Tree Model

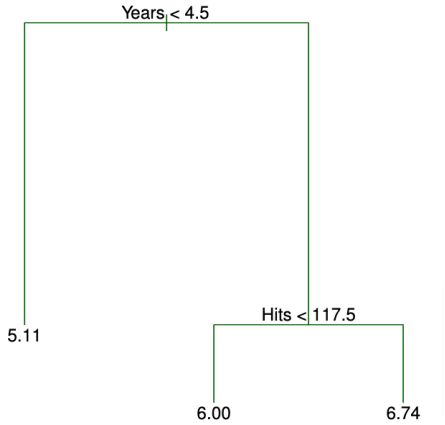
Decision Tree

Pros and Cons

- Pros: Simple and easy to interpret
  - Decision Tree는 각각의 node를 기준으로, feature와 해당 feature에 대한 cutoff value에 따라서 feature space(독립변수 X의 집합)를 분할하는 방식으로 regression & classification을 수행한다.
  - 따라서 어떤 변수와 값을 기준으로 분할되었는지에 따라서 개별 feature가 전체 model training에 미친 영향 등 해석이 보다 쉽다.
- Cons: Bad performance
  - 단일 decision tree는 다른 linear model에 비해서 그 성능이 좋다고 볼 수 없으며, 특히 가지가 많아질 경우 variance가 매우 높아질 가능성에 취약하다.
  - 따라서 Bagging, Random Forest, Boosting 등 여러 개의 tree를 모아서 사용하는 방식이 더 자주 사용된다.

Regression Tree

- Internal node에는 split criterion(feature & cutoff value)가 assign되며,
- Leaf node에는 각각의 split criterion에 따른 observation들의 target value가 assign된다.
  - 이때 split criterion을 만족하는 경우 left subtree, 만족하지 않는 경우 right subtree로 이동한다.



  - 위 그림의 경우 career time이 4.5년 미만인 선수들은 `Years < 4.5`의 left subtree로, 4.5년 이상인 선수들은 right subtree로 이동
    - 다시 right subtree의 선수들 가운데 시즌 안타가 117.5개 미만인 선수들은 `Hits < 117.5`의 left subtree, 이상인 선수들은 right subtree로 이동
    - 이러한 작업을 recursive하게 반복
  - 위의 tree가 regression의 최종 tree라고 한다면, leaf node에는 각각의 path를 타고 내려가는 기준을 만족한 observation(선수)의 target value(log salary)가 assign된다.

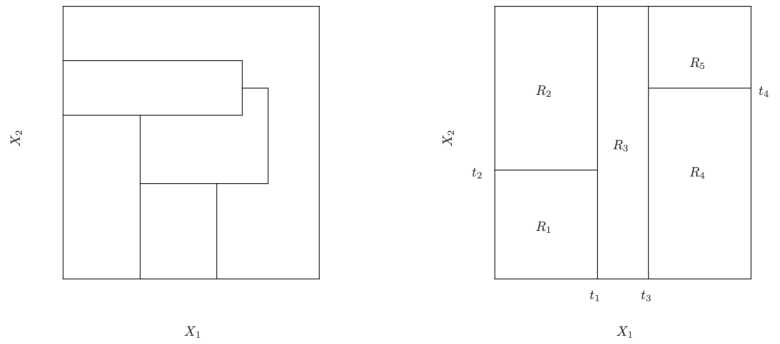
    - 즉, 위의 경우 4.5년 미만 play한 선수는 평균 5.11의 log salary(→ $1,650,000)를 받게 되며,
    - 4.5년 이상 play한 선수들 가운데 직전 시즌 안타가 117.5개 미만인 선수들은 평균 6의 log salary(→ $4,034,287), 117.5개 이상인 선수들은 평균 6.74(→ $8,455,607)의 log salary를 받는다.

Interpretation

- Split이 먼저 발생한 feature일수록 중요한 feature이다.
    - MSE의 감소분을 기준으로 feature & cutoff value를 정하는 greedy algorithm을 사용하기 때문
- 특정한 subtree에서 split이 발생하지 않았다는 의미는 **곧 Error를 특정 threshold 이상으로 줄여주는 significant variable&cutoff value를 찾지 못했다**는 의미와 같다.
    - 여기서의 Error는 Training error를 의미한다.

