프로젝트 기반 데이터 과학자 양성과정(Data Science)
Machine Learning 및 분석실습

8주차
지도 학습

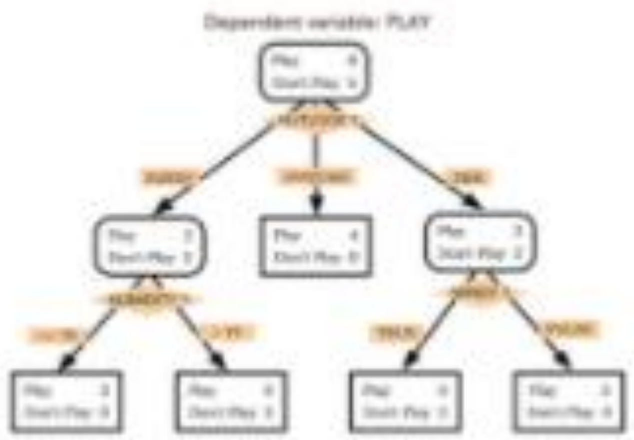
Gradient Boosting

강사 : 최영진

1. Gradient Boosting

❖ Random Forest

- 의사결정 트리의 오버피팅 한계를 극복하기 위한 전략으로 **랜덤 포레스트(Random Forest) 등장**
- 데이터에 의사결정나무 여러 개를 동시에 적용해서 학습성능을 높이는 앙상블 기법
- 동일한 데이터로부터 복원추출을 통해 30개 이상의 데이터 셋을 만들어 각각에 의사결정나무를 적용한 뒤 학습 결과를 취합하는 방식



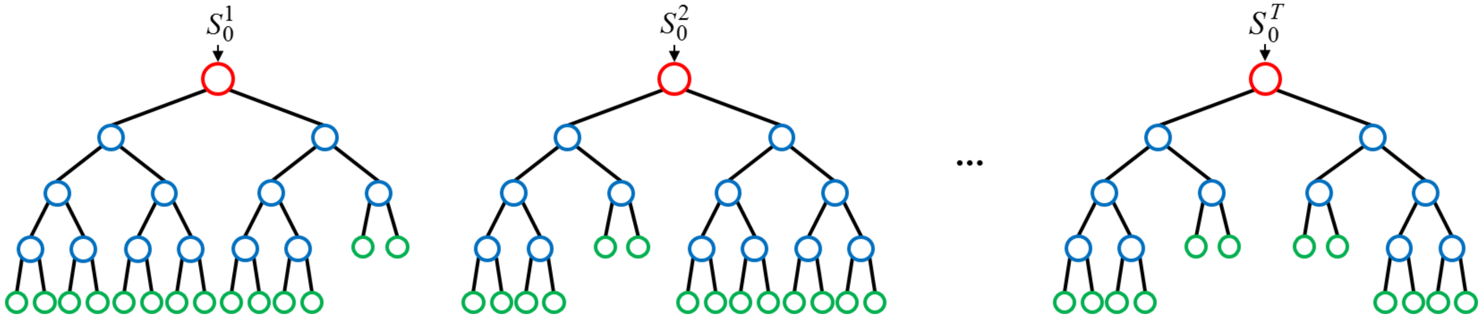
Tree

Random Forest

1. Gradient Boosting

❖ Random Forest

- 배깅(bagging): bootstrap aggregating의 약자로, 부트스트랩(bootstrap)을 통해 조금씩 다른 훈련 데이터에 대해 훈련된 기초 분류기(base learner)들을 결합(aggregating)시키는 방법
- bootstrap sampling(복원추출)을 사용하며 decision tree 생성으로 algorithm으로 진행
- 트리들의 편향은 그대로 유지하면서, 분산은 감소시키기 때문에 포레스트의 성능을 향상



부트스트랩 방법을 통해 T개의 훈련 데이터셋을 생성
T개의 기초 분류기(트리)들을 훈련
기초 분류기(트리)들을 하나의 분류기(랜덤 포레스트)로 결합(평균 또는 과반수투표 방식 이용).

1. Gradient Boosting

❖ Random Forest

- 장점
  - 다양성을 극대화 하여 예측력이 상당히 우수한 편
  - 다수의 트리의 예측 결과를 종합하여 의사결정을 진행하기 때문에 안정성도 상당히 높음
  - 랜덤화(randomization)는 포레스트가 노이즈가 포함된 데이터에 대해서도 강인
- 단점
  - 다수의 트리를 이용한 의사결정 기법을 이용하기 때문에 기존의 트리가 갖는 장점 중 하나인 설명력을 잃음

1. Gradient Boosting

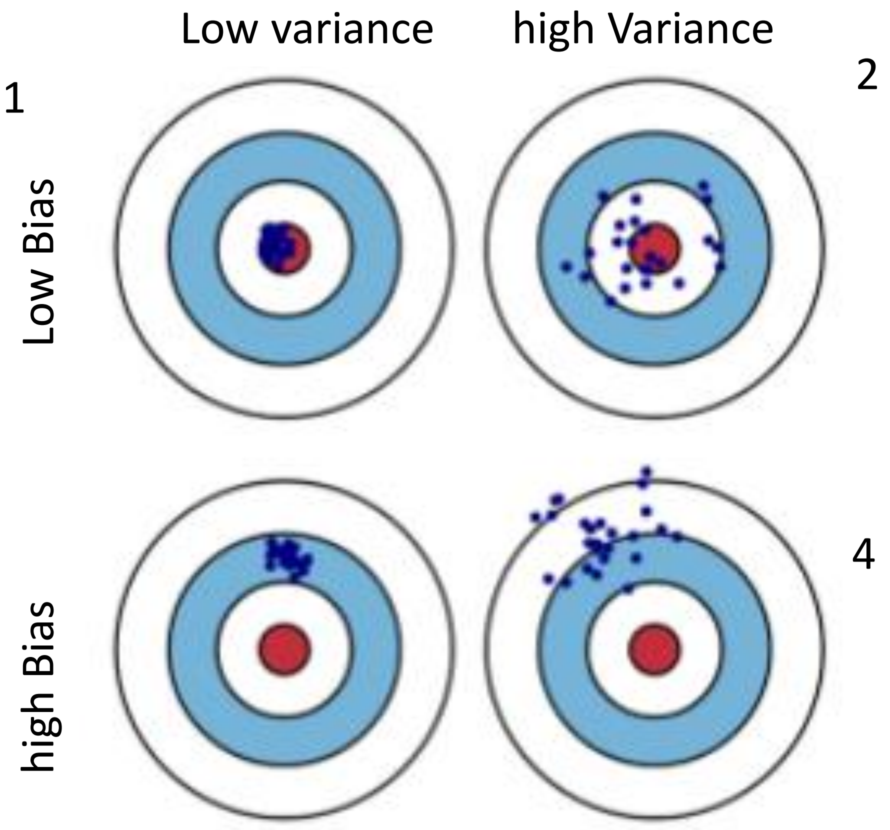
❖ **Ensenbles(앙상블)**

- Error 최소화
  - 다양한 모델의 결과를 종합하여 데이터의 오류를 줄여줌
- Overfitting 감소
  - 각 모델별로 bias가 존재
  - Bias를 종합하여 결과를 생성하고, overfitting을 줄임
- **bias와 variance는 모델의 loss 또는 error를 의미**

1. Gradient Boosting

❖ **Ensenbles(앙상블)**

- **bias와 variance는 모델의 loss 또는 error를 의미**



1번 과녁
예측값들이 대체로 정답 근방 → 편향이 낮음
예측값들끼리 서로 몰릴 경우→ 분산이 낮음

2번 과녁
예측값들이 대체로 정답 근방 →편향이 낮음
예측값들끼리 서로 흩어져 있는 경우 → 분산이 높음

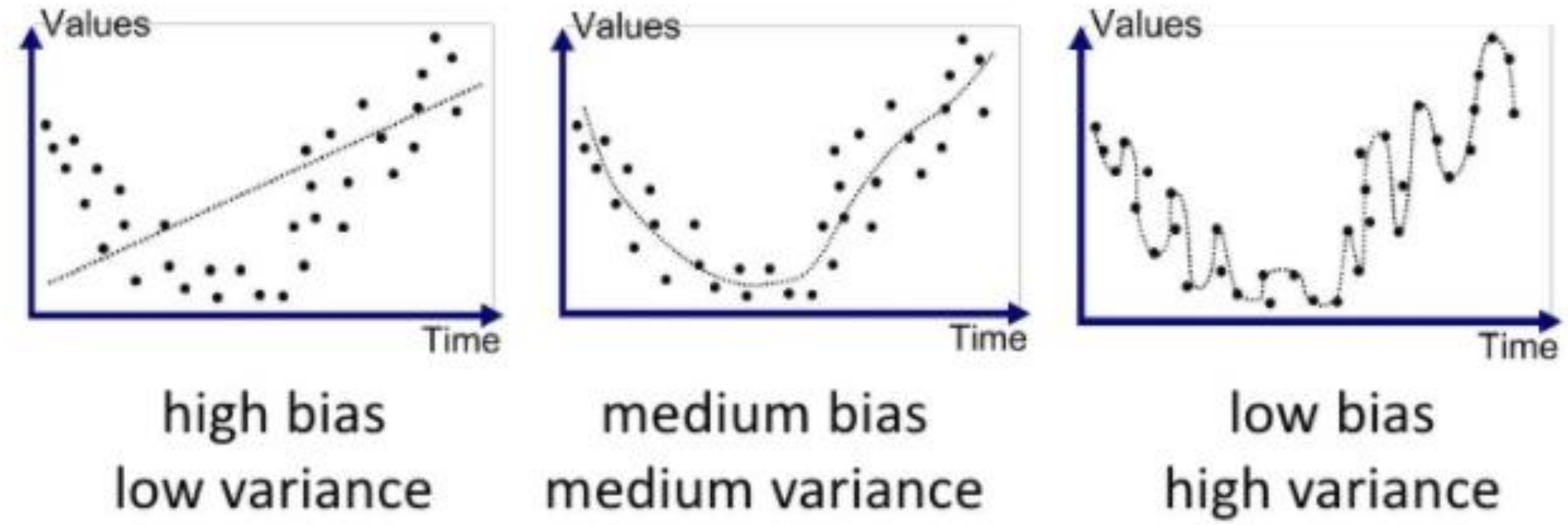
3번 과녁
예측값들이 대체로 정답으로부터 멀어져 있는 경우 → 편향이 높음
예측값들끼리 서로 몰려 있는 경우 → 분산이 낮음

4번 과녁
예측값들끼리 대체로 정답으로부터 멀어져 있는 경우 → 편향이 높음
예측값들끼리 서로 흩어져 있는 경우 → 분산이 높음

1. Gradient Boosting

❖ **Ensenbles(앙상블)**

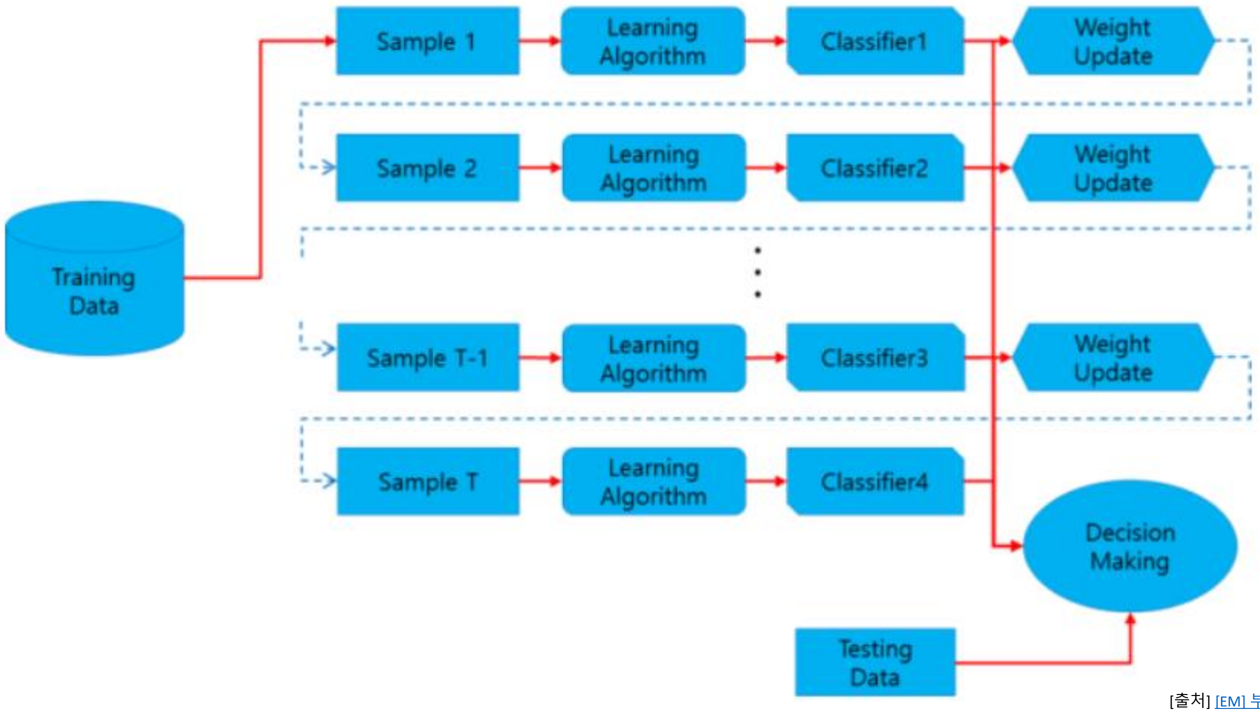
▪ **예측값들과 정답이 대체로 멀리 떨어져 있으면 결과의 편향(bias)이 높음**
예측값들이 자기들끼리 대체로 멀리 흩어져있으면 결과의 분산(variance)이 높음



1. Gradient Boosting

❖ Boosting이란?

