11. 앙상블학습(Ensemble Learning)

우리는 지금까지 많은 종류의 분류기법(classifier)과 회귀기법(regression)을 논의해왔다. 앙상블은 이러한 기법들의 모임이며 이 기법들을 이용하여 예측성능을 높이는 학습을 앙상블학습(Ensemble learning)이라고 한다. 그러므로 앙상블학습은 분석의 종착점이라고 할 수 있으며 대부분 분류기법에 다양하게 사용되므로 분류를 중심으로 설명을 진행하고자 한다. 필요시, 앙상블회귀도 논의할 것이다. 가장 쉬운 앙상블학습은 투표분류기법(voting classifier)이다. 2개 이상의 분류기법들을 동일한 학습데이터에 각각 적합시킨 후 이를 이용해 분류기법별로 새로운 자료에 대한 클래스를 예측하고 자료별로 예측된 클래스 중 빈도가 가장 높은 클래스를 이 새로운 자료의 최종 클래스로 할당하는 방법이다. 이러한 할당 방법을 hard voting이라고 한다. 각 분류기법이 특정 클래스에 속할 확률을 제공한다면 각 분류기법별로 특정관측치가 클래스에 속할 확률을 구한 뒤 그 클래스에 속할 평균 확률을 계산한다. 평균확률이 가장 큰 클래스로 이 특정관측치의 클래스로 할당하는 것을 soft voting이라고 한다. 이러한 앙상블기법은 앙상블에 이용되는 분류기법들이 개념적으로 독립적일 때 효과가 높게 나타난다. 앙상블러닝은 딥러닝을 쓸만큼 자료의 크기가 충분하지 않을 때 딥러닝만큼의 성능을 보이는 머신러닝모형이다. 정밀도가 낮은 수많은 모형을 학습시켜 이를 통합한 정밀도가 높은 메타모형(meta model)이 앙상블모형이다.

11.1 Bagging, Pasting, 그리고 Random forest

앙상블기법은 사실 통계학에서 가장 중요한 개념인 대수의 법칙(law of large number, LLN)에 기초하고 있다. LLN은 독립적인 표본들로부터 계산된 표본 적률은 표본의 수가 증가하면서 모적률로 접근한다는 이론이다. 다시 말해서 표본의 크기가 클수록 표본 평균은 모평균으로 접근하므로 독립적인 분류기법이 많을수록 분류의 정밀도가 높아지게 된다. ~~어러한 목적으로~~ 독립적인 분류기법이 많으면 정밀도가 높아지겠지만 현재까지 개발된 분류기법의 종류가 LLN을 적용할 만큼 많지도 않고 서로 간에 독립적인지 검정할 수 있는 방법 또한 없다. 이러한 이유 때문에 적은 수의 분류기법을 사용하되 학습데이터를 대폭 늘리는 방법을 고려하게 되었다.

학습데이터를 늘리는 방법은 두 가지가 있다. 첫 번째로 원래의 학습데이터를 붓스트랩(bootstrap)하는 방법이다. 즉, 원래의 학습데이터의 크기가 n개라면 이 학습데이터에서 임의로 n개의 표본을 ~~임의로~~ 뽑되 with replacement로 표본을 뽑는 방법

이다. 예를 들어 원래의 학습데이터가 $\{1,2,3,4,5\}$로 구성되어 있을 때 with replacement로 표본을 뽑는다는 것은 한번 뽑힌 표본도 다시 집어넣어 다시 뽑을 수 있기 때문에 새로운 학습데이터로 $\{1,1,4,5,5\}$를 만들 수 있다. 그러므로 with replacement로 표본을 뽑게 되면 동일한 표본을 반복적으로 뽑을 수 있다. 이러한 절차로 M개의 학습데이터를 생성한다. 이 방법을 bagging (bootstrap aggregating)이라고 한다. 두 번째로 원래의 학습데이터로부터 without replacement로 임의의 표본을 뽑아 새로운 학습데이터를 생성하는 방법이 pasting이다. 원래의 학습데이터가 $\{1,2,3,4,5\}$이라고 가정할 때 without replacement로 표본을 뽑아 새로운 학습데이터로 $\{1,3,5\}$을 만드는 방법이다. 그러므로 without replacement는 한번 뽑은 표본은 다시 뽑을 수 없다는 것을 의미한다. 분석에 이용되는 머신러닝이 분류일 때는 hard voting이나 soft voting을 이용하여 클래스를 예측한다.

예를 들어, 2개의 클래스로 목표변수를 분류하고자 할 때, 3가지의 분류기법이 있고 $M=100$으로 100개의 훈련데이터가 생성되었다고 가정하자. 특성변수 $\boldsymbol{x}$를 가진 새로운 관측치(즉, 시험데이터)의 클래스를 예측하는 것이 목적이다. 3가지 분류기법은 각각 100번의 예측클래스를 할당할 것이다. 300번의 할당 중 첫 번째 클래스에의 예측이 많은지 두 번째 클래스로의 예측이 많은지에 따라 새로운 관측치의 범주를 할당하면 hard voting이다. 한편, soft voting은 300번의 두 클래스에 대한 할당확률의 평균을 구해서 높은 평균 확률을 가진 클래스를 새로운 관측치의 클래스로 예측한다. 회귀인 경우, 300개의 목표변수 예측값의 평균을 목표변수 y의 최종예측값으로 한다. 일반적으로 bagging이 pasting보다 우수한 것으로 나타나(Geron, 2017) bagging을 주로 사용한다.

Bagging에서는 붓스트랩의 특성상 학습데이터의 일부가 반복적으로 뽑혀 붓스트랩 학습데이터로 사용되기 때문에 붓스트랩 학습데이터에 포함되지 않은 학습데이터 일부가 존재할 수밖에 없다. 학습데이터에 포함되지 않은 자료(out-of-bag, oob)를 검증데이터로 이용할 수 있다는 점 또한 bagging의 장점이다.

Random forest는 의사결정나무의 bagging 버전이라고 볼 수 있다. 분석 절차를 정리하면 다음과 같다.

<Random forest 분석 절차>

1. 크기가 n이고 d개의 특성변수를 가진 원래의 학습데이터에서 n개의 확률 붓스트랩 표본을 뽑아 새로운 학습데이터를 만든다.
2. 이 새로운 학습데이터를 이용하여 의사결정나무의 각 노드마다 d개의 특성변수 중 임의로 k개의 특성변수를 뽑아, 이 k개의 변수에 대해서 의사결정나무의 일반적인 절차를 따라 의사결정나무를 완성한다.
3. 절차 1-2를 M번 반복한다.

의사결정나무가 분류를 위한 것이라면 새로운 관측치의 특성변수를 M개의 분류나무에 적용하여 voting으로 결정하고, 회귀일 경우 M개의 목표변수 y 예측치의 평균을 최종예측치로 한다. Random forest에서 특성변수를 임의 추출하는 이유는 M개의 의사결정나무간의 상관관계를 최소화하기 위함이다. 이러한 과정을 거치지 않으면, 예를 들어 변별력이 뛰어난 특성변수가 존재할 때 M개의 의사결정나무 거의 대부분이 이 변수를 선택하게 되어 결과적으로 의사결정나무간의 상관관계가 높아지게 된다. 모든 Random forest는 학습데이터와 특성변수의 확률화(randomness) 때문에 과대적합의 가능성이 낮기 때문에 가지치기(prune)가 필요하지 않으며 k는 일반적으로 $k=\sqrt{d}$로 택한다. 그러므로 Random forest의 초모수는 절차 3에 있는 M이다. M이 클수록 대수의 법칙에 의해 Random forest의 성능이 좋아지므로 컴퓨터의 계산능력을 고려하여 가능한 한 큰 M을 선택하는 것이 좋다. 의사결정나무는 random forest뿐만 아니라 대부분의 앙상블러닝에 사용된다. 앙상블러닝 중 우수한 성능을 보여주고 있는 boosting을 설명하기 이전에 의사결정나무를 앙상블러닝의 기반모형으로 쓰는 이유를 우선 살펴보기로 한다.

11.2 앙상블러닝을 위한 통계적 머신러닝의 특성

머신러닝에서 분석 전에 실시하는 사전자료정리(preprocessing) 과정은 올바른 분석 결과와 성능향상에 매우 중요한 역할을 한다. 적절한 러닝알고리즘의 선택을 위해 다음과 같이 6가지 사항을 고려하여야 한다.

1. 자료의 타입에 민감한가?

실제 문제에서는 자료의 타입이 실수, 명명형(nominal), 범주형(categorical) 등 복합적 형태로 나타난다. 이럴 경우, 머신러닝은 자료의 타입에 덜 민감하여야 한다.

2. 결측(missing) 자료에 민감한가?

결측은 관측치 모두가 결측인 경우(unit missing)와 변수 중 일부가 결측인 경우(item missing)가 있다. 변수 중 일부가 결측인 경우가 대부분이며 이러한 관측치를 불완전 관측치(incomplete observation)이라고 한다. 결측자료가 많을 때 머신러닝이 결측자료에 민감하면 적절한 분석도구가 될 수 없을 것이다.

3. 자료의 이상치에 민감한가?

머신러닝에서 모수추정을 할 때 이상치(outlier)의 영향이 덜 민감한 머신러닝 분석방법을 채택하여야 한다. 손실함수로 제곱오차를 사용하는 머신러닝 알고리즘은 이상치에 매우 민감하므로 이상치를 제거하고 사용하여야 한다.

4. 자료의 표준화가 필요한가?

거리의 개념을 사용하는 머신러닝 손실함수에서는 특성변수 자료의 측도(scale) 문제를 없애기 위해서 자료의 표준화가 필요한 경우가 있으며 특히, 딥러닝에서는 필수적이다. 자료의 변환은 원래 자료가 가지고 있는 정보의 손실 또는 왜곡이 있을 수밖에 없으며, 특히 분석결과의 해석에 문제를 일으킬 수 있다.

5. 해석의 용이성

특성변수의 변화가 목적변수에 영향을 주는 정도를 측정하기 위해서는 머신러닝모형의 해석이 용이해야 한다. 통계학이나 계량 경제학에서 선형모형이 유용하게 사용되는 이유는 해석의 용이함 때문이다.

6. 성능

학습데이터(tranining data)에 모형을 적합시키고 시험데이터(test data)에 적용하여 평가한 머신러닝의 성능은 가장 중요하면서 최종 점검사항이며, 다른 머신러닝과의 비교를 위한 수단이다.

