

Mini
Project
2021.02.18-



Overview

④ 문제의 정의

- 기계학습 (데이터를 제공해준 데에 따른 바탕으로 학습을 진행)
- 다양한 성능지표는 기본으로 그 태도 고려의 차이현상을 예측해) 회귀
- 선형회귀 (Linear regression), 다항회귀 (polynomial regression)
- 회귀 SVM,及び Random Forest
- 인공신경망 (Artificial Neural Network)
- 지난 성능지표의 사용은 고려하지만 RNN, CNN, 또는 Transformer 사용 가능
- XGBoost
- Naive Bayes
- Ensemble
- 전처리 단계는 기존 데이터에서 고려된 지난 행동패턴을 통해 다음에 예상되는 구조를 예측한다.
예측할 수 있는 단계다.

↳ 이전의 시계열 데이터와 행동에서 변화에는
데이터의 고급 특성, 개별적 특성이 대용어
이용할 수 있다고 생각된다.

② 대안개요

→ 기존의 sales 데이터를 이용해서 2015년 1월의 구매자는 놓자마자 해당하는 각 shopping item들의 item-unt-month를 예측하는 것이 목표이다.

③ ~~Kernel~~의 개요

1. 모델 input으로 들어가게 될 base dataframe을 만든다.

(점/상점/상품을 기준으로 dataframe을 만든다)

2. 기존 Feature들로 새로운 Feature를 만든다.

- shop, item 기준 평균값, 등간값, 편차

- shop, item-group 기준 평균값

- 1달, 2달, 12달전, 상품이 팔린 개수, 주문수

- 이동평균값, MACD값, ATR값

- 할인율

3. 만든 Feature를 모두 base dataframe으로 합친다.

4. Machine Learning 모델을 이용해서 훈련하고 테스트한다.

1. Data EDA & Augmentation

Feature Engineering

① 1st Feature Engineering

할당된 추가한 특성을

① mean_item_price (M: 가격의 평균값)

② mean_item_cnt (제품의 구매수를 평균화)

③ item_price_unit

④ hist_min_item_price / hist_max_item_price

⑤ price_increase / price_decrease

⑥ item_cnt_min / item_cnt_max / item_cnt_mean / item_cnt_std

⑦ item_cnt_shifted1 / item_cnt_shifted2 / item_cnt_shifted3
(1단 칸 이동을 한 경우) (2단 칸을 한 경우) (3단 칸을 한 경우)

⑧ item_trend

⑨ shop_mean / item_mean / shop_item_mean

0~37일의 누적판권
24시간내 판권

3~30: Train

31~33: Valid

34: Test

35: Predict Target

이를 바탕으로 구하는 특성들

특성들은 예측이 원하는

Salaries.csv 파일을 제작해야하는데
그리고

cf. pandas dataframes 는立法 경제
axis의 범주에 맞춰서 처리해야 한다.

④ Second Feature Engineering

이전에 소개한 맥락과 같이 다른 조건들을 추가하였다.

<Tip>

(*) Grouping

- Datasets such as transactions rarely fit the definition of tidy data

→ we use grouping

- the key point of groupby operations is to decide the aggregation functions of the features

④ Aggregating Categorical Columns

- Highest Frequency : the MAX operation for categorical columns.

ex) `data.groupby('ID').agg(lambda d: d['value'].count().index[0])`

→ pandas의 agg() 모듈은 같은 카테고리 (여기서는 lambda)를

이동하면서만 그룹을 만들고 'sum', 'mean' 등과 같은

기본적으로 사용하는 함수를 적용할 수 있다.

obj	A	B	C
1	2	3	
4	5	6	
7	8	9	

이제는 dataframes로 표기

	A	B	C
sum	12	15	18
min	1	2	3

df.agg(['sum', 'min'])을 통해

이제는 그룹을 이용한 대신 그룹을

각각의 그룹에 대해 min을 계산

각각의 그룹에서 aggregation을
진행해낸다.

User	City	Visit
1	R	1
2	M	2
1	M	1
3	I	4
2	I	3
1	I	3
1	R	3

- Make a pivot table



Drop all binary flag columns

Drop all target features
to make dataframes

aggregate into matrix

User	R	I	M
1	1	3	1
2	0	4	2
3	0	1	0

- Other hot Encoding

Coursera Kaggle 강의(How to win a data science competition) week 3,4 Advanced Feature Engineering 요약

04 Nov 2018 in Data (/category/data/) on Machine Learning (/tag/datamachinelearning/)



1. Mean encodings

- Categorical feature로 groupby하여 target의 mean 값을 feature로 추가
 - mean뿐 아니라 median, std, min, max 등 해당 데이터의 성격을 잘 나타내는 통계 연산

ex)

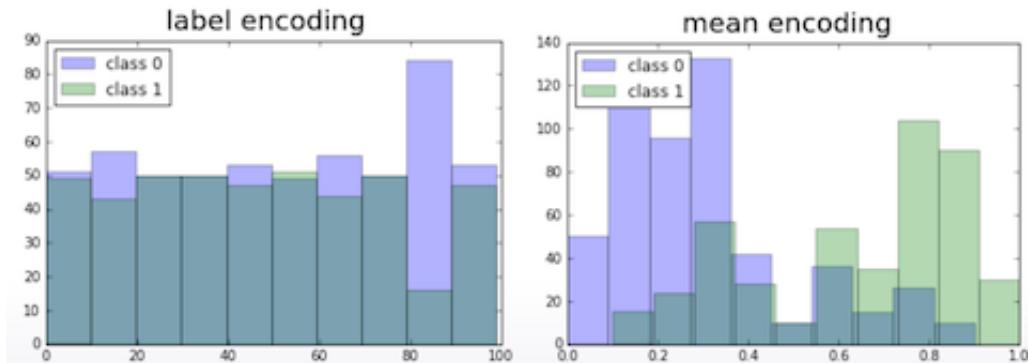
통계값으로 적합하기

	feature	feature_label	feature_mean	target
0	Moscow	1	0.4	0
1	Moscow	1	0.4	1
2	Moscow	1	0.4	1
3	Moscow	1	0.4	0
4	Moscow	1	0.4	0
5	Tver	2	0.8	1
6	Tver	2	0.8	1
7	Tver	2	0.8	1
8	Tver	2	0.8	0
9	Klin	0	0.0	0
10	Klin	0	0.0	0
11	Tver	2	0.8	1

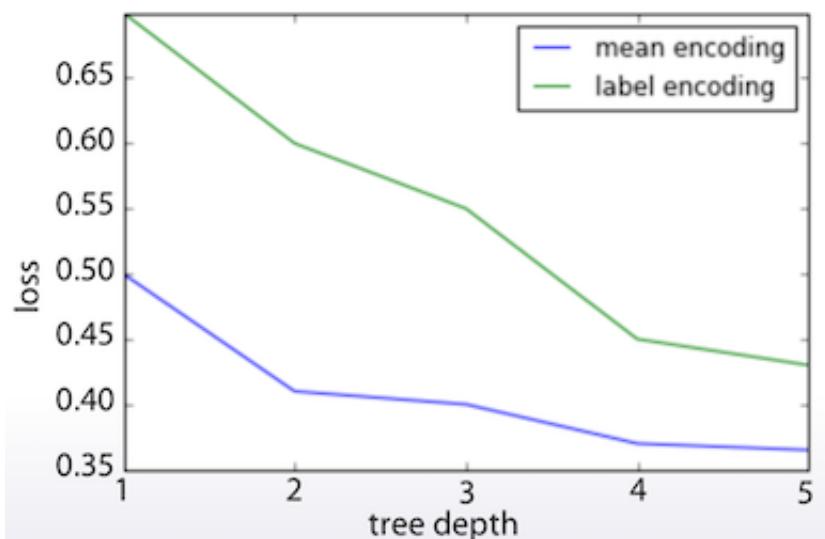
위 그림의 경우

- Moscow : 1(2개), 0(3개) -> 0.4 ($= \frac{2}{5}$)
- Tver : 1(4개), 0(1개) -> 0.8 ($= \frac{4}{5}$)

- Klin : 0 all \rightarrow 0.0



Label encoding의 경우 단순히 순서에 의해 부여된 숫자일뿐 target과의 관련성이 전혀 없지만, Mean encoding 의 경우 0 ~ 1 사이 값으로 target값과 연관성이 있다. 특히 0인 것과 아닌 것은 완전히 구분이 된다.



더 짧은 tree로도 더 좋은 성능을 발휘한다.

적용 방법들

- Likelihood : $\text{Goods} / (\text{Goods} + \text{Bads}) = \text{mean}(\text{target})$
- Weight of Evidence = $\ln(\text{Goods} / \text{Bads}) * 100$
- Count = Goods = $\text{sum}(\text{target})$
- Diff = Goods - Bads

cf. Goods : 1의 개수, Bads : 0의 개수

tree 개수가 증가하면서 정확도가 계속 증가한다면 아직 overfitting이 아니라는 뜻이다. 이럴

때 `mean encoding` 을 적용하여 더 빨리 정확도를 올릴 수 있다. 하지만 overfitting은 항상 주의해야 한다. 왜냐면 train, validation의 비율이 다를 경우 overfitting된 상태라면 결과가 좋지 않다.

