XGBoosting

AdaBoosting review

Gradient Boosting

XGBoosting

실습 in R

AdaBoosting review

Check

Review 부스팅

- 대표적인 부스팅 알고리즘 : AdaBoost, Gradient Boosting Machine, XGBoost, LightGBM, CatBoost 등등
- 오늘 배울 부스팅은 XGBoost로 Gradient Boosting의 심화판
- 먼저, 저번시간에 배운Adaboosting은 여러 개의 약한 학습기를 순차적으로 학습,예측 하면서 잘 못 예측한 데이터에 가중치를 부여해 오류를 개선해 나가며 학습
- 일반적으로 부스팅 알고리즘은 Decision tree 모형을 주로 사용하는 것으로 알려져 있고, 과적합에 강한 장점을 지닌다.
- 하지만 다른 앙상블 모형과 마찬가지로 분류결과에 대한 해석이 불가능하다는 단점을 지닌다.

Check

Gradient Boosting 과 AdaBoost 차이점

- Adaboosting의 경우 각각의 약 분류기마다의 가중치를 통해, 정확도를 높였다면, Gradient Boosting의 경우 'gradients'를 이용해 정확도를 높임 (뒤에서 보다 깊게 배워요~)
- 예를 통해 Gradient Boosting의 전반적인 흐름을 파악해보자!

  Given : $(x_1, y_1),\ (x_2,\ y_2),\ \cdots, (x_n, y_n)$

  주어진 데이터에 대해 F(x)라는 모델을 하나 만들었다고 가정하자.

  이 때, 두 개의 데이터 포인트의 y값에 대해 $y_1 = 0.9,\ y_2 = 1.3$ 이라고 가정하자.
- 우리가 모델 F(x)를 통해 만든 모델에 의해 output을 계산하니 $\widehat{y_1} = 0.8,\ \widehat{y_2} = 1.4$ 이다.
- 이 때, 모델 F(x)의 어떠한 parameter도 건드리지 않고, 모델의 예측 성능을 높이는 방법이 뭐가 있을까?!
- H(x)라는 새로운 모델을 도입해서 F(x) + H(x)로 예측을 하면 F(x)의 수정 없이 정확도를 높일 수 있다!!!

Gradient Boosting

Gradient Boosting Machine (GBM) 이란?

Check

- 여러 트리로 구성된 앙상블 모델로 결과를 예측할 때, gradient descen를 이용해 순차적으로 트리를 만들어가며 이전 트리의 오차를 보완하는 방식으로 boosting 하는 방법
- 가중치를 부여하는 방법으로는 Gradient descent (경사하강법) 을 사용 > 뒤에서 설명
- GBM은 예측 성능이 높지만 Greedy Algorithm 으로 과적합이 빠르게 되고 시간이 오래 걸린다는 단점이 있다.
- 회귀와 분류 모두 사용 가능하다

  - 최적해를 구하는 데에 사용되는 근사적인 방법
  - 매순간 최적이라고 생각되는 것을 선택해 나가는 방식으로 진행하여 최종적인 최적해에 도달하는 기법
  - 앞의 선택이 이후 선택에 영향을 주지 않음
  - 문제 전체에 대한 최적해가 부분문제에 대해서도 역시 최적해가 된다.

Check

Gradient 란?

- 어떤 다변수 함수 f($x_1, x_2, \dots x_n$) 이 있을 때, f의 Gradient는 다음과 같이 정의된다.

gradient 기호 ← $\nabla f = \left(\frac{\partial f}{\partial x1}, \frac{\partial f}{\partial x2}, \frac{\partial f}{\partial x3}, \dots\dots\dots, \frac{\partial f}{\partial xn} \right)$ → x1~xn으로 구성된 f함수에서 x3만 변수로 보고 나머지 n-1개의 변수는 상수 취급해서 x3에 대해서만 미분

- 즉, Gradient는 각 변수로의 일차 편미분 값으로 구성되는 벡터이며 이 벡터는 f의 값이 가장 가파르게 증가하는 방향을 나타내며 벡터의 크기는 그 증가의 가파른 정도 (기울기)를 나타낸다.
- 예시). f(x,y) = $x^2 + y^2$ 의 Gradient는 $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ = (2x,2y) 이므로 (1,1)에서 f 값이 최대로 증가하는 방향은 (2,2)이고, 그 기울기는 ||(2,2)|| = $\sqrt{8}$ 이다.
- 또한 Gradient에 음수를 취하면 즉, $-\nabla f$ 는 f값이 가장 가파르게 감소하는 방향을 나타내게 된다.
- Gradient의 특성은 어떤 함수를 지역적으로 선형근사를 하거나 혹은 함수의 극점 (최대, 최소값 지점)을 찾는 용도로 활용될 수 있다.

Check

< Gradient descent 방법의 직관적 이해 >

Q. 만약 내가 울창한 밀림에 혼자 갇혔다면 산 정상으로 가기 위해서는 어떤 방법을 쓸 수 있을까요?

A . 방법은 간단합니다. 비록 실제 산 정상이 어디에 있는지는 모르지만 현재 위치에서 가장 경사가 가파른 방향으로 산을 오르다 보면 언젠가는 산 정상에 다다르게 될것입니다 :D

Q. 만약 내가 산정상이 아닌 깊은 골짜기를 찾고 싶을 때는요?

A . 현재 위치에서 가장 가파른 내리막 방향으로 산을 내려가보면 됩니다.

이와 같이 어떤 함수의 극대점을 찾기 위해 현재 위치에서의 gradient 방향으로 이동해 나가는 방법을 gradient ascent 방법, 극소점을 찾기 위해 gradient 반대 방향으로 이동해 나가는 방법을 gradient descent 방법이라고 합니다.

Check

< Learning rate란? >

Learning rate 직관적 이해

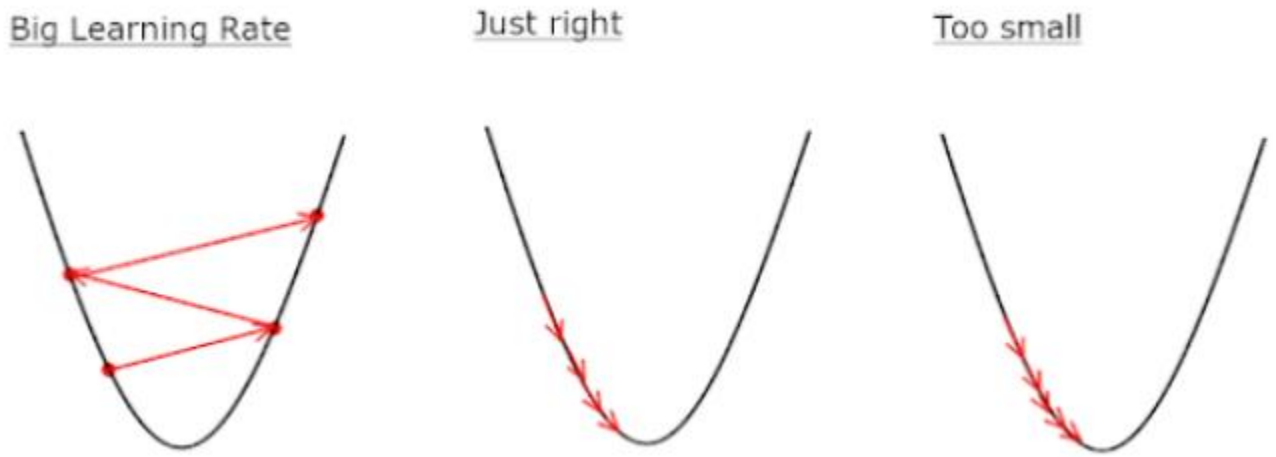
- 부산에서 서울까지 순간이동을 한다고 하자
- 서울방향으로 10km씩 순간이동을 하면 금방 도착할 것이다.
- 하지만 만약 1m씩 순간이동을 하면 너무 오래 걸린다.
- 그리고 1000km씩 순간이동을 하면 서울을 지나 오히려 서울과 더 멀어진 곳에 도착할 것이다.
- 즉, 목표점에 얼마큼씩 이동할 때 이 거리가 바로 학습률이다.

