4.5 – 4.7

GBM, XGBoost, LightGBM

CUAI 6조
C6AI : 김소영 김유현 박민형 장재용

Boosting

부스팅 알고리즘 : 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치를 부여를 통해 오류를 개선해 나가면서 학습하는 방식

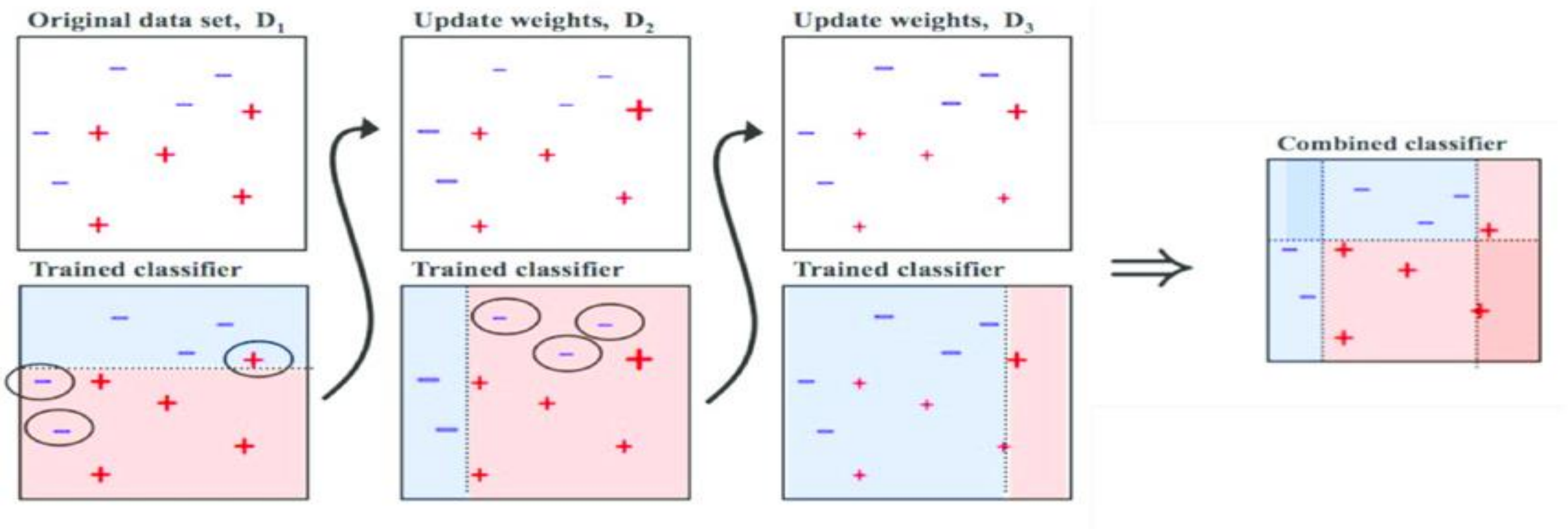
AdaBoost (Adaptive boosting)

Gradient Boosting Machine

AdaBoost

AdaBoost(Adaptive boosting) :

오류 데이터에 가중치를 부여하면서 부스팅을 수행



1. GBM

Gradient Boosting Machine

GBM

Gradient Boost Machine(GBM) : AdaBoost와 유사하나 가중치 업데이트를 경사 하강법(Gradient Descent)를 이용

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i-1}^{n} L(y_i, \gamma).$$

→ Baseline model

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

→ negative gradient = pseudo-residual

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

이 과정은 생략되기도 함

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

매우 작은 값

3. Output $F_M(x)$.

출처 : https://3months.tistory.com/368)

GBM - Parameters

parameters	Detail
Loss	경사 하강법에서 사용할 비용 함수 지정 기본값 : deviance
Learning_rate	Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수 기본값:0.1, 설정 가능 범위 : 0~1
n_estimators	Weak learner의 개수 기본값:100
subsample	Weak learner가 학습에 사용하는 데이터의 샘플링 비율 기본값:1(전체 데이터 사용을 의미)

Parameters - Learning_rate

learning_rate ▼	
장점 : 업데이트되는 값이 작아져서 최소 오류 값을 찾아 예측 성능이 높아질 가능성 높음	단점 : 시간이 오래 걸리고 반복 횟수가 부족하면 모든 weak learner의 반복이 완료돼도 최소 오류 값을 찾지 못할 수 있음

learning_rate ▲	
장점 : 빠른 수행이 가능	단점 : 최소 오류 값을 찾지 못하고 그냥 지나쳐 버려 예측 성능이 떨어질 가능성이 높음

Parameters – n_estimators

- n_estimators

weak learner 개수를 의미하며, 높을수록 성능 △, 수행시간 ▲

- learning_rate와 n_estimator의 관계

learning rate이 작으면 성능은 일정 수준까지 향상될 확률은 높아지는데 이 때 n_estimator을 줄여줌으로써 학습시간이 너무 길어지는 것을 방지 할 수 있다

