파이썬 머신러닝 완벽 가이드 - 분류

목차

1. 분류(Classification)의 개요

1. 분류에서의 다양한 머신러닝 알고리즘
    - 베이즈 통계와 생성 모델에 기반한 **나이브 베이즈(Naive Bayes)**
    - 독립변수와 종속변수의 선형 관계성에 기반한 **로지스틱 회귀(Logistic Regression)**
    - 데이터 균일도에 따른 규칙 기반의 **결정 트리(Decision Tree)**
    - 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 **서포트 벡터 머신(Support Vector Machine)**
    - 근접 거리를 기준으로 하는 **최소 근접 알고리즘(Nearest Neighbor)**
    - 심층 연결 기반의 **신경망(Neural Network)**
    - 서로 다른(또는 같은) 머신러닝 알고리즘을 결합한 **앙상블(Ensemble)**

2. 결정 트리

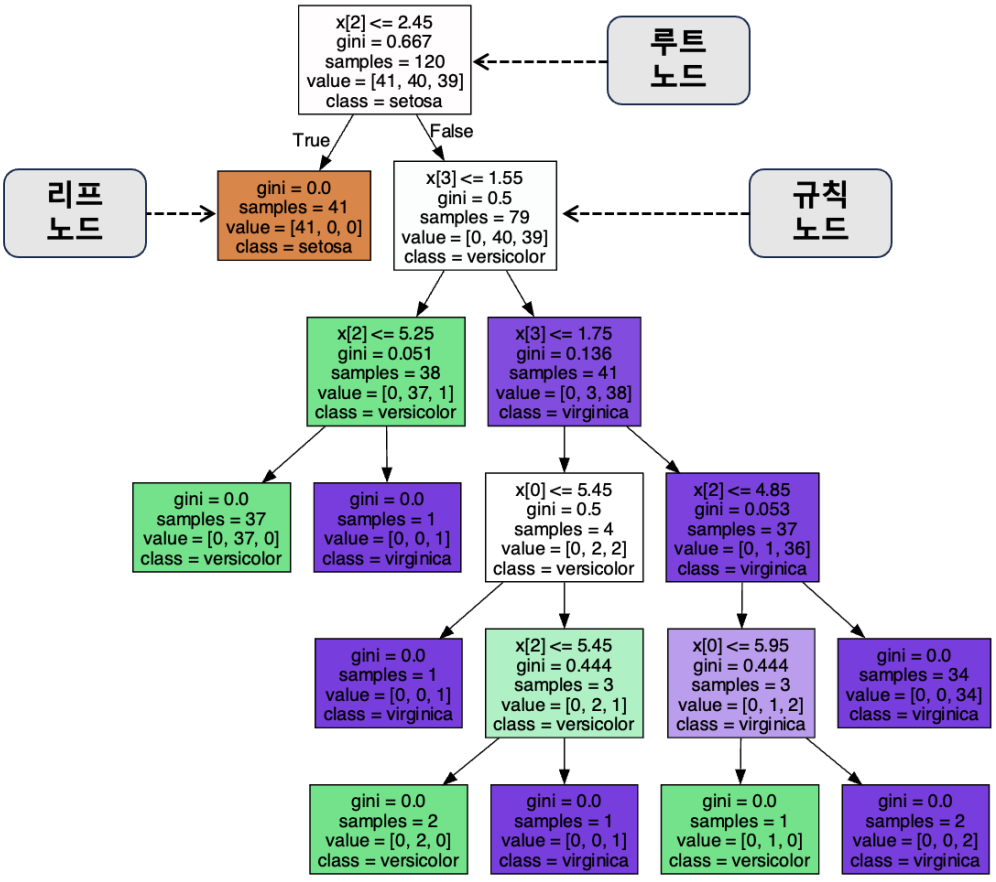
▼ 개념

a. 결정트리

- 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것.
- 일반적으로 규칙을 가장 쉽게 표현하는 방법은 if/else 기반으로 나타내는 것.
- 많은 규칙이 있다는 것은 분류를 결정하는 방식이 복잡해진다는 얘기이고, 이는 과적합으로 이어지기 쉬움
- 그러므로 트리의 깊이가 깊어질수록 결정 트리의 예측 성능이 저하될 가능성이 높음.

b. 노드

- 규칙 노드 : 규칙 조건이 되는 것
- 리프 노드 : 결정된 클래스 값



c. 정보의 균일도를 측정하는 방법

- 정보 이득 지수 = (1 - 엔트로피 지수)

        - 엔트로피 지수 : 주어진 데이터 집합의 혼잡도 / 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮다.
    - 지니 계수
        - 불평등의 정도를 나타내는 통계학적 지수
        - 지니 계수가 낮을수록 균일도가 높은 것으로 지니 계수가 낮은 속성을 기준으로 분할함.

▼ 특징

a. 장단점

- 장점
    - 쉽고 직관적임.
    - 특별한 경우를 제외하고 각 피처의 스케일링과 정규화 같은 전처리 작업이 필요 없음.
- 단점 : 과적합이 발생하여 정확도가 떨어짐.

b. 파라미터

파라미터 명	**설명**
min_samples_split	- default = 2 - 노드를 분할하기 위한 최소한의 샘플 데이터 수
min_samples_leaf	- 분할이 될 경우 왼쪽과 오른쪽의 브랜치 노드에서 가져야 할 최소한의 샘플 데이터 수
max_features	- 최적의 분할을 위해 고려할 최대 피처 개수 - default = None(전체 피처 선정) - “sqrt”는 √(전체 피처 개수) - “log”는 log_2〖전체 피처 개수〗
max_depth	- 트리의 최대 깊이를 규정 - default = None - 깊이가 깊어지면 min_samples_split 설정대로 최대 분할하여 과적합할 수 있으므로 적절한 값으로 제어 필요
max_leaf_nodes	- 말단 노드의 최대 개수

c. 과적합

▼ 예제코드 - 붓꽃 데이터

1. 결정 트리 예제 - 붓꽃 데이터

```
# 라이브러리
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

1.1 데이터 불러오기

```
# 데이터 불러오기
from sklearn.datasets import load_iris

iris_data = load_iris()
iris_data

# 데이터프레임으로 변환하기
iris_df = pd.DataFrame(data=iris_data.data, columns=iri
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

1.2 학습,테스트 데이터 세트 분리

```
# 학습, 테스트 데이터 세트로 분리
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iri

                                                    tes
                                                    ran

print(X_train.shape, X_test.shape, y_train.shape, y_tes
```

```
(120, 4) (30, 4) (120,) (30,)
```

1.3 Decision Tree 생성 및 학습

```
# Decision Tree 생성
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=156)

# Decision Tree 학습
clf.fit(X_train,y_train)
```