Tree-building Process

- 데이터에 존재하는 feature를 $X_1, \dots, X_p$라고 하자. 이때, decision tree는 개별 predictor를 사용하여 전체 feature space를 J개의 non-overlapping region으로 partition한다.
    - 이때 $J \neq p$, 즉 전체 predictor를 사용하지 않을 수도 있으며,
    - 특정 cutoff value를 기준으로 feature space(또는 subspace)를 완전히 둘로 나누어야 한다.
        - In other words, feature space가 high-dimensional rectangle 또는 box로 나누어져야 한다.



        - 즉 Decision tree의 결과로는 오른쪽과 같은 partition만 가능하다.
    - **<u>결국 각각의 region은 각각의 leaf node에 해당한다.</u>**
- 그리고 개별 영역 $R_1, \dots, R_J$는 각 영역 R_j(leaf node)에 속한 **<u>target value들의 평균 $\hat{y}_{R_j}$과 true value y_i 사이의 차이 RSS를 최소화</u>**하는 영역으로써 정해지는 것이 최종 goal이다.

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

    - RSS의 구체적인 의미는 각각의 region $R_1, \dots, R_j, \dots, R_J$를 돌면서, 개별 region에 속한 target value들의 평균$\left(\hat{y}_{R_j} \right)$과 true value들의 차이의 squared sum을 구하고, 다시 해당 squared sum을 모든 leaf node에 대해서 더한다는 의미
    - But 이것을 모든 feature space 상에 주어진 모든 (feature-cutoff value)에 대해서 계산하는 것은 computational burden이 매우 크므로, 실제로는 우회 방법을 사용한다.

- **현 단계에서 split에 사용할 feature**와 split의 **실제 기준이 되는 cutoff value를 결정**하는 방법을 사용하며
  - 해당 cutoff value는 split에 사용할 feature로부터 선택된다.
  - 그리고 (feature-cutoff value)는 **"현재 주어진 Tree로부터 best improvement를 보이는 조합"**으로 결정되며, 이러한 algorithm을 **greedy algorithm**이라고 한다.
    - 전체 데이터를 보고 split scheme을 미리 짜는 것이 아니라, 현재 상황에서 best decision을 내림
    - ex. 현재 주어진 tree T_t에 대해서 T_{t+1}로 grow하는 데 사용할 feature candidate을 X_j, X_j에서 선택될 cutoff candidate을 s라고 한다면, (X_j, s)에 의해 분할되는 영역을 다음과 같이 정의할 수 있다.

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

    - 그리고 해당 region R_1, R_2에 **대해서만,** 분할 결과에 따른 RSS를 측정한다.

$$L_j = \sum_{x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2} (y_i - \hat{y}_{R_2})^2$$

    - 즉, 현재 주어진 T_t 상에서 L_j 값이 가장 작아지는 (X_j, s)의 조합을 사용하여 분할하며, 그렇게 분할된 새로운 tree를 T_{t+1}이라 한다.
    - 다만 이 작업은 모든 feature space가 아니라 T_t상에서 이미 나눠진 영역들 가운데 하나에서 이루어진다.
  - 따라서, **어떤 region에서 split할지**의 여부 역시 결정사항이다.

Pruning

- Tree regression model은 leaf node에 속한 data point에 대해서 true value와 leaf의 평균을 최소화하는 방향으로 split(=training)된다.
  - 따라서 극단적인 경우 each leaf node에 each data point가 하나씩 assign될 경우, training error를 0으로 만들 수 있다.
  - 그러나 이는 overfitting이며, 그에 따라서 poor test performance와 함께 high variance를 보이는 model이 된다.
- 따라서 적절한 기준을 두어 split을 중단하는 작업이 필요하다.
  - 대표적인 예시로 RSS 감소분에 대한 threshold를 설정한 뒤, 해당 threshold를 넘지 못하는 단계에 도달하면 training을 중단하는 방법이 있다.
    - But 이러한 작업은 **too short-sighted**이라는 단점이 있는데, 현재 단계에서는 threshold를 넘지 못하더라도 다음 단계에서는 넘을 수도 있기 때문이다.
  - 따라서 다른 방법으로 해당 threshold를 넘을 때까지 기다려주는 split 횟수(e.g. early stopping rounds)를 설정한 뒤, 해당 횟수 이내에서도 threshold를 넘지 못할 경우 training을 중단하는 방법을 사용하기도 한다.
- 또는 split을 중단하는 대신 **매우 큰 tree T_0를 생성한 다음 leaf를 없애는 방법**을 사용하기도 하는데, 이를 pruning이라고 한다.
  - **Cost complex pruning** or **Weakest link pruning**이라고도 부른다.

