Tree-Based Methods

Predictor 공간을 조각조각으로 나누는 방법으로 어떤 관측에 대한 예측을 할 때에는 그 관측이 속하는 조각의 training observation의 평균이나 최빈값를 예측값으로 가져옵니다.

간단한 구조이기 때문에 Ch6,7에 등장했던 다른 모델들에 비해 예측력이 좋지는 않지만 해석력이 좋은 것이 장점입니다.

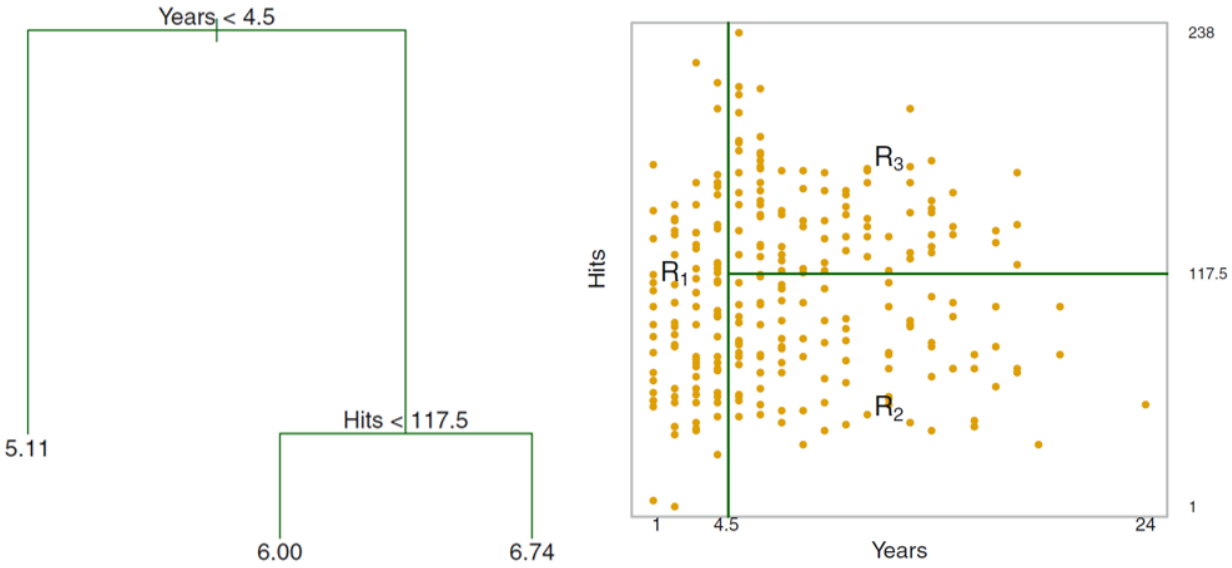
8.1 The basics of Decision Trees

여기에서는 decision tree를 regression tree와 classfication tree로 나누어 각각에 대해 설명합니다.

8.1.1 Regression Tree

Quantitative response를 예측하는 경우 사용됩니다.

쉬운 접근을 위해 *Years (얼마나 오래 메이저 리그에서 뛰었는지)*, *Hits (이전 해에 얼마나 많은 타수를 기록했는지)*를 predictor로 이용해서 *Salary*를 예측하는 경우를 살펴 봅시다. (Salary는 log 변환 되어있음) Regression tree에 이 데이터를 적합시켜 보면 다음과 같은 tree 구조가 만들어집니다.



Years < 4.5인 경우 평균 log salary는 5.11로 나타나는데, 이 클래스에 속할 경우 $e^{5.11}$, 약 $165174로 Salary를 예측할 수 있다는 의미입니다.

이 tree 구조를 predictor space에서 살펴보면, 다음과 같이 R_1, R_2, R_3로 구간이 나누어져 있음을 볼 수 있습니다.

각 구간에 대한 평균 log salary는 각가 5.11, 6.00, 6.74가 됩니다.

R_1, R_2, R_3는 tree의 *terminal node*또는 *leaves*라고 불립니다. 왼쪽 그림에서 tree의 가지가 갈라지는 부분은 *internal node*라고 부릅니다. 또한 각 node들을 이어주는 부분을 *branch*라고 부릅니다.

그렇다면 **Regression tree는 어떻게 해석해야 할까요?**

가장 먼저 Years를 기준으로 split이 일어나므로 Years가 Salary를 결정하는 가장 중요한 요소라는 것을 알 수 있습니다. Years<4.5인 경우 더이상 split이 일어나지 않으므로 경력이 적은 선수의 경우에는 Hit가 Salary에 별로 영향을 주지 않는다고 볼 수 있습니다. 하지만 메이저 리그에서 5년 이상일한 선수의 경우 Hit가 높으면 Salary도 높은 경향을 보입니다. 이처럼 Decision tree에서는 predictor가 response에 미치는 영향을 설명하기 쉽습니다. Predictor space를 조각낸 그림에서 볼 수 있듯 시각적으로 표현하기 쉽다는 것도 장점입니다.

이번에는 **Regression tree를 만드는 방법**에 대해서 이야기해 봅시다. 간단하게 설명하자면, Predictor space를 겹치지 않는 구간으로 나누고, 각 구간에 속한 observation에 대해서는 같은 prediction값을 준다는 것입니다.

예를 들어 R_1, R_2 두 구간으로 predictor 구간을 나누고 R_1에 속한 training observation들의 response의 평균이 10, R_2에서는 20이라고 가정했을 때, 만약 새로운 observation이 R_1에 속하면 response 예측값은 10이되고, R_2에 속한다면 20이 된다는 의미입니다.