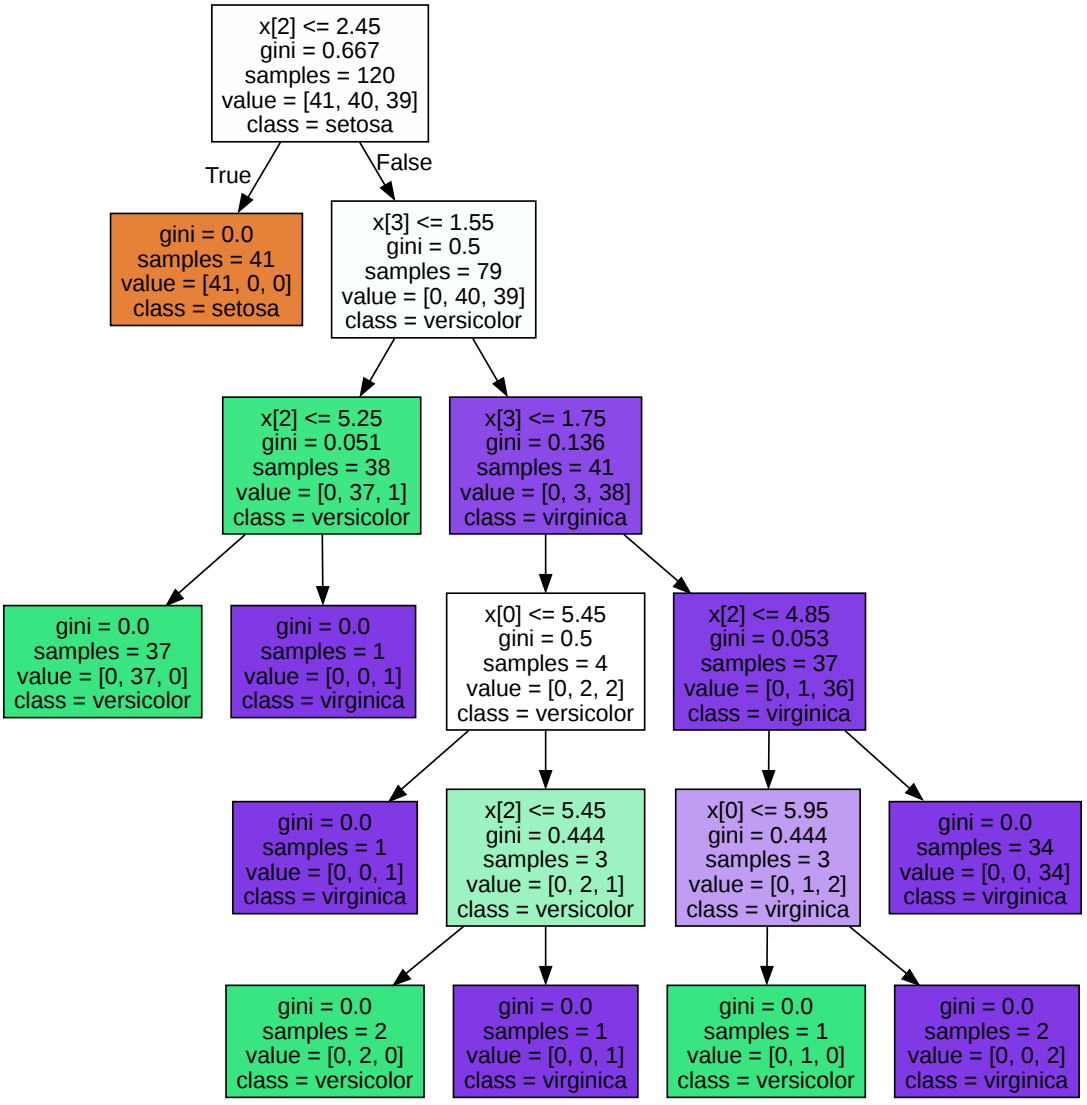
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=156)
```

1.4 시각화

```
# Graphviz 시각화 파일 저장
from sklearn.tree import export_graphviz

export_graphviz(clf,
```

```
                    out_file="tree.dot",
                    class_names=iris_data.target_names,
                    impurity=True,
                    filled=True)            # 노드의 시각화 시
```

```
# Graphviz 시각화 파일 읽기
import graphviz

with open("./tree.dot") as f:
    graph = f.read()

graphviz.Source(graph)
```



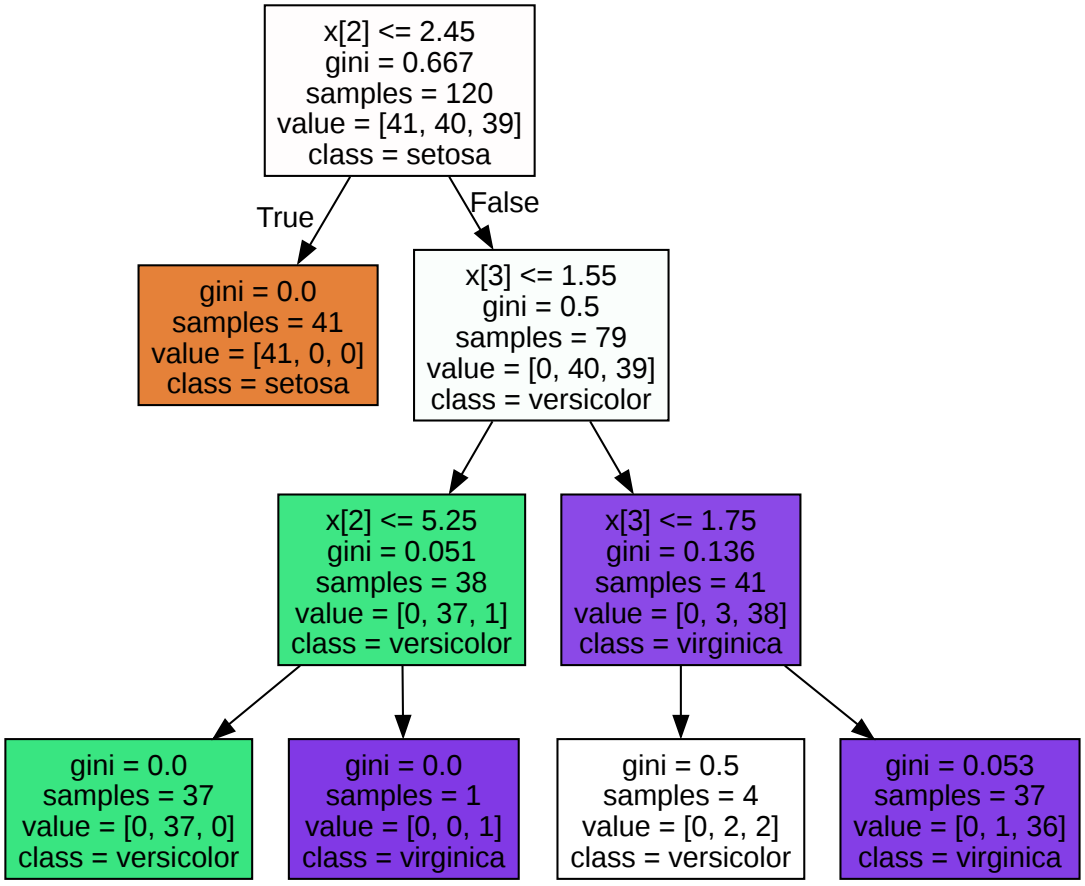
1.5 분석 내용

- 루트노드
  - “samples = 120” : 전체 데이터가 120개
  - “value = [41,40,39]” : Setosa(41개), Versicolor(40개), Virginica(39개)
- 첫번째 리프 노드
  - “samples = 41” : 샘플 데이터가 41개
  - “class = setosa” : 샘플 데이터 모두가 Setosa이므로 예측 클래스로 결정
- 규칙 노드 1
  - 상위 노드에서 “Petal width(mm) ≤ 1.55” 규칙이 True인 규칙 노드
  - “samples = 38” : 샘플 데이터가 38개
  - “value = [0, 37, 1]” : Versicolor(37개), Virginica(1개)
- 규칙 노드 2
  - 상위 노드에서 “Petal width(mm) ≤ 1.55” 규칙이 False인 규칙 노드
  - “samples = 41” : 샘플 데이터가 41개
  - “value = [0, 3, 38]” : Versicolor(3개), Virginica(38개)

2. 파라미터 적용

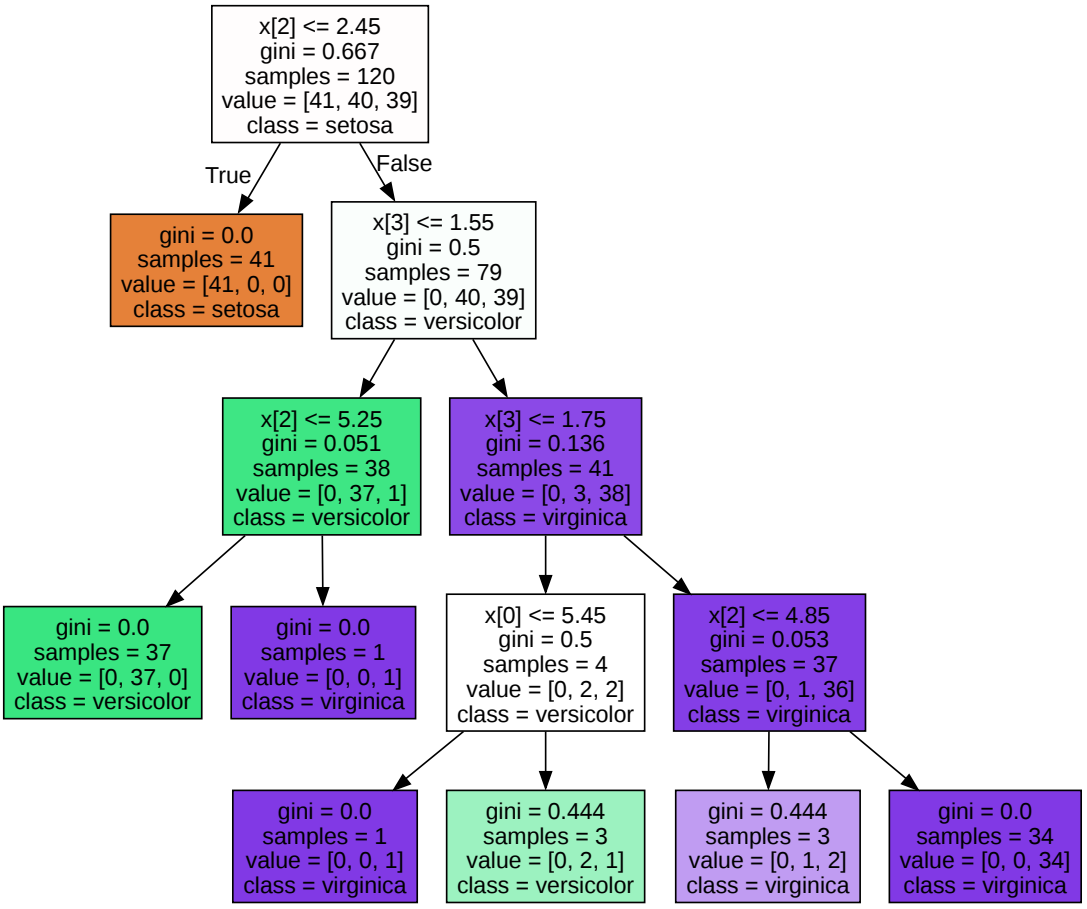
2.1 max_depth = 3

