분류

1. 분류의 개요

- 지도학습
  - 레이블(Label = 정답)이 있는 데이터를 학습하는 머신러닝 방식
  - 분류(Classification)
    - 데이터의 피처, 레이블 값을 학습 → 모델 생성 → 새로운 데이터에서 레이블 값 예측

알고리즘 이름	설명
나이브 베이즈	베이즈 통계, 생성 모델 기반
로지스틱 회귀	독립변수, 종속변수 선형 관계성 기반
결정 트리	데이터 균일도에 따른 규칙 기반
서포트 벡터 머신	클래스 간의 최대 분류 마진을 찾아줌
최소 근접	근접 거리 기준
신경망	심층 연결 기반 * 이미지, 영상, 음성, NLP 영역
앙상블	서로 다른(같을 때도 있음) 머신러닝 알고리즘 결합 * 정형데이터 예측 성능 높음

- **앙상블(Ensemble)**
  - 기본 알고리즘: **결정 트리(Decision Tree)**
    - 결정 트리의 단점: 과적합 ⇒ 예측 성능 떨어짐
    - 그러나! 결정 트리가 약한 학습기로 작용 ⇒ 앙상블에서 장점이 됨

      *약한 학습기: 예측 성능이 상대적으로 떨어지는 학습 알고리즘
  - 배깅(Bagging)
    - 랜덤 포레스트(Random Forest)
  - 부스팅(Boosting)

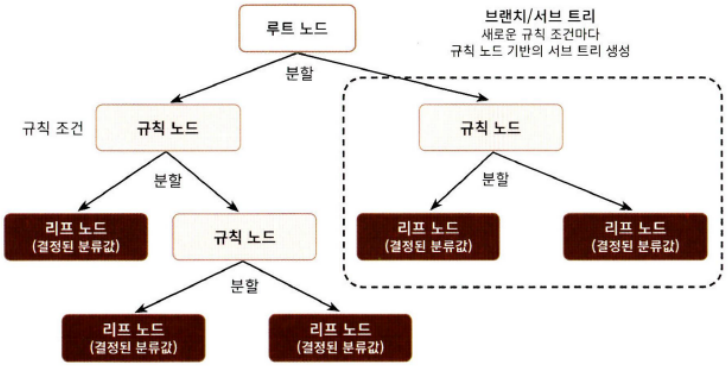
알고리즘 이름	설명
그래디언트 부스팅 (Gradient Boosting)	- 뛰어난 예측 성능 - 시간 오래 걸림
XgBoost, LightGBM	- 예측 성능 더 높음 - 시간 단축

2. 결정 트리

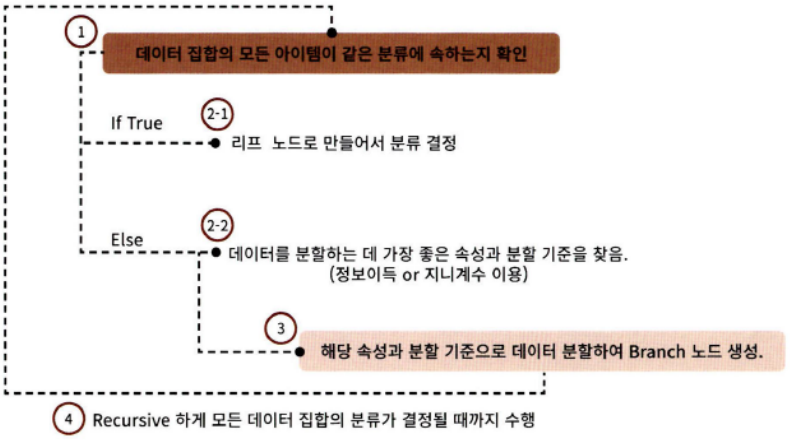
- 학습을 통해 규칙을 찾고, 트리 기반의 분류 규칙 (if/else) 만드는 방식
- 성능 결정 기준: 어떻게 규칙을 만들어야 가장 효율적인 분류인가?
  - 균일한 데이터 세트로 분할
    - 균일도 높음 → 데이터 구분하는데 필요한 정보의 양이 적음 (데이터 비슷함)

        - 균일도 측정
            1. 정보 이득(Information Gain) 지수
                - 1 - 엔트로피 지수
                    - 서로 다른 값이 섞여 있음 → 엔트로피 (혼잡도) 높음
                - 정보 이득 지수 높은 속성으로 분할
            2. 지니 계수 (불평등 지수)
                - 지니 계수 낮은 속성으로 분할
- 구조

이름	설명
규칙 노드	규칙 조건
리프 노드	결정된 분류값
브랜치/서브 트리	- 새로운 규칙 노드 생길 때마다 서브 트리 생성됨 - 자식 노드 있는 노드



- 알고리즘
    - **DecisionTreeClassifier**
        - 지니 계수 이용



결정 트리 모델 특징

장점	단점
1. 알고리즘 쉽고 직관적임 (기본 규칙: 균일도) 2. 시각화 가능함 3. 전처리 작업(피처 스케일링, 정규화) 필요 없음	1. 과적합으로 인해 정확도 떨어짐 ⇒ 트리 크기를 사전에 제한해야 함!