- Bagging과 유사하게 초기 샘플 데이터를 조작하여 다수의 분류기를 생성하는 기법 중 하나지만 가장 큰 차이는 순차적(Sequential)방법
- 이전 분류기의 학습 결과를 토대로 다음 분류기의 학습 데이터의 샘플가중치를 조정해 학습을 진행하는 방법

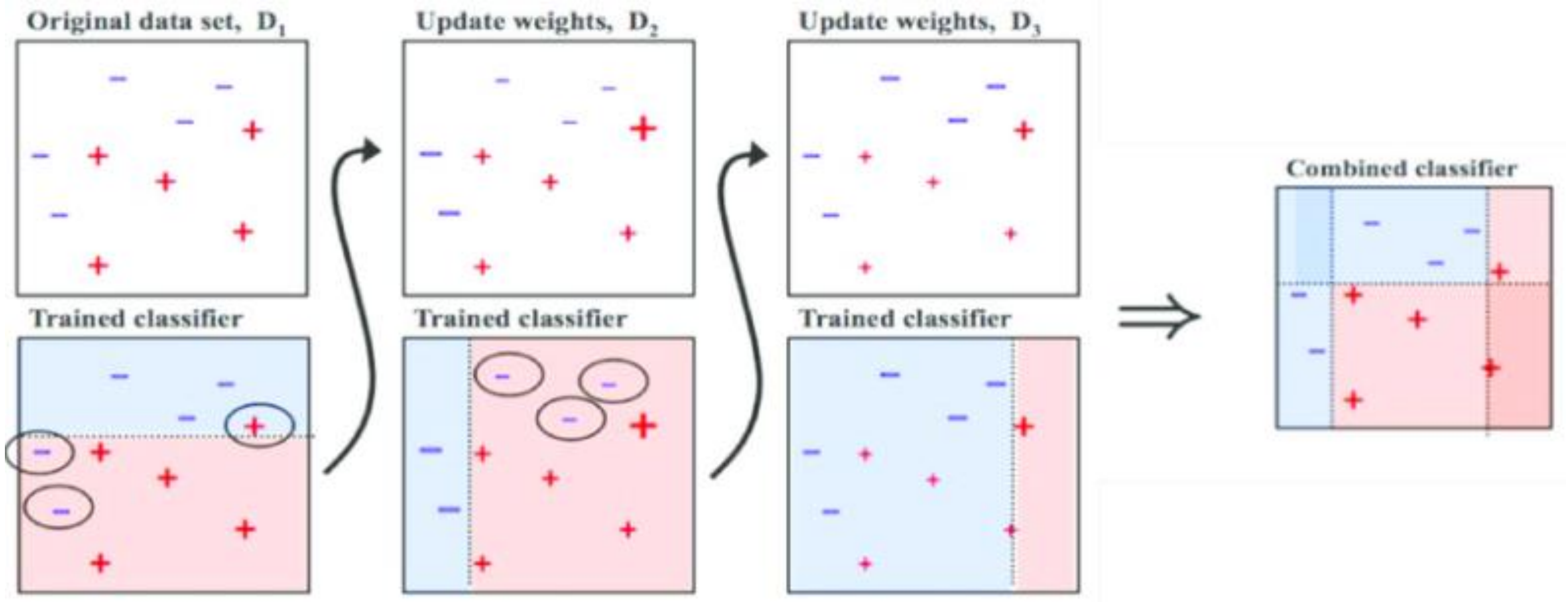


[출처] [EM] 부스팅(Boosting) 이론

1. Gradient Boosting

❖ Boosting이란?

- Bagging과 유사하게 초기 샘플 데이터를 조작하여 다수의 분류기를 생성하는 기법 중 하나지만 가장 큰 차이는 순차적(Sequential)방법
- 이전 분류기의 학습 결과를 토대로 다음 분류기의 학습 데이터의 샘플가중치를 조정해 학습을 진행하는 방법
- 약한 분류기를 결합하여 강한 분류기를 만드는 과정



1. Gradient Boosting

❖ **Boosting이란?**

- 부스팅의 대표적인 모델은 AdaBoost, Gradient Boost등
- Gradient Boost의 변형 모델로는 XGBoost, LightGBM, CatBoost 등

비교	특징
Adaboost	정답분류 및 오답에 가중치 부여
Gradient Boosting	lossFunction에 gradient를 통해 오답에 가중치 부여
Xgboost	GBM 대비 성능향상 시스템 자원 효율적 운용 Kaggle을 통한 성능검증(상위랭커)
Light GBM	Xgboost 대비 성능향상 2000개 이상의 대용량 데이터 사용가능

1. Gradient Boosting

❖ **Boosting과 bagging 비교**

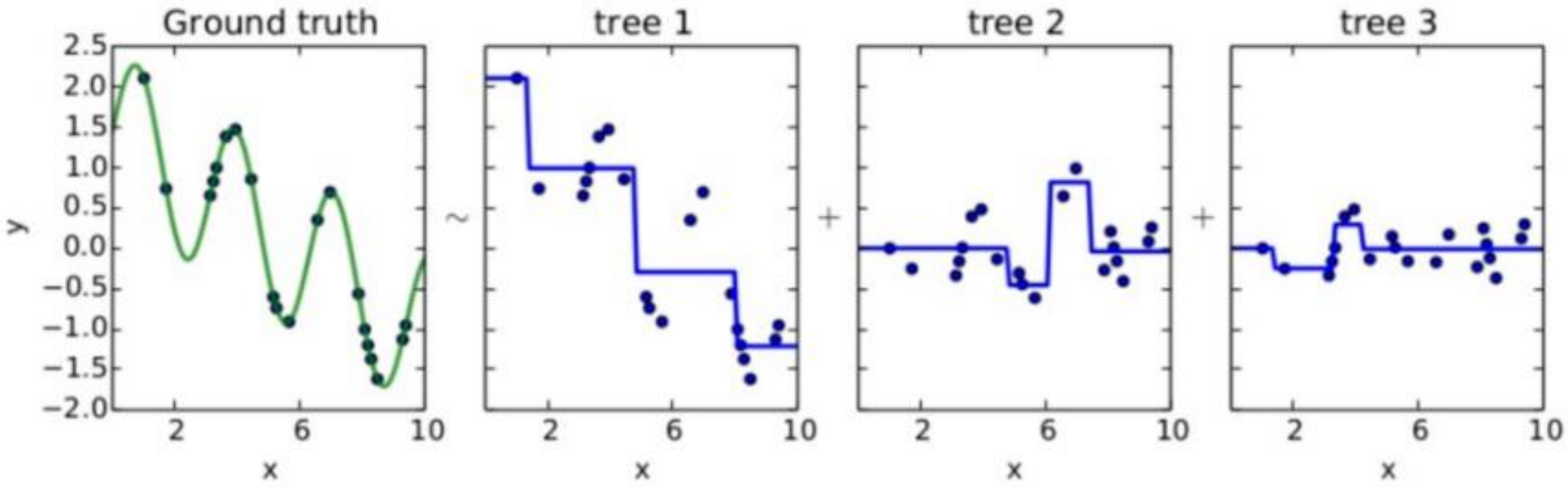
비교	Bagging	Boosting
특징	병렬 앙상블모델	연속 앙상블 모델
적합한 상황	High Variance , Low Bias	Low variance, High bias
대표알고리즘	RandomForest	Gradient Boosting Xgboosting
Sampling	Random sampling	RandomSampling with on error

https://people.cs.pitt.edu/~milos/courses/cs2750-Spring04/lectures/class23.pdf

https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

1. Gradient Boosting

❖ Gradient Boosting

- **약한 여러 개의 결정트리를 이용하여 강력한 모델을 만듬**
- Residual fitting의 방식 예측하는 타겟 추정 값은 모든 타겟 값의 평균
  tree1 를 통해 y를 예측하고 남은 잔차 (residual)을 다시 tree2 라는 모델을 통해 예측하고
  tree1+tree2 모델을 통해 y를 예측한다면 tree1보다 나은 tree2 모델
  tree2+tree3 모델을 통해 y를 예측한다면 tree2보다 나은 tree3 모델
- 

1. Gradient Boosting

❖ Gradient Boosting

- **약한 여러 개의 결정트리를 이용하여 강력한 모델을 만듬**
- powerful한 pre-pruning이 사용되며 1~5 정도 깊이의 tree를 사용하므로 메모리를 적게 사용하고 예측도 빠름
- parmeter설정에 random forest보다 조금 더 민감하지만 잘 조정하면 높은 정확도를 제공
- parameter는 이전 트리의 오차를 얼마나 강하게 보정할 것인가를 제어하는 **learning_rate**

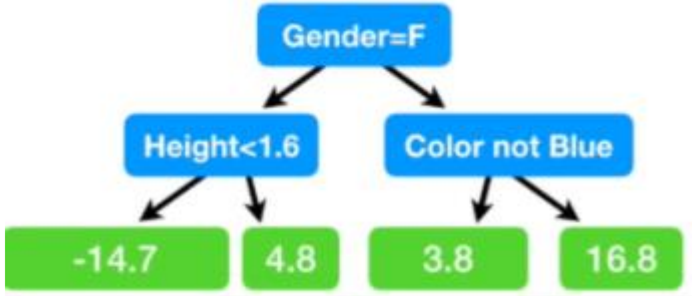
1. Gradient Boosting

❖ Gradient Boosting

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

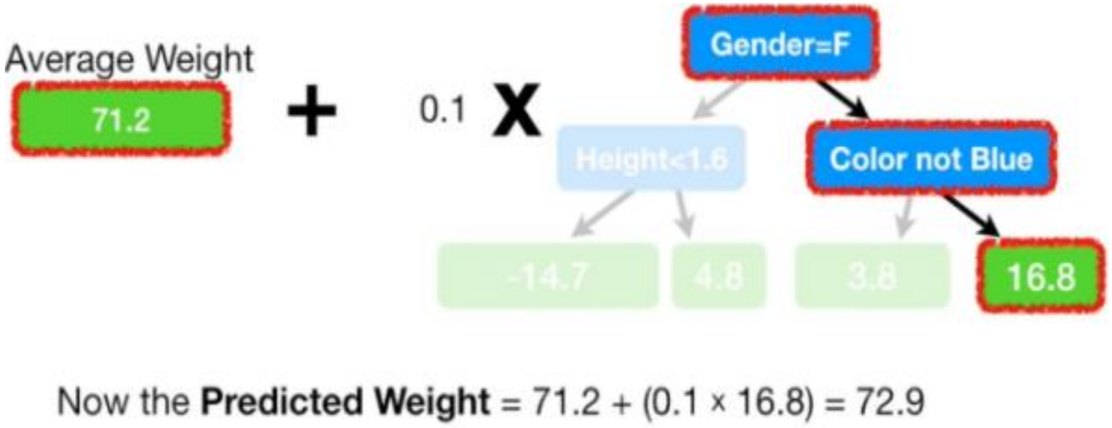
(88 + 76 + 56 + 73 + 77 + 57) / 6 = 71.2



초기에 구한 트리(여기서는 single leaf)와 두번째로 구한 트리를 조합하여 몸무게를 예측

1. Gradient Boosting

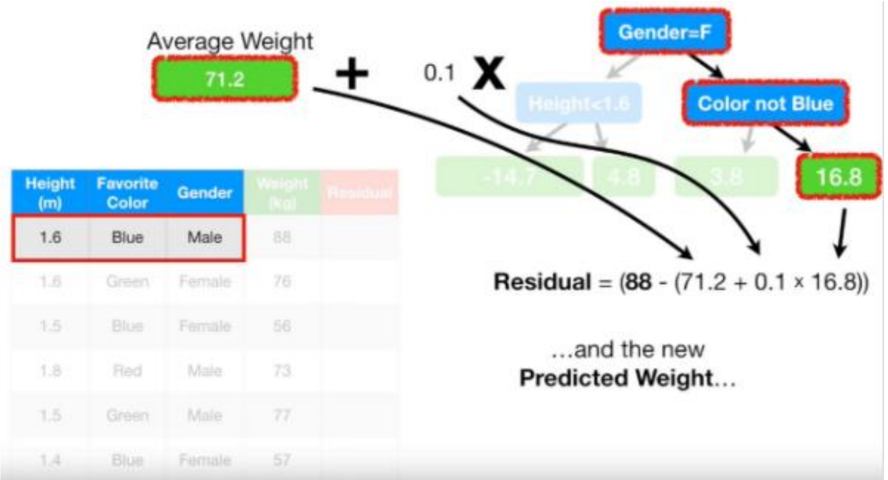
❖ 과적합 문제를 해결하기 위해 학습률(Learning Rate)이라는 것을 활용



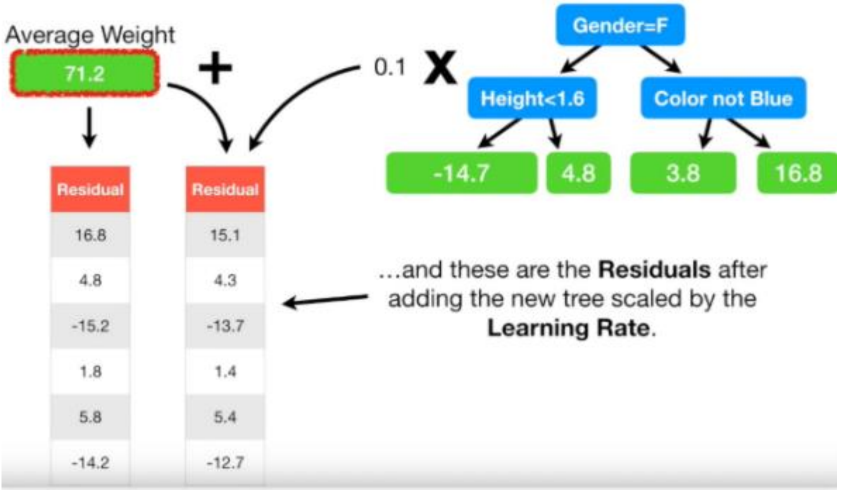
0.1일 경우 모델이 예측한 몸무게는 72.9
single leaf 모델이 예측한 71.2kg보다는 실제 값에 근접

