🔷

3주차

조 이름 : 10조

1. **N개의 샘플로 구성된 원 데이터에서 N 개의 부트스트래핑 샘플을 생성한다고 할 때, N 이 충분히 클 때 OOB 데이터의 비율을 구하시오.**

- 주어진 샘플을 리샘플하는 방식 → 원 샘플에서 복원 추출 방식으로 새로운 샘플을 생성하고 이때 각 샘플을 원래 샘플에서 무작위로 선택
- 알수 없는 모집단과 정상적으로 작동하지 않는 매개변수 추정량 특성에 대해 보다 많은 정보를 밝히기 위해 사용하는 방식

< 부트스트래핑 샘플링 방식 >

- OOB (Out of Bag) : 원 데이터가 어떤 부트스트래핑 샘플에도 포함되지 않을 확률

(1,2,3,4,5) 와 같이 5개의 자료가 주어졌을 때, 3이란 자료가 추출되지 않을 확률은 동사에 (1,2,4,5) 중 하나가 추출될 확률이므로 80% 이다. 5개의 자료가 주어져 있으므로 5개의 부트스트랩 샘플을 만든다고 하면 3번 데이터가 5개의 샘플에 포함되지 않을 확률 즉, 3번 자료가 한 번도 추출되지 않을 확률은 (80%)^5이다. 이를 일반화 하면 N 이 충분히 클 때 N 개의 데이터에서 N 개의 부트스트래핑 샘플을 만든다고 할 때 OOB는 아래와 같다.

$$\lim_{N\to\infty}\left(\frac{N-1}{N}\right)^N = \lim_{N\to\infty}\left(\frac{1}{\frac{N-1+1}{N-1}}\right)^N = \lim_{N\to\infty}\left(\frac{1}{1+\frac{1}{N-1}}\right)^N =$$

$$\therefore \frac{1}{e}$$

2. **6장 되새김 문제 2번**

각 노드에서 평균 제곱 오차를 최소화하는 방향을 분기를 수행하는 회귀 트리를 구현한다.

- 당뇨병 데이터셋

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

X,y = load_diabetes(return_X_y=True, as_frame=True)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_


train = pd.concat([X_train,y_train],axis=1)
X_cols,y_col=X.columns.tolist(),y.name
```

- 최대 깊이 4, 분할을 수행하는 최소 노드 크기를 4로 하는 결정 트리 회귀 모델을 구현

```
max_depth = 4
min_samples_split = 4

def eval_mse(left,right,y_col):
  mse,n1,n2=0, len(left), len(right)
  score = ((left[y_col] - left[y_col].mean())**2).mean()
  mse += score * n1 / (n1+n2)
  score = ((right[y_col] - right[y_col].mean())**2).mean()
  mse += score * n1 / (n1+n2)
  return mse

def eval_y(df,y_col):
  return df[y_col].mean()

tree = []
for i in range(0, 2**(max_depth + 1)):
  tree.append(dict({'struct':None}))
```

```
tree[1]['struct'] = train


for i in range(1, len(tree)):
  if not isinstance(tree[i]['struct'], pd.DataFrame):
    continue

  if i>= 2**max_depth:
    tree[i]['struct'] = eval_y(tree[i]['struct'],y_col)
    continue

  data = tree[i]['struct']
  a, b, c, d, e = '', float('inf'), float('inf'), None, None

  for X_col in X_cols:
    vals = np.sort(data[X_col].unique())

    for val in vals[1:]:
      left, right = data[data[X_col]<val], data[data[X_col]>=
      mse = eval_mse(left,right,y_col)

      if mse < c:
        a,b,c,d,e = X_col, val, mse, left, right

  tree[i]['col']=a
  tree[i]['val']=b

  if len(d) >= min_samples_split :
    tree[i << 1]['struct'] = d
  else :
    tree[i << 1]['struct'] = eval_y(d,y_col)

  if len(e) >= min_samples_split :
    tree[(i << 1) +1]['struct'] = e
  else :
    tree[(i << 1) + 1]['struct'] = eval_y(e,y_col)
```

- 평균 오차 평균 산출