Check

< Learning rate란? >

국어사전 : 최적 알고리즘에서 파라미터의 조정을 통하여 성능을 향상시키기 위하여 학습할 때 완급을 조절하는 상수

- learning rate가 너무 크면? overshooting 문제 발생. 학습이 이루어지지 않을 뿐만 아니라 때론 cost 값이 오히려 커지는 현상들이 발생
- learning rate가 너무 작으면? 학습시키는게 많은 시간이 걸리고 만약 cost function이 con-convex라면 local minimum에 빠질 수도 있다.
- learning rate의 범위는 0~1이며, 주로 0.1을 많이 사용하지만 데이터 마다 가장 적절한 learning rate 찾아야 함 (필요 시)



Check

gradient boosting 알고리즘

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$ 데이터&Loss Function 준비

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$ Loss Function의 합을 최소로 하는 예측값을 초기 예측값으로 설정

Step 2: for $m = 1$ to M: 아래의 (A)~(D)과정을 사전에 정한 M번만큼 반복할 것

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$ loss function 을 prediction 변수로 미분한 값을 구해서 나온 predition 변수에 앞에서 구한 예측값을 넣음

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1\ldots J_m$ (A)단계에서 구한 잔차로 회귀나무를 만들어라

(C) For $j = 1\ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$ 다음 식을 만족하는 감마jm을 찾아라

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ 앞에서 완성한 잔차회귀트리를 통해 타겟값을 예측해라

Step 3: Output $F_M(x)$

어렵지 않아요!
한 단계씩 해봅시다: o

gradient boosting 알고리즘

Check

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Input : 데이터와 Loss Function 준비

$n = 3$

x_1

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

y_2

regression 에 많이 사용되는 Loss Function

$$\frac{1}{2}(\textbf{Observed} - \textbf{Predicted})^2$$

위의 Loss Function이 아닌
다른 Loss Function 써도 무방합니다!

gradient boosting 알고리즘

Check

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step1. 일정한 값을 갖는 초기모델을 만들어보자

: 초기모델은 Loss Function을 최소로 하는 감마(예측값)의 값

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

- 감마는 Predicted 값을 의미합니다.

- $L(y_i, \gamma) = \frac{1}{2}$ (Observed - Predicted)^2

- $\sum_{i=1}^{n} L(y_i, \gamma) = \frac{1}{2}(88 - Predicted)^2 + \frac{1}{2}(76 - Predicted)^2 + \frac{1}{2}(56 - Predicted)^2$

gradient boosting 알고리즘

Check

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

아래 식을 최소로 하는 Predicted 값을 찾기 위해 Predicted에 대해서 미분한 식을 0으로 두고 식을 푼다.

$$\frac{1}{2}(88 - \text{Predicted})^2 + \longrightarrow -(88 - \text{Predicted}) +$$

$$\frac{1}{2}(76 - \text{Predicted})^2 + \longrightarrow -(76 - \text{Predicted}) +$$

$$\frac{1}{2}(56 - \text{Predicted})^2 \longrightarrow -(56 - \text{Predicted})$$

$$= 0$$

$$\frac{d}{d\,\text{Predicted}}$$

gradient boosting 알고리즘

Check

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

아래 식을 최소로 하는 Predicted 값을 찾기 위해 Predicted에 대해서 미분한 식을 0으로 두고 식을 푼다.

$$\textbf{Predicted} = \frac{88 + 76 + 56}{3}$$

즉,

$$F_0(x) = \frac{88 + 76 + 56}{3}$$

= 73.3

이 값이 의미하는 것은??

- the initial predicted value (초기 예측값)
- The leaf predicts that all samples will weigh 73.3
- We initialized the model with a constant value 73.3

> 모든 관측값들에 대해서 73.3 으로 예측한다는 말

즉, 모든 input 에 대해서 예측값으로 73.3이 나오게 한 모델을 만든 것

이때, 이 73.3은 input data의 타겟변수들의 평균

gradient boosting 알고리즘

Check

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

$$-\frac{d}{d\,Predicted}\frac{1}{2}(\text{Observed - Predicted})^2 \rightarrow \boxed{(\text{Observed - Predicted})}$$

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

m =1 일때는 F(x) 즉, Predicted 자리에 F_0(x) 를 대입

앞에서 F_0(x) = 73.3 으로 구했다.

즉, r_{im} = (Observed - 73.3)

이때, i = sample number (데이터 개수)

m = tree that we're trying to build (트리개수)

gradient boosting 알고리즘

Check

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

Height (m)	Favorite Color	Gender	Weight (kg)	$r_{i,1}$
1.6	Blue	Male	88	14.7
1.6	Green	Female	76	2.7
1.5	Blue	Female	56	-17.3

앞에서 $r_{i,1}$ = (Observed - 73.3) 라고 구했으니

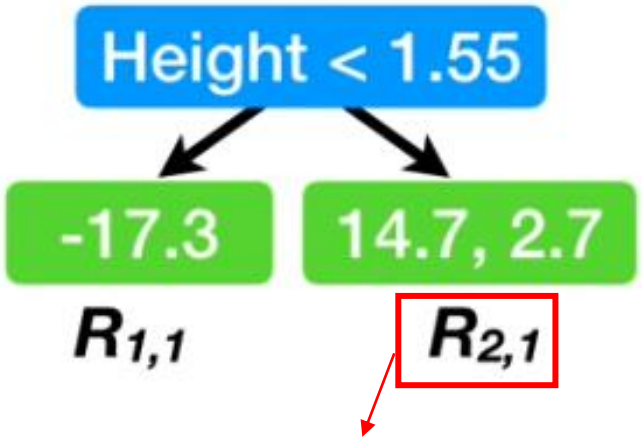
각 관측값의 Observed를 대입해주면

Step 2의 (A)과정 Compute $r_{i,m}$ 끝 !!!!

gradient boosting 알고리즘

Check

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$



Step2 (B)

: X데이터 (Height , Favorite Color, Gender) 로 $r_{i,1}$ 잔차를 예측하는 regression tree를 만든다.

그리고 각각의 terminal regions 을 $R_{j,m}$ 로 구분해주자.

이때, j = index for each leaf in the tree (해당 트리 각 leaf의 index)

m = index for the tree we just made (우리가 만든 트리 index)

That means j=2, since this is the second leaf,

and m =1 , since this is still the first tree

We've finished Part B of Step2 (step2 끝)

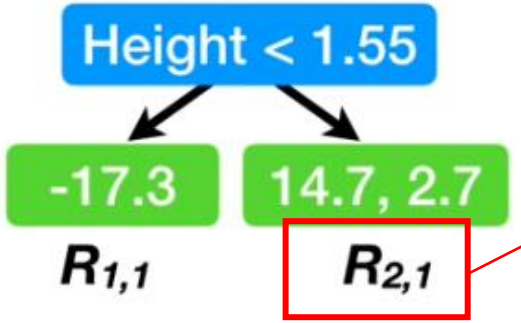
by fitting a Regression Tree to the residuals and labeling the leaves

(잔차에 대한 회귀트리 만들고 각 잎들에게 이름 붙여줌)

Check

gradient boosting 알고리즘

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$



$$\gamma_{2,1} = \underset{\gamma}{\operatorname{argmin}} \left[\frac{1}{2}(88 - (73.3 + \gamma))^2 + \frac{1}{2}(76 - (73.3 + \gamma))^2 \right]$$

- m = 1이니 $F_0(x) = 73.3$ (step 1에서 구함)
- y_i = 실제 weight 관측값
- 위의 식을 최소로 만드는 감마(예측값)를 찾는 것이니
  감마에 대해서 미분한 식을 0으로 두고 풀면 된다.