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

- 전체 **Tree에 대한 leaf node(=region)의 개수를** $|T|$라고 할 때,
  - 기존의 RSS 식에 $\alpha|T|$라는 constraint를 더해준 다음, 다시 개별 region에서의 RSS를 최소화하는 방향으로 optimal $|T|$를 찾는다.
    - 단, 여기서의 T는 $T \subset T_0$의 관계가 있는 initial tree T_0의 subtree이다.
  - 여기서 $\alpha \in (0, 1)$이며, 1에 가까울수록 tree size에 강한 penalty를 주어 size를 최대한 작게 유지하도록 하며, 0에 가까울수록 tree size를 크게 건드리지 않는다.
- 또한 모든 가능한 subtree를 고려하지 않고 α에 의해 indexing된 subtree를 고려한다.
  - In other words, 먼저 split에 사용된 feature일수록 RSS를 감소시키는 데 가장 기여도가 높은(=중요한) feature이므로, 항상 root로부터 시작하는 subtree를 고려한다.
- 가장 적절한 $|T|$는 Cross-validation을 통해 구해진다.
- 이러한 방법은 마치 Ridge 또는 LASSO의 background logic과 유사하며, 따라서 α 역시 λ와 마찬가지로 cross-validation 등을 통해 최적의 값을 찾을 수 있다.

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. For each $k = 1, ..., K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α

Regression Tree Algorithm

Classification Tree

- Regression tree와 마찬가지로 internal node는 split criterion을 나타내며, leaf node에는 각각의 path를 만족시키는 observation의 target value(discrete)들이 분포되어 있다.
  - 개별 leaf node = Feature space를 partition하는 non-overlapping region
  - 다만 Regression tree와의 차이는 prediction 방식
    - Regression tree는 개별 region에 들어있는 target value들의 평균으로서 predicted value를 계산했다면, Classification tree는 개별 region에 들어있는 discrete target value들 가운데 가장 많은 frequency를 보이는 class로 predicted class를 계산

Performance Metric

- Classifcation에서는 RSS를 사용할 수 없다. 따라서 이에 대한 대안으로서 여러 metric이 사용된다.
- **Classification error rate**

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

- 여기서 $\hat{p}_{mk}$는 m번째 region R_m에 대해서 k번째 class가 나온 비율을 의미한다.
- 즉, $\max_k \hat{p}_{mk}$란 가장 많이 나온 class(즉 R_m의 predicted class)의 비율을 의미하며,
  - 따라서 E는 1에서 predicted class의 비율을 뺀 값, **즉 R_m에 predicted class를 제외한 나머지 class의 비율**이 얼마나 되는지를 나타내는 metric이다.
  - 이 값이 낮을수록 좋다는 의미는, **각각의 leaf node에서 predicted class의 비율이 높으면 높을수록 classifcation이 잘 되었다는 의미**이다.

- **Gini Index**

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

- 특정 leaf node R_m에 대해서, 전체 class $\mathcal{C} = \{1, 2, \dots, K\}$에 걸쳐 **($k$번째 class의 비율) $\times$ (나머지 class의 비율)을 모두 더한 값**
- $K = 2$인 binary classification을 생각해보면, Gini index는 아래와 같이 전개된다.

$$\begin{aligned} G &= \hat{p}_{mk=0} \left(1 - \hat{p}_{mk=0}\right) + \hat{p}_{mk=1} \left(1 - \hat{p}_{mk=1}\right) \\ &= 2\hat{p}_{m0}\hat{p}_{m1} \end{aligned}$$

  - 이 값은 $\hat{p}_{m0} = \hat{p}_{m1}$일 때 최대가 되는데, 이는 새로운 test data가 들어갔을 때 0으로 분류될 확률과 1로 분류될 확률이 모두 50%로 동일하다는 의미이다.
    - 따라서 classifier의 prediction 결과가 사실상 guess와 차이가 없다는 의미로, worst classifier라는 것을 의미한다.
  - This implies that the **Gini index is a lower-the-better metric**.
- **Gini Index가 낮을수록 특정한 class k^*에 해당하는 observation들이 하나의 영역에 집중적으로 분류되었다**는 의미이다.
  - In this sense, Gini index는 **Gini impurity**라고도 불린다.