<표 11.2>는 앞에서 논의한 분석자료 준비과정(preprocessing)에서 고려해야 할 6가지 사항에 대한 개별 머신러닝 분석방법의 특성들을 보여주고 있다. 의사결정나무는 예측 성능(predictive performance)을 포함하여 분석 자료의 준비 과정이 거의 필요 없다는 것을 알 수 있다.

<표 11.2> Learning 방법의 특성들의 비교

특성	로지스틱	KNN	LDA	SVM	의사결정나무	최소제곱 선형모형	Neural network
자료 type 민감성	상	상	상	상	하	상	상
결측 자료 영향	상	중	상	상	하	상	상
이상치 민감성	상	하	상	상	하	상	상
표준화	선택	선택	선택	선택	불필요	불필요	필요
해석의 용이성	용이	난해	난해	난해	용이	용이	매우난해
성능	중간	중간	중간	중간	중간	중간	높음

그러므로, 주어진 자료에 머신러닝을 적용하기 위해 필수적인 분석자료 준비과정이 거의 필요없는 “off-the-shelf”(기성품) 분석 방법은 의사결정나무이다. 기울기부스팅(Gradient Boosted Method, GBM)과 Extreme Gradient Boosting (XGboost) 등 앙상블러닝은 기초모형으로 의사결정나무를 사용하며 여러 개의 의사결정나무를 순차적으로 결합하여 성능을 혁신적으로 높은 머신러닝기법이다.

11.3 아다부스트(AdaBoost)

Boosting의 뜻은 개선함(to improve)이다. 오직 노드가 1개인 의사결정나무를 고려하여 보자. 즉, 여러 개의 특성 변수 중 하나를 택해 2개의 구간으로 나누어 분류하는 의사결정나무를 가정해보자. 이 의사결정나무는 ~~너무나 간단하지만~~ 2개의 범주만 있는 간단한 경우라도, 임의분류(random classification)보다 약간 좋을 가능성이 높다. 이처럼 분류 성능이 임의분류보다 약간 좋은 머신러닝을 weak learner라고 한다. 그러나 이러한 weak learner도 다음과 같이 반복적으로 사용되면 지금까지 논의한 어떤 단일 분류기법보다 훨씬 성능이 좋은 머신러닝이 가능해진다.

먼저 weak learner를 이용해 분류를 한 후, n개의 학습데이터 중 분류가 올바르게 된 학습데이터의 가중치는 줄이고 오분류된 학습데이터에 대해서는 가중치를 높여서 기존 weak learner+새로운 weak learner로 분류를 한다. 이렇게 더해진 두 개의 weak learner들은 오 분류된 학습데이터에 상대적으로 높은 가중치가 부여됐으므로 오 분류된 학습데이터의 일부가 제대로 분류할 것이다. 오 분류된 학습데이터에 가중치를 높이고 세 번째 weak learner를 추가하여 오 분류된 학습데이터의 일부를 제대로 분류한다. 이러한 절차를 M번 반복하여 M개의 weak learner를 합하여 최종 분

류모형으로 사용한다. 이와 같이 추가적(additive)으로 개선되는 학습방법을 boosting이라고 한다. Bagging모형의 대표격인 random forest 모형과 boosting의 근본적인 차이는 다음과 같다. random forest에서는 M개의 독립적인 모형을 서로 다른 M개의 boostrap 학습데이터에 적용하여 각 예측치의 평균을 산출하는 반면, boosting에서는 한 개의 학습데이터에 순차적(sequential)으로 weak learner를 적용하여 최종 모형을 구성한다.

좀 더 구체적인 예를 살펴보기 위해 아다부스트(adaptive boosting)를 고려하여 보자. 아다부스트는 목적변수가 $y_i \in \{-1,1\}$인 경우에 적용된다. 즉, 두 개의 클래스가 있을 때 분류를 목적으로 하고 있다. 우리는 이러한 이항분류가 다항분류로 일반화하여 적용할 수 있는 one-versus-rest기법을 여러 번 논의한 바 있다. 다음의 간단한 예제를 고려하여 보자.

관측치	1	2	3	4	5	6	7	8
x	5	10	15	20	25	30	35	40
y	-1	-1	1	1	1	-1	-1	1
가중치	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

<표 11.1> adaboost를 위한 학습데이터

<표 11.1>은 특성변수가 오직 하나인 경우를 보여주고 있다. 제 6장에서 논의했듯이, 의사결정나무의 노드를 결정하기 위해 Gini지수, cross-entropy 등의 불순도 지수를 이용하여 정보획득(information gain, IG)이 가장 큰 노드는 $x = 12.5$라고 가정하자, 물론 두 개 이상의 특성변수가 있다면 각 변수별로 가장 큰 IG를 구해 이 중 가장 큰 IG를 가진 변수의 노드가 의사결정나무의 노드가 된다. 아다부스트는 일반적으로 노드가 하나인 decision tree인 소위, tree stump를 사용한다. $x < 12.5$이면 $f_1 = -1$로 분류하고 $x \geq 12.5$이면 $f_1 = 1$로 분류하는 tree stump이다. 그러므로 $f_1 = 2I[x \geq 12.5] - 1$이며 6, 7번째 관측치에 오분류가 발생하게 된다. 그러므로 총 오차는 $e_1 = \frac{2}{8} = 0.25$ 가 된다. 이 첫 번째 tree stump의 정보량(이를 amount of say라고 한다)은 $as_1 = \frac{1}{2} ln(\frac{1-e_1}{e_1}) = 0.55$가 된다.

실제 분류는 $\hat{y} = sign(as_1 \cdot f_1) = sign(0.55 \cdot (2I[x \geq 12.5] - 1))$으로 하여 6, 7번째 관측

치가 +부호를 가지고 있어 오분류가 발생한다. 이 as_1을 이용하여, 오분류가 발생한 관측치의 새로운 가중치는 $e^{0.55}=1.733$이 되고 오분류가 발생하지 않은 관측치의 새로운 가중치는 $e^{-0.55}=0.577$이 된다.

관측치	1	2	3	4	5	6	7	8
x	5	10	15	20	25	30	35	40
y	-1	-1	1	1	1	-1	-1	1
가중치	0.577	0.577	0.577	0.577	0.577	1.733	1.733	0.577
조정가중치	0.083	0.083	0.083	0.083	0.083	0.251	0.251	0.083

<표 11.2> 아다부스트를 위한 조정된 가중치의 계산

<표 11.2>는 오분류가 발생한 관측치와 잘 분류된 관측치의 표본 가중치를 계산한 것(아래에서 새로운 가중치의 계산에 대해 이론적 논의를 할 것이다)과 가중치의 합이 1이 되도록 만든 조정가중치를 보여주고 있다. 이제 다음 tree stump를 구성하기 위한 데이터를 만들어 보자. 0~1사이의 값을 균등분포(uniform distribution)로부터 8개의 표본을 임의 추출하여 새로운 표본을 구성한다. 예를 들어, 0.1이 추출되면 2번째 관측치가 표본에 포함되고, 0.415보다 크고 0.666보다 작은 값을 균등분포로부터 추출하면 6번째 관측치가 표본이 된다.

관측치	1	2	3	4	5	6	7	8
x	5	10	20	30	30	35	35	40
y	-1	-1	1	-1	-1	-1	-1	1
가중치	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

<표 11.3> 두 번째 tree stump를 위한 데이터셋

<표 11.3>은 균등분포로부터 구성된 자료를 보여주고 있다. <표 11.1>의 최초의 표본에서 오분류된 6, 7번째 자료가 2번 반복해서 표본에 포함된 것을 볼 수 있다. 즉, 오분류된 자료가 두 번째 tree stump를 구성하는데 가중치가 높아졌다는 것을 의미한다. 두 번째 tree stump는 첫 번째 stump를 구성하는 절차를 반복하므로 우선 각 자료의 가중치를 1/8로 원위치하고 불순도 지수에 따라, tree stump의 노드를 결정한다. 이 노드가 $x=25$이면 $f_2=2I[x\le 25]-1$이 되고 3개의 관측치가 오분류되었으므로 총 오차 $e_2=0.375$이며 $as_2=0.255$가 된다. 이를 이용하여 기존의 tree stump f_1에 f_2를 추가하여 다음의 y의 추정함수를 구한다.

$$\hat{y}=sign(as_1\cdot f_1+as_2\cdot f_2)=sign(0.55\cdot(2I[x\ge 12.5]-1)+0.255\cdot(2I[x\le 25]-1))$$

이 새로운 추정함수를 이용하여 전체자료를 재분류하고 총오차와 amount of say를

구해 새로운 표본가중치로 표본을 구성하여 as_3와 f_3를 산출한다. 이러한 반복을 M번 했다면 최종 예측함수는

$$\hat{y} = sign(\sum_{i=1}^{M} as_i \cdot f_i)$$

가 된다.

아다부스트의 이론적 배경을 살펴보기 위해 우선 <표 11.4>와 같이 네 가지의 대표적인 손실함수를 정의하고 이 손실함수를 최소로 하는(즉, 미분 값을 0으로 만드는) y 또는 $\tilde{y}$의 추정량 h의 기댓값을 구하여 정리하였다. <표 11.1>에서 지수손실함수와 로짓손실함수는 이항분류를 위한 비용함수이며 $\tilde{y}_i$={−1,1}, y_i={0, 1}, 그리고 $\pi_i = \Pr(\tilde{y}=1)$ 또는 $\Pr(y=1)$이다. 제곱오차와 절대오차는 일반적으로 y_i가 실수인 회귀러닝이며 미분에 $\boldsymbol{x}_i$가 주어진 조건부 기대치에 의해 h^*를 쉽게 구할 수 있다.