1. Gradient Boosting

❖ 과적합 문제를 해결하기 위해 학습률(Learning Rate)이라는 것을 활용

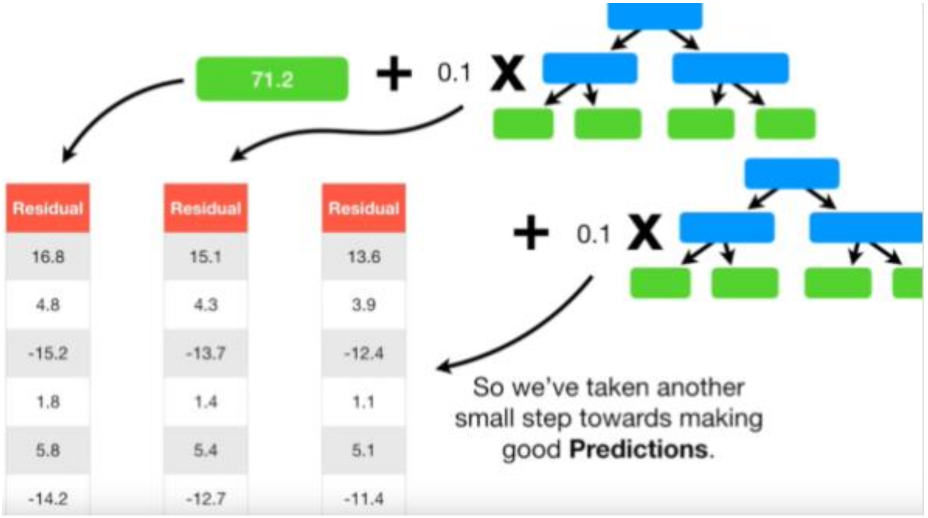


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7



1. Gradient Boosting

❖ 과적합 문제를 해결하기 위해 학습률(Learning Rate)이라는 것을 활용



1. Gradient Boosting

❖ Gradient Boosting 실습

- from sklearn.datasets import make_classification
- from sklearn.ensemble import GradientBoostingClassifier
- clf = GradientBoostingClassifier(random_state=0)
- clf.fit(X_train, y_train)
- clf.predict(X_test[:2])
- clf.score(X_test, y_test)
-

1. Gradient Boosting

❖ Gradient Boosting 실습

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn

cancer = load_breast_cancer()
col_names = cancer.feature_names
print(len(col_names))
X_Data = pd.DataFrame(cancer.data,columns = col_names )
y = pd.DataFrame(cancer.target)

X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=42)
```

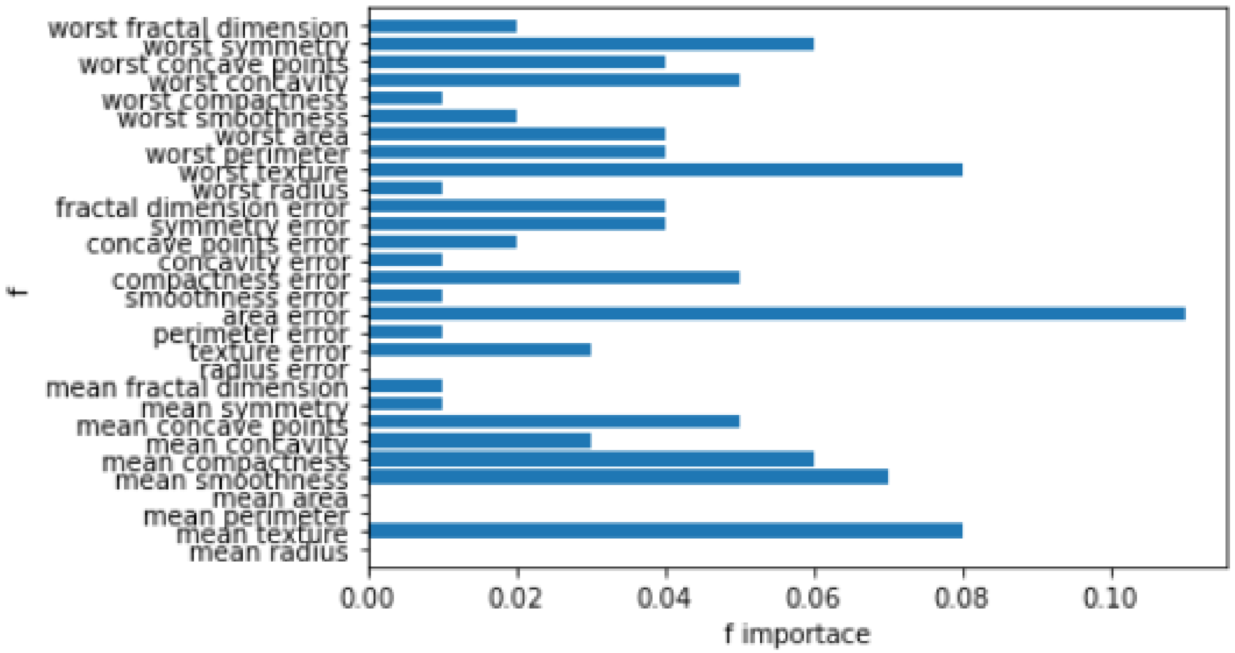
```
print("훈련 세트 정확도: {:.3f}".format(ada.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(ada.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.986
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```
n_features = cancer.data.shape[1]

plt.barh(range(n_features), ada.feature_importances_, align='center')
plt.yticks(np.arange(n_features), cancer.feature_names)
plt.xlabel("f importace")
plt.ylabel("f")
plt.ylim(-1, n_features)
plt.show()
```



1. Gradient Boosting

❖ Gradient Boosting 실습

```python
from sklearn.ensemble import GradientBoostingClassifier
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.965
```

```python
gbrt = GradientBoostingClassifier(random_state=0, max_depth=2)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.972
```

```python
gbrt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
훈련 세트 정확도: 0.988
테스트 세트 정확도: 0.965
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```python
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=50, learning_rate=learning_rate, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_test, y_test)))
```

```
Learning rate:  0.05
Accuracy score (training): 0.984
Accuracy score (validation): 0.958
Learning rate:  0.075
Accuracy score (training): 0.993
Accuracy score (validation): 0.958
Learning rate:  0.1
Accuracy score (training): 0.995
Accuracy score (validation): 0.958
Learning rate:  0.25
Accuracy score (training): 1.000
Accuracy score (validation): 0.979
Learning rate:  0.5
Accuracy score (training): 1.000
Accuracy score (validation): 0.972
Learning rate:  0.75
Accuracy score (training): 1.000
Accuracy score (validation): 0.965
Learning rate:  1
Accuracy score (training): 1.000
Accuracy score (validation): 0.944
```

1. Gradient Boosting

❖ Gradient Boosting 실습

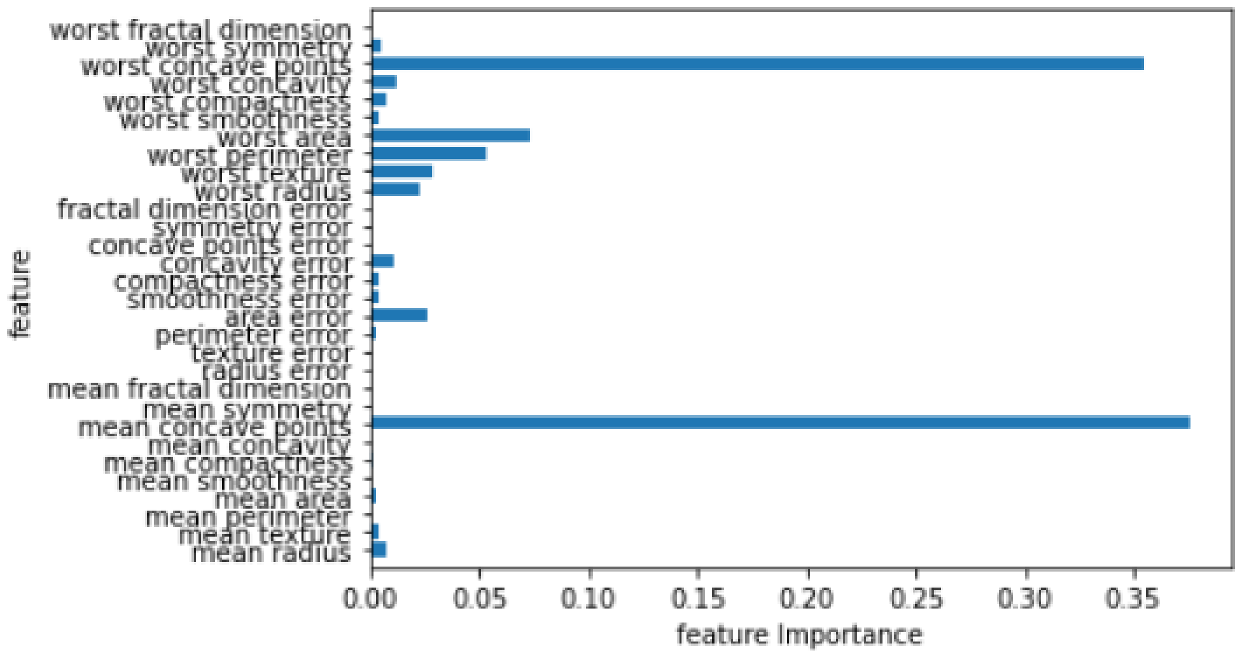
```python
from sklearn.metrics import classification_report
gb_clf2 = GradientBoostingClassifier(n_estimators=50, learning_rate=0.25, max_depth=2, random_state=0)
gb_clf2.fit(X_train, y_train)
predictions = gb_clf2.predict(X_test)

print("Classification Report")
print(classification_report(y_test, predictions))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        53
           1       0.98      0.99      0.98        90

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```
n_features = cancer.data.shape[1]

plt.barh(range(n_features), gb_clf2.feature_importances_, align='center')
plt.yticks(np.arange(n_features), cancer.feature_names)
plt.xlabel("feature Importance")
plt.ylabel("feature")
plt.ylim(-1, n_features)
plt.show()
```



1. Gradient Boosting

❖ Gradient Boosting 실습

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn
from sklearn.datasets import load_wine

wine = load_wine()
col_names = wine.feature_names
print(len(col_names))
X_Data = pd.DataFrame(wine.data,columns = col_names )
y = pd.DataFrame(wine.target)
X_train, X_test, y_train, y_test = train_test_split(
    wine.data, wine.target, test_size=0.3,random_state=0)
```

```
13
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```python
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=42)
```

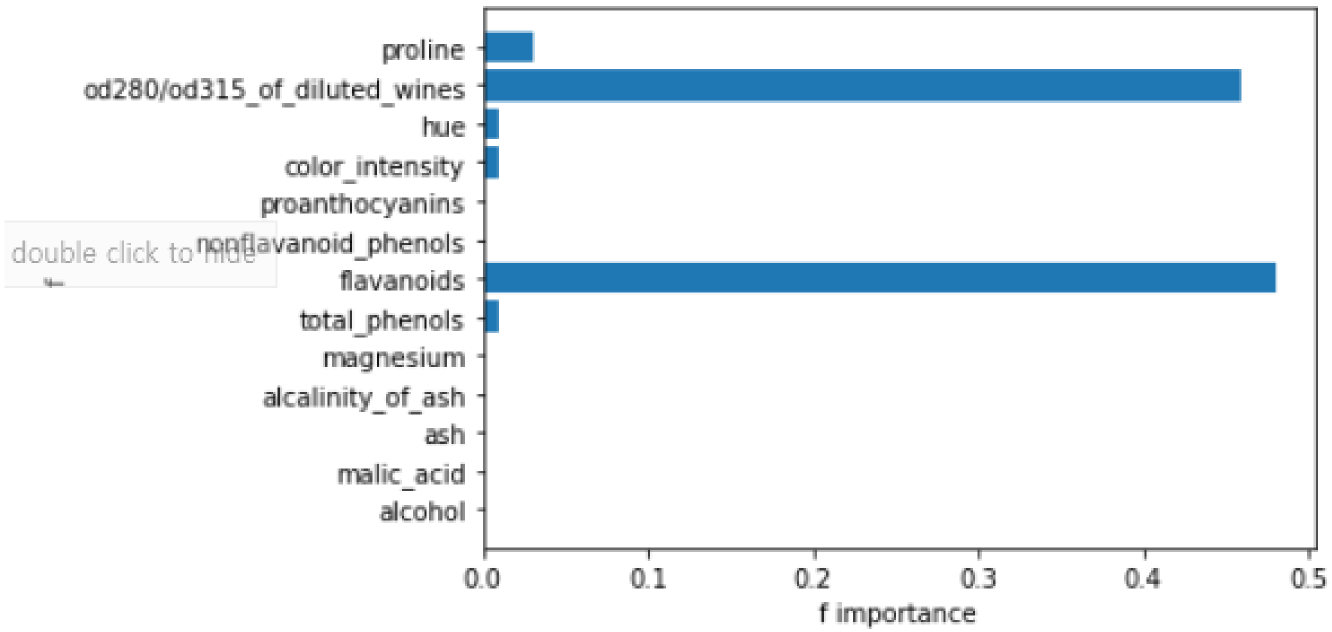
```python
print("훈련 세트 정확도: {:.3f}".format(ada.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(ada.score(X_test, y_test)))
```

```
훈련 세트 정확도: 0.976
테스트 세트 정확도: 0.889
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```
n_features = wine.data.shape[1]

plt.barh(range(n_features), ada.feature_importances_, align='center')
plt.yticks(np.arange(n_features), wine.feature_names)
plt.xlabel("f importance")
plt.ylabel("f")
plt.ylim(-1, n_features)
plt.show()
```



1. Gradient Boosting

❖ Gradient Boosting 실습

```python
from sklearn.ensemble import GradientBoostingClassifier
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.963
```

```python
gbrt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.944
```

1. Gradient Boosting

❖ Gradient Boosting 실습

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=50, learning_rate=learning_rate, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_test, y_test)))
```

