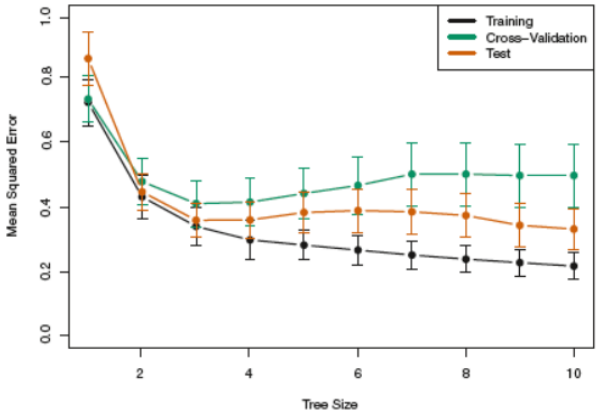


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

그림 8.4와 8.5는 Hitters 데이터에 대해서 9개의 변수를 사용하여 적합한 전체 회귀나무와 가지를 친 회귀나무를 보여준다. 먼저, 데이터를 훈련 세트와 테스트 세트로 나눈다. 그리고 훈련 데이터를 통해서 회귀 나무를 만들고 작은 나무를 만들기 위해서 α를 CV를 통해서 결정했다. 가지를 치기 이전의 회귀 나무는 그림 8.4에 나와있다. 그림 8.5에는 CV 오차와 테스트 오차가 나타나있다. 그림 8.5를 살펴보면 CV 오차가 최소가 되는 지점이 테스트 오차에 대한 추정치가 최소가 되는 지점과 비슷함을 확인할 수 있다. 또한 나무의 사이즈, 즉 세 개의 노드를 가질 때 CV 오차가 가장 작았고 테스트 오차는 이 때 가장 큰 MSE의 감소를 보였다. 물론 테스트 오차에 대한 추정치가 10개의 노드를 가질 때 최소 값을 가지지만 복잡도를 줄이는 차원에서 3개의 노드가 적절해 보인다.

8.1.2 Classification Trees

분류 나무는 범주형 자료를 예측하는 것 이외에는 회귀 나무와 매우 유사하다. 회귀 나무가 같은 terminal node에 있는 훈련 관측치의 평균 값으로 테스트 관측치에 대한 반응 변수를 예측함을 상기해보자. 반면 분류 나무에서는, 수치형 자료가 아니기 때문에 평균 값을 구할 수가 없어, 대신에 그 지역에 속하는 훈련 관측치에서 가장 빈번하게 일어나는 클래스가 회귀 나무에서의 평균 역할을 한다.
분류 나무를 키우는 것은 회귀 나무를 키우는 것과 꽤 유사하다. 회귀 나무에서와 같이 recursive binary splitting을 사용한다. 하지만 분류 나무에서는 SSE가 기준으로 사용되지 않는다. 분류 문제에서 SSE에 자연스럽게 대응되는 기준은 분류 오차 비율(Classification error rate)이다. 그 지역에 있는 훈련 관측치 중 가장 빈번하게 일어나는 클래스에 관측치를 배정하는 것이 계획이기 때문에, 분류 오차 비율은 그 지역에 가장 빈번하게 일어나는 클래스에 속하지 않는 훈련 데이터의 비율이다.