```
# Decision Tree 생성
from sklearn.tree import DecisionTreeClassifier

clf_para1 = DecisionTreeClassifier(random_state=156,
                                   max_depth=3)

# Decision Tree 학습
clf_para1.fit(X_train,y_train)
```



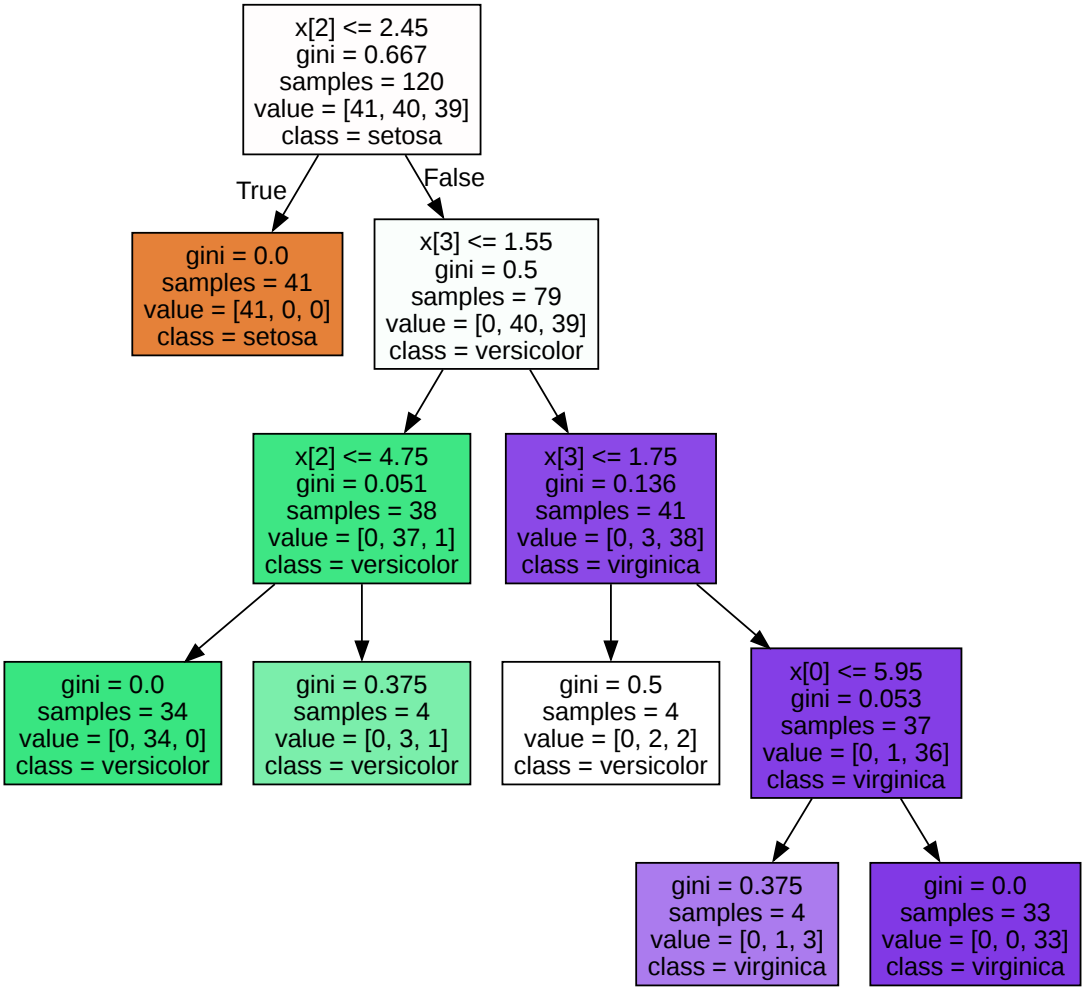
2.2 min_samples_split = 4 (분할 할 수 있는 샘플수를 지정하는 것)

```
# Decision Tree 생성
from sklearn.tree import DecisionTreeClassifier

clf_para2 = DecisionTreeClassifier(random_state=156,
                                   min_samples_split=4

# Decision Tree 학습
clf_para2.fit(X_train,y_train)
```



2.3 min_samples_leaf=4 (leaf가 될 수 있는 샘플수를 지정하는 것)

```
# Decision Tree 생성
from sklearn.tree import DecisionTreeClassifier

clf_para3 = DecisionTreeClassifier(random_state=156,
                                   min_samples_leaf=4)

# Decision Tree 학습
clf_para3.fit(X_train,y_train)
```



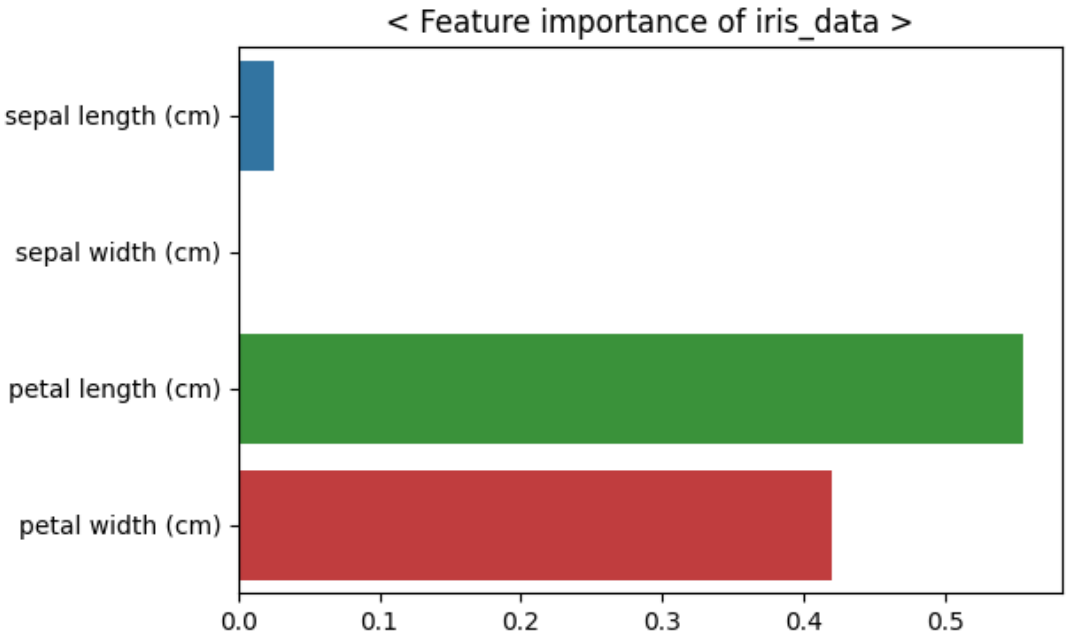
3. 추가 데이터 분석

3.1 Feature importances(특성 중요도)