gradient boosting 알고리즘

Check

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

$$\frac{d}{d\gamma}\left[\frac{1}{2}(14.7-\gamma))^2 + \frac{1}{2}(2.7-\gamma))^2\right] \longrightarrow -14.7 + \gamma + -2.7 + \gamma \quad = 0$$

$$\gamma = \frac{14.7 + 2.7}{2} = 8.7$$

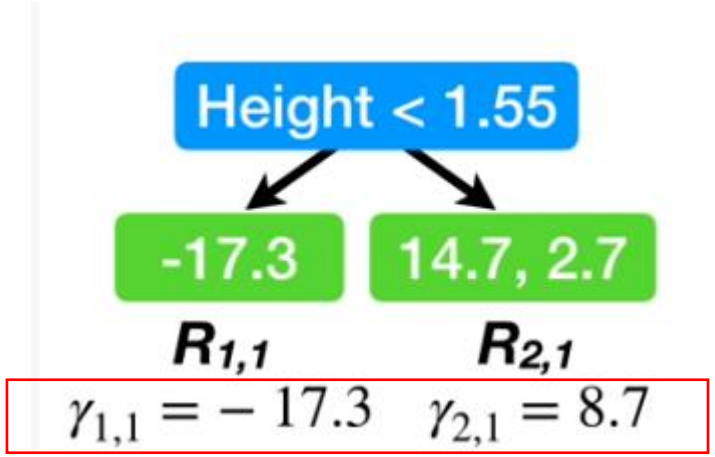
We end up with the average of the Residuals that ended in leaf $R_{2,1}$

(감마에 대해 미분한 값을 0으로 두고 푸니 감마가 leaf $R_{2,1}$ 에 있던 잔차들의 평균으로 나옴)

Check

gradient boosting 알고리즘

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$



We've finished Part C of Step2 by computing gamma values, or Output Values, for each leaf

(step2 C 끝

각 잎의 감마값(잔차 예측값) 찾음)

gradient boosting 알고리즘

Check

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

m = 10이니 대입해보면

$F_1(x) = F_0(x) + \nu$ (73.3 + Height < 1.55 → -17.3 ($R_{1,1}$), 14.7, 2.7 ($R_{2,1}$))

$\gamma_{1,1} = -17.3$ $\gamma_{2,1} = 8.7$

- learning rate (0,1)
- 이 문제에서는 0.1로 가정

실제예측

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

New Prediction for x2 :

$73.3 + 0.1 \times 8.7 = 74.2$

74.2 which is an improvement

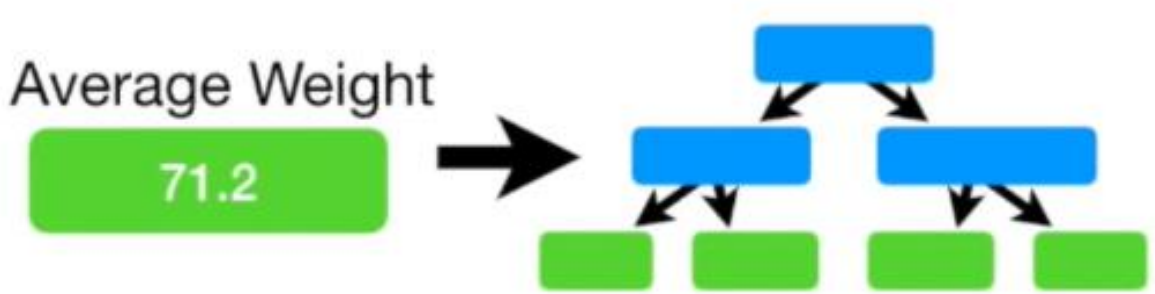
over the first Prediction, 73.3

초기모델은 73.3으로 예측했는데 잔차 트리를 하나 만든 후에는 74.2로 예측 > 예측력 향상)

1분 요약

gradient boosting 1분 요약

Check



- single leaf 모델이 예측한 타겟 추정 값
  : (88 + 76 + 56 + 73 + 77 + 57) / 6 = 71.2
- 즉, single leaf 모델로 몸무게를 예측한다면
  모든 새로운 데이터에 대해 71.2 kg 이라고 예측할 것이다.

- 이제 우리가 할 일은 single leaf 에서 예측한 값과 실제 값의 차이 즉, error를 반영한 새로운 트리를 만드는 것
- 오른쪽 그림처럼 leaf가 한 개가 아닌 4개, 혹은 8개, 32개인 트리를 만드는 것

gradient boosting 1분 요약

Check

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

- single leaf 모델이 예측한 타겟 추정 값
  : 71.2 kg
- 실제 관측값과 예측값의 차이 (error)를 구한다.
- 예). 88 kg – 71.2 kg = 16.8 kg
- 이를 Pseudo Residual 이라고 한다.

⇩

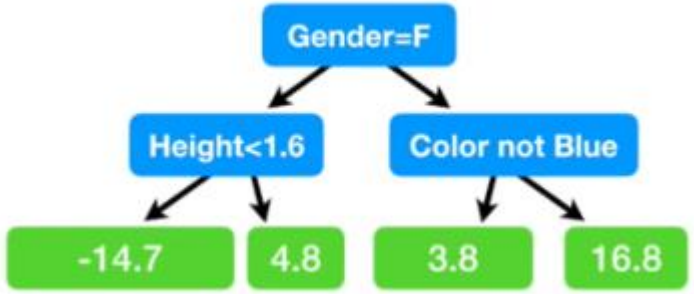
Height, Favorite Color, Gender 를 통해

Residual을 예측하는 tree를 만들 것 !

Check

gradient boosting 1분 요약

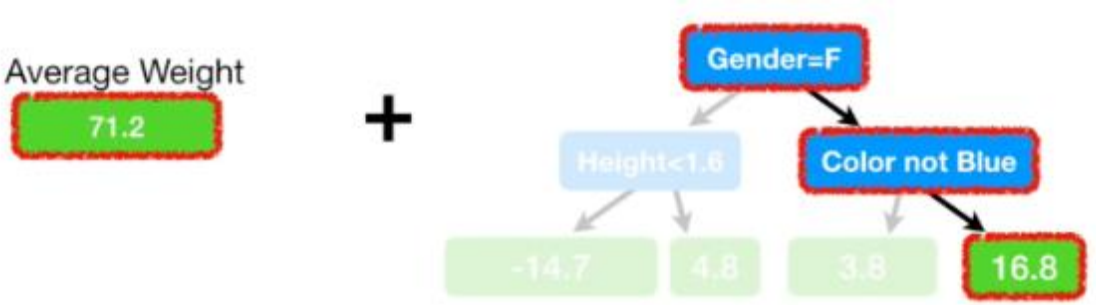
- 맨 처음 노드는 성별을 기준으로, 두번째 노드에서는
  키 1.6m / 파란색을 좋아하는지 여부에 따라 트리 생성
- leaf node에는 분류되는 건 Height가 아니라 앞에서 구한 Residual



- 이때, leaf node.에 두개의 Residual 값이 오면 평균으로 치환

gradient boosting 1분 요약

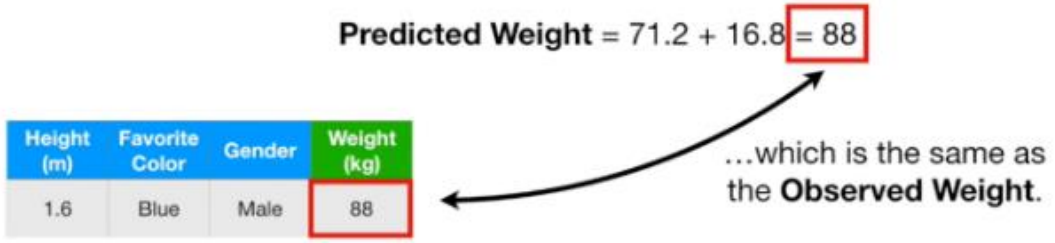
Check



- 처음에 구한 single leaf 와 두번째로 구한 트리를 조합해서 몸무게를 예측 (F := F + h)
- ex). 성별이 Male이고 파란색을 좋아하면 residual을 16.8로 예측
  Predicted Weight = 71.2 + 16.8 = 88 kg

- 예측한 몸무게와 실제 몸무게가 일치!!!
- 그렇다면 좋은 모델??
- train data에 과적합된 모델
- 이를 해결하기 위해서는?

