▼ Chap4. 분류

분류(Classification)의 개요

지도학습 - 명시적인 정답(레이블)이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식

분류는 지도학습의 대표적인 유형

> 💡 **분류란?**
> → 학습 데이터로 주어진 데이터의 피쳐와 레이블값(결정 값, 클래스 값)을 머신러닝 알고리즘으로 학습해 모델을 생성하고, 생성된 모델에 새로운 데이터 값이 주어졌을 때 **미지의 레이블 값을 예측**하는 것

→ 기존 데이터가 어떤 레이블에 속하는지 패턴을 알고리즘으로 인지한 뒤 새롭게 관측된 데이터에 대한 레이블을 판별하는 것

- 앙상블(Ensemble) - 서로 다른(또는 같은) 머신러닝 알고리즘을 결합

  → 일반적으로 배깅(Bagging)과 부스팅(Boosting) 방식으로 나뉨

  → 배깅 방식의 대표격인 랜덤 포레스트(Random Forest)는 뛰어난 예측 성능, 빠른 수행 시간 등의 장점을 갖고 있음

  → 부스팅 방식의 효시인 그래디언트 부스팅(Gradient Boosting)의 경우 뛰어난 예측 성능을 갖고 있지만 수행 시간이 너무 오래 걸려 최적화 모델 튜닝이 어려움

  → XgBoost(eXtra Gradient Boost)와 LightGBM 등 기존 그래디언트 부스팅의 예측 성능을 한 단계 발전시키면서 수행 시간을 단축시키는 알고리즘이 계속해서 등장

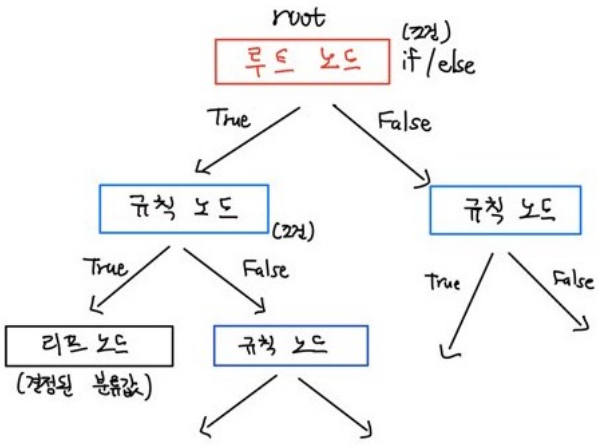
  → 앙상블의 기본 알고리즘으로 **결정 트리**를 일반적으로 사용함

결정 트리

→ 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리(Tree)기반의 분류 규칙을 만드는 것

→ 데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가가 알고리즘의 성능을 크게 좌우함.

- 결정트리의 구조



→ 데이터 세트의 피쳐가 결합해 규칙 조건을 만들 때마다 규칙 노드가 만들어짐

→ 많은 규칙이 있을수록 분류를 결정하는 방식이 더욱 복잡해짐(과적합)

→ 트리의 깊이(depth)가 깊어질수록 결정 트리의 예측 성능이 저하될 가능성이 높음

→ 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수있도록 결정 노드의 규칙이 정해져야 함. (어떻게 **분할, split 할 것인가**가 중요)

지니계수 : 지니 계수가 낮을수록 데이터 균일도가 높은 것으로 해석해 지니 계수가 낮은 속성을 기준으로 분할함.

→ DecisionTreeClassifier는 기본으로 지니 계수를 이용해 데이터 세트를 분할함

→ 지니 계수가 낮은 조건을 찾아서 자식 트리 노드에 걸쳐 반복적으로 분할한 뒤 데이터가 모두 특정 분류에 속하게 되면 분학을 멈추고 분류를 결정함.

- 결정 트리 모델의 특징

장점 1. 정보의 '균일도'라는 룰을 기반으로 하고 있어 쉽고 직관적이다

장점 2. 피쳐의 스케일링이나 정규화 등의 사전 가공 영향도가 크지 않음

단점. 과적합으로 인해 알고리즘 성능이 떨어지고, 이를 극복하기 위해 트리의 크기를 사전에 제한하는 튜닝이 필요하다.

