

# PNU STAT - ML & Tree Model (12.26)

머신러닝, 딥러닝의 최종목표??

어떨 때 머신러닝을 쓸 수 있는가?

머신러닝에 대한 오해

왜 많고 많은 모델 중 Tree Model?

의사결정나무 Remind

Ensemble

RandomForest

Boosting - Adaboost(Adaptive Boost)

Boosting - GBM(Gradient Boosting Machine)

What is gradient?

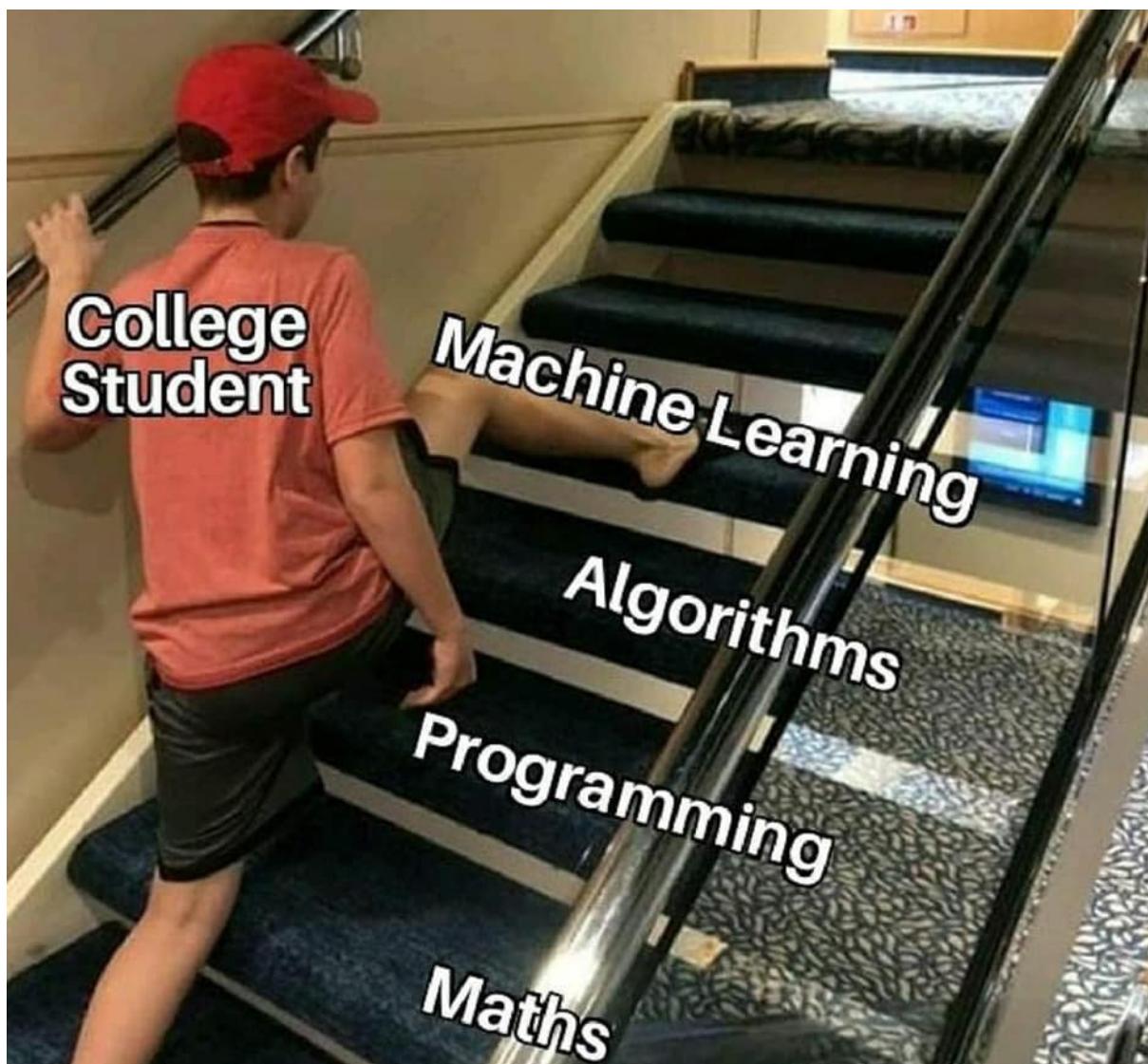
Boosting - XgBoost(eXtreme Boosting)

Boosting - LightGBM

Boosting - Catboost(unbiased boosting with categorical features, 2017)

여러분들한테 당부의 말씀

Reference



# 머신러닝, 딥러닝의 최종목표??

이안 굿펠로우

미국 연구원

출생: 미국

논문: Deep Learning of Representations and its Application to Computer Vision (2014)