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

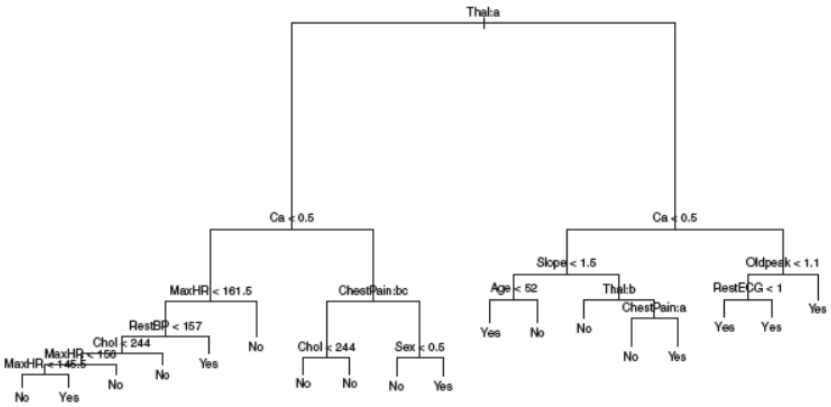
여기서 $\hat{p}_{mk}$는 m번째 지역에 k번째 클래스에 속하는 훈련 데이터의 비율이다. 하지만 이러한 분류 에러가 나무 성장(tree-growing)에 충분히 민감하지 않음이 밝혀졌고 실전에서는 다른 두 가지 방법이 선호된다. 지니 인덱스(Gini Index)의 정의는 아래와 같다.

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

이는 K개의 클래스에 대한 전반적인 분산을 측정한다. 만약 모든 $\hat{p}_{mk}$가 0이나 1에 가깝다면 지니 인덱스는 작은 값을 가진다. 이러한 이유로 지니 인덱스는 node purity에 대한 지표로써 많이 사용된다. 지니 인덱스가 작다는 것은, $\hat{p}_{mk}$가 0이나 1에 가깝다는 것이고, 0에 가깝다는 뜻은 m번째 지역에 k번째 클래스에 속하는 훈련 데이터의 비율이 거의 없다는 뜻이며 1에 가깝다는 뜻은 m번째 지역이 거의 k번째 클래스에 속하는 훈련 데이터로만 이루어져있다는 뜻이므로, 결국 작은 지니 인덱스는 순수하게 어떤 클래스로 이루어져있는지에 대한 지표로 사용될 수 있는 것이다.
지니 인덱스에 대한 대안으로는 entropy가 있다.

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

$0 \leq \hat{p}_{mk} \leq 1$이기 때문에, $0 \leq -\hat{p}_{mk} log \hat{p}_{mk}$이다. 만약 모든 $\hat{p}_{mk}$가 0에 가까운 값을 가진다면 $\hat{p}_{mk} log \hat{p}_{mk} = 0 \times (-\infty)$꼴인데, 0으로 가까워지는 속도가 더 빠르므로 0에 가까워지고 모든 $\hat{p}_{mk}$가 1에 가까운 값을 가진다면 $\hat{p}_{mk} log \hat{p}_{mk} = 1 \times 0$꼴이므로 0에 가까워 진다. 따라서 지니 인덱스와 비슷하게 엔트로피도 m번째 노드의 purity에 대한 지표로 사용된다. 분류 나무를 생성할 때, 지니 인덱스와 엔트로피가 분류 오차보다 node purtiy에 더 민감하므로 특정한 분리의 질을 평가할때 많이 사용된다. 예를 들어 각 클래스에 각 400개의 관측치가 있는 이진 분류 문제를 생각해보자. 이를 (400,400)이라 쓰겠다. 어떤 분리는 노드를 (300,100)과 (100,300)의 분류 결과를 만들고 다른 분리는 (200,400)과 (200,0)으로 분류한다고 하자. 두 분리 모두 분류 오류 비율은 0.25이지만 두 번째 분리가 pure node를 만들어 더 선호될 것이다. 지니 인덱스와 엔트로피 모두 두 번째 분리에서 낮은 값을 가져서 더 pure한 node를 구분할 수 있다.



가끔 가지를 만들어도 분류 결과가 동일하게 생성될 때가 있다. 이는 해당 분리가 node purity의 증가를 가져오기 때문에 시행된 것이다. 즉, 겉으로 보기에 RestECG<1인 노드와 RestECG>=1인 노드에서 가장 빈번하게 나온 값은 YES이지만 그 순도(Purity)가 다르다는 뜻이다. 구체적으로 데이터를 통해서 살펴보자.

```
heart = read.csv("C:/Users/sbh0613/Desktop/와이빅타/ISL/Chapter 8/Heart.csv")
attach(heart)
# The number of YES or NO when the terminal node is RestECG<1
table(heart$AHD[which(heart$Thal != 'normal' & heart$Ca >=0.5 & Oldpeak<1.1 & RestECG<1)])


##
##  No Yes
##   5   9

# The number of YES or NO when the terminal node is RestECG>=1
table(heart$AHD[which(heart$Thal != 'normal' & heart$Ca >=0.5 & Oldpeak<1.1 & RestECG>=1)])


##
##  No Yes
##   0  12
```