2. Regularization (정규화)

2.1 CV Loop regularization

- 매우 직관적이고 강력한(robust)한 방법

```
y_tr = df_tr['target'].values
skf = StratifiedKFold(y_tr, 5, shuffle=True, random_state=123)

for tr_ind, val_ind in skf:
    X_tr, X_val = df_tr.iloc[tr_ind], df_tr.iloc[val_ind]
    for col in cols:
        means = X_val[col].map(X_tr.groupby(col).target.mean())
        X_val[col+'_mean_target'] = means
    train_new.iloc[val_ind] = X_val

prior = df_tr['target'].mean() # global mean
train_new.fillna(prior, inplace=True)
```

- Fold를 나누어서 validation set의 feature를 train set의 `mean encoding` 으로 설정 (보통 4~5 정도면 충분)
 - NA값은 global mean으로 설정

단 LOO (Leave one out)같은 극단적인 경우는 주의

2.2 Smoothing

다른 regularization을 얼만큼 적용시킬지를 `Alpha` 값을 이용하여 설정 \\\n $\frac{\text{mean}(\text{target}) * \text{nrows} + \text{globalmean} * \alpha}{\text{nrows} + \alpha}$

$$\frac{\text{mean}(\text{target}) * \text{nrows} + \text{globalmean} * \alpha}{\text{nrows} + \alpha}$$

2.3 Noise

통상적으로 노이즈를 추가하면 encoding의 품질이 저하되어서 얼마나 넣어야 할지 고민을 많이 해야 한다. LOO와 함께 사용하면 효과적이다. (자세한 설명이 없음)

2.4 Expanding mean

```
cumsum = df_tr.groupby(col)['target'].cumsum() - df_tr['target']
cumcnt = df_tr.groupby(col).cumcount()
train_new[col+'_mean_target'] = cumsum/cumcnt
```

- CatBoost에는 내장된 방식
- Leakage를 줄일 수 있으나 품질이 불규칙적이다.

3. Generalizations and extensions

3.1 Regression과 Multiclass

- Regression : percentile, std, distribution bin등의 통계함수를 추가
- Multiclass : 각각의 class별로 encoding

3.2 Many-to-many relation

- Cross product
- Statistics from vectors

LONG REPRESENTATION					
User_id	APPS	Target	User_id	APP_id	Target
10	APP1; APP2; APP3	0	10	APP1	0
11	APP4; APP1	1	10	APP2	0
12	APP2	1	10	APP3	0
100	APP3; APP9	0	11	APP4	1
			11	APP1	1

APPS를 각각의 row로 나눔

3.3 Time series

time-series 데이터의 경우 앞의 방법들을 사용하는게 의미 없을 수 있다.

Day	User	Spend	Amount	Prev_user	Prev_spend_avg
1	101	FOOD	2.0	0.0	0.0
1	101	GAS	4.0	0.0	0.0
1	102	FOOD	3.0	0.0	0.0
2	101	GAS	4.0	6.0	4.0
2	101	TV	8.0	6.0	0.0
2	102	FOOD	2.0	3.0	2.5

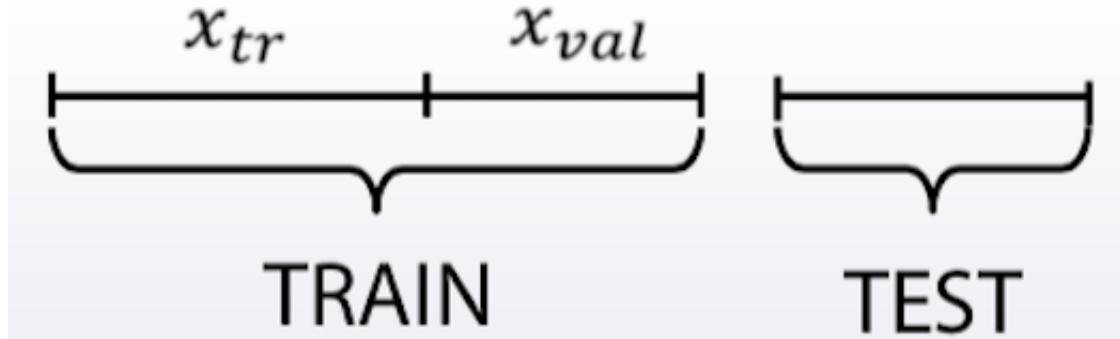
위 예제의 경우 Day-User-전날 Amount 합 과 Day-Spend-전날 Amount 평균 을 feature로 추가하였다.

3.4 Interactions and numerical features

- numeric feature를 bin하여 categorical feature로 만드는 방법이다.
 - EDA 과정을 통해서 숫자상의 어떤 점에서 target값의 분기가 일어나는지를 관찰하는 것이 유용하다.
- feature 2개 이상이 상호작용적으로 동작하는 경우에는 그것들을 합하여 **mean encoding** 하는 방법도 있다.

Correct validation reminder

- Local experiments: - Estimate encodings on X_tr - Map them to X_tr and X_val - Regularize on X_tr - Validate model on X_tr/ X_val split
- Submission: - Estimate encodings on whole Train data - Map them to Train and Test - Regularize on Train - Fit on Train



- Main advantages: - Compact transformation of categorical variables - Powerful basis for feature engineering
- Disadvantages: - Need careful validation, there a lot of ways to overfit - Significant improvements only on specific datasets

4. Statistics and distance based features

1개의 feature를 **groupby** 하여 계산한 다양한 통계값을 활용

	User_id	Page_id	Ad_price	Ad_position	Max_price	min_price	Min_price_position
0	4	6	95.874252	Bottom_right	474.63772	73.711548	Bottom_left
1	4	6	215.751007	Bottom_right	474.63772	73.711548	Bottom_left
2	4	6	474.637726	Bottom_left	474.63772	73.711548	Bottom_left
3	4	6	73.711548	Bottom_left	474.63772	73.711548	Bottom_left
4	4	6	79.288841	Bottom_right	474.63772	73.711548	Bottom_left
5	4	6	271.391785	Bottom_right	474.63772	73.711548	Bottom_left
6	4	6	296.529053	Bottom_right	474.63772	73.711548	Bottom_left
7	4	6	96.030029	Bottom_right	474.63772	73.711548	Bottom_left
8	4	6	130.175064	Bottom_left	474.63772	73.711548	Bottom_left
9	4	7	35.465202	Bottom_left	121.54219	35.465202	Bottom_left
10	4	7	121.542191	Bottom_right	121.54219	35.465202	Bottom_left

위 그림의 경우 **[User_id, Page_id, Ad_price, Ad_position]** 은 원래 있던 feature들이고 **[Max_price, min_price, Min_price_position]** 은 추가로 생성한 feature들이다.

- Max_price : Page_id로 groupby한 MAX(Ad_price)
- min_price : Page_id로 groupby한 MIN(Ad_price)
- Min_price_position : Page_id내에서 Ad_price == min_price 에 해당하는 Ad_posision

```

gb = df.groupby(['user_id', 'page_id'], as_index=False)
    .agg({'ad_price': {'max_price': np.max, 'min_price': np.min}})
gb.columns = ['user_id', 'page_id', 'min_price', 'max_price']

df = pd.merge(df, gb, how='left', on=['user_id', 'page_id'])

```

위에 예제로 보인 feature뿐만 아니라 다양한 것들을 생각해 볼 수 있다.

- user가 얼마나 많은 page를 방문했는지
- price의 표준편차
- 가장 많이 방문한 page
- 등등...

그런데, groupby 할 수 없는 feature는 어떻게 해야할까 ? 그럴 경우 kNN 과 같은 클러스터링 기법들로 대체가 가능하다.

- 500m, 1000m 이내에 있는 집의 수
- 500m, 1000m 이내에 있는 집의 평균 평당가
- 500m, 1000m 이내에 있는 학교/슈퍼마켓/주차장 수
- 가장 가까운 지하철역

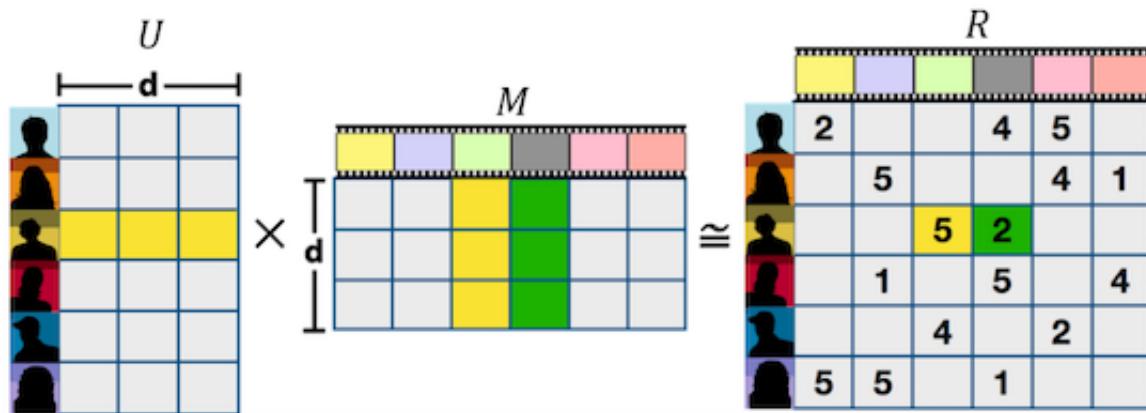
사례: Springleaf에 사용된 kNN feature들

- 모든 변수에 대해서 mean encoding
- 모든 point에서 Bray-Cutis metric을 이용하여 2000-NN을 찾음
$$\sum |u_i - v_i| / \sum |u_i + v_i|$$
- 2000개의 이웃으로 다양한 feature들을 계산
 - 5, 10, 15, 500, 2000개의 NN의 target mean값
 - 10개의 가까운 이웃들과의 distance mean값
 - 10개의 가까운 이웃중 target이 1/0인 것과의 distance mean
 - 등등...

기억력이 약한
다른 사람이 할 수 있음.

5. Matrix Factorizations for Feature Extraction

feature 추출에 행렬 분해 기법을 활용하는 방법들이다.