분야: 컴퓨터 과학

저서: Deep Learning Adaptive Computation and Machine Learning

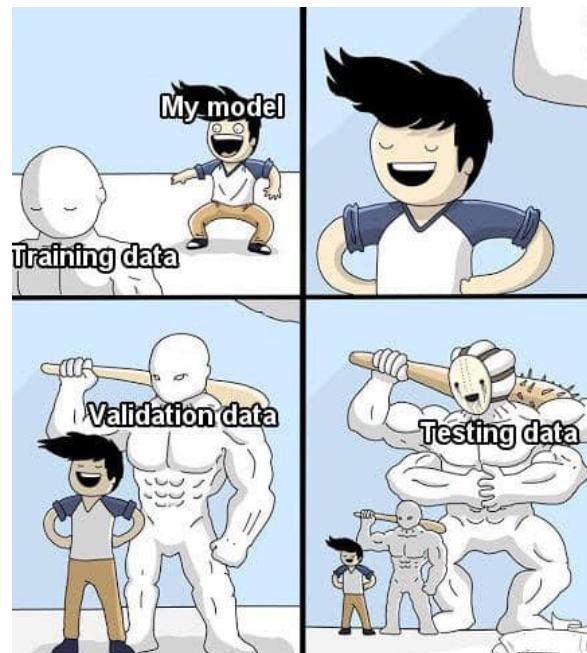
학력: 스탠포드 대학, 몬트리올 대학교

지도 교수: 오슈아 벤지오

Regularization도 중요한 개념이지만!

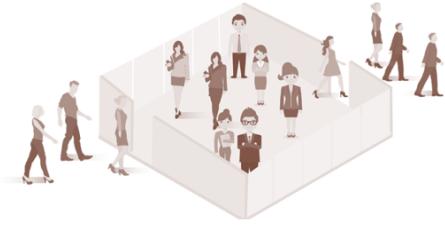
"Regularization is any modification we make to a learning algorithm that is intended **to reduce its generalization error but not its training error.**"  
[Ian Goodfellow et al., 2016]

**Generalization : Having good prediction on unseen data.**



$$\begin{aligned}
 \text{MSE}(\hat{\theta}) &= \mathbb{E}[(\hat{\theta} - \theta)^2] \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta\right)^2\right] \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2 + 2(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta) + (\mathbb{E}[\hat{\theta}] - \theta)^2\right] \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + \mathbb{E}[2(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta)] + \mathbb{E}\left[(\mathbb{E}[\hat{\theta}] - \theta)^2\right] \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + 2(\mathbb{E}[\hat{\theta}] - \theta) \mathbb{E}[\hat{\theta}] + (\mathbb{E}[\hat{\theta}] - \theta)^2 \quad \mathbb{E}[\hat{\theta}] - \theta = \text{const.} \\
 &\quad \text{linearity: } \mathbb{E}[\hat{\theta}] = \mathbb{E}[\mathbb{E}[\hat{\theta}]] \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + 2(\mathbb{E}[\hat{\theta}] - \theta)(\mathbb{E}[\hat{\theta}] - \mathbb{E}[\hat{\theta}]) + (\mathbb{E}[\hat{\theta}] - \theta)^2 \quad \mathbb{E}[\hat{\theta}] = \text{const.} \\
 &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + (\mathbb{E}[\hat{\theta}] - \theta)^2 \\
 &= \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta}, \theta)^2
 \end{aligned}$$

## 어떨 때 머신러닝을 쓸 수 있는가?

<p>누가 TM 퀄을 잘 받을 것인가?</p> 	<p>누가 이탈할 가능성이 높은가?</p> 
<p>질병고지에 따른 할증률을 어떻게 운영할 것인가?</p> 	<p>어떤 계약을 인수 또는 거절할 것인가?</p> 

## 머신러닝에 대한 오해

머신러닝은 컴퓨터가 경험에서부터 자동으로 몰랐던 사실을 찾아낸다

컴퓨터가 데이터에서부터 자동으로 몰랐던 사실을 찾아낸다

컴퓨터가 데이터에서부터 학습에 의해 몰랐던 사실을 찾아낸다

컴퓨터가 데이터에서부터 학습에 의해 내재된 규칙을 찾아낸다

사람이 이해하기 어려운

사람이 학습하기 어려운

사람이 추출하기 어려운

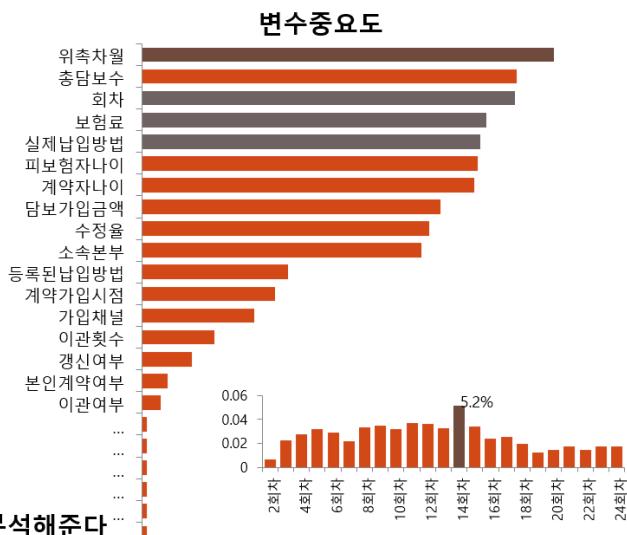
머신러닝을 돌렸는데 새로운 사실이 안나와요...