그렇다면 predictor를 어떻게 구간으로 나눌 수 있을까요? 이론적으로는 어떤 모양으로도 구간을 나눌 수 있지만 쉬운 해석을 위해 구간은 직사각형 모양(박스)으로만 나눌 수 있다고 가정합시다. J개의 구간으로 나누는 경우 우리의 목적은 RSS를 최소화시키는 $R_1, R_2 \ldots, R_j$라는 박스를 찾는 것입니다. 이렇게 된다면 response를 가장 잘 예측하는 predictor space로 나눌 수 있습니다.

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

위의 식에서 $\hat{y}_{R_j}$는 j번째 박스에 있는 traning observation들의 response값들의 평균을 의미합니다. 각각의 박스들에 대해 RSS를 구하고 이를 최소화시킵니다. 따라서 top-down (= greedy approach = recursive binary splitting)이라는 방법을 사용합니다. 처음에는 하나의 구역인 predictor space를 binary splitting을 반복해가면서 2개, 3개,..으로 계속 쪼개나가는 방법입니다. 궁극적으로 어떤 split이 좋을 지를 생각하지 않고 현재 가장 최선인 split을 찾는 방법이기 때문에 greedy라는 이름이 붙습니다.

Predictor space를 나누는 과정에 대해 좀 더 자세히 설명하자면, 각 splitting을 하기 위해서는 RSS의 감소량이 최대가 되게 하는 predictor X_j의 경계를 찾는 것이 중요합니다. 전체 predictor 각각에 대해 RSS의 감소량이 최대가 되는 경계를 찾아내어 tree를 만들어 내게 됩니다.

s라는 경계점을 기준으로 R1과 R2를 나눈다면

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

아래의 식을 최소화 시키는 j와 s값을 구하게 됩니다.

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

다음에는 이 과정을 반복해 best predictor와 best cutpoint를 찾아 각각의 구역에 대한 RSS값을 최소화 시킵니다. 두 번째의 splitting과정 부터는 전체의 predictor space가 아니라 이미 나누어진 predictor space에 대해서 spliting을 진행하게 됩니다. 역시 이 과정도 RSS값을 최소화되게 합니다. 계속 이 과정을 반복하는데 특정 조건이 되면 멈추도록 설계를 합니다.(각 region의 observation 갯수 5개 이하)

그렇다면, 새로운 데이터가 들어왔을 때 regression tree는 어떻게 결과값을 예측할까요? 우선 학습된 Regression tree를 이용해 predictor값에 따라 새로운 관측이 어떤 terminal node에 속하는 지 알아냅니다. 이후 같은 terminal node에 해당하는 training observation들의 response를 평균내고, 이 평균값을 예측값으로 씁니다.

Tree pruing

Tree가 오버피팅이 되면 test set을 잘 예측하지 못하게 됩니다. 따라서 tree가 너무 복잡해지지 않게 하는 것이 중요합니다. Bias는 약간 손해보더라도 variance를 줄여서 더 나은 해석을 할 수 있기 때문입니다. 가장 쉬은 방법으로는 split threshold를 주는 방법입니다. Tree에서 splitting을 할 때 RSS 감소량이 특정 기준 이상일 때만 splitting을 하는 방법인데, 이 방법은 *RSS 감소량이 낮은 split (bad split)* 이후에 *RSS 감소량이 매우 높은 split(good split)*이 나오는 경우 아예 split 자체를 하지 않는다는 점에서 총 RSS를 감소시키는 데 좋은 방법이 아닙니다.

따라서 우선 large tree T_0을 만들어두고 가지치기를 통해 subtree를 만들어내는 것이 가장 좋은 방법일 것입니다.

이 때 만들어진 subtree들 중 test error가 가장 작은 것을 선택해야 좋은 모델입니다. 그렇다면 어떤 방식으로 가장 좋은 subtree를 선택할까요?

Cost complexity pruning (= weakest link pruning)을 이용합니다. Lasso model과 비슷하게 penalty를 주는 방식인데, 위의 RSS에 subtree의 leave 갯수 $|T|$ 앞에 α를 곱한 항을 추가해서 이 식을 최소화시키도록 tree를 학습하는 방식입니다.

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

subtree의 leave 수가 너무 많으면 (tree가 너무 복잡하면) penalty항이 커지므로 tree가 너무 복잡해지는 것을 방지하게 됩니다. α는 turing parameter라고 부르는데, subtree의 complexity를 조절하는 역할을 합니다. 최적의 α는 cross validation이나 validation set을 이용해서 찾을 수 있습니다.

