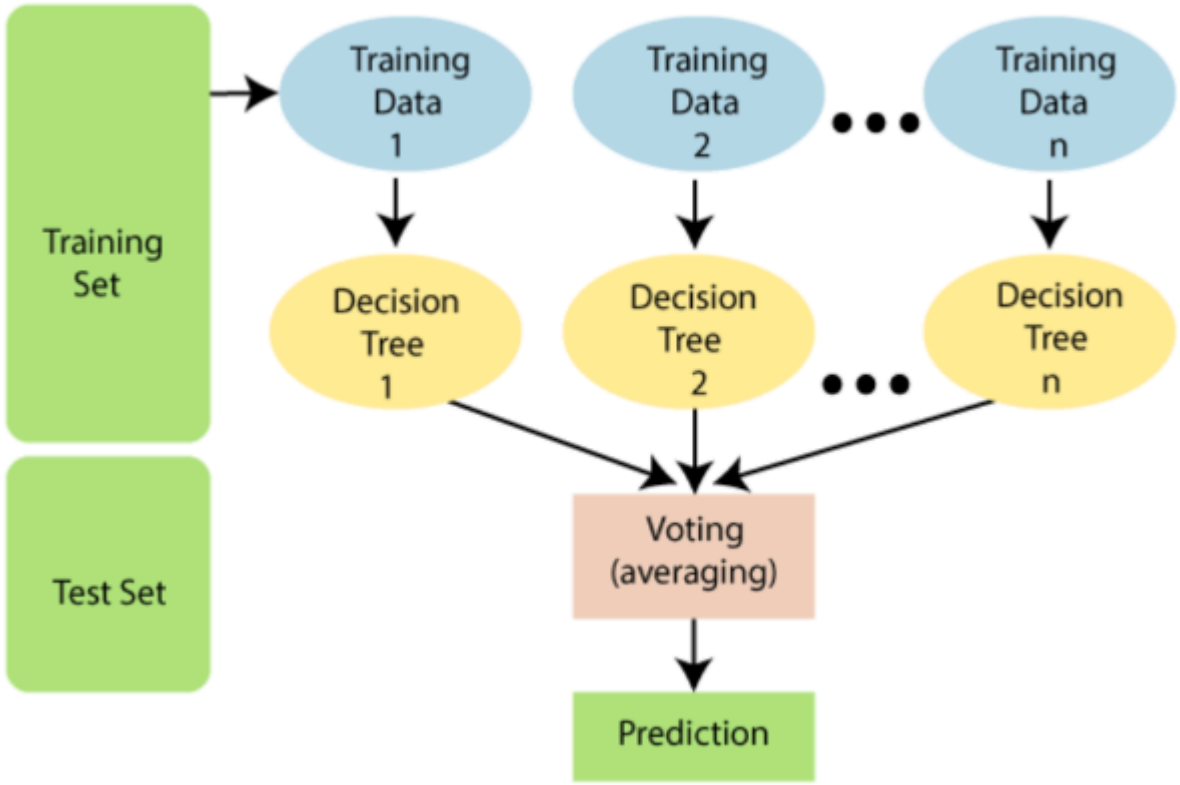
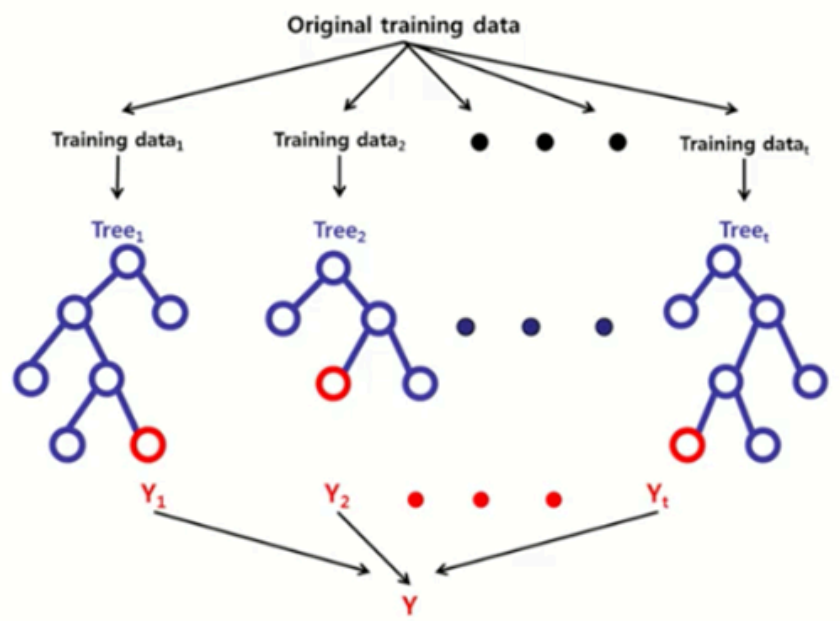
랜덤 포레스트 (Random Forest)

- 배깅의 일종으로, 학습시키는 모델을 결정트리(Decision Tree)로 정하여 이 들의 결과를 소프트 보팅의 방식을 통해 Aggregating하는 방법

  - 같은 알고리즘으로 여러 개의 분류기를 만들어서 보팅으로 최종 결정하는 알고리즘
  - 랜덤 포레스트는 결정 트리의 기반 알고리즘으로써 쉽고 직관적인 장점을 그대로 가지고 있음
  - 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정함





부트스트래핑 데이터분할

랜덤포레스트의 작동방식

- 1) 데이터의 부분집합(Bootstrap 샘플)을 생성
- 2) 각 노드에서 최적의 분할을 결정할 때, 전체 feature중 일부 feature만을 무작위로 선택하여 사용(이러한 기법을 "랜덤 서브스페이스(Random Subspace)" 또는 "랜덤 특성(Random Features)"이라고 함)
- 3) 각 결정 트리의 예측을 종합하여 최종 예측을 수행

