How to Win a Data Science Competition: learn from Top Kaggler

Week 4 - Hyperparameter tuning

Plan for the lecture

- Hyperparameter tuning in general
  - General pipeline 어떤 순서로,
  - Manual and automatic tuning 어떻게 튜닝하는지,
  - What should we understand about hyperparameters? 뭘 알아야 하는지,
- Models, libraries and hyperparameter optimization
  - Tree-based models 각 모델 별, 주요 하이퍼파라미터 소개
  - Neural networks
  - Linear models

How do we tune hyperparameters

1. **Select the most influential parameters**
   a. There are tons of parameters and we can't tune all of them

많은 **parameters** 중에서 중요한 것을 찾아내서 **subset** 구성 ← 캐글이나 깃헙 참고

2. **Understand, how exactly they influence the training**

어떤 파라미터를 증가시키면 **training** 은 어떻게 되고 **validation**은 어떻게 되는지

3. **Tune them!**
   a. Manually (change and examine)
   b. Automatically (hyperopt, etc.)

런 → 피드백 → 런 → 피드백 **(iterate)**

OR

소프트웨어 써도 됨, 그러나 메뉴얼한 방식이 더 빨리되고 좋다네요

Hyperparameter optimization software

- A lot of libraries to try:
  - *Hyperopt*
  - Scikit-optimize
  - Spearmint
  - GPyOpt
  - RoBO
  - SMAC3

```
def xgb_score(param):
    # run XGBoost with parameters `param`

def xgb_hyperopt():
    space = {
         'eta' : 0.01,
         'max_depth' :        hp.quniform('max_depth', 10, 30,1),
         'min_child_weight' : hp.quniform('min_child_weight', 0, 100, 1),
         'subsample' :        hp.quniform('subsample', 0.1, 1.0, 0.1),
         'gamma' :            hp.quniform('gamma', 0.0, 30, 0.5),
         'colsample_bytree' : hp.quniform('colsample_bytree', 0.1, 1.0, 0.1),

         'objective':'reg:linear',

         'nthread' : 28,
         'silent' : 1,
         'num_round' : 2500,
         'seed' : 2441,
         'early_stopping_rounds':100
         }


    best = fmin(xgb_score, space, algo=tpe.suggest, max_evals=1000)
```

Color-coding legend

1. Underfitting (bad)
2. **Good fit and generalization (good)**
3. Overfitting (bad)

- A parameter in red
  - *Increasing it impedes fitting*
  - Increase it to reduce overfitting
  - Decrease to allow model fit easier

A parameter in green
- *Increasing it leads to a better fit (overfit) on train set*
- Increase it, if model underfits
- Decrease if overfits

언더피팅되게 만듬

오버피팅되게 만듬

Thus 빨강과 초록색 그 사이 어느 '적절한' 지점을 찾아내는 것이 목표다

Plan for the video

- Tree-based models
  - GBDT: `XGBoost, LightGBM, CatBoost`
  - RandomForest/ExtraTrees
- Neural nets
  - `Pytorch, Tensorflow, Keras...`
- Linear models
  - SVM, logistic regression
  - `Vowpal Wabbit, FTRL`
- Factorization Machines (out of scope)
  - `libFM, libFFM`

Tree-based models

Model	Where
GBDT	*XGBoost* (dmlc/xgboost) *LightGBM* (Mictrosoft/LighGBM) *CatBoost* (catboost/catboost)
RandomForest, ExtraTrees	*scikit-learn*
Others	*RGF* (baidu/fast_rgf)

여기서 잠깐 ⏸

Bagging과 Boosting → 앙상블 기법

특징 비교

비교	Bagging	Boosting
특징	병렬 앙상블 모델 (각 모델은 서로 독립적)	연속 앙상블 (이전 모델의 오류를 고려)
목적	Variance 감소	Bias 감소
적합한 상황	복잡한 모델 (High variance, Low bias)	Low variance, High bias 모델
대표 알고리즘	Random Forest	Gradient Boosting, AdaBoost
Sampling	Random Sampling	Random Sampling with weight on error

https://www.slideshare.net/freepsw/boosting-bagging-vs-boosting

[참고] bias vs variance (1/4)

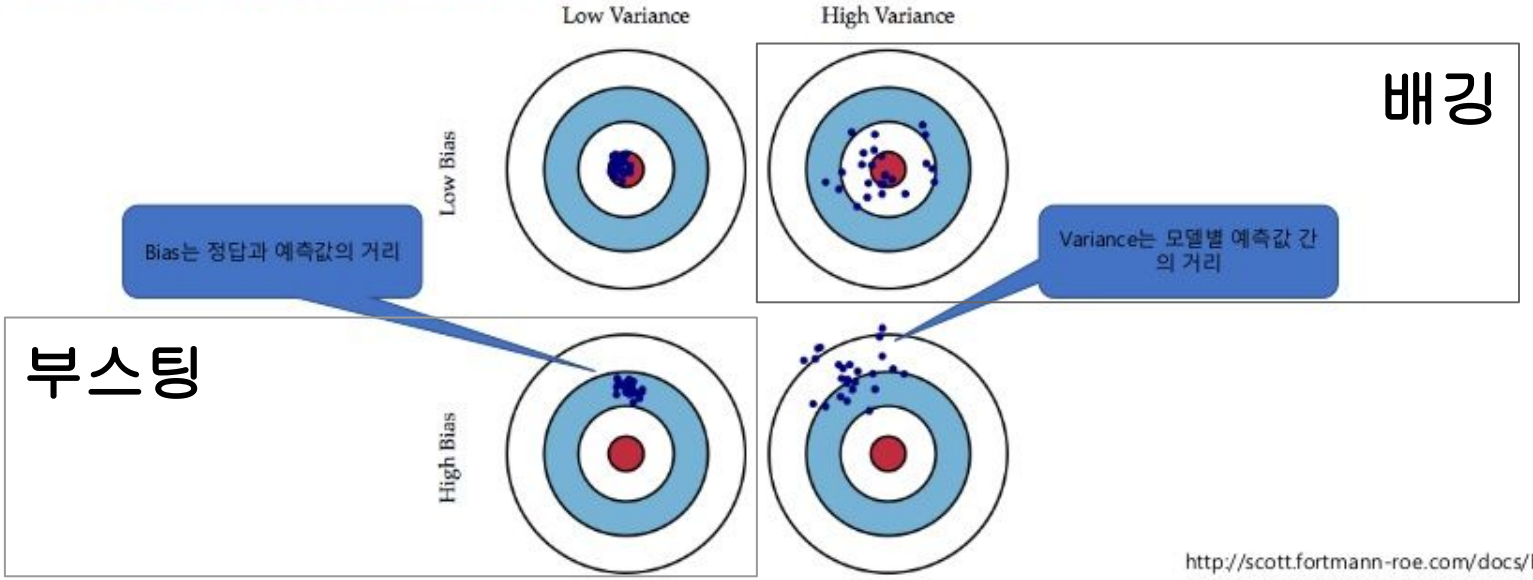
학습한 모델의 예측 오류는 크게 2개(Bias, Variance)오류로 이루어짐