```
error=0
for i in range(len(X_train)):
    row=X_train.iloc[i]
    ind=1
    node=tree[ind]
    while isinstance(node['struct'],pd.DataFrame):
        if row[node['col']]<node['val']:ind=ind<<1
        else: ind=(ind<<1)+1
        node=tree[ind]

    y_pred=node['struct']
    error+=np.abs(y_pred-y_train.iloc[i])
print(f'학습 데이터셋 MAE:{error/len(y_train): .2f}')


error=0
for i in range(len(X_test)):
    row=X_test.iloc[i]
    ind=1
    node=tree[ind]
    while isinstance(node['struct'],pd.DataFrame):
        if row[node['col']]<node['val']:ind=ind<<1
        else: ind=(ind<<1)+1
        node=tree[ind]

    y_pred=node['struct']
    error+=np.abs(y_pred-y_test.iloc[i])
print(f'테스트 데이터셋 MAE:{error/len(y_test): .2f}')
```

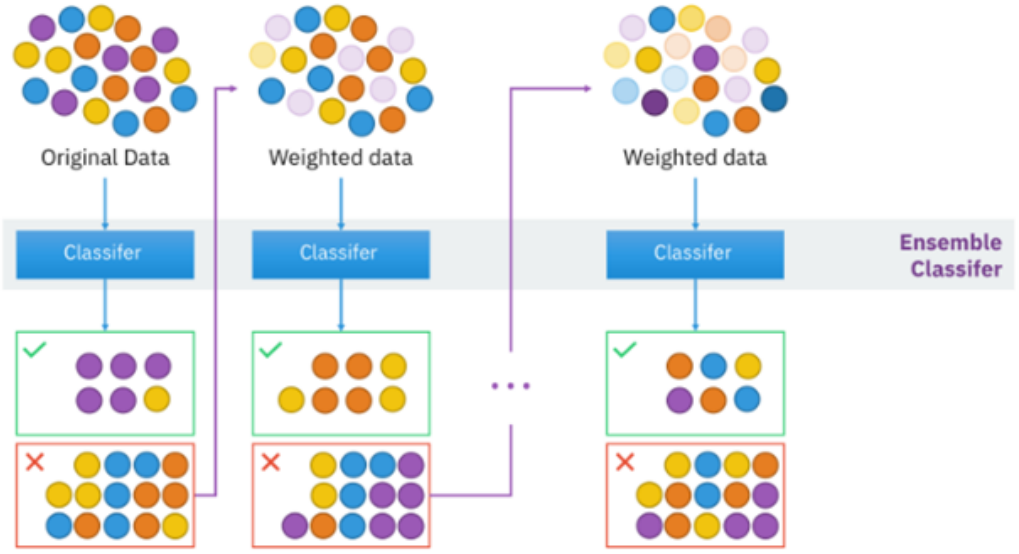
- 결과값

