10. Boosting
2019년 가을 학기

2019년 08월 23일
https://www.github.com/KU-BIG/KUBIG_2019_Autumn

우리는 앞선 강의에서 ensemble기법과 그 중에서도 bagging과 random forest를 다뤘다. 이번 강의에서는 나머지 Ensemble기법인 boosting에 대해서 살펴보도록 하겠다. Boosting의 여러 종류들과 bagging과의 차이점을 알아보도록 하겠다.

1. Introduction

Boosting은 bagging과 유사하지만 다른 ensemble의 대표적인 기법이다. Bagging은 병렬적인 모델을 결합한다면, boosting은 직렬적으로 모델을 결합하는 것이 가장 큰 차이이다. Bagging은 각 bag간의 관계가 전혀 없는 독립적인 시행으로 이루어진다. 이와 반면에 boosting은 이전 시행의 결과가 연속적으로 다음 시행으로 이어진다.

Bagging은 독립적으로 모델을 구성하지만 boosting은 이전 모델에서 실패한 것을 바탕으로 새 모델을 구성한다. 두 기법 모두 랜덤한 복원추출을 하지만 bagging은 매 sampling마다 동일한 가중치를 두는 반면에 boosting은 학습 오류가 큰 데이터에 가중치를 부여한다. Bagging은 overfitting을 해결하는데 도움이 되지만, boosting은 bias를 줄이는데 도움이 된다.

그렇다면 두 방법 중 어떤 방법을 선택하는 것이 좋을까? 두가지 방법 모두 여러 모델을 종합하여 결과를 내기에 단일 모델보다 분산을 감소시켜 안정성이 높다. 만약 단일 모델의 성능이 낮다면, boosting을 선택하는 것이 좋다. Boosting은 단일 모델의 장점을 최적화하고 단일 모델에서 나타날 수 있는 위험을 감소시키기 때문에 오류가 적은 결합 모델을 생성할 수 있기 때문이다. 반대로 단일 모델의 over-fitting이 문제라면 bagging을 선택하는 것이 낫다. Boosting은 오히려 over-fitting을 증가시키기 때문에 문제를 해결하는데 도움이 되지 않는다. 이번 강의에서는 boosting기법 중 Ada Boosting(Adaptive Boosting), Gradient Boosting, XG Boosting(Extreme Gradient Boosting)에 대해서 간단하게 다루고자 한다.