결정 트리 파라미터

- **DecisionTreeClassifier:** 분류 클래스
- **DecisionTreeRegressor**: 회귀 클래스
- CART(Classification and Regression Trees): 분류, 회귀 모두 사용 가능한 트리 알고리즘

파라미터 명	설명
min_samples_split	- 노드 분할하기 위한 최소한의 샘플 데이터 수 - 작게 설정할 수록 과적합 가능성 증가
min_samples_leaf	- 말단 노드(leaf) 되기 위한 최소한의 데이터 수 - 비대칭적 데이터 → 특정 클래스 데이터가 극도로 작을 수 있음 ⇒ 작게 설정해야 제데로 분류됨
max_features	- 분할 시 고려할 최대 피처 개수 - 디폴트 None : 전체 피처 선정 - int 형: 피처 수 / float형: 피처의 퍼센트 - sqrt: sqrt(전체 피처 개수) - auto: =sqrt - log: log2(전체 피처 개수)
max_depth	- 트리 최대 깊이 - 디폴트 None - 깊이 깊어지면 과적합 가능성 증가
max_leaf_nodes	말단 노드(leaf) 최대 개수

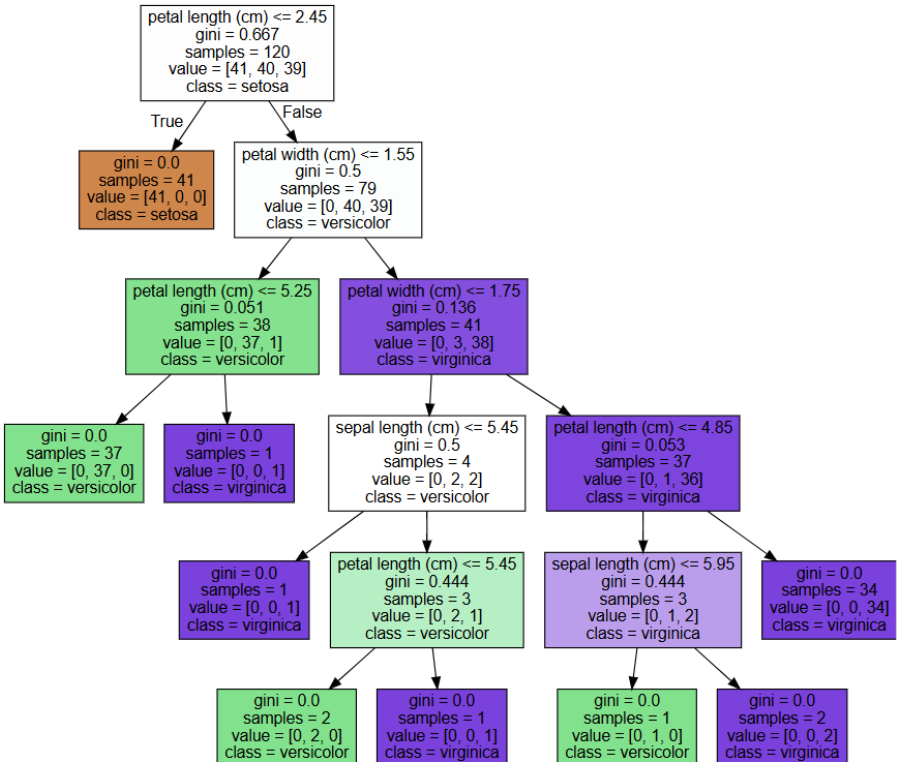
결정 트리 모델 시각화

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

# DecisionTreeClassifier 생성
dt_clf = DecisionTreeClassifier(random_state=156)

# data load
iris_data =load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data
                                        test_size=0.2, random_state=11)
```

```
# DecisionTreeClassifier 학습
dt_clf.fit(X_train, y_train)
```

```
from sklearn.tree import export_graphviz

# 호출 결과가 tree.dot
export_graphviz(dt_clf, out_file="tree.dot", class_names=iris_data.target_nam
            feature_names=iris_data.feature_names, impurity=True, filled=True
```

