04_머신러닝 분류

• 분류(Classification)

클래스 값
> 머신러닝 지도학습의 한 유형으로 레이블 값을 예측한다.

> 다양한 머신러닝 알고리즘으로 구현할 수 있다.

- 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
- 서로 다른 머신러닝 알고리즘을 결합한 앙상블(Ensemble)
- 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀(Logistic Regression) 분류
- 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 서포트 벡터 머신(Support Vector Machine)
- 거리가 가까운 데이터를 참조하는 최근접 이웃(K-Nearest Neighbor)
- 조건부 확률을 계산하는 방법을 적용한 나이브 베이즈(Naive Bayes)

머신러닝 분류

- 결정 트리 의사결정나무

```
from sklearn.tree import DecisionTreeClassifier
```

> ML 알고리즘 중 직관적으로 이해하기 쉬운 알고리즘

> 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리(Tree) 기반의 분류 규칙을 만든다.

> if/else 기반으로 데이터를 분류하여 답을 찾아낸다.

> 최대한 균일한 데이터 세트를 구성할 수 있도록 분할하는 것이 필요하다.

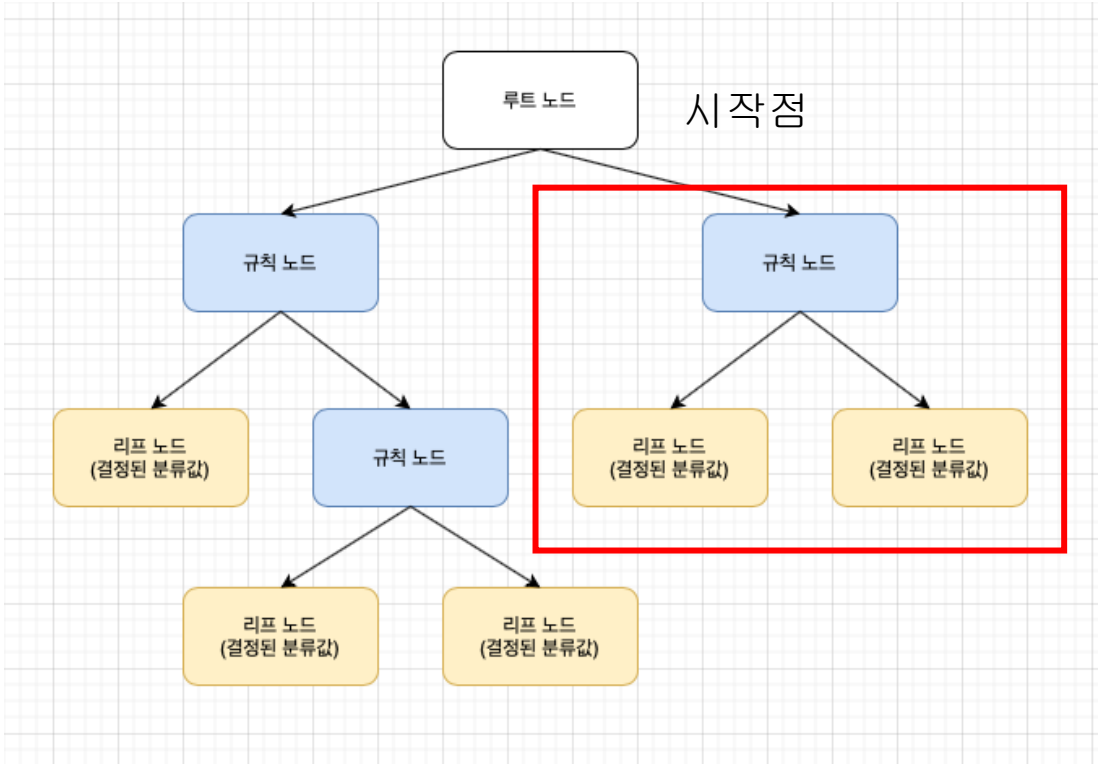
  - 균일한 데이터 : 정보를 쉽게 예측할 수 있는 데이터

> 정보 이득과 지니 계수를 이용하여 균일한 데이터인지 측정 가능하다.

+ 무작위성

• 결정 트리

> 규칙 노드 : 규칙 조건이 되는 노드로 if/else로 데이터를 구분한다.
> 리프 노드 : 결정된 클래스 값이다.
> 서브 트리 : 새로운 규칙 조건마다 생성된 구조



• 결정 트리

> 정보 이득

- 엔트로피라는 개념을 기반으로 한다.
- 엔트로피에서 전체 자식 노드의 엔트로피를 뺀 값으로 정의
- 즉, 1-엔트로피 지수가 클수록 정보 이득이 높다.
- 결정 트리는 정보 이득 지수가 최대값이 되는 분기를 찾는다.

엔트로피가 작을 수록

정보이득이 클수록

$$H(B) = -\sum_{i=1}^{2} p_i(y \mid B) \log p_i(y \mid B)$$

$$\mathrm{IG}(A,\{B,C\}) = H(A) - H(\{B,C\}) = H(A) - \frac{|B|}{|B|+|C|} H(B) - \frac{|C|}{|B|+|C|} H(C)$$

> 지니 계수

- 클래스가 잘못 분류될 확률의 가중 평균으로 정의한다.
- 0이 가장 평등하고 1로 갈수록 불평등하다.
- 결정 트리는 지니 계수가 최소값이 되는 분기를 찾는다.