- **Cross-entropy**
  - Gini impurity의 변형으로 다음과 같이 나타내어진다.

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

  - 음수 부호를 붙여주는 이유는 $\hat{p}_{mk} \leq 1$이므로 $\log$ 변환을 취해주면 음수가 되기 때문에, 다시 D값을 양수로 바꿔주기 위함이다.
  - 마찬가지로 binary classification을 생각해보면 아래와 같이 전개된다.

$$\begin{aligned} D &= -\left(\hat{p}_{m0} \log \hat{p}_{m0} + \hat{p}_{m1} \log \hat{p}_{m1}\right) \\ &= -\left[\hat{p}_{m0} \log \hat{p}_{m0} + \left(1 - \hat{p}_{m0}\right) \log \left(1 - \hat{p}_{m0}\right)\right] \\ &= -\left[\log \left(\hat{p}_{m0}\right)^{\hat{p}_{m0}} \left(1 - \hat{p}_{m0}\right)^{1 - \hat{p}_{m0}}\right] \end{aligned}$$

- Gini index와 마찬가지로 $\hat{p}_{m0} = 0.5$일 때 최댓값을 갖게 되며, 따라서 **cross-entropy 역시 lower-the-better metric**이다.

Comparison with other models

- Linearly separable한 데이터에 대해서도 Tree model은 pretty good performance를 보이지만,
  - 반대로 Tree model로 separation이 잘 되는 데이터는 Linear model로 fitting할 경우 성능이 매우 좋지 않다.
    - 즉, tree model의 범용성이 linear model보다 더욱 넓다는 의미

> Advantages and Disadvantages of Trees
>
> \+ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
> \+ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
> \+ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
> \+ Trees can easily handle qualitative predictors without the need to create dummy variables.
> − Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
> − Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Bagging

- Bootstrap Aggregation
  - Bootstrap을 통해서 **전체적인 variance를 줄일 수 있는 점을 활용**해 overfitting에 대해 vulnerable한 decision tree의 약점을 극복하는 방법

How it works

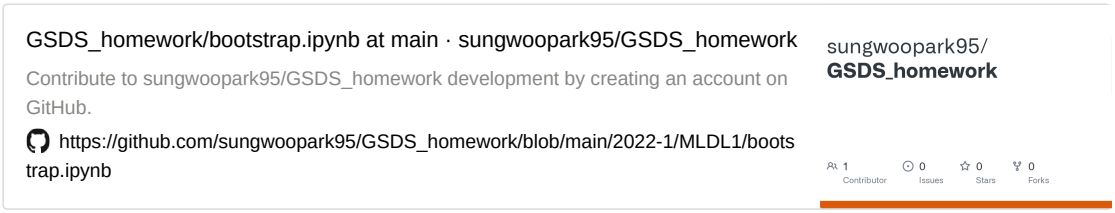
1. 현재 주어진 $n \times (p+1)$ 데이터에 대해서 B번의 sampling-with-replacement를 진행해 총 B개의 $n \times (p+1)$의 bootstrap sample을 만든다.
2. B개의 bootstrap sample에 대해 각각 1개의 model을 independently fitting한다. 따라서 총 B개의 model이 만들어진다.
3. Training 후 unseen data x^*를 개별 bootstrap model에 pass했을 때의 결과값을 $\hat{f}^{*b}(x^*)$라고 하자. $(b = 1, 2, \ldots, B)$
4. Then, overall bootstrap prediction은 개별 model output들의 평균값이다(in regression)

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

- 한편, classification의 bootstrap prediction은 개별 model로부터 예측된 총 B개의 class에 가운데 가장 많이 나온 class로 분류한다.