Bias

[Error due to Bias]
- Bias로 인한 에러는 예측값과 실제값 간의 차이
- (당연한 소리.. 그럼 뭐가 다른가?)
- 모델 학습시 여러 데이터를 사용하고, 반복하여 새로운 모델로 학습하면, 예측값 들의 범위를 확인 할 수 있다.
- Bias는 이 예측값 들의 범위가 정답과 얼마나 멀리 있는지 측정

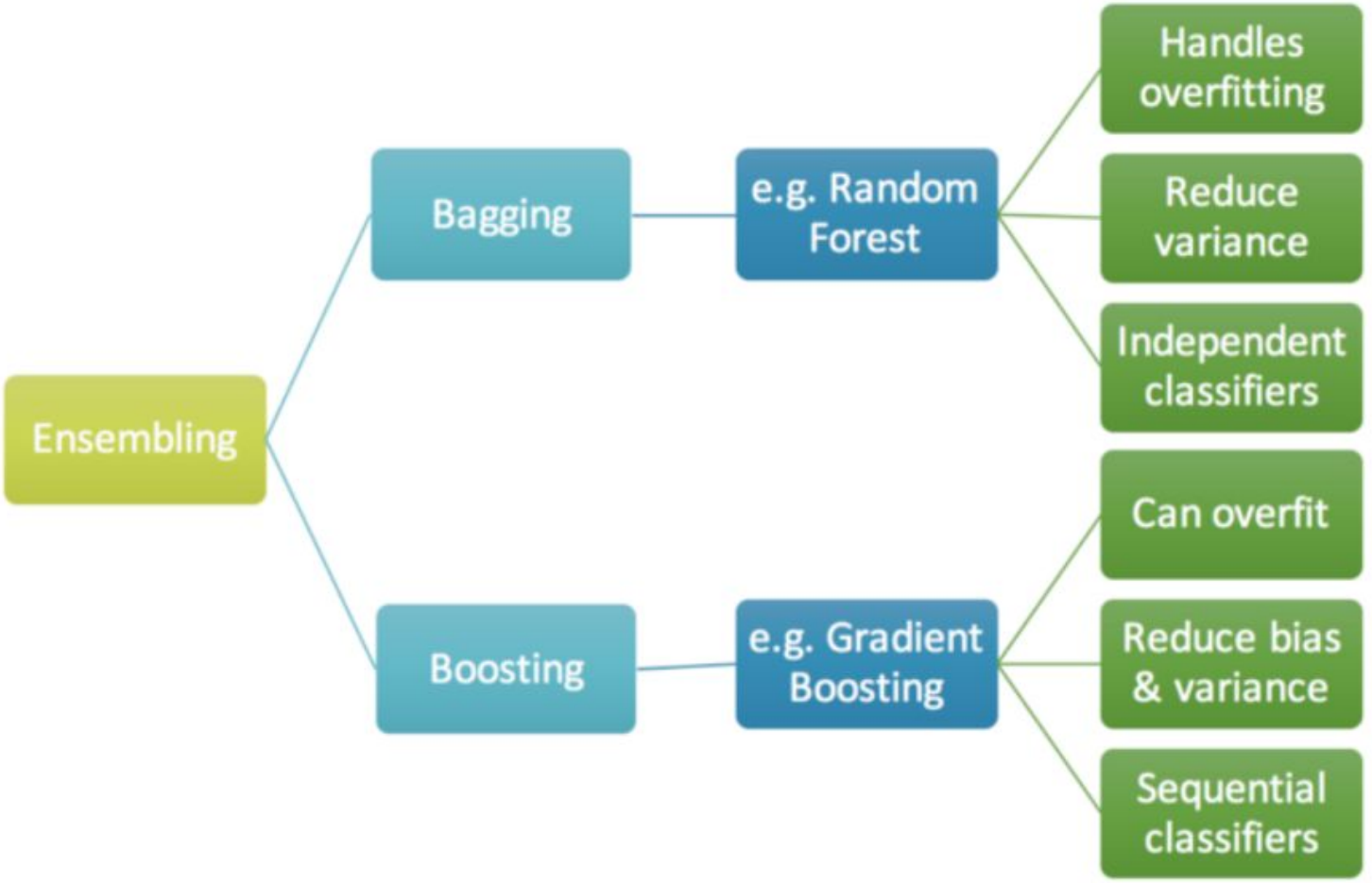
Variance

[Error due to Variance]
- 주어진 데이터로 학습한 모델이 예측한 값의 변동성 (분산, variance)
- 만약 여러 모델로 학습을 반복한다고 가정하면,
- Variance는 학습 된 모델별로 예측한 값들의 차이를 측정