작을 수록 균등해진다.

$$G_A = \sum_{i=1}^{2} p_i(1-p_i) = \sum_{i=1}^{2} (p_i - p_i^{\,2}) = 1 - \sum_{i=1}^{2} p_i^{\,2}$$

$$\mathrm{Gini\_impurity}(k, t_k) = \frac{|B|}{|B|+|C|} G_B + \frac{|C|}{|B|+|C|} G_C$$

• 결정 트리

> **결정 트리 분할 구조**

1. 데이터 집합의 모든 아이템이 같은 분류에 속하는지 확인
2-1. 리프 노드로 만들어서 분류 결정
2-2. 데이터를 분할하는 데 가장 좋은 속성과 분할 기준을 찾음
(정보 이득 or 지니 계수를 이용한다)
3. 해당 속성과 분할 기준으로 데이터 분할하여 Branch 노드 생성
4. 처음으로 돌아가 모든 데이터 집합의 분류가 결정될 때까지 수행

성장.

> **결정 트리의 장단점**

- 쉽고 직관적이라는 장점
- 데이터 균일도만 신경쓰면 피쳐의 스케일링이나 정규화 같은 전처리의 영향도가 크기 않다.
- 단점으로는 과적합으로 인해 정확도가 떨어질 수 있다.

-> 가지치기: 파라미터를 이용해서

스케일링이나 정규화 영향이 없으므로 할 필요 없음

• 결정 트리

> **결정 트리 파라미터**

- max_depth
  - 깊이의 상한선 기본값: None
- min_samples_split
  - 노드에서 분기를 진행하는 최소한의 샘플 숫자
- min_samples_leaf
  - 리프 노드에 있을 샘플 개수의 최솟값(과적합을 막고자 사용)
- max_features
  - 각 노드에서 분기를 확인할 피쳐 수
- max_leaf_nodes
  - 말단 노드(Leaf)의 최대 개수
- criterion
  - 분기 규칙 선택('gini' : 지니불순도 최소화, 'entropy' : 정보이득 최대화)

• 결정 트리 실습

> 와인 데이터 실습

```
from sklearn.datasets import load_wine
import pandas as pd

wine = load_wine()
df = pd.DataFrame(wine.data, columns=wine.feature_names)
df['class'] = wine.target
print(wine.target_names)
df.head()
```

Out :

```
['class_0' 'class_1' 'class_2']
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04

• 결정 트리 실습

> 학습 데이터 준비

```
from sklearn.model_selection import train_test_split

X_data = df.iloc[:, :-1]
y_data = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.2 , random_state= 156)

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

Out : (142, 13) (36, 13)
(142,) (36,)

• 결정 트리 실습

> 학습 / 예측 / 평가

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dt_clf = DecisionTreeClassifier(random_state=156)
dt_clf.fit(X_train , y_train)
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print('결정 트리 예측 정확도: {0:.4f}'.format(accuracy))
```

Out : 결정 트리 예측 정확도: 0.9444

• 결정 트리 실습

> GridSearchCV

```
from sklearn.model_selection import GridSearchCV

parameters = {'criterion':['gini', 'entropy'],
              'max_depth':[None, 2, 3, 5, 7],
              'min_samples_split':[2,3,5,7],
              'min_samples_leaf':[1,3,5,7]}
grid_dt = GridSearchCV(dt_clf, param_grid=parameters, cv=5)
grid_dt.fit(X_train, y_train)

scores_df = pd.DataFrame(grid_dt.cv_results_)
scores_df[['params', 'mean_test_score', 'rank_test_score']]
```

• 결정 트리 실습

> GridSearchCV

Out :

	params	mean_test_score	rank_test_score
0	{'criterion': 'gini', 'max_depth': None, 'min_...	0.851478	101
1	{'criterion': 'gini', 'max_depth': None, 'min_...	0.851478	101
2	{'criterion': 'gini', 'max_depth': None, 'min_...	0.858621	68
3	{'criterion': 'gini', 'max_depth': None, 'min_...	0.865517	65
4	{'criterion': 'gini', 'max_depth': None, 'min_...	0.858621	68
...	...	...	...
155	{'criterion': 'entropy', 'max_depth': 7, 'min_...	0.858621	68
156	{'criterion': 'entropy', 'max_depth': 7, 'min_...	0.865764	33
157	{'criterion': 'entropy', 'max_depth': 7, 'min_...	0.865764	33
158	{'criterion': 'entropy', 'max_depth': 7, 'min_...	0.865764	33
159	{'criterion': 'entropy', 'max_depth': 7, 'min_...	0.865764	33

• 결정 트리 실습

> GridSearchCV

```
print('GridSearchCV 최적 파라미터:', grid_dt.best_params_)
print(f'GridSearchCV 최고 정확도: {grid_dt.best_score_:.4f}')
```

Out :
```
GridSearchCV 최적 파라미터: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5}
GridSearchCV 최고 정확도: 0.8936
```