- **결정 트리 파라미터**

→ 분류를 위한 DecisionTreeClassifier

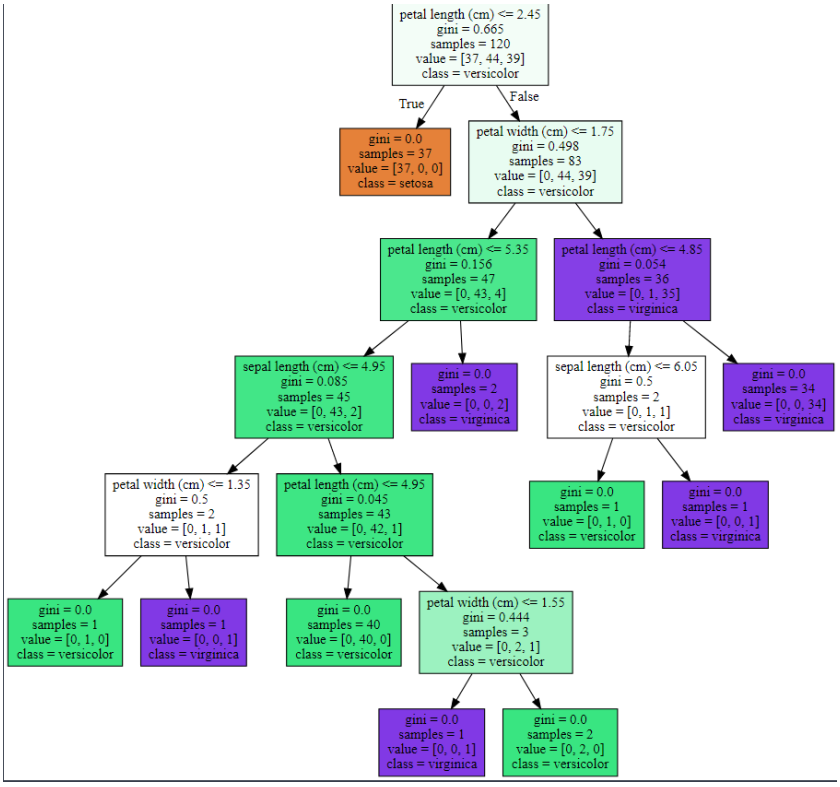
→ 회귀를 위한 DecisionTreeRegressor

→ 사이킷런의 결정 트리 구현은 CART(Classification And Regression Trees) 알고리즘 기반

1. **min_samples_split** : 노드를 분할하기 위한 최소한의 샘플 데이터 수(과적합을 제어하기 위함), default=2, 작게 설정할수록 분할되는 노드가 많아져 과적합 가능성 증가
2. **min_samples_leaf** : 말단 노드(Leaf)가 되기 위한 최소한의 샘플 데이터 수(과적합 제어 용도), 비대칭적 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우에는 작게 설정하는 것이 필요함.
3. **max_features** : 최적의 분할을 위해 고려할 최대 피쳐 개수, default=None(데이터 세트의 모든 피쳐 사용)
4. **max_depth** : 트리의 최대 깊이를 규정, default=None(완벽하게 클래스 결정 값이 될 때까지 깊이를 계속 키우며 분할 → 깊이가 깊어지면 min_samples_split 설정대로 최대 분할하여 과적합될 수 있어 적절한 값으로 제어 필요
5. **max_leaf_nodes** : 말단 노드의 최대 개수

- 결정 트리 모델의 시각화

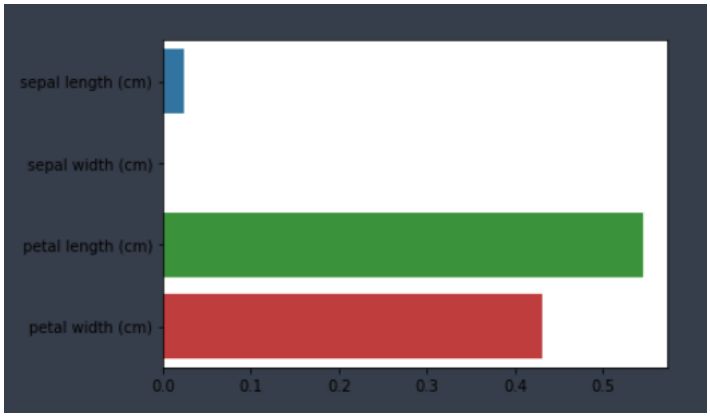
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

#DecisionTree Classifier 생성
dt_clf = DecisionTreeClassifier(random_state=123)

#붓꽃 데이터 로딩 및 학습, 테스트 데이터로 분리
iris_data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target, test_size=0.2, random_state=123)

#DecisionTreeClassifier 학습
dt_clf.fit(X_train, y_train)

from sklearn.tree import export_graphviz

#export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일 생성
export_graphviz(dt_clf, out_file='tree.dot', class_names=iris_data.target_names, feature_names=iris_data.feature_names, impurity=T

import graphviz
#위에서 생성한 tree,dot 파일을 Graphviz가 읽어서 주피터 노트북상에서 시각화
with open('tree.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



feature_importances_ 속성을 통해 피쳐의 중요한 역할 지표를 나타낼 수 있다.