<표 11.4> 네 가지 손실함수에 대한 최소 h의 기대값

손실함수명	손실	미분	h^*
제곱오차(Squared error)	$(y_i - h(\boldsymbol{x}_i))^2$	$y_i - h(\boldsymbol{x}_i)$	$E(y\|\boldsymbol{x}_i)$
절대오차(Absolute error)	$\|y_i - h(\boldsymbol{x}_i)\|$	$\text{sign}(y_i - h(\boldsymbol{x}_i))$	$\text{median}(y\|\boldsymbol{x}_i)$
지수(Exponential)	$\exp(-\tilde{y}_i h(\boldsymbol{x}))$	$-\tilde{y}_i exp(-\tilde{y}_i h(\boldsymbol{x}_i))$	$\frac{1}{2} log \frac{\pi_i}{1-\pi_i}$
로짓(Logit)	$\log(1+\exp(-y_i h(\boldsymbol{x}_i)))$	$y_i - \pi_i$	$\frac{1}{2} log \frac{\pi_i}{1-\pi_i}$

지수손실함수 $\exp(-\tilde{y}_i h(\boldsymbol{x}_i))$는 $\tilde{y}_i$가 -1 또는 1로 범주를 나타내므로 $\boldsymbol{x}_i$를 기반으로 예측치(classifier) $h(\boldsymbol{x}_i)$가 $\tilde{y}_i$와 같은 부호를 가지면서 동시에 $f(\boldsymbol{x}_i)$의 절대값이 커지게 되면 손실함수가 지수적으로 감소하게 된다. 한편 $h(\boldsymbol{x}_i)$와 $\tilde{y}_i$의 부호가 다르면 $-\tilde{y}_i h(\boldsymbol{x}_i)$가 양의 값이 되어 $h(\boldsymbol{x}_i)$의 절대값이 커질수록 손실함수가 지수적으로 증가함을 알 수 있다. 로짓손실함수도 지수손실함수와 유사하게 행태를 보이지만 로짓손실함수의 증감이 지수손실함수보다는 좀 더 선형에 가깝다고 할 수 있다.

지수손실함수 대한 최적 추정치 h^*를 도출해보면 다음과 같다.

$$l=\exp(-\tilde{y}h)\text{이므로 } \frac{\partial L}{\partial h}=-\tilde{y}\exp(-\tilde{y}h)$$

가 된다. h는 오직 $\boldsymbol{x}_i$의 함수이므로

$$E(-\tilde{y}exp(-\tilde{y}h)|\boldsymbol{x}_i) \ =-\exp(-h)P(\tilde{y}=1|\boldsymbol{x}_i)+\exp(h)P(\tilde{y}=-1|\boldsymbol{x}_i)$$

이다. 그러므로 $E(\frac{\partial L}{\partial h})=0$인 h의 해는

$$h^*=\frac{1}{2}log\frac{P(\tilde{y}=1|\boldsymbol{x}_i)}{P(\tilde{y}=-1|\boldsymbol{x}_i)}$$

가 된다. 이 식에 의해 $P(\tilde{y}=1|\boldsymbol{x}_i)>0.5$이면 즉 $h^*(\boldsymbol{x}_i)>0$이면 특성변수 $\boldsymbol{x}_i$에 대응되는 목표변수 y_i의 범주는 1로 예측하게 된다.

제 4장 로지스틱회귀에서

$$\pi_i=P(\tilde{y}=1|\boldsymbol{x}_i)=\frac{1}{1+e^{-2h(\boldsymbol{x}_i)}}$$

으로 정의하였다. 여기에서 $h(\boldsymbol{x}_i)=\frac{1}{2}\boldsymbol{w}^T\boldsymbol{x}_i$이다. $y=(\tilde{y}+1)/2$로 변환하면 $\tilde{y}=1$일 때 $y=1$이 되고 $\tilde{y}=-1$일 때 $y=0$이 된다. 로지스틱회귀에서는 y가 베르눌리(bernoulli) 분포를 따른다고 가정하므로 y의 로그우도함수는

$$L=y\log\pi_i+(1-y)\log(1-\pi_i) \qquad (11.1)$$

$\pi_i=(1+e^{-2h(\boldsymbol{x}_i)})^{-1}$이므로 (11.1)에 대입하여 정리하면 $L=-\log(1+e^{-2yh(\boldsymbol{x}_i)})$가 된다. $-L$이 손실함수이므로

$$l=-L=\log(1+e^{-2yh(\boldsymbol{x}_i)})$$

가 되어 <표 11.4>의 로짓손실함수와 일치함을 알 수 있다. 이와 같은 이유로 로짓(logit)이라 불린다.

아다부스트의 손실함수는 지수손실함수로 다음과 같다.

$$\sum_{i=1}^{n}\exp(-y_i(h_{m-1}(\boldsymbol{x}_i)+\beta f(\boldsymbol{x}_i)))=\sum_{i=1}^{n}w_i^{(m)}\exp(-\beta y_i f(\boldsymbol{x}_i)) \qquad (11.2)$$

가 된다. 여기에서 $y_i\in\{-1,1\}$이고 $w_i^m=\exp(-y_i h_{m-1}(\boldsymbol{x}_i))$이다. $h_{m-1}(\boldsymbol{x}_i)$는 $(m-1)$번

째 단계에서 계산된 y_i의 예측치(즉, $h_{m-1} = sign(\sum_{i=1}^{m-1} as_i \cdot f_i)$)로 $y_i h_{m-1}(\boldsymbol{x}_i)$가 음이면 (즉, $h_{m-1}(\boldsymbol{x}_i)$가 y_i의 범주를 잘못 예측하면) $w_i^{(m)}$가 커지게 된다. 그러므로 전 단계인 $(m-1)$번째 단계에서 예측오차가 큰 학습데이터에 더 큰 가중치를 부여하여 현 단계에서는 큰 가중치를 가진 학습데이터를 제대로 분류하도록 오차를 최소화하는 β와 f를 추정하게 된다.

(11.2)의 손실함수를 재 표현하면 다음과 같다. f는 앞에서 논의한데로 -1 또는 1 값을 가지므로

$$\begin{aligned} l &= \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i f(\boldsymbol{x}_i)) \\ &= \sum_{y_i = f(\boldsymbol{x}_i)} w_i^{(m)} e^{-\beta} + \sum_{y_i \neq f(\boldsymbol{x}_i)} w_i^{(m)} e^{\beta} \\ &= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq f(\boldsymbol{x}_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \end{aligned} \tag{11.3}$$

그러므로 (11.3)의 최소화는 $\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq f(\boldsymbol{x}_i))$의 최소화이다. w_i^m이 주어졌으므로 $\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq f(\boldsymbol{x}_i))$를 최소화하는 tree stump $f(\boldsymbol{x}_i)$는 불순도지수를 이용하여 구한다. (11.3)에서 β에 대해 미분을 취한 후 0으로 놓고 (11.3)을 최소로 하는 β를 구하면

$$\beta_m = \frac{1}{2} log \frac{1-e_m}{e_m}, \; e_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq f(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i^{(m)}} \tag{11.4}$$

이다. (11.4)에서 $w_i^{(m)}$은 표본 가중치로 이 표준 가중치로 학습표본을 재구성하게 되면 앞에서 살펴본 바와 같이 $w_i^{(m)} = \frac{1}{n}$이 된다. 그러므로 (11.4)에서 e_m은 잘못 분류한 비율이 되고 분류함수는 $\beta_m \cdot f(\boldsymbol{x}_i)$가 된다. 즉, <표 11.2> 예제에서 $\beta_m \cdot f(\boldsymbol{x}_i) = as_1 \cdot f_1$이고 $\hat{y}_i$는 -1 또는 1이 되므로 $\hat{y}_i = sign(as_1 \cdot f_1)$이 된다. <표 11-2>에서 새로운 가중치는

$w_i^{m+1} = \exp(-y_i h_m(\boldsymbol{x}_i))$이고 $h_m(\boldsymbol{x}_i) = \beta_m \cdot f(\boldsymbol{x}_i) = as_1 \cdot f_1$

이므로 $w_i^{m+1} = \exp(-y_i \cdot as_1 \cdot f_1)$이 된다. 즉, $y_i \cdot f_1$이 다른 부호(제대로 분류)이면 $w_i^{m+1} = \exp(as_1)$이 되고 같은 부호이면 $w_i^{m+1} = \exp(-as_1)$이 된다. 끝으로 w_i^{m+1}은 $\frac{w_i^{m+1}}{\sum_{i=1}^{n} w_i^{m+1}}$으로 재 조정된다.

11.4 기울기부스팅(Gradient Boosting Method)

의사결정나무의 전형적인 형태는 $b(\boldsymbol{x}, \boldsymbol{\gamma}) = \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x} \in R_j)$이다. 여기에서 J는 총 노드 수이고 결정해야 할 모수는 $\{\gamma_j, R_j\}$, $j = 1,2,...,J$ 이다. 여기에서 R_j는 노드 γ_j에 의해 만들어진 영역이다. 의사결정나무에서는 SSE(sum of squared errors), SAE(sum of absolute errors), 또는 불순도측도(Gini 계수, cross-entropy 등)를 이용하여 R_j를 top-down 형태로 결정하고 회귀나무인 경우 R_j에 포함된 y_i들의 평균을 γ_j의 추정치로, 분류나무인 경우 R_j 안에 가장 많은 범주를 γ_j의 추정치로 사용한다. 그러나, 회귀나무에서 주로 사용하는 제곱오차손실함수나 절대오차손실함수 등을 사용하여

$$\sum_{i=1}^{n} l(y_i, \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x}_i \in R_j)) \tag{11.5}$$

를 최소로 하는 γ_j, R_j를 구하는 것은 불가능에 가까운 계산을 요구하게 된다. (11.5)에서 의사결정나무 대신 일반적인 함수 $f(\boldsymbol{x}_i)$를 고려해보자. 그리고 손실함수 l이 미분 가능하다고 가정하자. 즉 (11.5) 대신

$$\sum_{i=1}^{n} l(y_i, f(\boldsymbol{x}_i)) \tag{11.6}$$

를 최소화하는 $f(\boldsymbol{x}_i)$를 구하고자 한다. (11.6)을 최소화하기 위해 기울기하강법(gradient decent)을 적용하면

$$\begin{aligned} f_m(\boldsymbol{x}_i) &= f_{m-1}(\boldsymbol{x}_i) - \eta_m \frac{\partial l(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)} \Big|_{f(\boldsymbol{x}_i) = f_{m-1}(\boldsymbol{x}_i)} \\ &= f_{m-1}(\boldsymbol{x}_i) + \eta_m g_{im} \end{aligned} \tag{11.7}$$

이 되고, 여기에서 $i=1,2,...,n$이고 $g_{im} = -\frac{\partial l(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}|_{f(\boldsymbol{x}_i)=f_{m-1}(\boldsymbol{x}_i)}$이다.