```
import graphviz

with open("tree.dot") as f:
  dot_graph = f.read()
graphviz.Source(dot_graph)
```

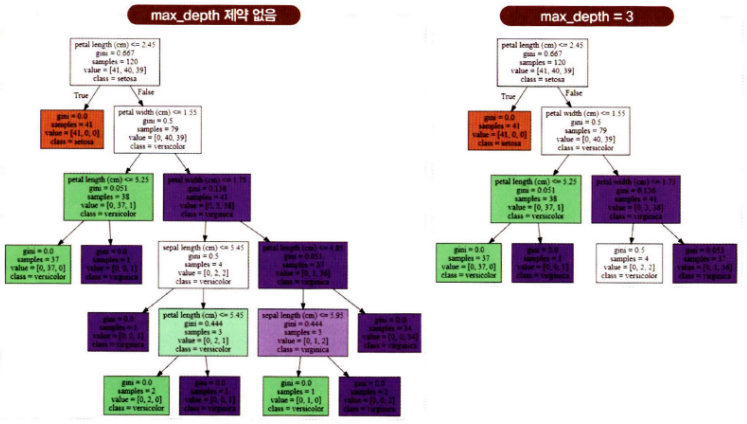


- 노드의 지표

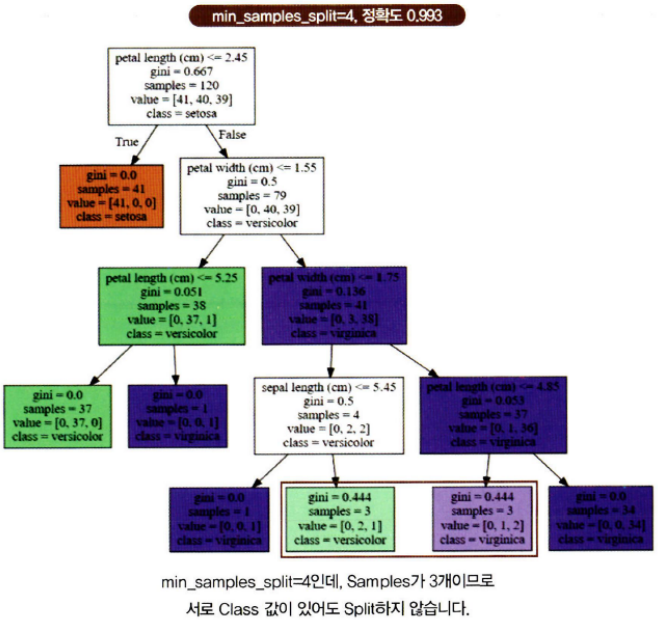
petal length≤2.45	규칙 조건
gini	value=[ ] 데이터 분포에서의 지니 계수
samples	규칙에 해당하는 데이터 건수
value=[ ]	클래스 값 기반의 데이터 건수 ex. value=[41, 40, 39] ; Setosa 41개, Vesicolor: 40개, Virginica: 39개

class	하위 노드 가질 때, 해당 클래스 수가 가장 많음 말단 노드일 때, 예측 클래스 결정값

- 노드의 색깔
  - 붓꽃 데이터 레이블 값
  - 색이 짙다 → 지니 계수 낮음, 레이블에 속하는 샘플 데이터 많음
- 하이퍼 파라미터 변경
  1. max_depth

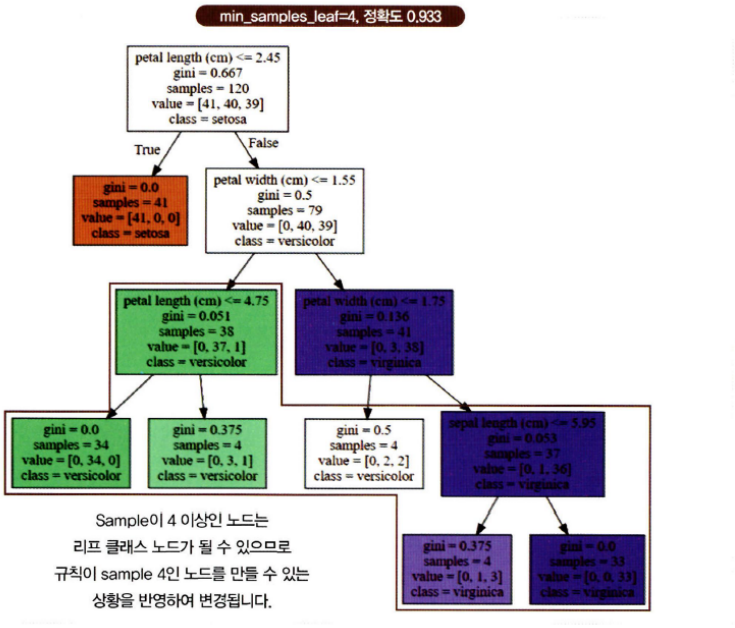


     - 깊이가 줄어든다 → 간단한 트리
  2. min_samples_split



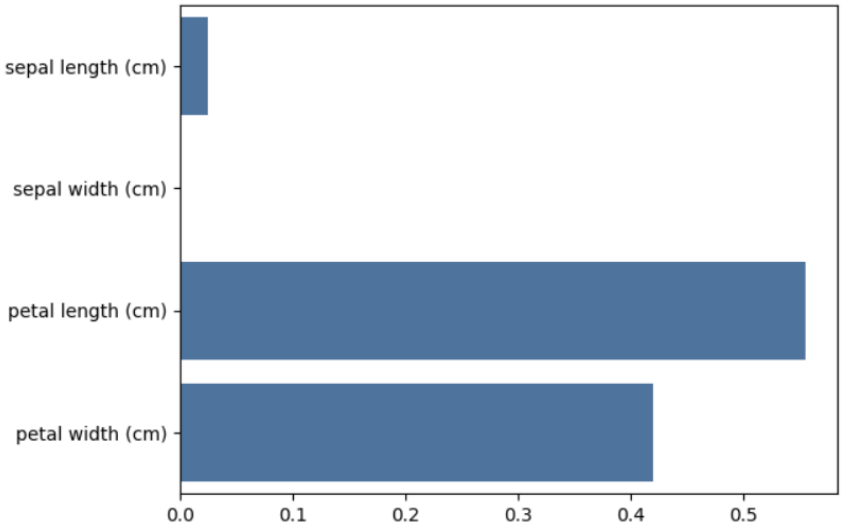
min_samples_split=4인데, Samples가 3개이므로
서로 Class 값이 있어도 Split하지 않습니다.

  3. min_samples_leaf



- 피처 중요도 - feature_importances_