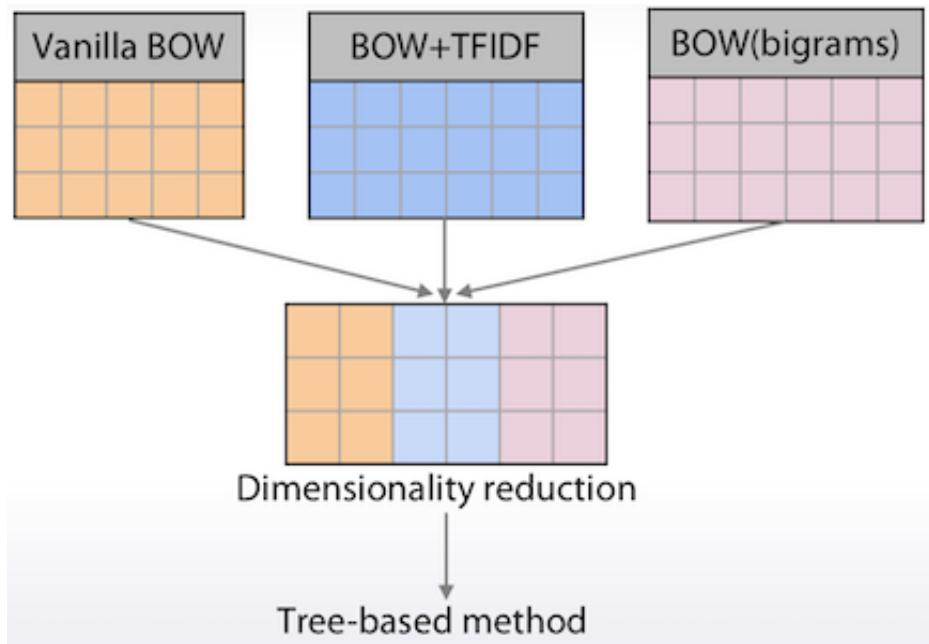
예를 들어 사용자별로 연령, 지역, 관심사, 성별 등의 feature들이 있고, 이러한 정보에 따른 등급 정보가 있는 경우 이 둘을 matrix 곱으로 표현할 수 있다.

	cat	is	dog	animal	nature
Cat is animal	1	0	0	1	0
Dog is animal	0	1	1	1	0
Cat is dog	1	1	1	0	0

Cat is animal			
Dog is animal			
Cat is dog			

cat			
is			
dog			
animal			
nature			

text 분류기의 경우에도 보통 sparse matrix로 표현되는 것을 각각의 matrix로 표현하는 것이 가능하다.



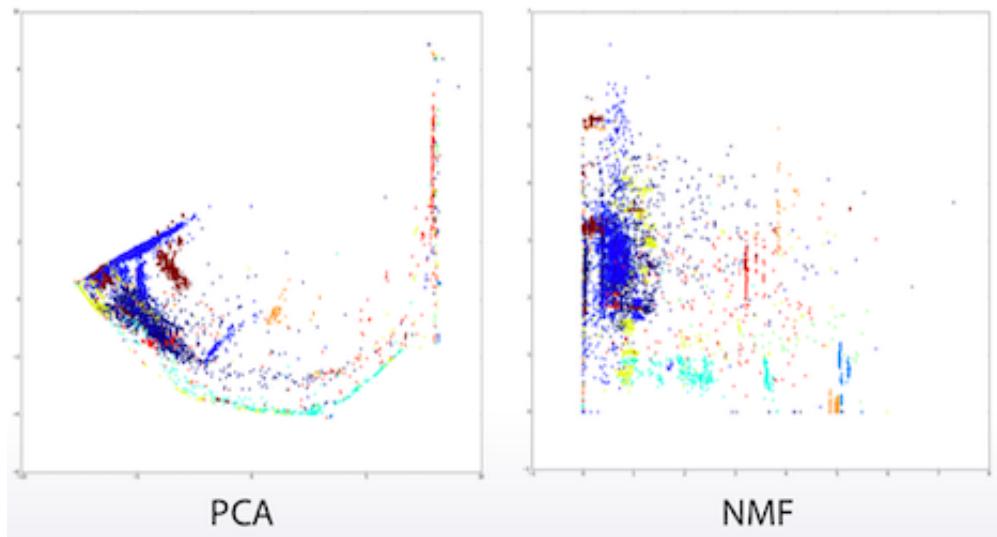
여러 방법들을 섞어서 사용하기도 한다.

Matrix Factorization 진행시

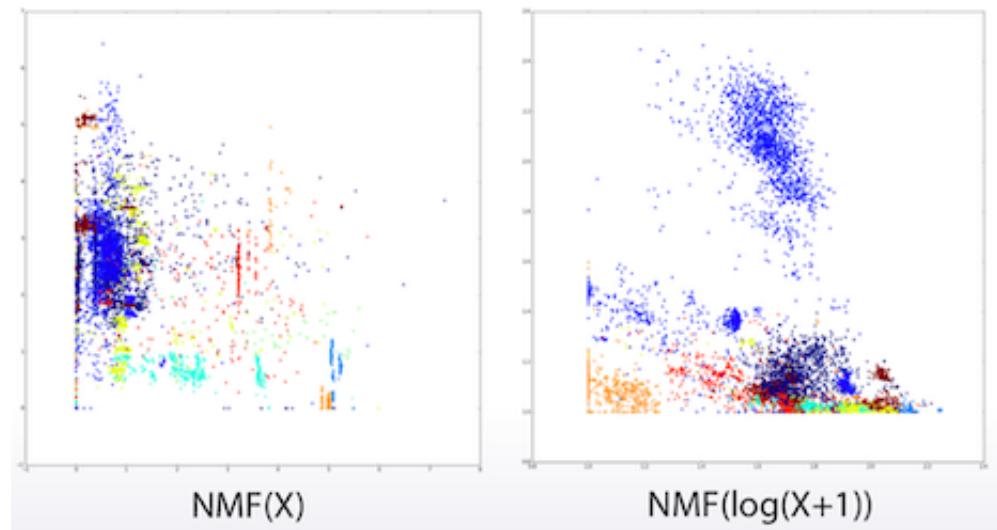
- 몇개의 column만을 적용시키기도 함.
- 다양한 것의 추가가 가능하다. (Ensemble에 효과적)
- 손실 변환(lossy transformation)이다.
 - 특정 작업에 의존적이므로 경험적으로 해야한다.
 - factor는 보통 5 ~ 100 사이로 설정한다.

`sklearn`에 Matrix Factorization의 구현체가 있음

- `SVD` , `PCA` : MF용 standard tool
- `TruncatedSVD` : sparse matrices용
- `NMF` (Non-negative MF) : count-like한 데이터에 좋음. 모든 데이터가 non-negative인게 보장됨



위 그림을 보면 **NMF** 를 적용하니 linear model처럼 분리되는 것을 볼 수 있다.



log(X+1) 를 적용하는 등의 다양한 trick을 시도해 볼 수 있다.

MF 적용시 train, test를 각각따로 적용하면 안된다. 둘을 함께 적용시켜야 올바른 결과가 나온다.

- Wrong way

```
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.fit_transform(X_test)
```

- Right way

```
X_all = np.concatenate([X_train, X_test])
pca.fit(X_all)
X_train_pca = pca.transform(X_train)
```

```
X_test_pca = pca.transform(X_test)
```

- MF는 차원 축소와 feature 추출에 좋다.
- categorical feature를 좀 더 현실적인 값으로 변경해준다.
- linear model화 되도록 적절한 trick을 적용하면 유용하다.

6. Feature interactions

...	category_ad	category_site	...	is_clicked
...	auto_part	game_news	...	0
...	music_tickets	music_news	..	1
...	mobile_phones	auto_blog	...	0
...	ad_site	...	is_clicked	
...	auto_part game_news	...	0	
...	music_tickets music_news	..	1	
...	mobile_phones auto_blog	...	0	

위 경우를 보면 category_ad와 category_site가 있는데, 정작 중요한 것은 그 둘의 조합이다. 이 경우 그 둘을 concat하여 ad_site로 하는게 더 효율적일 수 있다.

f1	f2
A	X
B	Y
B	Z
A	Z

부록성이 합쳐져 적용되는걸
따로 하는것은 허용하지 않으나
조합이 중요하다면
부록이 반드시 해석을 해주고
해석된다

위 경우에서 FI 방법에 대해서 2가지로 생각해보자.

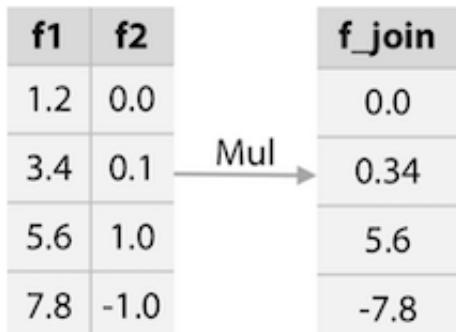
f1	f2	f_join	A X	B Y	B Z	A Z
A	X	A X	1			
B	Y	B Y		1		
B	Z	B Z			1	
A	Z	A Z				1

Join
OneHot
(f_join)

첫번째 방법은 앞에서 본 경우와 같이 f1과 f2를 조합하여 one-hot을 적용시킨 방법이다.



두번째 방법으로는 f1과 f2를 각각 one-hot한 다음 그 둘을 조합한다.



두 데이터는
one-hot 후 조합

Mul

f1	f2
1.2	0.0
3.4	0.1
5.6	1.0
7.8	-1.0

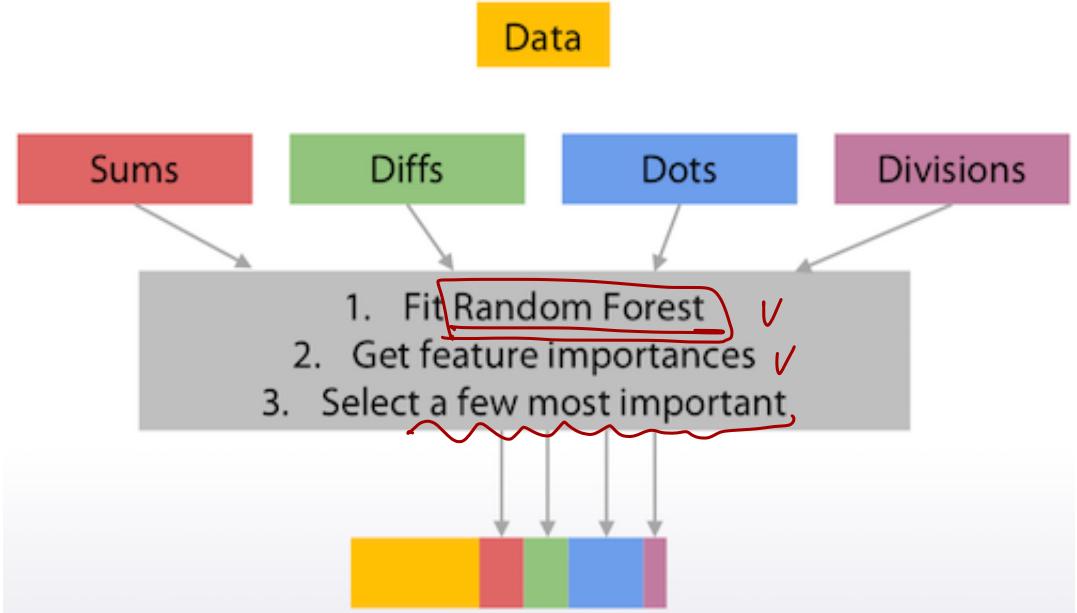
f_join
0.0
0.34
5.6
-7.8

F1의 또 다른 예로 f1과 f2가 둘 다 numeric일 경우 그 둘을 곱하는 방법도 있다.