(11.6)의 $f(\boldsymbol{x}_i)$에 (11.7)의 $f_m(\boldsymbol{x}_i)$를 대입하면

$$\sum_{i=1}^{n} l(y_i, f_{m-1}(\boldsymbol{x}_i)+\eta_m g_{im}) \tag{11.8}$$

이 되며 이러한 관계를 이용하여 (11.5)를 최소화하는 γ_j와 R_j를 다음과 같은 두 단계로 구현하여 보자. 먼저 (11.7)의 g_{im}을 구한 후,

$$\sum_{i=1}^{n}(g_{im} - \sum_{j=1}^{J}\gamma_{jm} I(\boldsymbol{x}_i \in R_{jm}))^2 \tag{11.9}$$

을 최소화 하는 r_{jm}을 산출한다. 여기에서 R_{jm}은 의사결정나무의 불순도 측도를 이용하여 결정되므로 (11.9)는 $(J-1)$개의 더미(dummy) 변수를 가진 일반적인 회귀모형의 손실함수이며 γ_{jm}은 최소제곱 추정치가 된다. g_{im}이 기울기이기 때문에 기울기부스팅(gradient boosting)이름으로 불러진다.

η_m은 적절한 손실함수 l에 의해 정의된 (11.8)에 의해

$$\sum_{i=1}^{n} l(y_i, f_{m-1}(\boldsymbol{x}_i)+\eta_m \sum_{j=1}^{J}\hat{\gamma}_{jm} I(\boldsymbol{x}_i \in R_{jm}))$$

을 최소화 하는 η_m을 구한다. 여기에서 $\hat{\gamma}_{jm}$은 (11.9)에 의한 최소제곱 추정치이다. (11.8)에서 가장 많이 사용되는 손실함수를 정리하면 다음과 같다.

도구	손실	$-\frac{\partial l}{\partial f(\boldsymbol{x}_i)}$	비고
회귀	$\sum_{i=1}^{n}(y_i-(f\boldsymbol{x}_i))^2$	$y_i - f(\boldsymbol{x}_i)$	제곱오차
	$\sum_{i=1}^{n}\|y_i - f(\boldsymbol{x}_i)\|$	$sign(y_i - f(\boldsymbol{x}_i))$	절대오차
	$(y_i-f(\boldsymbol{x}_i))^2$ if $\|y_i-f(\boldsymbol{x}_i)\|<\delta$ $2\delta\|y_i-f(\boldsymbol{x}_i)\|-\delta^2$ if $\|y_i-f(\boldsymbol{x}_i)\|\geq\delta$.	$y_i-f(\boldsymbol{x}_i)$ if $\|y_i-f(\boldsymbol{x}_i)\|<\delta$ $\delta sign\|y_i-f(\boldsymbol{x}_i)\|$ if $\|y_i-f(\boldsymbol{x}_i)\|\geq\delta$	후버손실(Huber loss)
분류	$\log(1+\exp(-y_i f(\boldsymbol{x}_i)))$	$y_i-\pi_i^* = y_i - \frac{1}{1+e^{-2f(\boldsymbol{x}_i)}}$	deviance loss=음의 다항분포 우도함수

* $\pi_i = P(y_i=1) = \frac{1}{1+e^{-2f(\boldsymbol{x}_i)}}$

<표 11.5> 손실함수와 기울기(gradient)

<표 11.5>를 이용하여 기울기나무부스팅(Gradient tree Boosting) 알고리즘을 정리하면 다음과 같다.

1. 초기치 $\sum_{i=1}^{n} l(y_i, \gamma)$를 최소화하는 $f_0(\boldsymbol{x}_i) = \gamma$

2. $m = 1$부터 M까지

   (a) $g_{im} = -[\frac{\partial l(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}]_{f=f_{m-1}}$, $i = 1,2,...,n$

   (b) $\sum_{i=1}^{n}(-g_{im} - \sum_{j=1}^{J}\gamma_{jm} I(\boldsymbol{x}_i \in R_{jm}))^2$을 최소화하는 $\hat{\gamma}_{jm}$을 구한다. 여기에서 R_{im}은 의사결정나무의 불순도측도에 의해 결정된 J개의 서로 배반인 영역이다.

   (c) $\sum_{i=1}^{n} l(y_i, f_{m-1}(\boldsymbol{x}_i) + \eta_m \sum_{j=1}^{J}\hat{\gamma}_{jm} I(\boldsymbol{x}_i \in R_{jm}))$을 최소화하는 η_m을 구한다.

   (d) $f_m(\boldsymbol{x}_i) = f_{m-1}(\boldsymbol{x}_i) + \hat{\eta}_m \sum_{j=1}^{J}\hat{\gamma}_{jm} I(\boldsymbol{x}_i \in R_{jm})$으로 업데이트한다.

3. $\widehat{f}(\boldsymbol{x}_i) = f_M(\boldsymbol{x}_i)$이다.

GBM알고리즘에 의해 GBM의 초모수는 총 추가모형의 수인 M과 의사결정나무의 총 영역수인 J 이다. 알고리즘에서는 J를 고정했지만, J_m으로 m 번째 추가모형에 의존하도록 설정해도 된다.

y가 분류이고 y의 범주수가 K일 때, 분류나무부스팅은 2-(a) ~ 2-(d)를 K번 반복하여 $\widehat{f^k}(\boldsymbol{x}_i), k = 1,...,K$를 구한다. 그러므로 $f_M(\boldsymbol{x}_i)$는 클레스 숫자만큼 K개가 산출되어

$$p_k(\boldsymbol{x}_i) = \frac{e^{\widehat{f^k}(\boldsymbol{x}_i)}}{\sum_{k=1}^{K} e^{\widehat{f^k}(\boldsymbol{x}_i)}}$$

에 의해 i번째 학습데이터가 범주 k에 속할 확률을 예측할 수 있다. GBM 알고리즘에 사용된 의사결정나무의 크기인 J는 일반적으로 $4 \le J \le 8$의 값을 갖는다.

11.4.1 GBM의 이해

GBM의 알고리즘을 이해하기 위해 y가 연속형이고 손실함수가 SSE(sum of squared error)인 경우를 고려하여 보자. 가장 간단한 경우로 특성변수 x는 1차원이고 weak learner로는 tree stump를 가정하자. 물론 실제 문제에서는 특성변수가 2개 이상이며 weak leaner는 깊이가 2~3인 의사결정나무를 일반적으로 사용한다.

x	1	2	3	4	5
y	5	12	14	20	39

<표 11.6> GBM의 이해를 위한 간단한 자료

<표 11.5>에 의해 $g_{im} = y_i - f(x_i)$이다. GBM알고리즘에 의해 SSE를 최소화하는 $f_0(x_i) = \bar{y}$이므로 $g_{i1} = y_i - \bar{y}$가 된다. 이 g_{i1}은 y_i의 추정치인 $\bar{y}$를 빼서 산출된 잔차로 해석할 수 있다.

<표 11.6>의 자료에서 $\bar{y}$= 14.6이 된다. 이 잔차에 대해 tree stump를 적용한다. 즉, 두 개의 영역 각각에서 계산한 SSE의 합이 최소가 되는 노드를 결정하고 GBM 알고리즘 2-(c)에 의해 이 잔차의 SSE를 최소로 하는 추정치는 각 영역의 평균이다. <표 11.7>의 $\eta_1\gamma_1$이 이에 해당하며 노드를 $x = 4.5$으로 했을 때 $x < 4.5$ 영역에 속한 y의 평균은 -5.3이고 $x \geq 4.5$ 영역에 속한 y의 평균은 21이다. GBM 알고리즘 2-(d)에 의해 y의 추정치는 $f_1 = f_0 + \eta_1\gamma_1$으로 update된다. 다시 잔차 $y - f_1$을 구하고 이 잔차에 tree stump를 적용한다. <표 11.7>에서 노드는 $x = 2.5$이고 두 영역의 평균은 각각 -4.2와 2.9(<표 11.7>의 $\eta_2\gamma_2$)로 계산되었다. $f_2 = f_1 + \eta_2\gamma_2$로 update되고 잔차 $y - f_2$를 계산한다. 이러한 절차를 잔차가 충분하게 작아질 때까지 반복한다. 그러므로 손실함수가 SSE일 때의 GBM은 잔차를 순차적으로 줄이는 머신러닝 기법이라고 해석할 수 있다.

x	y	$f_0 = \bar{y}$	$y - f_0$	$\eta_1\gamma_1$	f_1	$y - f_1$	$\eta_2\gamma_2$	f_2	$y - f_2$
1	5	18	-13	-5.3	12.7	-7.7	-4.2	8.5	-3.5
2	12	18	-6	-5.3	12.7	-0.7	-4.2	8.5	3.5
3	14	18	-4	-5.3	12.7	1.3	2.9	15.6	-1.6
4	20	18	2	-5.3	12.7	7.3	2.9	15.6	4.4
5	39	18	21	21	39	0	2.9	41.9	-2.9

<표 11.7>손실함수가 SSE일 때의 GBM

이제 손실함수가 <표 11.5>의 오차의 절대값(SAE) $\sum_{i=1}^{n}|y_i - f(\boldsymbol{x}_i)|$이면 이 오차의 절대값을 최소로 하는 $f_0 = median(y_i)$이므로(<표 11.4>에 의해) <표 11.6>데이터에서

$f_0 = 14$가 된다. 오차의 절대값의 미분은 <표 11.5>와 같이 $sign(y_i - f(\boldsymbol{x}_i))$이므로 $sign(y_i - f_0) = \{-1, -1, 0, 1, 1\}$이 된다. 이 자료를 대상으로 GBM 알고리즘 2-(b)에 의해 tree stump를 적용하면 최소의 SSE를 가지는 노드는 $x = 2.5$ 또는 $x = 3.5$가 된다. 만약 $x = 2.5$를 선택하면 GBM 알고리즘 2-(c)를 적용하여 두 영역의 잔차 $y_i - f_0$의 SAE를 최소로 하는 추정치는 각 영역의 중위수이므로 <표 11.8>과 같이 $\eta_1\gamma_1$이 구해진다. $f_1 = f_0 + \eta_1\gamma_1$이므로 $sign(y_i - f_{1i}) = \{-1, +1, -1, 0, +1\}$이 된다. 이 값에 대해 최소의 SSE를 가진 노드는 $x = 3.5$이며 두 영역의 중위수는 각각 -3.5와 9.5이다. 이 값이 <표 11.8>의 $\eta_2\gamma_2$이다. $f_2 = f_1 + \eta_2\gamma_2$이므로 이를 이용하여 $sign(y_i - f_{2i})$와 잔차 $y_i - f_{2i}$를 산출하고 잔차가 충분히 작아질 때까지 이를 반복한다. SSE와 비교하여 손실함수로 SAE를 사용하면, 의사결정나무의 노드는 $sign(y_i - f_m)$의 SSE에 의해 결정되고 추정치 $\eta_m\gamma_m$은 각 영역의 중위수가 된다. 중위수를 사용하기 때문에 y의 이상치에 민감하지 않은 GBM 기법이라고 할 수 있다. 그러나, 손실함수에 관계없이 y가 연속형인 회귀 목적함수이면 GBM은 잔차를 계속해서 줄여나가는 기법이라고 할 수 있다.