8.1.2 Classification Trees

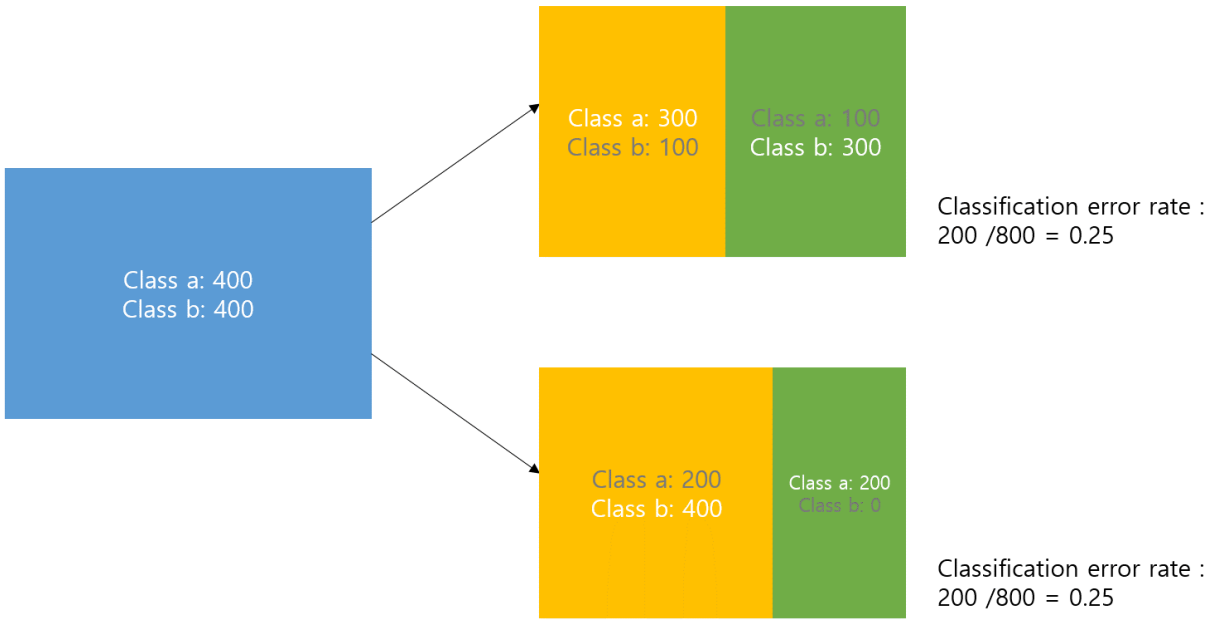
Qualitative response를 예측하는 경우 사용됩니다. Regression tree와 다르게 어떤 관측에 대한 response를 예측할 때 가장 많이 등장하는 라벨을 예측하게 됩니다. Regression tree에서는 RSS를 loss function으로 사용했지만 Classification tree에서는 response가 categorical하므로 RSS를 사용하지 못합니다.

대신에 classification tree에서는 ***classfication error rate***를 사용합니다. classification error rate는 나누어진 구역에서 다수에 속하지 않는 traning observation의 비율을 의미합니다.

classification error rate $E = 1 - \max_k(\hat{p}_{mk})$ **그래서 어떻게 하겠다는거??**

조금 더 자세히 설명을 해 보겠습니다. $\hat{p}_{mk}$는 'm' 구역에서 분류 'k'에 해당하는 데이터의 비율을 의미합니다. 만약 R_1에 'cat' 라벨을 단 observation이 6개, 'dog'라벨을 단 observation이 1개가 있다면 $\hat{p}_{1\,'cat'}$은 $\frac{6}{7}$, $\hat{p}_{1\,'dog'}$는 $\frac{1}{7}$가 됩니다. 여기서 $\max_k(\hat{p}_{mk})$의 의미는 'm' 구역에서 **다수**에 해당하는 분류 'k'의 비율을 의미합니다. 위의 예시에 따르면 $\hat{p}_{1\,'cat'} = \frac{6}{7}$에 해당됩니다.

그렇다면 여러 개의 구역으로 나누어진 경우 classification error rate는 어떻게 구할까요? (ESL 참조)



위의 그림과 같이 분류 'a' 400개,분류 'b' 400개로 이루어진 predictor space를 split한다고 생각해 봅시다. 위의 split으로 나누어진 노랑 초록 두 개의 구역은 각각 class 'a'가 다수, class 'b'가 다수가 됩니다.

여기서 $max(\hat{p}_{mk})$는 $\frac{\text{각 구역마다 다수에 속하는 데이터의 총 합}}{\text{전체 데이터 수}}$을 의미합니다. 각 구역마다 등장하는 빈도수가 최대가 되는 데이터들을 고려해 결국 전체적인 $\hat{p}_{mk}$를 최대로 만듭니다.

하지만 위의 split과 아래의 split을 보면, 위의 split보다 아래의 split이 더 순도가 높게 분류를 했음에도 불구하고 classification error rate는 동일합니다. 계속 split을 진행하면서 tree를 키워나간다면 분명 각 구역의 순도가 높아질텐데, classification error rate는 이것을 고려하지 못하기 때문에 Gini index, entropy를 대신 사용하기도 합니다.

Gini index는 node의 purity를 평가할 수 있는 요소로 Gini index가 크면 purity가 낮다는 것을 의미합니다.

$\hat{p}_{mk}(1-\hat{p}_{mk})$는 $\hat{p}_{mk}$가 0이나 1에 가까울 때 최소가 되고 이는 구역 'm'에서 특정 분류 'k'의 비율이 압도적으로 높다는 의미이므로 purity를 측정하는 지표가 될 수 있습니다.