랜덤포레스트의 주요 특징 및 장점

- 과적합을 감소시키며 모델의 일반화 성능을 향상시킴

- Out-of-Bag(OOB) 샘플을 사용하여 모델 성능을 평가할 수 있음

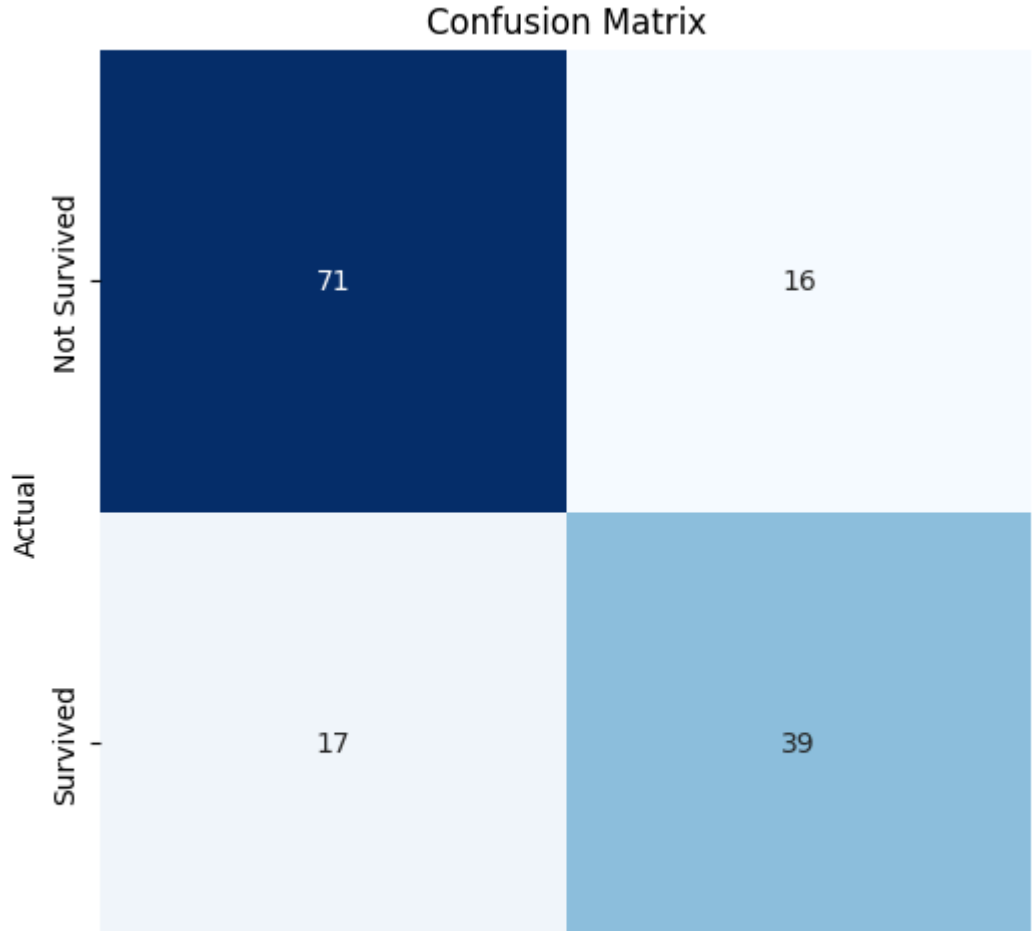
  > - Out-of-Bag(OOB) - 랜덤 포레스트의 각 트리가 생성될 때, 부트스트랩(복원 추출) 방법으로 선택되지 않은 데이터 샘플을 의미

- Feature 중요도를 측정할 수 있어, 어떤 feature가 예측에 가장 중요한지 파악할 수 있음
- 병렬 처리가 가능하여 대규모 데이터셋에 대해 빠른 학습이 가능함

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#데이터 불러오기
url='https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
df=pd.read_csv(url)

#데이터 전처리: 필요한 열 선택 및 결측치 처리
df=df[['Pclass','Sex','Age','SibSp','Parch','Fare','Survived']].dropna()

#범주형 변수를 숫자로 변환 (원핫 인코딩)
df=pd.get_dummies(df,columns=['Sex'],drop_first=True)

#Features(X)와 Target(y) 분리
X=df.drop('Survived',axis=1)
y=df['Survived']