```
Learning rate:  0.05
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
Learning rate:  0.075
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
Learning rate:  0.1
Accuracy score (training): 1.000
Accuracy score (validation): 0.981
Learning rate:  0.25
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
Learning rate:  0.5
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
Learning rate:  0.75
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
Learning rate:  1
Accuracy score (training): 1.000
Accuracy score (validation): 0.963
```

1. Gradient Boosting

❖ Gradient Boosting 실습

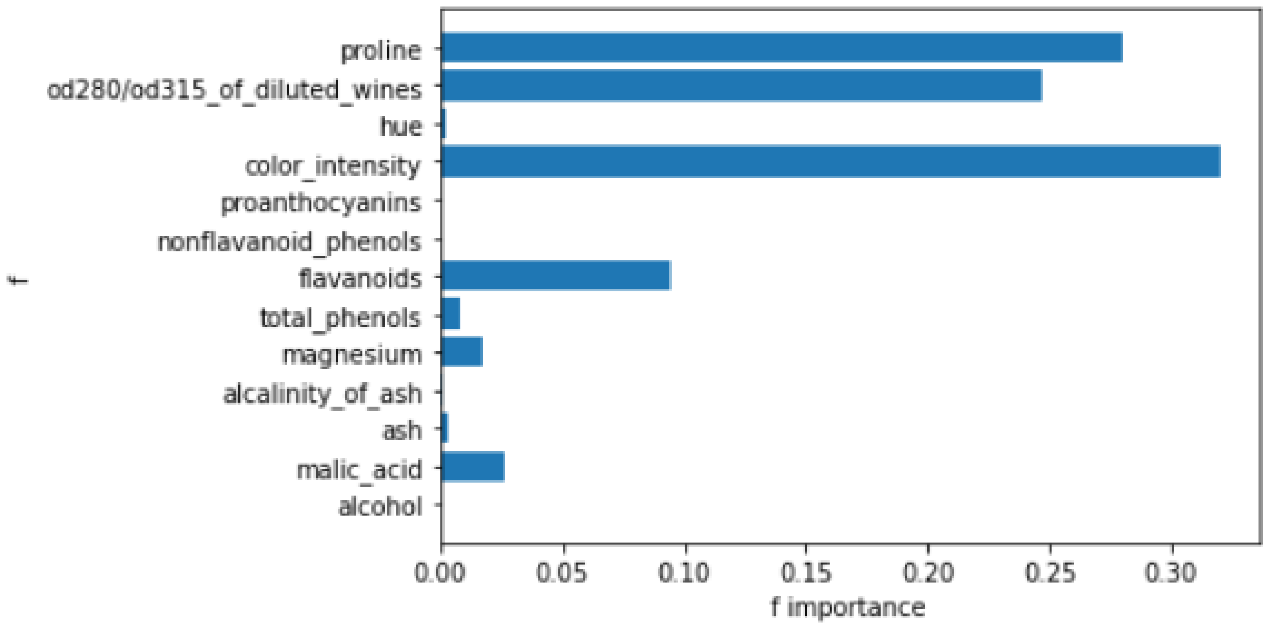
```
from sklearn.metrics import classification_report
gb_clf2 = GradientBoostingClassifier(n_estimators=50, learning_rate=0.1, max_depth=2, random_state=0)
gb_clf2.fit(X_train, y_train)
predictions = gb_clf2.predict(X_test)

print("Classification Report")
print(classification_report(y_test, predictions))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.95      0.98        22
           2       0.93      1.00      0.96        13

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

1. Gradient Boosting

❖ Gradient Boosting 실습

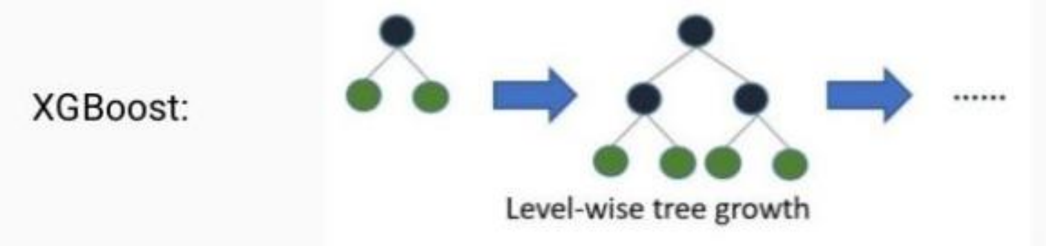
```
n_features = wine.data.shape[1]

plt.barh(range(n_features), gb_clf2.feature_importances_, align='center')
plt.yticks(np.arange(n_features), wine.feature_names)
plt.xlabel("f importance")
plt.ylabel("f")
plt.ylim(-1, n_features)
plt.show()
```



1. Gradient Boosting

❖ xgBoosting

- 트리기반의 앙상블 학습중 가장 최고의 모델로 **Gradient Boosting보다 빠름**
- **과적합을 방지와 분류. 회귀 둘다 가능하고 조기 종료(early stopping) 를 제공함**
- **Gradient boosting 기반으로** 학습시간이 느림/ 하이퍼 파라미터 존재
- Level-wise : 각 노드는 root노드와 가까운노드를 선회 / 수평성장
- 최대한 균형 잡힌 트리를 유지하며 분할하여 트리의 깊이를 최소화하여 오버피팅에 강한구조이지만 균형을 맞추기 위한 시간필요



n_estimators(혹은 num_boost_round) : 결정 트리의 개수

max_depth : 트리의 깊이

colsample_bytree : 컬럼의 샘플링 비율(random forest의 max_features와 비슷)

subsample : weak learner가 학습에 사용하는 데이터 샘플링 비율

learning_rete : 학습률

min_split_loss : 리프 노드를 추가적으로 나눌지 결정하는 값

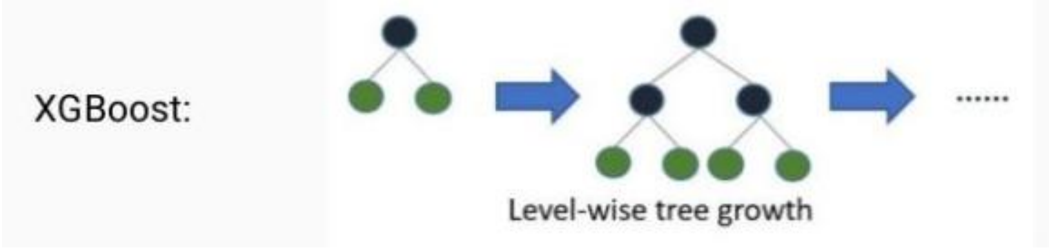
reg_lambda : L2 규제

reg_alpha : L1 규제

1. Gradient Boosting

❖ xgBoosting

- GBM의 경우 n_estimators에 지정된 횟수만큼 학습을 끝까지 수행하지만, XGB의 경우 오류가 더 이상 개선되지 않으면 수행을 중지
- n_estimators 를 200으로 설정하고, 조기 중단 파라미터 값을 50으로 설정하면, 1부터 200회까지 부스팅을 반복후 학습오류가 감소하지 않으면 더 이상 부스팅을 진행하지 않고 종료합



n_estimators(혹은 num_boost_round) : 결정 트리의 개수

max_depth : 트리의 깊이

colsample_bytree : 컬럼의 샘플링 비율(random forest의 max_features와 비슷)

subsample : weak learner가 학습에 사용하는 데이터 샘플링 비율

learning_rete : 학습률

min_split_loss : 리프 노드를 추가적으로 나눌지 결정하는 값

reg_lambda : L2 규제

reg_alpha : L1 규제

1. Gradient Boosting

❖ xgBoosting 설치

- #콘다환경에서 설치
- conda install -c anaconda py-xgboost
- pip install xgboost

1. Gradient Boosting

❖ xgBoosting 실습

- from xgboost import plot_importance
- from xgboost import XGBClassifier
- clf=xgb.XGBClassifier() # 파라미터 넣어줌. 모델생성
- clf.fit() # 파라미터 넣어줌. 데이터학습.
- evals_result=clf.evals_result()
- clf.feature_importances_
- clf.predict()

1. Gradient Boosting

❖ xgBoosting 실습

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn

cancer = load_breast_cancer()
col_names = cancer.feature_names
print(len(col_names))
X_Data = pd.DataFrame(cancer.data,columns = col_names )
y = pd.DataFrame(cancer.target)

X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

❖ xgBoosting 실습

```
from xgboost import plot_importance
from xgboost import XGBClassifier

import xgboost as xgb
```

```
xgbb = XGBClassifier(n_estimators=500,learning_rate=0.01, max_depth=5 ,random_state=42)
xgbb.fit(X_train, y_train)
```

```
XGBClassifier(learning_rate=0.01, max_depth=5, n_estimators=500,
              random_state=42)
```

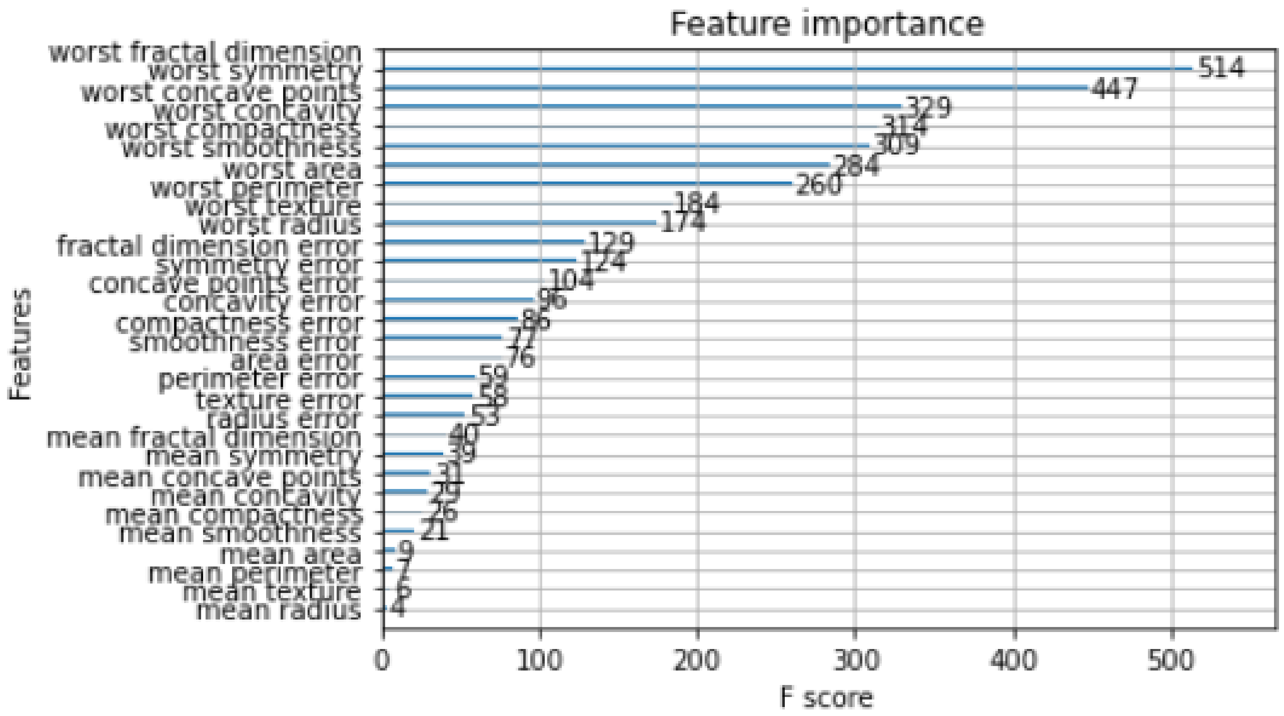
```
print("훈련 세트 정확도: {:.3f}".format(xgbb.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(xgbb.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.986
```

1. Gradient Boosting

❖ xgBoosting 실습

```python
import matplotlib.pyplot as plt
plot_importance(xgbb)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

❖ xgBoosting 실습

```
xg = XGBClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}
```

```
from sklearn.model_selection import GridSearchCV
```

```
gcv=GridSearchCV(xg, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
gcv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=XGBClassifier(), n_jobs=1,
             param_grid={'learing_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [4, 6, 8, 10, 12],
                         'n_estimators': [50, 100]},
             scoring='accuracy')
```

```
print('final params', gcv.best_params_)   # 최적의 파라미터 값 출력
print('best score', gcv.best_score_)
```

```
final params {'learing_rate': 0.01, 'max_depth': 4, 'n_estimators': 100}
best score 0.9530506155950753
```

1. Gradient Boosting

❖ xgBoosting 실습

```
cv_result_df=pd.DataFrame(gcv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)

```

```
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

	params	mean_test_score	rank_test_score
1	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.953051	1
31	{'learing_rate': 0.15, 'max_depth': 4, 'n_esti...	0.953051	1
21	{'learing_rate': 0.1, 'max_depth': 4, 'n_estim...	0.953051	1
11	{'learing_rate': 0.05, 'max_depth': 4, 'n_esti...	0.953051	1
0	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.950725	5
20	{'learing_rate': 0.1, 'max_depth': 4, 'n_estim...	0.950725	5
10	{'learing_rate': 0.05, 'max_depth': 4, 'n_esti...	0.950725	5
30	{'learing_rate': 0.15, 'max_depth': 4, 'n_esti...	0.950725	5
13	{'learing_rate': 0.05, 'max_depth': 6, 'n_esti...	0.950698	9
23	{'learing_rate': 0.1, 'max_depth': 6, 'n_estim...	0.950698	9

1. Gradient Boosting

❖ xgBoosting 실습

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn
from sklearn.datasets import load_wine

wine = load_wine()
col_names = wine.feature_names
print(len(col_names))
X_Data = pd.DataFrame(wine.data,columns = col_names )
y = pd.DataFrame(wine.target)
X_train, X_test, y_train, y_test = train_test_split(
    wine.data, wine.target, test_size=0.3,random_state=0)
```

```
13
```

```
from xgboost import plot_importance
from xgboost import XGBClassifier
import xgboost as xgb
```

```
xgb_wine = XGBClassifier(n_estimators = 400, learning_rate = 0.1 , max_depth = 3,objective = 'multi:softmax',random_state=42)
xgb_wine.fit(X_train, y_train)
```