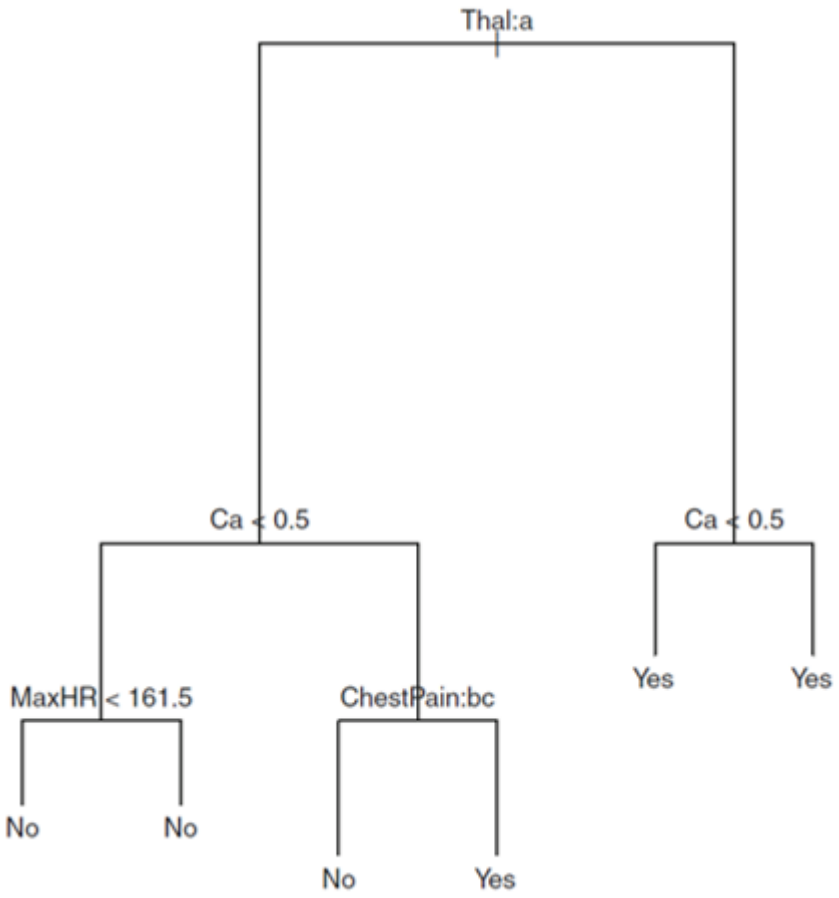
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

다음으로 사용될 수 있는 기준은 entropy입니다. Gini index와 마찬가지로 node purity를 평가할 수 있는 요소로서 purity가 높으면 entropy는 작은 값을 가집니다.

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

Gini index와 entropy 모두 node purity에 민감하기 때문에 classfication tree의 split 과정에서 split이 얼마나 잘 되었는 지를 평가하는 지표로 쓰이게 됩니다.

보충하기



Q. predictor가 qualitative한 경우?

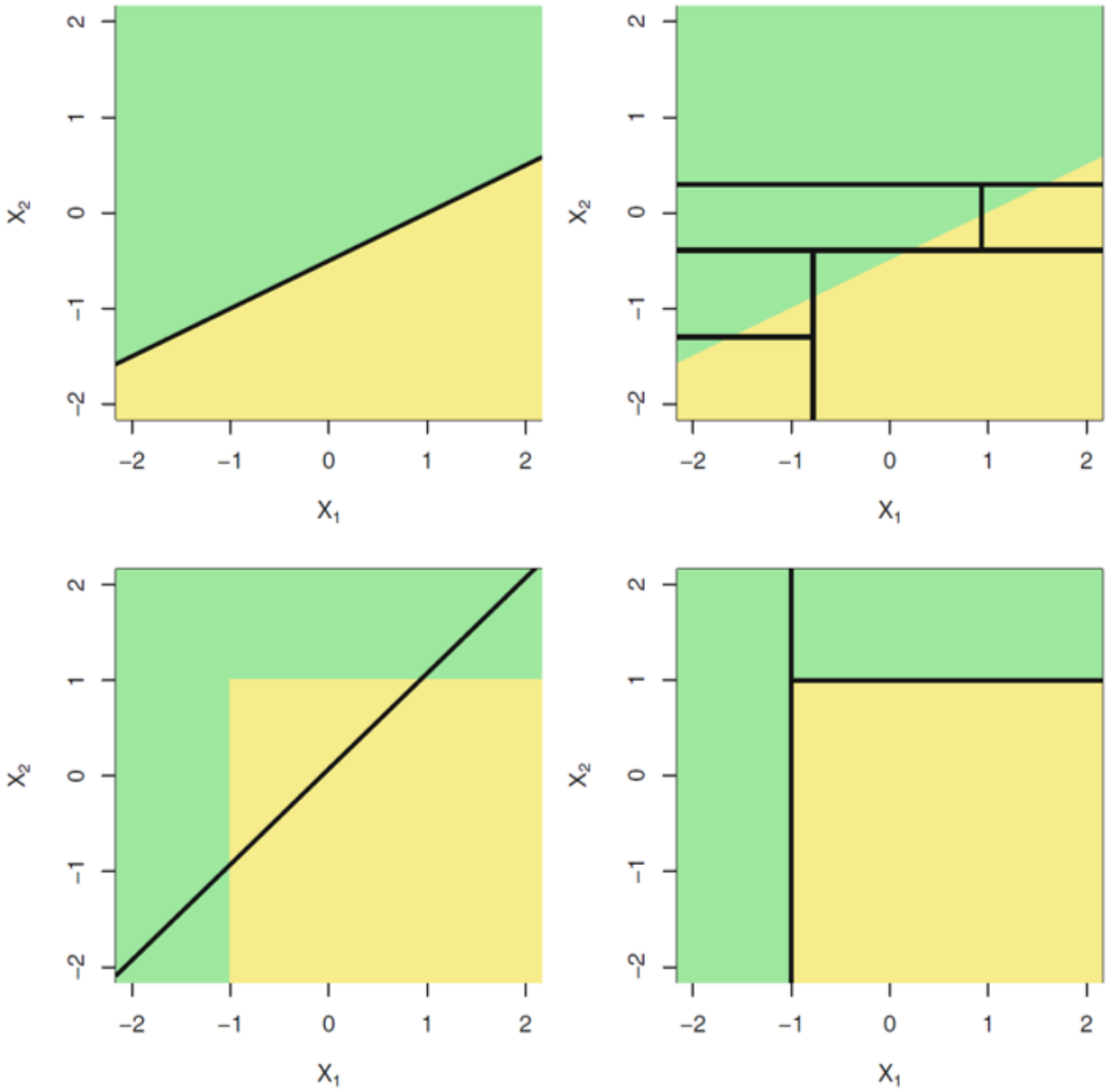
Decision tree의 강점은 predictor가 qualitative한 경우에도 쓸 수 있다는 점입니다. ChestPain predictor를 사용해 Heart disease 발병 여부를 예측하는 경우를 생각해 봅시다 (위 그림). ChestPain의 정도를 a,b,c 세 단계로 나누어 놓은 경우 decision tree에서 split을 하게 되면 response에 미치는 영향에 따라 a/ bc로 나눌 수 있습니다. 이 때 왼쪽 node는 chestPain 정도가 b,c인 경우 , 오른쪽 노드는 chest pain 정도가 a인 경우를 의미합니다.