x	y	f_0	$sign(y - f_0)$	$\eta_1\gamma_1$	f_1	$y - f_1$	$\eta_2\gamma_2$	f_2	$y - f_2$
1	5	14	-1	-5.5	8.5	-3.5	-3.5	5	0
2	12	14	-1	-5.5	8.5	3.5	-3.5	5	7
3	14	14	0	6	20	-6	-3.5	16.5	-1.5
4	20	14	1	6	20	0	9.5	29.5	-9.5
5	39	14	1	6	20	19	9.5	29.5	9.5

<표 11.8> 손실함수가 SAE일 때의 GBM

끝으로 y변수가 클래스일 때, 즉 분류가 목적일 때의 GBM을 고려하여 보자. 클래스의 수가 K개일 때, 하나-나머지(one-versus-rest)기법에 따라 첫 번째 클래스 대 나머지 클래스, 두 번째 클래스 대 나머지 클래스, ..., K 번째 클래스 대 나머지 클래스로 차례로 적용하여 분류기 $f_1, f_2, \cdots, f_K$를 각각 산출한다.

f_1을 산출하는 방법을 살펴보자. 우선 i 번째 데이터가 첫 번째 클래스에 속하면 $y_i = 1$, 그렇지 않으면 $y_i = 0$값을 할당한다. 이 후의 절차는 SSE 손실함수를 사용한 GBM과 동일하다. 좀 더 자세히 살펴보면, <표 11.5>에 의해 로짓 손실함수(즉, binary cross entropy)의 음의 미분 값, 즉 g_i는 $y_i - \pi_i$이므로 f_0는 y_i의 평균(총 학습데이터에서 첫 번째 클래스의 비율)이므로 잔차는 $y_i - f_0$가 되고 이 잔차들의 SSE

를 최소화하는 낮은 깊이의 의사결정나무를 결정하고 각 영역별로 $y_i - f_0$의 평균을 구한다. 이 값이 $\eta_1\gamma_1$이 되므로 $f_1 = \bar{y} + \eta_1\gamma_1$이고 다시 잔차 $y_i - f_1$을 이용하여 의사결정나무를 선정하고 영역별로 잔차의 평균인 $\eta_2\gamma_2$를 구한다. 이 절차를 잔차가 충분하게 작아질 때까지 반복한다. 그러므로 y_i를 0 또는 1로 바꾼 것을 제외하고 SSE를 이용한 GBM과 완전하게 일치한다는 것을 확인할 수 있다. 동일한 방법으로 $f_2, f_3, \cdots, f_K$를 구한다. 이 $f_1, f_2, \cdots, f_K$는 모두 의사결정나무의 복합함수이며 특성변수만의 함수이므로, 특성함수를 이 K의 함수에 모두 대입하여 이 중 가장 큰 값을 갖는 클래스를 이 특성함수의 클래스로 할당한다.

11.4.2 GBM의 해석

의사결정나무는 순차적으로 특성변수를 택해 특성변수값을 2개의 영역으로 나누기 때문에 해석이 쉽고 명료하다. 그러나 덧셈(additive)방식의 부스팅에 의한 의사결정나무는 수많은 의사결정로 구성되어 있어 별도의 해석 도구가 필요하다. 이에 해당하는 것에는 특성변수가 목적변수를 설명하는 중요도(importance)와 부분의존플롯(partial dependence plot)이 있다.

먼저, GBM이 회귀로 사용될 경우, 매 나무회귀에서 관심 특성변수가 2개의 영역으로 나누는데 사용된 결과로 줄어든 오차제곱합(SSE)을 구한다. GBM에서는 나무회귀를 M번 반복하므로 줄어든 오차제곱합의 평균을 구할 수 있고 특성변수 별로 줄어든 오차제곱합의 평균을 구할 수 있다. 특성변수별 줄어든 오차제곱합의 평균을 줄어든 전체 오차제곱합 평균으로 나누면 각 특성변수의 중요도를 구할 수 있다. 분류인 경우, 오차제곱합 대신 6.2절의 information gain을 이용하여 회귀와 동일한 절차에 따라 상대 중요도를 구하게 된다.

부분의존플롯은 선형회귀모형에서 회귀계수 β와 같은 역할을 한다. 예를 들어, $y = \beta_1 X_1 + \beta_2 X_2 + \epsilon$ 모형에서 β_1은 X_2의 효과를 제거한 후 X_1이 한 단위 증가하면 y는 β_1 단위만큼 증가한다는 의미를 갖는다. GBM에서의 부분의존플롯은 비교적 용이하다. GBM에서 최종적으로 구한 $f_M(\boldsymbol{x}_i)$에 대해 관심을 가지고 있는 특성변수의 다양한 값에 대해 $f_M(\boldsymbol{x}_i)$의 값을 플롯한다. 예를 들어, x_1변수가 관심변수일 때 표본 $i = 1,2,...,n$이므로 x_1 변수 관측치를 X축에 놓고 대응된 $f_M(\boldsymbol{x}_i)$를 Y축에 그리면 된

다. $\boldsymbol{x}_i=(x_{i1},x_{i2},x_{i3}),\ i=1,2,3,4$라면 즉 3개의 특성변수와 4개의 관측치가 있고 x_1변수가 $x_{21}<x_{11}<x_{41}<x_{31}$의 순서라면 대응되는 $f_M(\boldsymbol{x}_2),f_M(\boldsymbol{x}_1),f_M(\boldsymbol{x}_4),f_M(\boldsymbol{x}_3)$로 그림을 그리면 변수 x_1의 값이 증가할 때 $f_M(\boldsymbol{x})$의 행태를 알게 된다.

11.4 XGBoost(Extreme Gradient Boosting)

XGBoost는 GBM과 동일한 알고리즘을 사용하지만 자료의 크기가 딥러닝을 사용할 만큼 크지 못하거나 정형화된 자료에서는 가장 강력한 머신러닝 기법이라고 할 수 있다. GBM과 가장 큰 차이점은 계산속도가 GBM보다 훨씬 빠르고 과대적합을 방지하기 위한 정규화가 있다는 것이다. 참고로 GBM은 과대적합이 자주 일어나는데 모수에 대한 규제화의 기능이 없어 GridSearch 등의 기능을 이용하여 모수를 조절해야 한다. 이러한 관점에서 XGBoost를 규제화된 GBM (regularized GBM)이라고 한다. 또 다른 특징으로는 결측치를 모두 0으로 처리하여(물론 결측치를 다른 값으로 지정할 수 있다) 이 값을 결측치로 학습하게 하는 기능이 있으며, 특성변수를 임의로 일부만 뽑아 사용할 수도 있다. 그러므로 의사결정나무(CART)를 기초모형으로 쓰게 되면 이 XGboost는 random forest와 유사한 기능을 가지게 된다.

이제 XGboost의 작동원리를 구체적으로 살펴보도록 하자. GBM는 $\hat{y_i}=\sum_{k=0}^{M}\phi_k(\boldsymbol{x}_i)$의 형태로 $\phi_0(\boldsymbol{x}_i)$는 손실함수가 SSE일 때는 $\bar{y}$, SAE일 때는 y의 중위수, 로짓일 때는 특정 클래스의 비율인 것을 살펴본 바 있다. 이미 GBM에서 살펴본 바와 같이 손실함수가 무엇이든 개념적으로 동일하므로 손실함수가 SSE일 경우로 설명을 전개하고자 한다. $\phi_1(\boldsymbol{x}_i)$는 잔차 $y_i-\bar{y}$에 T 개의 영역으로 분리된 CART를 적용하여 구한 영역별 잔차 $y_i-\bar{y}$ 의 평균이고 $\phi_2(\boldsymbol{x}_i)$는 잔차 $y_i-(\phi_0(\boldsymbol{x}_i)+\phi_1(\boldsymbol{x}_i))$에 적용한 CART의 영역별 평균이며, 이를 반복하여 최종적으로 $\phi_M(\boldsymbol{x}_i)$는 잔차 $y_i-\sum_{k=0}^{M-1}\phi_k(\boldsymbol{x}_i)$에 적용한 CART의 영역별 평균으로 계산한 바 있다. $\phi_k(\boldsymbol{x}_i)$의 영역별 평균을 $w_j,\ j=1,2,\cdots,T$로 표기하고 (이 w_j은 k에 의존하지만 부호의 복잡함을 피하기 위해 생략함) $\hat{y}^{(m-1)}=\sum_{k=0}^{m-1}\phi_k(\boldsymbol{x}_i)$라고 표기하자.

이제 일반적인 손실함수 l에 대해 XGboost에서 정의한 $\phi_m(\boldsymbol{x}_i)$의 손실함수는

$$L^{(m)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{(m-1)} + \phi_m(\boldsymbol{x}_i)) + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \qquad (11.10)$$

으로 정의한다. $\phi_m(\boldsymbol{x}_i)$는 잔차 $y_i - \hat{y}^{(m-1)}$에 적용한 CART추정치이며 w_j는 영역 j의 $\phi_m(\boldsymbol{x}_i)$ 값이다. 그러므로 XGboost는 영역수 T 와 모수 w_j에 대한 규제를 통해 과대적합문제를 해결하려는 손실함수를 가지고 있다. 만약 $\gamma = \lambda = 0$이면 XGboost는 GBM과 동일한 결과를 산출하게 된다. (11.10)의 손실함수를 ϕ_m에 대해 테일러 전개에 의한 2차 항까지 근사하면

$$L^{(m)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}^{(m-1)}) + g_i \phi_m(\boldsymbol{x}_i) + \frac{1}{2} h_i \phi_m^2(\boldsymbol{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \qquad (11.11)$$

여기에서 $g_i = \dfrac{\partial l(y_i, \hat{y}^{(m-1)})}{\partial \hat{y}^{(m-1)}}$이고 $h_i = \dfrac{\partial^2 l(y_i, \hat{y}^{(m-1)})}{(\partial \hat{y}^{(m-1)})^2}$이다. (11.11)에서 $l(y_i, \hat{y}^{(m-1)})$은 이미 계산된 상수이므로 (11.11)를 최소화하는 $\phi_m(\boldsymbol{x}_i)$(동일하게 w_j)를 찾는 손실함수는

$$\ell^{(m)} = \sum_{i=1}^{n} [g_i \phi_m(\boldsymbol{x}_i) + \frac{1}{2} h_i \phi_m^2(\boldsymbol{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \qquad (11.12)$$