GBM – Code(1)

```
from sklearn.ensemble import GradientBoostingClassifier
import time
import warnings
warnings.filterwarnings('ignore')

X_train, X_test, y_train, y_test = get_human_dataset()
```

```
from sklearn.model_selection import GridSearchCV
gb_clf = GradientBoostingClassifier(random_state=0)
params={'n_estimators':[100,500],
        'learning_rate':[0.05,0.1]}
grid_cv=GridSearchCV(gb_clf, param_grid=params, cv=2, verbose=1)
start_time=time.time()
grid_cv.fit(X_train,y_train)
print("GridSearch 소요시간: {0:0.2f}".format(time.time()-start_time))
print('최적 하이퍼 파라미터:₩n',grid_cv.best_params)
print('최고 예측 정확도: {0:4f}'.format(grid_cv.best_score_))
gb_pred = grid_cv.best_estimator_.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)
print("GBM 정확도: {0:.0.4f}".format(gb_accuracy))
```

GBM

GridSearchCV

GBM – Code(2)

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed: 14.4min finished

GridSearch 소요시간: 1314.77
최적 하이퍼 파라미터:
 {'learning_rate': 0.1, 'n_estimators': 500}
최고 예측 정확도: 0.904108
GBM 정확도: 0.9454
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   8 out of   8 | elapsed:  6.6min finished

GridSearch 소요시간: 845.40
최적 하이퍼 파라미터:
 {'learning_rate': 0.1, 'n_estimators': 500}
최고 예측 정확도: 0.904108
GBM 정확도: 0.9454
```

줄어든 수행시간

동일 성능

2. XGBoost

eXtra Gradient Boost

eXtra Gredient Boost 주요 장점

빠른 수행 시간

GBM 대비 빠른 수행시간

과적합 규제

자체 과적합 규제 기능 보유

결손값 자체 처리

결손값 자체 처리 기능 보유

자체 내장된 교차 검증

조기 중단 기능으로

eXtra Gredient Boost 모듈종류

파이썬 래퍼

XGBoost 모듈

사이킷런 래퍼

XGBoost 모듈

eXtra Gredient Boost 모듈종류

파이썬 래퍼 XGBoost 모듈

```
dtrain=xgb.DMatrix(data=X_train, label=y_train)
dtest=xgb.DMatrix(data=X_test, label=y_test)
params={'max_depth':3, 'eta':0.1, 'objective':'binary:logistic',
        'eval_metric':'logloss','early_stoppings':100}
num_rounds=400
wlist=[(dtrain,'train'),(dtest,'eval')]
xgb_model=xgb.train(params=params, dtrain=dtrain, num_boost_round=num_rounds,evals=wlist)
pred_probs=xgb_model.predict(dtest)
```

사이킷런 래퍼 XGBoost 모듈

```
from xgboost import XGBClassifier
xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
evals=[(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric='logloss', eval_set=evals, verbose=True)
w_preds=xgb_wrapper.predict(X_test)
```

eXtra Gredient Boost 파라미터

```
xgb_wrapper = XGBClassifier(booster='gbtree',
                            nthread=4,
                            silent=1,
                            n_estimators=400,
                            learning_rate=0.1,
                            max_depth=3,
                            subsample=1,
                            min_child_weight=3,
                            gamma=0,
                            colsample_bytree=1,
                            colsample_bylevel=1,
                            reg_lambda=1,
                            reg_alpha=0,
                            objective='binary:logistic')
evals=[(X_test, y_test)]