→ ndarray로 값 반환, 값이 높을수록 해당 피쳐의 중요도가 높다는 의미

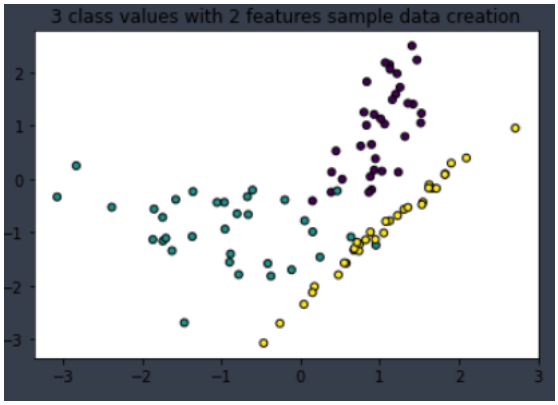
```
#피쳐별 중요도
import seaborn as sns
import numpy as np
%matplotlib inline

#feature importance 추출
print('Feature importance:',np.round(dt_clf.feature_importances_,3))

#feature별 importance 매핑
for name, value in zip(iris_data.feature_names, dt_clf.feature_importances_):
    print(name, ':', value)

#feature importance를 column 별로 시각화하기
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_names)
```



→ 여러 피쳐들 중 petal_length가 가장 피쳐 중요도가 높음을 알 수 있음.

- 결정 트리 과적합(Overfitting)

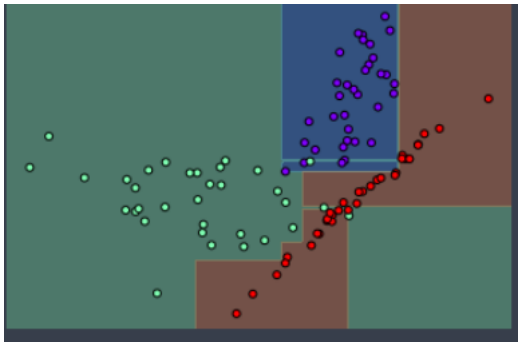
```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

plt.title('3 class values with 2 features sample data creation')

#2차원 시각화를 위해 피쳐는 2개, 클래스는 3가지 유형의 분류 샘플 데이터 생성
X_features, y_labels = make_classification(n_features=2, n_redundant=0, n_informative=2,
                                           n_classes=3, n_clusters_per_class=1, random_state=123)

#그래프 형태로 2개의 피쳐로 2차원 좌표 시각화, 각 클래스 값은 다른 색깔로 표시
plt.scatter(X_features[:,0], X_features[:,1], marker='o', c=y_labels, s=25, edgecolors='k')
```



→ 결정 트리의 하이퍼 파라미터를 디폴트 값으로 설정

→ visualize_boundary() 사용 (머신러닝 모델이 클래스 값을 예측하는 결정 기준을 색상과 경계로 나타내 모델이 어떻게 데이터 세트를 예측 분류하는지 잘 이해할 수 있도록 해줌, 유틸리티 함수)

```
from sklearn.tree import DecisionTreeClassifier

#특정한 트리 생성 제약 없는 결정 트리의 학습과 결정 경계 시각화
dt_clf = DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```

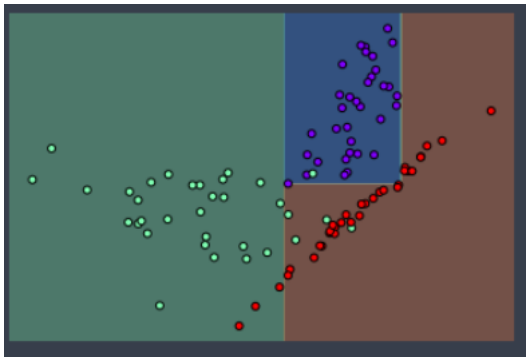


→ 일부 Outlier 데이터까지 분류하기 위해 분할이 자주 일어나 결정 기준 경계가 매우 많아짐

→ 복잡한 모델은 학습 데이터 세트의 특성과 약간만 다른 형태의 데이터 세트를 예측하면 예측 정확도가 떨어진다! (overfitting)

➕min_samples_leaf = 6 로 하이퍼 하라미터 변경

```
#min_samples_leaf = 6 설정
#6개 이하의 데이터는 리프 노드를 생성할 수 있도록 리프 노드 생성 규칙 완화
dt_clf = DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```



→ 아까보다 Outlier에 크게 반응하지 않고 좀 더 일반화ㅚㄴ 분류 규칙에 따라 분류됐음

→ 테스트 데이터 세트는 학습 데이터 세트와는 다른 데이터 세트이므로, 학습 데이터에만 지나치게 최적화된 분류 기준은 오히려 테스트 데이터 세트에서 정확도를 떨어뜨릴 수 있다.