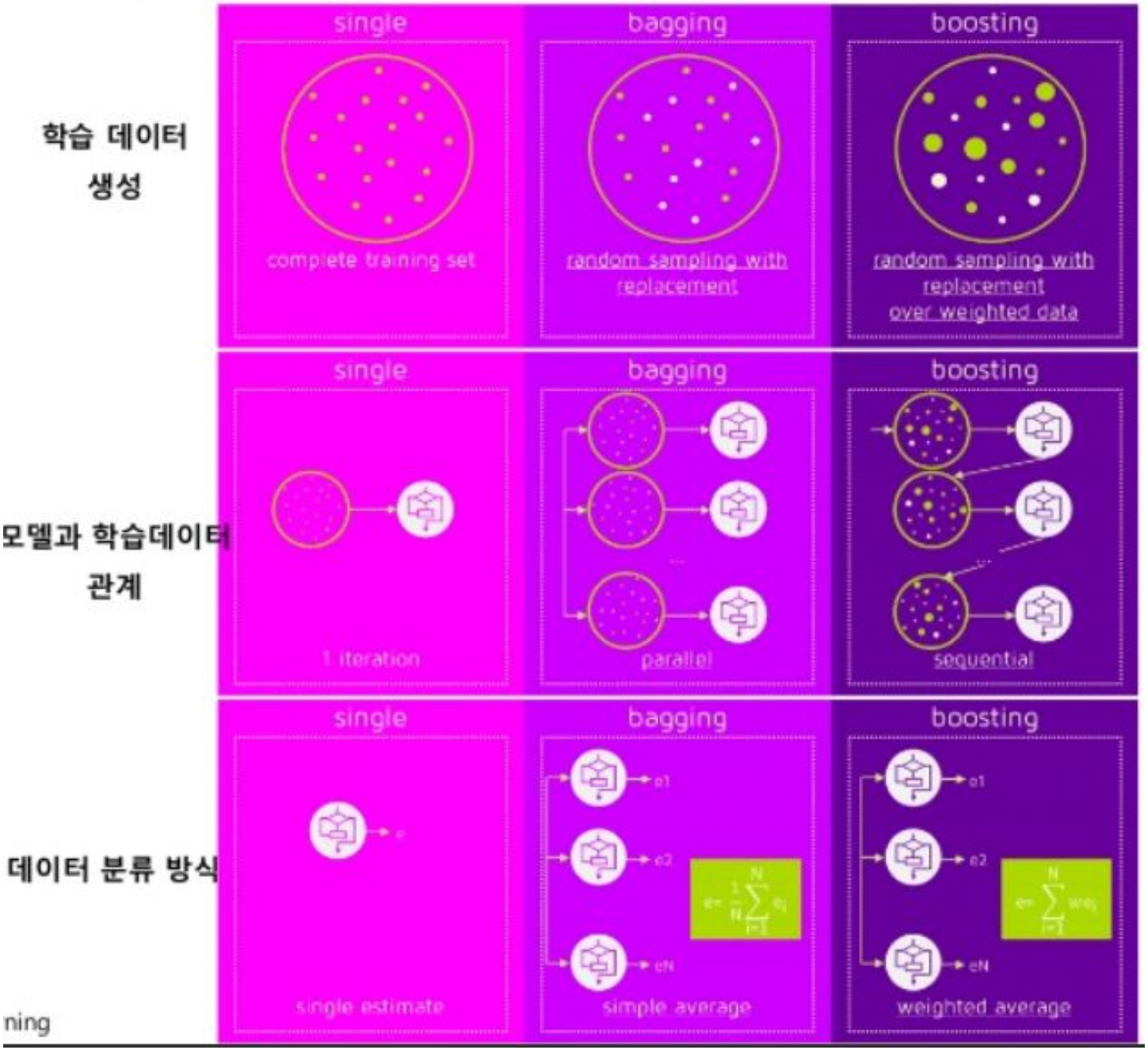


http://scott.fortmann-roe.com/docs/BiasVariance.html

Begging과 Boosting



핵심 개념 비교



https://www.slideshare.net/freepsw/boosting-bagging-vs-boosting

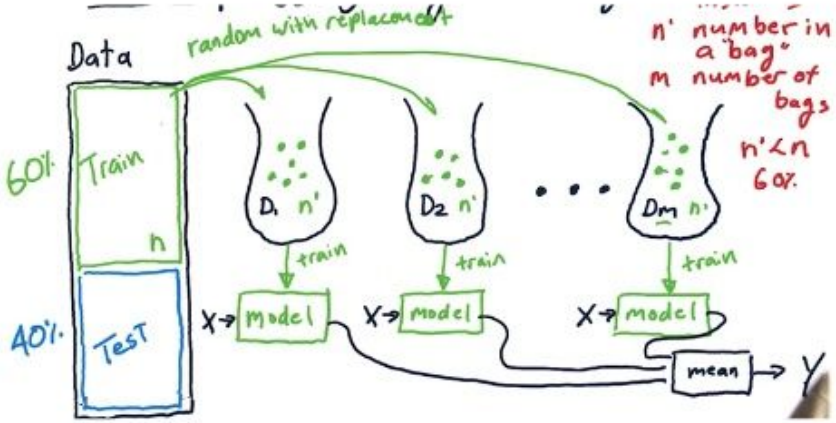
2. Bagging – Bootstrap aggregating

동일한 모델을 사용하고, 데이터만 분할하여 여러개 모델을 학습 (앙상블기법)

Bagging의 개념

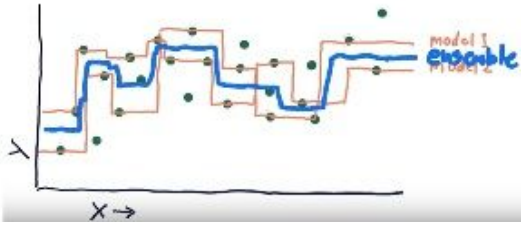
학습데이터를 랜덤으로 샘플링하여 여러개이 bag으로 분할하고, 각 bag별로 모델을 학습한 후, 각 결과를 합하여 최종 결과를 추출

- n : 전체 학습 데이터 수
- n' : bag에 포함된 데이터 수, 전체 데이터 중 샘플링된 데이터
- m : bag의 갯수, 학습할 모델별로 샘플링된 데이터 셋



어떻게 예측정확도를 높이나?

- 아래 그림에서 model1은 하나의 bag으로 학습된 모델이다.
- 각 모델별로 보면, 학습 데이터에 overfitting되어 테스트 데이터로 검증하면 예측성능이 낮다. (high variance)
- Bagging은 이렇게 weak model을 여러개 결합하여, 전체적으로 high variance → low variance로 변하면서 예측성능을 향상한다.
- 아래 그림을 보면 여러개 모델이 서로 보완하면서 예측한다.
- Bagging은 linear 모델에는 잘 사용하지 않는데, 굳이 데이터를 샘플링하여 여러개 모델을 만들 필요가 없다.
- 이미 더 좋은 linear모델이 있음.