xgb_wrapper.fit(X_train, y_train,
                early_stopping_rounds=100,
                eval_metric='logloss',
                eval_set=evals)
```

eXtra Gredient Boost 파라미터

일반 파라미터

NO	이름	내용	디폴트 값
01	**booster**	사용할 부스터의 종류를 선택하는 것. gbtree(tree based model) 또는 gbliner(linear model) 선택	gbtree
02	**silent**	실행 메시지를 출력하고 싶지 않은 경우 1로 설정.	0
03	**nthread**	CPU의 실행 스레드 개수를 조정.	CPU 전체 스레드 다 사용

부스터 파라미터

NO	이름	내용	디폴트 값
01	**learning rate**	부스팅을 반복적으로 수행할 때 업데이트 되는 학습률 값.	0.1
02	**n_esimators**	부스트 트리의 갯수	0
03	**max_depth**	트리의 최대 깊이	None
04	**subsample**	데이터를 샘플링하는 비율값.	1
05	**min_child_weight**	하위에 필요한 인스턴스 가중치의 최소 합.	1
06	**gamma**	트리의 리프노드의 추가분할을 결정할 최소 손실 감소값.	0

부스터 파라미터

NO	이름	내용	디폴트 값
01	**reg_lambda**	L2 regulation 적용값.	1
02	**reg_alpha**	L1 regulation 적용값.	0
03	**colsample_bytree**	트리 생성할 때 feature를 샘플링해주는 비율	1

eXtra Gredient Boost 파라미터

과적합 해결

learning_rate

gamma

min_child_weight

subsample,
colsample_bytree

eXtra Gredient Boost 파라미터

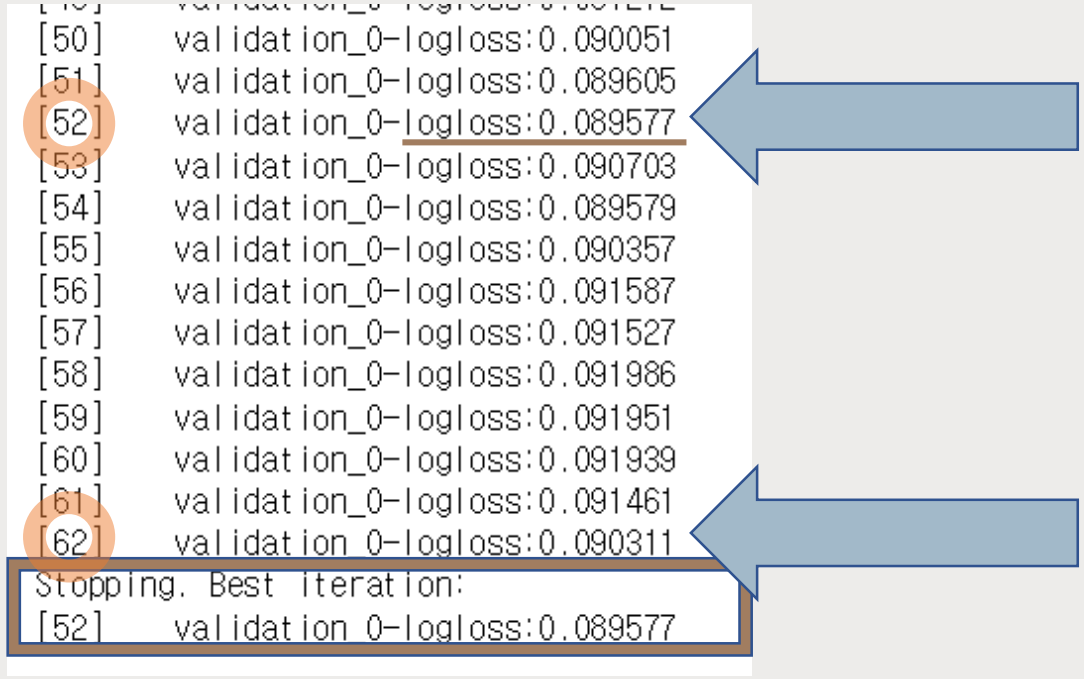
학습 태스크 파라미터

NO	이름	내용	디폴트 값
01	**objective**	최솟값을 가져야할 손실함수를 정의.	reg:linear
02	**eval_metric**	검증에서 사용되는 함수를 정의	default according to objective
03	**early_stopping_rounds**	**더 이상 개선이 없을 때 반복을 멈추고 조기 중단할 수 있는 최소 반복 횟수**	**None**

eXtra Gredient Boost 파라미터

Early_Stopping

```
xgb_wrapper = XGBClassifier()
evals=[(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=10,
                eval_metric='logloss', eval_set=evals, verbose=True)
w_preds=xgb_wrapper.predict(X_test)
```



XGBClassifier를 이용한 유방암 분석

1step

분석 전
데이터
엿보기

2step

XGBClassifier
모델 구축

3step

구축한 모델의
성능측정 및
피처중요도
알아보기

1step - 분석 전 데이터 엿보기 (Breast Cancer)

In [22]:
```
#사이킷런 래퍼 XGBoost클래스인 XGBClassifier임포트
from xgboost import XGBClassifier
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

dataset=load_breast_cancer()
X_features=dataset.data
y_label=dataset.target
cancer_df=pd.DataFrame(data=X_features,columns=dataset.feature_names)
cancer_df['target']=y_label
cancer_df.head(3)   #본격적인 데이터 분석 전 판다스 데이터 프레임 객체로 데이터 모양 확인
```

Out[22]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	con
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.6	2019.0	0.1622	
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.8	1956.0	0.1238	
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.5	1709.0	0.1444	

3 rows × 31 columns

```
#사이킷런 래퍼 XGBoost클래스인 XGBClassifier임포트
from xgboost import XGBClassifier
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

```
dataset=load_breast_cancer()
X_features=dataset.data
y_label=dataset.target
cancer_df=pd.DataFrame(data=X_features,columns=dataset.feature_names)
cancer_df['target']=y_label
cancer_df.head(3)  #본격적인 데이터 분석 전 판다스 데이터 프레임 객체로 데이터 모양 확인
```

30 Features (Breast Cancer Data)

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	con
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.6	2019.0	0.1622	
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.8	1956.0	0.1238	
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.5	1709.0	0.1444	

3 rows × 31 columns

2step-XGBClassifier모델 구축

```
#테스트셋 20% 분리
X_train,X_test,y_train,y_test=train_test_split(X_features,y_label,test_size=0.2,random_state=156)
```

```
evals=[(X_test,y_test)]
xgb_wrapper=XGBClassifier(n_estimators=400,learning_rate=0.1,max_depth=3)
xgb_wrapper.fit(X_train,y_train,
                early_stopping_rounds=100,eval_metric='logloss',eval_set=evals,
                verbose=True)
ws100_preds=xgb_wrapper.predict(X_test)
```

```
[0]     validation_0-logloss:0.61352
Will train until validation_0-logloss hasn't improved in 100 rounds.
[1]     validation_0-logloss:0.547842
[2]     validation_0-logloss:0.494247
[3]     validation_0-logloss:0.447986
[4]     validation_0-logloss:0.409109
[5]     validation_0-logloss:0.374977
[6]     validation_0-logloss:0.345714
[7]     validation_0-logloss:0.320529
[8]     validation_0-logloss:0.29721
[9]     validation_0-logloss:0.277991
[10]    validation_0-logloss:0.260302
[11]    validation_0-logloss:0.246037
[12]    validation_0-logloss:0.231556
[13]    validation_0-logloss:0.22005
[14]    validation_0-logloss:0.208572
[15]    validation_0-logloss:0.199993
[16]    validation_0-logloss:0.190118
[17]    validation_0-logloss:0.181818
[18]    validation_0-logloss:0.174729
```

```
#테스트셋 20% 분리
X_train,X_test,y_train,y_test=train_test_split(X_features,y_label,test_size=0.2,random_state=156)
```