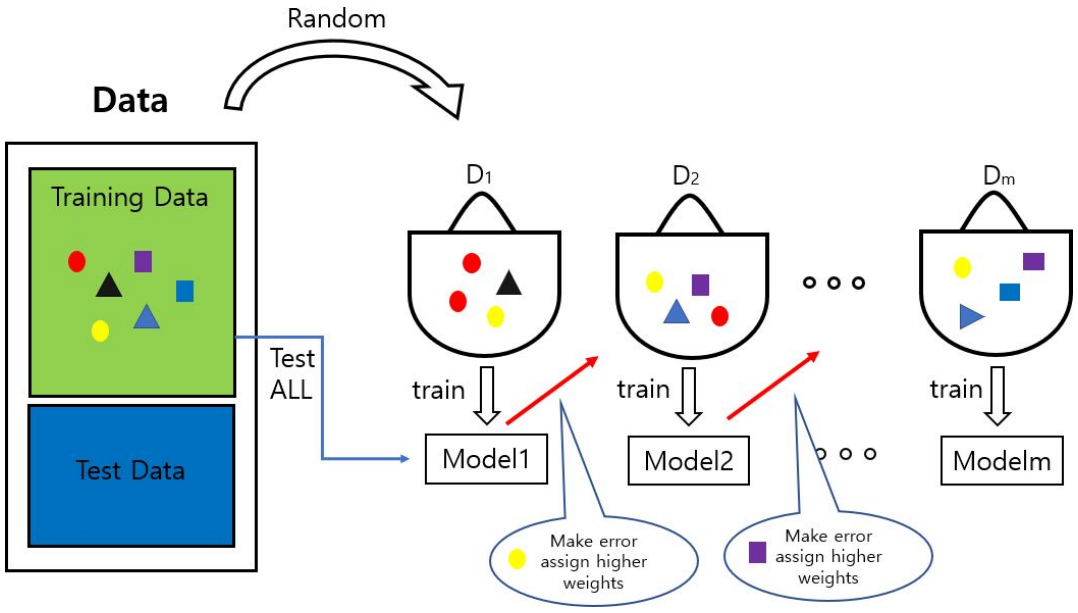
2. Ada Boosting

Ada Boosting은 Adaptive Boosting의 약자로 성능을 향상시키기 위하여 다른 많은 형태의 학습 알고리즘과 결합하여 사용할 수 있다. AdaBoost는 이전의 분류기에 의해 잘못 분류된 것들을 이어지는 약한 학습기들이 수정해줄 수 있다는 점에서 다양한 상황에 적용할 수 있다(adaptive). 간단하고 성능이 떨어지는 약분류기(weak classifier)들이 상호보완하도록 단계적으로 학습하여, 이를 조합하여 만들어진 최종 강분류기(strong classifier)의 성능을 증폭시키는 원리이다.

이 기법의 핵심은 이전 분류기에서 오분류된 것을 이어지는 약분류기들이 수정해 줄 수 있다는 것이다. 이전 모델에서 잘못 분류된 개체들을 다음 sample에 뽑힐 확률을 늘려 많이 뽑히게 하고, 이전에 제대로 분류된 개체들은 뽑힐 확률을 줄여 덜 뽑히게 한다. 개별 학습기의 성능이 비교적 떨어져도, 이전 분류에서 오류율이 0.5보다 낮다면 최종 모델은 강분류기로 수렴한다.

이러한 Boosting은 bias를 줄이려는 것에 초점을 맞춘 기법이므로 overfitting의 문제가 발생할 수 있다. 따라서 모델의 모수를 적절하게 조정하는 것이 중요하게 작용한다. Bias가 줄어들어 정확도는 올라가지만 outlier에 취약하다.

아래의 그림을 통해 전반적인 AdaBoosting에 대해서 살펴보자.



https://blog.naver.com/euleekwon/221377873711

먼저 training data set에서 랜덤으로 데이터를 뽑아 D1을 구성한다. D1의 데이터를 바탕으로 model1을 생성한 뒤 training data set 전체를 사용하여 성능을 테스트한다. 테스트 결과 위의 그림처럼 노란색의 에러가 크게 나왔다면 이는 model1에서 해결하지 못한 문제이기 때문에

가중치를 주어 D2를 구성할 때 뽑힐 확률을 증가시킨다. 즉 다음 model이 그 문제에 집중할 수 있게 하는 것이다. 마찬가지로 가중치가 부여된 training data set에서 D2를 뽑아 model2를 생성하고 테스트한 뒤 결과에 따라 가중치를 부여한다. 이 때 mini ensemble처럼 model1과 model2의 결과를 종합하여 가중치를 부여한다. 이런 방식으로 이전 model들의 test결과를 반영하여 다음 model을 생성하고, 분류모델의 경우는 weighted majority vote, 회귀모델의 경우에는 weighted sum을 사용하여 최종 모델을 생성한다.

단계별로 나누면

1단계) 현재 주어진 데이터셋에 대해 상대적으로 단순한 모델을 이용하여 학습을 진행한다.

2단계) 학습오류가 큰 개체의 선택확률을 증가시키고, 학습 오류가 작은 경우의 데이터 샘플의 선택확률을 감소시킨다.

3단계) 2단계에서 조정된 확률을 기반으로 다음 단계에서 학습할 데이터 셋을 구성. 이 과정을 통해 현재의 모델이 잘 해결하지 못하는 어려운 데이터 샘플에 집중하게 된다.

4단계) 종료 조건을 충족하는 오류가 발생할 경우까지 위의 단계를 반복하여 실행한다.

5단계) 일반적으로 분류모델의 경우는 weighted majority vote, 회귀모델의 경우에는 weighted sum을 사용하여 최종 모델을 생성한다.

2.1 Ada Boosting 알고리즘 간단 해석

Algorithm AdaBoost

1: Init data weights $\{w_n\}$ to $1/N$
2: **for** $m = 1$ to M **do**
3: fit a classifier $y_m(x)$ by minimizing weighted error function J_m:
4: $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5: compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6: evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7: update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: **end for**
9: Make predictions using the final model: $Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$

위의 알고리즘을 간단하게 설명하면 아래와 같다.