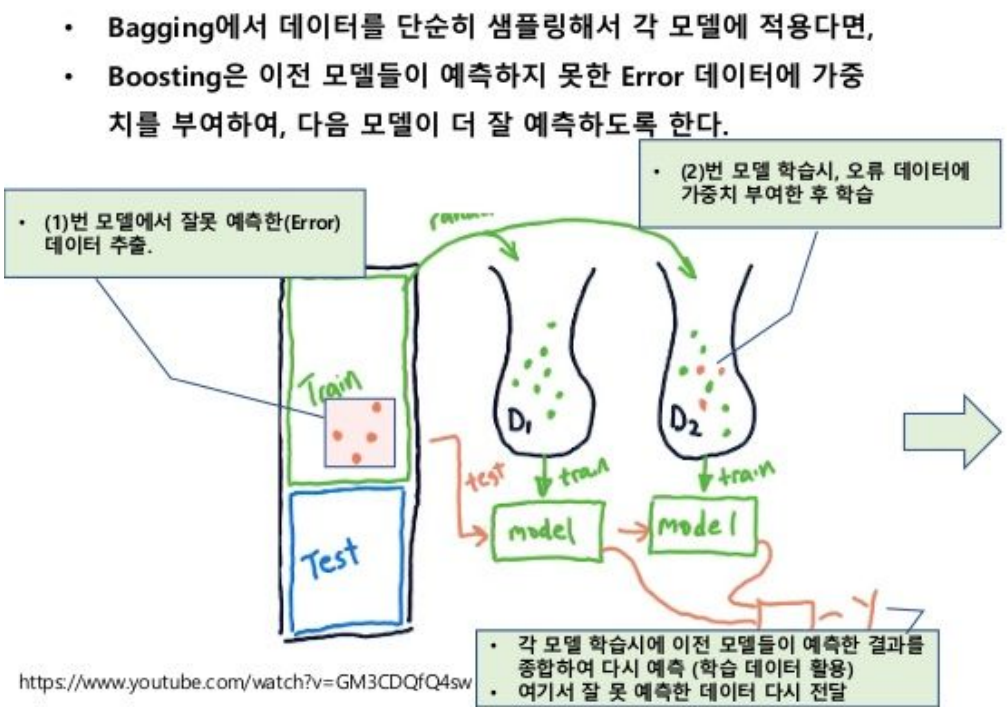


3. Boosting

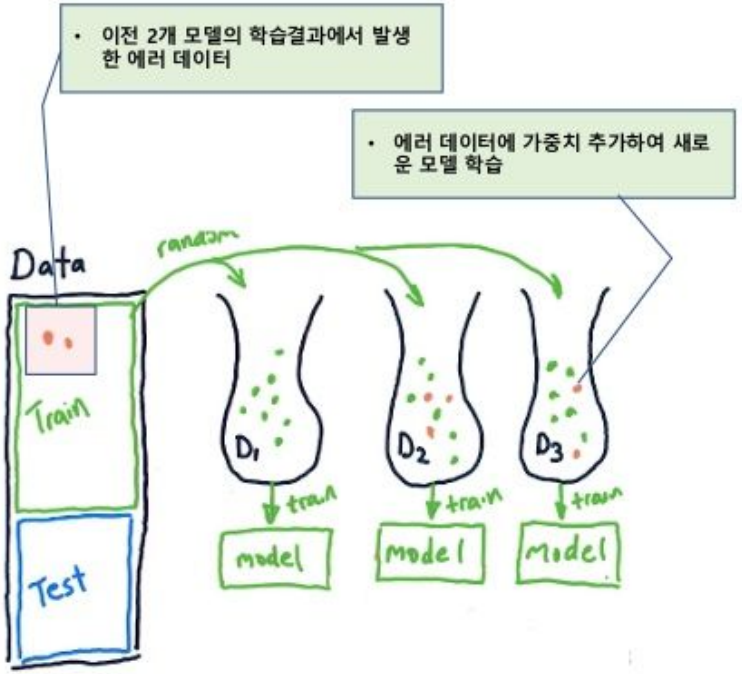
Bagging의 변형으로, 모델이 잘 예측하지 못하는 부분을 개선하기 위한 모델

Ada Boost (Adaptive, '애다'로 발음)

- Bagging에서 데이터를 단순히 샘플링해서 각 모델에 적용다면,
- Boosting은 이전 모델들이 예측하지 못한 Error 데이터에 가중치를 부여하여, 다음 모델이 더 잘 예측하도록 한다.



https://www.youtube.com/watch?v=GM3CDQfQ4sw



3. Boosting

Boosting 알고리즘

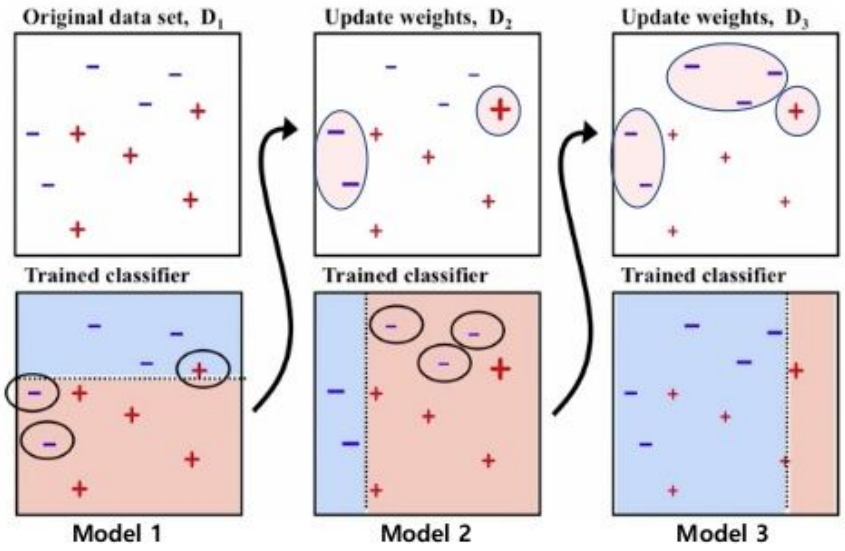
알고리즘	특징	비고
AdaBoost	• 다수결을 통한 정답 분류 및 오답에 가중치 부여	
GBM	• Loss Function의 gradient를 통해 오답에 가중치 부여	gradient_boosting.pdf
Xgboost	• GBM 대비 성능향상 • 시스템 자원 효율적 활용 (CPU, Mem) • Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)	2014년 공개 boosting-algorithm-xgboost
Light GBM	• Xgboost 대비 성능향상 및 자원소모 최소화 • Xgboost가 처리하지 못하는 대용량 데이터 학습 가능 • Approximates the split (근사치의 분할)을 통한 성능 향상	2016년 공개 light-gbm-vs-xgboost

3-1. AdaBoost (Adaptive Boosting)

AdaBoost를 이용하여 데이터를 분류하는 예시

Boosting Example

- Model1에서 잘못 예측한 데이터에 가중치를 부여
- Model2는 잘못 예측한 데이터를 분류하는데 더 집중
- Model3는 Model1, 2가 잘못 예측한 데이터를 분류하는데 집중