테스트 셋 20%분리

Eval set을 테스트 셋으로 정의

XGB분류기 모델 설정

```
evals=[(X_test,y_test)]
xgb_wrapper=XGBClassifier(n_estimators=400,learning_rate=0.1,max_depth=3)
xgb_wrapper.fit(X_train,y_train,
                early_stopping_rounds=100,eval_metric='logloss',eval_set=evals,
                verbose=True)
ws100_preds=xgb_wrapper.predict(X_test)
```

학습시키기

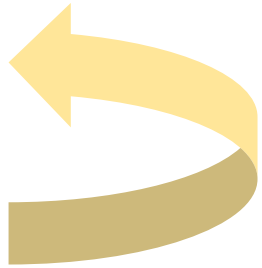
테스트 셋으로 예측하기

Finding Minimum ‘LogLoss’

```
[0]     validation_0-logloss:0.61352
Will train until validation_0-logloss hasn't improved in 100 rounds.
[1]     validation_0-logloss:0.547842
[2]     validation_0-logloss:0.494247
[3]     validation_0-logloss:0.447986
[4]     validation_0-logloss:0.409109
[5]     validation_0-logloss:0.374977
[6]     validation_0-logloss:0.345714
[7]     validation_0-logloss:0.320529
[8]     validation_0-logloss:0.29721
[9]     validation_0-logloss:0.277991
[10]    validation_0-logloss:0.260302
[11]    validation_0-logloss:0.246037
[12]    validation_0-logloss:0.231556
[13]    validation_0-logloss:0.22005
[14]    validation_0-logloss:0.208572
[15]    validation_0-logloss:0.199993
[16]    validation_0-logloss:0.190118
[17]    validation_0-logloss:0.181818
[18]    validation_0-logloss:0.174729
```



```
[311]   validation_0-logloss:0.085948
Stopping. Best iteration:
[211]   validation_0-logloss:0.085593
```

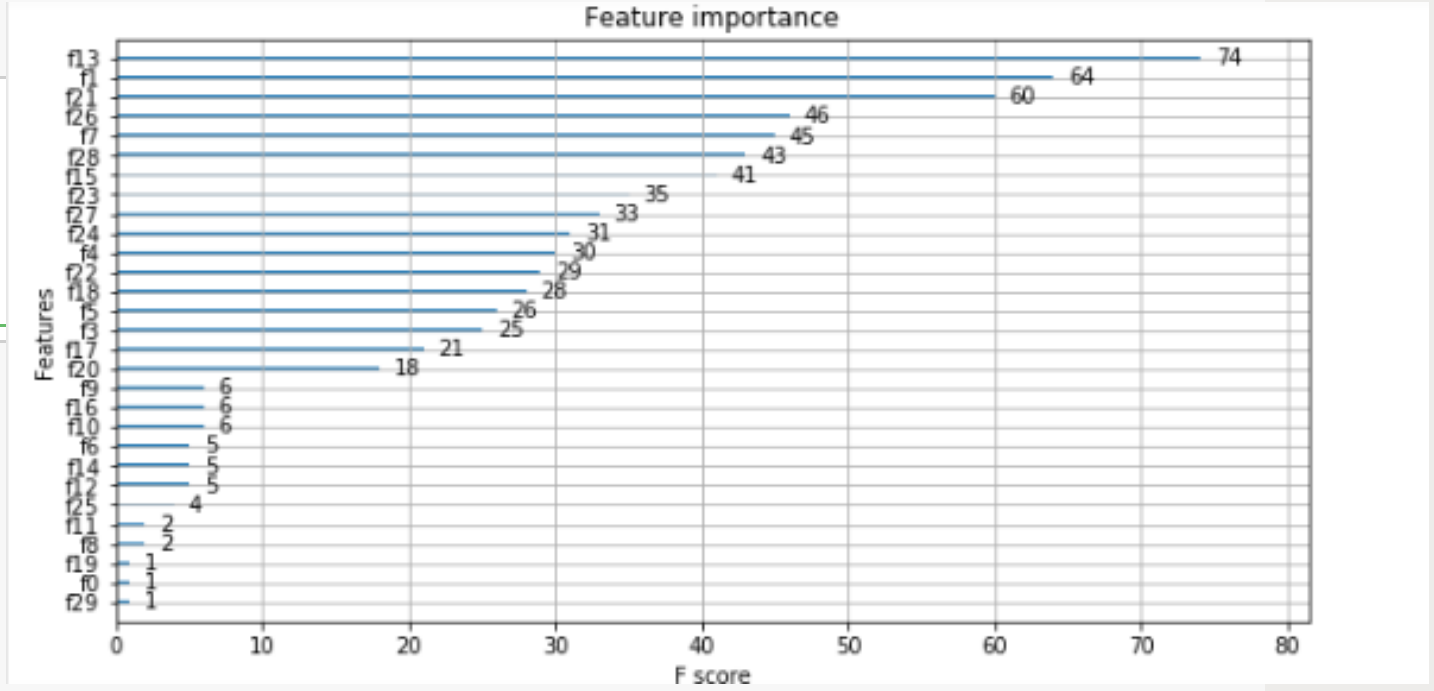


3step-구축한 모델의 성능측정 및 피처중요도 알아보기

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix

def get_clf_eval(y_test , pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}'.format(accuracy, precision, recall))

get_clf_eval(y_test,ws100_preds)
```

```
오차 행렬
[[34  3]
 [ 1 76]]
정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870
```