1. 첫 번째 분류기인 $y_1(x)$는 데이터 샘플에 모두 같은 가중치(1/N)를 부여하여 학습한다.
2. for m=1 to M do
3. weighted error function J_m을 최소화하는 방향으로 모델 $y_m(x)$를 적합한다.
4. J_m은 Indicator function에 분류기가 오분류할 확률이 들어있다. 앞에 가중치를 곱해줌으로써 이전 분류기가 잘못 학습한 data를 다시 잘못 학습할 경우를 크게 반영한다.

5. ε은 base classifier의 가중치가 부여된 오류를 나타낸다. 가중치 $w_n^{(m)}$ 이 1/N으로 동일하다면 ε은 다음과 같다.

$$\varepsilon_m = \frac{1}{N}\sum_{n=1}^{N} I(y_m(x_n) \neq t_n)$$

6. α는 데이터에 부여하는 가중치를 의미한다. 따라서 ε가 작을수록 큰 α값이 다음 학습할 데이터 샘플의 새로운 가중치로서 적용된다. α값은 $\frac{1}{2}ln(\frac{1-\varepsilon_i}{\varepsilon_i})$ 도 쓸 수 있다.

7. weight update 수식은 다음과 같이 표현될 수 있다.

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } y_m(x_n) = t_n \\ e^{\alpha_i} & \text{if } y_m(x_n) \neq t_n \end{cases}$$

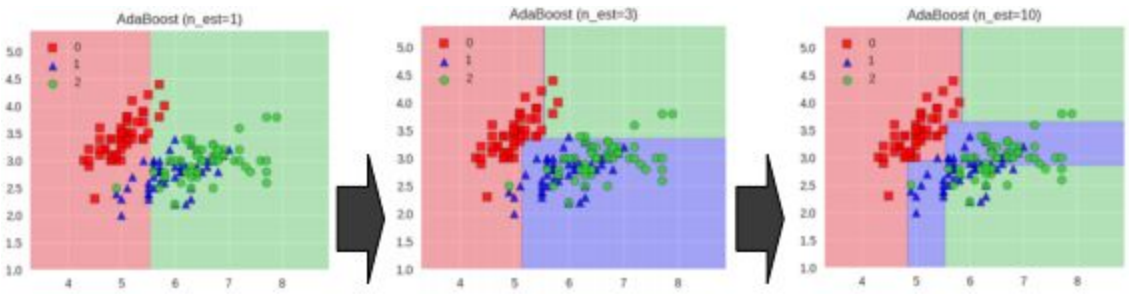
where Z_i is normalized factor used to ensure that $\sum_m w_m^{(n+1)} = 1$.

8. End for loop
9. 최종 모형은 다음과 같이도 쓸 수 있다.

$$Y_M(x) = \text{argmax}_{\,t} \sum_{m=1}^{M} \alpha_m I(y_m(x) = t)$$

10. 이러한 과정의 반복을 통해 잘못 분류한 데이터 샘플에 관하여는 가중치가 증가하고 바르게 분석한 데이터 샘플에 대해서는 가중치가 감소한다.

2.2 Decision tree 모델을 이용한 Ada Boosting 예시



위의 경우에 사용된 기초 학습자는 depth=1인 Decision tree이다. Boosting 기법을 통해 각 단계에서 오분류된 부분에 집중하여 반복하여 데이터 샘플을 재구성하여 Decision tree 알고리즘을 적용한다.

3. Gradient Boosting

Gradient Boosting은 회귀분석 또는 분류 분석을 수행할 수 있는 예측 모형이며, 앙상블 방법 중 부스팅에 속한다. Tabular format 데이터에 대해 예측에서 엄청난 성능을 보여준다. LightGBM, CatBoost, XGBoost 등이 모두 Gradient Boosting 알고리즘을 구현하였다.