NO!



Learning Rate 활용

gradient boosting 1분 요약

Check

- 앞에서 구한 모델에 학습률을 0.1로 할 경우
- ex). 성별이 Male이고 파란색을 좋아하면 residual을 16.8로 예측

  Predicted Weight = 71.2 + (0.1 X16.8) = 72.9 kg
- 학습률 안 곱했을 때 결과보다는 실제값과 차이가 있지만

  첫 single leaf 모델이 예측한 값보다는 실제값 (88kg)에 더 가까워 짐

Gradient Boost모델은 실제 값에 조금씩

가까워지는 방향으로 학습

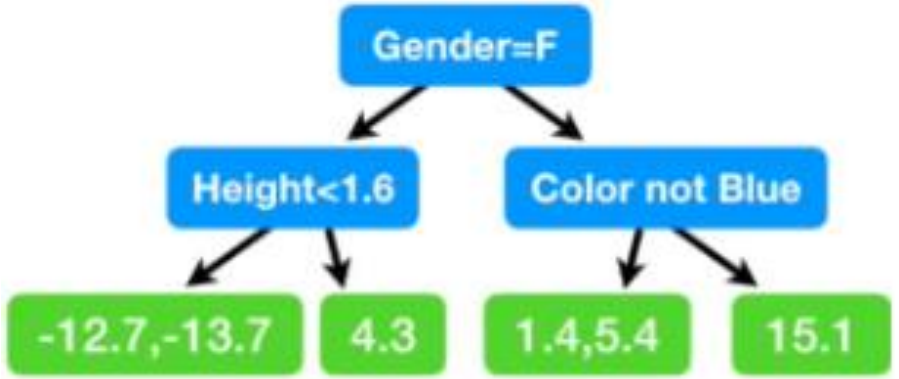
gradient boosting 1분 요약

Check

Height (m)	Favorite Color	Gender	Weight (kg)	1 Residual	2 Residual
1.6	Blue	Male	88	15.1	16.8
1.6	Green	Female	76	4.3	4.8
1.5	Blue	Female	56	-13.7	-15.2
1.8	Red	Male	73	1.4	1.8
1.5	Green	Male	77	5.4	5.8
1.4	Blue	Female	57	-12.7	-14.2

- 앞에서 구한 모델에 의한 예측값을 통해 다시 residual 구함 (1)
- single leaf 모델에 의한 예측값을 통해 구한 residual (2)
- 새롭게 구한 residual이 초기 residual보다 작아짐을 알 수 있다.
- 즉, 조금씩 실제 값으로 다가가고 있다는 뜻 (F := F + h1 + h2)

- 위에서 나온 residual로 새로운 트리를 만듬 (앞 선 방법과 동일)
- 이 과정을 계속 **반복**



gradient boosting 1분 요약

Check

- 앞에서 구한 모델에 학습률을 0.1로 할 경우
- ex). 성별이 Male이고 파란색을 좋아하면
- tree1) residual을 16.8로 예측

  Predicted Weight = 71.2 + (0.1 X16.8) = 72.9 kg
- tree2) residual을 15.1로 예측
- Predicted Weight = 71.2 + (0.1X16.8) + (0.1X15.1) = 74.4
- 점점 실제 값(88kg) 에 가까워 지고 있음을 확인 가능

Gradient Boosting - 분류

Gradient Boosting - 분류

Check

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Input : 데이터와 Loss Function을 준비

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

Loss Function

: $-[y_i \log(p) + (1 - y_i)\log(1-\mathrm{p})]$

Gradient Boosting - 분류

Check

Loss Function p에 대한 함수를 log(odds)의 함수로 변환시켜주는 과정

Loss Function

: $-[y_i \log(p) + (1 - y_i)\log(1-p)]$

= - [observed X log(p) + (1-observed) X log(1-p)

= -observed X log(p) –log(1-p) + observed X log(1-p)

= -observed X [log(p) – log(1-p)] – log(1-p)

= -observed X log(odds) – log(1-p)

$$\log(1 - p) = \log\left(1 - \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}\right) = \log\left(\frac{1 + e^{\log(odds)}}{1 + e^{\log(odds)}} - \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}\right) = \log\left(\frac{1}{1 + e^{\log(odds)}}\right)$$

= **-Observed** x log(odds) + log(1+ $e^{\log(odds)}$)

Gradient Boosting - 분류

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$\frac{d}{d\ log(odds)}$ -**Observed** × log(odds) + log(1 + $e^{log(odds)}$) = -**Observed** + ***p***

감마 : log(odds) = log($\frac{p}{1-p}$)

참고 : p = $\frac{e^{log(odds)}}{1+e^{log(odds)}}$

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

-**1** × log(odds) + log(1+ $e^{log(odds)}$) → -1 + $\frac{e^{log(odds)}}{1 + e^{log(odds)}}$ → 1-p

-**1** × log(odds) + log(1+ $e^{log(odds)}$) → -1 + $\frac{e^{log(odds)}}{1 + e^{log(odds)}}$ → 1-p

-**0** × log(odds) + log(1+ $e^{log(odds)}$) → -0 + $\frac{e^{log(odds)}}{1 + e^{log(odds)}}$ → -p

$\frac{d}{d\ log(odds)}$

2 - 3p=0
즉, p = 2/3

log(odds) = log(2) = 0.69 = $F_0(x)$

- 모든 input data에 대한 log(odds)의 예측값은 0.69이다.
- 모든 input data에 대한 p (확률)의 예측값은 2/3 = 0.67 이다.

Gradient Boosting - 분류

Check

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

(**Observed** - ***p***) = **Pseudo Residual**

m=1일때, p값은 2/3 이니 대입 :

r_{im} = observed – 0.67

Likes Popcorn	Age	Favorite Color	Loves Troll 2	$r_{i,1}$
Yes	12	Blue	Yes	0.33
No	87	Green	Yes	0.33
No	44	Blue	No	-0.67

= 1-0.67

= 1-0.67

= 0-0.67

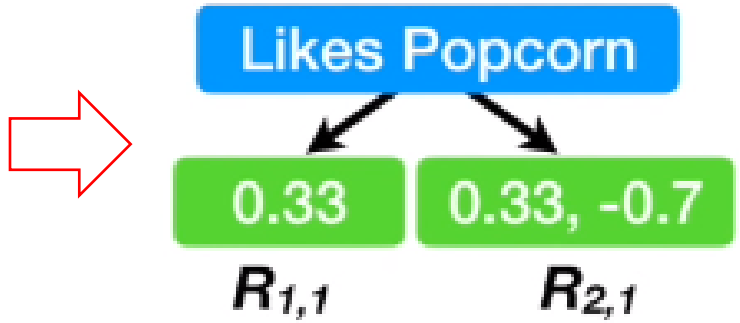
Gradient Boosting - 분류

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

We will build a regression tree using **Likes Popcorn**, **Age** and **Favorite Color**…

…to predict the **Residuals**.

Likes Popcorn	Age	Favorite Color	Loves Troll 2	$r_{i,1}$
Yes	12	Blue	Yes	0.33
No	87	Green	Yes	0.33
No	44	Blue	No	-0.67

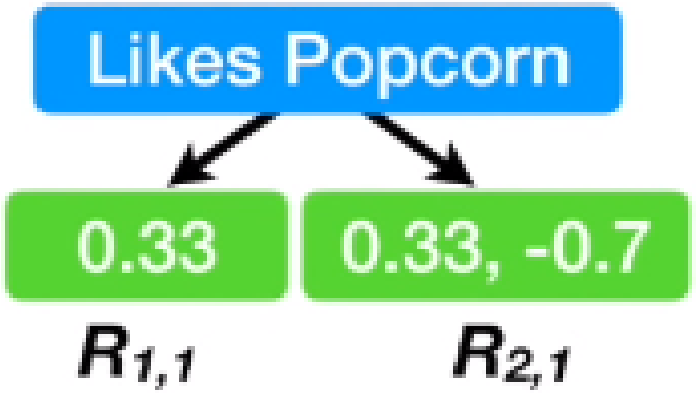


Gradient Boosting - 분류

https://www.youtube.com/watch?v=StWY5QWMXCw&t=1s

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

$$L(y_1, F_{m-1}(x_1) + \gamma) = -y_1 \times [F_{m-1}(x_1) + \gamma] + \log(1 + e^{F_{m-1}(x_1)+\gamma})$$



위의 function을 감마에 대해 미분 is hard…

$$L(y_1, F_{m-1}(x_1) + \gamma) \approx L(y_1, F_{m-i}(x_1)) + \frac{d}{dF()}(y_1, F_{m-1}(x_1))\gamma + \frac{1}{2}\frac{d^2}{dF()^2}(y_1, F_{m-1}(x_1))\gamma^2$$

So, 테일러 전개로 나타냄 > 자세히 알고 싶은 분은 위 링크 참고해주세요!