```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fix,ax =plt.subplots(figsize=(10,12))

plot_importance(xgb_wrapper,ax=ax)
```



```python
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix

def get_clf_eval(y_test , pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}'.format(accuracy, precision, recall))

get_clf_eval(y_test,ws100_preds)
```

3장의
get_clf_eval
함수를 이용

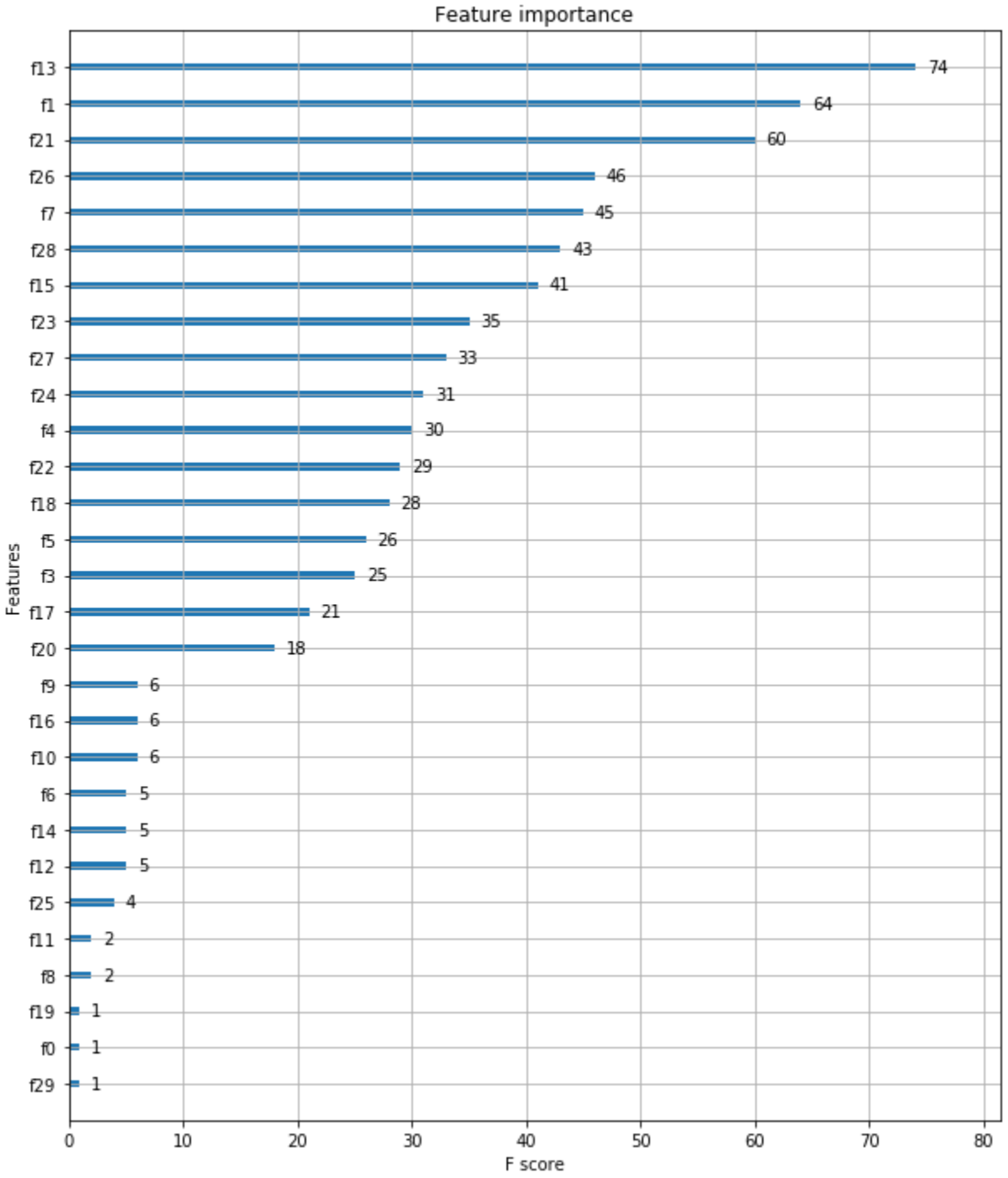
```
오차 행렬
[[34  3]
 [ 1 76]]
```

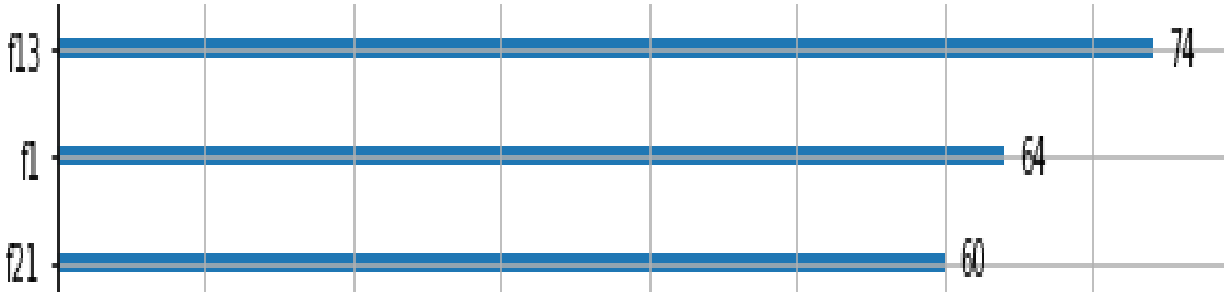
TP	FN
FP	TN

정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870

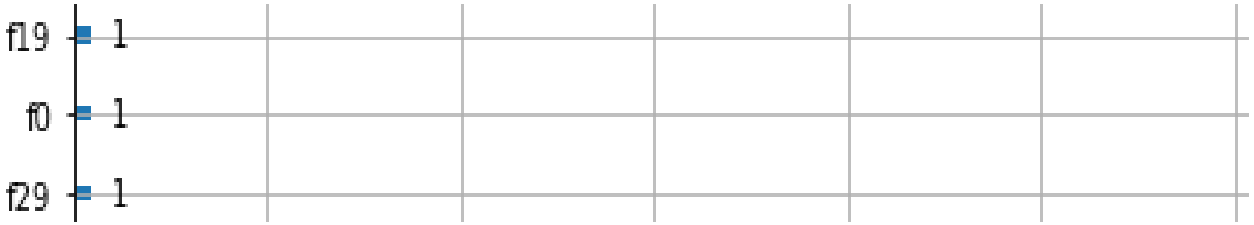
```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fix,ax =plt.subplots(figsize=(10,12))