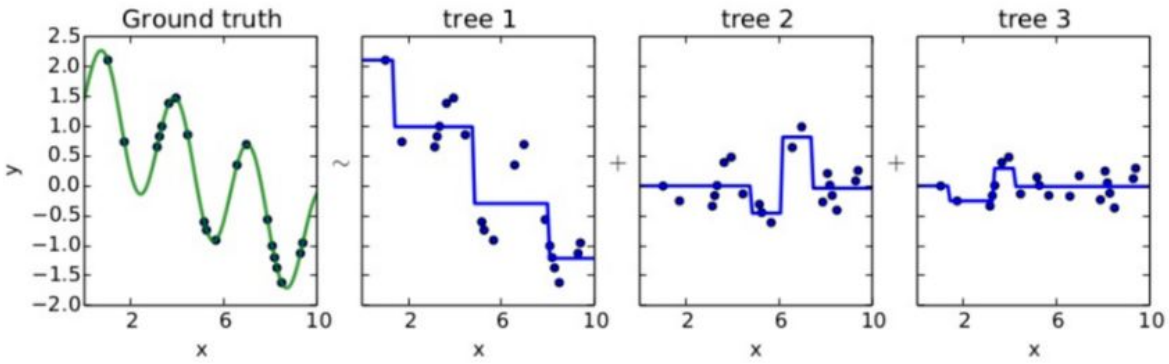
이 기법은 경사하강법(Gradient descent)을 사용해서 Ada Boosting보다 성능을 개선하였다. Ada Boosting은 높은 가중치를 가진 -잘못 분류될 가능성이 큰- 데이터(outlier)가 존재하게 되면 성능이 크게 떨어질 수 있는 단점이 있다. 이러한 포인트 근처의 다른 포인트들이 의도하지 않게 잘못된 분류로 될 가능성이 높기 때문이다. 따라서 Gradient Boosting기법에서는 가중치를 계산하는 방식에서 경사하강법을 통해 오류를 최소화하는 최적화 문제로 해법을 찾는다.

Gradient Boosting은 과적합 가능성이 여전히 존재하고 Bias와 Variance를 동시에 줄일 수 있다. Outlier에 덜 민감하게 반응하므로 attribute(variable)의 scale을 조정하지 않아도 된다. 또한 categorical data가 아닌 연속적인(continuous) attribute에서도 잘 작동한다. Regression, classification 모두 적용가능하나 loss function은 달라진다. Ada Boosting보다 학습 시간이 오래 걸리기에 경사하강법 적용시 적절한 변수 조정이 필요하다. 고차원 데이터에서는 잘 작동하지 않을 수 있다. Gradient descent의 단점인 local optimal에 갇힐 위험이 존재한다.

3.1 Gradient Boosting 알고리즘 간단 해석

이 파트는 [Deep Play] https://3months.tistory.com/368를 많이 참고하였습니다.

A 모델을 적합시킨 뒤 residual을 다시 B 모델로 예측하고 A+B 모델을 통해 y를 예측하면 A보다 나은 B 모델 생성 가능한 원리와 비슷하다. 아래의 plot을 보자.



https://3months.tistory.com/368

tree1을 통해 target value를 예측하고 남은 잔차를 tree2로 예측한다. tree2에서 남은 잔차를 tree3를 통해 다시 예측한다. tree1에서 tree3로 갈수록 잔차가 줄어들고 있는 것을 확인할 수 있다. 이 때 tree 1, 2, 3에서의 모델을 약분류기로 하고 이를 결합한 최종 모델을 강분류기라고 한다. 보통 약분류기는 층이 얕은 간단한 decition tree를 많이 사용하며 이를 Gradient Boosting tree라고도 한다.

이런 방식을 사용하면 잔차는 분류기가 추가될 때마다 계속 감소하게 된다. 즉 training set을 잘 설명하는 모형을 만들 수 있게 된다. 하지만 이런 방식으로는 Bias는 줄일 수 있어도, 과적합이 일어날 수 있는 단점이 있다. 따라서 Gradient Boosting을 사용할 시 sampling, penalizing 등의 regularization 기술을 이용하여 advanced 된 모델을 이용하는 것이 보편적이다.

Gradient Boosting의 기본적인 알고리즘은 기존 분류기에서 발생한 오차를 손실함수로 표현하여 경사하강법으로 손실함수를 최소화하는 방향으로 최적화하여 최적의 분류기를 찾아내는 것이다. 즉 현재의 $F_{m-1}(x)$가 주어졌을 때, 경사하강법을 통해 손실함수 L을 최소화하는 Decision tree h(x)를 찾고 그때의 개선 모델은 $F_m(x)$를 생성한다. 이 과정을 반복하여 최종 모델을 도출한다.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

$$F_m(x) = F_{m-1}(x) + argmin_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$$

좀 더 자세히 들어가보자. Loss function을 squared error로 설정할 시 residual은 negative gradient가 된다. 따라서 residual에 fitting해서 다음 모델을 순차적으로 만들어 가는 것은 negative gradient를 이용해 다음 모델을 순차적으로 만들어 나가는 것으로 볼 수 있다.