앙상블 학습

→ 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법 aka 집단지성

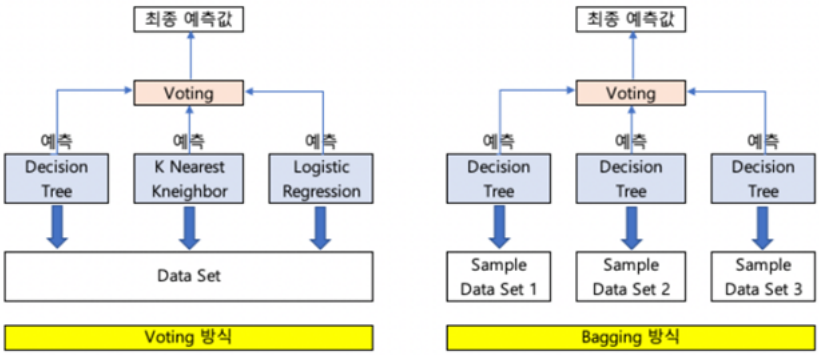
→ 앙상블 학습의 유형: 보팅(Voting), 배깅(Bagging), 부스팅(Boosting), 스태깅 등

→ 보팅과 배깅은 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식

🧙보팅과 배깅의 차이점

→ 보팅은 일반적으로 서로 다른 알고리즘을 가진 분류기를 결합하는 것

→ 배깅은 각각의 분류기가 모두 같은 유형의 알고리즘 기반이지만 데이터 샘플링을 서로 다르게 가져가면서 학습을 수행해 보팅을 수행하는 것 ex) 랜덤 포레스트 알고리즘

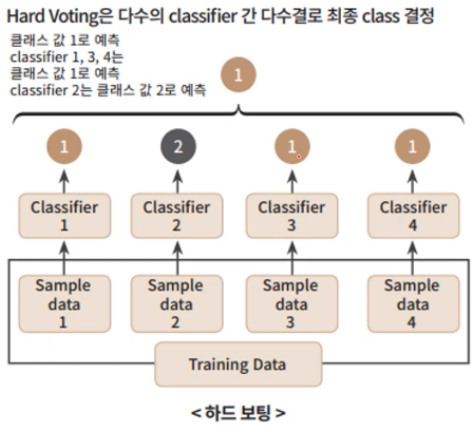


- 보팅 유형 - 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting)

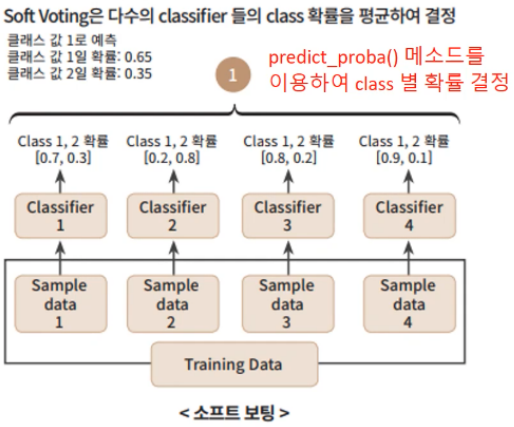
→ 하드 보팅은 다수결 원칙과 비슷 : 예측한 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정하는 것

→ 소프트 보팅 : 분류기들의 레이블 값 결정 확률을 모두 더하고 이를 평균해서 이들 중 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정

→ 일반적으로 **소프트 보팅 방법으로 선정**됨.