```
학습 데이터셋 MAE: 64.33
테스트 데이터셋 MAE: 65.81
```

3. **그레디언트 부스팅 모델은 과적합에 취약한 이유와 과적합을 방지하는 규제 기법에 대해 서술해 보시오.**

💡 Gradient Boosting Model (GBM)

앙상블 모델 중 하나로, 여러 개의 약한 학습기를 결합하여 강한 학습기를 구축하는 알고리즘
ex) XGBoost, LightGBM, CatBoost



📌 GBM 특징

- **약한 학습기 :** 일반적으로 Decision Trees가 사용되며, 각각은 데이터의 일부 측면에만 초점을 맞춘 약한 모델을 사용함.
- **순차적 학습 :** 각각의 약한 학습기는 이전 모델의 오차를 보완하기 위해 순차적으로 훈련됨.
- **경사 하강법 :** 모델이 예측한 값과 실제 값을 비교하여 오차를 계산하고, 그 오차에 대한 그래디언트(기울기)를 사용하여 새로운 모델을 훈련.

📌 GBM 과적합에 취약한 이유

- **복잡한 구조** : 복잡한 구조로 인해 훈련 데이터 셋의 노이즈까지 학습하여 과적합 발생

- **순차적인 학습** : 이전에 이미 과적합된 상태를 계속해서 이어가며 잔차(경사)를 줄이는 방식으로만 학습하므로 과적합 패턴을 지속적으로 하게될 가능성이 높음.

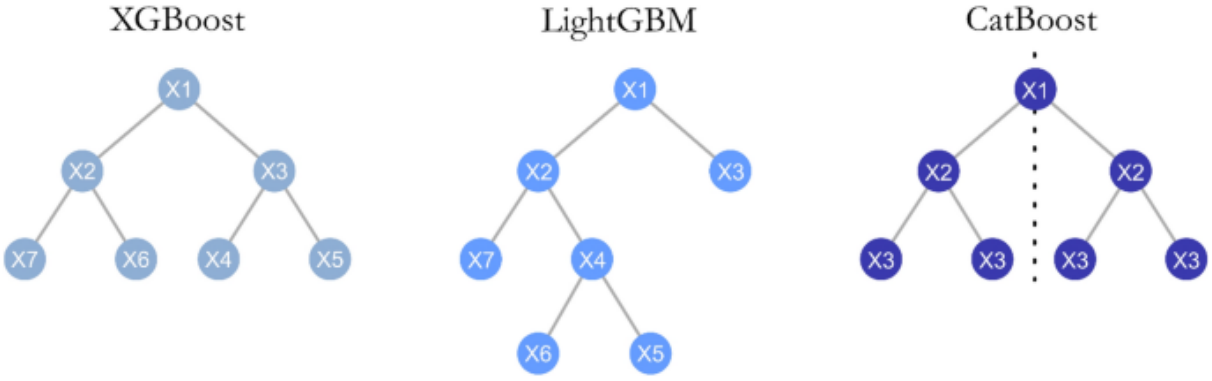
📌 GBM의 과적합을 방지할 규제 기법

- **Tree Constraints :** 트리의 깊이, 분기 수, 잎의 최소 샘플 수 등을 조절하여 모델의 과적합을 방지하거나 모델의 일반화 능력을 향상
- **Shrinkage :** 모든 특성의 계수를 축소하여 모델을 더 간단하게 만들고 예측 성능을 향상시키는 기법으로, 일반적으로 L1 또는 L2 규제 사용하여 모델의 복잡성 조절
- **Random sampling :** 데이터셋에서 무작위로 샘플을 추출하는 방법으로, 모델의 학습이나 평가에 무작위성을 도입하여 모델의 성능 향상.
  ex) 부트스트랩 샘플링
- **Penalized Learning :**모델의 복잡성에 대한 페널티를 부여하여 과적합을 방지하고 일반화 능력을 향상시키는 기법으로 일반적으로 L1 또는 L2 규제를 통해 페널티를 부여하며, 이는 모델의 가중치에 특정 값들을 추가하거나 제약을 가하는 방식

4. XGBoost, LightGBM, CatBoost 모델의 특징을 서술하시오.

- XGBoost, LightGBM, CatBoost 모두 Gradient Boosting Model

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features:** It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100



📌 XGBoost (eXtreme Gradient Boosting)

- Regularization을 통한 과적합 방지
- 트리의 병렬 학습을 지원하여 빠른 속도와 효율성을 제공

📌 LightGBM (Light Gradient Boosting Machine)

- 트리 분할 방법 중 하나로 '균형 트리 분할' 대신 '*리프 중심 트리 분할(Leaf-wise tree growth)*'을 사용

- 적은 메모리 사용량으로 대용량 데이터셋을 처리
- 빠른 학습 속도와 높은 성능 제공

❓ *리프 중심 트리 분할 (Leaf-wise tree growth): 트리의 균형을 맞추지 않고, 최대 손실 값 (max delta loss)을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리가 생성*

📌 CatBoost (Categorical Boosting)

- 범주형 데이터 처리에 강점 가진 알고리즘
- 자동으로 범주형 변수를 처리하고, 원-핫 인코딩 등의 추가 전처리 없이 모델에 통합 가능
- 과적합을 방지하기 위한 강력한 대규모 모델 규제를 제공
- GPU를 이용한 가속화 지원

5. 7장 되새김 문제 3번

퍼뮤테이션 기반 피쳐 중요도를 트리 계열 모델이 아닌 일반 머신러닝 모델의 관점에서 계산한다.

📌 피처 중요도

- 모델이 예측을 수행할 때 각 피처가 예측에 얼마나 영향을 미치는지를 나타내는 지표
- 피처별로 목표 변수 y의 예측에 기여한 중요도 계산
- 중요도가 높은 피처는 목표값과 상관관계가 높은 피처
- mdi, permutation 기법이 존재

📌 퍼뮤테이션 기반 피처 중요도

- 모델의 예측 성능에 특정 피처가 얼마나 기여하는지를 평가하는 방법 중 하나
- 모델이 훈련된 후에 각 피처의 값을 무작위로 섞어서 모델의 성능 변화를 측정하고, 이를 기반으로 피처의 중요도를 계산
- 상관성이 높은 두 중요한 피처 중 한 피처만 높은 피처 중요도를 가지도록 계산하며 퍼뮤테이션 이후 중요치 않은 피처로 성능이 상승할 수 있으며 중요도가 음수값도 가능하다

는 단점

- 대부분의 지도학습에 적용 가능

📌 릿지회귀모델

- L2 정규화를 적용한 선형 회귀 모델
- 가중치 크기를 제한하여 다중공선성 줄임

- 당뇨병 데이터

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

diabetes = load_diabetes(as_frame = True)
X_train, X_val, y_train, y_val = train_test_split(diabetes.da
                                                  diabetes.ta
                                                  random_stat
```