이렇게 곱, 합, 차, 나누기 등을 적용하는 방법들을 생각해 보자.

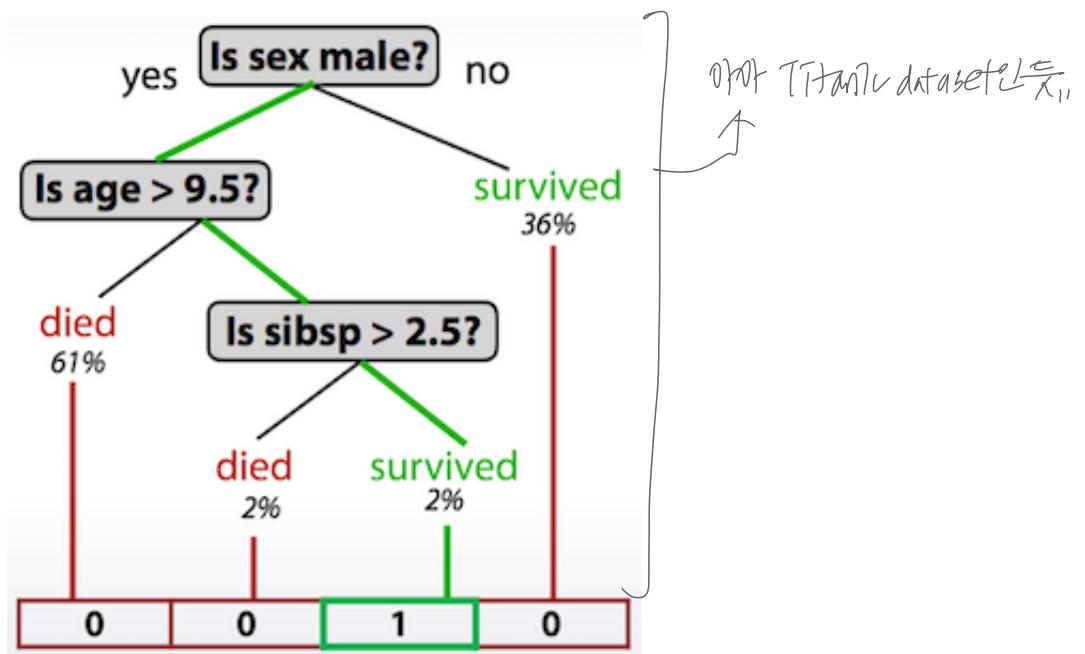
모든 feature에 그렇게하기에는 조합을 할 수 있는 경우의 수가 너무나도 많다.

어떻게 유의미한 것들만을 골라내서 차원을 줄일 수 있을까?



- Random Forest로 일단 돌려본 다음
- Feature Importance를 보고
- 유의미한 것들만을 추출

~~Decision Tree~~ 를 이용하여 feature를 추출하는 방법도 있다.



```
# In sklearn
tree_model.apply()
```

```
# In XGBoost
booster.predict(pred_leaf=True)
```

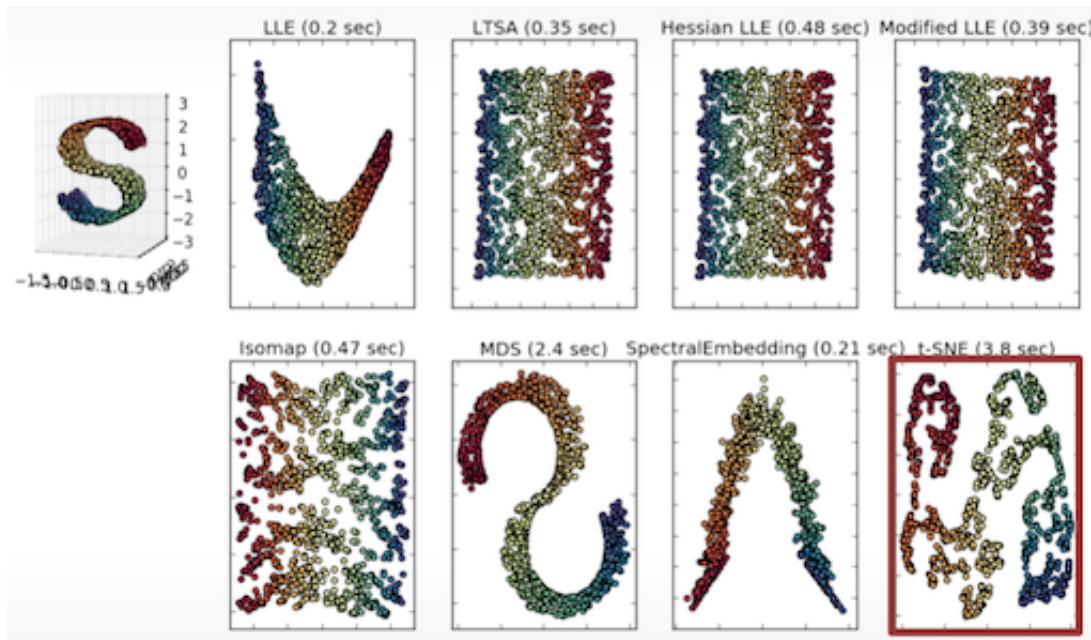
sklearn DecisionTree()는 원본 Feature를 기반으로 나누는 방식이고 Xgboost는 booster에 적용을 해서 Feature Importance를 통해 그걸 중요도가 높은 변수를 선택하는 방식.

7. tSNE

앞서 **NMF** 를 이용하여 linear model에 가깝게 변화시키는 것을 본 적이 있다.

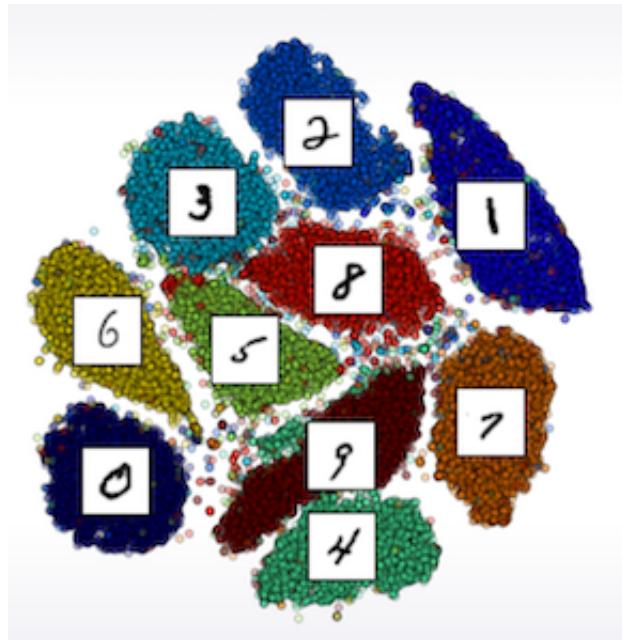
7.1 Manifold Learning

비선형 차원 축소 방법 (non-linear method of dimensionality reduction)



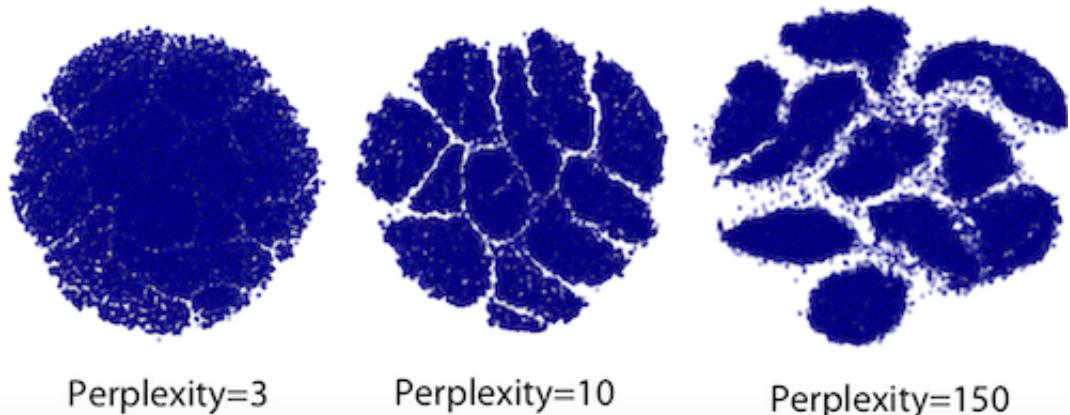
데이터를 차원변환시켜서 해석하기 쉬운형태로 분리하는게 가능하다.

7.2 MNIST의 tSNE 반영 결과



2차원 공간에 투영된 700차원 공간

Perplexity hyperparameter를 어떻게 설정하느냐에 따라 결과가 달라진다.



여기에 대해 정리된 내용은 다음과 같다.