<하드 보팅>



<소프트 보팅>

- 보팅 분류기(Voting Classifier)

→ 사이킷런은 보팅 방식의 앙상블을 구현한 VotingClassifier 클래스를 제공함

보팅 방식의 앙상블을 이용해 위스콘신 유방암 데이터 세트 예측 분석

❕ 위스콘신 유방암 데이터 세트는 유방암의 악성종양, 양성종양 여부를 결정하는 이진 분류 데이터 세트, 많은 피쳐 有, load_breast_cancer( ) 함수를 통해 데이터 세트 생성 가능

```
#위스콘신 데이터 세트
import pandas as pd

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression #로지스틱회귀
from sklearn.neighbors import KNeighborsClassifier #KNN
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)
```

사이킷런은 VotingClassifier 클래스를 이용해 보팅 분류기를 생성할 수 있음

→ 주요 생성 인자: estimators(리스트 값으로 보팅에 사용될 여러 개의 Classifier 객체들을 튜플 형식으로 입력 받음), voting(디폴트는 hard, soft 선택시 소프트 보팅 방식 적용)

```
#개별 모델은 로지스틱 회귀와 KNN임
lr_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=8)

#개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf = VotingClassifier(estimators=[('LR', lr_clf), ('KNN', knn_clf)], voting='soft')

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2, random_state=123)

#VotingClassifier 학습, 예측, 평가
vo_clf.fit(X_train, y_train)
pred = vo_clf.predict(X_test)
print('Voting 분류기 정확도:', 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(class_name,'정확도:',accuracy_score(y_test,pred))
```

```
Voting 분류기 정확도: 0.9736842105263158
LogisticRegression 정확도: 0.9824561403508771
KNeighborsClassifier 정확도: 0.9649122807017544
```

→ 보팅으로 여러 개의 분류기를 결합한다고 해서 무조건 기반 분류기보다 **예측 성능이 향상되지는 않음**

→ **BUT 보팅을 포함해 배깅과 부스팅 등의 앙상블 방법은 전반적으로 다른 단일 머신러닝 알고리즘보다 뛰어난 예측 성능을 가지는 경우가 많다**.

→ 앙상블 학습에서는 매우 많은 분류기를 결합해 다양한 상황을 학습하게 함으로써 편향-분산 트레이드오프의 효과를 극대화할 수 있다.

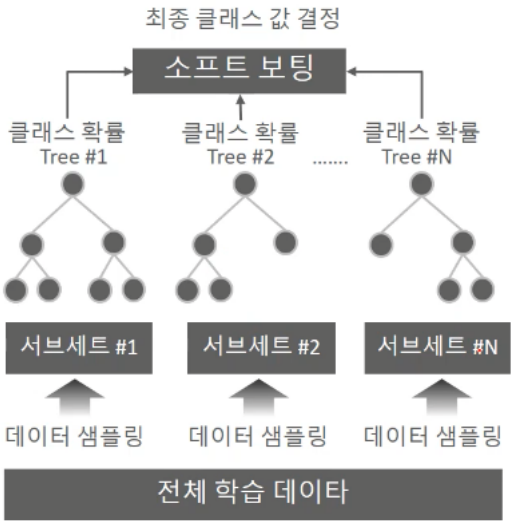
랜덤 포레스트

배깅 : 같은 알고리즘으로 여러 개의 분류기를 만들어 보팅으로 최종 결정하는 알고리즘

→ 배깅의 대표적인 알고리즘이 **랜덤 포레스트!**

1. 랜덤 포레스트는 앙상블 알고리즘 중 비교적 빠른 수행속도를 갖고 있음.
2. 다양한 영역에서 높은 예측 성능을 보임
3. 랜덤 포레스트의 기반 알고리즘 → 결정 트리 (쉽고 직관적인 장점)

⇒ 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측을 결정함.



→ 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 **일부가 중첩**되게 샘플링된 데이터 세트

⇒ 부트스트래핑(bootstrapping) 방식

데이터가 중첩된 개별 데이터 세트에 결정 트리 분류기를 각각 적용하는 것이 랜덤 포레스트!

사이킷런은 RandomForestClassifier 클래스를 통해 랜덤 포레스트 기반의 분류 지원

- 랜덤 포레스트 하이퍼 파라미터 및 튜닝

트리 기반의 앙상블 알고리즘의 단점은 많은 하이퍼 파라미터와 튜닝을 위한 많은 시간 소요

1. n_estimators : 랜덤 포레스트에서 결정 트리의 개수 지정, default = 10, 많이 설정할수록 좋은 성능 기대 가능, but 늘릴수록 학습 수행 시간이 오래 걸린다.

2. max_features : 결정 트리에 사용된 max_features 파라미터와 같음 but default가 None이 아닌 auto(sqrt)와 같음 → 트리를 분할하는 피쳐를 참조할 때 전체 피쳐가 아니라 sqrt(전체 피쳐 개수)만큼 참조한다.
3. max_depth, min_samples_leaf → 과적합 개선 가능