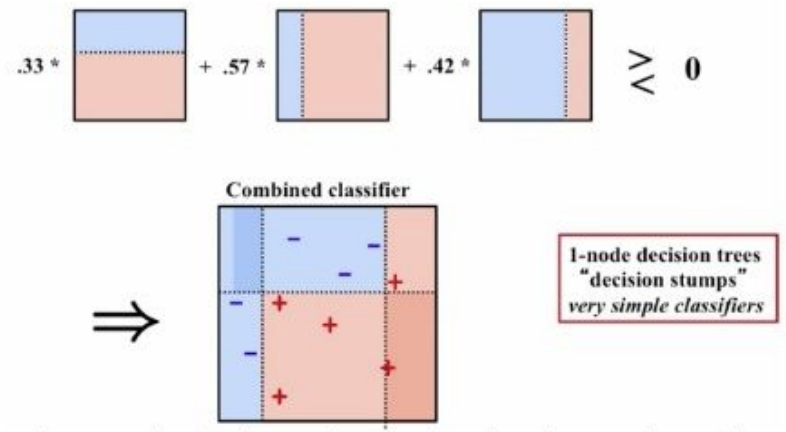


각 모델별 가중치를 고려한 예측 모델

- Cost Function : 가중치(W)를 반영하여 계산

$$J(\theta) = \sum_i w_i J_i(\theta, x^{(i)})$$

- 3개의 모델별로 계산된 가중치를 합산하여 최종 모델을 생성



https://www.youtube.com/watch?v=ix6IvwbVpw0&list=PL4zv4UkoVTPflyFDJdJtz248-8Wdz_nct&index=3

3-2. GBM(Gradient Boosting)

Gradient Boosting의 개념 및 학습 절차

개념

- AdaBoost과 기본 개념은 동일하고,
- 가중치(D)를 계산하는 방식에서
- Gradient Descent를 이용하여 최적의 파라미터를 찾아낸다.

[Easy Example]

- Model의 예측정확도 80%인 함수 `Y = M(x) + error`
- 만약 error를 줄일 수 있다면? (error가 Y와 연관성이 있을경우)
- 정확도 84%로 증가

```
error = G(x) + error2
```

- 이렇게 error를 세분화하여

```
error2 = H(x) + error3
```

- 정리한 모델의 함수

```
Y = M(x) + G(x) + H(x) + error3
```

- 각 함수별 최적 weight를 찾으면, 예측정확도는 더 높아짐.
- → Gradient Descent 알고리즘으로 최적 weight 계산

```
Y = alpha * M(x) + beta * G(x) + gamma * H(x) + error4
```

고려사항

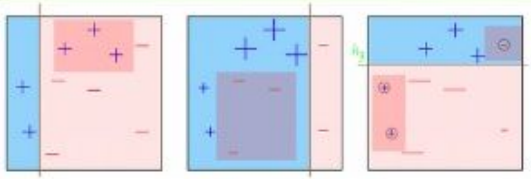
- 2가지 의문점
  - 정말 white noise가 아닌 error가 식별가능해?
  - 만약 가능하다면, 예측 정확도가 거의 100% 가능?
- Boosting은 과적합 가능성이 높아서, 적절한 시점에 멈춰야 함

https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/

3-2. GBM(Gradient Boosting)

Gradient Boosting의 개념 및 학습 절차

Gradient Descent를 이용한 weight 계산



- 1번 Wezk model에서는 3개의 오분류(에러)가 발생
- 2번은 3개 에러를 제대로 분류하기 위해 가중치 부여. (다시 3개 에러 생김)
- 3번은 다시 3개 에러를 해결하기 위한 모델 생성 (다시 3개 에러 발생)
- 최적의 weight(가중치)를 찾을 때 까지 반복

[그럼 어떻게 가중치를 부여할까?]

- 초기 데이터 가중치(D) = 1/n (n : 전체 학습 데이터 개수)
- 오류 (ε) : $\frac{\text{오류 데이터}}{\text{전체 학습 데이터}}$ 각 모델의 오류
- 모델의 가중치(α) = $\frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$
- Weak model의 함수 : *h(t)*
- 가중치 업데이트(D) :

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

데이터 가중치는 어떻게 최적화하나?

- α : 는 learning rate 역할을 수행, 지수함수의
- y : 정답 (1 or -1)
- $h(x)$: 모델의 예측값

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

- exp함수의 인자값(α)으로 학습의 방향(최적의 weight 탐색)을 확인
- 만약 학습이 잘못 되고 있다면, $-\alpha y_i h_t(x_i)$
  - $-\alpha * 1 * -1 = \alpha$ (정답은 1, 예측은 -1)
  - $-\alpha * -1 * 1 = \alpha$ (정답은 -1, 예측은 1)
- 만약 학습이 잘되고 있으면
  - $-\alpha * 1 * 1 = -\alpha$ (정답은 1, 예측은 1)
- ➔ 따라서, 예측이 틀리면 가중치(D) 증가

https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/

3-3. XGBoost (eXtreme Gradient Boosting)

XGBoost의 개념

개념

- XGBoost ?
  - GBM + 분산/병렬 처리
  - 지도학습으로 변수(x)를 학습하여 정답(y)를 예측

- Xgboost가 지원하는 모델
  - Binary classification
  - Multiclass classification
  - Regression
  - Learning to Rank

지도학습 용어

- Model : 변수(x)로 정답(y)를 예측하는 함수 $\hat{y}_i = \sum_j \theta_j x_{ij}$
- θ : 세타(가중치, 파라미터)
  - 학습을 통해 정답을 잘 예측하도록 조정

- Objective Function
  - 학습 데이터에 최적화된 파라미터 찾는 함수
  - Training Loss + Regularization

$$Obj(\Theta) = L(\theta) + \Omega(\Theta)$$

- L : Training Loss (Loss Function)

$$L(\theta) = \sum (y_i - \hat{y}_i)^2$$

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

Logistic regression

- Ω : Regularization
  - 모델의 복잡도를 조절 (과적합 방지)