plot_importance(xgb_wrapper,ax=ax)
```

Xgboost의 plot_importance함수의 기능은?
=>데이터의 피처 중요도를 그래프로 보여준다.
피처 중요도란?
=> label의 결정에 각 피처가 얼마나 영향을 주는 지에 대한 척도





피처 중요도가 높은 성분들



피처 중요도가 낮은 성분들

3. Light GBM

Light Gradient Boosting Machine

XGBoost VS Light GBM

XGBoost

과적합이 일어날 확률이 적다	1. 메모리 사용량이 많다 2. 수행시간이 길다 3. 파라미터 튜닝 <- (GPU, 병렬처리)

Light GBM

1. 학습과 예측수행에 걸리는 시간이 짧다 2. XGBoost와 성능 동일 3. 메모리 사용량이 비교적 적다 4. 명목형 변수의 자동변환과 최적분할	과적합이 발생하기 쉽다

Light GBM - Parameters

Parameters	Detail
Num_iterations[n_estimators](100)	반복 수행하려는 트리의 개수 (많을수록 성능 △, 과적합확률 ▲)
learning_rate(0.1)	(작을수록 성능△, 과적합확률▲, 수행시간 ▲)
Max_depth(-1)	0이하일 때는 깊이 제한 X (과적합확률 ▲)
Boosting(gbdt)	부스팅의 트리로 사용할 모델
Min_data_in_leaf(20)	최종 결정클래스가 되기 위한 최소 데이터의 양 (낮을수록 과적합확률 ▲)
feature_fraction(1)	개별 트리를 학습할 때마다 무작위로 선택하는 피처 비율. (낮을수록 과적합확률 ▲)
bagging_fraction(1) [sub_sample]	데이터를 랜덤 샘플링하는 비율 (낮을수록 과적합확률▲)
Lambda(0)	정규화(Regularization) 정도 조절 (낮을수록 과적합확률▲)

Light GBM – Hyper Parameter Tuning, **HOW** ???

1. 성능 향상

Max_depth△	Learning_rate ▽
N_estimators △	Min_data_in_leaf ▽
Num_leaves △	

2. 과적합 감소

Max_depth▽	LambdaL2, L1△
N_estimators ▽	Min_data_in_leaf △
Num_leaves ▽	

Light GBM – Practice

(Breast Cancer Detection)

```python
from lightgbm import LGBMClassifier

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

dataset=load_breast_cancer()
ftr=dataset.data
target=dataset.target

X_train, X_test, y_train, y_test = train_test_split(ftr,target,test_size=0.2, random_state=156)

lgbm_wrapper=LGBMClassifier(n_estimators=400)

evals=[(X_test,y_test)]
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric='logloss', # earlystopping 지정(100)
                 eval_set=evals, verbose=True)
preds=lgbm_wrapper.predict(X_test)
```