- **Perplexity** hyperparameter값에 크게 의존적이다.
 - 5 ~ 100 사이 값이 결과가 대체로 좋았다.
- 확률적 특징때문에 같은 데이터, perplexity 값이라도 다른 결과가 나온다.
 - train, test는 같이 투영되어야 한다.
- feature가 많으면 오래 걸린다.
 - 그래서 보통 투영하기전에 차원축소를 한다.

- tSNE의 구현체는 `sklearn`에 있다.
 - 하지만, 파이썬의 개별패키지로 하는게 더 빠르기 때문에 추천하지는 않는다.

이 글이 도움이 되셨다면 공감 및 광고 클릭을 부탁드립니다 :)



Share this post



([http://twitter.com/share?text=Coursera Kaggle 강의\(How to win a data science competition\) week 3,4 Advanced Feature Engineering 요약 &url=https://DevStarSJ.github.io/data/2018/11/04/kaggle.coursera.competition.03.02/](http://twitter.com/share?text=Coursera Kaggle 강의(How to win a data science competition) week 3,4 Advanced Feature Engineering 요약 &url=https://DevStarSJ.github.io/data/2018/11/04/kaggle.coursera.competition.03.02/))



(<https://www.facebook.com/sharer/sharer.php?url=https://DevStarSJ.github.io/data/2018/11/04/kaggle.coursera.competition.03.02/>)

About



Server Developer & Machine Learning Engineer

Related Posts

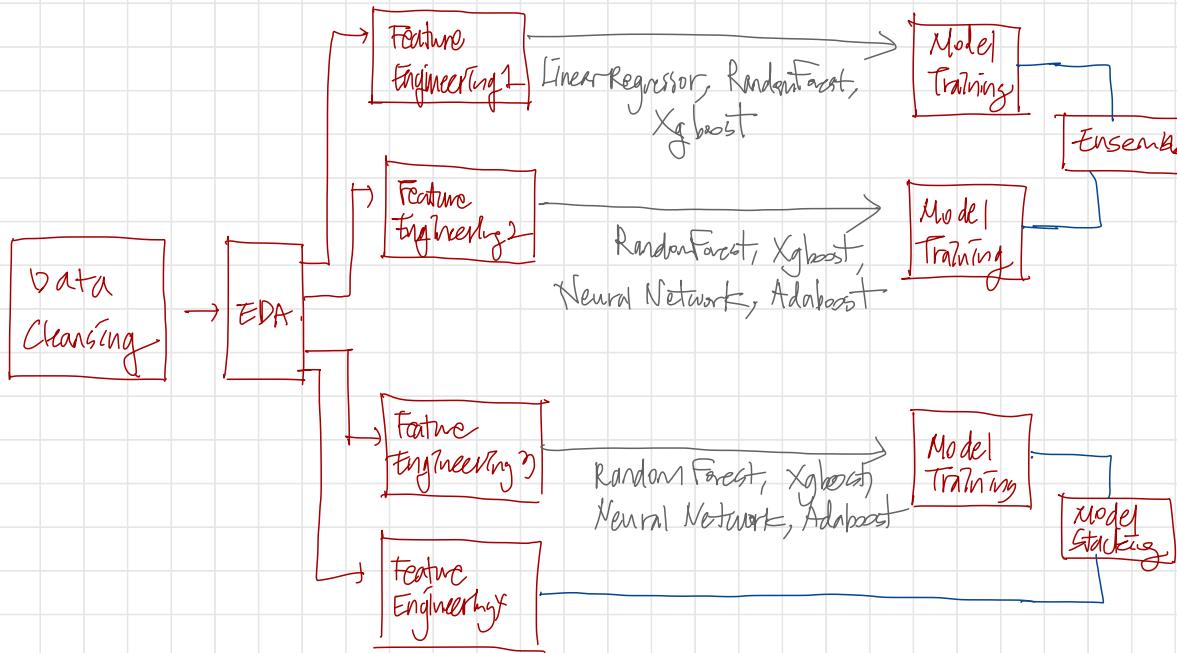
Coursera Kaggle 강의(How to win a data science competition) week 4-4

Ensemble 요약 (</data/2018/10/30/kaggle.coursera.competition.04.02/>) 30 Oct 2018

Coursera Kaggle 강의(How to win a data science competition) week 4-1

Hyperparameter Tuning 요약 (</data/2018/10/29/kaggle.coursera.competition.04.01/>) 29

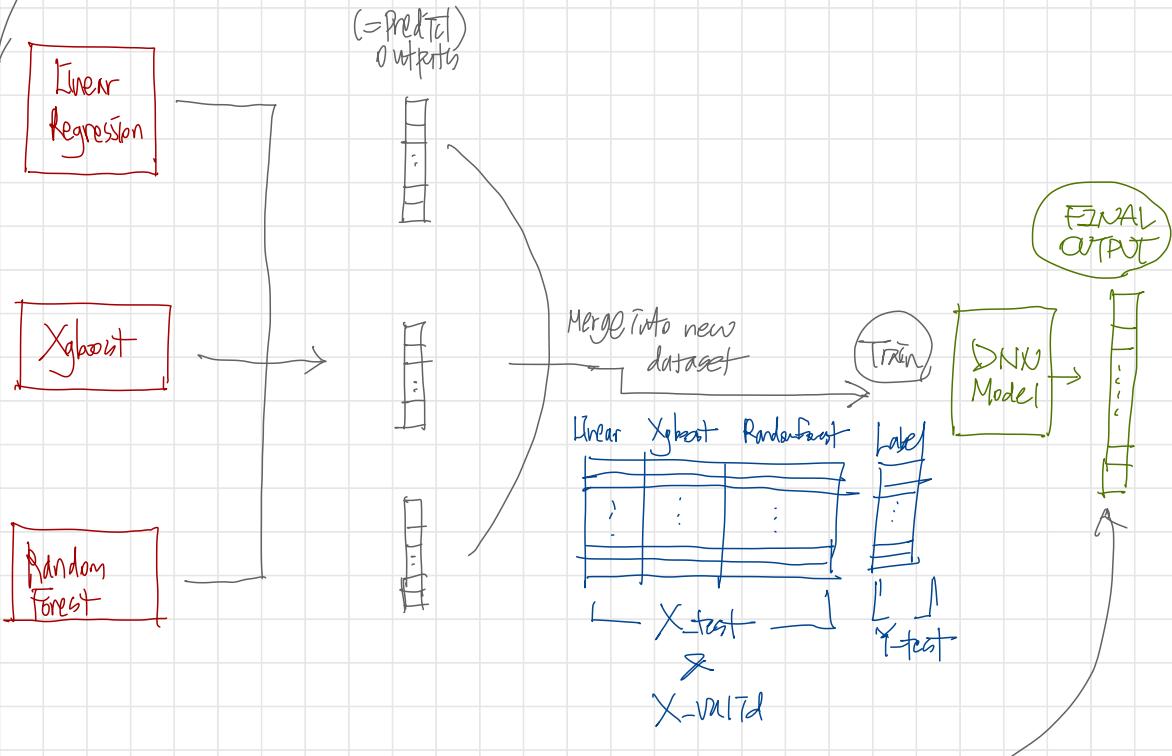
2. Modeling



④ 이렇게 조작된 것을 만들고(모델)의 결과는 데이터셋을 만들어
DNN 모델에 넣고(학습시킨다) 조작된 결과를 어떻게 활용할까? 같은가?

⑤ 예전에는 모델 구조를 활용해 feature importance를 구해서
feature engineering 내용을 추적하고, 제일 중요한 특성을 활용하는
모델로 학습하는 방식을 선택한다.

Input = $X_{\text{test}} // X_{\text{val/7d}}$



기존 모델의
결과를 합친 후 X-Val을
빼고 나면 TRAIN이 됨,
X-test를 뺏거나면 해당 DNN
가 예상하는 결과를
output으로 얻게 되어라.

④ Linear Regression의 단점과 문제점이 있어서 그냥 Linear Regression 대신 Neural Network를 추가하고 Catboost 또한 추가된다.

⑤ 유익한 Feature를 찾는 Feature Engineering. 1. 어떤 특성을 선택할지 고민

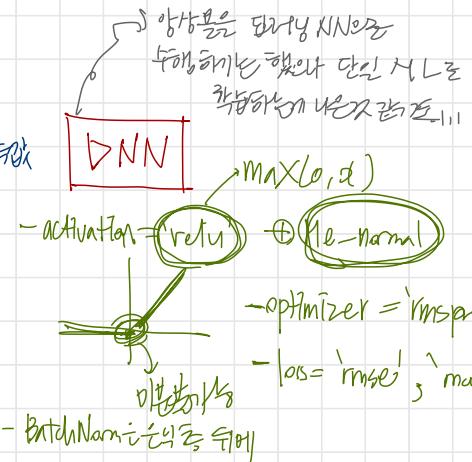
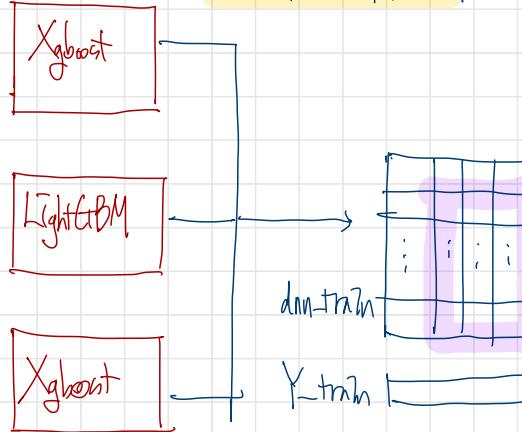
Feature Engineering을 대체하기로 함

→ lag data와 결합함을 이용해 plot_importance를 통해서 확인이 가능하다.

∴ Feature Engineering은 실제로 어떤 특성이 분류를 성공해서 사용하는 부분이다.