http://xgboost.readthedocs.io/en/latest/model.html

https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/

https://www.slideshare.net/ShuaiZhang33/rg-xgboost20170306

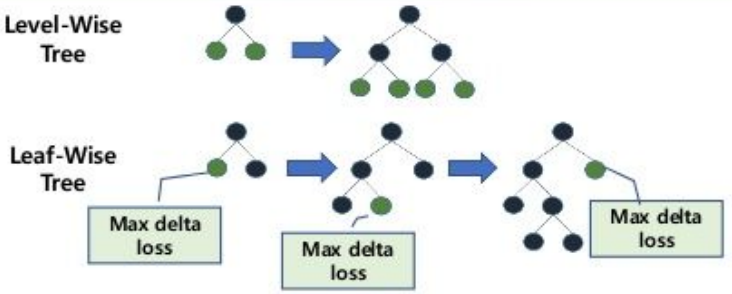
3-4. Light GBM

Light GBM 개념

개념

- Light GBM ?
  - Decision Tree 알고리즘기반의 GBM 프레임워크 (빠르고, 높은 성능)
  - Ranking, classification 등의 문제에 활용
- 무엇이 다른가?
  - Leaf-wise로 tree를 성장(수직 방향) , 다른 알고리즘 (Level-wise)
  - 최대 delta loss의 leaf를 성장
  - 동일한 leaf를 성장할때, Leaf-wise가 loss를 더 줄일 수 있다.
- 왜 Light GBM이 인기 있나?
  - 대량의 데이터를 병렬로 빠르게 학습가능 (Low Memory, GPU 활용가능)
  - 예측정확도가 더 높음(Leaf-wise tree의 장점 → 과접합에 민감)
- 얼마나 빠른가? (link) : XGBoost 대비 2~10배 (동일한 파라미터 설정시)
- 그렇게 좋은데 왜 많은 많이 안쓰지?
  - Light GBM이 설치된 툴이 많이 없음. XGBoost(2014), Light GBM(2016)
- 어디에 활용해야 하나?
  - Leaf-wise Tree는 overfitting에 민감하여, 대량의 데이터 학습에 적합
  - 적어도 10,000 건 이상

Level-wise vs Leaf-wise



- Level-wise
  - 각 노드는 root노드와 가까운 노드를 우선 순회, 수평 성장
  - XGBoost, Random Forest
- Leaf-wise
  - 가장 Loss변화가 큰 노드에서 데이터를 분할하여 성장, 수직 성장
  - 학습 데이터가 많은 경우 뛰어난 성능
  - Light GBM, XGBoost

https://blogs.technet.microsoft.com/machinelearning/2017/07/25/lessons-learned-benchmarking-fast-machine-learning-algorithms/ https://github.com/Microsoft/LightGBM/blob/master/docs/Features.rst
https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves

트리의 최대 깊이,

start 값으로 **7** 추천

경험상 변동성 높음 **(2~27)**

모델을 깊게 설정하더라도 모델이 오버핏되지 않는다면, **feature** 간의 **interation**이 많다는 것일 수 있으므로 새로운 **feature**를 뽑아내도록 시도 → '다중공선성?'

> If you increase the depth and can not get the model to overfit, that is, the model is becoming better and better on the validation set as you increase the depth.
> It can be a sign that there are a lot of important interactions to extract from the data. So it's better to stop tuning and try to generate some features.

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction

연구에 의해서, 전체데이터를 사용하는 것보다 일부 데이터를 사용하는 것이 **variance**가 작은 것으로 알려졌음

모델이 천천히 **fit**되면서 **generalize** 되는 경향이 있음 (일종의 **Regularization**)

GBDT

XGBoost	LightGBM
• max_depth • subsample • colsample_bytree, colsample_bylevel	• max_depth/num_leaves • bagging_fraction • feature_fraction

위와 유사하게 오버피팅을 막아주는 **feature**

Colsample by tree. Sub sample ratio of columns when constructing each tree.
Colsample by level. Sub sample ratio of columns for each split, in each level

---

여기까지 모델을 오버피팅 시켜주는 **param**들

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction
• min_child_weight, lambda, alpha	• min_data_in_leaf, lambda_l1, lambda_l2

min_child_weight 는 강사 경험상 중요한 **freture**였으며 **{0, 5, 15, 300}** 등 다양한 범위의 변수를 넣어가며 테스팅 해보는 것을 추천함

데이터를 정규화해주는 역할

GBDT

XGBoost	LightGBM

ETA 는 **learning weight** 역할을 함
num_round는 **learning step** → 몇개의 트리를 만들지

각 스탭별로 **eta**가 반영되어 새로운 트리가 만들어 지고,
스텝이 길어질수록 데이터에 오버피팅 될 가능성 있음

'적절한' **parameter** 설정이 여기서도 중요하다

tip)
- 학습률을 **0.1** 혹은 **0.01**로 고정
- **round** 수 변화시켜가며 모니터링
  적정 **round**를 찾았을 때, 특정 비율만큼 **learning weight**와 **round**를 곱하고 나눠줌

XGBoost	LightGBM
• eta num_round	• learning_rate num_iterations
Others:	Others:
• seed	• *_seed