누가 이탈(탈락)할 가능성이 높은가?



컴퓨터가 데이터에서부터 ...

→ 우리가 넣은 변수에 대한 결과만을 분석해준다



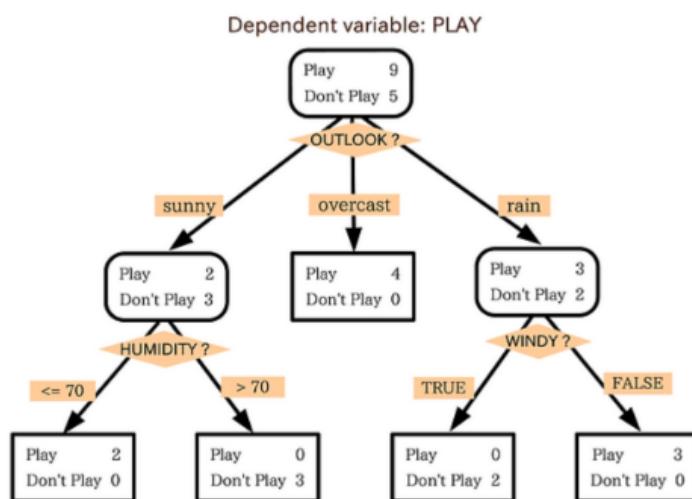
## 왜 많고 많은 모델 중 Tree Model?

1. 정형 데이터에서 기가 막히게 잘 맞추는 모델이 Tree Model!!, 해석하기도 좋고 **baseline** 모델로 접근 했을 때에도 성능이 괜찮음
2. 머신러닝 모델들은 기본적으로 독립변수들을 서로 독립적이라고 가정하고 접근
3. 어떠한 특별한 가정을 세운다 → 가정을 기반으로 여러 변수를 만든다 → 그 모델이 성능에 많은 기여를 한다 → 해석하기에 용이
4. 딥러닝 딥러닝 노래를 부르는 시대!(확실히 부흥기) 하지만, 캐글이나 머신러닝 대회에서는 **boosting** 계열 알고리즘이 석권!
5. 딥러닝 모델은 층도 많고, 오버피팅도 심하고 학습 시간 오래걸리고 장비도 좋아야하고.... (가난한 학생인데...)



## 의사결정나무 Remind

- <https://ratsgo.github.io/machine-learning/2017/03/26/tree/>
- 주요 특징
  1. 변수들로 기준("수 많은 변수 중에 ??에 가장 영향을 주는 변수가 무엇일까?)을 만들고, 이것을 통하여 샘플을 분류하고 분류하고 집단의 성질을 통해 추정하는 모형
  2. 의사결정나무는 규칙들을 표현 → 규칙은 문장 형태로 표현될 수 있다.(and 조건을 이어나가)
  3. 스무고개 같음(예/아니오의 연속 질문 → 처음 질문의 답에 따라서 다음 질문이 달라짐)



- 장단점

- 장점 1 : 신경망, 판별분석, 회귀분석 보다 이해하기 쉽고(해석하기 쉽고) 설명력이 높다, 직관적
  - 장점 2 : 자료 가공이 조금 상대적으로 적음
  - 장점 3 : 비선형적인 관계를 설명 가능(교호작용 등이 반영)
  - 장점 4 : Target이 연속형, 범주형 모두 설명 가능
  - 단점 : 변동성이 너무 큼, 샘플에 민감
- 

- 구성요소는 다 아시죠? → <https://dreamlog.tistory.com/576>
- 어떻게 Fitting? (비용복잡도Cost Complexity를 기준으로 가지치기)
  - 주요 알고리즘 및 지표

- **Gini impurity(지니 불순도).**

- CART 알고리즘에서 활용
    - 특정 그룹에 이질적인 것이 얼마나 섞였는지를 계산하는 지표
    - 0에 가까울 수록 순도가 높음

$$I_G(p) = \sum_{i=1}^J p_i \sum_{k \neq i} p_k = \sum_{i=1}^J p_i (1 - p_i) = \sum_{i=1}^J (p_i - p_i^2) = \sum_{i=1}^J p_i - \sum_{i=1}^J p_i^2 = 1 - \sum_{i=1}^J p_i^2$$