$X_{\text{train}} \Rightarrow$ dataset 2018년 1월~12월의 feature
 $X_{\text{valid}} \Rightarrow$ dataset 2018년 1월
 $X_{\text{test}} \Rightarrow$ dataset 2018년 2월~3월

$Y_{\text{train}} \Rightarrow$ dataset 2018년 1월~12월의 item-cat
 $Y_{\text{valid}} \Rightarrow$ " 2018년 1월
 $Y_{\text{test}} \Rightarrow$ " 2018년 2월~3월

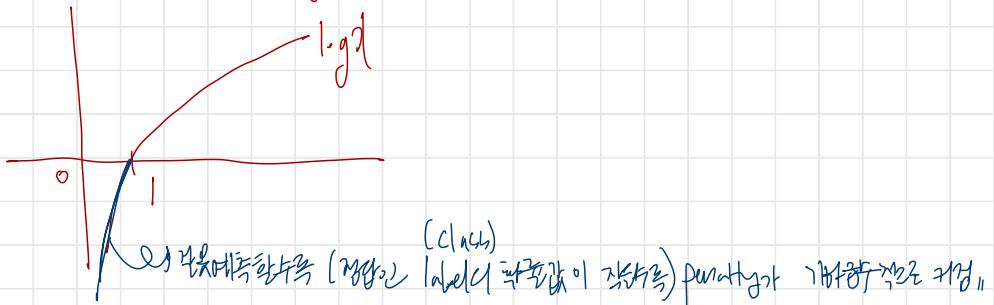


* 계속 log loss를 이용해서 문제를 계산해야 한다고 생각하였다.

알고보니 log loss가 아니라 rmse(=root mean squared error)이용하는게 더 나을 것 같았다.

log loss는 결국 퍼셉트론으로 구현할 때 cross-entropy-loss라고하는데,
이는 원래적으로 회기 (Logistic / Linear Regression)에서나 이전 classification에서
사용하고 있다.

흔히 예제로 softmax 확률과 헝카를 적용해서 출력값이 각 class별 확률값(정답)이
되도록 해야 한다. log loss의 수식을 살펴보면



DNN 모델이 학습을 하면서 class-weight를 직접 설정해주는 방식을 사용할 것이다. 그래서 앞서 만든 (dnn-train, Y-train) (dnn-Valid, Y-valid) (dnn-test)는 그대로 이용하지만

이 두 데이터로 구간 X와 B004로 학습을 시키고

"Plot_Importance" 은 농도별로 model 01, model 02, model 03을

더 높은 모델로 초기 weight를 미리 DNN에 넣어준다.

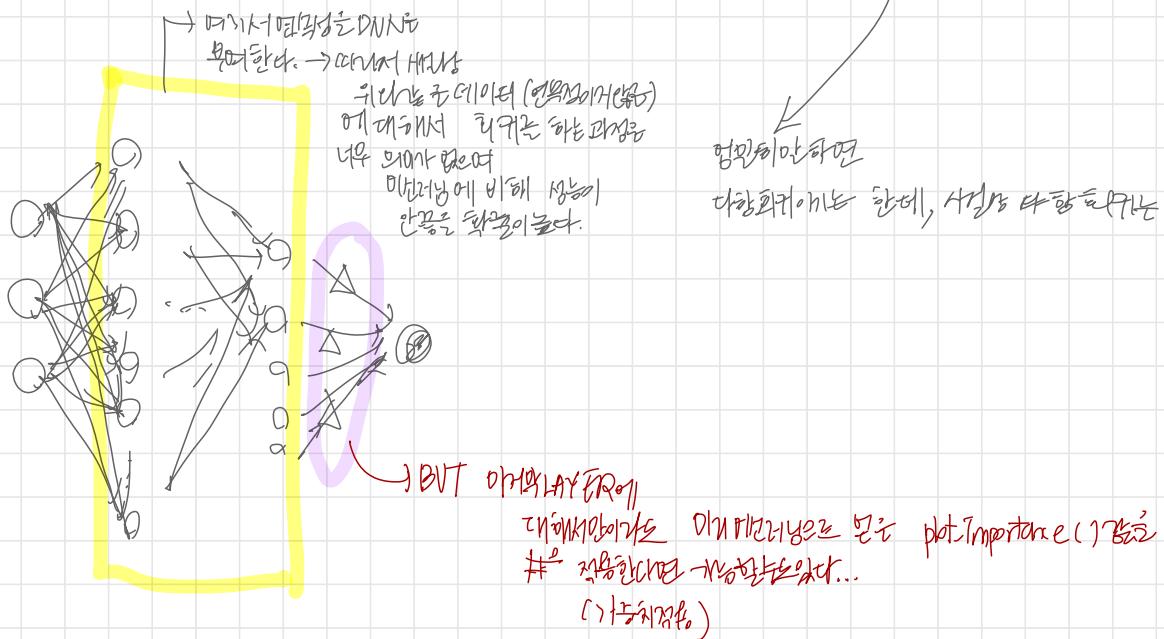
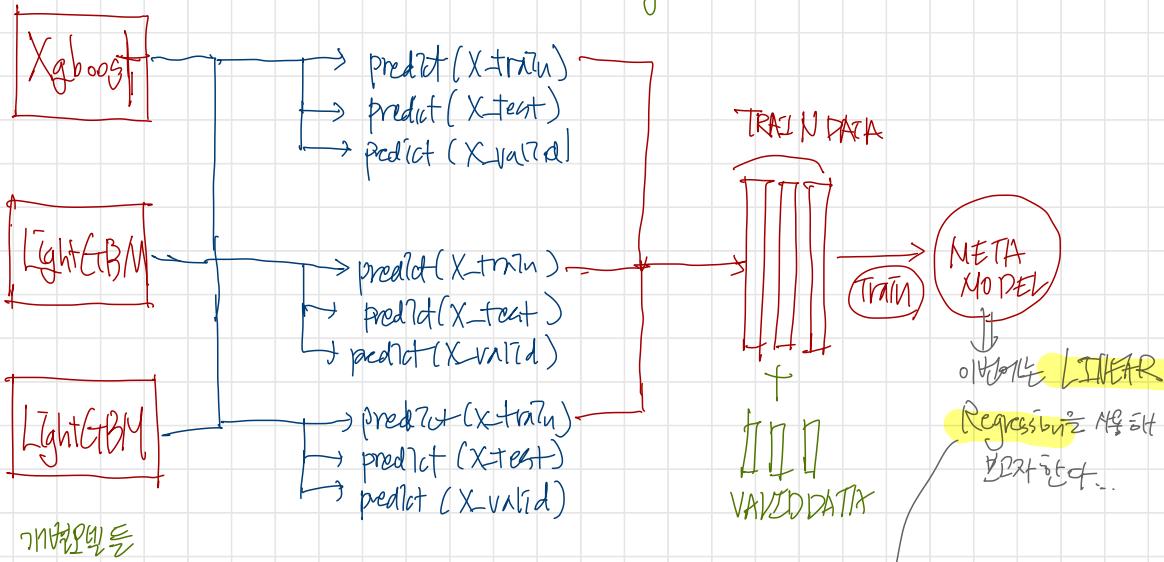
① 경제학자로서는 ↗
이익을 사용자 정의 등을 이용해야 하기 때문에

Class MyDense (tf.keras.layers.Layer):

이용해 구현하는데
API를 활용하여 전행되었을 때,

이경계내로 제대로 학습에 간접적일 것이다. 일반적으로 학습률을 이용해 학습률을 진화하는 경로를 만드는다.

(*) DNN을 추가하기로 했지만, 이전기원은 stacking Ensemble을 선택했습니다.
그동안 머신러닝 모델은 사용하지 않았지만 뉘웠다는 판단이 들었다.
→ cross valid set 기반의 stacking을 이용하게 한다.



MODEL 01(XGBoost)

```
[ ] model01 = XGBRegressor(
    max_depth=10,
    n_estimators=1000,
    min_child_weight=0.5,
    colsample_bytree=0.8,
    subsample=0.8,
    eta=0.05,
    seed=42)

[ ] model01.fit(X_train[feature_name], Y_train, eval_metric = ['rse'],
    eval_set=[(X_val[feature_name], Y_val)], early_stopping_rounds=50,
    verbose = 20)
```

MODEL 02 (LightGBM)

```
[ ] params = {
    'objective': 'mse',
    'metric': 'rse',
    'num_leaves': 2 ** 8 - 1,
    'learning_rate': 0.0025,
    'feature_fraction': 0.75,
    'bagging_fraction': 0.75,
    'bagging_freq': 7,
    'seed': 1,
    'verbose': 1
}

feature_name_indexes = [
    'country_part',
    'item_category_common',
    'item_category_code',
    'city_code',
]

lgb_train = lgb.Dataset(X_train[feature_name], Y_train)
lgb_eval = lgb.Dataset(X_valid[feature_name], Y_valid, reference=lgb_train)

[ ] evals_result = {}
model02 = lgb.train(
    params,
    lgb_train,
    num_boost_round=2000,
    valid_sets=[lgb_train, lgb_eval],
    feature_name = feature_name,
    categorical_feature = feature_name_indexes,
    verbose_eval=200,
    evals_result = evals_result,
    early_stopping_rounds = 150)
```

Final Model 1 (LightGBM) → Score: 0.85991

```
[ ] from sklearn.model_selection import KFold

def get_stacking_base_dataset(model, X_train, y_train, X_test, n_folds):
    kf = KFold(n_folds, shuffle = True, random_state = 42)
    train_fold_ids = np.zeros(len(X_train))
    test_fold_ids = np.zeros(len(X_test), dtype = int)
    print(model.__class__.__name__, "Model Started")
    for folder_counter, (train_index, valid_index) in enumerate(kf.split(X_train)):
        print("Fit Model Fold", folder_counter + 1)
        X_T = X_train[train_index]
        y_T = y_train[train_index]
        X_V = X_train[valid_index]

        if model == model01:
            evals = [(X_val[feature_name], Y_val, eval_metric = ['rse']),
                    eval_set=[(X_val[feature_name], Y_val)], evals_result = evals_result,
                    early_stopping_rounds=50, verbose = 20)
        else:
            evals = [()]
            evals[0] = (X_val[feature_name], Y_val, eval_metric = ['rse'],
                        eval_set=[(X_val[feature_name], Y_val)], evals_result = evals_result,
                        early_stopping_rounds=150)

        model.fit(X_T, y_T, evals)
        train_fold_ids[valid_index] = folder_counter
        test_fold_ids[valid_index] = folder_counter
    print("All folds are trained")
    print("Generate Submission File")
    submission = pd.DataFrame()
    submission['id'] = X_test['id']
    submission['label'] = model.predict(X_test)
    submission.to_csv('stacking_base.csv', index=False)
```