```
# feature importance 추출
print("Feature importance:\n{0}".format(np.round(clf.fe
print("---" * 20)

# feature importance 매칭
for name, value in zip(iris_data.feature_names, clf.fea
    print("{0} : {1:.3f}".format(name, value))
```

```
Feature importance:
[0.    0.    0.558 0.442]
------------------------------------------------------------
sepal length (cm) : 0.000
sepal width (cm) : 0.000
petal length (cm) : 0.558
petal width (cm) : 0.442
```

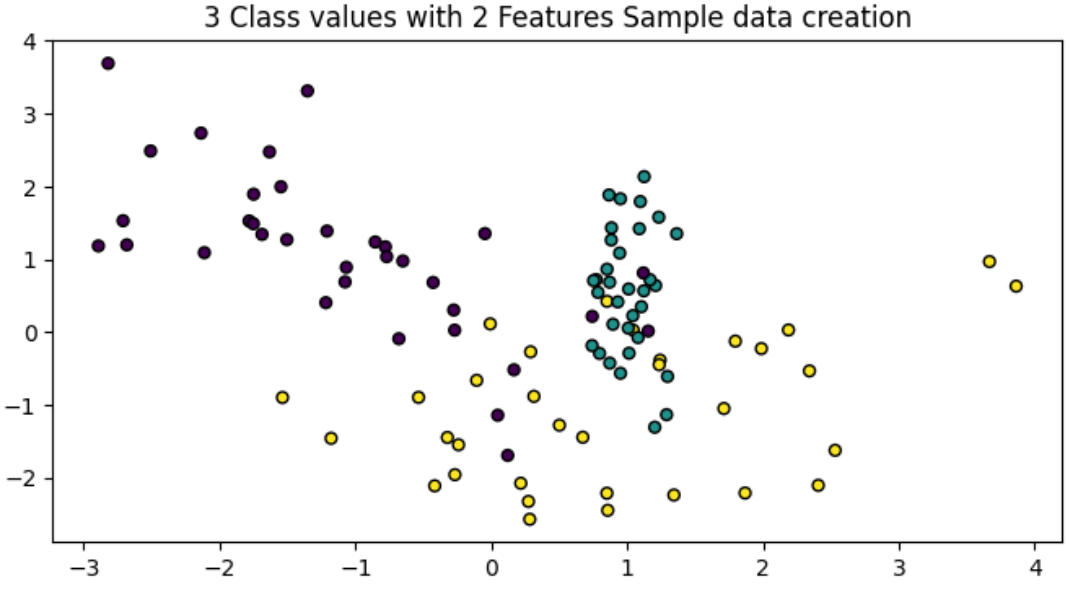
```
# feature importance를 column별로 시각화

plt.figure(figsize=(6,4))
sns.barplot(x=clf.feature_importances_, y=iris_data.fea

plt.title("< Feature importance of iris_data >")

plt.show()
```



3.2 결정 트리 과적합(Overfitting)

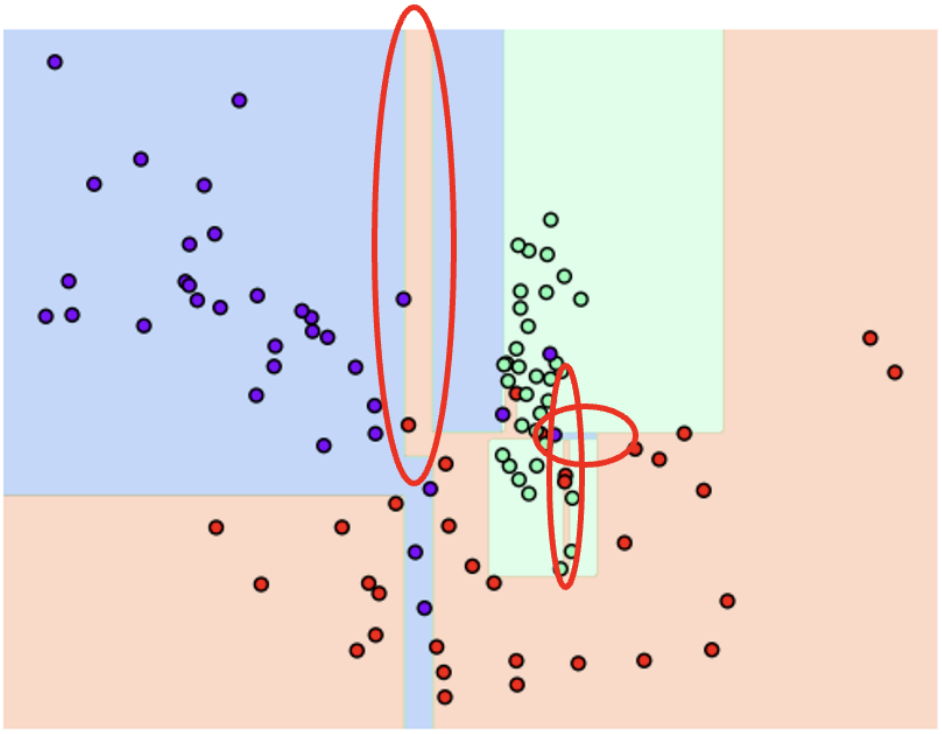
```
from sklearn.datasets import make_classification

plt.figure(figsize=(8,4))
```

```
plt.title("3 Class values with 2 Features Sample data c

# 2차원 시각화를 위해서 피처는 2개, 클래스는 3가지 유형의 분류 샘플
X_features, y_labels = make_classification(n_features=2
                                           n_redundant=
                                           n_informativ
                                           n_classes=3,
                                           n_clusters_p
                                           random_state

# 그래프 형태로 2개의 피처로 2차원 좌표 시각화, 각 클래스 값은 다른
plt.scatter(X_features[:,0], X_features[:,1],
            marker="o", c=y_labels,              # 마커 도
            s=25, edgecolors="k")                # 각 데이

plt.show()
```

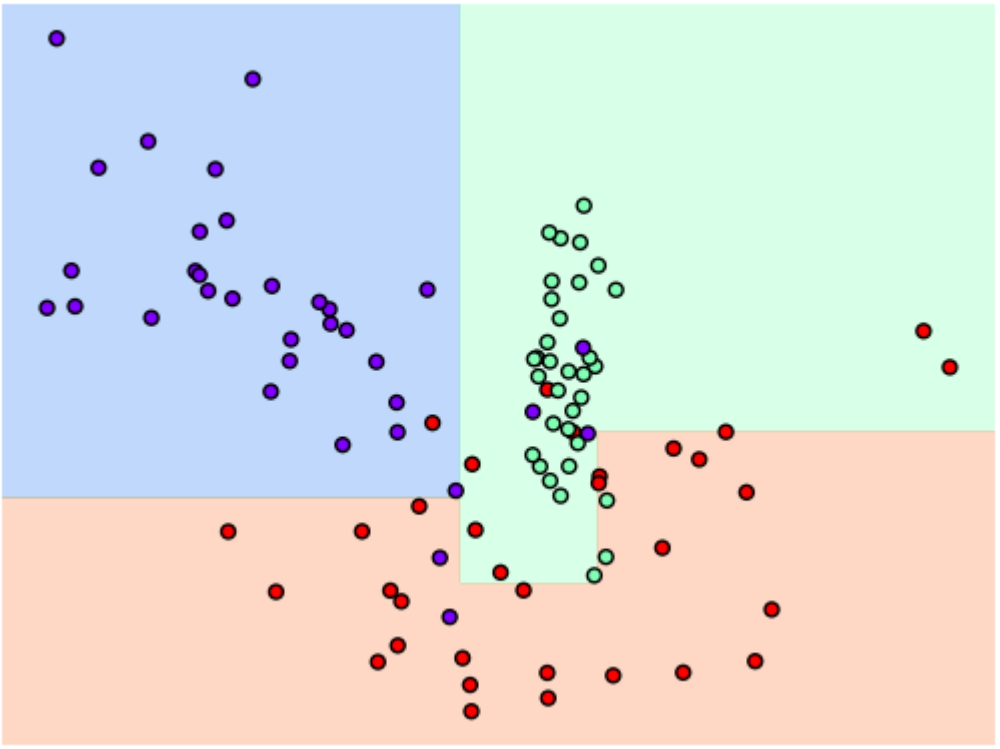



```
# 특정한 트리 생성 제약없는 결정 트리의 Decsion Boundary 시각화.
clf = DecisionTreeClassifier(random_state=156)
```