```
XGBClassifier(n_estimators=400, objective='multi:softprob', random_state=42)
```

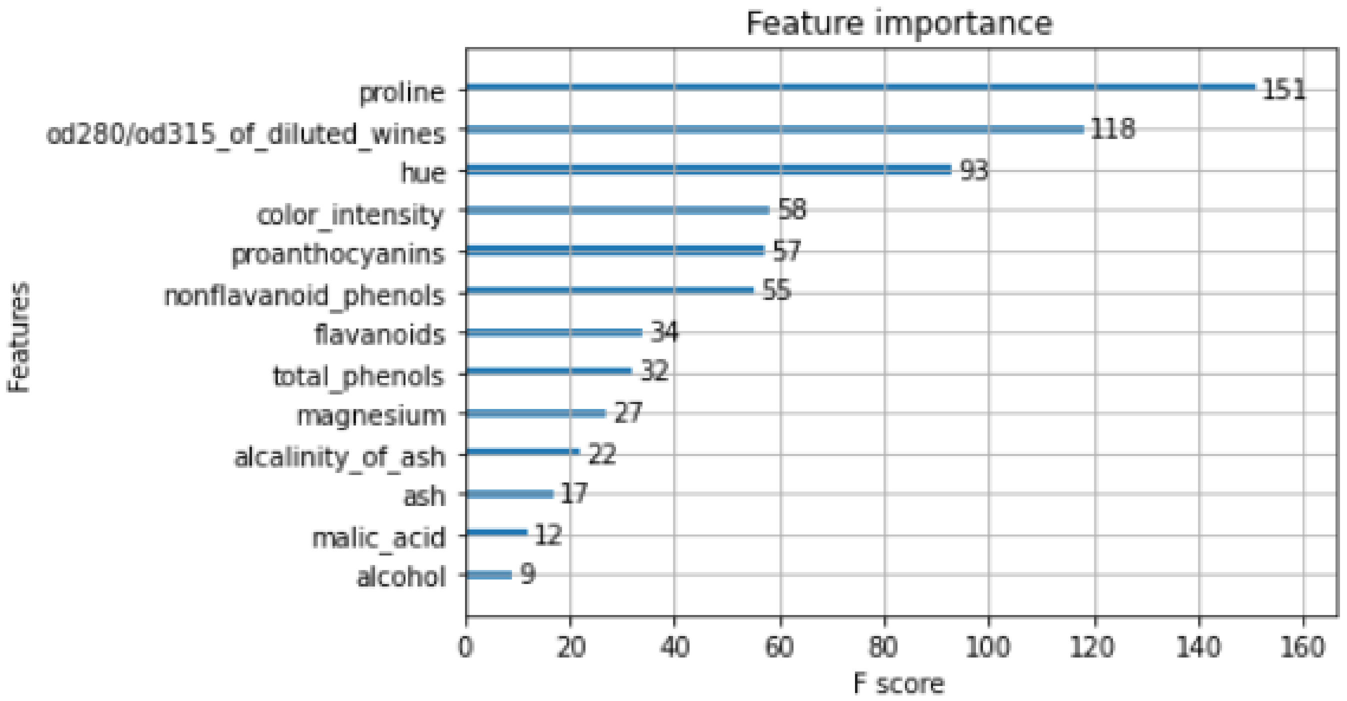
```
print("훈련 세트 정확도: {:.3f}".format(xgbb.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(xgbb.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.963
```

1. Gradient Boosting

❖ xgBoosting 실습

```python
import matplotlib.pyplot as plt
plot_importance(xgb_wine)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

❖ xgBoosting 실습

```python
def get_eval(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred,  average='macro')
    recall = recall_score(y_test, y_pred,
                                            average='macro')
    f_scro= f1_score(y_test, y_pred,
                                            average='macro')
    print('\n정확도: {:.4f}'.format(accuracy))
    print('정밀도: {:.4f}'.format(precision))
    print('재현율: {:.4f}'.format(recall))
    print('fscore: {:.4f}'.format(f_scro))

```

❖ xgBoosting 실습

```
xgb_wine = XGBClassifier(n_estimators = 400, learning_rate = 0.1 , max_depth = 3,objective = 'multi:softmax')

xgb_wine.fit(X_train, y_train, early_stopping_rounds = 100, eval_metric="mlogloss", eval_set = [(X_test, y_test)], verbose=True)
ws100_preds = xgb_wine.predict(X_test)
get_eval(y_test, ws100_preds)
```

```
[0]	validation_0-mlogloss:0.97868
Will train until validation_0-mlogloss hasn't improved in 100 rounds.
[1]	validation_0-mlogloss:0.884696
[2]	validation_0-mlogloss:0.803847
[3]	validation_0-mlogloss:0.729879
```

```
[278]	validation_0-mlogloss:0.10632
[279]	validation_0-mlogloss:0.10632
[280]	validation_0-mlogloss:0.10632
Stopping. Best iteration:
[180]	validation_0-mlogloss:0.106316
```

```
정확도: 0.9630
정밀도: 0.9595
재현율: 0.9697
fscore: 0.9632
```

1. Gradient Boosting

❖ xgBoosting 실습

```
xgbbb = XGBClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}
```

```
from sklearn.model_selection import GridSearchCV
```

```
gcv=GridSearchCV(xgbbb, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
gcv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=XGBClassifier(), n_jobs=1,
             param_grid={'learing_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [4, 6, 8, 10, 12],
                         'n_estimators': [50, 100]},
             scoring='accuracy')
```

```
print('final params', gcv.best_params_)   # 최적의 파라미터 값 출력
print('best score', gcv.best_score_)
```

```
final params {'learing_rate': 0.01, 'max_depth': 4, 'n_estimators': 50}
best score 0.9436666666666665
```

1. Gradient Boosting

❖ xgBoosting 실습

```
cv_result_df=pd.DataFrame(gcv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)

```

```
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

	params	mean_test_score	rank_test_score
0	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.943667	1
22	{'learing_rate': 0.1, 'max_depth': 6, 'n_estim...	0.943667	1
23	{'learing_rate': 0.1, 'max_depth': 6, 'n_estim...	0.943667	1
24	{'learing_rate': 0.1, 'max_depth': 8, 'n_estim...	0.943667	1
25	{'learing_rate': 0.1, 'max_depth': 8, 'n_estim...	0.943667	1
26	{'learing_rate': 0.1, 'max_depth': 10, 'n_esti...	0.943667	1
27	{'learing_rate': 0.1, 'max_depth': 10, 'n_esti...	0.943667	1
28	{'learing_rate': 0.1, 'max_depth': 12, 'n_esti...	0.943667	1
21	{'learing_rate': 0.1, 'max_depth': 4, 'n_estim...	0.943667	1
29	{'learing_rate': 0.1, 'max_depth': 12, 'n_esti...	0.943667	1

❖ xgBoosting 실습

```
from xgboost import XGBRegressor # 회귀트리 모델
from xgboost import plot_importance # 중요변수 시각화

from sklearn.datasets import load_boston # dataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
boston = load_boston()
X = boston.data
y = boston.target
col_names = boston.feature_names
```

```
print(X.shape), # (506, 13) # 13개의 칼럼
print(y.shape)
print(col_names)

```

```
(506, 13)
(506,)
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

1. Gradient Boosting

❖ xgBoosting 실습

```python
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
print(x_train.shape) # (354, 13)
print(x_test.shape) # (152, 13)
```

```
(354, 13)
(152, 13)
```

```python
model = XGBRegressor(objective ='reg:squarederror')
model.fit(x_train, y_train)

```

```
XGBRegressor(objective='reg:squarederror')
```

```python
y_predd = model.predict(x_test)
y_true = y_test
```

```python
mse = mean_squared_error(y_true, y_predd)
mse # 12.755539633559654
```

```
9.561559682544305
```

•objective기본값 : reg:squarederror
•목적함수이다. 이 함수를 통해 나온 값이 최소화되는 방향으로 학습.
•종류가 너무 다양해 자주 쓰는 것들만 설명하자면

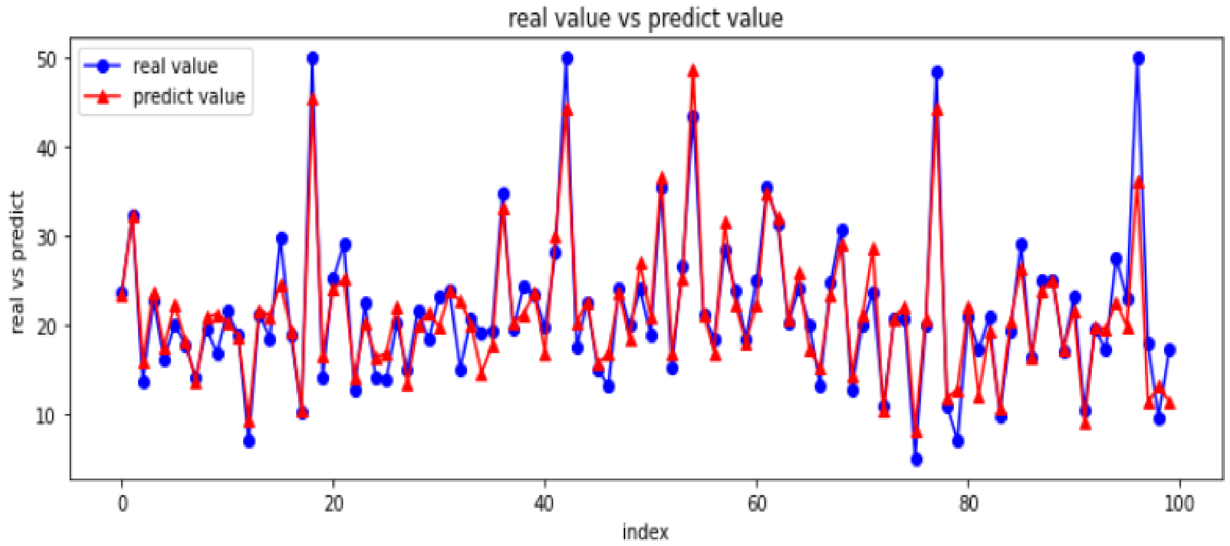
- reg:squarederror / reg:squaredlogerror : 오차 제곱 / 오차 로그 제곱
- binary:logistic : 이항 분류(binary class)에 사용.
- multi:softmax / multi:softprob : 다항 분류(multi class)에 사용.

1. Gradient Boosting

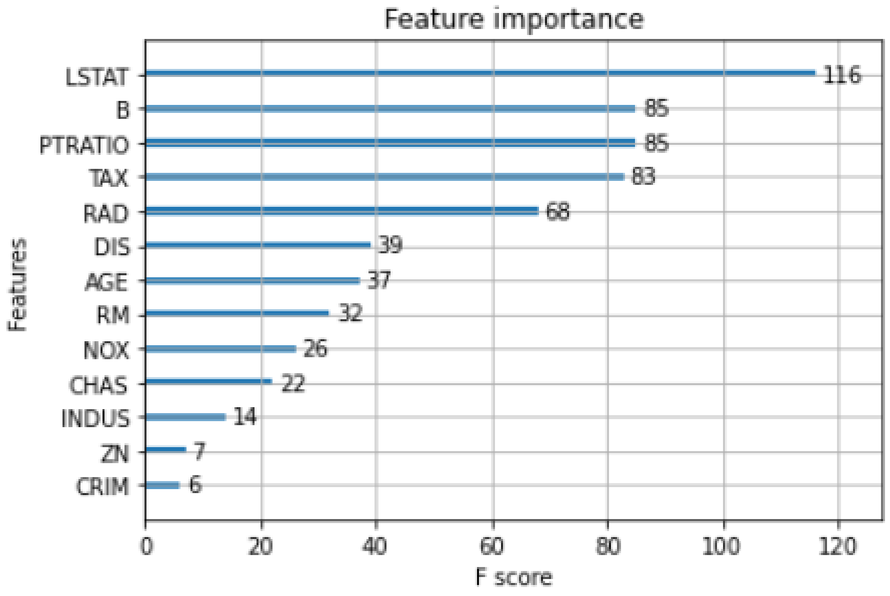
❖ xgBoosting 실습

```python
import matplotlib.pyplot as plt

fig = plt.figure( figsize = (12, 4) )
chart = fig.add_subplot(1,1,1)
chart.plot(y_true[:100], marker='o', color='blue', label='real value')
chart.plot(y_predd[:100], marker='^', color='red', label='predict value')
chart.set_title('real value vs predict value')
plt.xlabel('index')
plt.ylabel('real vs predict')
plt.legend(loc = 'best')
```