```
model = grid_dt.best_estimator_
pred = model.predict(X_test)
accuracy_score(y_test, pred)
```

Out : 1.0

• 결정 트리 실습

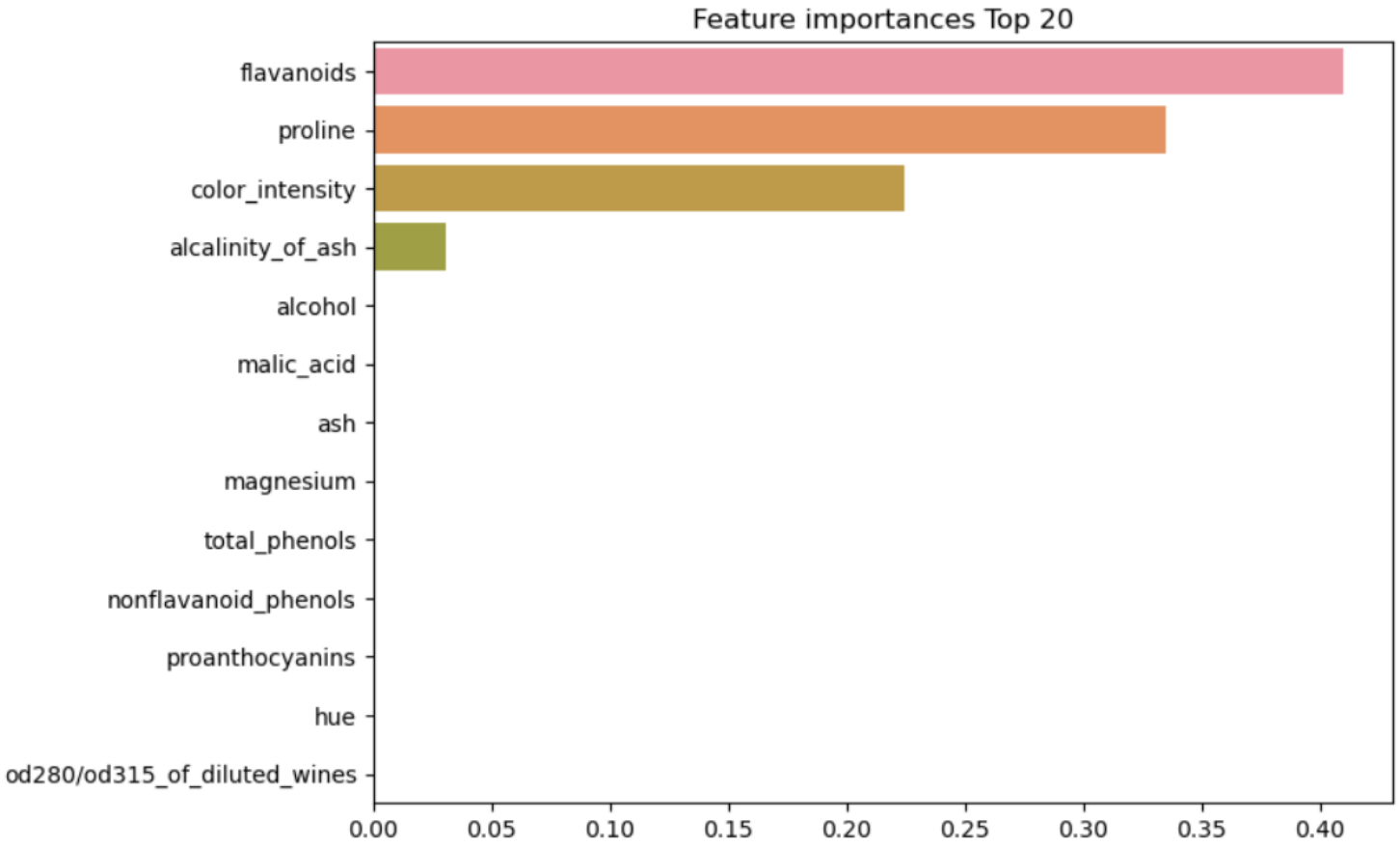
> 피쳐 중요도 그래프

```
import seaborn as sns

ftr_importances_values = model.feature_importances_
ftr_importances = pd.Series(ftr_importances_values,index=X_train.columns)
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```

• 결정 트리 실습

> 피쳐 중요도 그래프

Out :



• 결정 트리 실습

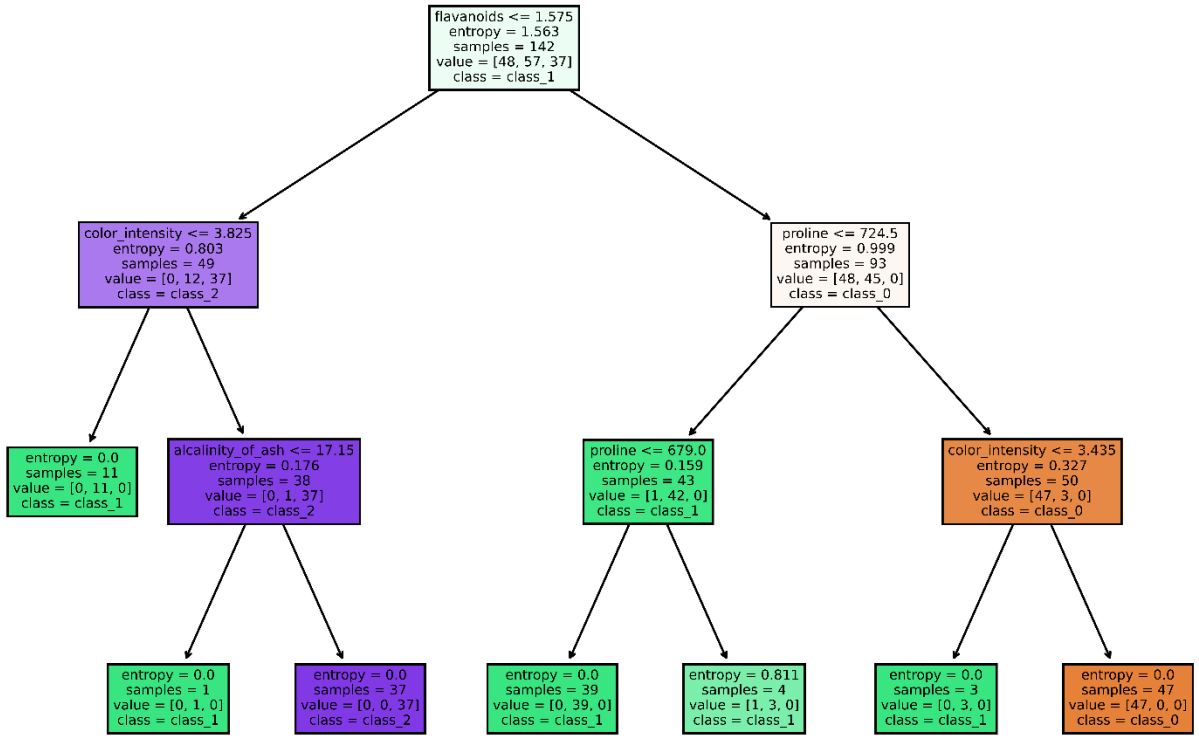
> 트리 구조 그래프

```
from sklearn.tree import plot_tree


plt.figure(figsize=(10,7), dpi=1200)
plot_tree(model, filled=True,
          feature_names=wine.feature_names,
          class_names=list(wine.target_names))
```