Out-of-Bag Error Estimation

- Bootstrap은 기본적으로 sampling with replacement을 통해 n개의 observation을 재추출하므로, 중복되는 sample들이 반드시 발생한다.
  - 그렇기 때문에 전체 n개의 original observation 중에서 n개를 재추출함에도 불구하고 bootstrap sample에 포함되지 않는 original observation이 존재한다.
  - 평균적으로 약 $e^{-1} \approx 0.3679$만큼의 original observation이 포함되지 않는다.



- 이렇듯 **개별 bootstrap sample B_b $(b = 1, 2, \dots, B)$마다 포함되지 않는 original observation을 out-of-bag observation**이라 부른다.
  - 평균적으로 약 36%의 original observation이 bootstrap에 포함되지 않는다는 것은, 다시 말하면 1개의 original observation x_i가 bootstrap sample에 포함되지 않을 확률이 약 0.36이라는 의미이다.
  - 따라서 총 B개의 bootstrap sample 가운데 x_i는 평균적으로 약 $0.36 \times B$개의 bootstrap model의 OOB에 속할 것으로 기대할 수 있다.
- 즉, 개별 observation x_i에 대한 prediction은, **해당 x_i가 OOB에 속해있는 bootstrap sample 약 $0.36B$개를 fitting한 model에서 계산된 predicted value**를 활용해 계산할 수 있다.
  - Regression인 경우 평균, Classification인 경우 majority vote
- 이 방법을 반복하면 전체 observation $x_1, \dots, x_n$에 대한 predicted value를 구할 수 있으며, 이를 통해 얻어진 predicted value와 true value 사이의 차이를 통해 error를 추정할 수 있다.
  - 이렇게 추정된 error를 OOB error라고 부른다.

Random Forest

- Bagging + Random selection of predictors
  - 기본적으로 bagging의 방법을 그대로 사용하되, bootstrap sample B개에 fitting되는 총 B개의 model에 대해서 개별 모델 간의 independence를 높이기 위해 사용되는 방법
  - Bagging과 달리 **base model이 반드시 tree model임**
    - cf) bagging에서 사용되는 B개의 individual model은 꼭 tree일 필요는 없음
- **How to decorrelate each tree?**
  - 전체 $n \times (p+1)$의 데이터에서 bootstrap sample을 만들었으므로 개별 bootstrap sample의 feature space는 p-dimensional이다.
  - Random forest는 이때 개별 bootstrap sample B_b $(b = 1, \dots, B)$에 대해서 m $(m < p)$개의 feature를 random하게 추출한 뒤, 그렇게 추출한 m개의 feature만을 실제 분할에 사용한다.

        - 그렇게 설정한 m개의 feature를 모두 분할에 사용했다면, 그대로 tree 성장을 멈춘다.
        - **“좀 덜 자세한 individual tree를 만들어 variance를 줄이되, 그런 individual tree를 여러 개 random하게 만들어서 높아진 bias를 address하자”**
    - Typically 선택하는 m개는 대략 $\sqrt{p}$로 정함 → 그렇게 했을 때 Test error가 가장 좋기 때문
    - 한편, random하게 추출된 m이 작으면 작을수록 개별 tree에 따라서 분할하는 feature가 서로 겹치지 않을 확률이 높아지며, 그에 따라서 개별 tree 사이의 independence가 높아진다.
- **한편, bagging과 random forest는 bootstrap sample의 숫자, 즉 model의 개수가 늘어남에 따라서 overfitting의 위험이 낮아지게 된다.**
- 또한, bagging과 random forest에서 base model로 tree를 사용하는 경우, **개별 tree에 대해서는 pruning을 수행하지 않는다.**

Boosting

- Boosting의 특징
    - Bagging처럼 decision tree 이외의 여러 base model에 대해서도 적용할 수 있는 알고리즘이다.
        - Random forest는 반드시 base model이 decision tree이다.
    - 또한 bagging, random forest처럼 B**개의 서로 다른 base model**을 사용한다.
        - 즉 B개의 서로 다른 model을 각각 1번씩, 총 B번 fitting이 이루어지는 셈이다.
        - 다만 **base model의 구조는 서로 같아야 한다.** 예를 들어, base model로 tree를 사용하는 경우 split 횟수(=leaf node의 개수-1)는 모든 B개의 tree에 대해 동일하게 적용되어야 한다.
    - 다만 bagging, random forest와 달리 생성되는 **여러 base model이 서로 independent하지 않다**.
        - Independent bootstrap sample이 아닌 **original data**에 대해서 **sequentially training**하기 때문이다.