I_j를 $\phi_m(\boldsymbol{x}_i)$의 j 번째 영역에 포함된 표본이라고 할 때 I_j안의 모든 표본은 $\phi_m(\boldsymbol{x}_i) = w_j$이므로 (11.12)는 다음과 같이 재 표현된다.

$$\ell^{(m)} = \sum_{j=1}^{T} [w_j \sum_{i \in I_j} g_i + \frac{1}{2} w_j^2 (\lambda + \sum_{i \in I_j} h_i)] + \gamma T \qquad (11.13)$$

$\ell^{(m)}$을 최소화하는 w_j를 구하면

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\lambda + \sum_{i \in I_j} h_i}$$

이며 이를 (11.13)에 대입하면

$$\ell^{(m)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\lambda + \sum_{i \in I_j} h_i} + \gamma T \qquad (11.14)$$

(11.14)에서 $\phi_m(\boldsymbol{x}_i)$의 j 번째 영역에 의한 손실함수의 기여도는

$$-\frac{1}{2}(\sum_{i \in I_j} g_i)^2 / (\lambda + \sum_{i \in I_j} h_i)$$

이다. 그러므로 영역 j를 분할하여 생긴 영역을 $I_j = I_L \bigcup I_R$으로 표현하여 줄어든 손실함수 값은

$$IG_j = \frac{1}{2}[\frac{(\sum_{i\in I_L} g_i)^2}{(\lambda + \sum_{i\in I_L} h_i)} + \frac{(\sum_{i\in I_R} g_i)^2}{(\lambda + \sum_{i\in I_R} h_i)} - \frac{(\sum_{i\in I_j} g_i)^2}{(\lambda + \sum_{i\in I_j} h_i)}]$$

이 된다. 그러므로 XGboost에서는 IG_j가 가장 큰 특성변수의 노드를 선택하는 기준으로 사용한다. 정리하면 다음과 같다.

1. 손실함수에 따라 ϕ_0를 y의 전체평균 또는 중위수 또는 특정클래스의 비율을 구한 후, $\hat{y}^{(0)} = \phi_0$로 놓는다.
2. $l(y_i, \hat{y}^{(0)})$를 $\hat{y}^{(0)}$에 대해 미분하여 g_i와 h_i를 계산하고 IG의 값이 가장 큰 특성변수와 노드를 구한다.
3. $m = 1$부터 M까지
   (1) 잔차 $y_i - \sum_{k=0}^{m-1} \phi_k$와 선택된 특성변수 노드의 영역별로 ϕ_m을 구한다.
   (2) $\hat{y}^{(m)} = \sum_{k=0}^{m} \phi_k$로 놓고 $l(y_i, \hat{y}^{(m)})$을 $\hat{y}^{(m)}$에 대해 미분하여 g_i와 h_i를 구한다.
   (3) ϕ_m의 영역별로 IG를 최대로 하는 특성변수와 의사결정나무의 새로운 분할을 위한 노드를 선택한다.
4. (11.12)의 XGboost 손실함수인 $l^{(m)}$이 더 이상 감소하지 않을 때까지 M을 줄이거나 증가시키되, 과대적합이 되지 않도록 한다.

XGboost에서 M, T, γ, λ 초모수이다. XGBoost는 scikit-learn 라이브러리를 사용할 수 있으나 독립적인 라이브러리로 사용한다. XGBoost의 사용 예는 다음 장의 실제 적용사례에서 좀 더 자세하게 논의될 것이다.

11.4 적용 사례

아래의 자료는 이미 여러 번 사용한 Iris 데이터로 자료의 사전작업을 보여주고 있다.

```
import seaborn as sns
iris = sns.load_dataset('iris')
X = iris.drop('species', axis=1)
y=iris['species']

from sklearn.preprocessing import LabelEncoder
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3, random_state=1,
stratify=y)
```

아래 프로그램은 random forest, SVM, 그리고 로지스틱회귀를 통해 Iris꽃 품종을 분류하고 각각의 정밀도를 출력하고 있다. VotingClassifier 클래스를 호출하여 이 세 가지의 분류법을 앙상블 머신러닝으로 합쳐서 그 결과를 보여 주고 있다. 앙상블러닝이 개별분류법보다 최소한 같거나 우수한다는 것을 알 수 있다.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_cl=LogisticRegression()
rf_cl=RandomForestClassifier()
svm_cl=SVC()
voting_cl=VotingClassifier(estimators=[('lr', log_cl), ('rf', rf_cl), ('svc',
svm_cl)],voting='hard')
voting_cl.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
for cl in (log_cl, rf_cl, svm_cl, voting_cl):
    cl.fit(X_train, y_train)
    y_pred = cl.predict(X_test)
    print(cl.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
(out) LogisticRegression 0.9777777777777777
      RandomForestClassifier 0.9555555555555556
      SVC 0.9777777777777777
      VotingClassifier 0.9777777777777777
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bag_cl = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
max_samples=100, bootstrap=True)
bag_cl.fit(X_train, y_train)
y_pred = bag_cl.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
(out) 0.9777777777777777
```

위 예제는 Bagging learning을 위한 프로그램이다. 500개의 Decision tree 모형을 사용하였으며 max_sample=100은 붓스트랩 표본의 크기이다. 총 training 개수에서 60%의 표본을 이용하여 bagging learning을 하려면 max_sample=0.6 지정하면 된다. 그러므로 Bagging은 Bootstrap 표본의 크기를 조절할 수 있다.

```
bag_cl2 = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
bootstrap=True, oob_score=True)
bag_cl2.fit(X_train, y_train)
print(bag_cl2.oob_score_)
```

```
(out) 0.9428571428571428
```

위 예제는 out_of_bag (oob) 자료를 이용하여 검증데이터의 정밀도를 구한 프로그램이다. BaggingClassifier 클래스에서 oob_score=True로 지정해주면 된다.

```
from sklearn.ensemble import AdaBoostClassifier
ada_t = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2), n_estimators=500,
random_state=1)
ada_t.fit(X_train, y_train)
y_train_pred=ada_t.predict(X_train)
y_test_pred=ada_t.predict(X_test)

from sklearn.metrics import accuracy_score
ada_train=accuracy_score(y_train, y_train_pred)
ada_test=accuracy_score(y_test, y_test_pred)

print("Adaboost train/test accuracy %0.3f/%0.3f" %(ada_train, ada_test))
```

```
(out) Adaboost train/test accuracy 1.000/0.978
```

위 프로그램은 앙상블러닝 중 Adaboost를 적용한 사례이다. AdaBoostClassifier 클래스를 호출하여 실행하며 깊이가 2인 분류나무를 이용하여 Adaboost를 Iris 자료에 M=500번을 적용하여 최신화하였다. 낮은 깊이의 분류나무를 적용하였음에도 학습데이타의 정밀도는 1이고 시험데이터의 정밀도는 0.978로 나타났다.

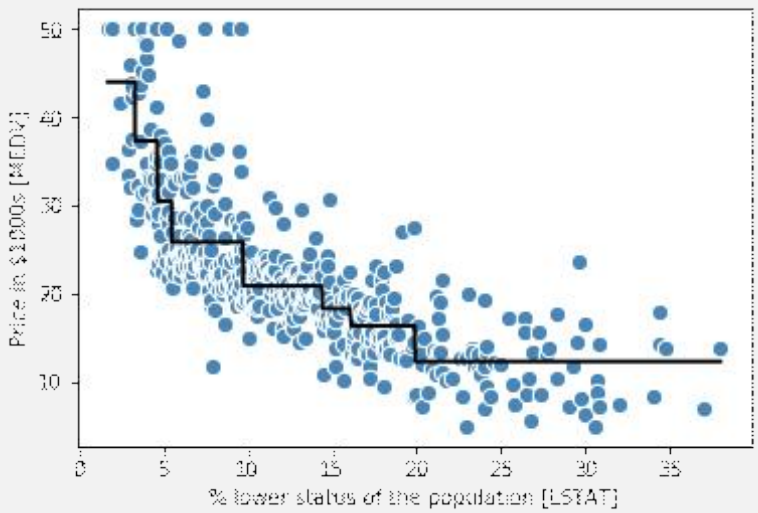
아래 예제는 기울기부스팅을 Iris 데이터에 적용한 프로그램이다. 기울기부스팅에 의한 분류는 GradientBoostingClassifier 클래스를 호출하여 실행한다. M=100이고 사용한 분류나무는 깊이가 2를 낮은 깊이의 의사결정나무이다. staged_predict는 매 stage ($m=1,2,...,M$)마다 예측치를 산출해주므로 시험데이터에 이를 적용하여 정밀도를 계산할 수 있게 한다. 시험데이터의 정밀도가 가장 높은(np.argmax(accuracies)) m을 찾아, 이 m을 M으로 기울기부스팅을 학습데이터에 다시 적용하여 (gbcl_best=GradientBoostingClassifier (max_depth=2, n_estimators=best_n_estimator) GBM모형을 추정하고 있다.

```
import seaborn as sns
iris = sns.load_dataset('iris')
X = iris.drop('species', axis=1)
y=iris['species']

from sklearn.preprocessing import LabelEncoder
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3, random_state=1,
stratify=y)

from sklearn.ensemble import GradientBoostingClassifier
gbcl=GradientBoostingClassifier(n_estimators=100, max_depth=2)
gbcl.fit(X_train, y_train)
accuracies=[accuracy_score(y_test,y_pred) for y_pred in
gbcl.staged_predict(X_test)]
best_n_estimator=np.argmax(accuracies)
gbcl_best=GradientBoostingClassifier(max_depth=2, n_estimators=best_n_estimator)
gbcl_best.fit(X_train, y_train)
y_train_pred=gbcl_best.predict(X_train)
y_test_pred=gbcl_best.predict(X_test)
print(accuracy_score(y_train, y_train_pred))
print(accuracy_score(y_test, y_test_pred))
```

```
(out) 0.9523809523809523
      0.9555555555555556
```

지금까지 논의한 앙상블모형의 적용은 모두 분류이기 때문에 RandomForestClassifier, BaggingClassifier, 그리고 GradientBoostingClassifier 등으로 클래스를 Classifier로 사용하였다. 만약 회귀를 원한다면 단순하게 Classifier를 Regressor로 바꿔서 사용하면 된다. 다음의 예제는 House 데이터에 회귀나무모형을 적용한 것이며, 오직 하나의 특성변수만을 이용한 것은 추정된 회귀선을 2차원 상에 보이기 위함이다. 추정된 회귀선은 step 함수 형태의 비선형회귀임을 알 수 있다.