- 
- Entropy, Information Gain(의사결정나무에서의 특정 노드 이전과 이후의 엔트로피 차이)
  - 엔트로피
    - 고등학교 과학시간에 '무질서의 정도', 골고루 섞여 있으면 높다.
    - 정보이론에서는 정보량의 크기(해가 서쪽에서 뜬다)

$$H(T) = I_E(p_1, p_2, \dots, p_J) = - \sum_{i=1}^J p_i \log_2 p_i$$

- Information Gain(정보 획득량) : 특정 그룹의 엔트로피와 분할된 하위 그룹의 엔트로피 차이를 계산
  - IG 값이 클 수록 '변별력이 좋다'(지정된 속성이 얼마나 잘 sample간을 구분하는가에 대한 수치)
  - 특정 변수를 기준으로 sample을 구분할 때 '감소되는 엔트로피의 양'

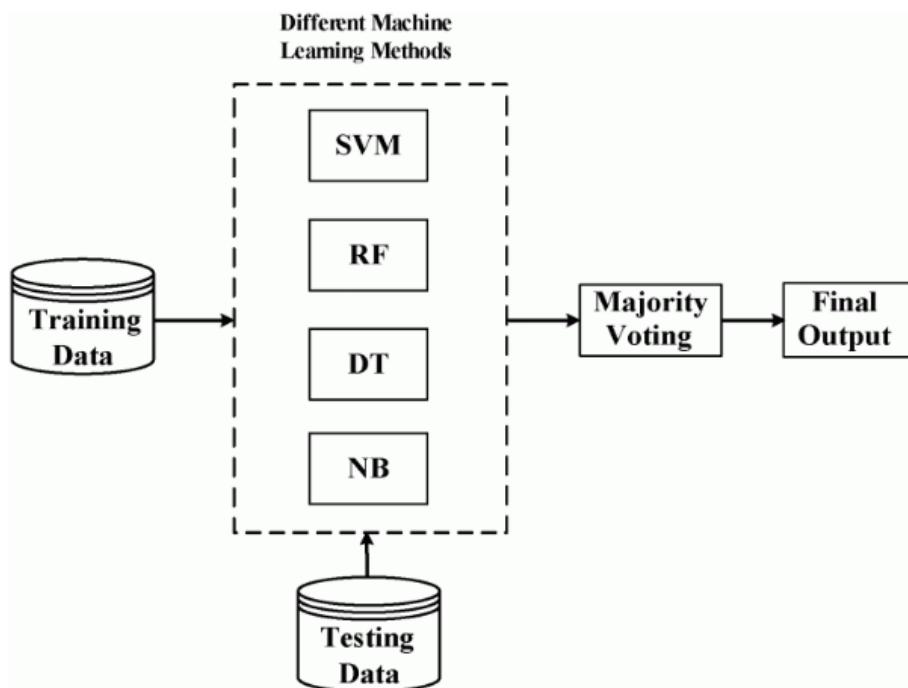
$$\begin{aligned} \text{Information Gain} &= \overbrace{IG(T, a)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}} \\ &= - \sum_{i=1}^J p_i \log_2 p_i - \sum_a p(a) \sum_{i=1}^J -Pr(i|a) \log_2 Pr(i|a) \end{aligned}$$



- R packages : **rpart**, party, **caret**, C50, randomForest, xgboost

## Ensemble

- Ensemble : '조화'라는 의미
- Ensemble Learning : 여러개의 기본 모델(weak learner, classifier 등)을 활용하여 하나의 새로운 모델을 만들어 내는 개념

