BDA x 이지스 퍼블리싱 머신러닝 스터디 **2**회차 과제

조 이름 : 8조

<1번 문제**>**

부트스트랩 샘플링은 복원추출을 통해 원 데이터에서 샘플을 무작위로 선택하는 방법이다.
각 샘플이 선택되지 않을 확률은 각선택이 다른 샘플에 영향을 미치지 않는 독립적인 사건으로 간주된다.

부트스트랩에서 하나의 샘플이 선택되지 않을 확률은 $(1-\frac{1}{N})$ 이다.
여러 개의 샘플이 모두 선택되지 않을 확률은 각 샘플의 선택 확률을 모두 곱한 것이다.
따라서 모든 샘플이 선택되지 않을 확률은 $(1-\frac{1}{N})^N$ 이 된다.

N이 크다면 이 값은 e^{-1} 로 수렴하게 되며, 해당 값은 0.368이다.
따라서 모든 샘플이 선택되지 않을 확률은 $1-e^{-1}$ 로, 0.632가 된다.

즉, N이 충분히 크면 부트스트랩 샘플에서 선택되지 않을 샘플의 비율은 63.2%가 된다.

<2번 문제>

2. 6장 되새김 문제 2번

In [2]:
```
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_size=0.33,random_state=1234)
train = pd.concat([X_train,y_train],axis=1)
X_cols,y_col = X.columns.tolist(),y.name
```

In [3]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
```

In [4]:
```
max_depth = 4 #트리의 최대 깊이를 설정
min_samples_split = 4 #노드를 분할하기 위한 최소 샘플 수를 설정

regressor = DecisionTreeRegressor(max_depth=max_depth, min_samples_split=min_samples_split, random_state=1234)
regressor.fit(X_train, y_train) #모델을 학습 데이터에 맞춤.
```

Out[4]:
```
DecisionTreeRegressor(max_depth=4, min_samples_split=4, random_state=1234)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [5]:
```
y_pred = regressor.predict(X_test) #학습한 모델을 사용하여 테스트 데이터에 대한 예측값을 계산
```

In [6]:
```
mae = mean_absolute_error(y_test, y_pred) #테스트 데이터에 대한 실제 값과 예측 값 간의 평균 절대 오차를 계산
```

In [7]:
```
mae #평균 절대 오차(MAE) : 45.6687656115543
```

Out[7]:
```
45.6687656115543
```

<3번 문제**>**

3번

그레이디언트 부스팅 모델은 과적합에 취약하다. 그 이유를 제시하고, 과적합을 방지하는 규제 기법에 대해서 서술해보시오.

과적합에 취약한 이유에는 두 가지가 있습니다.

1. 복잡한 모델 구조: 그레이디언트 부스팅 모델은 결정 트리를 기반으로 하며, 트리의 깊이와 개수가 늘어날수록 모델이 데이터에 더 맞춰져 과적합이 발생할 가능성이 높아집니다.

2. 노이즈와 이상치에 민감: 그레이디언트 부스팅은 학습 데이터에 노이즈나 이상치에 민감할 수 있습니다. 훈련 데이터에 존재하는 작은 노이즈나 이상치가 모델에 큰 영향을 미칠 수 있습니다.

과적합을 방지하기 위해 다양한 규제 기법을 사용할 수 있습니다. 그 중에서도 그레이디언트 부스팅에서 널리 사용되는 규제 기법은

1. 트리의 크기 제한: 각 결정 트리의 깊이나 리프 노드의 개수를 제한함으로써 모델의 복잡성을 줄일 수 있습니다. 이는 트리의 과적합을 방지하고 일반화 성능을 향상시킬 수 있습니다.

2. 축소 (Shrinkage): 학습률을 감소시킴으로써 각 트리의 기여를 축소하는 방법입니다. 작은 학습률은 모델을 더 안정적으로 만들어주고, 과적합을 방지할 수 있습니다.

3. 샘플링: 훈련 데이터에서 무작위로 일부 샘플을 선택하여 사용하는 방법으로, 이를 통해 다양성을 유지하면서 모델을 훈련할 수 있습니다. 이는 과적합을 줄이는 데 도움이 됩니다.

4. 특성 중요도 제어: 모든 특성을 사용하지 않고 중요한 특성만을 선택하여 모델을 구성함으로써, 불필요한 특성에 의한 과적합을 방지할 수 있습니다.

이러한 규제 기법들을 조합하여 적절히 사용함으로써 그레이디언트 부스팅 모델의 과적합을 효과적으로 제어할 수 있습니다.

<4번 문제**>**

4번 XGBoost, LightGBM, CatBoost 모델의 특징을 서술하시오.