Q. split으로 나눈 노드가 같은 prediction을 반환한다?

MaxHR에 의해 나누어진 노드들을 비교해 보면 둘다 No라는 예측값을 가집니다. 이럴거면 왜 나누었냐는 생각을 할 수도 있지만 두 노드는 purity면에서 차이가 납니다. 왼쪽 노드의 경우 purity가 낮고 오른쪽 노드는 purity가 높은데, 만약 test set을 넣었을 때 왼쪽 노드에 배정이 된다면 'Heart disease가 없긴 한데, 신뢰도는 떨어진다'라고 말할 수 있고 오른쪽 노드에 배정이 된다면 그 반대로 말할 수 있습니다.

8.1.3 Trees vs Linear Models

Linear model은 일차 함수를 이용해 predictor space를 나누고 tree 모델은 predictor space를 직사각형 모양으로 조각조각 냅니다. 따라서 데이터의 모양에 따라 어떤 모델을 쓸 지 달라집니다.



response에 따른 predictor space의 경계가 직선인 경우 linear model, 정사각형으로 경계가 만들어져있다면 decision model이 더 적합한 모델입니다.

8.1.4 Advantages and Disadvantages of Trees

시각화하기 좋고, 설명하기 쉬우며 dummy variable없이 qualitative predictor를 사용할 수 있습니다.

하지만 prediction accuracy가 떨어지며 data의 변동에 영향을 많이 받습니다.

8.2 Improving Decision Trees

지금까지는 기본적인 tree model에 대해 알아보았다면 지금부터는 tree model을 향상시킬 수 있는 방법에 대해 알아봅니다. 여기에서 소개할 Bagging, Random Forest, Bootstrap은 Ensemble algorithm의 일종으로 여러 training set에 대해 decision tree들을 적합시키고 이들을 종합해서 조금 더 예측력이 좋은 모델을 만드는 것이 목표입니다.

8.2.1 Bagging

Boostrap aggregation의 준말입니다.

> Boostrap이란? (Ch 5.2)
>
> Bootstrap은 실제로는 계산하기 어려운 추정량들의 불확실성(uncertainty. 그것이 어떤 통계량의 분산이던, 평균이던 뭐던)을 계산하는데 널리 쓰이는 통계기법
>
> **가지고 있는 원래의 data set에서 복원추출로 새로운 샘플을 뽑아낸다**

Statistical learning에서 쓰는 bootstrap은 여러 개의 tree를 적합시키기 위한 training set을 추출하는 목적이 있습니다.

간단히 말하면 boostrap으로 하나의 population에서 여러 개의 training set을 추출한 다음, 이들 각각에 대해 tree를 적합시킨 후 결과들의 평균을 내어 더 나은 모델을 만드는 방식입니다.