• 결정 트리 실습

> 트리 구조 그래프

Out :



• 앙상블 학습

단독 알고니즘은 없고 만들거나 가져와서 사용

> **여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 예측을 도출하는 법**

> **전통적인 세 가지 방법**

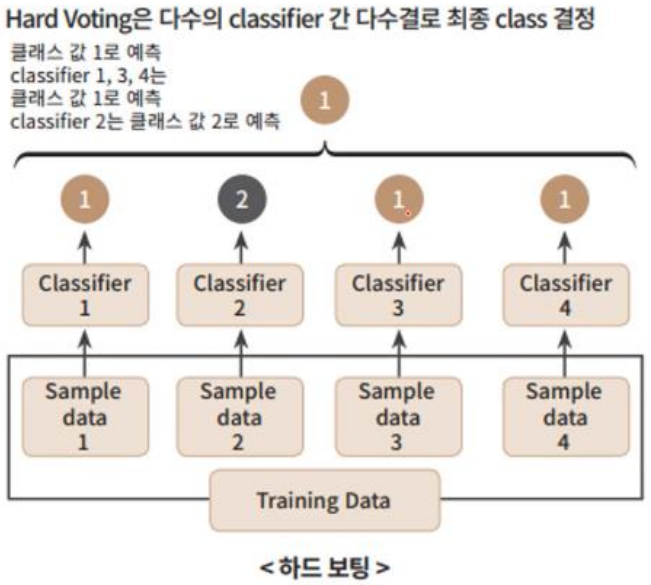
- 보팅(Voting) : 한 데이터셋에서 다른 알고리즘을 사용
- 배깅(Bagging) : 한 분류기로 서로 다른 데이터 샘플을 학습 LandomForest → 다수결 방식
- 부스팅(Boosting) : 순차적으로 학습-예측하면서 오류를 개선해가며 가중치를 주는 방식
  한 데이터셋, 한 분류기 ==> 오류를 개선하며 가중치 조정하며 반복 과정
  딥러닝의 학습방법 XGBoost

스태킹(Stacking) 성능이 비슷한 여러 모델로 작업하여 예측한 값을 최종 모델을 사용

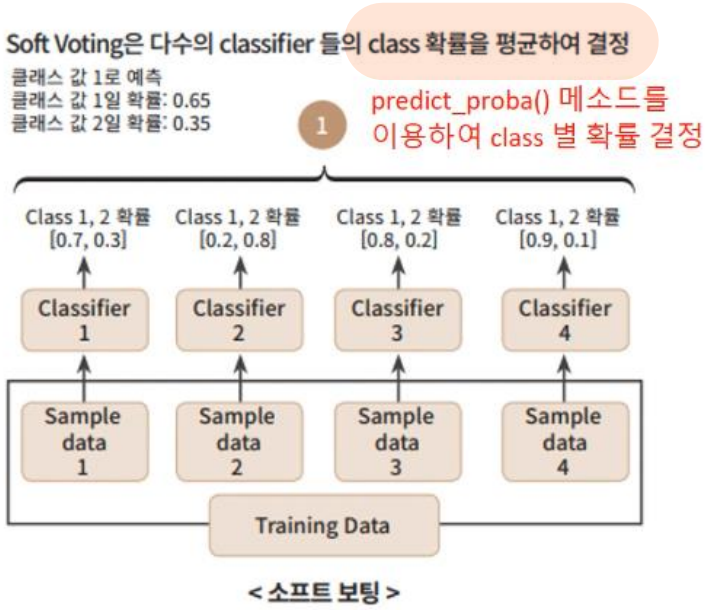
• 앙상블 학습

> 보팅 유형

- 하드 보팅 가장 많이 나온 안/ 다수결 채택
- 소프트 보팅 모두 안의 평균



<하드 보팅>



<소프트 보팅>

많이 사용

• 앙상블 학습

> 보팅 분류기(Voting Classifier)

```
import pandas as pd
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
data_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
data_df.head(3)
```

• 앙상블 학습

> 보팅 분류기(Voting Classifier)

Out :

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...

worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
25.38	17.33	184.6	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
24.99	23.41	158.8	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
23.57	25.53	152.5	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758

• 앙상블 학습

> 보팅 분류기(Voting Classifier)

```
lr_clf = LogisticRegression(solver='liblinear')
dt_clf = DecisionTreeClassifier(random_state=0)
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),
                                       ('DT',dt_clf)] , voting='soft' )

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2 , random_state= 0 )
vo_clf.fit(X_train , y_train)
pred = vo_clf.predict(X_test)
print(f'Voting 분류기 정확도: {accuracy_score(y_test,pred):.4f}')
```