위 결과를 보면, RestECG<1일 때는 YES가 많이 있지만 NO도 존재한다. 하지만 RestECG>=1일 때는 오직 YES만 존재하여 그 순도가 1이다. 만약 RestECG에 대해서 노드를 분리하지 않았다면 순도가 1이 될 수는 없을텐데 분리를 한 이후에도 동일하게 YES로 나옴에도 순도를 올리기 위해서 노드를 분리했음을 알 수 있다. 그렇다면 노드의 순도가 왜 중요할까? 만약에 테스트 데이터가 RestECG을 통해 나누기 직전까지의 기준으로 데이터를 YES, NO로 나누었다고 생각해보자. 그렇다면 물론 해당 데이터를 YES로 예측하겠지만 그에 대한 확실함이 떨어질 것이다. 하지만 RestECG<1인 기준을 적용하였을 때는, 1 이상인 값에 대해서는 훈련 데이터의 순도가 1이므로, 매우 확실하게 테스트 데이터에 대해서 YES라고 예측할 것이다. 즉 순도는 얼마나 '확실하게' 분류하는

그 믿음이라고 생각하면 될 듯 하다. 그 믿음을 확실하게 할수록 예측하는 입장에서는 좋기 때문에 node purity가 중요한 것이고 따라서 이러한 node purity을 잘 감지하지 못하는 분류 오류보다는 지니 인덱스나 엔트로피를 더 선호하는 것이다.

8.1.3 Trees Versus Linear Models

선형 회귀는 아래와 같은 모델을,

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

회귀 나무는 아래와 같은 모델을 가정한다.

$$f(X) = \sum_{m=1}^{M} c_m \cdot 1(X \in R_m)$$

선형 회귀는 반응변수와 예측변수 간에 선형성이라는 강력한 가정이 성립한다면 성능이 다른 모델들보다 뛰어나다. 하지만 현실에서 그러한 선형적인 관계는 많지 않다. 비선형적이고 복잡한 관계에 대해서는 나무 기반 모델이 성능이 더 좋을 것이다. 이것뿐만 아니라 나무 기반 모델은 해석과 시각화 측면에서도 뛰어나기 때문에 이를 선택하는 경우도 다반사다.