<matplotlib.legend.Legend at 0x1d09f24fb70>



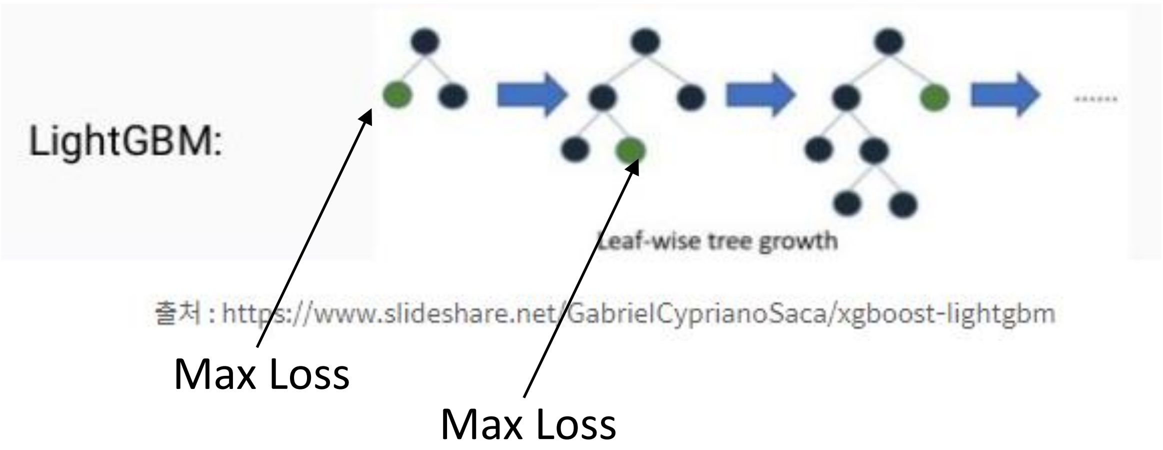

```python
import matplotlib.pyplot as plt
plot_importance(model)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

❖ LightGBM

- Xgboosting 을 해결하기 위해 나온 모델로 대용량의 데이터에서 사용 가능
- Level-wise 방법의 Xgboosting은 tree의 균형을 잡아주지만, 연산이 많고 LightGBM은 트리의 균형은 맞추지 않고 리프노드를 지속적으로 분할
- leaf-wise 알고리즘은 level-wise 알고리즘보다 더 많은 loss, 손실을 줄임 Loss 변화가 큰 노드에서 데이터를 분할 성장. 수직 성장
- 학습데이터가 많을 경우 뛰어난 성능



출처 : https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm

n_estimators : 반복하려는 트리의 개수

learning_rate : 학습률

max_depth : 트리의 최대 깊이

min_child_samples : 리프 노드가 되기 위한 최소한의 샘플 데이터 수

num_leaves : 하나의 트리가 가질 수 있는 최대 리프 개수

feature_fraction : 트리를 학습할 때마다 선택하는 feature의 비율

reg_lambda : L2 regularization

reg_alpha : L1 regularization

1. Gradient Boosting

❖ LightGBM

- **장점**
  - XGBoost 대비 더 빠른 학습과 예측 수행 시간
  - 더 작은 메모리 사용량
  - 카테고리형 피처의 자동 변환과 최적 분할

- **단점**
  - 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽고 (공식 문서상 대략 10,000건 이하의 데이터 세트)

1. Gradient Boosting

❖ **LightGBM 설치**

- **#콘다환경에서 설치**
- **conda install -c conda-forge lightgbm**
- **pip install lightgbm**

1. Gradient Boosting

❖ LightGBM 실습

- from lightgbm import LGBMClassifier , plot_importance
- model_lgb=LGBMClassifier()
- model_lgb.fit(x_train, y_train)
- model_lgb. predict(x_train, y_train)

1. Gradient Boosting

❖ LightGBM 실습

파라미터 명	설명
objective	- ‘reg:linear’ : 회귀 - binary:logistic : 이진분류 - multi:softmax : 다중분류, 클래스 반환 - multi:softprob : 다중분류, 확률반환
eval_metric	- 검증에 사용되는 함수정의 - 회귀 분석인 경우 'rmse'를, 클래스 분류 문제인 경우 'error' --- - rmse : Root Mean Squared Error - mae : mean absolute error - logloss : Negative log-likelihood - error : binary classification error rate - merror : multiclass classification error rate - mlogloss: Multiclass logloss - auc: Area Under Curve
early_stopping_rounds	**eval_set** : 성능평가를 위한 평가용 데이터 세트를 설정 **eval_metric** : 평가 세트에 적용할 성능 평가 방법 (반복마다 eval_set으로 지정된 데이터 세트에서 eval_metric의 지정된 평가 지표로 예측 오류를 측정)

1. Gradient Boosting

❖ LightGBM

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn

cancer = load_breast_cancer()
col_names = cancer.feature_names
print(len(col_names))
X_Data = pd.DataFrame(cancer.data,columns = col_names )
y = pd.DataFrame(cancer.target)

X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

1. Gradient Boosting

❖ LightGBM

```
from lightgbm import LGBMClassifier , plot_importance
```

```
lgb = LGBMClassifier(n_estimators=500,random_state=42)
lgb.fit(X_train, y_train)
```

```
LGBMClassifier(n_estimators=500, random_state=42)
```

```
print("훈련 세트 정확도: {:.3f}".format(lgb.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(lgb.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.979
```

1. Gradient Boosting

❖ LightGBM

```python
evals= [(X_test, y_test)]
```

```python
lgb.fit(X_train, y_train, early_stopping_rounds=100, eval_metric= "logloss", eval_set=evals, verbose=True)
```

```
[485]	valid_0's binary_logloss: 0.0353702
[486]	valid_0's binary_logloss: 0.0353702
[487]	valid_0's binary_logloss: 0.0353702
[488]	valid_0's binary_logloss: 0.0353702
[489]	valid_0's binary_logloss: 0.0353702
[490]	valid_0's binary_logloss: 0.0353702
[491]	valid_0's binary_logloss: 0.0353702
[492]	valid_0's binary_logloss: 0.0353702
[493]	valid_0's binary_logloss: 0.0353702
[494]	valid_0's binary_logloss: 0.0353702
[495]	valid_0's binary_logloss: 0.0353702
[496]	valid_0's binary_logloss: 0.0353702
[497]	valid_0's binary_logloss: 0.0353702
[498]	valid_0's binary_logloss: 0.0353702
[499]	valid_0's binary_logloss: 0.0353702
[500]	valid_0's binary_logloss: 0.0353702
Did not meet early stopping. Best iteration is:
[427]	valid_0's binary_logloss: 0.0350867

LGBMClassifier(n_estimators=500, random_state=42)
```

early_stopping_rounds 파라미터 : 조기 중단을 위한 라운드를 설정합니다.
조기 중단 기능 수행을 위해서는 반드시 eval_set과 eval_metric이 함께 설정되어야 합니다.

- **eval_set** : 성능평가를 위한 평가용 데이터 세트를 설정
- **eval_metric** : 평가 세트에 적용할 성능 평가 방법
  (반복마다 eval_set으로 지정된 데이터 세트에서 eval_metric의 지정된 평가 지표로 예측 오류를 측정)

1. Gradient Boosting

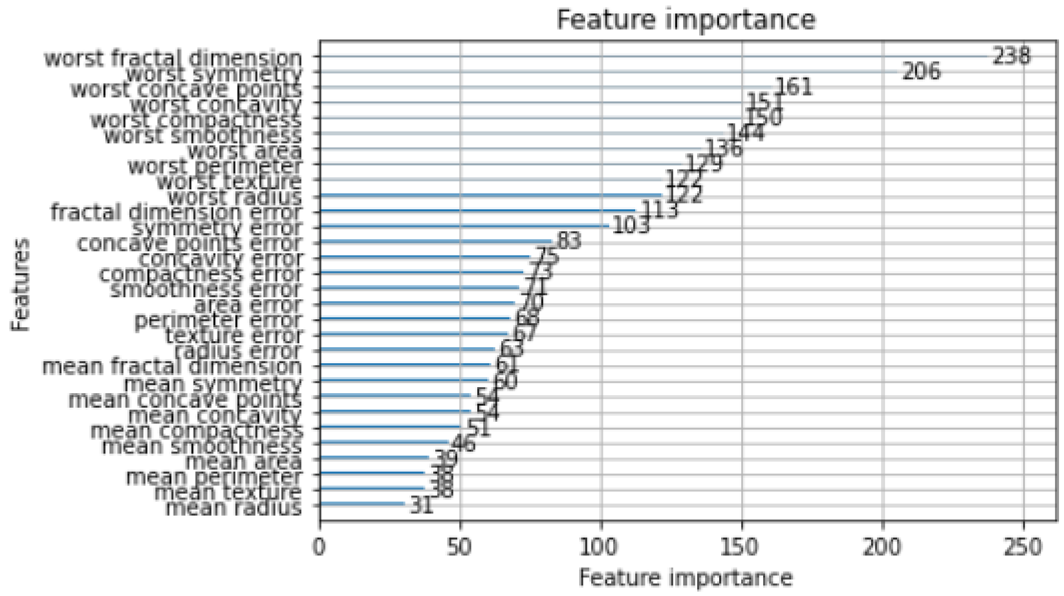
❖ LightGBM

```python
evals= [(X_test, y_test)]
```

```python
lgb.fit(X_train, y_train, early_stopping_rounds=100, eval_metric= "logloss", eval_set=evals, verbose=True)
```

```
[486]	valid_0's binary_logloss: 0.0353702
[487]	valid_0's binary_logloss: 0.0353702
[488]	valid_0's binary_logloss: 0.0353702
[489]	valid_0's binary_logloss: 0.0353702
[490]	valid_0's binary_logloss: 0.0353702
[491]	valid_0's binary_logloss: 0.0353702
[492]	valid_0's binary_logloss: 0.0353702
[493]	valid_0's binary_logloss: 0.0353702
[494]	valid_0's binary_logloss: 0.0353702
[495]	valid_0's binary_logloss: 0.0353702
[496]	valid_0's binary_logloss: 0.0353702
[497]	valid_0's binary_logloss: 0.0353702
[498]	valid_0's binary_logloss: 0.0353702
[499]	valid_0's binary_logloss: 0.0353702
[500]	valid_0's binary_logloss: 0.0353702
Did not meet early stopping. Best iteration is:
[427]	valid_0's binary_logloss: 0.0350867

LGBMClassifier(n_estimators=500, random_state=42)
```

```python
import matplotlib.pyplot as plt
plot_importance(lgb)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

❖ LightGBM

```
lb = LGBMClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}
```

```
lgb_cv=GridSearchCV(lb, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
lgb_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=LGBMClassifier(), n_jobs=1,
             param_grid={'learing_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [4, 6, 8, 10, 12],
                         'n_estimators': [50, 100]},
             scoring='accuracy')
```

```
print('final params', lgb_cv.best_params_)   # 최적의 파라미터 값 출력
print('best score', lgb_cv.best_score_)
```

```
final params {'learing_rate': 0.01, 'max_depth': 10, 'n_estimators': 100}
best score 0.9741997264021889
```

1. Gradient Boosting

❖ LightGBM

```
cv_result_df=pd.DataFrame(lgb_cv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)
```

```
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

	params	mean_test_score	rank_test_score
19	{'learing_rate': 0.05, 'max_depth': 12, 'n_est...	0.974200	1
37	{'learing_rate': 0.15, 'max_depth': 10, 'n_est...	0.974200	1
29	{'learing_rate': 0.1, 'max_depth': 12, 'n_esti...	0.974200	1
27	{'learing_rate': 0.1, 'max_depth': 10, 'n_esti...	0.974200	1
17	{'learing_rate': 0.05, 'max_depth': 10, 'n_est...	0.974200	1
9	{'learing_rate': 0.01, 'max_depth': 12, 'n_est...	0.974200	1
7	{'learing_rate': 0.01, 'max_depth': 10, 'n_est...	0.974200	1
39	{'learing_rate': 0.15, 'max_depth': 12, 'n_est...	0.974200	1
1	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.974118	9
11	{'learing_rate': 0.05, 'max_depth': 4, 'n_esti...	0.974118	9

1. Gradient Boosting

❖ LightGBM

```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import mglearn
from sklearn.datasets import load_wine

wine = load_wine()
col_names = wine.feature_names
print(len(col_names))
X_Data = pd.DataFrame(wine.data,columns = col_names )
y = pd.DataFrame(wine.target)
X_train, X_test, y_train, y_test = train_test_split(
    wine.data, wine.target, test_size=0.3,random_state=0)
```

```
from lightgbm import LGBMClassifier , plot_importance
```

```
lgbb = LGBMClassifier(n_estimators=500,random_state=42)
lgbb.fit(X_train, y_train)
```

```
LGBMClassifier(n_estimators=500, random_state=42)
```

```
print("훈련 세트 정확도: {:.3f}".format(lgbb.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(lgbb.score(X_test, y_test)))
```

```
훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.963
```

1. Gradient Boosting

❖ LightGBM

```
lgb = LGBMClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}
```

```
lgb_cv=GridSearchCV(lgb, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
lgb_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=LGBMClassifier(), n_jobs=1,
             param_grid={'learing_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [4, 6, 8, 10, 12],
                         'n_estimators': [50, 100]},
             scoring='accuracy')
```

```
print('final params', lgb_cv.best_params_)   # 최적의 파라미터 값 출력
print('best score', lgb_cv.best_score_)
```

```
final params {'learing_rate': 0.01, 'max_depth': 4, 'n_estimators': 100}
best score 0.9596666666666666
```

1. Gradient Boosting

❖ LightGBM

```
cv_result_df=pd.DataFrame(lgb_cv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)
```

```
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10
```

	params	mean_test_score	rank_test_score
19	{'learing_rate': 0.05, 'max_depth': 12, 'n_est...	0.959667	1
37	{'learing_rate': 0.15, 'max_depth': 10, 'n_est...	0.959667	1
35	{'learing_rate': 0.15, 'max_depth': 8, 'n_esti...	0.959667	1
33	{'learing_rate': 0.15, 'max_depth': 6, 'n_esti...	0.959667	1
31	{'learing_rate': 0.15, 'max_depth': 4, 'n_esti...	0.959667	1
29	{'learing_rate': 0.1, 'max_depth': 12, 'n_esti...	0.959667	1
27	{'learing_rate': 0.1, 'max_depth': 10, 'n_esti...	0.959667	1

1. Gradient Boosting

❖ LightGBM