#데이터를 학습용과 테스트용으로 분리
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

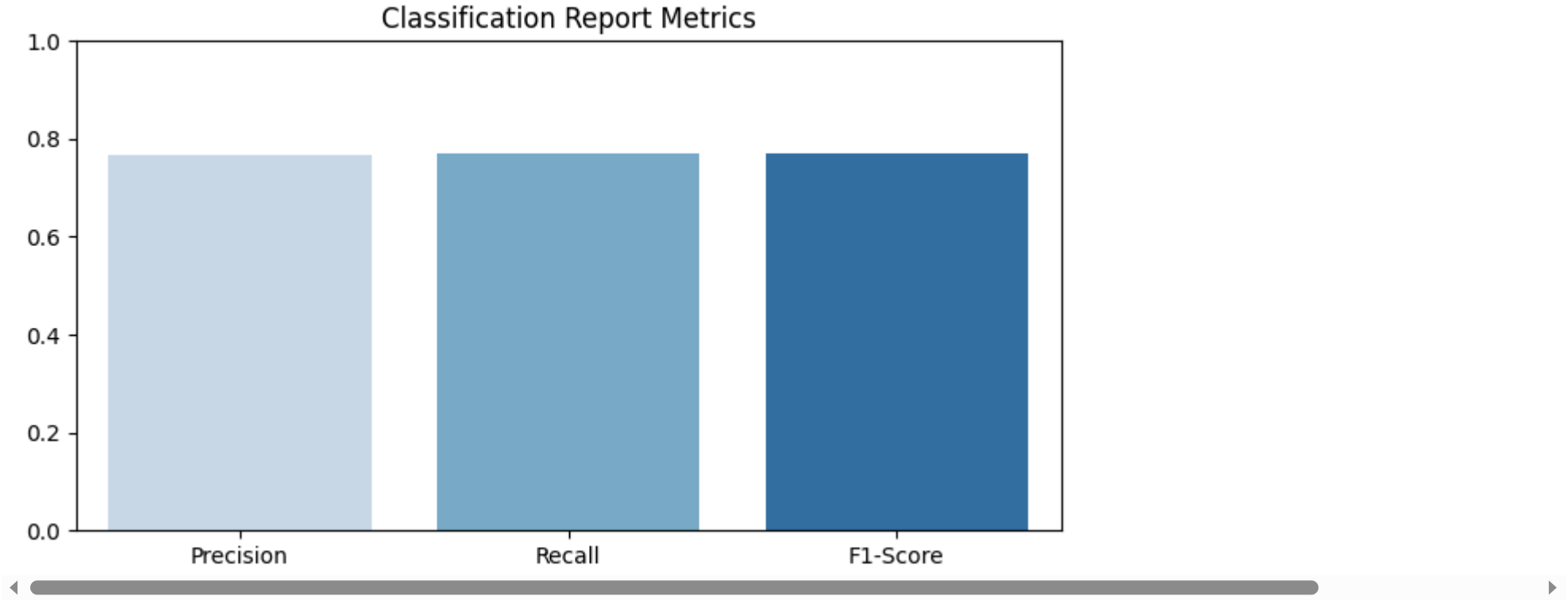
```
!pip install iexfinance --upgrade
# 랜덤포레스트 모델 생성 및 학습
rf_model=RandomForestClassifier(n_estimators=100,random_state=42)
rf_model.fit(X_train,y_train)

#모델 평가
y_pred=rf_model.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
conf_matrix=confusion_matrix(y_test,y_pred)
class_report=classification_report(y_test,y_pred)

#결과 출력
print(f'Accuracy:{accuracy:.2f}')

# Confusion Matrix 시각화
plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix,annot=True,fmt='d',cmap='Blues',cbar=False,xticklabels=['Not Survived','Survived'],yticklabels=['Not Survived','Survi
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()# Classification Report 추출
class_report_str=classification_report(y_test,y_pred)

#Classification Report 문자열을 Pandas DataFrame으로 변환
#class_report_df=pd.read_csv(pd.compat.StringIO(class_report_str),sep='WWs+')

# 출력
#print(class_report_df)
```

```
Requirement already satisfied: iexfinance in /usr/local/lib/python3.11/dist-packages (0.5.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from iexfinance) (2.2.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from iexfinance) (2.32.3)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas->iexfinance) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->iexfinance) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->iexfinance) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->iexfinance) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->iexfinance) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->iexfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->iexfinance) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->iexfinance) (2025.1.31)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->iexfinance) (1.17.0
Accuracy:0.77
```



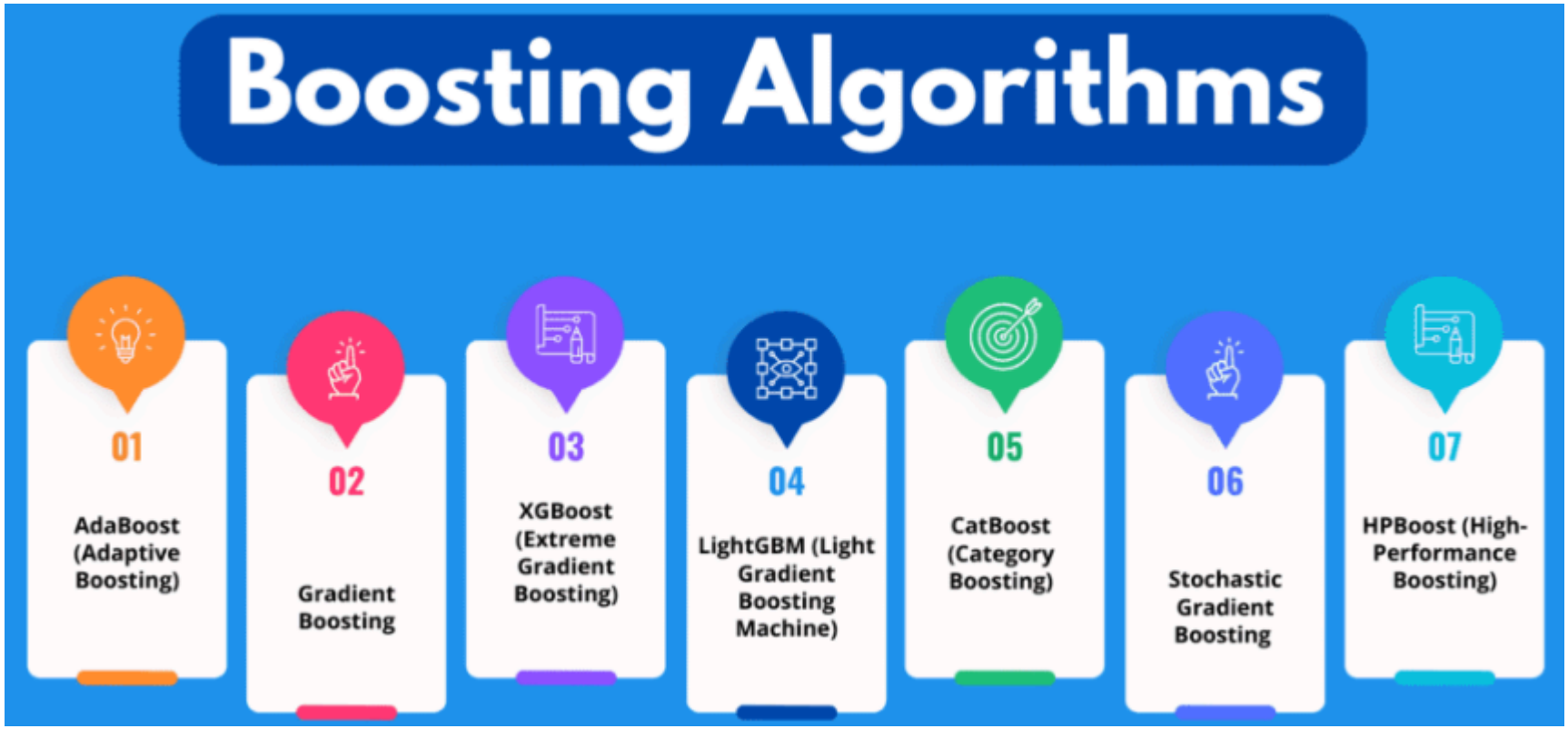

```
#Classification Report 시각화
report_dict=classification_report(y_test,y_pred,output_dict=True)
precision=report_dict['weighted avg']['precision']
recall=report_dict['weighted avg']['recall']
f1_score=report_dict['weighted avg']['f1-score']
print('Classification Report:₩n',class_report)

#시각화
plt.figure(figsize=(8,4))
sns.barplot(x=['Precision','Recall','F1-Score'],y=[precision,recall,f1_score],palette='Blues')
plt.title('Classification Report Metrics')
plt.ylim(0,1)
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.82      0.81        87
           1       0.71      0.70      0.70        56

    accuracy                           0.77       143
   macro avg       0.76      0.76      0.76       143
weighted avg       0.77      0.77      0.77       143

<ipython-input-10-2ddc091c67bf>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=Fal

  sns.barplot(x=['Precision','Recall','F1-Score'],y=[precision,recall,f1_score],palette='Blues')
```