```
import seaborn as sns
import numpy as np
%matplotlib inline
## 노트북을 실행한 브라우저에서 그림 확인 가능

# feature importance 추출
print("Feature importances:\n{0}".format(np.round(dt_clf.feature_importanc

# feature별 importance 매핑
for name, value in zip(iris_data.feature_names, dt_clf.feature_importances
  print('{0}: {1:.3f}'.format(name, value))

# feature importance 컬럼별 시각화
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_names)
```



결정 트리 과적합

- 테스트용 데이터 세트 생성

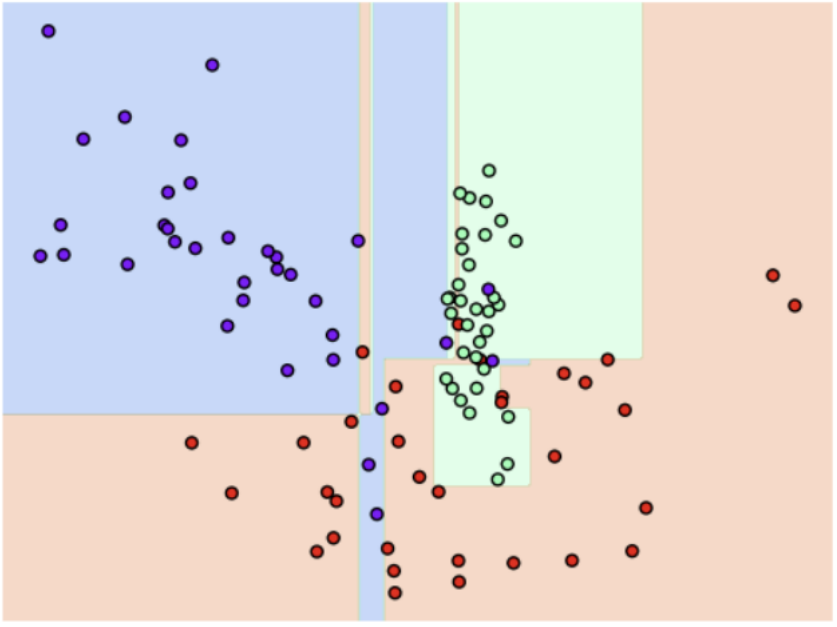
```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('3 Class values with 2 Features Sample data Creation')

# 피처 2개, 클래스 3가지 유형 분류 데이터 샘플 생성
X_features, y_labels = make_classification(n_features=2, n_redundant=0, n_
                                           n_classes=3, n_clusters_per_cla

# 그래프 형태로 2개의 피처 2차원 좌표 시각화, 각 클래스는 다른 색으로 표현
plt.scatter(X_features[:,0], X_features[:,1], marker='o', c=y_labels, s=25
```

- 결정 기준 경계 시각화

```
# Classifier의 Decision Boundary를 시각화 하는 함수
def visualize_boundary(model, X, y):
    fig,ax = plt.subplots()

    # 학습 데이타 scatter plot으로 나타내기
    ax.scatter(X[:, 0], X[:, 1], c=y, s=25, cmap='rainbow', edgecolor='k',
               clim=(y.min(), y.max()), zorder=3)
    ax.axis('tight')
    ax.axis('off')
    xlim_start , xlim_end = ax.get_xlim()
    ylim_start , ylim_end = ax.get_ylim()