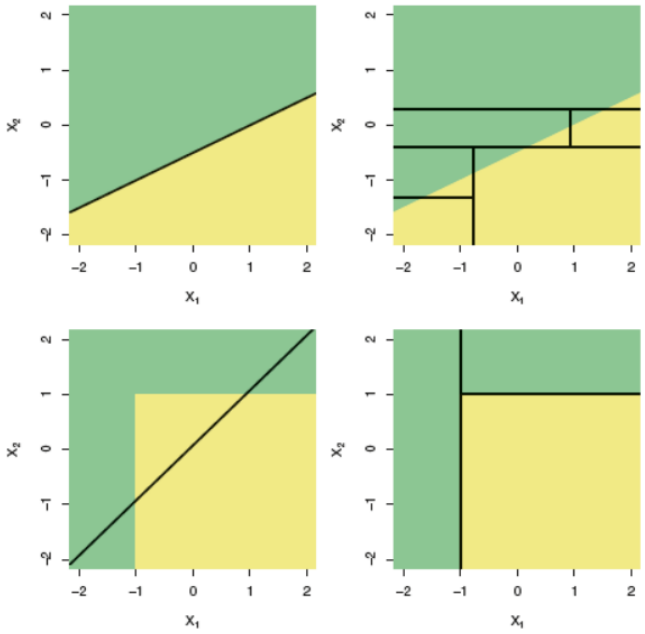


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

8.1.4 Advantages and Disadvantages of Trees

- 나무는 사람들에게 설명하기 굉장히 쉽다. 사실 선형 회귀보다 해석에서 더 강점을 가지고 있다.
- 나무는 그래프적으로 보여질 수 있고 비전문가도 이를 쉽게 해석할 수 있다.
- 선형 회귀는 더미 변수를 만들어서 모델링을 해야 했지만 나무는 범주형 변수에 대해서도 더미 변수를 만들 필요 없이 모델링을 할 수 있다.
- 정확도의 측면에서는 이 책에서 소개된 다른 기법들에 비해서 떨어질 수 있다.
- 나무는 robust하지 않다. 즉 데이터의 작은 변화가 마지막 추정된 나무에 큰 변화를 가져올 수 있다.

따라서 많은 decision trees을 bagging, random forests, boosting을 이용해서 합침으로써 예측력이 상당히 향상될 수 있다. 이제 이 개념들을 살펴보자.

8.2 Bagging, Random Forests, Boosting

8.2.1 Bagging

5단원에서 Bootstrap는 관심이 있는 quantity의 표준 오차를 간접적으로 계산하기 위해 사용되는 강려한 툴이다. 여기서는 decision trees의 성능을 향상시키기 위해 사용되는 bootstrap의 측면을 살펴본다.
decision trees는 높은 분산을 가진다는 단점이 있다. 즉, 이 말은 데이터를 랜덤하게 두 부분으로 나눌 때마다, 해당하는 훈련 데이터에 대한 decision trees가 꽤 다르다는 뜻이다. Bootstrap aggregation 또는 bagging은 통계적 기법의 분산을 줄이기 위해 일반적으로 사용되는 기법이다. 특히, decision trees에 유용하게 사용되므로, 해당 챕터에서 이를 소개한다.
모집단의 분산이 σ^2로 알려진 경우에, 모집단으로부터 n개의 random sample을 $Z_1, \cdots, Z_n$이라 하면 이들의 평균인 $\overline{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$의 분산은 σ^2/n으로 모집단의 분산보다 더 작아진다. 다시 말해서, 관측치를 평균내는 것은 분산을 줄인다. 따라서 분산을 줄이고 통계적 방법의 예측 정확성을 높이기 위해서는 모집단으로부터 많은 훈련 데이터를 가지고, 각각의 훈련 세트를 이용하여 예측 모델을 적합한 뒤, 결과로 나오는 예측치를 평균낸ㄴ 것이다. 다시 말해서 B개의 서로 다른 훈련 세트를 이용하여 $\hat{f}^1(x), \cdots, \hat{f}^B(x)$를 계산하고 $\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$을 구하여 하나의 낮은 분산을 가지는 통계적 학습 모델을 얻는 것이다. 물론 일반적으로 많은 훈련 데이터 세트를 가지고 있지 않기 때문에 이러한 방법은 실용적이지 않다. 대신 bootstrap을 통해서 하나의 훈련 데이터를 통해 반복적으로 복원 추출을 한다면 다수의 훈련 데이터 세트를 생성할 수 있을 것이다. 즉, B번의 서로 다른 bootstrapped 훈련 데이터를 생성하고 이를 통해 b번째 모델을 통한 예측값을 생성하고 이를 모두 평균낸다. 이를 bagging이라고 한다.

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

bagging은 특히 decision trees에 유용하다. bagging을 회귀 나무에 적용하기 위해서 B개의 회귀 나무를 B개의 bootstrapped 훈련 데이터를 통해서 만들고 결과로 나오는 예측을 모두 평균낸다. 이러한 나무는 깊게 생성되고 가지를 치지 않는다. 따라서 각각의 나무는 높은 분산을 가지고 있겠지만 이에 따라 낮은 편향(bias)을 가진다. 이러한 B개의 나무를 평균내는 것은 분산을 줄임이 증명되었다.

