4️⃣

04 _ 분류 part1

1. 분류

- 지도학습 : 명시적인 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식

→ 학습 데이터로 주어진 데이터의 피처와 레이블 값(결정 값, 클래스 값)을 머신러닝 알고리즘으로 학습해 모델 생성, 생성된 모델에 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측하는 것

분류의 알고리즘

1. 나이브 베이즈 : 베이즈 통계와 생성 모델에 기반
2. 로지스틱 회귀 : 독립변수와 종속변수의 선형 관계성 기반
3. 결정 트리 : 데이터 균일도에 따른 규칙 기반
4. 서포트 벡터 머신 : 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 역할
5. 최소 근접 (Nearest Neighbor) : 근접 거리를 기반 (KNN으로 많이 사용)
6. 신경망 : 심층 연결 기반
7. 앙상블 : 서로 다른(or 같은) 머신러닝 알고리즘 결합

앙상블 기법

- 분류에서 가장 각광을 받는 분야 중 하나, 정형 데이터의 예측 분석 영역에서는 매우 높은 예측 성능을 자랑함
- 앙상블의 분류
  - 배깅 : 랜덤 포레스트가 대표적
  - 부스팅 : Gradient Boosting, XGBoost, LightGBM등이 대표적, 기존보다 수행 성능을 높이면서, 수행시간은 단축시키는 알고리즘이 계속 등장
- 결정 트리 : 앙상블의 기본 알고리즘으로 일반적으로 사용함.

- 장점 : 쉽고, 유연함, 사전 데이터 가공의 영향이 매우 적음
- 단점 : 예측 성능을 향상시키기 위해서는 복잡한 구조를 가져야 하는데, 이로 인해서 오히려 과적합이 발생할 수 있음.

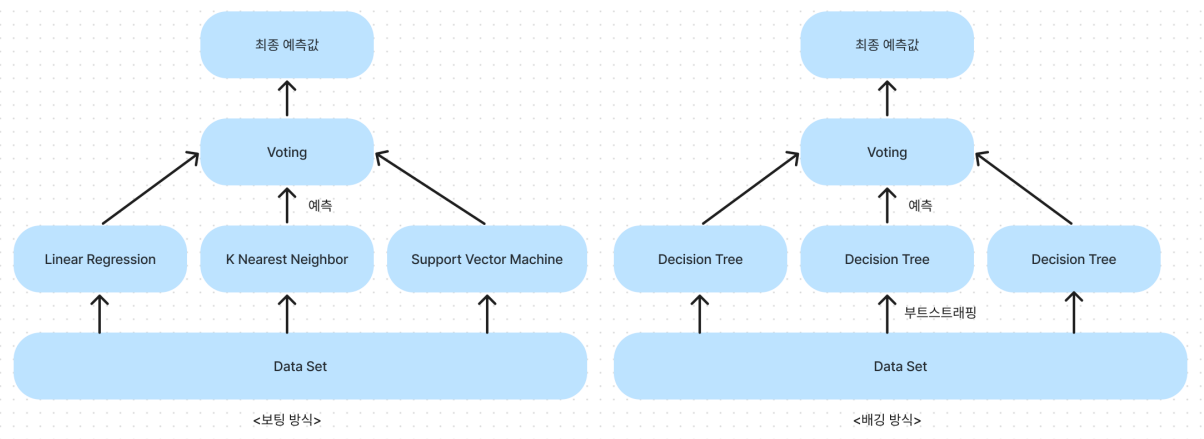
→ 그런데, 앙상블에서는 이것이 오히려 장점으로 연결됨

앙상블의 개념 : 매우 많은 여러개의 약한 학습기 (예측이 상대적으로 떨어지는 학습 알고리즘)을 결합해 확률적 보완, 오류가 발생한 부분에 대한 가중치를 계속 업데이트하면서 예측 성능 보완

2. 결정 트리

: 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만든다.

→ (조건문) 따라서, 데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가가 알고리즘 성능을 좌우한다.