```
clf.fit(X_features, y_labels)
visualize_boundary(clf, X_features, y_labels)
```



```
# min_samples_leaf=6 으로 트리 생성 조건을 제약한 Decision Bo
clf = DecisionTreeClassifier(min_samples_leaf=6, random_
visualize_boundary(clf, X_features, y_labels)
```



3. 앙상블 학습

▼ 개요

a. 앙상블 학습

- 여러 개의 분류기를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법을 말함.

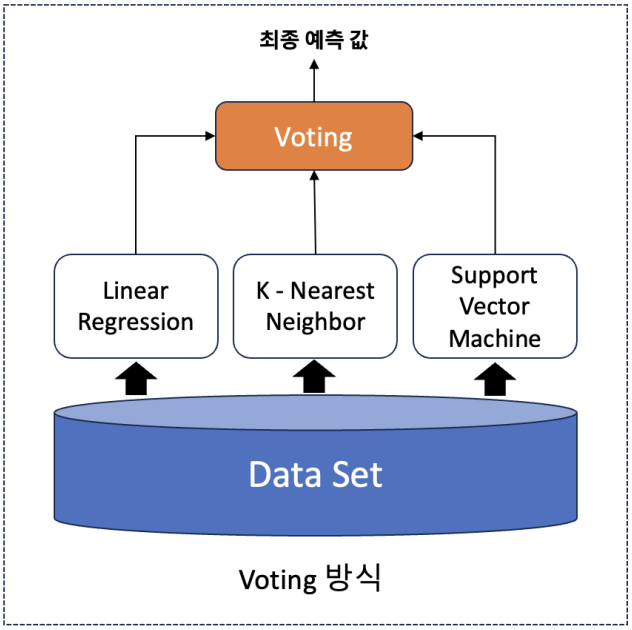
b. 장단점

- 장점
  - 성능을 분산시키기 때문에 과적합(overfitting) 감소 효과가 있음.
  - 개별 모델 성능이 잘 안 나올 때 앙상블 학습을 이용하면 성능이 향상될 수 있음.
- 단점
  - 모델 결과에 대한 해석이 어려움.
  - 예측 시간이 오래 걸림.

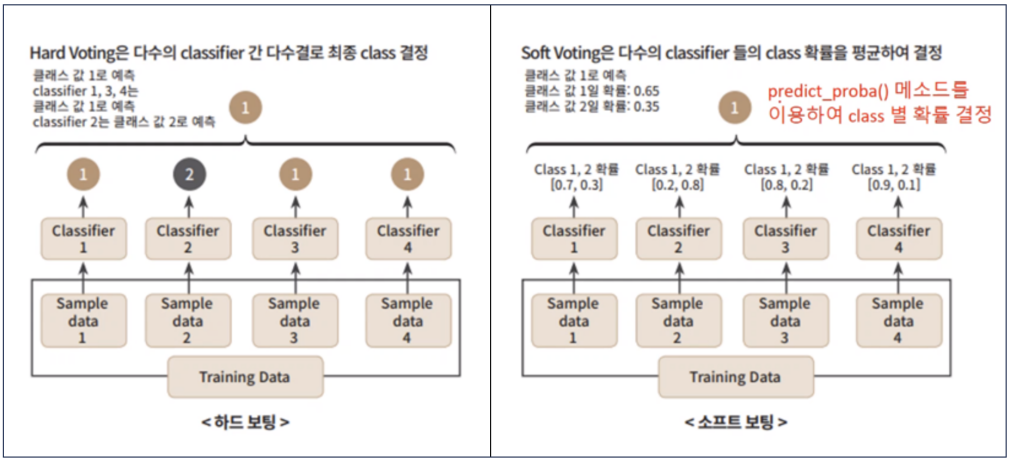
▼ 유형

a. 보팅(Voting) : 같은 하나의 데이터셋을 여러 개의 분류기를 통해 학습하고 예측한 결과를 가지고 보팅을 통해 최종 예측 결과를 선정하는 방식

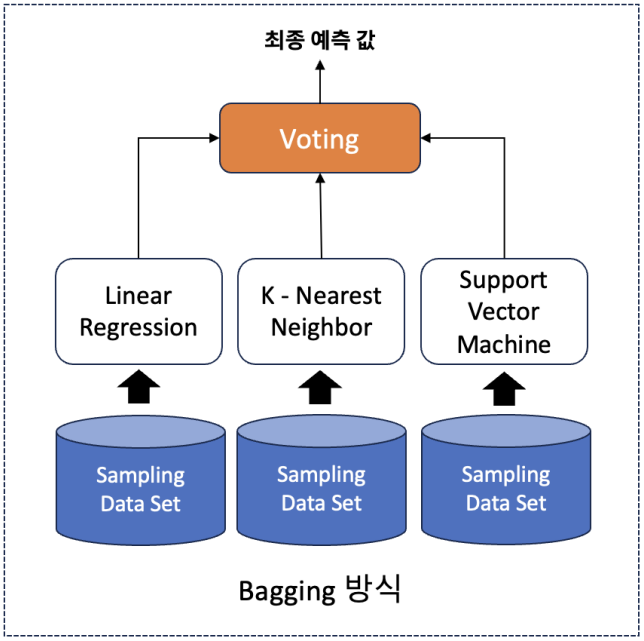
- 하드 보팅(Hard Voting) : 예측한 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정하는 방식. (ex. 다수결 원칙)
- 소프트 보팅(Soft Voting) : 분류기들의 레이블 값 결정 확률을 모두 더하고 이를 평균해서 이들 중 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정하는 방식



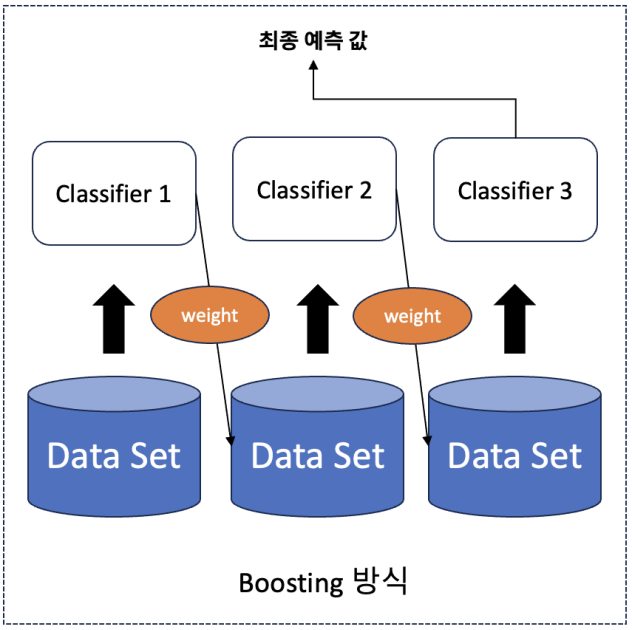
Voting 방식



b. 배깅(Bagging) : 부트스트래핑 방식으로 샘플링된 데이터 세트에 대해서 학습을 통해 개별적인 예측을 수행한 결과는 보팅을 통해서 최종 예측 결과를 선정하는 방식



c. 부스팅(Boosting) : 여러 개의 분류기가 순차적으로 학습을 수행하되, 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서는 올바르게 예측할 수 있도록 다음 분류기에게는 가중치(weight)를 부여하면서 학습과 예측을 진행하는 방식



▼ 실습 - 위스콘신 유방암 데이터셋

1. 앙상블 학습 실습예제 - 위스콘신 유방암 데이터셋

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

1.1 데이터셋 불러오기

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

df = pd.DataFrame(cancer.data, columns=cancer.feature_n
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	wors area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.(
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.(
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.(
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.(

5 rows × 30 columns

1.2 학습, 테스트 데이터 세트 분리

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(can

                                                    tes
                                                    ran
```

1.3 보팅 분류기 및 개별 모델 생성 및 학습

```
# 로지스틱 회귀, KNN 모델
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

lr_clf = LogisticRegression(solver="liblinear")
```

```
knn_clf = KNeighborsClassifier(n_neighbors=8)

# 소프트 보팅 기반의 앙상블 모델로 구현한 분류기 생성
from sklearn.ensemble import VotingClassifier

vo_clf = VotingClassifier(estimators=[("LR",lr_clf),("K
                          voting="soft")
```

1.4 보팅 분류기 및 개별 모델 평가

```
# VotingClassifier 학습/예측/평가
from sklearn.metrics import accuracy_score

vo_clf.fit(X_train, y_train)
y_pred = vo_clf.predict(X_test)

print("Votin 분류기 정확도: {0:.4f}".format(accuracy_score

# 개별 모델의 학습 평가
models = [lr_clf, knn_clf]
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    class_name = model.__class__.__name__
    print("{0} 정확도 : {1:.4f}".format(class_name,
```

```
Voting 분류기 정확도: 0.9561
LogisticRegression 정확도 : 0.9474
KNeighborsClassifier 정확도 : 0.9386
```

---

4. 랜덤 포레스트

▼ 개요

a. 랜덤 포레스트

- 배깅의 대표적인 알고리즘 중 하나임.