부스팅 알고리즘

- 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치를 부여를 통해 오류를 개선해 나가면서 학습하는 방식
- 대표적으로 AdaBoost(Adaptive boosting)와 그래디언트 부스트가 있음

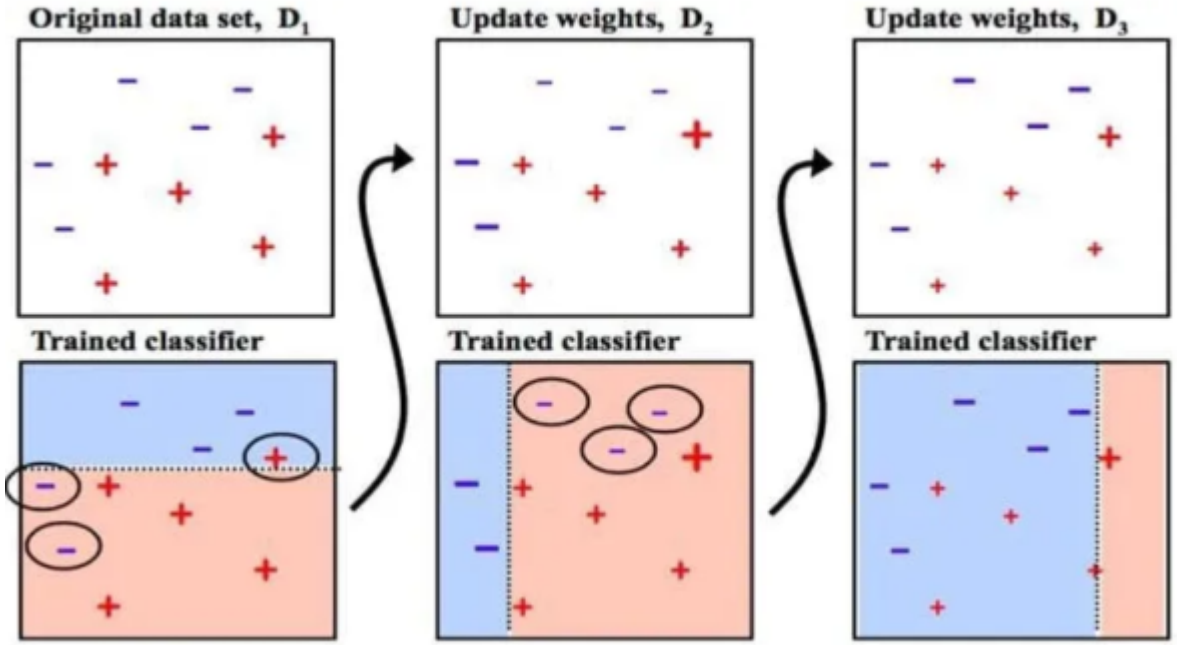


AdaBoost

- 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 대표적인 알고리즘

> - a) 먼저, 첫번째 약한 학습기로 최초 데이터(D1) 분류를 한 뒤, 잘못 분류된 것들에 대해 표시하여 해당 데이터에 가중치 값이 부여된 업데이트 된 데이터셋(D2)을 구축
> - b) 업데이트 된 데이터로 다시 약한 학습기로 분류를 한 뒤, 다시 잘못 분류된 것들에 대해 표시하고 마찬가지로 해당 데이터에 가중치 값이 부여된 업데이트 된 데이터셋(D3)을 구축

- c) 지금까지 총 3번의 약한 학습기가 구축되었고, 이 학습기들을 가중치에 따라 합치면 원하는 결과가 분류되는 분류기가 만들어짐



```
#base_estimator : 앙상블에 포함될 기본 분류기 종류, 기본값으로 DecisionTreeClassifier(max_depth=1)를 사용
#n_estimators : 모델에 포함될 분류기 개수
#learning_rate : 학습률
#algorithm : 부스팅 알고리즘, 기본값='SAMME.R'
#random_state : 난수 seed 설정
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
train_x, test_x, train_y, test_y = train_test_split(cancer['data'], cancer['target'], test_size=0.2)
```

```
13
14 ada_boost = AdaBoostClassifier(n_estimators=500, learning_rate=0.1)
15 ada_boost.fit(train_x, train_y)
16 pred = ada_boost.predict(test_x)
17 acc = accuracy_score(test_y, pred)
18 print('AdaBoost 정확도: {0:.4f}'.format(acc))
```

```
AdaBoost 정확도: 0.9561
```

GBM(Gradient Boost Machine)

- 에이다 부스트와 유사하나, 가중치 업데이트를 경사 하강법(Gradient Descent)을 이용하는 것이 큰 차이임
  - 경사 하강법은 머신러닝과 딥 러닝에서 모델을 학습시키기 위해 사용되는 최적화 알고리즘 중 하나
    - 이는, 모델의 손실 함수(Loss Function)를 최소화하기 위해 모델의 파라미터(parameter)를 조정하는 방법 중 하나(특히, 다수의 변수와 데이터를 포함한 빅데이터에 대한 분석 시, 빠른 연산과 연산 오류를 방지하기위해 자주 사용됨
    - 분류의 실제 결괏값을 y, 피처를 x1, x2,..., xn 그리고 이 예 함수를 F(x)라고 하면 h(x) = y - F(x)이고, 이 식이 최소화되는 방향성을 가지고 경사 하강법으로 가중치를 업데이트하는 것
- GBM은 CART 기반으로 분류와, 회귀 모두 가능