Final Model 2 (DNN) → Score: 0.85846

```
[ ] import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.layers import Dense
from tensorflow.keras import Model

[ ] batch_size = 100
initializer = tf.keras.initializers.HeNormal()

DNN_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, input_shape = [30], activation = 'relu', kernel_initializer=initializer),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(32, activation = 'relu', kernel_initializer=initializer),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(1)
])

DNN_model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param # 
dense_5 (Dense)              (None, 32)               128    
batch_normalization_5 (Batch Normalization) (None, 32)   128    
dropout_2 (Dropout)           (None, 32)               0      
dense_6 (Dense)              (None, 3)                99    
batch_normalization_6 (Batch Normalization) (None, 3)   12    
dense_7 (Dense)              (None, 1)                4      
_________________________________________________________________
total_params: 371
trainable_params: 30
non_trainable_params: 70
```



MODEL 03 (LightGBM)

```
[ ] params = {
    'objective': 'mse',
    'metric': 'rse',
    'num_leaves': 2 ** 8 - 1,
    'learning_rate': 0.0003,
    'feature_fraction': 0.75,
    'bagging_fraction': 0.75,
    'bagging_freq': 7,
    'seed': 1,
    'verbose': 1
}

feature_name_indexes = [
    'country_part',
    'item_category_common',
    'item_category_code',
    'city_code',
]

lgb_train = lgb.Dataset(X_train[feature_name], Y_train)
lgb_eval = lgb.Dataset(X_valid[feature_name], Y_valid, reference=lgb_train)

evals_result = {}
model03 = lgb.train(
    params,
    lgb_train,
    num_boost_round=2000,
    valid_sets=[lgb_train, lgb_eval],
    feature_name = feature_name,
    categorical_feature = feature_name_indexes,
    verbose_eval=200,
    evals_result = evals_result,
    early_stopping_rounds = 150)
```

Optimizer

- 외로 Adam을 이용해 보다 rmsProp을 이용하고 애초에 주어지는 $1e-3$ 이라는 학습률을 유지했을 때 validation_loss와 validation_accuracy가 더 좋았던 것 같다.
- learning_rate: $1e-3$ 에서 $1e-5$ 이상적인 값이다.
- Adam의 경우에는 $\text{max}(0, x)$ 라는 함수를 이용해서 비선형성을 부과하기 때문에 음수가 나오지만 않는다면 크게 문제가 되는 부분이 없다고 한다.

Weight Initializer

- 나중에 ReLU activation function을 사용하기 때문에 HeNormalization을 사용하였다.

Batch Normalization

- 논문에 의하면 비선형 함수 이전에 넣는것이 좋다고 한다.(미만한 차이지만)
- 그리고 데이터셋을 배치 사이즈가 길수록 효과를 보인다고 한다.
- 비선형 함수 이전에 배치 위해서는 반드시 비선형 함수를 따로 추가해야 한다.

Coursera Kaggle 강의(How to win a data science competition) week 4-1 Hyperparameter Tuning 요약

29 Oct 2018 in Data (/category/data/) on Machine Learning (/tag/datamachinelearning/)

Hyperparameter Tuning

- 어떤 hyperparameter가 모델에 가장 큰 영향을 미치는지 알아야 한다.
- 모든 hyperparameter를 튜닝할 시간이 없다.
- 수동 : 변경 후 결과 측정
- 자동화된 라이브러리
 - Hyperopt
 - Scikit-optimize
 - Spearmint
 - GpyOpt
 - RobO
 - SMAC3

일반적으로 수동으로 하는것이 더 빠르다.

Hyperparameter 중 해당 값이 커질수록 underfitting 되는 것도 있고 (red), 해당 값이 커질수록 overfitting 되는 것도 있다. (green)

1. Tree-based models

XGBoost, LightGBM, CatBoost, sklearn의 RandomForest 및 Extra Trees models, Regularized Greedy Forest 등...

방법이 일반적으로 제일 광범위

1.1 Gradient Boosting

트리를 만드는 매개변수

- max_depth (XGBoost, LightGBM) (green) : tree의 최대 깊이. 퀄리티가 더 빠르게 됨. 하지만 학습 시간이 굉장히 오래 걸리게됨. 통상적으로 7로 시작하는게 좋음.
- num_leaves (LightGBM) (green): 트리의 리프수를 제어
- subsample (XGBoost), bagging_fraction (LightGBM) (green): 0 ~ 1
- colsample_bytree, colsample_bylevel (XGBoost), feature_fraction (LightGBM) (green) : tree 분할시 일부만을 사용
- min_child_weight (XGBoost), min_data_in_leaf (LightGBM) / lambda, alpha (XGBoost), lambda_l1, lambda_l2 (LightGBM) (red): 정규화 매개변수,
 min_child_weight가 가장 중요. 최소값은 0 (가장 보수적 모델). 작업에 따라 최적 값은 0, 5, 15, 300 되므로 넓은 값을 사용하는 것에 주저하지 마라.

학습관련 매개변수

- eta (XGBoost), learning_rate (LightGBM) (green): 학습 가중치. 너무 낮으면 수렴하지 않을 수 있음.
- num_round (XGBoost), num_iterations (LightGBM) (green): 학습 회수

random seed를 바꾸는 방법도 있지만, 일반적으로 큰 차이가 없다.

1.2 Random Forest, Extra Trees

Extra Trees는 Random Forest의 무작위 버전이며 매개변수가 동일

- n_estimators : 트리 수, 매우 작은 수(10)로 설정하여 시간이 얼마나 걸리는지 본 다음 300 정

도의 큰 값으로 설정하는 식으로 접근. 어느 시점 이후에는 그 변화가 미비할 수 있으므로 적당한 값을 찾는게 좋기는 하지만, 클 수록 좋음.

- max_depth (green) : 트리의 깊이. 7로 시작하는게 좋으나 일반적으로 Random Forest의 최적값은 Gradient Boosting보다 높으므로 주저없이 높게 설정해라. None으로 설정하면 깊이가 무제한으로 생성된다.
- max_features (green) : subsample(XGBoost) 과 비슷함.
- min_samples_leaf (red) : min_child_weight (XGBoost)와 비슷

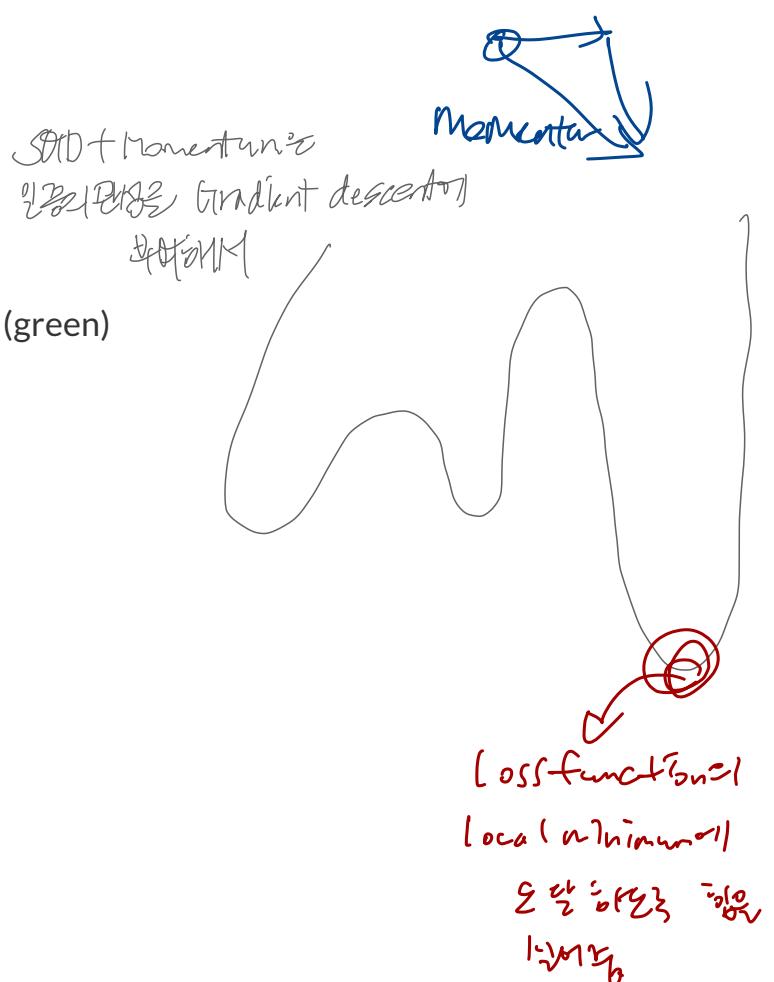
Random Forest Classifier의 경우 트리 분할을 향상시키는 기준으로 Gini 와 Entropy를 사용하는데, 둘 다 시도해서 성능이 좋은 것을 선택해야 한다. 자주 Gini가 더 좋지만, Entropy가 더 좋은 경우도 있다.

- random_state : 랜덤 시드
- n_jobs : 보유한 코어 수로 설정

2. Neural Nets

Keras, PyTorch, Tensorflow, MxNet, ...

- layer 의 뉴론 수 (green)
- layer 수 (green)
- Optimizers
 - SGD + momentum (red)
 - Adam / Adamdelta / Adagrad / ... (green)
- Batch size (green)
- Learning rate
- Regularization
 - L1 / L2 for weights (red)
 - Dropout / Dropconnect (red)
 - Static dropconnect (red)



2-1. Xgboost

- 일련의 예측기로부터 예측을 수행하여 가장 좋은 모드로 업데이트하는 Ensemble 기법
- Xgboost는 망울증에서도 최적값을 기반으로 작동한다.