```
[114]   valid_0's binary_logloss: 0.166509      valid_0's binary_logloss: 0.166509
[115]   valid_0's binary_logloss: 0.165823      valid_0's binary_logloss: 0.165823
[116]   valid_0's binary_logloss: 0.167059      valid_0's binary_logloss: 0.167059
[117]   valid_0's binary_logloss: 0.169086      valid_0's binary_logloss: 0.169086
[118]   valid_0's binary_logloss: 0.170012      valid_0's binary_logloss: 0.170012
[119]   valid_0's binary_logloss: 0.168639      valid_0's binary_logloss: 0.168639
[120]   valid_0's binary_logloss: 0.16907       valid_0's binary_logloss: 0.16907
[121]   valid_0's binary_logloss: 0.16918       valid_0's binary_logloss: 0.16918
[122]   valid_0's binary_logloss: 0.170233      valid_0's binary_logloss: 0.170233
[123]   valid_0's binary_logloss: 0.165655      valid_0's binary_logloss: 0.165655
[124]   valid_0's binary_logloss: 0.16695       valid_0's binary_logloss: 0.16695
[125]   valid_0's binary_logloss: 0.170955      valid_0's binary_logloss: 0.170955
[126]   valid_0's binary_logloss: 0.168916      valid_0's binary_logloss: 0.168916
[127]   valid_0's binary_logloss: 0.172316      valid_0's binary_logloss: 0.172316
[128]   valid_0's binary_logloss: 0.173734      valid_0's binary_logloss: 0.173734
[129]   valid_0's binary_logloss: 0.174309      valid_0's binary_logloss: 0.174309
[130]   valid_0's binary_logloss: 0.176719      valid_0's binary_logloss: 0.176719
[131]   valid_0's binary_logloss: 0.176591      valid_0's binary_logloss: 0.176591
[132]   valid_0's binary_logloss: 0.180168      valid_0's binary_logloss: 0.180168
[133]   valid_0's binary_logloss: 0.179856      valid_0's binary_logloss: 0.179856
[134]   valid_0's binary_logloss: 0.179251      valid_0's binary_logloss: 0.179251
[135]   valid_0's binary_logloss: 0.18315       valid_0's binary_logloss: 0.18315
[136]   valid_0's binary_logloss: 0.184656      valid_0's binary_logloss: 0.184656
[137]   valid_0's binary_logloss: 0.187475      valid_0's binary_logloss: 0.187475
[138]   valid_0's binary_logloss: 0.188721      valid_0's binary_logloss: 0.188721
[139]   valid_0's binary_logloss: 0.188542      valid_0's binary_logloss: 0.188542
[140]   valid_0's binary_logloss: 0.18817       valid_0's binary_logloss: 0.18817
[141]   valid_0's binary_logloss: 0.185899      valid_0's binary_logloss: 0.185899
[142]   valid_0's binary_logloss: 0.185452      valid_0's binary_logloss: 0.185452
[143]   valid_0's binary_logloss: 0.186084      valid_0's binary_logloss: 0.186084
[144]   valid_0's binary_logloss: 0.185302      valid_0's binary_logloss: 0.185302
[145]   valid_0's binary_logloss: 0.187856      valid_0's binary_logloss: 0.187856
[146]   valid_0's binary_logloss: 0.190334      valid_0's binary_logloss: 0.190334
[147]   valid_0's binary_logloss: 0.192769      valid_0's binary_logloss: 0.192769
Early stopping, best iteration is:
[47]    valid_0's binary_logloss: 0.126108      valid_0's binary_logloss: 0.126108
```

loss 감소

Light GBM – Practice

(Breast Cancer Detection)

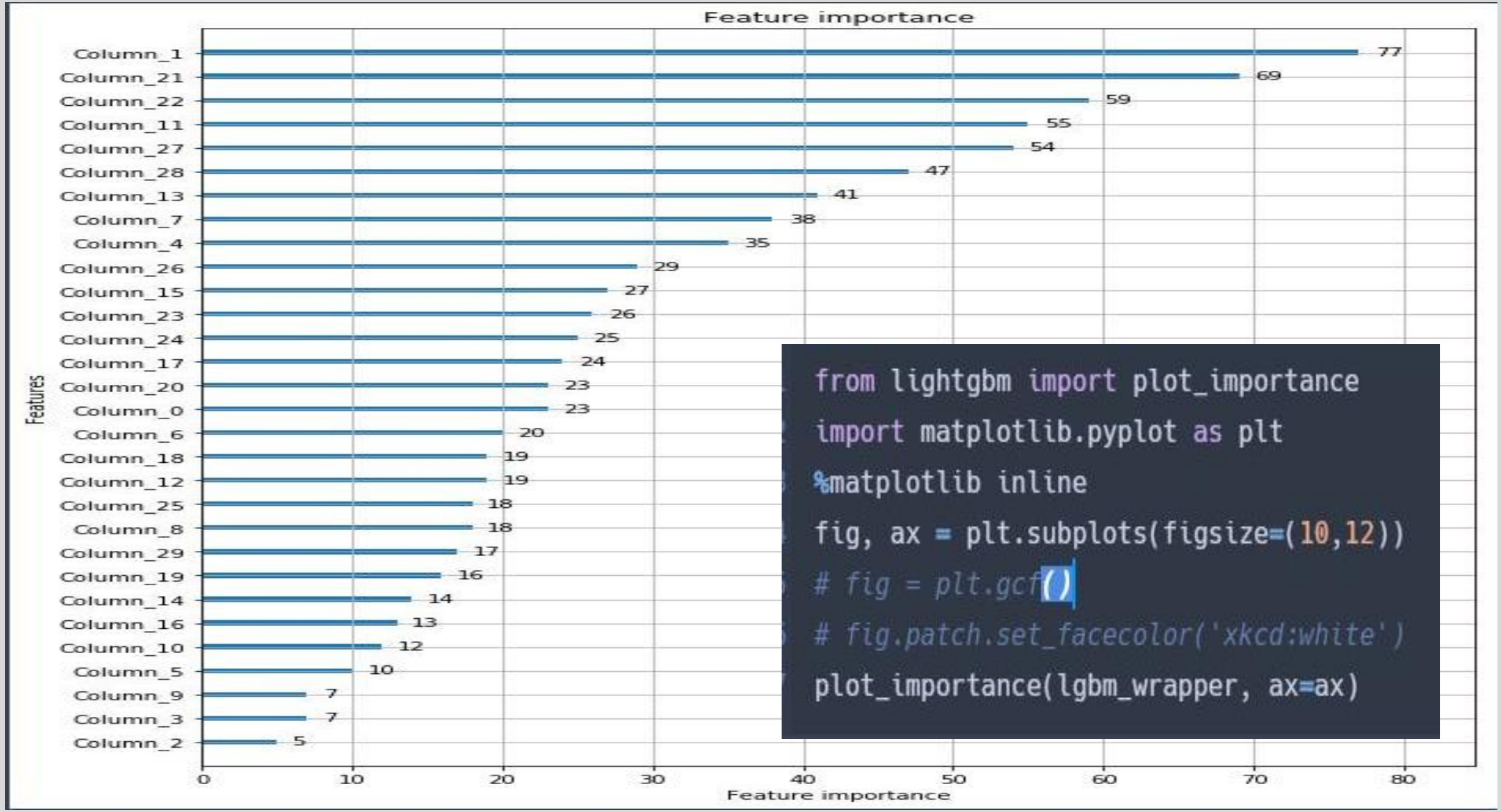
```python
from lightgbm import LGBMClassifier