그렇다면 분류 나무에도 동일하게 적용이 될까? 주어진 테스트 데이터에 대해서 B개의 나무를 통해서 예측된 클래스를 기록하고 majority vote를 시행한다. 전체적인 예측은 B개의 예측에서 가장 많이 나타난 클래스로 한다. 그림 8.8은 Heart 데이터에 trees을 bagging 함으로써 얻은 결과를 보여준다.

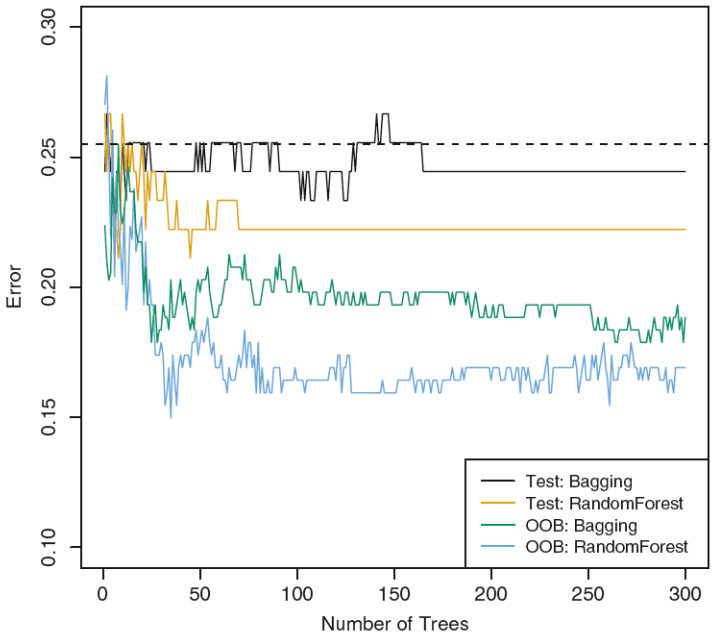


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.*

테스트 오차에 대한 추정치가 B, 즉 나무의 수에 대한 함수로 나타나 있다. 검정색 선이 decision trees에 bagging을 적용해서 나온 데스트 오차에 대한 추정치인데, 하나의 나무를 사용했을 때보다 살짝 낮음을 확인할 수 있다. bagging에서 나무의 수, B는 그렇게 중요한 파라미터가 아니다. 왜냐하면 B를 매우 크게 조정을 해도 과적합으로 이어지지 않기 때문이다. 따라서 현실에서는 B를 충분히 크게 설정한다.

Out-of-Bag Error Estimation

CV를 시행하지 않고도, bagged 모델의 테스트 오차를 추정할 수 있는 아주 직관적인 방법이 있다. bagging의 핵심은 관측치로부터 계속적으로 bootstrapped 샘플을 뽑는 것임을 상기해보자. 평균적으로 각각의 bagged tree는 관측치의 2/3을 사용한다.(5단원 연습문제 2 참조) 나머지 1/3 관측치, 즉 bagged tree을 적합하는데 사용되지 않는 것들을 out-of-bag(OOB) 관측치라고 부른다. OOB 개념을 이용해보자. i번째 관측치에 대한 반응 값을 그 관측치가 OOB이었던 각 나무를 이용해어 예측할 수 있다. 이를 통해 i번째 관측치에 대한 예측값이 B/3개가 나올 것이다. i번째 관측치에 대해 하나의 예측값을 얻기 위해서 이 예측값들을 평균내거나 majority vote를 시행한다. 이는 i번째 관측치에 대한 OOB 예측값이 되는 것이다. OOB 오차는 bagged 모델에서 테스트 오차를 추정하기 위한 좋은 방법인데, 그 이유는 OOB는 bagging에서 뽑힌 샘플이 아니므로 각 단계에서 생성된 bagged tree를 만들 때, OOB가 사용되지 않았기 때문이다(데이터를 훈련 데이터와 테스트 데이터로 나누고 테스트 데이터로

모델 형성에 쓰이지 않은 테스트 데이터를 통해 모델을 평가하는 것과 비슷한 맥락) 특히, OOB 접근법은 데이터가 매우 커서 CV의 계산량이 매우 많을 때, CV 없이도 테스트 오차를 나름 정확하게 추정할 수 있는 큰 장점이 있다.