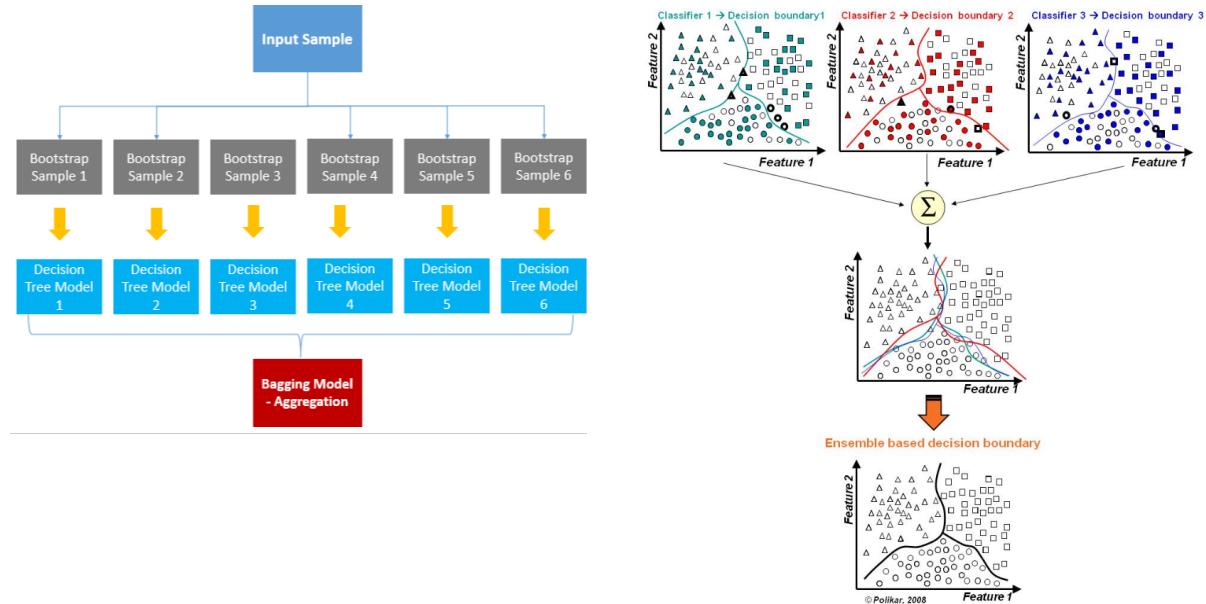


- Ensemble Learning 종류
  - **Bagging** : 모델을 다양하게 만들기 위해 데이터를 재구성(모든 변수 사용, 샘플링)
  - **RandomForest** : 배깅이랑 산출방식은 같은데 변수도 샘플링함
  - **Boosting** : 맞추기 어려운 데이터에 대해 좀더 가중치를 두어 학습하는 개념(오답노트)
    - 경험상 : **LightGBM > Catboost > XgBoost > GBM > AdaBoost**

- **Stacking** : 모델의 output값을 새로운 독립변수로 사용(PCA loading, SVM, KNN Etc)

### Bagging(Bootstrap AGGREGATING)

- 복원추출로 데이터 추출하여 여러 분류기를 만들어 결과값을 평균으로 산출
- 깊이 들어간 의사결정나무의 단점 : 분산 증가, 편향 감소
- **Bagging 장점** : Tree들의 편향 유지, 분산 감소
- **Bagging 단점**(모든 양상별 모형의 단점) : 모형 해석 어려움



### RandomForest

- Bagging Model의 분산은 각각 트리들의 분산과 트리들의 공분산으로 이루어져 있음
- 전체 데이터에서 복원 추출했으나, 각각의 트리들은 중복되는 데이터를 다수 가지고 있기 때문에 **독립이라는 보장**이 없다  $\rightarrow 2\text{Cov}(X,Y) \neq 0$

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$$

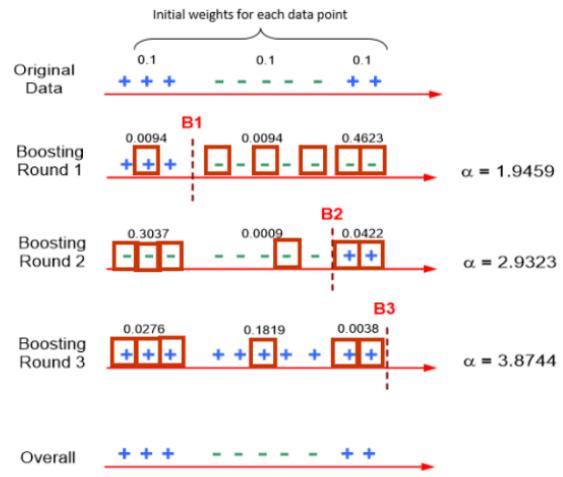
- Tree가 증가함에 따라 오히려 모델 전체의 분산이 증가 할 수 있다.  $\rightarrow$  각 분류기간 공분산을 줄일 방법이 필요
- RF의 핵심은 **데이터 샘플링 + 변수도 Random하게( $\sqrt{p}$ 개)** 추출해 분류기를 생성함  $\rightarrow$  **모델간의 공분산 감소시키는 것이 컨셉**
- 일반적으로 뽑을 변수의 수는  **$\sqrt{p}$**  사용  $\rightarrow$  모델의 분산을 줄여 일반적으로 Bagging보다 성능이 좋음

### Boosting - Adaboost(Adaptive Boost)

- (공통) **오분류된 데이터에 초점을 맞추어 더 많은 가중치를 주는 방식**
  - 공통이라고 하면 AdaBoost, GBM, LogitBoost, TotalBoost, LPBoost, MadaBoost .... ETC

- 초기에는 모든 데이터가 동일한 가중치 → **Round** 종료 후 가중치와 중요도 계산
  - 복원추출시에 데이터 가중치 분포 고려**
  - 오분류된 데이터에 Weight를 크게, 정분류된 데이터에는 Weight를 작게 설정 → Weight별로 샘플을 재수집하며 계속 Weight를 업데이트(Round별로)!
- 