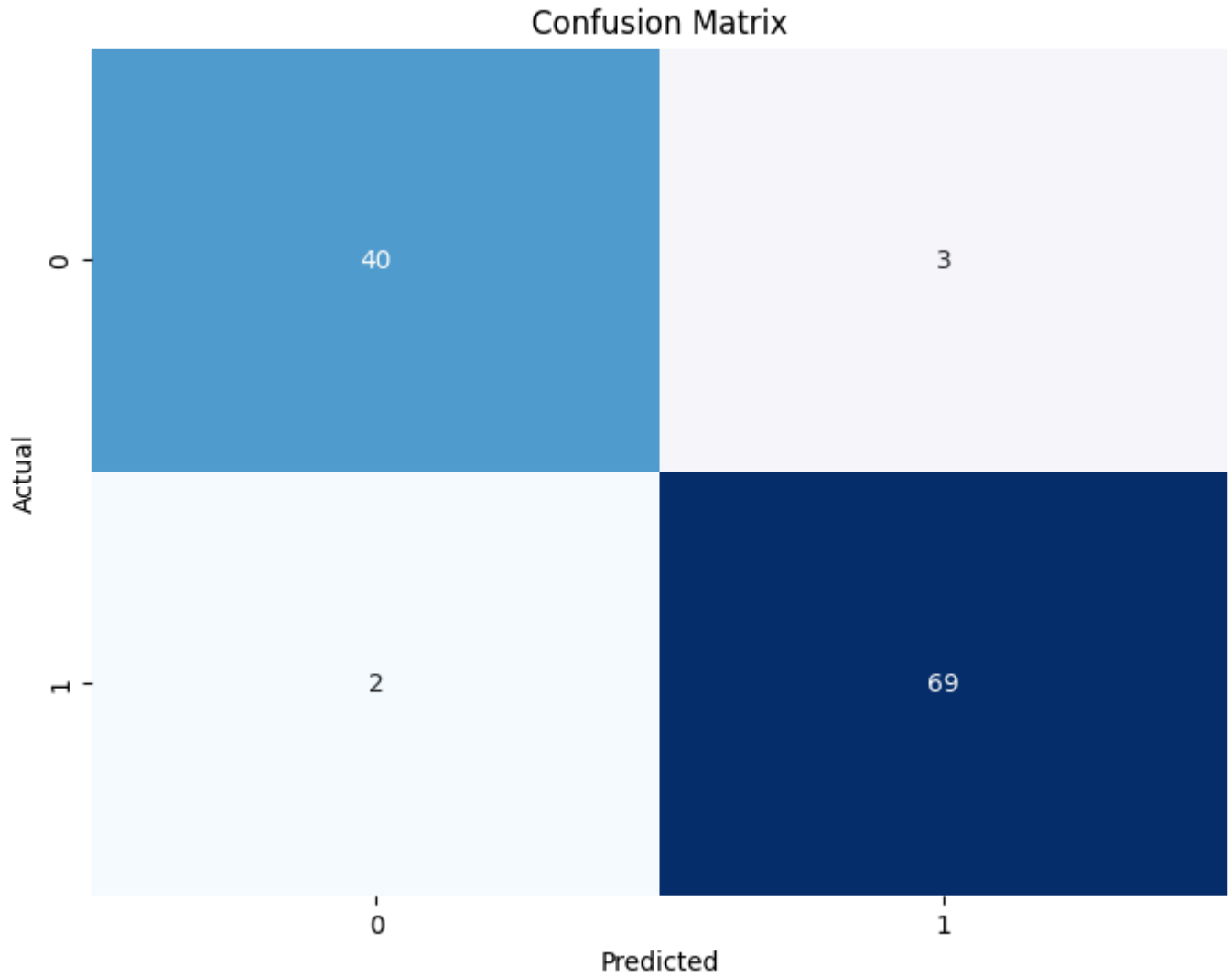
```
1 #loss: 손실 함수를 지정합니다. (deviance, exponential), default는 deviance
2 #n_estimators: 약한 분류기 개수
3 #learning_rate: (0~1) default=0.1, 너무 작게 설정하면 모든 약한 학습기 반복이 완료돼도 최소 오류 값을 찾지 못할 수 있고, 너무 크면 global mir
4 #max_depth: 최대 깊이
```

```
#min_samples_split: 노드를 분할하기 위한 최소한의 심플 데이터수
#min_samples_leaf: 리프 노드가 되기 위한 최소한의 심플 데이터수
#max_features: 최적의 분할을 위해 고려할 피처의 최대 개수(sqrt, auto, log2)
#max_depth: 트리의 최대 깊이
#max_leaf_nodes: 리프 노드의 최대 개수
#subsample: 약한 학습기에 사용하는 데이터 샘플링 비율입니다. (0 ~ 1 사이의 값) default = 1

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
train_x, test_x, train_y, test_y = train_test_split(cancer['data'], cancer['target'], test_size=0.2)

gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.1)
gb.fit(train_x, train_y)
pred = gb.predict(test_x)
acc = accuracy_score(test_y, pred)
print('GradientBoosting 정확도: {0:.4f}'.format(acc))
```

```
GradientBoosting 정확도: 0.9649
```

```
# 필요한 라이브러리 임포트
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score

#Breast Cancer 데이터 로드
data = load_breast_cancer()
X, y = data.data, data.target

#데이터 분할 (학습 데이터와 테스트 데이터)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#GBM 모델 생성
gbm_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth= 3,random_state=42)

#모델 학습
gbm_classifier.fit(X_train, y_train)

#테스트 데이터에 대한 예측
y_pred=gbm_classifier.predict(X_test)

#Confusion Matrix 계산
confusion_mat=confusion_matrix(y_test,y_pred)

#Confusion Matrix 시각화
plt.figure(figsize=(8,6))
sns.heatmap(confusion_mat,annot=True,fmt="d",cmap="Blues",cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

#Classification Report 계산
classification_rep = classification_report(y_test, y_pred)

#결과 출력
print("Classification Report:₩n",classification_rep)

#약 96%의 정확도(Accuracy)로 양성과 음성을 구분해냄
#전체 114개의 데이터 중 단 5개만 잘못분류되었으며, 양성(1)의 경우 97%라는 꽤나 높은 재현율(Recall)을 보여줌
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

XGBoost(eXtrea Gradient Boost)

- GBM에 기반하고 있지만, GBM의 단점인 느린 수행 시간 및 과적합 규제 등의 문제를 해결해서 매우 각광을 받고 있음
- 특히 병렬 CPU 환경에서 병렬 학습이 가능해 기존 GBM보다 빠르게 학습을 완료할 수 있음
- 기존 GBM은 과적합 교제 기능이 없으나 XGBoost는 자체에 과적합 규제 기능이 있음
- 다른 GBM과 마찬가지로 XGBoost도 max_depth 파라미터로 분할 깊이를 조정하기도 하지만, tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기해서 분할 수를 더 줄이는 추가적인 장점을 가지고 있음
- Early Stopping(조기 종료) 기능이 있습니다.

> - 일반 파라미터 - 일반적으로 실행 시 스레드의 개수나 slient 모드 등의 선택을 위한 파라미터로서 디폴트 파라미터 값을 바꾸는 경우는 거의 없음
> - 부스터 파라미터 - 트리 최적화, 부스팅, regularization 등과 관련 파라미터 등을 지칭함
> - 학습 테스트 파라미터 - 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정하는 파라미터임