Variable Importance Measures

앞에서 말했듯이, bagging은 하나의 나무를 사용하는 것보다는 향상된 정확도를 보여준다. 하지만 모델을 해석하기 어려울 수 있다. 하나의 나무만을 사용 했을 때는 그림 8.1과 같이 쉽게 설명을 할 수 있지만 여러 개의 나무를 사용한다면 이러한 장점이 사라질 것이다. 따라서 bagging은 해석력에 대한 대가로 정확성을 얻는다고 할 수 있다. 그럼에도, 회귀 나무의 예에서는 SSE를 통해서, 분류 나무의 경우에는 Gini index을 통해서 각 예측 변수의 중요도를 얻을 수 있다. 회귀 나무를 bagging할 때는, 예측변수에 대해서 분리를 할 때 감소되는 SSE를 기록하고 B개의 나무로 부터 나온 이 SSE를 평균낸 값이 크면 중요한 예측변수라고 하는 것이다. 비슷하게 분류 나무를 bagging할 때에는 Gini index를 기준으로 한다.
그림 8.9에 Heart 데이터의 예측변수가 가지는 중요도가 나타나있다.

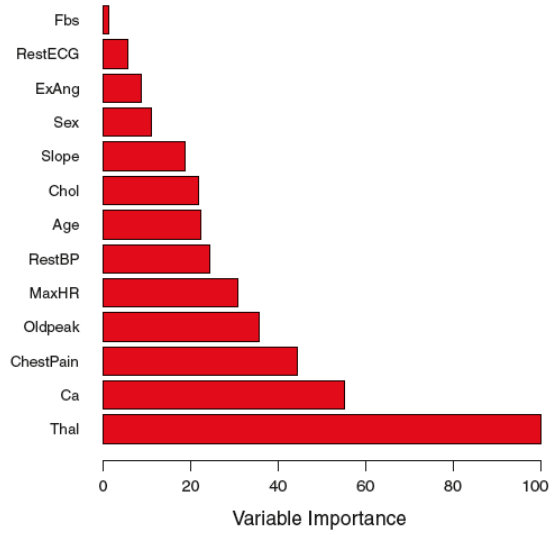


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

8.2.2 Random Forests

random forests은 나무들을 decorrelate하게 수정함으로써 bagged trees의 업그레이드 버전이라고 할 수 있다. decision trees을 만드는 각 과정에서 전체 p개의 예측변수 중에서 랜덤 샘플로 m개의 예측변수가 뽑혀 이 들이 분리 후보로 선정된다. 분리는 이러한 m개의 예측변수 중 하나의 변수만 사용한다. 각 분리 단계에서 새로운 m개의 예측변수 샘플이 뽑히고 보통 $m \approx \sqrt{p}$가 되도록 선정한다.
다시 말해서, random forest에서는 나무의 각 분리에서 알고리즘이 사용 가능한 예측변수의 대다수를 고려하도록 혀용되지 않는다. 왜 이러한 방법을 사용할까? 데이터 세트에 매우 강력한 예측변수가 있고 적당하게 강력한 예측변수가 많이 있다고 하자. 이러한 경우에, bagged trees 생성 과정에서는 가장 강력한 예측변수를 맨 위의

분리에 사용할 것이다. 결과적으로 모든 bagged trees가 서로 꽤 비슷할 것이다. 따라서 bagged trees로부터 나온 예측값들이 매우 correlated할 것이다. 이렇게 서로 높게 correlated된 값들을 평균내는 것은 분산을 크게 감소시키지 않는다. 이러한 상황에서는 bagging이 여러 개의 나무를 사용함에도 불구하고 하나의 나무를 사용할 때에 비해서 분산이 크게 감소되지 않을 것이다.