- 깊이(depth)가 깊어질 수록 예측 성능이 저하될 가능성이 커진다.
  - 많은 규칙 존재 → 분류를 결정하는 방식이 복잡해진 것 → 과적합될 가능성 높음
- 데이터를 분류할 때 최대한 많은 데이터가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 한다.(최대한 균일한 데이터세트를 구성할 수 있도록 분할 필요)
- 결정 노드는 정보 균일도가 높은 데이터 세트를 먼저 선택할 수 있도록 규칙 조건을 만듦
- 정보 균일도를 측정하는 방법

  1. 정보이득 Information Gain
     - 엔트로피 기반 : 주어진 데이터 집합의 혼잡도. 서로 다른 값이 섞여있으면 엔트로피 값 증가

- 정보이득 지수 = 1 - 엔트로피 지수
- 결정트리는 정보이득 지수로 분할. 즉, 정보이득이 높은 속성을 기준으로 분할
- 정보이득=부모의불순도−(부모의샘플수왼쪽노드샘플수*왼쪽노드불순도)−(부모의샘플수오른쪽노드샘플수*오른쪽노드불순도)

2. 지니계수
    - 0이 가장 평등하고 1로 갈수록 불평등
    - 지니계수가 낮을수록 데이터터 균일도가 높은 것으로 해석. 지니계수가 낮은 속성을 기준으로 분할
        - 그러면 정보이득 지수 같은 지표를 만들자면 = 1 - 지니계수 (?)
    - 지니불순도=1−음성클래스비율2+음성클래스비율2
    - 사이킷런 DecisionTreeClassifier은 기본으로 지니계수를 이용해 데이터 분할

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

결정 트리 모델의 특징

- 장점 : 정보의 '균일도' 라는 룰을 기반으로 하고 있어 알고리즘이 직관적, 균일도만 신경쓰면 되므로 각 피처의 스케일링과 정규화 같은전처리 작업이 (일반적인 경우) 필요 없음
- 단점 : 과적합으로 (테스트) 정확도가 떨어짐 (트리가 계속 깊어질수록)
    - (학습) 모델의 정확도를 높이기 위해 계속 조건을 추가하며 트리 깊이가 깊어지면, 테스트 정확도가 떨어질 것임
    - 오히려 완벽한 규칙을 만들 수 없다고 먼저 인정하고, 트리의 크기를 사전에 제한하는 것이 오히려 성능 튜닝에 더 도움이 될 것

결정 트리의 파라미터

- 사이킷런 결정 트리 = DesicionTreeClassifier (for 분류) , DisicionTreeRegressor (for 회귀)
    - 결정 트리 구현은 CART (Classfication And Regression Trees) 알고리즘 기반 ⇒ 분류뿐 아니라 회귀에서도 사용될 수 있음
    - 하위의 파라미터는 동일함

- `min_samples_split`
  - 노드를 분할하기 위한 최소한의 샘플 데이터 수
  - 디폴트 = 2
  - 작게 설정할 수록 분할되는 노드가 많아진다. (과적합 가능성 증가)
- `min_samples_leaf`
  - 말단 노드(leaf)가 되기 위한 최소한의 샘플 데이터 수
  - 비대칭적 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우는 작게 설정 필요
- `max_features`
  - 최적의 분할을 위해 고려할 최대 피처 개수
  - 디폴트 = None : 전체 피처 선정
  - int 형으로 지정 : 대상 피처의 개수
  - float 형으로 지정: 전체 피처 중 대상 피처의 퍼센트
  - 'sqrt' : 전체피처개수 만큼 선정 (='auto')
  - 'log' : *log*2(전체피처개수) 만큼 선정
  - 'None' : 전체 피처 선정
- `max_depth`
  - 트리의 최대 깊이 규정
  - 디폴트 = None : 완벽하게 클래스 결정 값이 될 때까지 깊이를 계속 키우며 분할하거나, 노드가 가지는 데이터 갯수가 min_samples_split보다 작아질 대까지 계속 깊이를 증가시킴
  - 깊이가 깊어지면 min_samples_split 설정대로 초대분할하여 과적합할 수 있으므로 적절한 값으로 제어 필요
- `max_leaf_nodes`
  - 말단 노드의 최대 개수