- ① 모든 데이터에 대해 가중치를 동일하게 0.1로 설정
- ② Round 1에서 빨간색 네모의 데이터가 수집되며 이를 기반으로 분류 기준값인 B1을 설정(B1보다 작은 경우 +, 큰 경우 -로 분류)
- ③ 데이터  $i$ 에 대해  $m$ 번째 round에서의 가중치를 업데이트 (오분류된 데이터에 가중치를 크게, 정분류된 데이터에 가중치를 작게 설정)
- ④ 업데이트한 가중치의 확률로 샘플을 재수집
- ⑤ 4번의 결과로 Round 2의 빨간색 네모의 데이터가 수집
- ⑥ 수집된 데이터로 모형을 학습한 결과 B2가 분류 기준값으로 도출됨
- ⑦ 가중치를 업데이트
- ⑧ 이와 같은 방법을 설정한 반복 횟수만큼 반복
- ⑨ 예시에서 Round 3까지의 결과를 종합하면 Original Data 와 동일한 결과가 도출됨



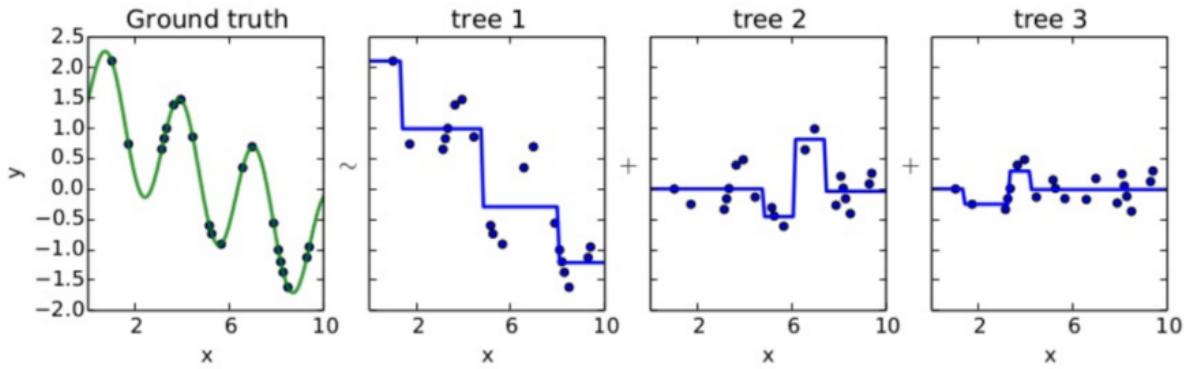
- 최종 모델 결정 방법

$$H(x) = \text{sign}\left\{\sum_{m=1}^M \alpha_m h_m(x)\right\}$$

- H(x)** : 최종 분류기(final classifier)
  - h\_m** : m라운드에서 생성된 약한 분류기
  - alpha\_m** : m라운드에서 생성된 약한 분류기에 대한 가중치(크면 epsilon\_m이 작다는 의미 → 분류기 m이 좋은 성능을 보임)
- 

## Boosting - GBM(Gradient Boosting Machine)

- Boosting 기법들끼리 큰 차이점 : 오분류된 데이터를 다음 **Round**에 무슨 짓을 할까??? (AdaBoost는 오분류된 데이터들에 더 큰 가중치를 주어서 샘플링에 힘을 주었음)
- residual**에 대해 계속 학습해 나가면서 error를 최소화(<https://3months.tistory.com/368>)



$x$ 를 입력받아  $y$ 를 예측하는 모델  $h_0$ 가 있다고 하면,

$$y = h_0(x) + \text{error}$$

$$\text{error} = h_1(x) + \text{error2}$$

$$\text{error2} = h_2(x) + \text{error3}$$

$$\text{error3} = h_3(x) + \text{error4}$$

(고등학교 수열시간에 많이 보신 형태!!)