```
evals= [(X_test, y_test)]
```

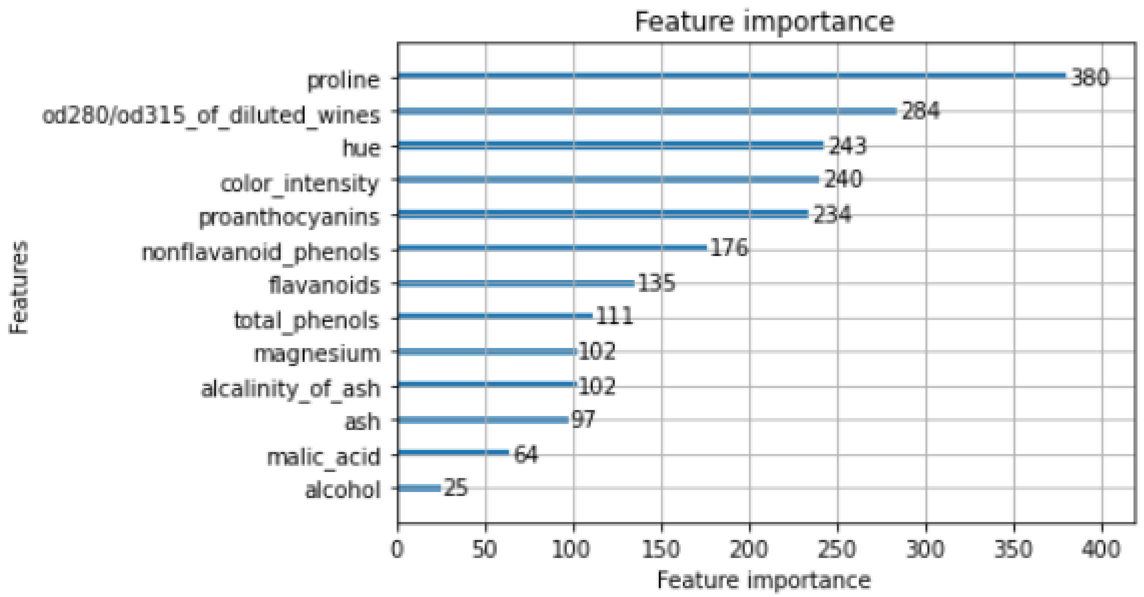
```
lgb.fit(X_train, y_train, early_stopping_rounds=100, eval_metric= "logloss", eval_set=evals, verbose=True)
```

```
[85]	valid_0's multi_logloss: 0.123742
[86]	valid_0's multi_logloss: 0.11971
[87]	valid_0's multi_logloss: 0.121348
[88]	valid_0's multi_logloss: 0.12351
[89]	valid_0's multi_logloss: 0.123325
[90]	valid_0's multi_logloss: 0.124799
[91]	valid_0's multi_logloss: 0.125354
[92]	valid_0's multi_logloss: 0.127083
[93]	valid_0's multi_logloss: 0.124206
[94]	valid_0's multi_logloss: 0.125782
[95]	valid_0's multi_logloss: 0.127472
[96]	valid_0's multi_logloss: 0.126509
[97]	valid_0's multi_logloss: 0.12761
[98]	valid_0's multi_logloss: 0.129902
[99]	valid_0's multi_logloss: 0.132568
[100]	valid_0's multi_logloss: 0.131501
Did not meet early stopping. Best iteration is:
[65]	valid_0's multi_logloss: 0.108457

LGBMClassifier()
```

early_stopping_rounds 파라미터 : 조기 중단을 위한 라운드를 설정합니다.
조기 중단 기능 수행을 위해서는 반드시 eval_set과 eval_metric이 함께 설정되어야 합니다.

- **eval_set** : 성능평가를 위한 평가용 데이터 세트를 설정
- **eval_metric** : 평가 세트에 적용할 성능 평가 방법
  (반복마다 eval_set으로 지정된 데이터 세트에서 eval_metric의 지정된 평가 지표로 예측 오류를 측정)

```
import matplotlib.pyplot as plt
plot_importance(lgbb)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

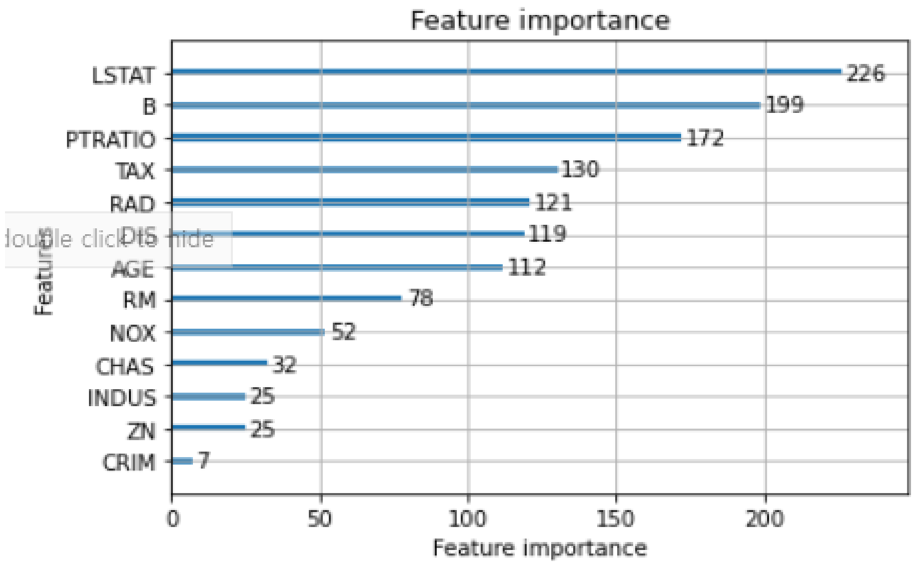
❖ LightGBM

```python
import lightgbm as lgb
```

```python
model_lgb=lgb.LGBMRegressor()
model_lgb.fit(x_train, y_train)
model_lgb
```

```
LGBMRegressor()
```

```python
import matplotlib.pyplot as plt
lgb.plot_importance(model_lgb)
plt.yticks(range(len(col_names)), col_names)
plt.show()
```



1. Gradient Boosting

❖ LightGBM

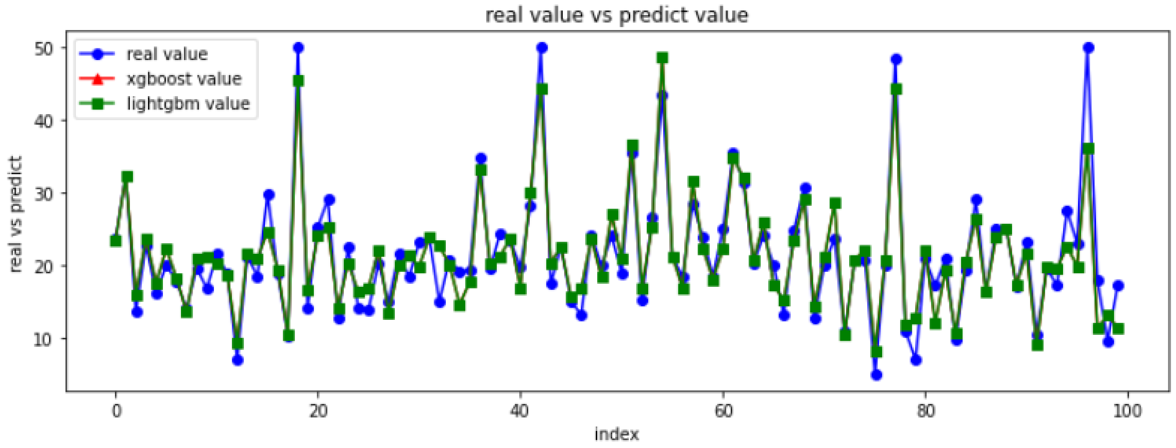
```
y_pred = model.predict(x_test)
y_true = y_test
```

```
mse = mean_squared_error(y_true, y_pred)
mse
```

9.561559682544305

```
import matplotlib.pyplot as plt

# y_true.shape # (5160,) : 5160개의 데이터 => 많으니까 100개만 출력 시도

fig = plt.figure( figsize = (12, 4) )
chart  = fig.add_subplot(1,1,1)
chart.plot(y_true[:100], marker='o', color='blue', label='real value')
chart.plot(y_pred[:100], marker='^', color='red', label='xgboost value')
chart.plot(y_predd[:100], marker='s', color='green', label='lightgbm value')
chart.set_title('real value vs predict value')
plt.xlabel('index')
plt.ylabel('real vs predict')
plt.legend(loc = 'best')

# 위에 블럭 실행
```

```
<matplotlib.legend.Legend at 0x1d09e639588>
```



1. Gradient Boosting

❖ LightGBM

```
model = XGBRegressor(objective ='reg:squarederror')
model.fit(x_train, y_train)

```

```
XGBRegressor(objective='reg:squarederror')
```

```
y_predd = model.predict(x_test)
y_true = y_test
```

```
mse = mean_squared_error(y_true, y_predd)
mse
```

```
9.561559682544305
```

```
import lightgbm as lgb
```

```
model_lgb=lgb.LGBMRegressor()
model_lgb.fit(x_train, y_train)
model_lgb
```

```
LGBMRegressor()
```

```
y_pred = model.predict(x_test)
y_true = y_test
```

```
mse = mean_squared_error(y_true, y_pred)
mse
```

```
9.561559682544305
```

1. Gradient Boosting

❖ Titanic

pclass : 1, 2, 3등석 정보를 각각 1, 2, 3으로 저장

survived 생존 여부. survived(생존), dead(사망)

name 이름

sex 성별. female(여성), male(남성)

age 나이

sibsp 함께 탑승한 형제 또는 배우자의 수

parch 함께 탑승한 부모 또는 자녀의 수

ticket 티켓 번호

fare 티켓 요금

cabin 선실 번호

embarked 탑승한 곳. C(Cherbourg), Q(Queenstown), S(Southampton)

1. Gradient Boosting

❖ Titanic

```
data=pd.read_csv("D:/big_data/titanic.csv")
#상위 데이터 보여주기
display(data.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

1. Gradient Boosting

❖ **Titanic**

```
print(data.shape)
data.columns.values

data.info()
```

```
(891, 12)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

1. Gradient Boosting

❖ Titanic

```
y_data = data["Survived"]
data.drop(labels="Survived", axis=1, inplace=True)
display(data.head())
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

1. Gradient Boosting

❖ Titanic

```
drop_columns = ["Name", "Age", "SibSp", "Ticket", "Cabin", "Parch", "Embarked"]
data.drop(labels=drop_columns, axis=1, inplace=True)
display(data.head())
```

	PassengerId	Pclass	Sex	Fare
0	1	3	male	7.2500
1	2	1	female	71.2833
2	3	3	female	7.9250
3	4	1	female	53.1000
4	5	3	male	8.0500

1. Gradient Boosting

❖ Titanic

```
data = pd.get_dummies(data, columns=["Sex"])
data.fillna(value=0.0, inplace=True)
display(data.head())
```

	PassengerId	Pclass	Fare	Sex_female	Sex_male
0	1	3	7.2500	0	1
1	2	1	71.2833	1	0
2	3	3	7.9250	1	0
3	4	1	53.1000	1	0
4	5	3	8.0500	0	1

1. Gradient Boosting

❖ Titanic

```
state = 42
test_size = 0.30

X_train, X_val, y_train, y_val = train_test_split(data, y_data,
    test_size=test_size, random_state=state)
```

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_features=2, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, y_val)))
```

1. Gradient Boosting

❖ Titanic

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_features=2, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, y_val)))
```

```
Learning rate:  0.05
Accuracy score (training): 0.788
Accuracy score (validation): 0.772
Learning rate:  0.075
Accuracy score (training): 0.791
Accuracy score (validation): 0.772
Learning rate:  0.1
Accuracy score (training): 0.804
Accuracy score (validation): 0.769
Learning rate:  0.25
Accuracy score (training): 0.823
Accuracy score (validation): 0.757
Learning rate:  0.5
Accuracy score (training): 0.844
Accuracy score (validation): 0.799
Learning rate:  0.75
Accuracy score (training): 0.862
Accuracy score (validation): 0.802
Learning rate:  1
Accuracy score (training): 0.867
Accuracy score (validation): 0.799
```

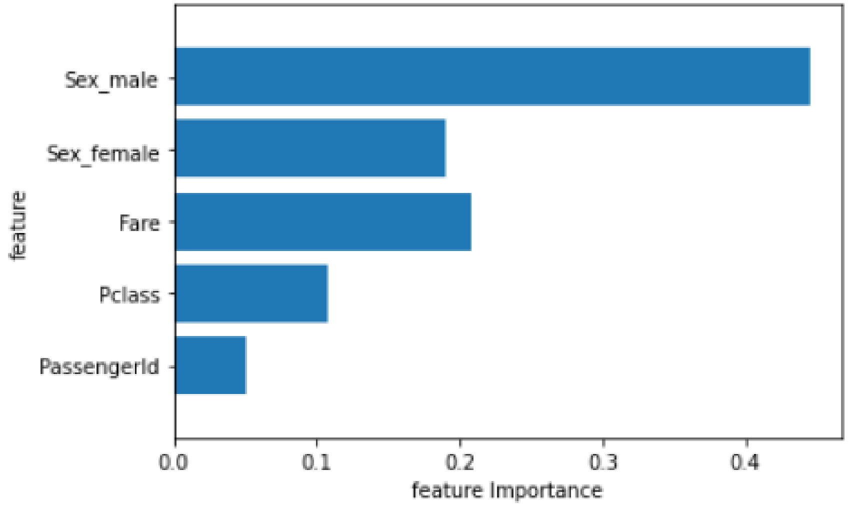
1. Gradient Boosting

❖ Titanic

```
gb_clf2 = GradientBoostingClassifier(n_estimators=20, learning_rate=0.5, max_features=2, max_depth=2, random_state=0)
gb_clf2.fit(X_train, y_train)
predictions = gb_clf2.predict(X_val)

print("Classification Report")
print(classification_report(y_val, predictions))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.93      0.84       157
           1       0.86      0.61      0.72       111

    accuracy                           0.80       268
   macro avg       0.82      0.77      0.78       268
weighted avg       0.81      0.80      0.79       268
```

```
n_features = len(data.columns)

plt.barh(range(n_features), gb_clf2.feature_importances_, align='center')
plt.yticks(np.arange(n_features), data.columns)
plt.xlabel("feature Importance")
plt.ylabel("feature")
plt.ylim(-1, n_features)
plt.show()
```



1. Gradient Boosting

❖ Titanic

```
from xgboost import plot_importance
from xgboost import XGBClassifier
import xgboost as xgb
```

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = XGBClassifier(n_estimators=20, learning_rate=learning_rate, max_features=2, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, y_val)))
```

```
Learning rate:  0.05
Accuracy score (training): 0.793
Accuracy score (validation): 0.772
Learning rate:  0.075
Accuracy score (training): 0.793
Accuracy score (validation): 0.772
Learning rate:  0.1
Accuracy score (training): 0.793
Accuracy score (validation): 0.772
Learning rate:  0.25
Accuracy score (training): 0.836
Accuracy score (validation): 0.806
Learning rate:  0.5
Accuracy score (training): 0.851
Accuracy score (validation): 0.817
Learning rate:  0.75
Accuracy score (training): 0.859
Accuracy score (validation): 0.810
Learning rate:  1
Accuracy score (training): 0.873
Accuracy score (validation): 0.784
```