- 학습 데이터셋으로 규제페널티 알바를 0.01로 하는 릿지 회귀모델을 학습

```
from sklearn.linear_model import Ridge
from sklearn.inspection import permutation_importance
import pandas as pd

# 규제패널티 = 0.01 릿지회귀모델 학습
model = Ridge(alpha = 1e-2).fit(X_train, y_train)
```

- 계산한 모델과 검증 데이터셋으로 퍼뮤테이션 기반 피처 중요도 계산, 이때 퍼뮤테이션은 30회 실시하도록 하며 random_state = 0, scoring = 'neg_mean_squared_error'를 적용

```

# 퍼뮤테이션 기반 피처 중요도 계산
```

```
pi = permutation_importance(model, X_val, y_val, n_repeats =


# n_repeats = 300번 만큼 섞음
pi_series = pd.Series(pi.importances_mean, index = X_train.co
print(f'퍼뮤테이션 기반 피처별 중요도: {pi_series.sort_values(ascen
```

- 결과값

```
퍼뮤테이션 기반 피처별 중요도: [1033.90895267  860.19597331  465.99
```

6. **8장 되새김 문제 2번**

과적합이 발생하기 쉬운 GBT 모델에 후진 소거법을 사용하여 과적합을 방지하고 비교한다.

- 함수를 사용하여 가상의 분류 데이터 생성. 데이터에는 100개의 특징이 있고, 이 중에서 30개는 종속변수를 결정하는데 유용한 피쳐의 수, 15개는 독립변수 중 다른 독립 변수의 선형 조합으로 나타나는 피쳐의 수, 5개는 독립변수 중 단순 중복된 피쳐의 수
- 종속변수의 클래스는 2 (2진 분류)

```
import pandas as pd
import numpy as np

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import RFE

X,y=make_classification(n_samples=300,
                        n_features=100,
                        n_informative=30,
                        n_redundant=15,
                        n_repeated=5,
```

```
                            n_classes=2,
                            flip_y=0.05,
                            random_state=1234)

X=pd.DataFrame(X,columns=['feature_'+str(i) for i in range(1,
y=pd.Series(y,name='target')

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=
```

- 후진소거법 없이 전체 피쳐로 모델 학습

```
clf=GradientBoostingClassifier(random_state=1234)
y_pred=clf.fit(X_train,y_train).predict(X_test)


print(f' 정확도 :{(y_pred==y_test).mean()*100:.2f}%')
```

- 결과값

```

정확도 :71.11%
```

- 후진소거법을 사용해 20개의 피쳐만 남겨 모델 학습

```
selector=RFE(clf,n_features_to_select=20,step=1)
selector=selector.fit(X_train,y_train)
```

```
X_train2=X_train.iloc[:,selector.support_]
X_test2=X_test.iloc[:,selector.support_]
```

```
y_pred=clf.fit(X_train2,y_train).predict(X_test2)
print(f' 정확도 :{(y_pred==y_test).mean()*100:.2f}%')
```

- 결과값

```
정확도 :76.67%
```