How it works

- Input: Original training data $\{x_i, y_i\}_{i=1}^n$ $(x_i = (x_{i1}, \dots, x_{ip}))$와 B개의 base model

1. 먼저, 개별 observation에 대한 training 결과를 $\hat{f}(x_i)$이라 하고, 이 값을 0으로 초기화한다. 또한, x_i에 대한 residual을 r_i라 하고, 이 값을 true value y_i로 초기화한다.
2. First base model $\hat{f}_1$에 대해서 전체 training data $\{x_i, r_i\}_{i=1}^n$를 fitting한다.
    a. 이때의 split 횟수를 d라 하면, 나머지 $B_2, \dots, B_B$ model들 모두 d번만 split해야 한다.
3. Training된 결과를 $\hat{f}_1(x)$ $(x := \{x_i\}_{i=1}^n)$이라 하고, 아래와 같이 전체 training 결과 $\hat{f}(x)$와 residual r을 다음과 같이 update한다.

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_1(x)$$
$$r \leftarrow r - \lambda \hat{f}_1(x)$$

4. 2 ~ 3을 $b = 2, \dots, B$에 대해 반복한다.

- 결론적으로 도출되는 모형은 다음과 같이 나타내어질 수 있다.

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

Background logic

- **<u>"Residual fitting"을 통한 "slow learning"</u>**
- 단일 decision tree(를 비롯한 다른 base model)를 사용하는 것에 비해서 overfitting의 위험이 낮다.
  - Base model은 shallow tree(typcially $d = 1$)를 사용하기 때문에 variance가 낮으며,
  - 다만 **<u>이전 단계에서 fitting된 model의 정보를 사용하면서 여러 번 fitting하는 반복 학습</u>**을 통해 bias를 낮추도록 설계되었다고 할 수 있다.
- 한편, shrinkage parameter λ는 학습 속도를 더욱 느리게 만드는 역할을 수행한다.
- Residual을 반복적으로 학습하기 때문에 자연스럽게 **이전 단계에서 residual의 감소가 크지 않았던 observation의 target r_i가 current tree에서 커지게 되고, 그에 따라 해당 observation에 focusing**을 할 수 있다.
  - 따라서 결론적으로 전체 observation의 residual을 최대한 줄일 수 있게 된다.