```
import pandas as pd
house = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                    'python-machine-learning-book-2nd-edition'
                    '/master/code/ch10/housing.data.txt',header=None,sep='₩s+')
house.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

from sklearn.tree import DecisionTreeRegressor
X = house[['LSTAT']].values
y = house['MEDV'].values
tree = DecisionTreeRegressor(max_depth=3)
tree.fit(X, y)
sort_idx = X.flatten().argsort()

def lin_regplot(X, y, model):
    plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
    plt.plot(X, model.predict(X), color='black', lw=2)
    return None

import matplotlib.pyplot as plt
lin_regplot(X[sort_idx], y[sort_idx], tree)
plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000s [MEDV]')
plt.show()
```

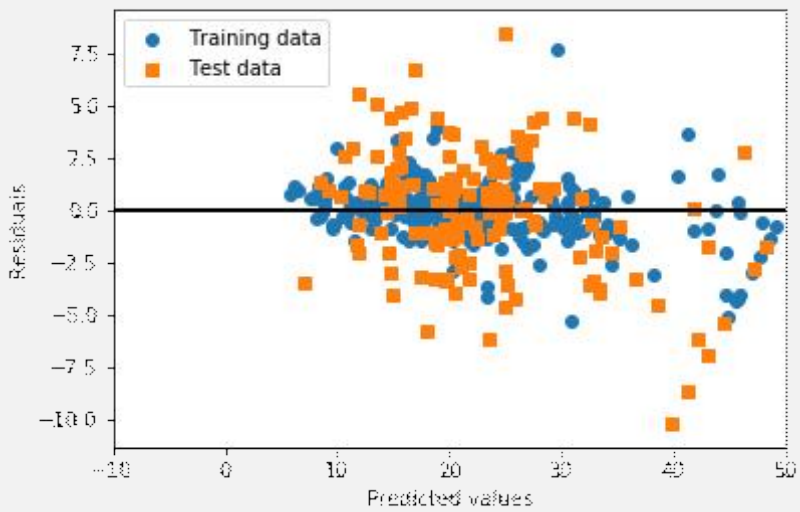


다음은 random forest모형을 House 데이터에 적용한 예이다. random forest모형은 RandomForestRegressor 클래스를 호출하여 실행되며 1000개의 회귀나무를 이용하며 노드를 분류할 때 MSE를 사용하고 있다. 과대적합이 되어 모수에 대한 규제화가 필요해 보인다.

```
X=house.iloc[:, :-1].values
y=house['MEDV'].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

from sklearn.ensemble import RandomForestRegressor #max_depth=5
forest=RandomForestRegressor(n_estimators=1000, criterion='mse', random_state=1)
forest.fit(X_train, y_train)
y_train_pred=forest.predict(X_train)
y_test_pred=forest.predict(X_test)

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
print('MSE train : %0.3f, test: %0.3f' %(mean_squared_error(y_train, y_train_pred),
mean_squared_error(y_test, y_test_pred)))
print('R**2 train : %0.3f, test: %0.3f' %(r2_score(y_train, y_train_pred),
r2_score(y_test, y_test_pred)))
```

```
(out) MSE train : 1.582, test: 8.264
      R**2 train : 0.981, test: 0.910
```

```
plt.scatter(y_train_pred, y_train_pred - y_train,marker='o',label='Training data')
plt.scatter(y_test_pred,y_test_pred - y_test,marker='s',label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, lw=2)
plt.xlim([-10, 50])
plt.show()
```



아래 프로그램은 House 데이터에 XGboost를 적용한 사례이다. Random forest모형에서 자료를 살펴보지 않아 여기에서는 좀 더 자세하게 살펴보고자 한다. 총 13개의 특성변수로 구성되어 있으며 맨 마지막 열에 있는 'MEDV'가 주택가격으로 단위는 1,000$이다.

```
import pandas as pd
house = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                    'python-machine-learning-book-2nd-edition'
                    '/master/code/ch10/housing.data.txt',header=None,sep='₩s+')
house.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
house.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

house.info()는 데이터셋 house에 대한 총 표본 수(506개), 총변수 수(14개), 변수별로 결측여부, 변수의 자료타입 등을 보여주고 있다.

```
print(house.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM        506 non-null float64
ZN          506 non-null float64
INDUS       506 non-null float64
CHAS        506 non-null int64
NOX         506 non-null float64
RM          506 non-null float64
AGE         506 non-null float64
DIS         506 non-null float64
RAD         506 non-null int64
TAX         506 non-null float64
PTRATIO     506 non-null float64
B           506 non-null float64
LSTAT       506 non-null float64
MEDV        506 non-null float64
dtypes: float64(12), int64(2)
memory usage: 55.4 KB
None
```

XGboost적용은 먼저 xgboost를 호출하여 시작한다. XGboost가 호출되지 않으면

anaconda의 prompt 창으로 들어가서 “pip3 install xgboost”를 입력하여 먼저 xgboost를 설치하여야 한다. XGboost는 회귀를 위해서는 XGBRegressor 클래스를 이용하고 분류가 목적인 경우에는 XGBClassifier 클래스를 이용한다. objective는 손실함수를 정의하는 것으로 회귀인 경우, ‘reg:squarederror’가 디폴트이며 이는 SSE를 의미한다. XGBClassifier를 사용할 때의 objective는 이항 분류일 경우, ‘binary:logistic’을 쓰고 다항 분류일 경우에는 ‘multi:softmax’ 또는 ‘multi:softprob’을 지정하면 된다. 이 두 가지 다중분류 손실함수는 모두 softmax 함수이지만 softprob는 출력을 확률로 해준다는 차이점이 있다. booster는 추정에 의사결정나무(‘gbtree’)를 사용할지, 아니면 선형모형(‘gblinear’)을 사용할지, 또는 dart라는 또 다른 의사결정나무를 쓸지(‘dart’)를 지정한다. 디폴트는 gbtree로, 일반적으로 가장 우수하다. colsample_bytree는 의사결정나무에서 사용하는 특성변수들 중 몇 %를 사용할지 지정해 주는 것으로 random forest모형과 동일하게 과대적합을 방지하는데 중요한 역할을 한다. learning_rate는 학습율을, max_depth는 의사결정나무의 깊이를 지정해주고, gamma는 의사결정나무의 총 영역수에 대한 규제화이며 alpha는 L_1규제화의 정도를 조정하고 lambda는 L_2규제화의 정도를 조절한다. 또한 n_estimators는 몇 개의 의사결정나무를 사용할 것이지를 지정해 준다. 즉, n_estimators는 M을 말한다. 아래의 결과를 보면 앞에서 사용한 random forest모형보다 과대적합이 많이 완화된 것을 볼 수 있다.

```
# House data에 xgboost 적용.
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

X=house.iloc[:, :-1].values
y=house['MEDV'].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

xg_reg=xgb.XGBRegressor(objective='reg:squarederror', booster='gbtree',
colsample_bytree=0.75, learning_rate=0.1,max_depth=5, alpha=10, n_estimators=30)
xg_reg.fit(X_train, y_train)
pred_train=xg_reg.predict(X_train)
pred_test=xg_reg.predict(X_test)
rmse_train=np.sqrt(mean_squared_error(y_train,pred_train))
rmse_test=np.sqrt(mean_squared_error(y_test,pred_test))
print('RMSE train : %0.3f, test: %0.3f' %(rmse_train, rmse_test))
```

```
RMSE train : 2.034, test: 3.427
```

XGBoost에서도 cv함수를 이용하여 교차검증이 가능하다. 그러나 주의해야 할 것은 XGBoost에서 사용하는 자료타입으로 변환하기 위해 아래와 같이 DMatrix를 이용하여야 한다. DMatrix의 data는 넘파이 2차원(2D) 자료여야 ~~하며 label은 1D자료여야~~ 한다. 그러므로 다항분류일 경우, 문자일 경우 LabelEncoder를 이용하여 0~(총범주수-1)로 자료를 전환해야 한다. xgb.cv에서 nfold는 학습데이터를 5등분한다는 뜻이며 그 중 하나는 검증데이터로 사용된다는 의미이다. num_boost_round는 몇 개의 의사결정나무를 XGBoost에 사용할 것인지를 지정하는 것으로 앞에서 사용한 n_estimators와 동일하다. early_stopping_rounds=20는 연속하여 20번까지 의사결정나무를 썼는데도 metric(우리 예제에서 rmse)가 더 이상 줄어들지 않으면 20번째의 의사결정나무에서 멈추라는 명령어이다.

```
paras={'objective':'reg:squarederror','colsample_bytree':0.6,'max_depth':5, 'alpha':10}
data_dim1=xgb.DMatrix(data=X_train,label=y_train)
cv_result=xgb.cv(dtrain=data_dim1, params=paras, nfold=5,num_boost_round=60,
                 early_stopping_rounds=20,metrics='rmse', as_pandas=True, seed=1)
cv_result.head()
```

	train-rmse-mean	test-rmse-std	test-rmse-mean	test-rmse-std
0	17.139679	0.223515	17.271275	1.054852
1	12.606229	0.223895	12.896409	1.042451
2	9.444665	0.226033	9.855797	1.089853
3	7.218817	0.181744	7.906023	1.086968
4	5.632334	0.200208	6.552482	1.144863