- 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을 하게 됨.
- 개별적인 분류기의 기반 알고리즘은 결정 트리이지만 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트임.

b. 부트스트래핑(Boostrapping)

- 원래의 데이터셋으로부터 랜덤 샘플링을 통해 학습데이터를 늘리는 방법(ex. 복원 추출을 허용한 표본 재추출 방법)

▼ 실습 - 사용자 행동 인식 데이터셋

1. 랜덤 포레스트 학습 예제 - 사용자 행동 인식 데이터셋

1.1 랜덤 포레스트 학습, 예측, 평가

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# 랜덤 포레스트 학습
rf_clf = RandomForestClassifier(random_state=0, max_dep
rf_clf.fit(X_train, y_train)

# 랜덤 포레스트 예측
y_pred = rf_clf.predict(X_test)

# 랜덤 포레스트 평가
accuracy = accuracy_score(y_test, y_pred)
print("랜덤 포레스트 정확도 : {0:.4f}".format(accuracy))
```

```
랜덤 포레스트 정확도 : 0.9196
```

1.2 랜덤 포레스트 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV

params = {
```

```
    "max_depth" : [8, 16, 24],
    "min_samples_leaf" : [1, 6, 12],
    "min_samples_split" : [2, 8, 16]
}

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(n_estimators=100,
                                random_state=0,
                                n_jobs=-1)

grid_cv = GridSearchCV(rf_clf, param_grid=params, cv=2,
grid_cv.fit(X_train, y_train)

print("최적 하이퍼 파라미터: \n", grid_cv.best_params_)
print("--- " * 20)
print("최고 예측 정확도: {0:.4f}".format(grid_cv.best_scor
```

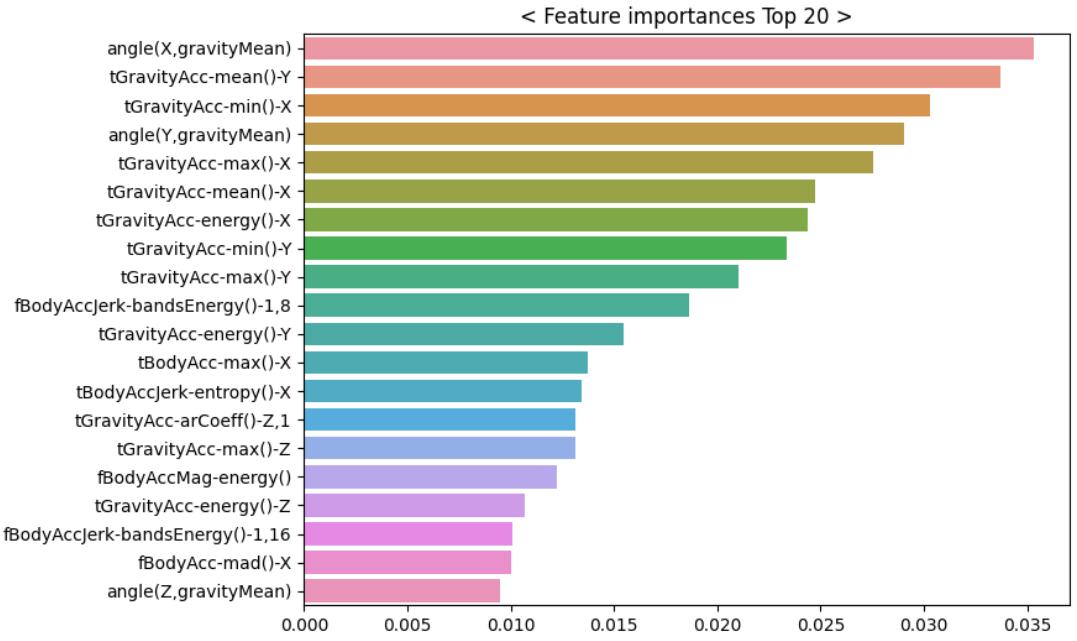
```
최적 하이퍼 파라미터:
 {'max_depth': 16, 'min_samples_leaf': 6, 'min_samples_split': 2}
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
최고 예측 정확도: 0.9165
```

```
# 최적의 하이퍼 파라미터를 적용한 랜덤 포레스트 예측, 평가
rf_clf1 = RandomForestClassifier(n_estimators=100,
                                 min_samples_leaf=6,
                                 max_depth=16,
                                 min_samples_split=2,
                                 random_state=0)

rf_clf1.fit(X_train, y_train)
y_pred = rf_clf1.predict(X_test)
print("예측 정확도: {0:.4f}".format(accuracy_score(y_test,
```

```
예측 정확도: 0.9260
```

1.3 특성 중요도 시각화

```
importance_values = rf_clf1.feature_importances_
importances = pd.Series(importance_values, index=X_trai
top_20 = importances.sort_values(ascending=False)[:20]

plt.figure(figsize=(8,6))
plt.title("< Feature importances Top 20 >")
sns.barplot(x=top_20, y=top_20.index)
plt.show()
```



5. GBM(Gradient Boosting Machine)

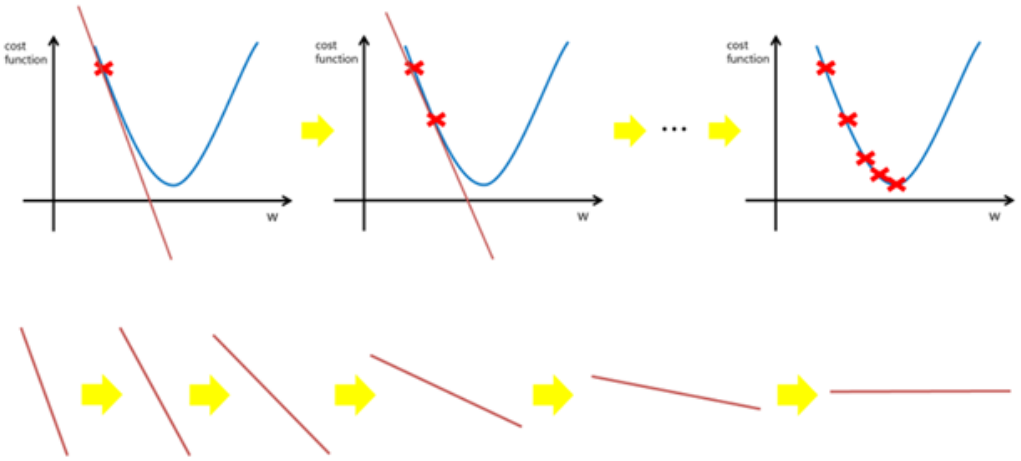
▼ 개요

a. 부스팅(Boosting)

- 여러 개의 약한 학습기를 순차적으로 학습 → 예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식.
- 대표적인 알고리즘 : AdaBoost, GBM, XGBoost, LightGBM

b. GBM

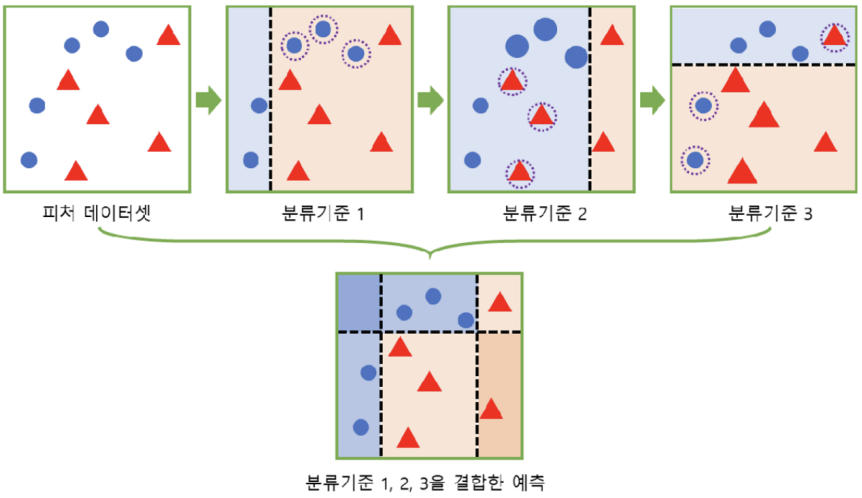
- 에이다부스트와 유사하나, 가중치 업데이트를 경사하강법(Gradient Descent)을 이용하는 차이가 있음.
    - 경사하강법 : 기울기를 줄여나감으로써 오류를 최소화.