    # 호출 파라미터로 들어온 training 데이타로 model 학습 .
    model.fit(X, y)
    # meshgrid 형태인 모든 좌표값으로 예측 수행.
```

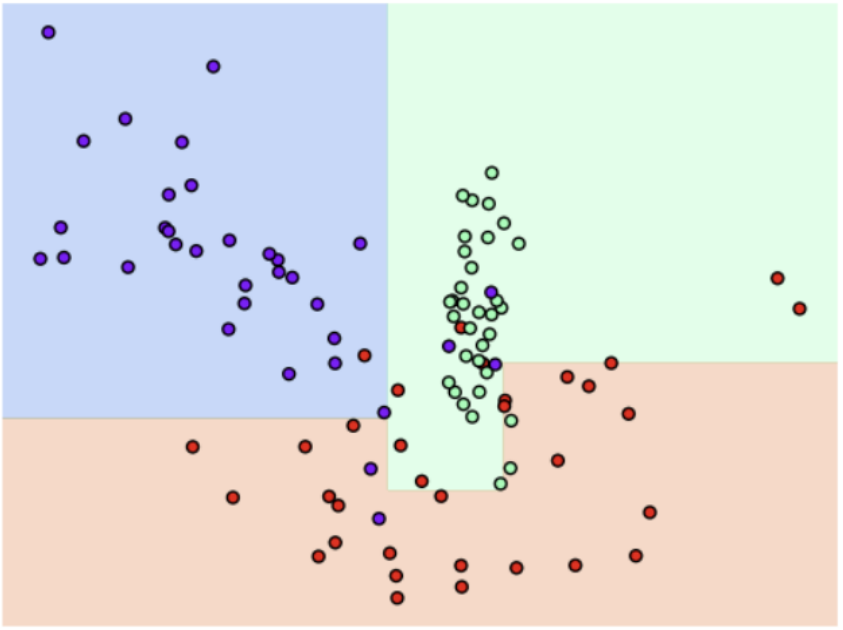
```
    xx, yy = np.meshgrid(np.linspace(xlim_start,xlim_end, num=200),np.lins
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    # contourf() 를 이용하여 class boundary 를 visualization 수행.
    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels=np.arange(n_classes + 1) - 0.5,
                           cmap='rainbow', clim=(y.min(), y.max()),
                           zorder=1)
```

```
from sklearn.tree import DecisionTreeClassifier

# 결정 트리 학습, 결정 경계 시각화
dt_clf=DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```



- 분할이 자주 일어남 → 결정 기준 경계가 많아짐 (복잡해짐!)

```
# min_samples_leaf=6으로 노드 생성 규칙 완화

# min_samples_leaf=6으로 트리 생성 조건 제약, 결정 경계 시각화
dt_clf=DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels
visualize_boundary(dt_clf, X_features, y_labels)
```



- 학습 데이터에만 과적합된 분류 기준은 예측 성능 떨어뜨릴 수 있음

  ⇒ 노드 생성 규칙 완화한 모델의 성능 더 뛰어날 수 있음

결정 트리 실습 - 사용자 행동 인식 데이터 세트

```
from google.colab import drive
drive.mount('/content/gdrive/')

path = "/content/gdrive/MyDrive/Euron/"
```

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

feature_name_df = pd.read_csv(path+'human_activity/features.txt', sep='\s+',
                              header=None, names=['column_index', 'column_name'])

# 피처명 index 제거, 피처명만 list 객체로 생성
feature_name = feature_name_df.iloc[:,1].values.tolist()
print("전체에서 10개만 추출:", feature_name[:10])
```

```
# 중복된 피처명 알아보기
feature_dup_df=feature_name_df.groupby('column_name').count()
print(feature_dup_df[feature_dup_df['column_index']>1].count())
feature_dup_df[feature_dup_df['column_index']>1].head()
```

- 42개의 피처명 중복

```
# 원본 피처명에 _1, _2 추가하기
def get_new_feature_name_df(old_feature_name_df):
  feature_dup_df=pd.DataFrame(data=old_feature_name_df.groupby('column_name')
  feature_dup_df=feature_dup_df.reset_index()
  new_feature_name_df=pd.merge(old_feature_name_df.reset_index(),feature_dup_
  new_feature_name_df['column_name']=new_feature_name_df[['column_name', 'dup
  new_feature_name_df=new_feature_name_df.drop(['index'],axis=1)
  return new_feature_name_df
```

```
import pandas as pd

def get_human_dataset():

  # read_csv에서 공백 문자를 sep로 할당
  feature_name_df=pd.read_csv(path+'human_activity/features.txt',sep='\s+',
                          header=None, names=['column_index', 'column_name'])

  # 중복된 피처명 수정하는 get_new_feature_name_df( ) 이용, 신규 피처명 DataFrame 설정
  new_feature_name_df=get_new_feature_name_df(feature_name_df)