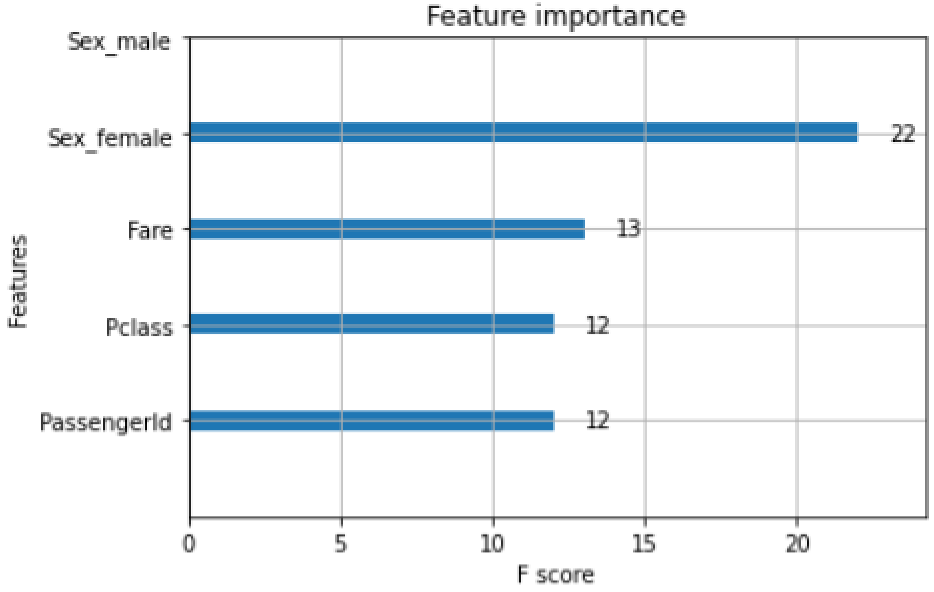
1. Gradient Boosting

❖ Titanic

```python
xg_clf2 = XGBClassifier(n_estimators=20, learning_rate=0.25, max_features=2, max_depth=2, random_state=0)
xg_clf2.fit(X_train, y_train)
predictions = xg_clf2.predict(X_val)

print("Confusion Matrix:")
print(confusion_matrix(y_val, predictions))

print("Classification Report")
print(classification_report(y_val, predictions))
```

```
Confusion Matrix:
[[146  11]
 [ 41  70]]
Classification Report
              precision    recall  f1-score   support

           0       0.78      0.93      0.85       157
           1       0.86      0.63      0.73       111

    accuracy                           0.81       268
   macro avg       0.82      0.78      0.79       268
weighted avg       0.82      0.81      0.80       268
```

```python
import matplotlib.pyplot as plt
plot_importance(xg_clf2)
plt.yticks(range(n_features),  data.columns)
plt.show()
```



1. Gradient Boosting

❖ Titanic

```
from lightgbm import LGBMClassifier , plot_importance
```

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = LGBMClassifier(n_estimators=20, learning_rate=learning_rate, max_features=2 , random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, y_val)))
```

```
Learning rate:  0.05
Accuracy score (training): 0.830
Accuracy score (validation): 0.784
Learning rate:  0.075
Accuracy score (training): 0.839
Accuracy score (validation): 0.787
Learning rate:  0.1
Accuracy score (training): 0.844
Accuracy score (validation): 0.791
Learning rate:  0.25
Accuracy score (training): 0.892
Accuracy score (validation): 0.810
Learning rate:  0.5
Accuracy score (training): 0.950
Accuracy score (validation): 0.791
Learning rate:  0.75
Accuracy score (training): 0.978
Accuracy score (validation): 0.787
Learning rate:  1
Accuracy score (training): 0.984
Accuracy score (validation): 0.780
```

1. Gradient Boosting

❖ Titanic

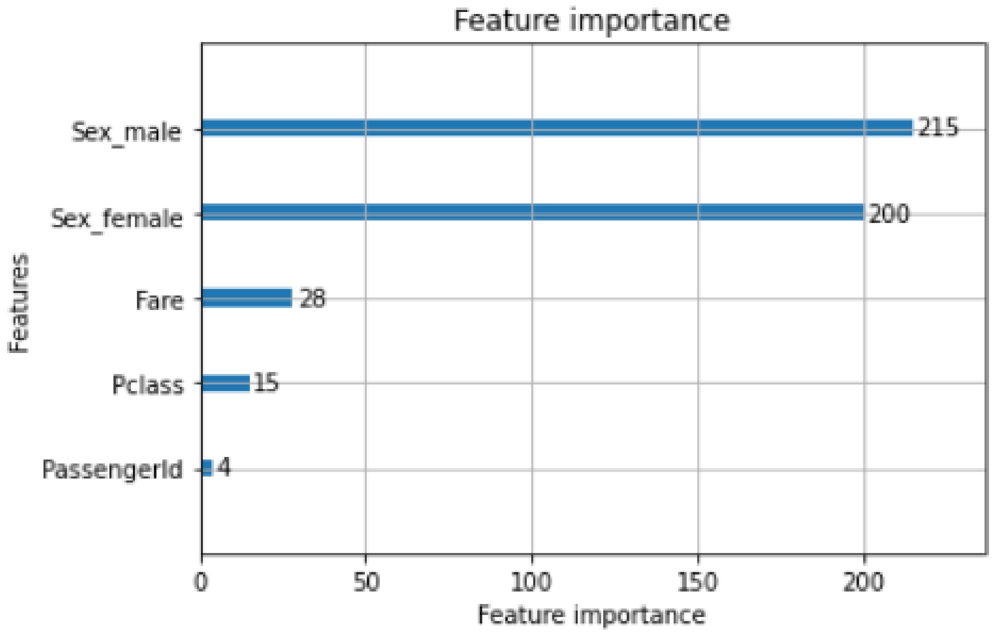
```python
LG_clf2 = LGBMClassifier(n_estimators=20, learning_rate=0.25, max_features=2, random_state=0)
LG_clf2.fit(X_train, y_train)
predictions = LG_clf2.predict(X_val)

print("Confusion Matrix:")
print(confusion_matrix(y_val, predictions))

print("Classification Report")
print(classification_report(y_val, predictions))
```

```
Confusion Matrix:
[[142  15]
 [ 36  75]]
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       157
           1       0.83      0.68      0.75       111

    accuracy                           0.81       268
   macro avg       0.82      0.79      0.80       268
weighted avg       0.81      0.81      0.81       268
```

1. Gradient Boosting

❖ Titanic

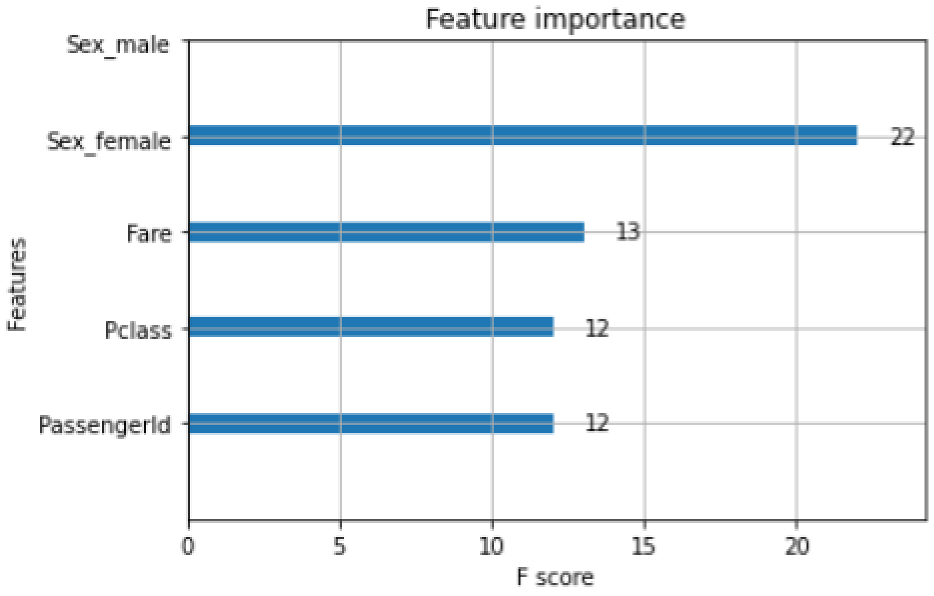
```python
import matplotlib.pyplot as plt
plot_importance(LG_clf2)
plt.yticks(range(n_features),  data.columns)
plt.show()
```



1. Gradient Boosting

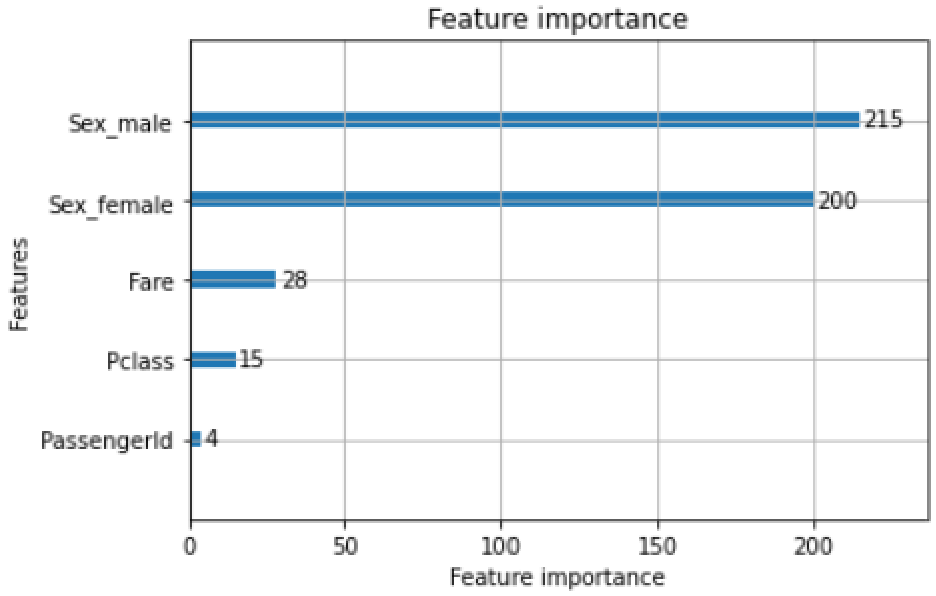
❖ Titanic

```python
import matplotlib.pyplot as plt
plot_importance(xg_clf2)
plt.yticks(range(n_features), data.columns)
plt.show()
```



81%의 예측력

```python
import matplotlib.pyplot as plt
plot_importance(LG_clf2)
plt.yticks(range(n_features), data.columns)
plt.show()
```



81%의 예측력

1. Gradient Boosting

❖ Titanic

```python
xgg = XGBClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}


xgg_cv=GridSearchCV(xgg, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
xgg_cv.fit(X_train, y_train)

print('final params', xgg_cv.best_params_)   # 최적의 파라미터 값 출력
print('best score', xgg_cv.best_score_)

cv_result_df=pd.DataFrame(xgg_cv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

```
final params {'learing_rate': 0.01, 'max_depth': 4, 'n_estimators': 50}
best score 0.8042064516129033
```

	params	mean_test_score	rank_test_score
0	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.804206	1
10	{'learing_rate': 0.05, 'max_depth': 4, 'n_esti...	0.804206	1
30	{'learing_rate': 0.15, 'max_depth': 4, 'n_esti...	0.804206	1
20	{'learing_rate': 0.1, 'max_depth': 4, 'n_estim...	0.804206	1
31	{'learing_rate': 0.15, 'max_depth': 4, 'n_esti...	0.797781	5
21	{'learing_rate': 0.1, 'max_depth': 4, 'n_estim...	0.797781	5
1	{'learing_rate': 0.01, 'max_depth': 4, 'n_esti...	0.797781	5

```python
lb = LGBMClassifier()

param_grid={
    'max_depth':[4,6,8,10,12],
    'n_estimators':[50,100],
    'learing_rate':[0.01,0.05,0.1,0.15]}


lgb_cv=GridSearchCV(lb, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=1)
lgb_cv.fit(X_train, y_train)

print('final params', lgb_cv.best_params_)   # 최적의 파라미터 값 출력
print('best score', lgb_cv.best_score_)

cv_result_df=pd.DataFrame(lgb_cv.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

```
final params {'learing_rate': 0.01, 'max_depth': 10, 'n_estimators': 50}
best score 0.7913161290322581
```

	params	mean_test_score	rank_test_score
36	{'learing_rate': 0.15, 'max_depth': 10, 'n_est...	0.791316	1
26	{'learing_rate': 0.1, 'max_depth': 10, 'n_esti...	0.791316	1
6	{'learing_rate': 0.01, 'max_depth': 10, 'n_est...	0.791316	1
16	{'learing_rate': 0.05, 'max_depth': 10, 'n_est...	0.791316	1
33	{'learing_rate': 0.15, 'max_depth': 6, 'n_esti...	0.789729	5
3	{'learing_rate': 0.01, 'max_depth': 6, 'n_esti...	0.789729	5
23	{'learing_rate': 0.1, 'max_depth': 6, 'n_estim...	0.789729	5
13	{'learing_rate': 0.05, 'max_depth': 6, 'n_esti...	0.789729	5
19	{'learing_rate': 0.05, 'max_depth': 12, 'n_est...	0.788168	9

1. Gradient Boosting

❖ reference

- 모든 강의자료는 고려대 강필성 교수님 강의와 김성범 교수님 강의를 참고했음
- ratsgo’s blog ,https://ratsgo.github.io/
- 안드레아스 뮐러, 세라 가이도 지음, 박해선 옮김, "파이썬 라이브러리를 활용한 머신러닝", 한빛미디어(2017)
- https://injo.tistory.com/
- https://lsjsj92.tistory.com/
- https://scikit-learn.org/stable/index.html

감사합니다