- 장단점
    - 장점
        - 예측 성능이 뛰어남
    - 단점
        - 과적합(Overfitting)의 위험이 있음
        - 학습시간이 오래 걸림

c. 에이다 부스트(AdaBoost)

- 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 대표적인 알고리즘
- 첫번째 학습기(분류기준 1)을 통해 분류, 두번째 학습기(분류기준 2)을 통해 분류, 세번째 학습기(분류기준 3)을 통해 분류한 것으로 각각 가중치를 부여하여 모두 결합해 예측을 수행.



▼ 실습 - 사용자 행동 데이터 세트

a. GBM 예제 - 사용자 행동 데이터 세트

1.1 GBM 학습, 예측, 평가

```
# GBM 수행 시간 측정을 위함. 시작 시간 설정.
import time
import warnings

start_time = time.time()
```

```
# GBM 학습
from sklearn.ensemble import GradientBoostingClassifier

gb_clf = GradientBoostingClassifier(random_state=0)
gb_clf.fit(X_train, y_train)


# GBM 예측, 평가
from sklearn.metrics import accuracy_score

y_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, y_pred)
```

```
print("GBM 정확도: {0:.4f}".format(gb_accuracy))
print("GBM 수행 시간: {0:.1f}".format(time.time() - start
print("-----" * 20)
```

1.2 GBM 하이퍼 파라미터 튜닝

```
# 튜닝할 파라미터들 정의
params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3]
}

# GridSearchCV를 사용하여 최적의 파라미터 탐색
from sklearn.model_selection import GridSearchCV

grid_cv = GridSearchCV(gb_clf, params, cv=3, n_jobs=-1)
grid_cv.fit(X_train, y_train)

# 최적의 파라미터 출력
best_params = grid_cv.best_params_
print("최적의 파라미터:", best_params)

# 최적의 파라미터를 적용한 GBM 모델 생성 및 학습
best_gb_clf = GradientBoostingClassifier(random_state=0
best_gb_clf.fit(X_train, y_train)

# 예측 및 평가
y_pred_best = best_gb_clf.predict(X_test)
best_gb_accuracy = accuracy_score(y_test, y_pred_best)

# 결과 출력
print("최적 파라미터 적용 GBM 정확도: {0:.4f}".format(best_g
print("최적 파라미터 적용 GBM 수행 시간: {0:.1f}".format(time
```

6. XGBoost(eXtra Gradient Boost)

▼ 개요

a. XGBoost

- 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나

b. 장점

- 뛰어난 예측 성능 : 일반적으로 분류와 회귀 영역에서 뛰어난 예측 성능을 발휘
- GBM 대비 빠른 수행 시간 : 일반적인 GBM과 달리 XGBoost는 병렬 수행 및 다양한 기능으로 빠른 수행 성능을 보장. 하지만 GBM에 비해 빠른 편이지 다른 머신러닝 알고리즘에 비해 빠르다는 것은 아님.
- 과적합 규제 : 자체에 과적합 규제 기능으로 과적합에 좀 더 강한 내구성을 가지고 있음
- 나무 가지치기(Tree pruning) : max depth 파라미터를 이용하여 트리 분할 깊이를 조절할 수도 있지만, tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기 해서 분할 수를 더 줄이는 추가적인 장점을 가지고 있음
- 자체 내장된 교차 검증 : 반복 수행 시마다 내부적으로 학습 데이터 세트와 평가 데이터 세트에 대한 교차 검증을 수행해 최적화된 반복 수행 횟수를 가질 수 있음
- 결손값 자체 처리 : 결손값을 자체 처리할 수 있는 기능을 가지고 있음
- 다양한 평가지표 제공 : 예측 모델의 성능을 평가하기 위한 다양한 평가 지표를 제공 (ex. 분류 작업의 경우 정확도, 정밀도, 재현율, F1 score 등이 가능하고, 회귀 작업의 경우 평균 제곱 오차(MSE)나 R^2를 사용)

c. 단점

- 매개변수 튜닝의 어려움 : 다양한 매개변수를 활용하여 모델을 조정할 수 있지만, 이 매개변수들을 효과적으로 조합하여 최적의 모델을 구축하는 것은 어려울 수 있음
- 자원 소모 : 많은 메모리와 프로세싱 파워를 요구함
- 해석이 어려운 모델 : 앙상블 학습 방법이기 때문에 앙상블된 모델의 예측과 예측에 기여한 개별 트리의 의미를 해석하기는 어려울 수 있음

▼ 실습 - 위스콘신 유방암 데이터셋

1. 파이썬 래퍼 XGBoost 적용

1.1 데이터 불러오기

```
# 데이터셋 불러오기
from sklearn.datasets import load_breast_cancer

dataset = load_breast_cancer()
X_features= dataset.data
y_label = dataset.target

cancer_df = pd.DataFrame(data=dataset.data, columns=dat
cancer_df['target']= dataset.target
cancer_df.head(3)
```

1.2 학습, 검증, 테스트 데이터 분리

```
# cancer_df에서 feature용 DataFrame과 Label용 Series 객체
X_features = cancer_df.iloc[:, :-1]
y_label = cancer_df.iloc[:,-1]

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(X_fea
                                                  test_size=0.2,

# 위에서 만든 X_train, y_train을 다시 쪼개서 90%는 학습과 10%
X_tr, X_val, y_tr, y_val= train_test_split(X_train, y_t
print(X_train.shape , X_test.shape)
print(X_tr.shape, X_val.shape)
```

```
(455, 30) (114, 30)
(409, 30) (46, 30)
```

```
# 만약 구버전 XGBoost에서 DataFrame으로 DMatrix 생성이 안될 경
# 학습, 검증, 테스트용 DMatrix를 생성.
```

```
import xgboost as xgb

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)
```

1.3 XGBoost 학습, 예측, 평가

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

# 학습 데이터 셋은 'train' 또는 평가 데이터 셋은 'eval' 로 명기합
eval_list = [(dtr,'train'),(dval,'eval')] # 또는 eval_li

# 하이퍼 파라미터와 early stopping 파라미터를 train( ) 함수의
xgb_model = xgb.train(params = params , dtrain=dtr,
                                          num_boost_r
                      early_stopping_rounds=50,
                                          evals=eval_
```