계산과정 생략 > 계산 결과만 보면

$$\gamma = \frac{\text{Residual}}{p \times (1 - p)}$$

leaf에 잔차 1개있을 때 ($R_{1,1}$)

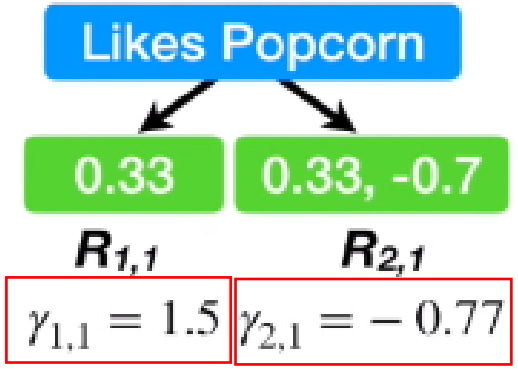
$$\gamma = \frac{\text{Residual}_2 + \text{Residual}_3}{[p_2 \times (1 - p_2)] + [p_3 \times (1 - p_3)]}$$

leaf에 잔차 2개있을 때 ($R_{2,1}$)

Gradient Boosting - 분류

Check

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$



$$\gamma = \frac{\text{Residual}}{p \times (1 - p)}$$

$$\frac{0.33}{(0.67 \text{ X } (1 - 0.67))} = 1.5$$

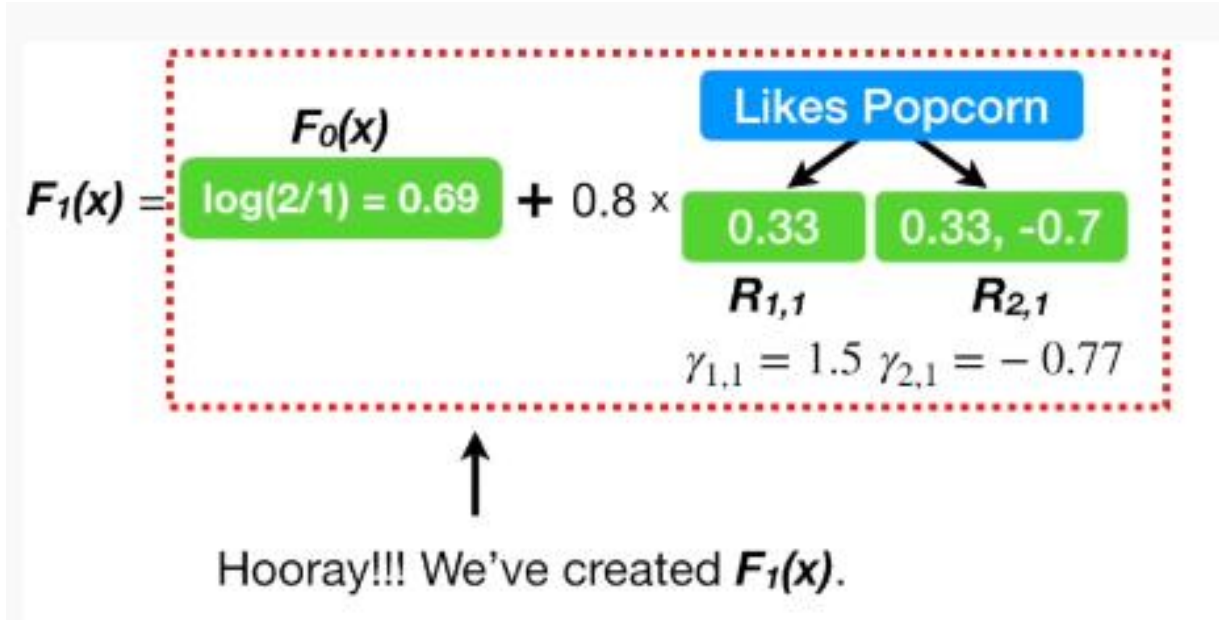
$$\gamma = \frac{\text{Residual}_2 + \text{Residual}_3}{[\,p_2 \times (1 - p_2)\,] + [\,p_3 \times (1 - p_3)\,]}$$

$$\frac{0.33 + (-0.7)}{\big(0.67 \text{ X } (1 - 0.67)\big) + (0.67 \text{ X } (1 - 0.67))} = -0.77$$

Gradient Boosting - 분류

Check

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$



실제예측

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

0.69 + 0.8 × 1.5 = 1.89

(Yes,12,Blue)의 Predicted log(odds)

Gradient Boosting - 분류

Check

$F_2(x) = F_0(x)$ log(2/1) = 0.69 + 0.8 × (Likes Popcorn: 0.33 / 0.33, -0.7) + 0.8 × (Age > 65.5: 0.48 / 0.13, -0.5)

$R_{1,1}$ $R_{2,1}$ $\gamma_{1,1} = 1.5$ $\gamma_{2,1} = -0.77$

$R_{1,2}$ $R_{2,2}$ $\gamma_{1,2} = 1.9$ $\gamma_{2,2} = -1.1$

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	90	Green	???

new data

The predicted **log(odds)** that this person will **Love Troll 2** = log(2/1) + (0.8 × 1.5) + (0.8 × 1.9) = 3.4

The predicted probability that this person will **Love Troll 2** = $\frac{e^{3.4}}{1 + e^{3.4}}$ = 0.97

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	90	Green	YES!!!

gradient boosting 장단점

Check

Gradient Boost ?

장점 : 예측 성능이 좋다

단점 :

- 수행시간이 오래 걸리고, 하이퍼 파라미터 튜닝 노력이 필요하다
- 약한 학습기의 순차적인 예측 오류 보정을 통해 학습을 진행하기 때문에 CPU 코어 시스템을 사용하더라도 병렬 처리가 지원되지 않아서 대용량 데이터의 경우 학습에 매우 많은 시간이 필요
- greedy algorithm 이고, 과적합이 빠르게 되는 경향이 있다.

XGBoosting

Check

XGBoost란?

eXtreme Gradient Boosting 의 약자로, GBM을 속도와 성능면에서 향상시킨 알고리즘

장점 1 : GBM 모델들의 실행에 비해 빠르다

- 병렬처리와 최적화를 장점으로 내세우는 gradient boosting 알고리즘

http://machinelearningkorea.com/2019/07/25/xgboost-%EC%9D%98-%EB%B3%91%EB%A0%AC%EC%B2%98%EB%A6%AC%EA%B0%80-%EC%96%B4%EB%96%BB%EA%B2%8C-%EA%B0%80%EB%8A%A5%ED%95%A0%EA%B9%8C/

장점 2 : 성능이 더 좋다

- Regularization => Overfitting 방지
- Missing Value => In-built routine to handle missing values

https://discuss.xgboost.ai/t/how-does-xgboost-handle-missing-data/217

- Built-in Cross-Validation => 반복수행시마다 내부적으로 학습데이터와 평가데이터 셋에 대한 교차 검증을 수행해 이로 인해 최적의 값 찾기 쉬움 (<-> GBM은 grid-search)

Check

XGBoost – Object function

XG Boost는 기존의 GBM 모델에 정규화 항을 추가한 셈

Object Function = $Obj(\Theta) = L(\Theta) + \Omega(\Theta)$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

이때, L(θ) = $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$

이는 회귀문제일때는 : $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$

분류문제일때는 : $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$

XGBoost - Object function

Check

앞서 loss function이 최소가 되게 하는 잔차(notation= γ)를 이차방정식 미분하여 구했습니다!