사용 가능한 Feature에 대해서 모든 것들을 선택하거나 어떤 그룹 중에서만 선택해서 계산을 한다.

Xgboost = Extreme Gradient Boosting

④ Gradient Boosting

- AdaBoost와 마찬가지로 망울증에 이전까지의 예측은 보정할수 있도록 예측기를 부여점으로 추가한다.

하지만 AdaBoost처럼 반복마다 망울의 가중치를 조정하는 대신에

이전 예측에 대한 점수(오차)에 새로운 예측기를 학습시킨다.

→ AdaBoost의 경우에는 각각 예측한 결과에 대해서 가중치를 부여하여 해당 가중치를 적용해서 학습되었다.

Randomized Search
CV 사용!

⑤ Xgboost 사용한 parameter 설정

max_depth=8, n_estimators=500, min_child_weight=1000,
subsample_bytree=0.7, subsample=0.7, eta=0.3, seed=0

(=learning_rate)

이 경우에는
Grid Search CV로
최적의 hyperparameters
찾아야함.
(0.1, 0.5, 1.5, 500, 1000)
다시하고 진행할 예정

- learning_rate 만큼마다 트레이닝 데이터에 가중치를 부여한다.

- 이를 0.1로 나누어 설정하면 망울증을 초기화하는데 학습하기 위해서 많은 트레이닝 필요하지만 예측이 성능은 일반적으로 높아지게 된다.

learning_rate
적당한 범위 → 트레이닝 × → 과적합

learningrate
너무작으면 → 트레이닝 × → 과적합

→ Overfitting이 될까?

test data로 테스트하는

예측이 아래에 있으면 좋음

적당한 범위로 설정하자.

2-2. Linear Regression

- Python Linear Regression은 평균과 rmse(= \sqrt{mse})이 매우 좋다.

↳ chart 등에 feature engineering을 진행할 때다.

예를 들어서 shop에 대한 item_cnt, item_cnt shifted 1, item_trend, mean-item_cnt, shop-mean의 차이를 이용해보자.

2-3. Random Forest

- Bagging의 장점을 적용한 decision tree 기반의 알고리즘이다.

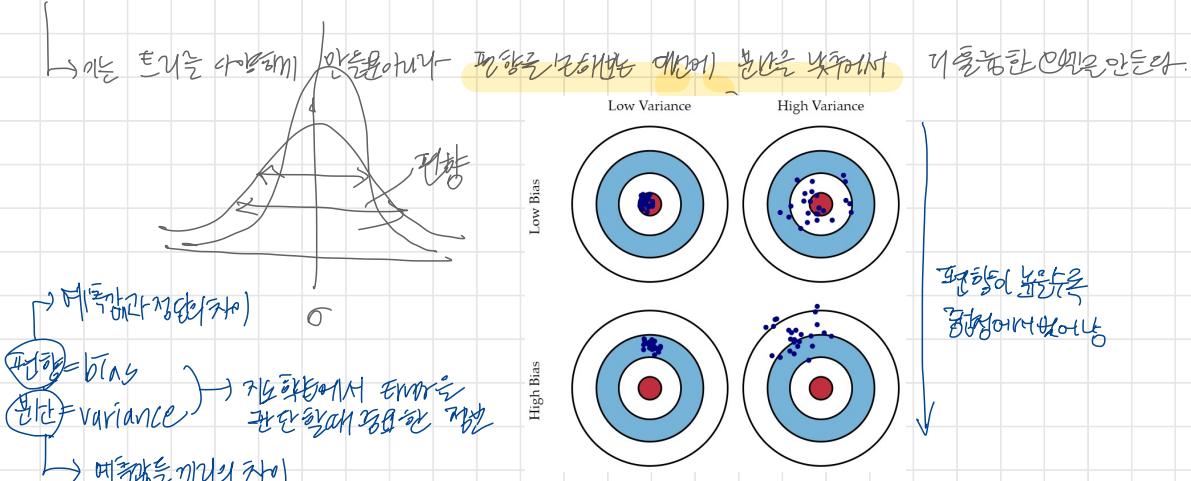
- max_samples를 100% 대로 켜고 설정한다.

- Random Forest 알고리즘은 트리의 노드를 분할할 때에 각각의 특성에서 최적의 특성을 찾는 대신
여러개의 선택한 특성들 중에서 최적의 특성을 찾는 노드의 유연성을 조합한다.

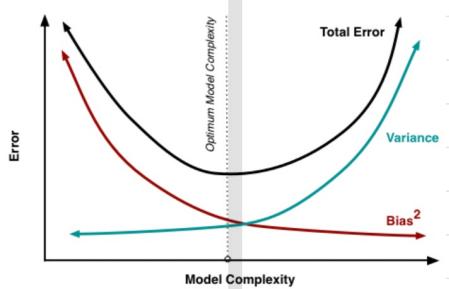
Hyperparameters
 n_estimators=50, max_depth=7, random_state=0, n_jobs=-1

교차검증

정밀도 72
MM 100%



표현력-분산 Trade off



모델 복잡성은 분산에 영향 미친다
분산이 더 커짐

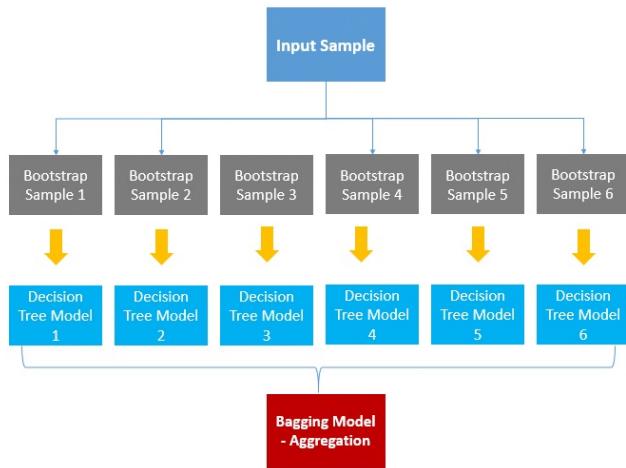
모델 복잡성이 높아지면 예측 오류가 되는 확률이 줄어들게 된다.

2-4. Adaboost

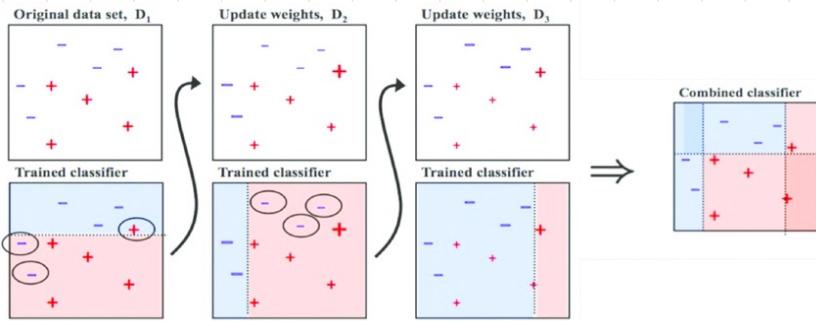
3. Ensemble Models

3-1. DNN

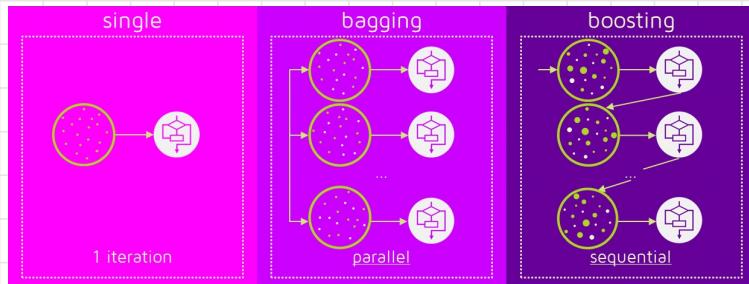
3-2. Bagging



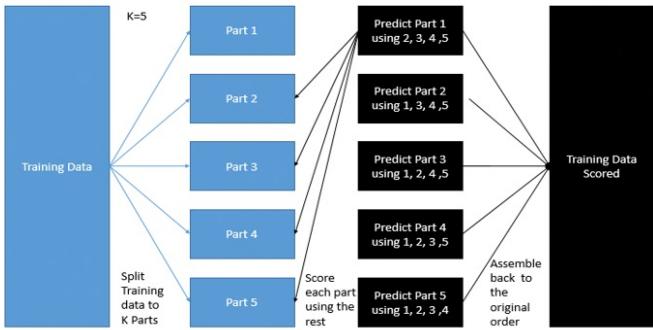
3-3. Boosting



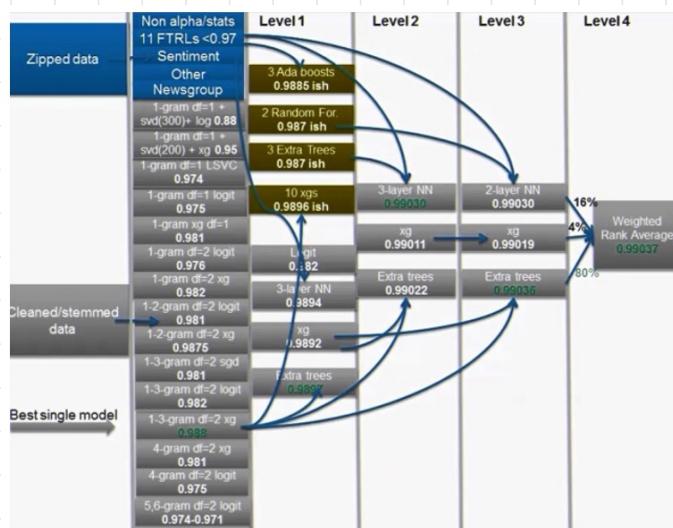
Model For Final Output



↓
sample² 험제 대입하면서 예측
보통 10개를 적용한다.
Bagging Method은 off-line



k-fold ~~fitting~~



4. Evaluating

5. Final Result