$$y = h_0(x) + h_1(x) + h_2(x) + \dots + \text{smaller error}$$

- Error를 많이 줄여보자~

## What is gradient?

- Define loss func

$$L(y_i, f(x_i)) = 0.5 * (y_i - f(x_i))^2$$

- Loss func의 gradient

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

- Negative gradient = Residual

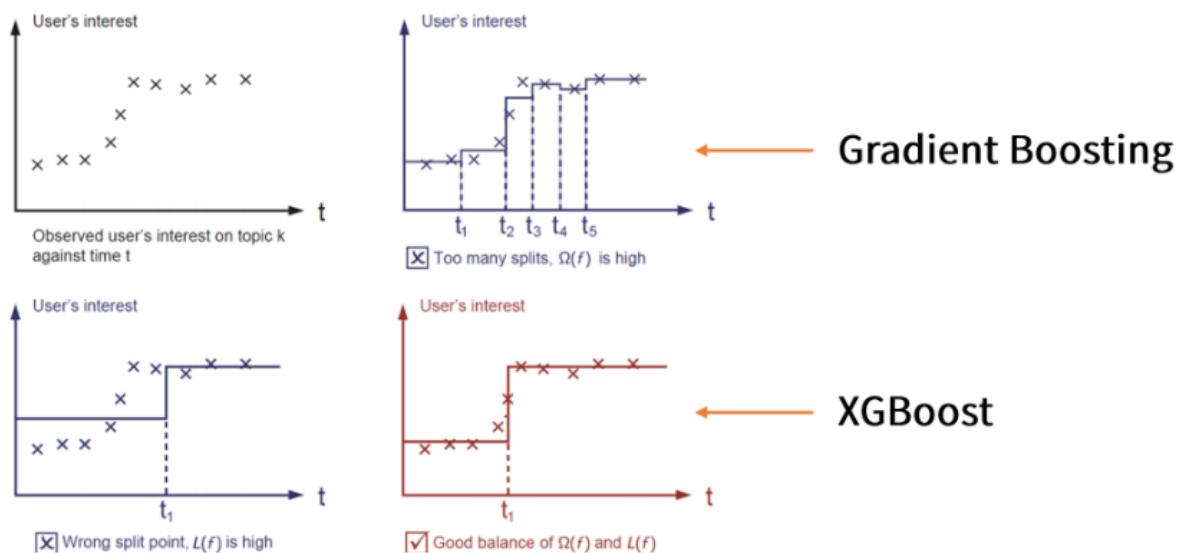
$$-(f(x_i) - y_i) = y_i - f(x_i)$$

- Neagative gradient를 최소화 시키면서 학습 시키기 때문에 gradient boosting이라 부름

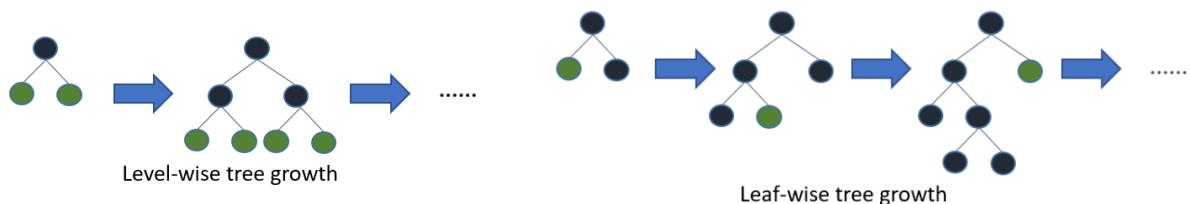
## Boosting - XgBoost(eXtreme Boosting)

- 기존 GBM이랑 큰 차이 없음. 단지 **목적식에 Regularization Term**이 추가됨(Ridge, Lasso와 같은) → **오버피팅을 잡아주자!!!**
- 이 알고리즘 때문에 옛날 캐글에서 꽤나 우수한 성적이 나옴
- 왼쪽 Term : Loss Function
- 오른쪽 Term : Regularization

$$obj^{(t)} = \sum_{i=1}^n l(Y_i, \hat{Y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i)$$



## Boosting - LightGBM



- 넘 좋고 달달~!! **동일한 hyperparameter**에서 Xgboost보다 약 2배 이상 빠름
- 이전 boosting기법은 **균형 트리 분할(level wise)** → LGBM은 **리프 중심 트리 분할(leaf wise)**가 큰 컨셉
- 공식 문서에서 데이터 **최소 10000개 이상 권장**(오버피팅이 끝내주게 잘 일어남), 학습데이터 많을 수록 좋다.
- gpu애매하면 **cpu로 돌리는 거 권장(경험상)**
- **h2o에서 xgboost에서 옵션 조금만 바꿔서 lgbm을 사용. (tree\_method, grow\_policy)**

[https://github.com/2econsulting/2econsulting.github.io/blob/master/\\_posts\\_w\\_code/GBMvsLightGBM\\_R.r](https://github.com/2econsulting/2econsulting.github.io/blob/master/_posts_w_code/GBMvsLightGBM_R.r)

```
# In h2o, there is no specific module for LightGBM yet.
# Instead, using "tree_method='hist'" and "grow_policy='lossguide'" in h2o.xgboost() offer "LightGBM algorithm" to us
# you can find detail information in h2o website => http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/xgboost.html
# parameter = default
start.time <- Sys.time()
ml_LGB <- h2o.xgboost( training_frame = train,
                        validation_frame = valid,
                        x=1:19,
                        y=20,
                        tree_method="hist",
                        grow_policy="lossguide",
                        seed = 1234)
```

## Boosting - Catboost(unbiased boosting with categorical features, 2017)

- 잔차 추정의 분산을 최소로 하면서 bias도 피하는 기법
- 범주형 변수가 많으면 꽤나 좋은 효과를 발휘함(캐글에서 종종보임)
- 관측치를 포함한 채로 boosting하지말고, 관측치를 뺀채로 학습 → 그 관측치에 대한 unbiased residual을 구하고 학습하는 컨셉
- 범주형 변수를 One-hot으로 바꾸는 방식이 아닌, **Numeric하게** 변환하는 방법 제안!
- 논문(<https://arxiv.org/abs/1706.09516>, Cat > LGBM > Xgboost)

Table 8: Comparison with baselines: logloss / zero-one loss, relative increase is presented in the brackets.