이것을 조금 Generalize한 버전을 보여드릴게요~ (Taylor expansion)

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

t번째 단계에서의 예측값

= t-1번째에서의 예측값 + 잔차에 적합시킨 t번째 트리

이라고 할 때,

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

우리의 목표는 Object function을 minimize하는 tree f를 찾는 것임을 잊지말자!

-> XG Boost는 이때 테일러 2차 급수전개를 이용.

1. Theoretical benefit : 이론적으로 앞선 잔차를 이용한 식과 같은 결과
2. Engineering benefit : loss function이 무엇이 되든지 코드를 짤 때 module를 분리할 수 있다는 장점이 있다.

XGBoost – Object function

Check

앞서 loss function이 최소가 되게 하는 잔차(notation= γ)를 이차방정식 미분하여 구했습니다!

이것을 조금 Generalize한 버전을 보여드릴게요~ (Taylor expansion)

이해가 안되면 여기로
https://towardsdatascience.com/xgboost-mathematics-explained-58262530904a

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

t번째 단계에서의 예측값

= t-1번째에서의 예측값 + 잔차에 적합시킨 t번째 트리

이라고 할 때,

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

우리의 목표는 Object function을 minimize하는 tree f를 찾는 것임을 잊지말자!

-> XG Boost는 이때 테일러 2차 급수전개를 이용.

1. Theoretical benefit : 이론적으로 앞선 잔차를 이용한 식과 같은 결과
2. Engineering benefit : loss function이 무엇이 되든지 코드를 짤 때 module를 분리할 수 있다는 장점이 있다.

XGBoost – Regularization

Check

그럼 도대체 Object function의 Regularization part는 무슨 의미일까?

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss　　Complexity of the Trees

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves　　L2 norm of leaf scores

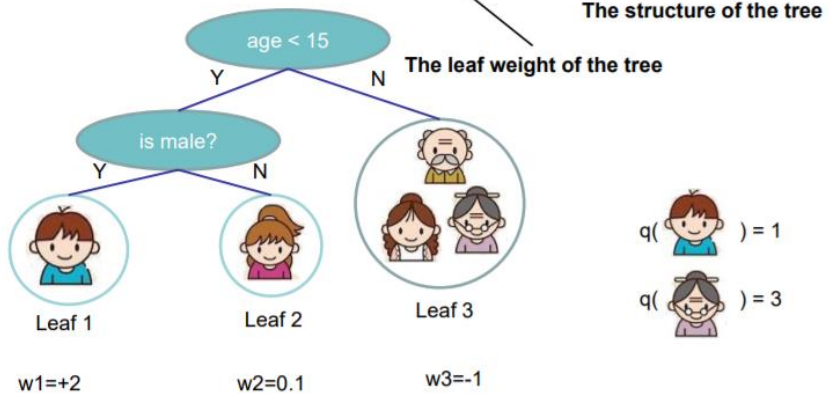
- ft : t-1번째 예측값의 잔차에 대한 트리
- γ(gamma):definable penalty.
  감마값이 커지면 가지치기를 더 많이 하게 된다. (∵ Gain-gamma <0 이면 prune하니까)
- T: number of terminal nodes
- λ(lambda): Regularization penalty.
  람다값이 커지면 개별관측치에 대한 예측민감도 ↓ = Regularization의 역할.
- Wj :j번째 leaf node의 leaf weight (leaf score)

Cf) 갑자기 Wj?

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

The structure of the tree

The leaf weight of the tree



XGBoost – Regularization

Check

그럼 도대체 Object function의 Regularization part는 무슨 의미일까?

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

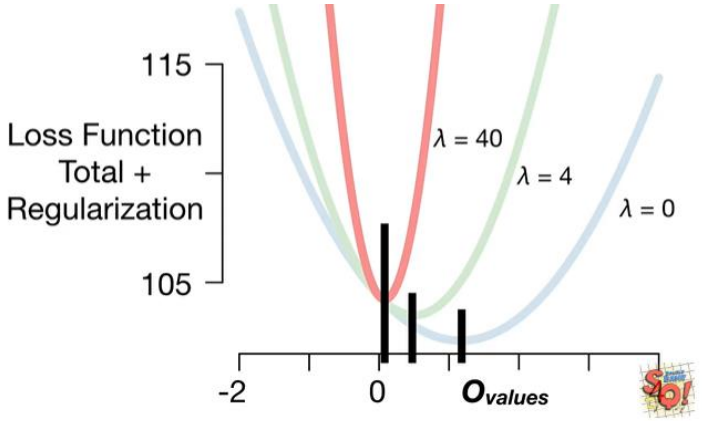
Training loss　　Complexity of the Trees

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves　　L2 norm of leaf scores

- ft : t-1번째 예측값의 잔차에 대한 트리
- γ(gamma):definable penalty.
  감마값이 커지면 가지치기를 더 많이 하게 된다. (∵ Gain-gamma <0
- T: number of terminal nodes
- λ(lambda): Regularization penalty.
  람다값이 커지면 개별관측치에 대한 예측민감도 ↓ = Regularization의 역할.
- Wj :j번째 leaf node의 leaf weight (leaf score)

Q. λ(lambda)가 커지면?



Ft값이 0에 가까워 짐
= 즉 t-1번째 예측값에서 t번째 예측값으로의 개선이 거의 안이루어짐
= t-1단계에서 확실히 맞추지 못한 미세한 개별 관측치마다
민감하게 예측하지 않겠다
= Regularization

XGBoost – Regularization

Check

그럼 도대체 Object function의 Regularization part는 무슨 의미일까?

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

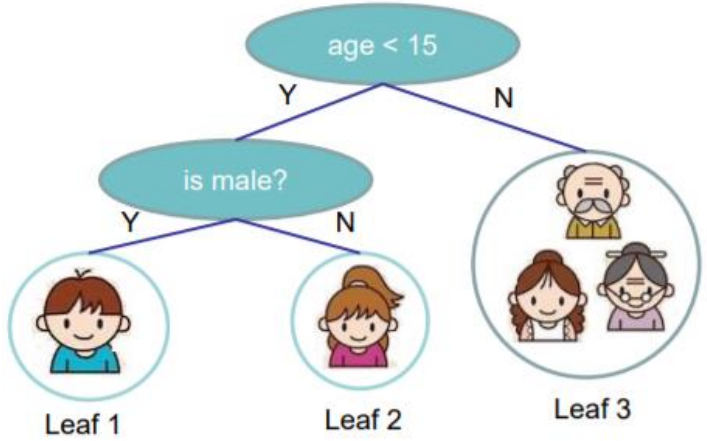
Training loss　　Complexity of the Trees

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves　　L2 norm of leaf scores

- ft : t-1번째 예측값의 잔차에 대한 트리
- γ(gamma):definable penalty.
  감마값이 커지면 가지치기를 더 많이 하게 된다. (∵ Gain-gamma <0
- T: number of terminal nodes
- λ(lambda): Regularization penalty.
  람다값이 커지면 개별관측치에 대한 예측민감도 ↓ = Regularization의 역할.
- Wj :j번째 leaf node의 leaf weight (leaf score)

Q. 이 트리의 오메가 값은?



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

XGBoost – Object function

Check

다시 돌아와서… $Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$

결국엔 위 object function을 w에 대한 식으로 나타내면,

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2$$

이때, $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

이 Object function을 minimize하는 w(leaf score)를 찾는게 우리의 목표다!

위 식은 w에 대한 식이므로 t시점에 상수가 되는 항들을 정리하면 아래와 같다.

Object function = $\sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T$

XGBoost – Object function

Check

즉, 이렇게 정리된다.

예를 들어, $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$ 라고 할 때,

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

이 식을 minimize하는 w값과 그때의 object function 값은 다음과 같다! (계산은 생략…)

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

XG Boost : 정규화 항까지 추가하여 기존 GBM모델의 성능을 향상시킨 빠른 모델!