```
import xgboost as xgb
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
train_x, test_x, train_y, test_y = train_test_split(cancer['data'], cancer['target'], test_size=0.2)

# XGBoost는 학습용 테스트용 데이터 세트를 위해 DMatrix객체를 생성해야함
dtrain = xgb.DMatrix(data=train_x, label=train_y)
```

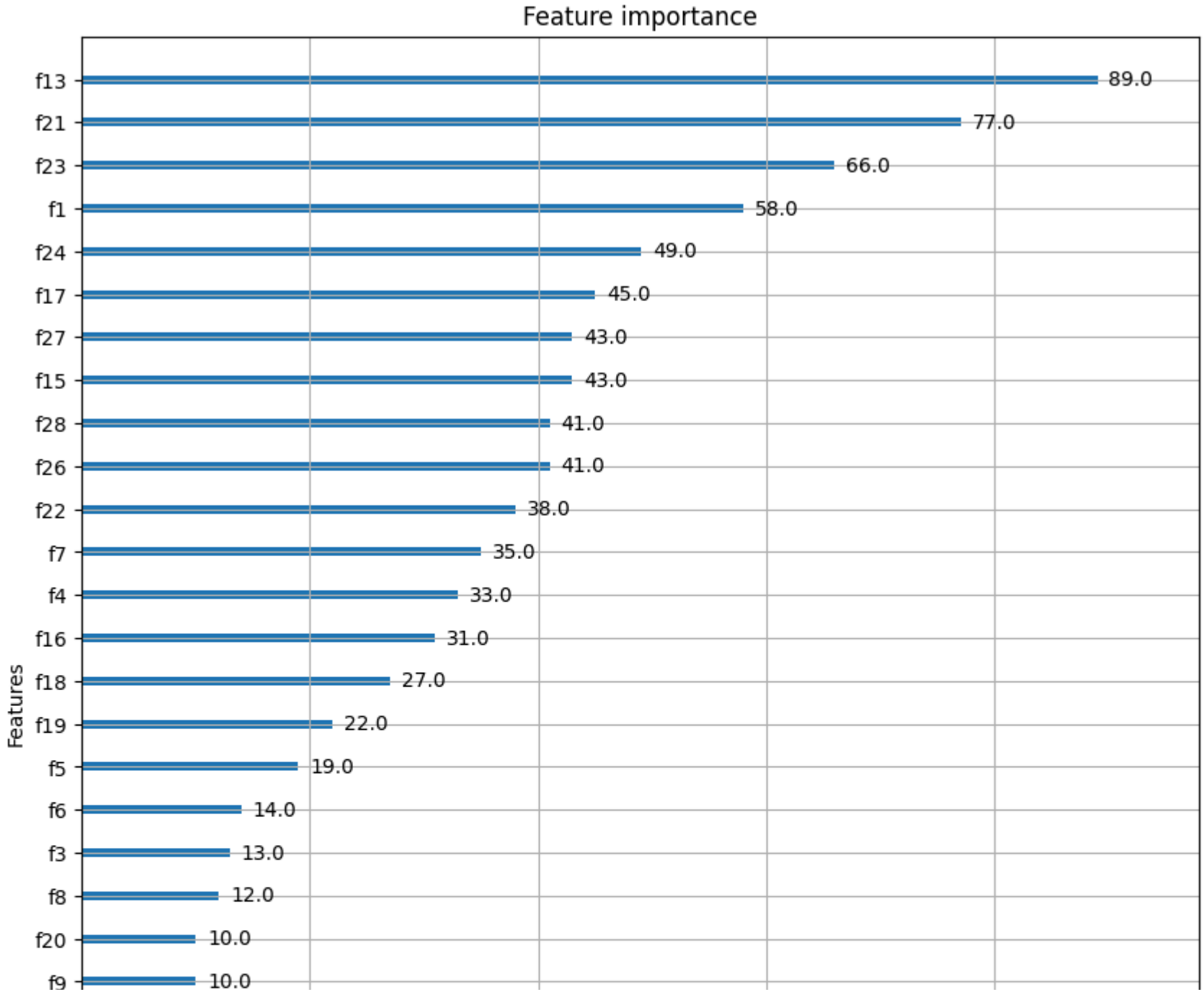
```
dtest = xgb.DMatrix(data=test_x, label=test_y)

params = {'max_depth':3,
          'eta': 0.1,
          'objective':'binary:logistic',
          'eval_metric':'logloss',
          'early_stoppings':100}

num_rounds = 400
# 조기 종료를 위해 eval_set과 eval_metric이 함께 설정 돼어야 함
wlist = [(dtrain, 'train'), (dtest, 'eval')]
xgb_model = xgb.train(params=params, dtrain=dtrain, num_boost_round=num_rounds,
                      early_stopping_rounds=100, evals=wlist)