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

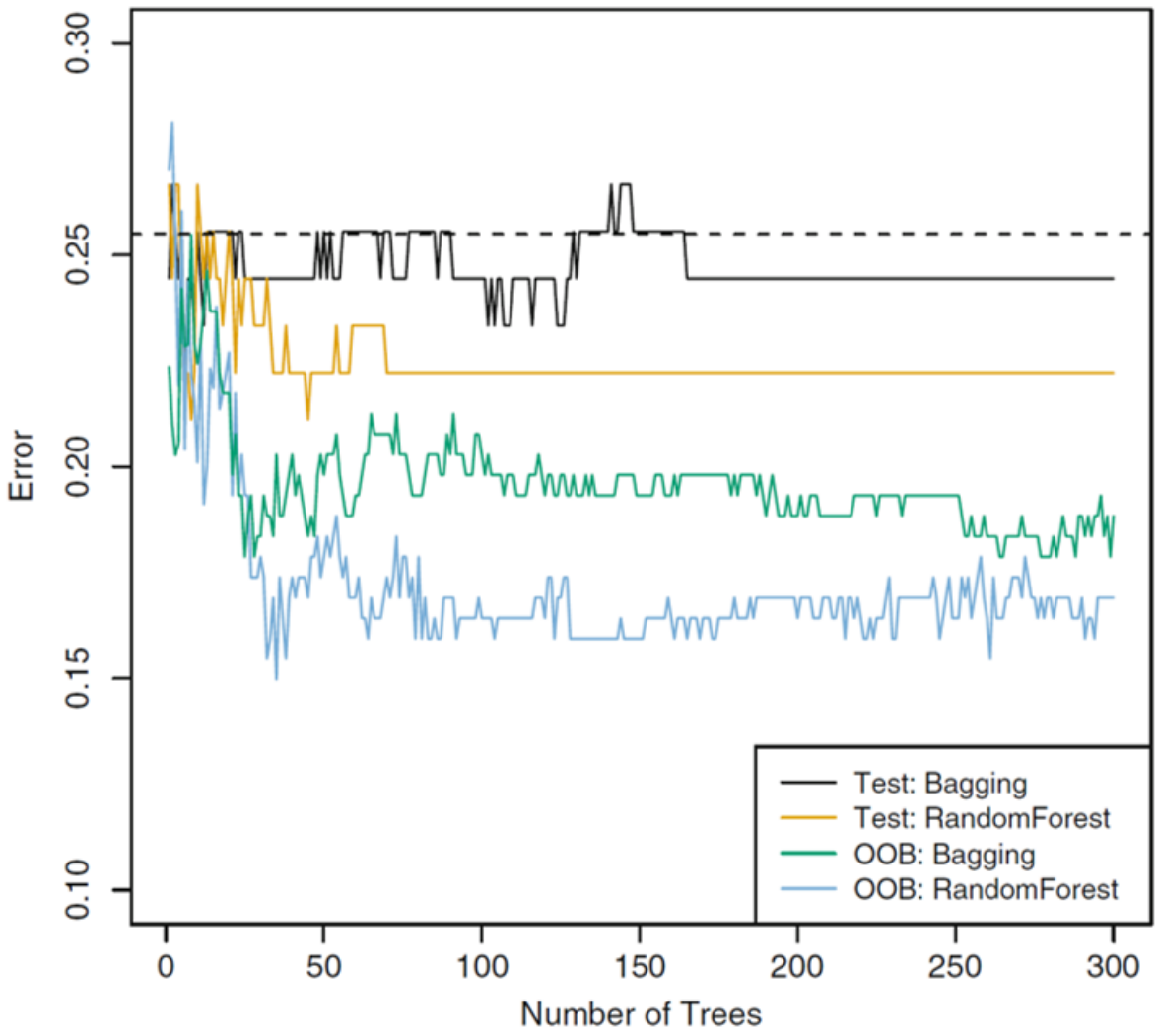
여기서 $\hat{f}$는 $\sum_{b=1}^{M} c_m * 1_{(x \in R_m)}$으로 각 decision tree를 나타내는 function입니다. 여기서 c_m은 region m의 training observation들의 평균값을 의미합니다.

Tree가 깊어지면 variance가 낮아지고 bias가 높아져 test error rate가 높아지는데, Bagging에서는 따로 tree pruning을 하지 않습니다. 하지만 B개의 deep tree를 평균을 내면 variance가 줄어들기 때문에 accuracy가 증가합니다.

response(Y)가 qualitative할 때도 쓸 수 있다는 게 신기한 점인데, 이 때는 가장 많이 등장한 클래스를 택해서 분류합니다.

Test error를 계산할 때는 cross validation이나 validation set approach를 쓸 수 있는데, bagging에서는 boostrap 때 포함되지 않았던 변수들 (*out-of-bag observation = OOB*)을 가지고 각 모델의 test error를 추정할 수 있습니다.

각 tree마다 생기는 OOB를 각 tree에 대한 test set으로 활용해 OOB MSE (regression) 또는 classfication error (classification probelm)을 구하면 이는 bagged model의 test error로 사용할 수 있습니다.



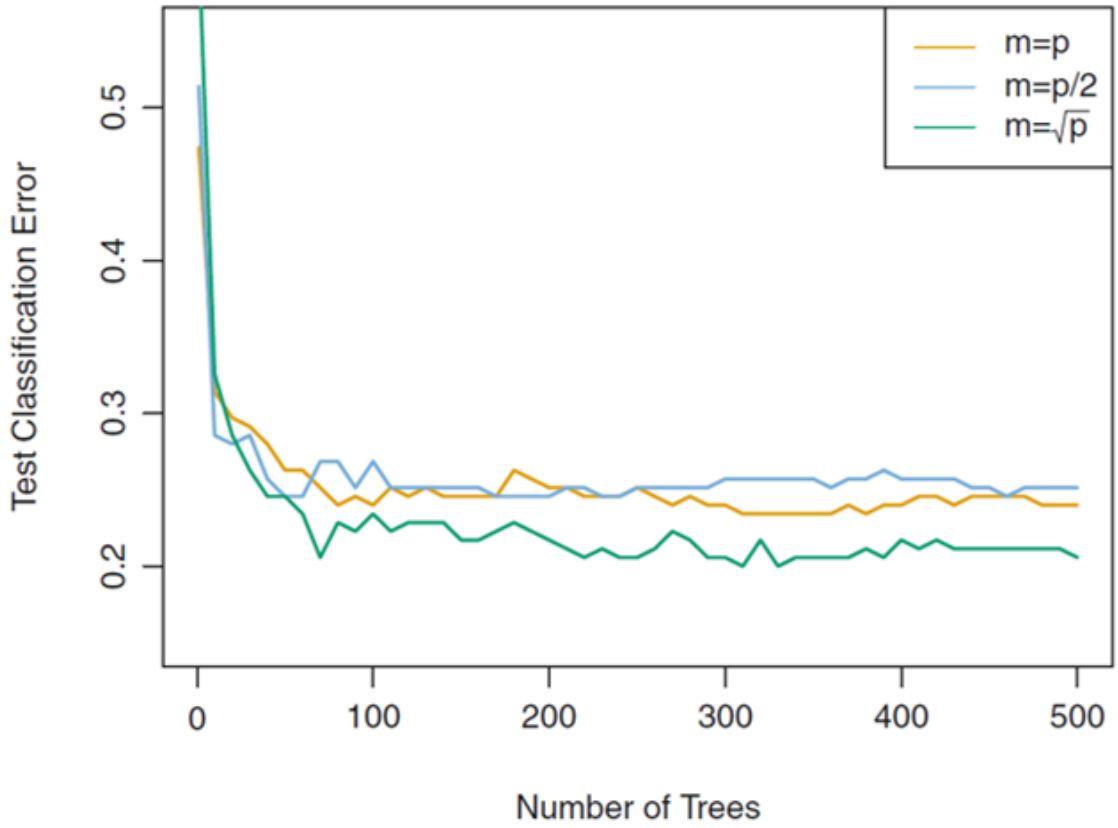
Bagging의 문제점은 accuracy는 높여주지만 해석력은 떨어뜨린다는 점입니다. 여러 개의 tree들을 모두 모아 평균을 내었기 때문에 각 tree에서 각각의 predictor가 response에 미치는 영향을 명확하게 알 수 없습니다.