  # DataFrame에 피처명을 칼럼으로 부여하기 위해 list로 변환
  feature_name=new_feature_name_df.iloc[:,1].values.tolist()

  # 학습/테스트 피처 데이터를 DataFrame으로 로딩, 칼럼명은 feature_name 적용
  X_train=pd.read_csv(path+'human_activity/train/X_train.txt',sep='\s+',names
  X_test=pd.read_csv(path+'human_activity/test/X_test.txt',sep='\s+',names=fe

  # 학습/테스트 레이블 데이터를 DataFrame으로 로딩, 칼럼명은 action으로 부여
  y_train=pd.read_csv(path+'human_activity/train/y_train.txt',sep='\s+',heade
  y_test=pd.read_csv(path+'human_activity/test/y_test.txt',sep='\s+',header=N

  # 학습/테스트용 DataFrame 모두 반환
  return X_train,X_test,y_train,y_test

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

```
print('## 학습 피처 데이터셋 info() ##')
print(X_train.info())
```

```
print(y_train['action'].value_counts())
```

```
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))

# DecisionTreeClassifier 하이퍼 파라미터 추출
print('DecisionTreeClassifier 기본 하이퍼 파라미처 :\n', dt_clf.get_params())
```

```
from sklearn.model_selection import GridSearchCV

params={
    'max_depth': [6,8,10,12,16,20,24]
}

grid_cv=GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, ver
grid_cv.fit(X_train, y_train)
print('GridSearchCV 최고 평균 정확도 수치: {0: .4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최고 하이퍼 파라미터: ', grid_cv.best_params_)
```

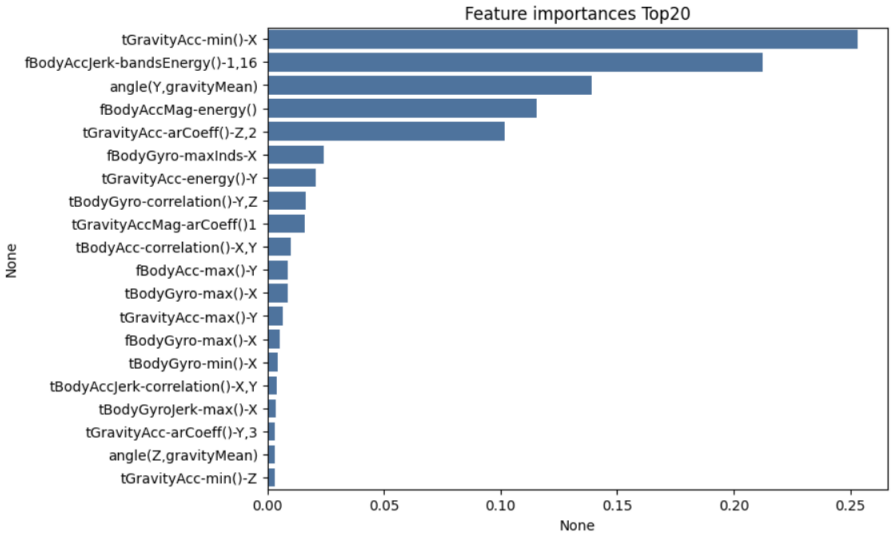
```
# GridSearchCV의 cv_results_ 속성을 DataFrame으로 생성
cv_results_df=pd.DataFrame(grid_cv.cv_results_)

# max_depth 파라미터 값, 테스트 세트, 학습 데이터 세트의 정확도 수치 추출
cv_results_df[['param_max_depth', 'mean_test_score']]
```

```
max_depths=[6,8,10,12,16,20,24]
# max_depth 변화하면서 그때마다 학습/테스트 세트에서 예측 성능 측정
for depth in max_depths:
  dt_clf=DecisionTreeClassifier(max_depth=depth, random_state=156)
  dt_clf.fit(X_train, y_train)
  pred=dt_clf.predict(X_test)
  accuracy=accuracy_score(y_test,pred)
  print('max_depth={0} 정확도: {1:.4f}'.format(depth, accuracy))
```

```
params ={
    'max_depth': [8,12,16,20],
    'min_samples_split': [16,24]
}

grid_cv=GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, ver
grid_cv.fit(X_train, y_train)
print('GridSearchCV 최고 평균 정확도 수치: {0: .4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최고 하이퍼 파라미터: ', grid_cv.best_params_)
```

```
best_df_clf=grid_cv.best_estimator_
pred1=best_df_clf.predict(X_test)
accuracy=accuracy_score(y_test, pred1)
print('결정 트리 예측 정확도 :{0: .4f}'.format(accuracy))
```

```
import seaborn as sns

ftr_importances_values=best_df_clf.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 Top20')
sns.barplot(x=ftr_top20, y=ftr_top20.index)
plt.show()
```