random forest는 예측변수의 일부만을 각 분리에서 고려하도록 강요함으로써 이러한 문제를 해결한다. 따라서 강력한 변수순으로 고려되기 보다는, 다른 변수들도 분리에 사용될 기회를 더 가져서 나무들이 서로 다른 모습을 가지도록 설정한다. 이러한 과정을 나무들을 decorrelating함으로써 결론적으로 이들의 평균은 분산의 큰 감소를 가져온다고 생각하는 것이다.

bagging과 random forest의 가장 큰 차이점은 일부 예측변수를 뽑는 개수 m이다. 예를 들어 random forest에서 $m = p$로 설정하면, 이것은 bagging을 하는 것과 다름 없다. 그림 8.8에서는 $m = \sqrt{p}$를 사용하였고 이는 테스트 오차 추정치의 큰 감소를 보였다.

서로 상관이 있는 예측변수가 많이 있다면 작은 m을 사용하는 것이 random forest 수행 시에 특히 효과적이다.

bagging과 마찬가지로 random forest는 B가 크더라도 과적합되지 않으므로 충분히 큰 B를 실전에서 많이 사용한다.

8.2.3 Boosting

bagging과 마찬가지로 boosting 또한 통계적 학습 방법을 향상시킬 수 있는 일반적인 접근법이다. 여기서는 decision trees에 한정하여 boosting을 논의한다.

bagging은 bootstrap된 여러 샘플들을 통해 생성된 예측값을 평균이라는 방법으로 혼합하여 하나의 예측값을 만드는데, 이 과정에서 사용되는 bootstrap 샘플은 다른 나무에 독립이다. 이와는 다르게 boosting은 나무들이 연속적으로 생성된다. 각 나무는 이전에 생성된 나무의 정보를 이용하여 생성된다. boosting은 bootstrap 샘플링을 포함하지 않는다. 대신 각 나무는 원래의 데이터 세트의 변형된 버전으로 적합된다. 이를 자세히 살펴보자.

회귀 나무에서의 상황을 살펴보자. bagging과 마찬가지로 boosting 또한 많은 수의 decision trees, $\hat{f}^1, \cdots, \hat{f}^B$를 만든다. boosting 알고리즘은 아래와 같다.

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

하나의 큰 나무를 적합하는 것이 아니라 boosting은 천천히 적합한다. 현재 모델이 주어졌을 때, 그 모델로부터 얻은 잔차에 decision tree을 적합한다. 다시 말해서, 결과인 Y, 반응 값보다는 현재 잔차에 나무를 적합하는 것이다. 이렇게 생성된 decision tree를 적합된 함수에 더하고 잔차를 업데이트 한다. 각각의 나무는 적은 수의 terminal nodes를 가지며 이는 알고리즘에서 파라미터 d에 의해 결정된다. 작은 나무를 잔차에 적합함으로써 $\hat{f}$가 잘 작동하지 못하는 곳, 즉 잔차에서의 성능을 서서히 향상시킨다. shrinkage parameter인 λ는 심지어 더 이러한 과정을 늦추고 잔차에 적합되는 나무들이 더 다양한 모양을 할 가능성을 열어준다. 일반적으로, 느리게 배우는 통계적 학습은 성능이 좋다. 다시 한번 주목하자면, bagging과 다르게 boosting은 이미 생성된 나무에 강력하게 의존한다.
회귀 나무에 boosting을 적용하였을 때를 살펴보았는데 분류 나무에 대한 설명은 더 복잡하므로 생략하도록 한다. boosting은 세 개의 파라미터를 가진다.

1. 나무의 수인 B. bagging이나 random forests와는 다르게 boosting에서는 B가 매우 크다면 과적합의 우려가 있다. CV를 통해서 B를 결정한다.
2. shrinkage parameter인 λ. 이는 boosting이 배우는 속도를 조절한다. 주로 0.01, 0.001을 사용하고 상황에 따라서 달라질 수 있다.
3. 각 나무에서의 분리 수인 d. 이는 boosted ensemble의 복잡도를 조절한다.