하지만 RSS (bagging regression tree)와 Gini index (bagging classfication tree)를 이용해서 predictor가 전체 prediction에 미치는 영향은 알 수 있습니다. 각 predictor에 대해 split으로 인해 총 RSS의 감소량 평균, 총 Gini index 감소량 평균을 계산하는 것이 그 방법입니다.

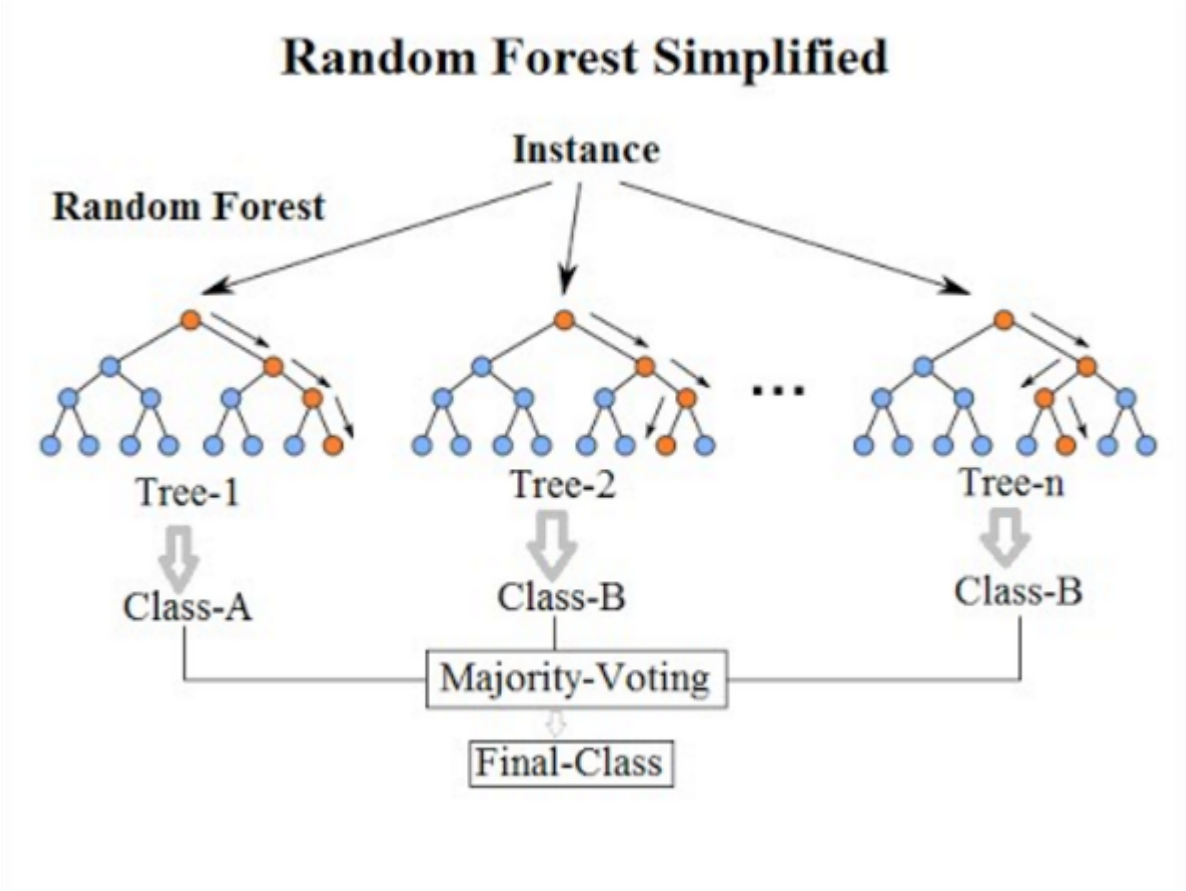
8.2.2 Random Forests

Bagging에서와 마찬가지로 Random Forest도 bootstrap으로 추출해낸 여러 개의 training sample을 가지고 decision tree들을 만들어냅니다. 하지만 각각의 decision tree들을 만들 때 총 p개의 predictor 중에서 무작위로 선택된 일부 m개의 predictor만 split에 이용됩니다. 이 때 m을 얼마로 설정할 것인가의 문제가 남는데 보통 $m \approx \sqrt{p}$로 설정합니다. 아래의 그림을 보면 $m = \sqrt{p}$로 설정한 경우 test error가 가장 낮게 나오는 것을 볼 수 있습니다.