실습 in R

Check

1. Load packages and data

# 패키지 설치
install.packages("xgboost")
코드를 통해 "xgboost" library를 설치한 뒤,

좌측처럼 패키지와 데이터를 불러와 줍니다!!

```
# 패키지 불러오기
library(xgboost)
# 데이터 불러오기 (내장된 iris 데이터 사용할게요~)
data(iris)
```

데이터도 한번
확인해보시고!

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

2. Label conversion & Split the data for training and test (7:3)

Check

XGBoost의 경우, class가 0부터 시작하는 integer format으로 구성돼있습니다!

1) 그에 맞게 데이터를 수정해주기 위해, 'iris' 데이터의 species 변수는 따로 빼서 먼저 저장해준 후,
2) label 이라는 변수에 0부터 시작되는 class가 될 수 있도록 포맷을 바꿔줍니다!!

```
> # XGBoost는 class들이 0부터 시작하는 integer format으로 형성되어 있음
> # 따라서 그 형식에 맞게 바꿔준다!
> species = iris$Species
> label = as.integer(iris$Species)-1
> iris$Species = NULL
```

Sample 함수를 이용해, train set과 test set을 만들어준 뒤, 잘 만들어졌는지 확인!

```
> set.seed(113)
> n = nrow(iris)
> train.index = sample(n,floor(0.7*n))
> train.data = as.matrix(iris[train.index,])
> train.label = label[train.index]
> test.data = as.matrix(iris[-train.index,])
> test.label = label[-train.index]
> length(train.label)
[1] 105
> length(test.label)
[1] 45
```

3. Create the xgb.Dmatrix objects

Check

두 데이터 셋을 데이터프레임 형태에서
xgb.matrix 형태로 변경시켜주기 위해
xgb.Dmatrix 함수로 변환시켜줍니다~

작업이 잘 수행됐는지 확인도 한 번~!

```
> # Transform the two data sets into xgb.Matrix
> xgb.train = xgb.DMatrix(data=train.data,label=train.label)
> xgb.test = xgb.DMatrix(data=test.data,label=test.label)
> xgb.train
xgb.DMatrix  dim: 105 x 4  info: label  colnames: yes
> xgb.test
xgb.DMatrix  dim: 45 x 4  info: label  colnames: yes
```

이제 모델을 돌려보기 전까지 **라이브러리에서 요구하는 형태로 데이터 셋을 수정**하는
작업을 마쳤습니다~!!

3. Create the xgb.Dmatrix objects

Check

다른 데이터를 사용해서 한번 더 해보겠습니다!

```
> data
            y x1 x2 x3 x4 x5 x6 x7 x8 x9
1      benign  5  1  1  1  2  1  3  1  1
2      benign  5  4  4  5  7 10  3  2  1
3      benign  3  1  1  1  2  2  3  1  1
4      benign  6  8  8  1  3  4  3  7  1
5      benign  4  1  1  3  2  1  3  1  1
6   malignant  8 10 10  8  7 10  9  7  1
7      benign  1  1  1  1  2 10  3  1  1

'data.frame':   683 obs. of  10 variables:
 $ y : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1
 $ x1: int  5 5 3 6 4 8 1 2 2 4 ...
 $ x2: int  1 4 1 8 1 10 1 1 1 2 ...
 $ x3: int  1 4 1 8 1 10 1 2 1 1 ...
 $ x4: int  1 5 1 1 3 8 1 1 1 1 ...
 $ x5: int  2 7 2 3 2 7 2 2 2 2 ...
 $ x6: int  1 10 2 4 1 10 10 1 1 1 ...
 $ x7: int  3 3 3 3 3 9 3 3 1 2 ...
 $ x8: int  1 2 1 7 1 7 1 1 1 1 ...
 $ x9: int  1 1 1 1 1 1 1 1 5 1 ...
```

마찬가지로 숫자로 바꿔주고, 1을 빼주면 0과 1로 만들어집니다!!

```
> data$y <- as.numeric(data$y) -1
> head(data$y)
[1] 0 0 0 0 0 1
```

Factor형 변수를 0부터 시작하는 숫자로 만들었다면, 앞과 같은 내용으로 xgb.matrix를 만들 수 있습니다!!

```
> x<-model.matrix(~.,data=data %>% select(-y))
> x <- as.matrix(x[,-1])
> x<-as.data.frame(x)
> train_mat<-as.matrix(x[,-1])
> xgb.matrix<-xgb.DMatrix(train_mat,label=data$y)
> xgb.matrix
xgb.DMatrix  dim: 683 x 8  info: label  colnames: yes
```

4. Define the main parameters

Check

이제 본격적으로 모델을 설계해보겠습니다!!
파라미터를 설정할 건데요,,,상당히 다양한 함수가 존재합니다ㅠㅠ,,,

하나씩 천천히 공부해보겠습니다~

```
> # Define the parameters for multinomial classification
> num_class = length(levels(species))
> num_class
[1] 3
> params = list(
+   booster="gbtree",
+   eta=0.001,
+   max_depth=5,
+   gamma=3,
+   subsample=0.75,
+   colsample_bytree=1,
+   objective="multi:softprob",
+   eval_metric="mlogloss",
+   num_class=num_class
+ )
```

우선 XGBoost의 파라미터는 아래와 같이
1) General Parameters
2) Booster Parameters
3) Learning Task Parameters
3개 카테고리로 구분되는데요,,!

하나씩 살펴보겠습니다!!

4. Define the main parameters

Check

```
> # Define the parameters for multinomial classification
> num_class = length(levels(species))
> num_class
[1] 3
> params = list(
+   booster="gbtree",
+   eta=0.001,
+   max_depth=5,
+   gamma=3,
+   subsample=0.75,
+   colsample_bytree=1,
+   objective="multi:softprob",
+   eval_metric="mlogloss",
+   num_class=num_class
+ )
```

첫번째, **General Parameters** 입니다!!

XGBoost의 전반적인 기능을 정의하며,
booster [default=gbtree] 에는 두가지 옵션이 있습니다
- gbtree: tree-based models
- gblinear: linear models

일반적으로 gbtree의 성능이 더 좋다고 합니다!

4. Define the main parameters

Check

```
> # Define the parameters for multinomial classification
> num_class = length(levels(species))
> num_class
[1] 3
> params = list(
+   booster="gbtree",
+   eta=0.001,
+   max_depth=5,
+   gamma=3,
+   subsample=0.75,
+   colsample_bytree=1,
+   objective="multi:softprob",
+   eval_metric="mlogloss",
+   num_class=num_class
+ )
```

두번째, **Booster Parameters** 입니다!!
(booster = "gbtree" 기준으로 공부할게요!!)

1. eta의 경우가 앞에서 배운 learning rate!!
- Default = 0.3, 일반적으로 0.01 ~ 0.2로 설정
2. gamma의 경우 분할을 수행하는데 필요한 최소 손실 감소를 지정할 수 있습니다.
3. subsample [default=1]
- 각 트리마다의 관측 데이터 샘플링 비율.
- 값을 적게 주면 over-fitting을 방지, 값을 너무 작게 주면 under-fitting.
- 일반적으로 0.5-1
4. colsample_bytree [default=1]
- 각 트리마다의 feature 샘플링 비율.
- 일반적으로 0.5-1

4. Define the main parameters

Check

```
> # Define the parameters for multinomial classification
> num_class = length(levels(species))
> num_class
[1] 3
> params = list(
+   booster="gbtree",
+   eta=0.001,
+   max_depth=5,
+   gamma=3,
+   subsample=0.75,
+   colsample_bytree=1,
+   objective="multi:softprob",
+   eval_metric="mlogloss",
+   num_class=num_class
+ )
```

마지막, **Learning task Parameters** 입니다!!
각 단계에서 계산할 최적화 목표를 정의합니다.