# xgboost의 predict는 결괏값이 아닌 확률 값을 반환함
pred_prob = xgb_model.predict(dtest)
preds = [1 if i > 0.5 else 0 for i in pred_prob]
print(pred_prob[:10])
print(preds[:10])
print('XGBoost의 정확도: {:.4f}'.format(accuracy_score(test_y, preds)))
```

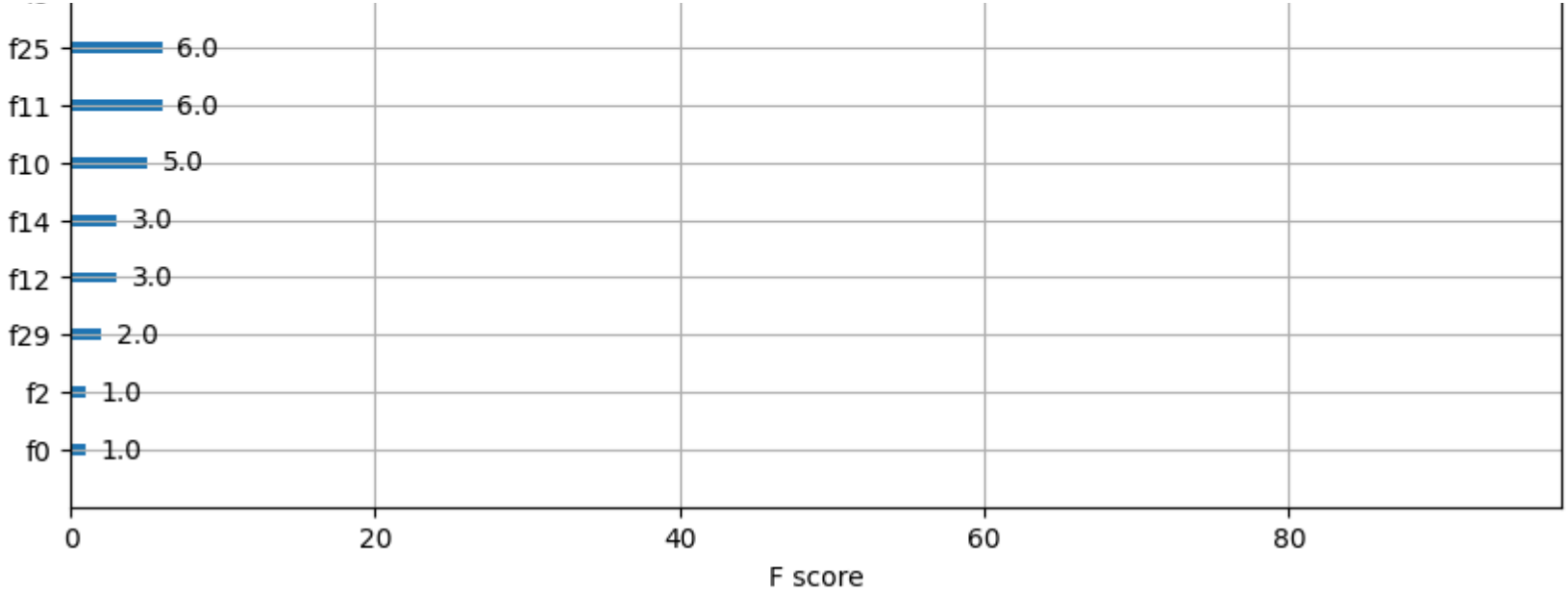
```
[0]	train-logloss:0.58736	eval-logloss:0.56060
[1]	train-logloss:0.52092	eval-logloss:0.50124
[2]	train-logloss:0.46579	eval-logloss:0.45212
[3]	train-logloss:0.41826	eval-logloss:0.40998
[4]	train-logloss:0.37956	eval-logloss:0.37737
[5]	train-logloss:0.34485	eval-logloss:0.34641
[6]	train-logloss:0.31422	eval-logloss:0.31935
[7]	train-logloss:0.28719	eval-logloss:0.29536
[8]	train-logloss:0.26278	eval-logloss:0.27343
[9]	train-logloss:0.24239	eval-logloss:0.25449
[10]	train-logloss:0.22316	eval-logloss:0.23791
[11]	train-logloss:0.20619	eval-logloss:0.22337
[12]	train-logloss:0.19117	eval-logloss:0.21022
[13]	train-logloss:0.17776	eval-logloss:0.19845
[14]	train-logloss:0.16627	eval-logloss:0.18787
[15]	train-logloss:0.15514	eval-logloss:0.17630
[16]	train-logloss:0.14498	eval-logloss:0.16574
[17]	train-logloss:0.13610	eval-logloss:0.15855
```