결정 트리 모델의 시각화

```
from sklearn.tree import export_graphviz
```

```
export_graphviz(estimator, out_file='파일명', class_names=클래스 명칭,
                                feature_names=피처명칭, impurity=True, filled=True)
```

```
# 위에서 생성한 파일을 graphviz가 읽어서 주피터 노트북 상에서 시각화
import graphviz

with open ('파일명') as f: dot_graph = f.read()
```

```
graphviz.Source(dot_graph)
```

- 시각화 결과의 구성들 :
  - 조건 : 피처 조건이 있는 것은 자식 노드를 만들기 위한 조건 규칙, 이게 없으면 리프 노드
  - gini(지니계수) : value=로 주어진 데이터 분포에서의 지니 계수
  - samples : 현 규칙에 해당하는(적용되는) 데이터 건수
  - value=: 클래스 값 기반의 데이터 건수
  - 색상 : 레이블 값, 선명도가 높을 수록 지니계수 낮음
- `feautre_importance_ 속성` : ndarray로 반환, 피처 순서대로 값 할당, 값이 높을 수록 중요도 높음 : `estimator.feature_importance_`

  ```
  import seaborn as sns
  import numpy as np
  %matplotlib inline

  # feature importance 추출
  print("Feature importances:\n{0}".format(np.round(dt_clf.feature_importances_, 3)))

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

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

- make_classification(): 분류를 위한 테스트용 데이터를 쉽게 만들 수 있다.

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

plt.title("3 Class values with 2 Features Sample data cr
eation")

# 2차원 시각화를 위해서 feature는 2개, 결정값 클래스는 3가지 유형
의 classification 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=0)

# plot 형태로 2개의 feature로 2차원 좌표 시각화, 각 클래스값은 다
른 색깔로 표시됨.
plt.scatter(X_features[:, 0], X_features[:, 1], marker
='o', c=y_labels, s=25, cmap='rainbow', edgecolor='k')
```

- 반환되는 객체: 피처 데이터 세트, 클래스 레이블 데이터 세트
- 예시: 피처 2개, 클래스 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=0)
```

- visualize_boundary(): 머신러닝 모델이 만든 결정 기준을 색상과 경계로 나타낸다.