Out : Voting 분류기 정확도: 0.9123

• 앙상블 학습

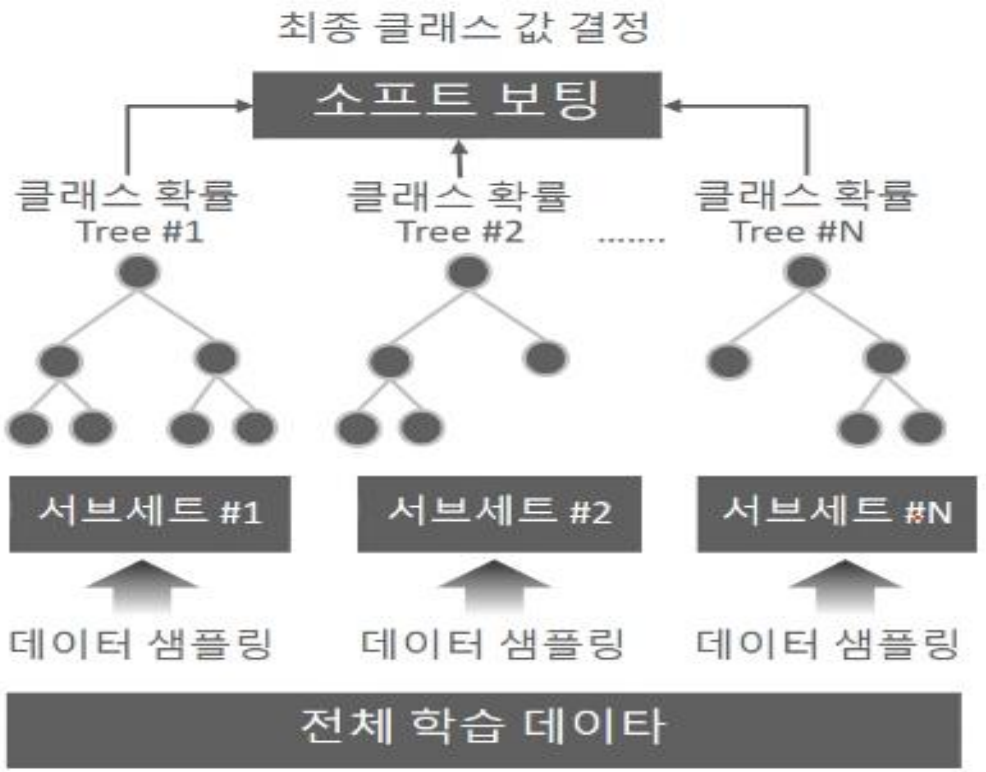
> 보팅 분류기(Voting Classifier)

```
# 각 모델의 정확도 계산
classifiers = [lr_clf, dt_clf]
for clf in classifiers:
    clf.fit(X_train , y_train)
    pred = clf.predict(X_test)
    class_name= clf.__class__.__name__
    print(f'{class_name} 정확도: {accuracy_score(y_test, pred):.4f}')
```

Out : LogisticRegression 정확도: 0.9561
DecisionTreeClassifier 정확도: 0.9123

• 앙상블 학습

> **랜덤 포레스트**
- 배깅의 대표적인 알고리즘
- 결정 트리 기반의 앙상블 학습



1개의 알고리즘 사용
데이터 중복가능/랜덤

• 앙상블 학습

> **부트스트래핑**

- 여러 개의 데이터 세트를 중첩되게 분리하는 방식 중복

• 앙상블 학습

> 랜덤포레스트 실습

```
from sklearn.ensemble import RandomForestClassifier


cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2 , random_state= 0)
rf_clf = RandomForestClassifier(random_state=0)
rf_clf.fit(X_train , y_train)
pred = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print(f'랜덤 포레스트 정확도: {accuracy:.4f}')
```

Out : 랜덤 포레스트 정확도: 0.9649

• 앙상블 학습

> 랜덤포레스트 하이퍼 파라미터 및 튜닝

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators':[100],
          'max_depth' : [6, 8, 10, 12],
          'min_samples_leaf' : [8, 12, 18 ],
          'min_samples_split' : [8, 16, 20]}

rf_clf = RandomForestClassifier(random_state=0)
grid_cv = GridSearchCV(rf_clf , param_grid=params , cv=3, n_jobs=-1, verbose=True)
grid_cv.fit(X_train , y_train)
```

• 앙상블 학습

> 랜덤포레스트 하이퍼 파라미터 및 튜닝

```
print('GridSearchCV 최적 파라미터:', grid_cv.best_params_)
print(f'GridSearchCV 최고 정확도: {grid_cv.best_score_:.4f}')
```

Out :
```
GridSearchCV 최적 파라미터: {'max_depth': 6, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 100}
GridSearchCV 최고 정확도: 0.9385
```