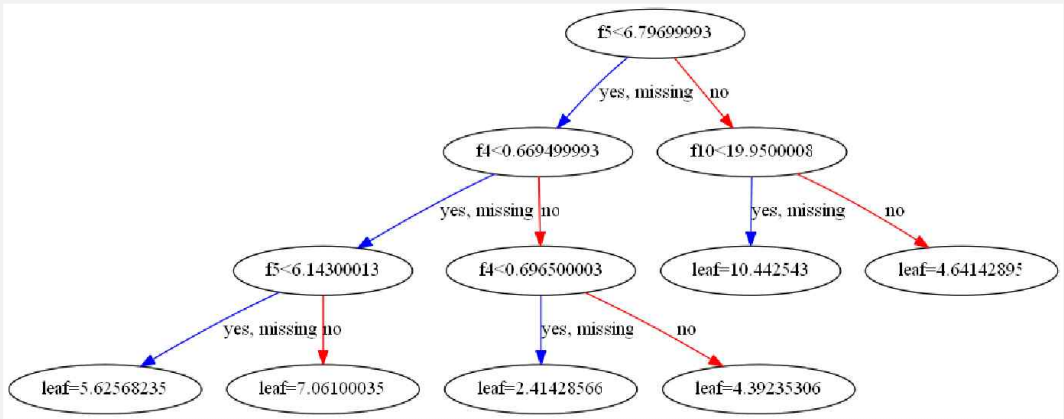
아래의 결과로 맨 마지막 60번째(그러므로 총 60개)의 의사결정나무에서 계산된 시험 데이터의 rmse=3.86임을 보여주고 있다.

```
print(cv_result['test-rmse-mean'].tail(1))
```

```
59    3.865178
Name: test-rmse-mean, dtype: float64
```

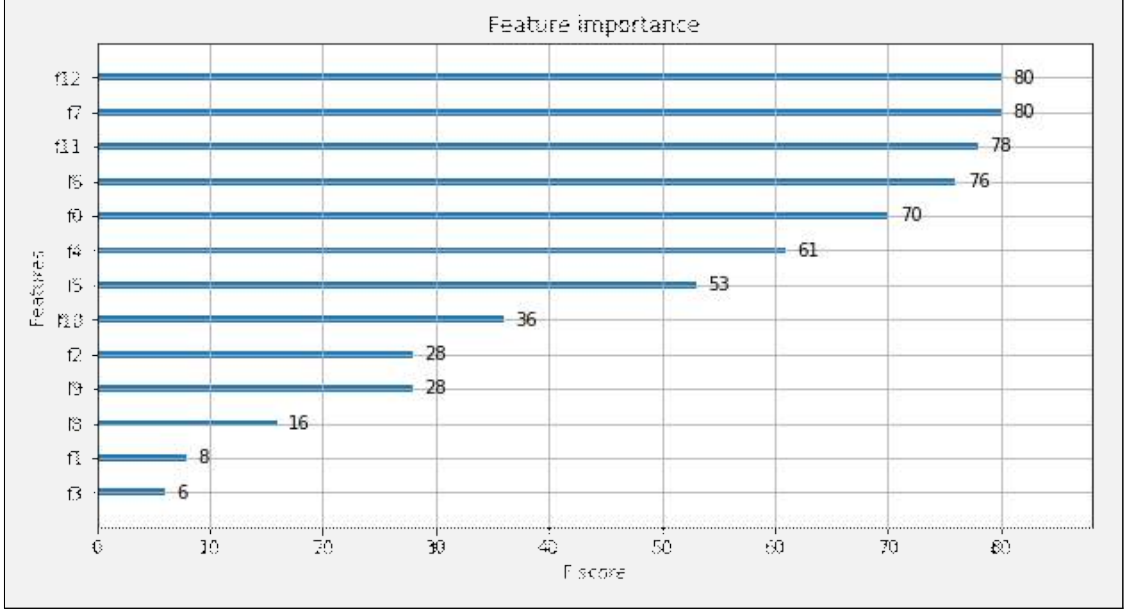
XGBoost는 사용한 의사결정나무의 구체적 분류를 출력할 수 있다. 아래 프로그램은 첫 번째 의사결정나무의 분류를 보여주고 있다. plot_tree로 이러한 그림을 출력하였으며 이 그림의 출력을 위해 matplotlib.pyplot을 호출하였다. xgb.train은 앞에서 xg_reg=xgb.XGBRegressor 를 사용한 xg_reg.fit과 동일하지만 xgb.train이 속도가 훨씬 빠르다.

```
xg_reg1=xgb.train(params=paras, dtrain=data_dim1, num_boost_round=60)
import matplotlib.pyplot as plt
xgb.plot_tree(xg_reg1,num_trees=0)
plt.rcParams['figure.figsize']=[50,10]
plt.show()
```



다음은 plot_importance를 이용하여 변수별로 중요도를 출력하여 각 설명변수의 상대적 중요도를 측정할 수 있다.

```
xgb.plot_importance(xg_reg1)
plt.rcParams['figure.figsize']=[5,5]
plt.show()
```



다음은 XGBoost에 gridsearch를 교차검증을 통해 구하는 프로그램이다. XGBoost의 초모수를 결정하는데 사용한다.

```
from sklearn.model_selection import GridSearchCV
param={'max_depth':range(3,10,2),'colsample_bytree':[i/100.0 for i in
range(75,90,5)]}
xgsearch=GridSearchCV(estimator=xgb.XGBRegressor(objective='reg:squarederror',
max_depth=5,alpha=10),param_grid=param,
                              scoring='neg_mean_squared_error', cv=5)
xgsearch.fit(X_train,y_train)
xgsearch.best_params_, xgsearch.best_score_
```

```
({'colsample_bytree': 0.75, 'max_depth': 5}, -12.103208228411457)
```

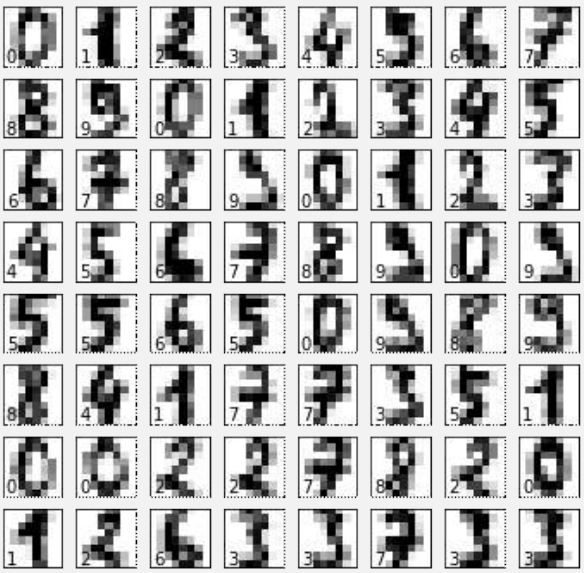
제 5장 Naive Bayes에서 다룬 0~9까지 손글씨를 random forest를 이용하여 식별하고자 한다.

```
from sklearn.datasets import load_digits
digits=load_digits()
print(digits.keys())
```

```
(out) dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

여기에서 'target'은 y변수로, 0~9가 10개의 class 변수이며, data는 X 변수로 8×8 pixel data를 64차원의 특성변수로 전환한 것이다. 그리고 'images'는 8×8 pixel 자료를 의미한다. 다시 5장으로 가서 데이터의 형태를 보는 번거로움을 피하기 위해 손글씨와 이에 대응하는 참 값을 다음의 프로그램으로 그려보기로 한다.

```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(6,6)) # figure size in inches
for i in range(64):
    ax=fig.add_subplot(8, 8, i+1, xticks=[], yticks=[])
    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')
    ax.text(0, 7, str(digits.target[i]))
```



위 프로그램에서 ax.text는 실제 참 값을 손글씨 옆에 출력하는 명령어이다.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(digits.data, digits.target,
random_state=0)

from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=1000)
rfc.fit(X_train, y_train)
y_test_pred=rfc.predict(X_test)

from sklearn import metrics
print(metrics.classification_report(y_test_pred, y_test))
```

	precision	recall	f1-score	support
0	1.00	0.97	0.99	38
1	1.00	0.96	0.98	45
2	0.95	1.00	0.98	42
3	0.98	0.98	0.98	45
4	0.97	1.00	0.99	37
5	0.98	0.98	0.98	48
6	1.00	1.00	1.00	52
7	1.00	0.96	0.98	50
8	0.94	0.98	0.96	46
9	0.98	0.98	0.98	47
micro avg	0.98	0.98	0.98	450
macro avg	0.98	0.98	0.98	450
weighted avg	0.98	0.98	0.98	450

정밀도가 평균적으로 98%로 나타나 Naive Bayes의 83%보다 월등하게 향상되었음을 알 수 있다.

아래 예제는 회귀를 위해 기울기부스팅 앙상블을 House 데이터에 적용한 사례이다. GradientBoostingRegressor 클래스를 호출하여 실행하며 반복을 120으로 잠정적으로 설정하고 회귀나무의 깊이를 3으로 선택하였다.

bst_n_estimators=np.argmin(errors)에 의해 가장 낮은 MSE를 보여주는 M을 결정한 후 모형을 재추정하고 있다. best.feature_importances_ 속성(attribute)를 이용하여 0부터 12로 label된 특성변수(CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT')의 상대적인 기여도를 출력하고 있다. 6번째 특성변수(RM)의 기여도는 26.6%, 13번째 특성변수(LSTAT)의 기여

도는 51.4%로 13개의 변수 중 이 두 개의 특성변수가 78% 기여하는 것으로 나타났다.

```
import pandas as pd
house = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                    'python-machine-learning-book-2nd-edition'
                    '/master/code/ch10/housing.data.txt',header=None,sep='₩s+')
house.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
from sklearn.tree import DecisionTreeRegressor
X=house.iloc[:, :-1].values
y=house['MEDV'].values

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)

from sklearn.ensemble import GradientBoostingRegressor
gbrg=GradientBoostingRegressor(n_estimators=120, max_depth=3)
gbrg.fit(X_train, y_train)
errors=[mean_squared_error(y_test, y_pred) for y_pred in
gbrg.staged_predict(X_test)]
bst_n_estimators=np.argmin(errors)
gbrg_best=GradientBoostingRegressor(max_depth=3, n_estimators=bst_n_estimators)
gbrg_best.fit(X_train, y_train)
print(gbrg_best.feature_importances_)
```

```
(out) [4.91366516e-02 4.83846814e-05 5.67729554e-03 6.21104369e-04
 1.61785430e-02 2.66285836e-01 9.96807655e-03 7.76471669e-02
 4.74499481e-03 1.53524849e-02 2.51302405e-02 1.56344608e-02
 5.13574760e-01]
```

이 두 개의 특성변수에 대한 값의 증가에 따른 목표변수 y의 변화를 살펴보기 위해 plot_partial_dependence 함수를 이용하여 그림을 그리고 있다. features=(5,12,(5,12))으로 지정하여 5번, 12번 특성변수의 플롯을 각자 그리고 (5,12)에 의해 5번과 12번의 2차원 그림을 보여주는 명령어이다.

```
from sklearn.ensemble.partial_dependence import plot_partial_dependence
fig=plot_partial_dependence(gbrg_best, X_train, features=(5,12,(5,12)))
```