3. 앙상블 학습

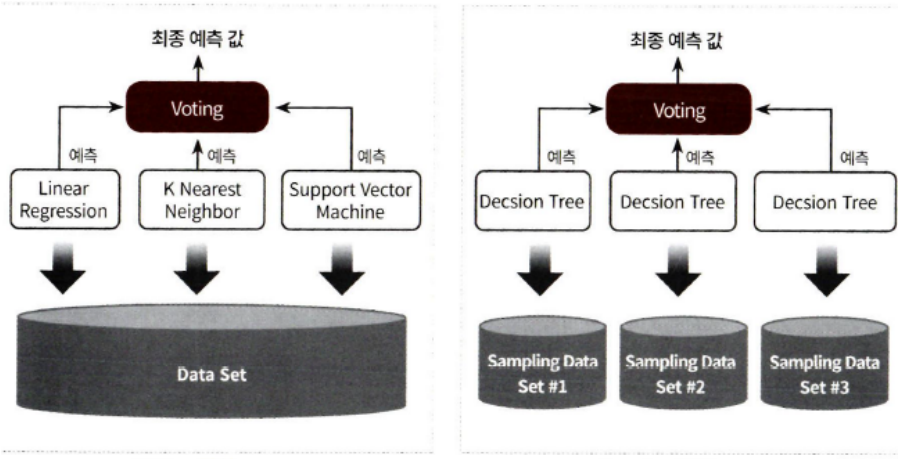
- 여러 개의 분류기(Classifier) 생성 → 예측 결합 → 최종 예측 결과
- 집단 지성 이용!

학습 유형 이름	설명	예시
보팅	- 투표로 최종 예측 결과 (다수결) - 서로 다른 분류기, 같은 샘플 데이터	
배깅	- 투표로 최종 예측 결과 (다수결) - 분류기 같지만 다른 샘플 데이터 * **부트스트래핑(Boostrapping) 분할**	랜덤 포레스트
부스팅	여러 분류기가 순차적으로 학습 → 예측이 틀린 데이터 발생 → 해당 데이터에 대해 다른 분류기에 가중치(weight) 부여함	그래디언트 부스트 XGBoost LightGBM

학습 유형 이름	설명	예시
스태킹	여러 모델의 예측 결과값을 학습 데이터로 → 다른 모델로 재학습 → 결과 예측	

cf. **부트스트래핑(Boostrapping) 분할:** 중첩 허용

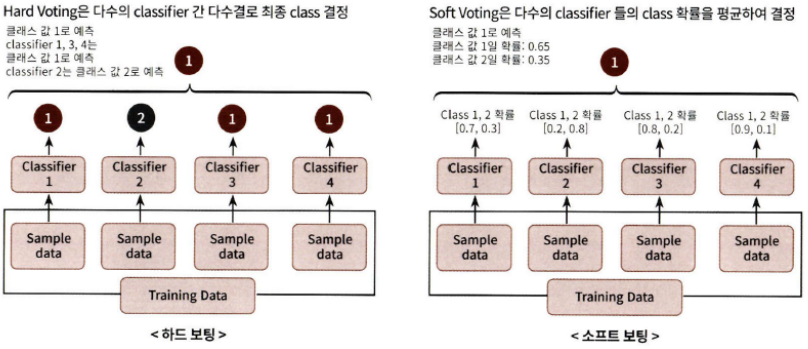
교차 검증: 중첩 허용하지 않음



Voting 방식 Bagging 방식

보팅 유형

1. 하드 보팅(Hard Voting)
   - 다수결
2. 소프트 보팅(Soft Voting)
   a. 레이블 값 결정 확률 더한다
   b. 평균을 구한다
   c. 확률 가장 높은 레이블 값 = 최종 결괏값
   - 일반적으로 적용되는 방법



보팅 분류기

- VotingClassifier

인자	설명
estimators	- list 값 - 보팅에 사용될 분류기 객체를 튜플 형식으로 입력
voting	‘hard’, ‘soft’ (기본은 ‘hard’)

```
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)
```

```
# 개별 모델은 로지스틱 회귀, KNN
lr_clf=LogisticRegression()
knn_clf=KNeighborsClassifier(n_neighbors=8)

# 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf=VotingClassifier(estimators=[('LR', lr_clf),('KNN', knn_clf)],voting='
X_train, X_test, y_train, y_test=train_test_split(cancer.data, cancer.target,

# VotingClassifier 학습/예측/평가
vo_clf.fit(X_train, y_train)
pred=vo_clf.predict(X_test)
print('Voting 분류기 정확도: {0: .4f}'.format(accuracy_score(y_test,pred)))

# 개별 모델의 학습/예측/평가
classifiers=[lr_clf, knn_clf]
for classifier in classifiers:
  classifier.fit(X_train, y_train)
  pred=classifier.predict(X_test)
  class_name=classifier.__class__.__name__
  print('{0} 정확도: {1: .4f}'.format(class_name, accuracy_score(y_test, pred)
```