```
model = grid_cv.best_estimator_
pred = model.predict(X_test)
accuracy_score(y_test, pred)
```

Out : 0.9649122807017544

• 앙상블 학습

> 피쳐 중요도 그래프

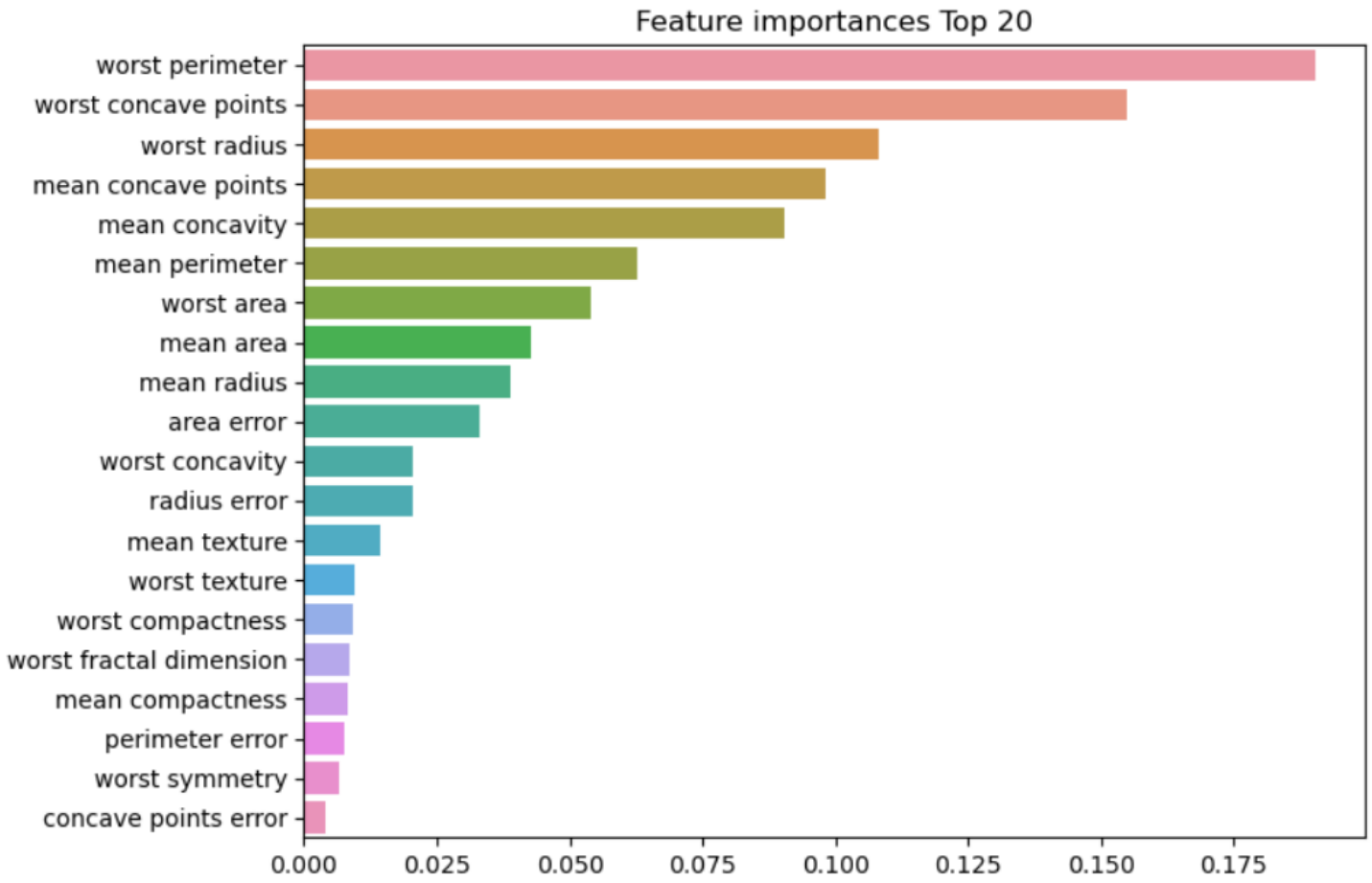
```
import matplotlib.pyplot as plt
import seaborn as sns

ftr_importances_values = model.feature_importances_
ftr_importances = pd.Series(ftr_importances_values, index=cancer.feature_names )
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
plt.draw()
```

머신러닝 분류

- 앙상블 학습

> 피쳐 중요도 그래프

Out :



• 앙상블 학습

> GBM(Gradient Boosting Machine)

- 경사 하강법(Gradient Descent)를 사용하여 가중치 업데이트를 하는 방식

미분: 딥러닝에서 잘 쓰임

> Gradient Descent

- 오류 값: 실제 값 - 예측 값
- 예측 함수를 F(x) 라고 하면 오류식 h(x)=y−F(x)
- 이 오류식 h(x)=y−F(x)를 최소화 하는 방향성을 가지고
- 반복적으로 가중치 값을 업데이트 하는 것이 Gradient Descent

h(x): 손실함수- 엔트로피,확률로 계산

경사하강법 익히기
로그 미분?

2-binary
3-category

> GBM 하이퍼 파라미터

- loss : 경사 하강법에서 사용할 비용 함수 지정
- learning_rate : 학습을 진행할 때마다 적용하는 학습률, 0.1
- n_estimators : weak learner의 개수
- subsample : weak learner가 학습에 사용하는 데이터 샘플링의 비율
- 나머진 의사결정나무의 파라미터와 같다.

둘이 반비례로 설정하여 과적합 조정

회귀: MSE

평균제곱 오차(MSE)

• 앙상블 학습

> GBM(Gradient Boosting Machine)