```
[0]     train-logloss:0.62480   eval-logloss:0.63104
[1]     train-logloss:0.58674   eval-logloss:0.60478
[2]     train-logloss:0.55226   eval-logloss:0.58223
[3]     train-logloss:0.52086   eval-logloss:0.56184
[4]     train-logloss:0.49192   eval-logloss:0.54118
[5]     train-logloss:0.46537   eval-logloss:0.52223
[6]     train-logloss:0.44029   eval-logloss:0.50287
[7]     train-logloss:0.41666   eval-logloss:0.48620
[8]     train-logloss:0.39525   eval-logloss:0.46974
[9]     train-logloss:0.37542   eval-logloss:0.45497
[10]    train-logloss:0.35701   eval-logloss:0.44131
[11]    train-logloss:0.33982   eval-logloss:0.43134
[12]    train-logloss:0.32297   eval-logloss:0.41972
[13]    train-logloss:0.30725   eval-logloss:0.40902
[14]    train-logloss:0.29327   eval-logloss:0.39883
[15]    train-logloss:0.27946   eval-logloss:0.38968
[16]    train-logloss:0.26691   eval-logloss:0.38150
[17]    train-logloss:0.25473   eval-logloss:0.37368
[18]    train-logloss:0.24385   eval-logloss:0.36666
[19]    train-logloss:0.23338   eval-logloss:0.35994
[20]    train-logloss:0.22320   eval-logloss:0.35374
[21]    train-logloss:0.21363   eval-logloss:0.34704
[22]    train-logloss:0.20487   eval-logloss:0.34206
[23]    train-logloss:0.19634   eval-logloss:0.33621
[24]    train-logloss:0.18830   eval-logloss:0.33178
[25]    train-logloss:0.18093   eval-logloss:0.32774
[26]    train-logloss:0.17374   eval-logloss:0.32297
[27]    train-logloss:0.16695   eval-logloss:0.31855
[28]    train-logloss:0.16059   eval-logloss:0.31495
[29]    train-logloss:0.15450   eval-logloss:0.31173
[30]    train-logloss:0.14875   eval-logloss:0.30735
[31]    train-logloss:0.14329   eval-logloss:0.30463
[32]    train-logloss:0.13807   eval-logloss:0.30242
[33]    train-logloss:0.13325   eval-logloss:0.29922
[34]    train-logloss:0.12864   eval-logloss:0.29722
...
[247]   train-logloss:0.00935   eval-logloss:0.23838
[248]   train-logloss:0.00933   eval-logloss:0.23821
[249]   train-logloss:0.00931   eval-logloss:0.23872
[250]   train-logloss:0.00925   eval-logloss:0.23805
```

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

# 예측 확률이 0.5 보다 크면 1 , 그렇지 않으면 0 으로 예측값 결정하
```

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

```
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]
```

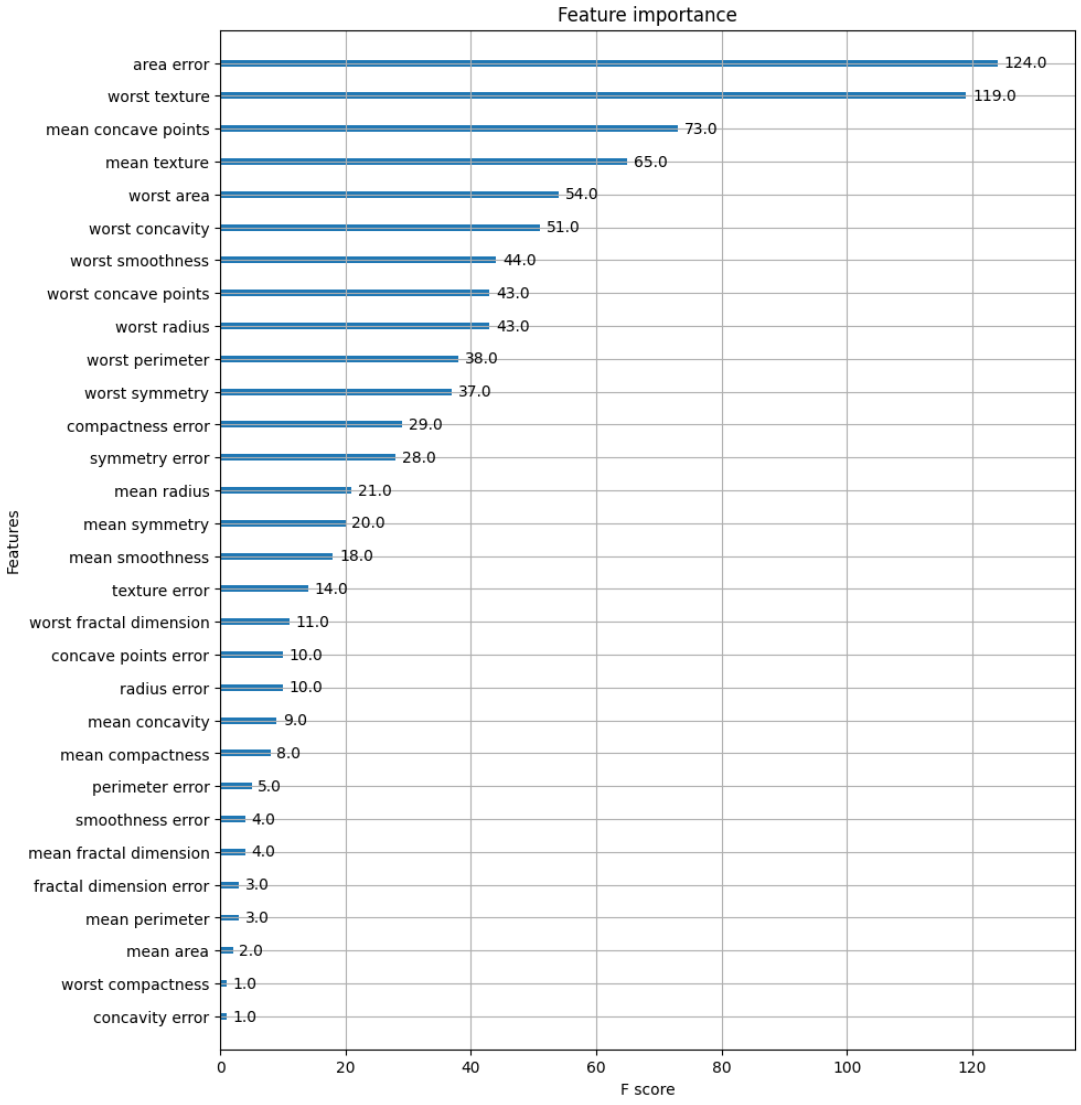
```
# 평가
get_clf_eval(y_test , preds, pred_probs)
```

```
오차 행렬
[[35  2]
 [ 2 75]]
정확도: 0.9649, 정밀도: 0.9740, 재현율: 0.9740,    F1: 0.9740, AUC:0.9965
```

- 실제 negative를 모델이 negative로 예측 : 35
- 실제 positive를 모델이 positive로 예측 : 75

1.4 특성 중요도 시각화

```
# 기본 평가 지표(F 스코어) - 해당 피처가 트리 분할 시 얼마나 자주
from xgboost import plot_importance

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(xgb_model, ax=ax)
plt.savefig('p239_xgb_feature_importance.tif', format='
```



2. 사이킷런 래퍼 XGBoost 적용

2.1 xgboost 학습, 예측, 평가

```
`# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400, learning_

evals = [(X_test, y_test)]

xgb_wrapper.fit(X_train, y_train, early_stopping_rounds
                                  eval_metric="logloss",
```

```
                  eval_set=evals, verbose=True)

ws100_preds = xgb_wrapper.predict(X_test)
ws100_pred_proba = xgb_wrapper.predict_proba(X_test)[:,
```

```
[0]     validation_0-logloss:0.56554
[1]     validation_0-logloss:0.50669
[2]     validation_0-logloss:0.45868
[3]     validation_0-logloss:0.41822
[4]     validation_0-logloss:0.38103
[5]     validation_0-logloss:0.35137
[6]     validation_0-logloss:0.32588
[7]     validation_0-logloss:0.30127
[8]     validation_0-logloss:0.28197
[9]     validation_0-logloss:0.26265
[10]    validation_0-logloss:0.24821
[11]    validation_0-logloss:0.23231
[12]    validation_0-logloss:0.22079
[13]    validation_0-logloss:0.20795
[14]    validation_0-logloss:0.19764
[15]    validation_0-logloss:0.18950
[16]    validation_0-logloss:0.18052
[17]    validation_0-logloss:0.17246
[18]    validation_0-logloss:0.16512
[19]    validation_0-logloss:0.15828
[20]    validation_0-logloss:0.15436
```

```
[135]   validation_0-logloss:0.08934
[136]   validation_0-logloss:0.08891
[137]   validation_0-logloss:0.08949
[138]   validation_0-logloss:0.08962
[139]   validation_0-logloss:0.08969
[140]   validation_0-logloss:0.08963
[141]   validation_0-logloss:0.08948
[142]   validation_0-logloss:0.08961
[143]   validation_0-logloss:0.09038
[144]   validation_0-logloss:0.09033
...
[196]   validation_0-logloss:0.08787
[197]   validation_0-logloss:0.08790
[198]   validation_0-logloss:0.08792
[199]   validation_0-logloss:0.08800
```

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

```
오차 행렬
[[34  3]
 [ 1 76]]
정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870,    F1: 0.9744, AUC:0.9951
```

2.2 특성 중요도 시각화

```
fig, ax = plt.subplots(figsize=(10, 12))
# 사이킷런 래퍼 클래스를 입력해도 무방.
plot_importance(xgb_wrapper, ax=ax)
```