(왜인지는 잘...ㅎㅎ)

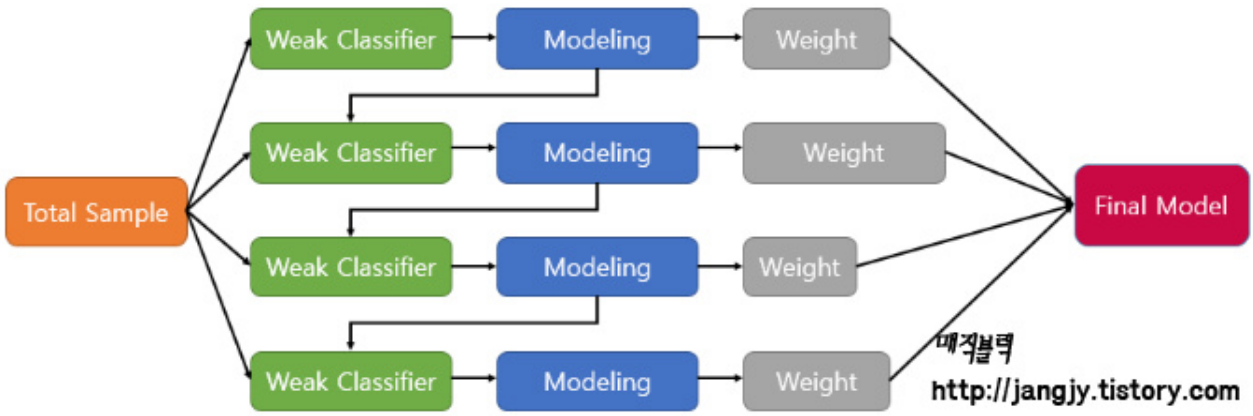


랜덤으로 predictor 일부만을 선택해서 모델을 적합시키는 것의 장점은 약한 영향력을 가지는 predictor도 고려할 수 있다는 점입니다. 만약 하나의 strong predictor가 있는 경우 bagging을 하게 되면 strong predictor 때문에 모든 tree가 비슷한 모양을 가지게 됩니다. 이렇게 하면 ensemble의 의미가 없어지는데, random forest에서는 일부를 random으로 선택하기 때문에 가장 강한 predictor보다는 약하지만 충분히 강한 predictor의 영향도 고려할 수 있게 됩니다.



8.2.3 Boosting

Boosting에서는 Bagging이나 Random Forest와는 다르게 tree를 점진적으로 키워 나갑니다. 이전에 만들어진 tree로부터의 정보를 사용하여 이전 트리에서 나쁜 예측을 한 데이터에 대해서는 패널티를 주고 다음 tree에서 학습해서 모델을 '개선'시켜 나갑니다.



Boosting도 여러 개의 training sample을 사용하는데, boosting에 사용되는 training sample은 bootstrap이 아니라 cross validation을 통해 얻어집니다. 따라서 training sample들은 중복되는 부분이 많습니다.

그리고 response가 아니라 residual에 대해 model을 학습한다는 것도 baggin ㅎ과의 차이점입니다. λ라는 shrinking parameter (= penalty parameter = tuning parameter)를 이용해 $\hat{f}$와 residual을 업데이트 하면서 천천히 점진적으로 모델을 학습시킵니다. 최종적으로는 residual을 0에 가깝게 만드는 방향으로 모델이 학습됩니다.

이 때 d는 boosted ensemble의 복잡성을 결정하는데 이는 tree가 split d개가 되도록 학습시킨다는 것을 의미합니다.

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,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

알고리즘을 좀 더 자세히 살펴봅시다.

우선 처음에는 모델을 $\hat{f}(x) = 0$으로 설정하고 residual = 데이터가 가지고 있는 y값으로 설정합니다.

그 다음 첫 번째 training sample에 대해 $\hat{f}^1(x)$ tree를 적합시켜 residual 예측값을 알아냅니다 (8.10). 최초의 residual은 y값이므로 이 경우에 residual 예측값은 y값이 됩니다.

그 다음 $\hat{f}(x)$를 $\hat{f}(x) + \lambda \hat{f}^b(x)$로 업데이트 합니다. 여기에서 λ는 일종의 learning rate같은 느낌입니다.

그 다음에는 residual r_i를 $r_i - \lambda \hat{f}^b(x_i)$로 업데이트 합니다. 여기에서도 역시 λ를 활용해서 업데이트 되는 정도를 조절할 수 있습니다. 또한 업데이트하는 residual은 위 그림에서의 weight 역할을 하게 됩니다. 예를 들어 training sample 1의 '데이터 1'의 y값이 1이라고 가정하고 $\lambda = 0.5$라고 가정합시다. 만약 $\hat{f}^1(x)$이 예측을 잘 해서 1을 냈다면 '데이터 1'에 대한 residual은 $0.5 = 1 - 0.5 * 1$로 줄어들게 됩니다. 이번 tree에서 '데이터1'에 대해 예측을 잘해서 residual이 충분히 줄어들었다면 다음 번 tree에서 '데이터1'을 학습 시킬 때의 부담은 줄어듭니다. 대신 residual이 큰 다른 데이터에 집중해서 tree를 학습시키겠죠.

이런 식으로 다른 training sample에 대해서도 차례차례 tree를 적합시키면서 이전 tree에서 예측이 잘 되지 않았던 부분을 집중적으로 고려해서 tree를 개선시켜 나갑니다.