1. XGBoost

XGBoost는 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나이다. 유명한 캐글 경연 대회(Kaggle Contest)에서 상위를 차지한 많은 데이터 과학자가 XGBoost를 이용하면서 널리 알려졌다. 압도적인 수치의 차이는 아니지만, 분류에 있어서 일반적으로 다른 머신러닝보다 뛰어난 예측 성능을 나타낸다. XGBoost는 GBM에 기반하고 있지만, GBM의 단점인 느린 수행 시간 및 과적합 규제(Regularizaion) 부재 등의 문제를 해결해서 매우 각광을 받고 있다. 특히 XGBoost는 병렬 CPU 환경에서 병렬 학습이 가능해 기존 GBM보다 빠르게 학습을 완료할 수 있다. XGBoost의 주요 장점은 다음과 같다. 뛰어난 예측 성능, GBM 대비 빠른 수행 시간, 과적합 규제, Tree pruning(나무 가지치기), 자체 내장된 교차 검증, 결손값 자체 처리가 있다.

2. LightGBM

LightGBM의 가장 큰 장점은 XGBoost보다 학습에 걸리는 시간이 훨씬 적다는 점이다. 또한 메모리 사용량도 상대적으로 적다. LightGBM의 Light'는 바로 이러한 장점 때문에 붙여진 것 같다. 하지만 Light'라는 이미지가 자칫 가벼움을 뜻하게 되어 LightGBM의 예측 성능이 상대적으로 떨어진다든가 기능상의 부족함이 있을 것으로 인식되기 쉽다. 마치 소형 자동차 대 중대형 자동차와 같은 비교 이미지로 비칠 수 있지만 실상은 절대 그렇지 않습니다. LightGBM과 XGBoost의 예측 성능은 별다른 차이가 없다. 또한 기능상의 다양성은 LightGBM이 약간 더 많다. 아무래도 LightGBM이 XGBoost보다 2년 후에 만들어지다 보니 XGBoost의 장점은 계승하고 단점은 보완하는 방식으로 개발됐기 때문일 것이다. LightGBM의 한 가지 단점으로 알려진 것은 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽다는 것이다. 적은 데이터 세트의 기준은 애매하지만, 일반적으로 10,000건 이하의 데이터 세트 정도라고 LightGBM의 공식 문서에서 기술하고 있다.

3. CatBoost

CatBoost는 대부분이 범주형 변수로 이루어진 데이터셋에서 예측 성능이 우수한 것으로 알려졌다. CatBoost의 장점은 특이한 범주형 변수 처리 가능, 변수를 그대로 모델에 넣어주면 알아서 Orderd Target Encoding을 진행하는 것이다. 또한 시계열 데이터를 효율적으로 처리해준다. CatBoost의 단점은 메모리 사용량, 높은 학습 시간 복잡도 이다.

<5번 문제>

5번

```
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance   #퍼뮤테이션 기반 피처 중요도 계산
from sklearn.linear_model import Ridge   #릿지 회귀 모델 구현 클래스
```

[문제]

```
diabetes = load_diabetes(as_frame=True)
X_train, X_val, y_train, y_val = train_test_split(diabetes.data,
                                                  diabetes.target,
                                                  random_state=0)

##규제 페널티 alpha=0.01로 하는 릿지 회귀 모델 학습
model = Ridge(alpha=0.01).fit(X_train, y_train)  #릿지 회귀 모델 생성 후 학습 데이터 적합

#p216 참고
# importances = model.feature_importances_

## => p217 랜덤포레스트의 피처 중요도 계산 과정을 참고하여 작성했지만 ridge 모델에는 'feature_importances_'를 제공하지 않는다 하여 오류 남

result = permutation_importance(model,
                                X_val, y_val,
                                n_repeats=30, #퍼뮤테이션 30회 실시
                                random_state=0,
                                scoring='neg_mean_squared_error')
```

-> 답안에서는 result.importances_mean 사용함

```
importances = result.importances_mean
#릿지 모델의 피처 중요도
result_series = pd.Series(importances, index=X_train.columns)
print(f'퍼뮤테이션 기반 피처별 중요도 :
{result_series.sort_values(ascending=False)}')
```

```
퍼뮤테이션 기반 피처별 중요도 : s5     1013.866346
bmi     872.725677
bp      438.662751
sex     277.375913
s1      209.487178
s4       15.823473
s6       13.836377
s3       10.124231
s2       10.093097
age      -9.890948
dtype: float64
```

<6번 문제>

```
import numpy as np
import pandas as pd

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, 101)])
y = pd.Series(y, name='target')

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

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

print(f'전체 피처를 사용한 GBT 모델의 정확도: {(y_pred == y_test).mean()*100:.2f}%')
```

```
전체 피처를 사용한 GBT 모델의 정확도: 66.67%
```

```
clf = GradientBoostingClassifier(random_state=1234)
selector = RFE(clf, n_features_to_select=20, step=1)
selector = selector.fit(X_train, y_train)
selector.support_[:10]
```

```
array([False, False, False,  True, False,  True,  True, False, False,
        True])
```

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

clf = GradientBoostingClassifier(random_state=1234)

y_pred = clf.fit(X_train2, y_train).predict(X_test2)

print(f'RFE 클래스 기반 후진 소거법을 적용한 GBT 모델의 정확도: {(y_pred == y_test).mean()*100:.2f}%')
```

```
RFE 클래스 기반 후진 소거법을 적용한 GBT 모델의 정확도: 72.73%
```