- 여러 분류기를 결합한다고 예측 성능이 높아지는 것 아님
- ML 모델 평가 요소: 높은 유연성이 있는가?
  - “편향-분산 트레이드오프” 극복해야 함
    - 편향: 모델이 너무 단순해서 생기는 오류 - 과소적합(under-fitting)

- 분산: 모델이 지나치게 복잡해서 생기는 오류 - 과대적합(over-fitting)
- error = 편향 error + 분산 error + 그 외

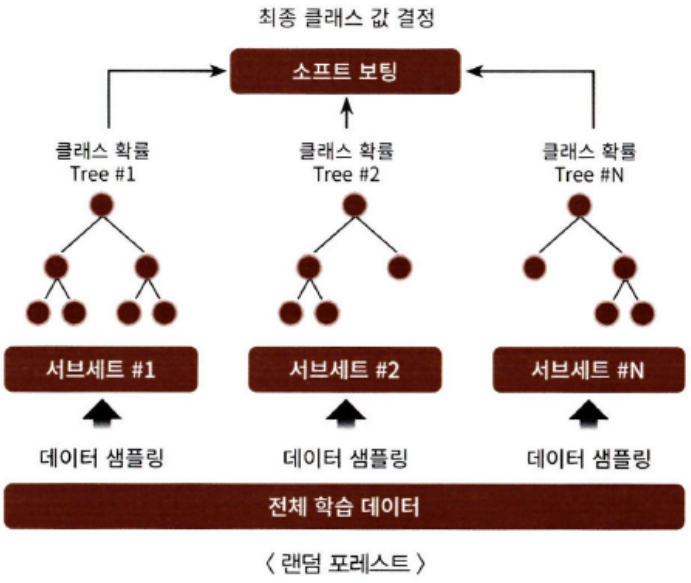
  모델 복잡함 → 편향 줄어듦, 분산 높아짐

  모델 단순함 → 분산 줄어듦, 편향 높아짐

4. 랜덤 포레스트

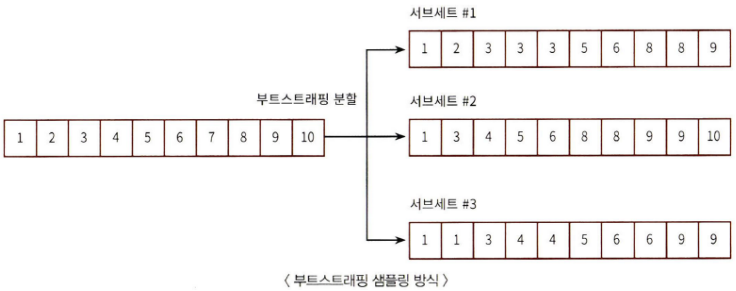
랜덤 포레스트 개요, 실습

- 기반 알고리즘: 결정 트리



〈 랜덤 포레스트 〉

1. 전체 데이터에서 샘플링한다 (**부트 스트래핑**)
   - 중첩 허용하여 샘플링
     - ex. 원본 데이터 건수 10개, n_estimators=3



〈 부트스트래핑 샘플링 방식 〉

2. 결정 트리 분류기 각각 적용하여 학습한다
   - **RandomForestClassifier**
3. 모든 분류기가 보팅한다

4. 최종 예측 값 결정한다

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

X_train, X_test, y_train, y_test=get_human_dataset()

# 랜덤 포레스트 학습/평가
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('랜덤 포레스트 정확도: {0: .4f}'.format(accuracy))
```

랜덤 포레스트 하이퍼 파라미터, 튜닝

- 하이퍼 파라미터가 많아서 시간이 많이 소모됨

이름	설명
n_estimators	- 결정 트리 개수 - 디폴트 10개 - 늘릴 수록 시간이 오래 걸림
max_features	- 분할 시 고려할 최대 피처 개수 - 디폴트 auto (=sqrt)
max_depth	- 트리 최대 깊이 - 디폴트 None - 깊이 깊어지면 과적합 가능성 증가
min_samples_leaf	- 말단 노드(leaf) 되기 위한 최소한의 데이터 수 - 비대칭적 데이터 → 특정 클래스 데이터가 극도로 작을 수 있음 ⇒ 작게 설정해야 제데로 분류됨

```
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, n_jobs=1)
grid_cv=GridSearchCV(rf_clf, param_grid=params, cv=2, n_jobs=1)
grid_cv.fit(X_train, y_train)
```

```
print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
print('최고 예측 정확도: {0: .4f}'.format(grid_cv.best_score_))
```

```
rf_clf1=RandomForestClassifier(n_estimators=300, max_depth=10, min_samples_le
                               min_samples_split=8, random_state=0)
rf_clf1.fit(X_train, y_train)
pred=rf_clf1.predict(X_test)
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

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

ftr_importances_values=rf_clf1.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 Top20')
sns.barplot(x=ftr_top20, y=ftr_top20.index)
plt.show()
```