```
from sklearn.ensemble import GradientBoostingClassifier


cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2 , random_state= 0)
gb_clf = GradientBoostingClassifier(random_state=0)
gb_clf.fit(X_train , y_train)
gb_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)
print(f'GBM 정확도: {gb_accuracy:.4f}')
```

Out : GBM 정확도: 0.9649

• XGBoost(eXtra Gradient Boost)

> 트리 기반 앙상블 학습에서 각광받는 알고리즘 중 하나

- 병렬 CPU 환경에서 병렬 학습이 가능

> 주요 장점

- 예측 성능이 뛰어남
- GBM 대비 빠른 수행 시간
- Regularization(과적합 규제) 선형에 적용되는 규제 가능
- Tree pruning(나무 가지치기) : 이득이 없는 분할을 가지치기
- 자체 내장 교차 검증
- 결손값 자체 처리

• XGBoost(eXtra Gradient Boost)

> **주요 파라미터**

- eda : learning rate와 같은 파라미터
- num_boost_rounds : weak learner의 개수
- min_child_weight : 분할 조절
- gamma : 리프 노드를 나눌지 결정하는 최소 값
- max_depth : 트리의 깊이
- objective : 손실함수 정의, (binary:logistic, multi:softmax)
- eval_metric : 검증 유형

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
import xgboost as xgb
from xgboost import plot_importance
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
dataset = load_breast_cancer()

cancer_df = pd.DataFrame(data=dataset.data, columns=dataset.feature_names)
cancer_df['target'] = dataset.target
X_features = cancer_df.iloc[:, :-1]
y_label = cancer_df.iloc[:, -1]

X_train, X_test, y_train, y_test=train_test_split(X_features, y_label,
                                                  test_size=0.2, random_state=156 )
X_tr, X_val, y_tr, y_val= train_test_split(X_train, y_train,
                                           test_size=0.1, random_state=156 )
```

최종평가때 사용

딥러닝도 이런식으로

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
dtr = xgb.DMatrix(data=X_tr, label=y_tr)
dval = xgb.DMatrix(data=X_val, label=y_val)
dtest = xgb.DMatrix(data=X_test , label=y_test)

params = {'max_depth':3,'eta': 0.05,'objective':'binary:logistic','eval_metric':'logloss'}
num_rounds = 400
```

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
eval_list = [(dtr,'train'),(dval,'eval')]
# 또는 eval_list = [(dval,'eval')]

xgb_model = xgb.train(params = params , dtrain=dtr ,
                      num_boost_round=num_rounds ,
                      early_stopping_rounds=50, evals=eval_list )
```

Out :

```
[0]    train-logloss:0.65016    eval-logloss:0.66183
[1]    train-logloss:0.61131    eval-logloss:0.63609
[2]    train-logloss:0.57563    eval-logloss:0.61144
[3]    train-logloss:0.54310    eval-logloss:0.59204
```

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
pred_probs = xgb_model.predict(dtest)
print('predict( ) 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨')
print(np.round(pred_probs[:10],3))


preds = [ 1 if x > 0.5 else 0 for x in pred_probs ]
print('예측값 10개만 표시:',preds[:10])
```

Out :

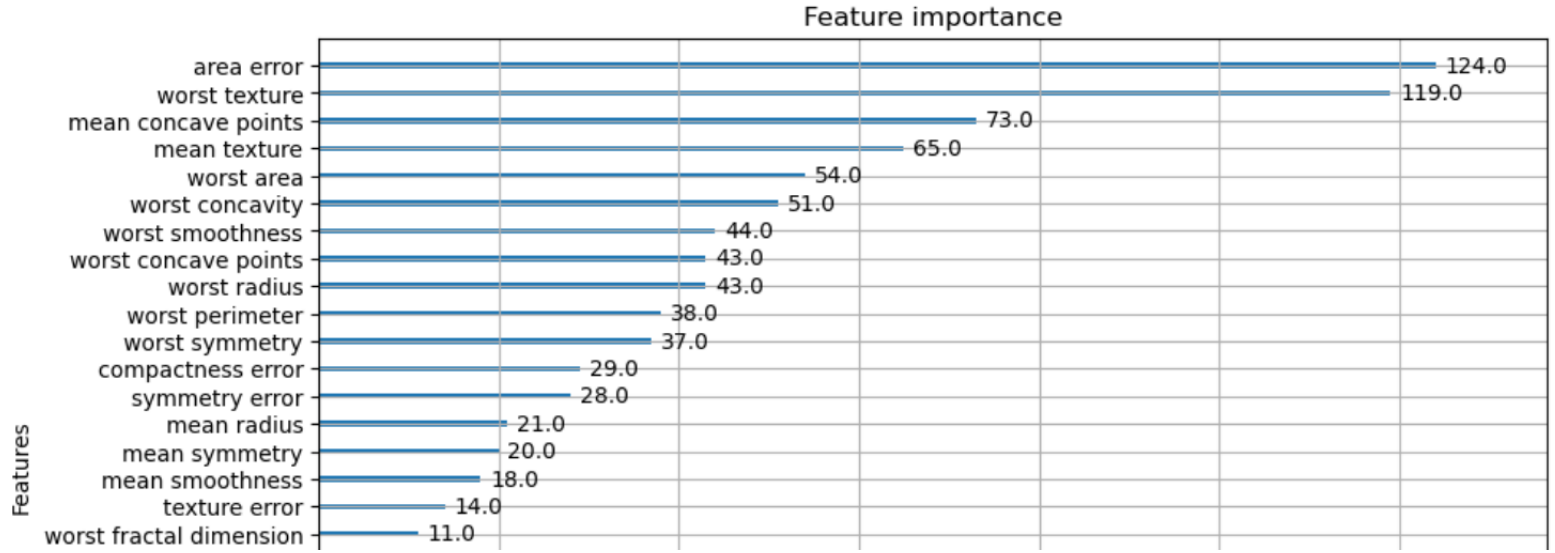
```
predict( ) 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨
[0.938 0.004 0.75  0.049 0.98  1.    0.999 0.999 0.998 0.001]
예측값 10개만 표시: [1, 0, 1, 0, 1, 1, 1, 1, 1, 0]
```

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10, 7))
plot_importance(xgb_model, ax=ax)
```

Out :



• XGBoost(eXtra Gradient Boost)

> 사이킷런 래퍼 XGBoost의 개요 및 적용

```
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)
preds = xgb_wrapper.predict(X_test)
pred_proba = xgb_wrapper.predict_proba(X_test)[:, 1]
```