	CatBoost	LightGBM	XGBoost
Adult	<b>0.2695 / 0.1267</b>	0.2760 (+2.4%) / 0.1291 (+1.9%)	0.2754 (+2.2%) / 0.1280 (+1.0%)
Amazon	<b>0.1394 / 0.0442</b>	0.1636 (+17%) / 0.0533 (+21%)	0.1633 (+17%) / 0.0532 (+21%)
Click	<b>0.3917 / 0.1561</b>	0.3963 (+1.2%) / 0.1580 (+1.2%)	0.3962 (+1.2%) / 0.1581 (+1.2%)
Epsilon	<b>0.2647 / 0.1086</b>	0.2703 (+1.5%) / 0.114 (+4.1%)	0.2993 (+11%) / 0.1276 (+1.2%)
Appetency	<b>0.0715 / 0.01768</b>	0.0718 (+0.4%) / 0.01772 (+0.2%)	0.0718 (+0.4%) / 0.01780 (+0.7%)
Churn	<b>0.2319 / 0.0719</b>	0.2320 (+0.1%) / 0.0723 (+0.6%)	0.2331 (+0.5%) / 0.0730 (+1.6%)
Internet	<b>0.2089 / 0.0937</b>	0.2231 (+6.8%) / 0.1017 (+8.6%)	0.2253 (+7.9%) / 0.1012 (+8.0%)
Upselling	<b>0.1662 / 0.0490</b>	0.1668 (+0.3%) / 0.0491 (+0.1%)	0.1663 (+0.04%) / 0.0492 (+0.3%)
Kick	<b>0.2855 / 0.0949</b>	0.2957 (+3.5%) / 0.0991 (+4.4%)	0.2946 (+3.2%) / 0.0988 (+4.1%)

Table 9: Comparison with baselines: logloss / zero-one loss (relative increase for baselines).

	Raw setting of CatBoost	LightGBM	XGBoost
Adult	0.2800 / 0.1288	-1.4% / +0.2%	-1.7% / -0.6%
Amazon	0.1631 / 0.0533	+0.3% / 0%	+0.1% / -0.2%
Click	0.3961 / 0.1581	+0.1% / -0.1%	0% / 0%
Appetency	0.0724 / 0.0179	-0.8% / -1.0%	-0.8% / -0.4%
Churn	0.2316 / 0.0718	+0.2% / +0.7%	+0.6% / +1.6%
Internet	0.2223 / 0.0993	+0.4% / +2.4%	+1.4% / +1.9%
Upselling	0.1679 / 0.0493	-0.7% / -0.4%	-1.0% / -0.2%
Kick	0.2955 / 0.0993	+0.1% / -0.4%	-0.3% / -0.2%
Average		-0.2% / +0.2%	-0.2% / +0.2%

## 여러분들한테 당부의 말씀

- 공모전(정형데이터)을 참가하실텐데 오늘 소개해준 모델로 **baseline** 잡아서 하시는 것을 추천합니다!
- 저는 많이 쓰이는 모델들을 소개해주었습니다. → 상을 펴서 **수저만 차린 수준** → 알아서 **공식문서** 보면서 해보는 것을 추천합니다. 공식문서랑 친해지세요!!!
- 방법론보다는 더 중요한 것은 **전처리 전처리 전처리**가 중요합니다. 실제로 여러분들이 하는 프로젝트에서 모델링은 5~10%입니다.
- 공모전(데이터, 캐글, 카카오아레나 등) **상위권 코드**를 보면서 여러분들이 하향식 공부법을 추천합니다.
- 혹시나 (머신러닝)논문에 쓰시는 데이터가 너무 안좋다면 한번 이걸로 실험해볼 수는 있겠습니다.
- **Stacking**이라고 또 다른 기법이 있는데 시간이 기가 막히게 걸리기 때문에 컴퓨터 사양이 자신 있는 분들은 따로 공부해보시는 거 추천합니다.

## Reference

Bias vs. Variance 개념 정리

[https://modulabs-biomedical.github.io/Bias\\_vs\\_Variance](https://modulabs-biomedical.github.io/Bias_vs_Variance)

결정 트리 학습법

[https://ko.wikipedia.org/wiki/%EA%B2%BD%EC%A0%95\\_%ED%8A%B8%EB%A6%AC\\_%ED%95%99%EC%8A%B5%EB%B2%95](https://ko.wikipedia.org/wiki/%EA%B2%BD%EC%A0%95_%ED%8A%B8%EB%A6%AC_%ED%95%99%EC%8A%B5%EB%B2%95)

의사결정나무(Decision Tree)

<https://ratsgo.github.io/machine%20learning/2017/03/26/tree/>

자동등록방지를 위해 보안절차를 거치고 있습니다.

<http://solarisailab.com/deep-learning-glossary>

Bagging과 Boosting 그리고 Stacking

<https://swalloow.github.io/bagging-boosting>