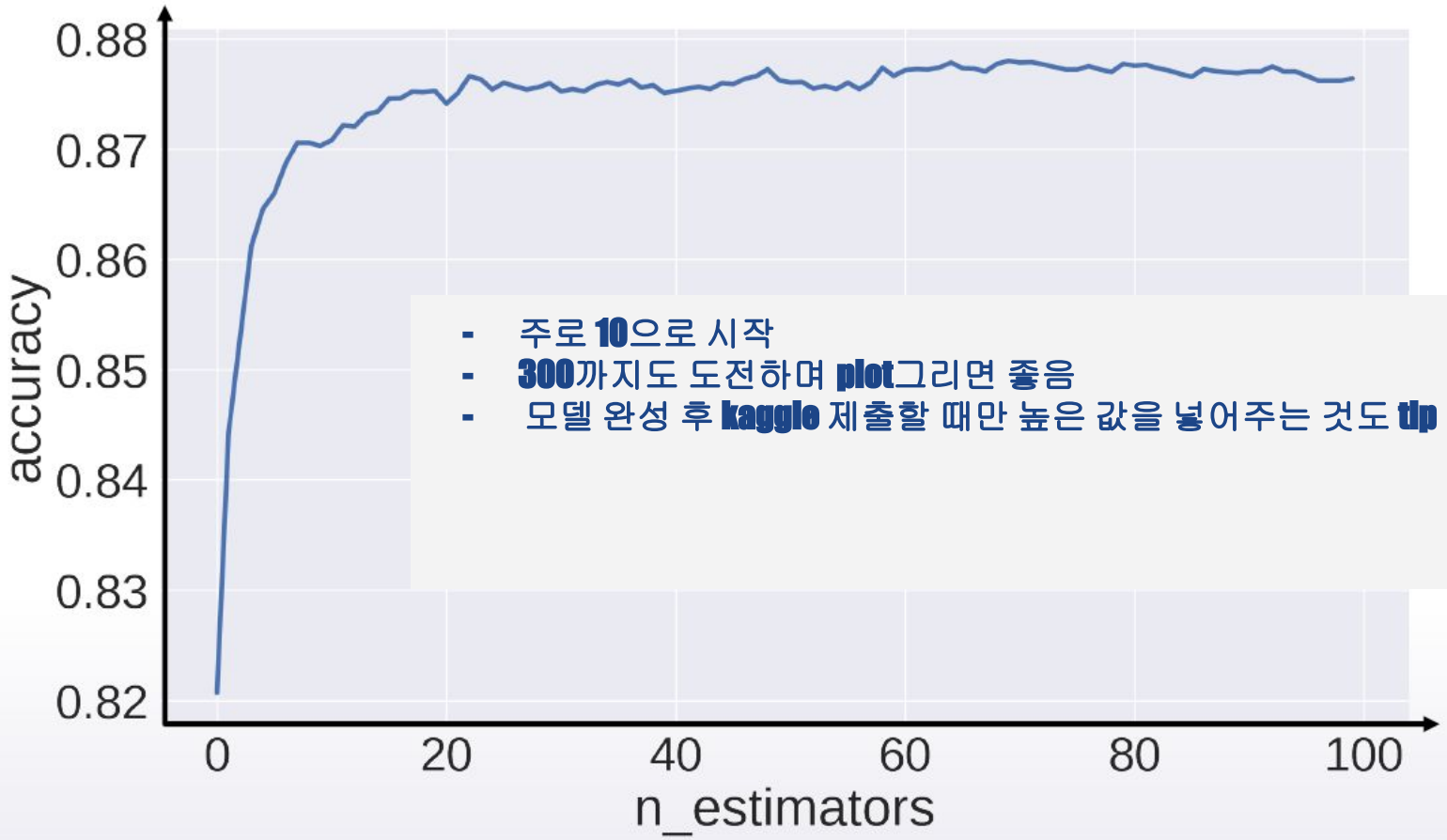
GBDT

XGBoost	LightGBM
특정 값으로 고정해주는 것이 좋음 그러나, 하나의 **seed**에 너무 편향된 모델은 아닌지 테스트 하기 위하여 다양한 **seed**를 만들어 체크해볼 필요성 있음	
Others: • seed	Others: • *_seed

sklearn.RandomForest/ExtraTrees

ExtraTree 는 **R.F.** 가 더 **randomized** 된 버전일 뿐임으로 같은 **param**을 지님
R.F. 에서 각 **Tree**는 독립적으로 이루어있으므로 오버피팅의 우려가 적음

- N_estimators (the higher the better)



- 주로 **10**으로 시작
- **300**까지도 도전하며 **plot**그리면 좋음
- 모델 완성 후 **kaggle** 제출할 때만 높은 값을 넣어주는 것도 **tip**

sklearn.RandomForest/ExtraTrees

- N_estimators (the
- max_depth
- max_features
- min_samples_leaf

정규화 인자

- 트리의 최대 깊이, 설정안해도 무관함
- 데이터 셋에서 반복되는 값이나 주요 피처 간의 인터렉션이 많이 있을 때 최대깊이를 설정하지 않는 것이 좋음

Max_features is similar to call sample parameter from XGBoost.
The more features I use to decipher a split, the faster the training.
But on the other hand, you don't want to use too few features.

왜 많은 **feature**를 쓸 때, 빨라지는지 이해가 안됨;;;

Others:

- *criterion*
- **random_state**
- **n_jobs**

분할이 얼마나 잘됐는지 평가하기 위해서 지니계수와 정보엔트로피를 사용할 수 있음 → 주로 지니를 쓰긴했는데 **perfomance check** 필요함

random_state → **seed** 고정

n_jobs → 기기의 **core** 수 설정, **sklearn**에서는 **core**를 **1**개만 자동으로 인식한다네요

Neural Nets

- Number of neurons per layer
- Number of layers
- Optimizers
  - SGD + momentum
  - Adam/Adadelta/Adagrad/...
    - In practice lead to more overfitting
- Batch size
- Learning rate
- Regularization
  - L2/L1 for weights
  - Dropout/Dropconnect
  - Static dropconnect

학습이 겁나 천천히 되면서, 언더피팅되는 경향이 있음

Adaptive Method는 빠르게 **training set**에 **fit**하는 대신 오버피팅하는 경험적 결론을 냄

분류나 회귀에서 더 좋다고 말함

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

Neural Nets

- Number of neurons per layer
- Number of layers
- Optimizers
  - SGD + momentum
  - Adam/Adadelta/Adagrad/...
    - In practice lead to more overfitting
- Batch size
- Learning rate
- Regularization
  - L2/L1 for weights
  - Dropout/Dropconnect
  - Static dropconnect

한번에 많은 데이터를 넘기는 것이므로,오버피팅될 우려있음

32나 64를 기준으로 값을 올리고 내리기 시도 추천함

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

Neural Nets

- Number of neurons per layer
- Number of layers
- Optimizers
  - SGD + momentum
  - Adam/Adadelta/Adagrad/...
    - In practice lead to more overfitting
- Batch size
- Learning rate
- Regularization
  - L2/L1 for weights
  - Dropout/Dropconnect
  - Static dropconnect

적정한 **rate**가 필요하다

Batch Size와 **trade-off** 가능

어쨋든 배치사이즈가 커질수록 오버피팅가능성있는거 다시 생각하라

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

Neural Nets

- Number of neurons per layer
- Number of layers
- Optimizers
  - SGD + momentum
  - Adam/Adadelta/Adagrad/...
    - In practice lead to more overfitting
- Batch size
- Learning rate
- Regularization
  - L2/L1 for weights
  - Dropout/Dropconnect
  - Static dropconnect