학습횟수

학습률

X_val,y_val

**규제강도를 조정하는 파라미터 reg_alpha= , reg_lamdba= 도 있음

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

```
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print(f'정확도: {accuracy:.4f}, 정밀도: {precision:.4f}, 재현율: {recall:.4f}')
    print(f'F1: {f1:.4f}, AUC:{roc_auc:.4f}')
```

• XGBoost(eXtra Gradient Boost)

> XGBoost 실습

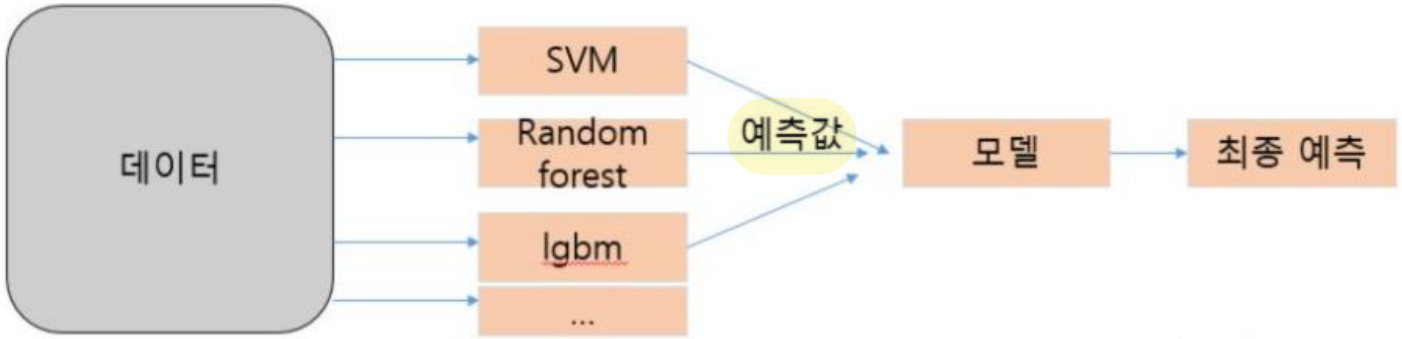
```
get_clf_eval(y_test, ws100_preds, ws100_pred_proba)
```

Out :

```
오차 행렬
[[46  1]
 [ 2 65]]
정확도: 0.9737, 정밀도: 0.9848, 재현율: 0.9701
F1: 0.9774, AUC:0.9994
```

• 스태킹 앙상블

> 기본구조



> 성능이 비슷한 여러 모델로 예측한 값을 최종 모델에 적용하여 예측을 수행

voting과 유사함

• 스태킹 앙상블

> 기본 스태킹 모델

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression


# 4개의 모델(KNN, RF, Ada, DT)를 통해 개별 예측을 하고
# 최종으로 Logistic 모델을 사용하여 예측 예정
```

• 스태킹 앙상블

> 기본 스태킹 모델

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer_data = load_breast_cancer()
X_data = cancer_data.data
y_label = cancer_data.target
X_train , X_test , y_train , y_test = train_test_split(X_data, y_label,
                                                       test_size=0.2 , random_state=0)
```

• 스태킹 앙상블

> 기본 스태킹 모델

```
# 개별 ML 모델을 위한 Classifier 생성.
knn_clf =KNeighborsClassifier(n_neighbors=4)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=0)
dt_clf = DecisionTreeClassifier(random_state=0)
ada_clf = AdaBoostClassifier(n_estimators=100)

# 최종 Stacking 모델을 위한 Classifier생성.
lr_final = LogisticRegression(solver='liblinear')
```

• 스태킹 앙상블

> 기본 스태킹 모델

```
# 개별 모델들을 학습.
knn_clf.fit(X_train, y_train)
rf_clf.fit(X_train , y_train)
dt_clf.fit(X_train , y_train)
ada_clf.fit(X_train, y_train)

# 학습된 개별 모델들이 각자 반환하는 학습 데이터 셋을 생성
knn_train = knn_clf.predict(X_train)
rf_train = rf_clf.predict(X_train)
dt_train = dt_clf.predict(X_train)
ada_train = ada_clf.predict(X_train)
```

• 스태킹 앙상블

> 기본 스태킹 모델

```
# 학습된 개별 모델들이 각자 반환하는 테스트 데이터 셋을 생성
knn_test = knn_clf.predict(X_test)
rf_test = rf_clf.predict(X_test)
dt_test = dt_clf.predict(X_test)
ada_test = ada_clf.predict(X_test)

# 학습 데이터와 테스트 데이터로 합치기
pred_train = np.vstack([knn_train, rf_train, dt_train, ada_train]).T
pred_test = np.vstack([knn_test, rf_test, dt_test, ada_test]).T
```

• 스태킹 앙상블

> 기본 스태킹 모델

```
lr_final.fit(pred_train, y_train)
final = lr_final.predict(pred_test)


print(f'최종 모델의 정확도: {accuracy_score(y_test , final):.4f}')
```

Out : 최종 모델의 정확도: 0.9737

• 스태킹 앙상블

> 스태킹 실습

```
from sklearn.ensemble import StackingClassifier

stack = StackingClassifier([('knn', knn_clf),
                            ('rf', rf_clf),
                            ('dt', dt_clf),
                            ('ada', ada_clf)],
                            final_estimator = lr_clf)
```

• 스태킹 앙상블

> 스태킹 실습

```
stack.fit(X_train, y_train)


pred = stack.predict(X_test)
accuracy_score(y_test, pred)
```

Out : 0.9649122807017544