Residual = loss function을 squared error로 설정할 시 negative gradient이다. 따라서 residual에 fitting해서 다음 모델을 순차적으로 만들어 나가는 것은 negative gradient를 이용해 다음 모델을 순차적으로 만들어 나가기에 Gradient Boosting이라고 할 수 있다.

$$\text{Loss function} : L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

$$\text{Negative gradient} : \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial [\frac{1}{2}(y_i - f(x_i))^2]}{\partial f(x_i)} = \text{residual}$$

만약 regression이 아닌 classification 문제일 때는 위와 다른 loss function이 필요하다. 하지만 이러한 경우에도 negative gradient를 이용하여 새로운 model을 적합하고 이를 합산해나가는 식으로 최종 모델을 만들게 된다. 결국 negative gradient는 loss function이 minimize하기 위한

방향을 의미한다. 이 방향으로 새로운 모형을 fitting하여 이전의 것과 결합하면 loss function이 감소하는 방향으로의 새 모형이 업데이트 된다.

아래는 Gradient boosting의 pseudo algorithm이다.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

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

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

https://en.wikipedia.org/wiki/Gradient_boosting

1. $F_0(x)$ = Baseline model
2. Pseudo-residuals가 negative gradient라는 것을 확인할 수 있다.
3. 이에 대해 h(x)라는 모델을 적합시킨다.
4. $\gamma_m(x)$ 를 구하는 절차는 생략되는 경우도 있다. 이 값은 주로 매우 작은 값으로 Learning rate, 즉 학습속도에 해당한다.
5. 그 후 모델을 업데이트한다. Learning rate를 통해서 pseudo-residual에 fitting된 모형을 얼마나 반영할 것인가를 정하게 된다.

Loss 최소화 하는 값을 optimization으로 풀거나, 0.001~0.01 정도로 매우 작은 값으로 설정하는 것이 보편적이다.

4. XG Boosting

XG Boosting은 Extreme Gradient Boosting의 약자로 기존의 Gradient Boosting의 속도문제를 해결 하기 위해 전산속도와 모델의 성능에 초점을 맞춘 기법이다. Boosting model답게 과적합이 잘 일어나지 않으며, 다른 알고리즘과의 연계성이 좋다. 최근의 앙상블 모델중에서 가장 우수한 알고리즘으로 평가받아 각종 대회에서 사용된다.

XGBoost 역시 Boosting의 일종으로, 이전 모델의 오류를 고려하여 Bias를 감소시키고 다음 classifier에 그를 반영하는 방식을 사용한다. XGBoost는 그 중에서도 특히 Gradient Boosting Machine 대비 속도와 성능의 비약적인 향상이 이루어졌고, 또한 CPU와 같은 시스템 자원을 효율적으로 활용할 수 있는 알고리즘이다. 좀 더 구체적으로는 Gradient Boosting에 분산/병렬 처리가 추가된 알고리즘이라 볼 수 있다.

가장 큰 장점으로는 우선 다른 Gradient boosting 모델들의 실행에 비해 무척이나 빠르다는 것이다. 또한 구조화된 데이터셋에서 분류, 회귀 예측모형을 만드는 문제에 적절하게 활용될 수 있다. XGBoost는 분류기를 발견하고, 분산처리를 사용하여 빠른 속도로 적합한 비중 파라미터를 찾는다. 분류기는 Regression Score를 사용하여 accuracy score를 측정하고, 각 순서에 따라 강한 분류기부터 약한 분류기까지 랜덤하게 생성된다고 한다. 이렇게 만들어진 분류기를 Tree라 하고, 분류기를 조합한 최종 알고리즘을 Forest라고 한다. XGBoost는 트리를 만들 때 CART Ensemble model을 활용한다.

Reference

https://www.youtube.com/watch?v=GM3CDQfQ4sw&feature=youtu.be

https://blog.naver.com/euleekwon?Redirect=Log&logNo=221377873711

https://ko.wikipedia.org/wiki/에이다부스트

https://3months.tistory.com/368