import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_t

dataset=load_breast_cancer()
ftr=dataset.data
target=dataset.target

X_train, X_test, y_train, y_test = train_te

lgbm_wrapper=LGBMClassifier(n_estimators=400)

evals=[(X_test,y_test)]
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric='logloss', # earlystopping 지정(100)
                 eval_set=evals, verbose=True)
preds=lgbm_wrapper.predict(X_test)
```

```
[114]   valid_0's binary_logloss: 0.166509      valid_0's binary_logloss: 0.166509
[115]   valid_0's binary_logloss: 0.165823      valid_0's binary_logloss: 0.165823
[116]   valid_0's binary_logloss: 0.167059      valid_0's binary_logloss: 0.167059
[117]   valid_0's binary_logloss: 0.169086      valid_0's binary_logloss: 0.169086
[118]   valid_0's binary_logloss: 0.170012      valid_0's binary_logloss: 0.170012
[119]   valid_0's binary_logloss: 0.168639      valid_0's binary_logloss: 0.168639
[120]   valid_0's binary_logloss: 0.16907       valid_0's binary_logloss: 0.16907
[121]   valid_0's binary_logloss: 0.16918       valid_0's binary_logloss: 0.16918
[122]   valid_0's binary_logloss: 0.170233      valid_0's binary_logloss: 0.170233
[123]   valid_0's binary_logloss: 0.165655      valid_0's binary_logloss: 0.165655
                                                                          gloss: 0.16695
                                                                          gloss: 0.170955
                                                                          gloss: 0.168916
                                                                          gloss: 0.172316
                                                                          gloss: 0.173734
                                                                          gloss: 0.174309
                                                                          gloss: 0.176719
                                                                          gloss: 0.176591
                                                                          gloss: 0.180168
                                                                          gloss: 0.179856
                                                                          gloss: 0.179251
                                                                          gloss: 0.18315
                                                                          gloss: 0.184656
                                                                          gloss: 0.187475
[138]   valid_0's binary_logloss: 0.188721      valid_0's binary_logloss: 0.188721
[139]   valid_0's binary_logloss: 0.188542      valid_0's binary_logloss: 0.188542
[140]   valid_0's binary_logloss: 0.18817       valid_0's binary_logloss: 0.18817
[141]   valid_0's binary_logloss: 0.185899      valid_0's binary_logloss: 0.185899
[142]   valid_0's binary_logloss: 0.185452      valid_0's binary_logloss: 0.185452
[143]   valid_0's binary_logloss: 0.186084      valid_0's binary_logloss: 0.186084
[144]   valid_0's binary_logloss: 0.185302      valid_0's binary_logloss: 0.185302
[145]   valid_0's binary_logloss: 0.187856      valid_0's binary_logloss: 0.187856
[146]   valid_0's binary_logloss: 0.190334      valid_0's binary_logloss: 0.190334
[147]   valid_0's binary_logloss: 0.192769      valid_0's binary_logloss: 0.192769
Early stopping, best iteration is:
[47]    valid_0's binary_logloss: 0.126108      valid_0's binary_logloss: 0.126108
```

```
1  get_clf_eval(y_test,preds)

오차 행렬
[[33  4]
 [ 2 75]]
정확도: 0.9474, 정밀도: 0.9494, 재현율: 0.9740,    F1: 0.9615, AUC:0.9330
```

Light GBM – Practice

(Breast Cancer Detection)




```
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots(figsize=(10,12))
# fig = plt.gcf()
# fig.patch.set_facecolor('xkcd:white')
plot_importance(lgbm_wrapper, ax=ax)
```

과제??

Iris data

GBM

XGBoost

LightGBM

Wine data

GBM

XGBoost

LightGBM

Q&A

THANK YOU