L1 / L2 정규화보다 최근에는 **dropout**을 주로 쓰는 추세,

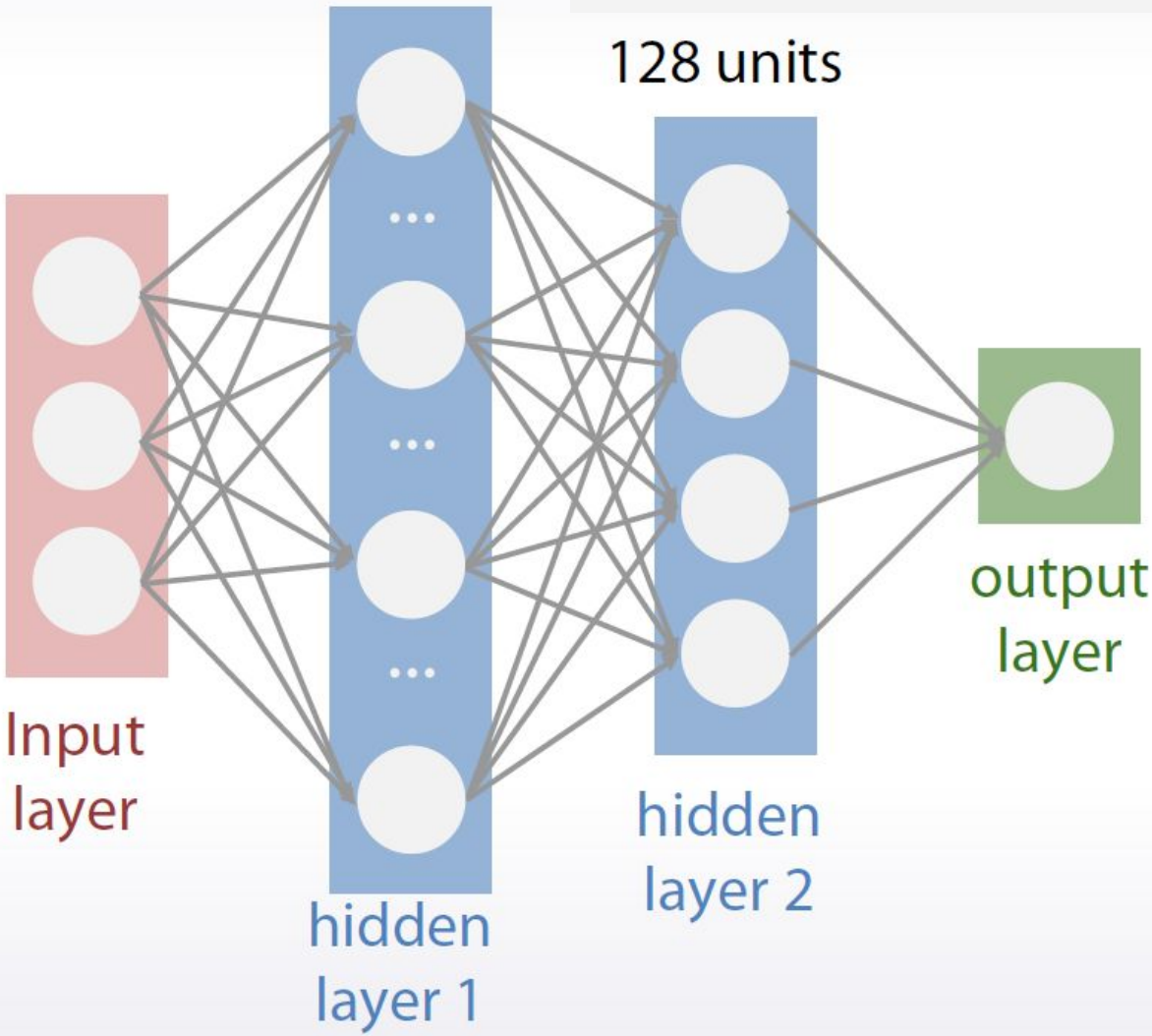
dropout 확률과 **layer**는 직접정의해도 되나, 끝부분에 두는 것이 일반적임

이걸로 진짜 중요한 **feature**를 찾는데 도움이 된다.
그러나 첫번째 **layer**에는 두지말도록 ← 너무 큰 정보손실이 발생하니까

Static dropconnect

Static dropconnect

첫 **hidden layer**를 굉장히 큰 **units**으로 구성
정규화하기 위해 랜덤하게 **99%**를 **drop**
learning 내내 **static**하게 **drop**하는 **connection** 유지



https://stats.stackexchange.com/questions/201569/difference-between-dropout-and-dropconnect

Linear models

libLinear & libSVM → SVM으로 분류나 회귀 실행하는 라이브러리

sklearn에서 멀티코어로 하려면 따로 설정해줘야함

- **Scikit-learn**
  - SVC/SVR
    - Sklearn wraps libLinear and libSVM
    - Compile yourself for multicore support
  - LogisticRegression/LinearRegression + regularizers
  - SGDClassifier/SGDRegressor
- **Vowpal Wabbit**
  - FTRL

온라인에서 러닝시키는 방법 → 대용량 데이터 처리가능

Flow The Regularized Leader (FTRL)

Linear models

L2 와 L1 등의 정규화가 주요 하이퍼파라미터

- Regularization parameter (C, alpha, lambda, …)
  - Start with very small value and increase it.
  - SVC starts to work slower as C increases

SVM 같은 경우, 10 같은 작은 seed에서 시작해서 점점 크게 테스팅함

10 → -6 → X10 → -6 → x10 ….

- Regularization type
  - *L1/L2/L1+L2* -- try each
  - *L1* can be used for feature selection

L1은 feature selection할 때 사용
L1 / L2 / L1+L2는 각각 모두 시도

L1 이 L2 보다 outlier에 robust 하다고 알려져있음
(https://light-tree.tistory.com/125)

Tips

1)하이퍼파라미터 튜닝에 너무 많은 시간을 쏟지 말기
더 이상 아이디어가 없거나 여분의 계산 리소스가 있는 경우에만 시도하기

- **Don't spend too much time tuning hyperparameters**
  - Only if you don't have any more ideas or you have spare computational resources
- **Be patient**
  - It can take thousands of rounds for GBDT or neural nets to fit
- **Average everything**
  - Over random seed
  - Or over small deviations from optimal parameters
    - e.g. average *max_depth=4,5,6* for an optimal *5*

2)참고 견디자
GBDT 또는 신경망을 수천번 돌려야 할 수도 있음
3)모든 것을 평균
파라미터도 평균!

참고) 오버피팅과 언더피팅

https://nittaku.tistory.com/289

참고) 파라미터 vs 하이퍼 파라미터

모델내부에서 확인가능한 값 → 데이터로 산출 가능한 값

모델외부에서 얻어진 것

신경망 학습을 통해서 튜닝 또는 최적화 해야하는 주변수가 아니라,학습 진도율이나 일반화 변수처럼, 사람들이 선험적 지식으로 설정을 하거나 또는 외부 모델 메커니즘을 통해 자동으로 설정이 되는 변수를 말한다.

http://blog.naver.com/PostView.nhn?blogId=tjdudwo93&logNo=221067763334&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView

참고) XGB 부스트

https://zzsza.github.io/data/2018/07/03/xgboost/ : xgboost 리뷰