1. objective [default=reg:linear]
- binary:logistic : 이진 분류를 위한 로지스틱 회귀, 예측된 확률을 반환한다. (not class)
- multi:softmax : softmax를 사용한 다중 클래스 분류, 예측된 클래스를 반환한다. (not probabilities)
- multi:softprob : softmax와 같지만 각 클래스에 대한 예상 확률을 반환한다.

2. eval_metric [default according to objective]
- 회귀 분석인 경우 'rmse'를, 클래스 분류 문제 경우 'error'를 default로 사용.
- rmse : root mean square error
- error : Binary classification error rate

5. Train the model

Check

이제 모델을 만들겠습니다!!

```
> # Train the XGBoost classifer
> xgb.fit=xgb.train(
+   params=params,
+   data=xgb.train,
+   nrounds=2500,
+   nthreads=1,
+   early_stopping_rounds=10,
+   watchlist=list(val1=xgb.train,val2=xgb.test),
+   verbose=0
+   #verbose = 1로 두면 each round마다 결과를 확인할 수 있다.
+ )
```

또 새로운 파라미터들이 나오는데요,,,,!!
몇 개만 짚어서 알아보겠습니다!!

1. nrounds : 총 반복 횟수
2. early_stopping_rounds :
   k로 설정했을 때, k번의 반복동안 정확도 개선이 안된다면, 멈추게 되어 연산량을 줄입니다

https://xgboost.readthedocs.io/en/latest/parameter.html
여기 참고하시면 parameter 전체 설명 나와있으니까
참고 하시면 좋습니다!!

5. Train the model

Check

잘 만들어졌는지 모델 확인!!

```
> # Review the final model and results
> xgb.fit
##### xgb.Booster
raw: 1.7 Mb
call:
  xgb.train(params = params, data = xgb.train, nrounds = 2500,
    watchlist = list(val1 = xgb.train, val2 = xgb.test), verbose = 0,
    early_stopping_rounds = 10, nthreads = 1)
params (as set within xgb.train):
  booster = "gbtree", eta = "0.001", subsample = "0.8", colsample_bytree = "1", objective = "multi:softprob", eval_metric = "mlogloss", num_class = "3", nthreads = "1", silent = "1"
xgb.attributes:
  best_iteration, best_msg, best_ntreelimit, best_score, niter
callbacks:
  cb.evaluation.log()
  cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
    verbose = verbose)
# of features: 4
niter: 2069
best_iteration : 2059
best_ntreelimit : 2059
best_score : 0.464746
nfeatures : 4
evaluation_log:
    iter val1_mlogloss val2_mlogloss
       1      1.097263      1.097510
       2      1.095919      1.096417
---
    2068      0.134320      0.464774
    2069      0.134202      0.464781
```

6. Predict test set outcomes

이제 test set에 대하여 잘 어떻게 예측했는지를 뽑아줍니다!!

```
> # Predict outcomes with the test data
> xgb.pred = predict(xgb.fit,test.data,reshape=T)
> xgb.pred = as.data.frame(xgb.pred)
> xgb.pred
          V1         V2         V3
1  0.87463510 0.06486092 0.06050399
2  0.87463510 0.06486092 0.06050399
3  0.87403774 0.06549962 0.06046267
4  0.87463510 0.06486092 0.06050399
5  0.87463510 0.06486092 0.06050399
```

보시면 column 명이 V1, V2, V3로 되어있습니다 -> 바꿔줄게요~

```
> colnames(xgb.pred) <- c("setosa", "versicolor", "virginica")
> xgb.pred
      setosa versicolor  virginica
1  0.87463510 0.06486092 0.06050399
2  0.87463510 0.06486092 0.06050399
3  0.87403774 0.06549962 0.06046267
4  0.87463510 0.06486092 0.06050399
5  0.87463510 0.06486092 0.06050399
```

잘 바꼈습니다~

6. Predict test set outcomes

Check

Softprob함수로 돌렸기 때문에,
확률이 나오게 됩니다. 이 확률 값이 제일 높은 class로 예측할 수 있게 apply함수로 setosa, versicolor, virginica 세 개중에 가장 높은 확률의 class로 바꿔주겠습니다~

```
> # Use the predicted label with the highest probability
> xgb.pred$prediction = apply(xgb.pred,1,function(x) colnames(xgb.pred)[which.max(x)])
> xgb.pred$prediction <- as.factor(xgb.pred$prediction)
> xgb.pred$prediction
 [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
[10] setosa     setosa     setosa     setosa     setosa     versicolor versicolor versicolor versicolor versicolor
[19] versicolor versicolor virginica  versicolor versicolor versicolor versicolor versicolor versicolor versicolor
[28] versicolor virginica  versicolor virginica  virginica  virginica  virginica  virginica  versicolor
[37] virginica  virginica  virginica  versicolor virginica  versicolor versicolor virginica  virginica
Levels: setosa versicolor virginica
> xgb.pred$label = levels(species)[test.label+1]
> xgb.pred$label <- as.factor(xgb.pred$label)
```

6. Predict test set outcomes

Check

prediction column에는 모델이 예측한 class가, label column에는 실제 class가 담기게 됩니다.

```
> head(xgb.pred)
     setosa versicolor  virginica prediction  label
1 0.8746351 0.06486092 0.06050399     setosa setosa
2 0.8746351 0.06486092 0.06050399     setosa setosa
3 0.8740377 0.06549962 0.06046267     setosa setosa
4 0.8746351 0.06486092 0.06050399     setosa setosa
5 0.8746351 0.06486092 0.06050399     setosa setosa
6 0.8746351 0.06486092 0.06050399     setosa setosa
```

마지막으로 caret library 내에 confusionMatrix 함수를 이용하여 Accuracy를 계산해보면! 0.8667이 나오게 됩니다~

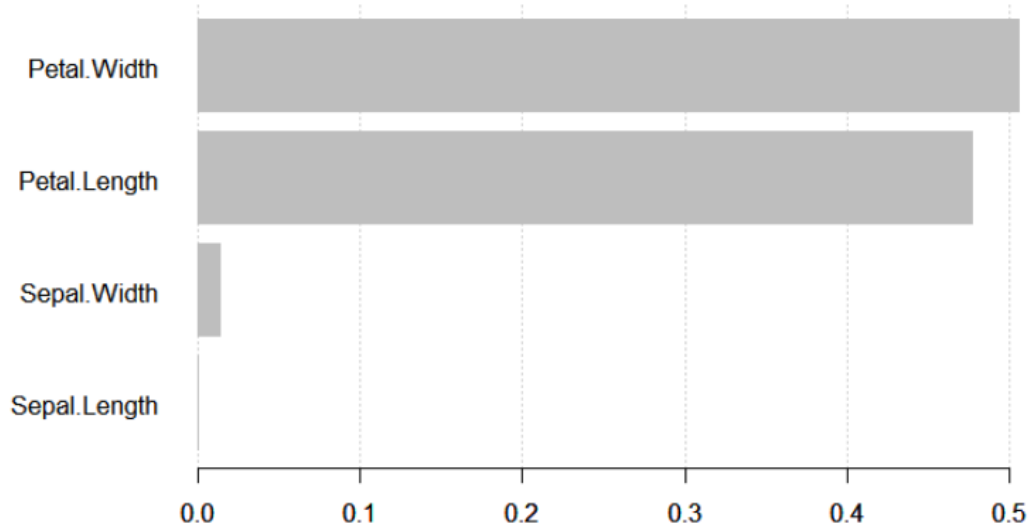
```
> confusionMatrix(xgb.pred$prediction, xgb.pred$label)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         14          0         0
  versicolor      0         13         5
  virginica       0          1        12

Overall Statistics

               Accuracy : 0.8667
                 95% CI : (0.7321, 0.9495)
    No Information Rate : 0.3778
    P-Value [Acc > NIR] : 1.688e-11

                  Kappa : 0.8009
```

7. Feature importance check

Check

이 친구도 tree 기반 모델이기 때문에, feature_importance를 뽑을 수 있습니다!!
feature_importance를 뽑아 plotting 해보면!

```
> var_mat <- xgb.importance(feature_names = colnames(test.data), model = xgb.fit)
> xgb.plot.importance(importance_matrix = var_mat)
```



감사합니다