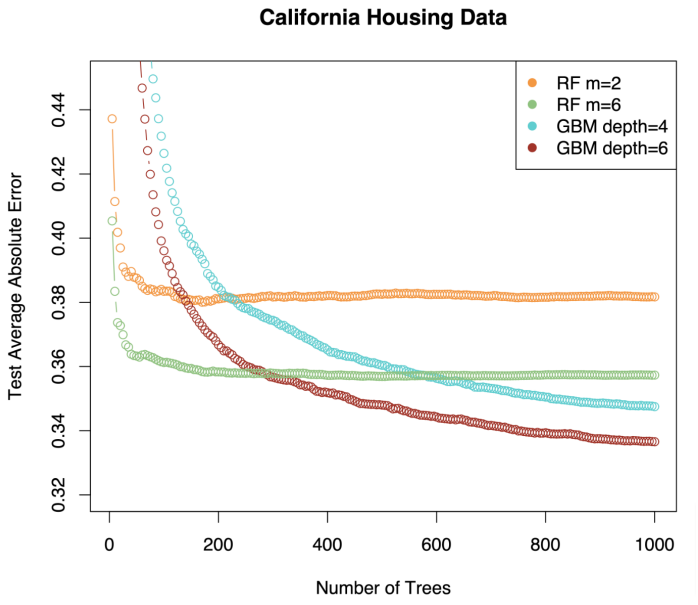
Classification Boosting

- Building a classification boosting tree, briefly.
  1. Initial prediction
     - Individual observation에 대한 initial prediction을 나타내는 tree를 만든다.
     - 이때 individual leaf node에 포함된 값은 log-odds 값이다.
  2. Calculate residuals
     - Residual은 True label $\in \{1, 2, \ldots, K\}$에서 estimated probability를 뺀 값으로 정의한다.
     - Ex. $K = 4$인 경우,
       - 특정 $y_i = 3$이라면 true label vector는 $(0, 0, 1, 0)$으로 정의된다.
       - 이때 estimated probability가 $(0.2, 0.05, 0.65, 0.1)$로 추정되었다면, residual은 (1-0.65)=0.35로 계산되는 것이다.
  3. Predict residuals
     - Probability나 log odds 대신 residual을 predict하는 decision tree(base model)를 구축한다.

Hyper-parameters for Boosting

Symbol	Name	Description	비고
B	The number of base models (or decision trees)	Bagging 및 Random forest에서는 independent model의 개수, 즉 전체 bootstrap sample의 개수였다면, Boosting의 경우 sequentially 학습할 base model의 개수를 의미함	B가 너무 큰 경우 overfitting의 위험이 있음. 따라서 cross-validation을 통해 적절한 개수를 찾아야 함
λ	Shrinkage parameter (or Learning rate)	개별 base model의 output을 반영하는 정도를 의미하며, 작을수록 학습에 필요한 B의 개수는 늘어나게 됨.	0.01 ~ 0.001로 정하는 경향이 있으며, 너무 작을 경우 역시 overfitting의 위험이 있음

Symbol	Name	Description	비고
d	Depth of a base model	base model의 성장을 어느 정도 허용할지를 나타냄. 허용해주는 범위가 넓을수록 base model이 복잡해질 가능성이 높으며, 그에 따른 overfitting위험이 있음	Only when the base model is a decision tree, $d = 1$인 경우(stump)의 성능이 좋다고 알려져 있음



- 이 그림을 통해 알 수 있는 것
  - Random forest에 비해 boosting의 학습 속도가 느리다.
  - Boosting에서 base model의 복잡도가 적당히 높을 경우 좋은 성능을 보이며,
  - Random forest 역시 individual tree 사이의 상관성이 적당히 높은 경우가 아주 낮은 경우에 비해서 좋은 성능을 보인다.

Comparison

	Bagging	Random Forest	Boosting
Base model	Decision tree 이외의 model을 사용할 수 있음	Decision tree	Decision tree 이외의 model을 사용할 수 있음
B의 의미	Individual tree의 개수 & Bootstrap sample의 개수	Individual tree의 개수 & Bootstrap sample의 개수	Individual tree의 개수 (only) → Training에서는 original data만을 사용함
개별 tree의 independence	Independent	More independent than in bagging	Least independent → 사실상 Dependent
Overfitting	B에 의해서는 발생하지 않으나, 개별 tree의 depth에 의해 발생 가능 → 오히려 B가 늘어날수록 표본평균의 분산이 줄어들어 robust prediction을 얻을 수 있음	대체로 bagging과 같은 특징을 공유함 But 개별 tree에서 split에 사용할 feature의 개수 m가 많을수록 overfitting 위험이 있음	B가 많아짐으로 인해서 발생할 수 있음 → 따라서 cross-validation을 사용해 적절한 base model의 개수를 설정해야 함.

	Bagging	Random Forest	Boosting
학습 방법	Individual tree의 학습 결과에 대한 평균(regression) 및 majority vote(classification)	Individual tree의 학습 결과에 대한 평균(regression) 및 majority vote(classification)	Current model에서는 previous model로부터 줄어든 residual에 fitting → 결론적으로 개별 tree의 학습 결과에 대한 weighted average
Base model에 대한 pruning	하지 않음	하지 않음	하지 않음 → 일반적으로 아주 shallow depth의 tree를 사용함

Variable Importance

- Fitting하게 되는 individual base model을 split할 때 개별 feature가 metric 감소에 기여한 정도를 전체 B 개에 걸쳐서 평균한 값으로서 해당 feature의 importance를 측정한다.
  - Regression의 경우 개별 feature를 사용해 split했을 때 감소한 RSS의 평균
  - Classification의 경우 개별 feature를 사용해 split했을 때 감소한 Gini index/Cross entropy의 평균