```
[18]    train-logloss:0.12789   eval-logloss:0.14976
[19]    train-logloss:0.12080   eval-logloss:0.14368
[20]    train-logloss:0.11413   eval-logloss:0.13746
[21]    train-logloss:0.10772   eval-logloss:0.13350
[22]    train-logloss:0.10199   eval-logloss:0.12778
[23]    train-logloss:0.09633   eval-logloss:0.12430
[24]    train-logloss:0.09076   eval-logloss:0.12088
[25]    train-logloss:0.08611   eval-logloss:0.11828
[26]    train-logloss:0.08201   eval-logloss:0.11344
[27]    train-logloss:0.07817   eval-logloss:0.10953
[28]    train-logloss:0.07470   eval-logloss:0.10974
[29]    train-logloss:0.07103   eval-logloss:0.10764
[30]    train-logloss:0.06803   eval-logloss:0.10522
[31]    train-logloss:0.06550   eval-logloss:0.10261
[32]    train-logloss:0.06274   eval-logloss:0.09932
[33]    train-logloss:0.06033   eval-logloss:0.09768
[34]    train-logloss:0.05776   eval-logloss:0.09561
[35]    train-logloss:0.05554   eval-logloss:0.09314
[36]    train-logloss:0.05273   eval-logloss:0.09018
[37]    train-logloss:0.05060   eval-logloss:0.09011
[38]    train-logloss:0.04883   eval-logloss:0.08770
[39]    train-logloss:0.04712   eval-logloss:0.08543
[40]    train-logloss:0.04541   eval-logloss:0.08446
[41]    train-logloss:0.04399   eval-logloss:0.08420
[42]    train-logloss:0.04230   eval-logloss:0.08241
[43]    train-logloss:0.04089   eval-logloss:0.08018
[44]    train-logloss:0.03943   eval-logloss:0.07878
[45]    train-logloss:0.03796   eval-logloss:0.07780
[46]    train-logloss:0.03663   eval-logloss:0.07726
[47]    train-logloss:0.03556   eval-logloss:0.07564
[48]    train-logloss:0.03443   eval-logloss:0.07554
[49]    train-logloss:0.03328   eval-logloss:0.07502
[50]    train-logloss:0.03216   eval-logloss:0.07441
[51]    train-logloss:0.03097   eval-logloss:0.07309
[52]    train-logloss:0.03000   eval-logloss:0.07255
[53]    train-logloss:0.02906   eval-logloss:0.07268
[54]    train-logloss:0.02830   eval-logloss:0.07233
[55]    train-logloss:0.02751   eval-logloss:0.07149
[56]    train-logloss:0.02678   eval-logloss:0.07177
[57]    train-logloss:0.02613   eval-logloss:0.07064
```

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

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(xgb_model, ax=ax)
```

```
<Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```





LightGBM

- XGBoost보다 학습시간이 훨씬 더 적음
- 메모리 사용량이 낮음
- 한 가지 단점으로 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉬움
- 모든 트리구조의 알고리즘은 level-wise구조이지만, Light GBM은 leaf-wise 구조암

> - 균형 잡힌 트리를 생성하는 이유는 오버 피팅에 보다 더 강한 구조를 가질 수 있다고 알려져 있기 때문임
> - 반대로 균형을 맞추기 위한 시간이 필요하다는 단점이 있음
> - 리프 중심 트리 분할 방식은 트리의 균형을 맞추지 않고, 최대 손실 값(max delta loss)을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리가 생성됨
> - 동일한 leaf를 확장할 때, leaf-wise알고리즘이 level-wise 알고리즘보다 더 많은 loss를 줄일 수 있음

```
!pip install lightgbm==3.3.2
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
train_x, test_x, train_y, test_y = train_test_split(cancer['data'], cancer['target'], test_size=0.2)

lgb_wrapper = LGBMClassifier(n_estimators=400)
evals = [(test_x, test_y)]
lgb_wrapper.fit(train_x, train_y, early_stopping_rounds=100, eval_metric='logloss', eval_set=evals, verbose=True)

preds = lgb_wrapper.predict(test_x)
pred_proba = lgb_wrapper.predict_proba(test_x)[:, 1]

print(preds[:10])
print(pred_proba[:10])

print("LightGBM 정확도 : {:.4f}".format(accuracy_score(preds, test_y)))
```

```
Collecting lightgbm==3.3.2
  Downloading lightgbm-3.3.2-py3-none-manylinux1_x86_64.whl.metadata (15 kB)
Requirement already satisfied: wheel in /usr/local/lib/python3.11/dist-packages (from lightgbm==3.3.2) (0.45.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from lightgbm==3.3.2) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from lightgbm==3.3.2) (1.13.1)
Requirement already satisfied: scikit-learn!=0.22.0 in /usr/local/lib/python3.11/dist-packages (from lightgbm==3.3.2) (1.6.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn!=0.22.0->lightgbm==3.3.2) (1.4.
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn!=0.22.0->lightgbm==3.3.2
Downloading lightgbm-3.3.2-py3-none-manylinux1_x86_64.whl (2.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 31.7 MB/s eta 0:00:00
Installing collected packages: lightgbm
  Attempting uninstall: lightgbm
    Found existing installation: lightgbm 4.5.0
    Uninstalling lightgbm-4.5.0:
      Successfully uninstalled lightgbm-4.5.0
Successfully installed lightgbm-3.3.2
/usr/local/lib/python3.11/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_fin
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/lightgbm/sklearn.py:726: UserWarning: 'early_stopping_rounds' argument is deprecated and will be r
  _log_warning("'early_stopping_rounds' argument is deprecated and will be removed in a future release of LightGBM. "
/usr/local/lib/python3.11/dist-packages/lightgbm/sklearn.py:736: UserWarning: 'verbose' argument is deprecated and will be removed in a fu
  _log_warning("'verbose' argument is deprecated and will be removed in a future release of LightGBM. "
[1]     valid_0's binary_logloss: 0.57559
[2]     valid_0's binary_logloss: 0.519952
[3]     valid_0's binary_logloss: 0.466354
[4]     valid_0's binary_logloss: 0.423217
[5]     valid_0's binary_logloss: 0.383642
[6]     valid_0's binary_logloss: 0.351098
[7]     valid_0's binary_logloss: 0.320676
[8]     valid_0's binary_logloss: 0.298528
[9]     valid_0's binary_logloss: 0.275598
[10]    valid_0's binary_logloss: 0.254882
[11]    valid_0's binary_logloss: 0.238015
[12]    valid_0's binary_logloss: 0.224463
```

```
[13]    valid_0's binary_logloss: 0.209962
[14]    valid_0's binary_logloss: 0.200112
[15]    valid_0's binary_logloss: 0.188014
[16]    valid_0's binary_logloss: 0.178343
[17]    valid_0's binary_logloss: 0.168179
[18]    valid_0's binary_logloss: 0.161907
[19]    valid_0's binary_logloss: 0.153884
[20]    valid_0's binary_logloss: 0.149566
[21]    valid_0's binary_logloss: 0.142302
[22]    valid_0's binary_logloss: 0.138866
[23]    valid_0's binary_logloss: 0.134435
[24]    valid_0's binary_logloss: 0.128657
[25]    valid_0's binary_logloss: 0.12351
[26]    valid_0's binary_logloss: 0.121807
[27]    valid_0's binary_logloss: 0.121009
[28]    valid_0's binary_logloss: 0.118835
```

```
from lightgbm import plot_importance
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgb_wrapper, ax=ax)
```

```
<Axes: title={'center': 'Feature importance'}, xlabel='Feature importance', ylabel='Features'>
```