```
visualize_boundary(estimator, X_features, y_labels)
```

3. 앙상블

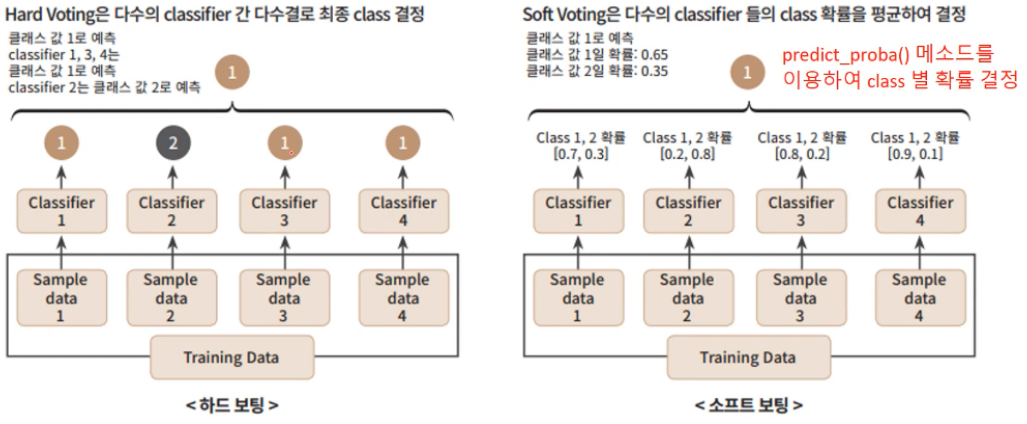
: 여러 개의 분류기(classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

→ 보팅Voting, 배깅Bagging, 부스팅Boosting + 스태킹Stacking

- 보팅 : 서로 다른 알고리즘을 가진 분류기 결합
- 배깅 : 같은 유형의 알고리즘을 가진 분류기를 사용하지만, 데이터 샘플링을 다르게 가져감
    - 부트스트래핑 Bootstrapping: 개별 분류기에게 데이터를 샘플링해서 추출하는 방식
    - 데이터 세트 간의 중첩 허용 (cf. 교차검증은 중복 불허)
        - ex) 10,000개의 데이터를 10개의 분류기가 배깅 방식으로 나눠도, 각 1,000개의 데이터 내에서 중복된 데이터가 있을 수 있음
- 부스팅 : 여러 개의 분류기가 순차적으로 학습을 수행하되, 앞의 분류기의 틀린 예측에 대해서 다음 분류기에는 가중치를 부여하면서 학습과 예측을 진행하는 방식
    - 대표적인 모듈: 그래디언트 부스트, XGBoost, LightGBM
- 스태킹 : 여러 가지 다른 모델의 예측 결과값을 "다시 학습 데이터로 만들어서" 다른 모델(메타 모델)로 재학습, 예측하는 방법

보팅

1. 유형



1. 하드 보팅 Hard Voting: 다수결원칙. 예측 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정

2. 소프트 보팅 Soft Voting : 분류기들의 레이블 값 결정 확률을 모두 더하고 이를 평균해서, 이들 중 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정(일반적으로 소프트 보팅 사용)

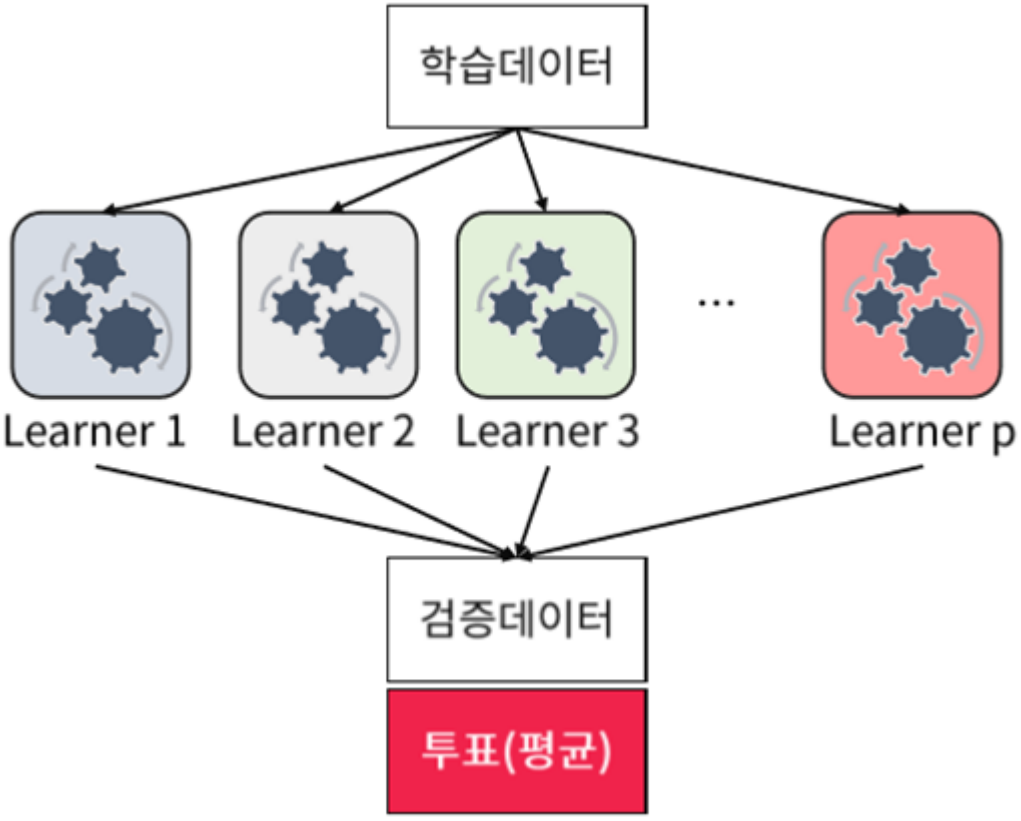
2. 사용 (보팅 분류기)

```
from sklearn.ensemble import VotingClassifier
# 로지스틱 회귀, KNN 기반 소프트 보팅 방식 분류기 만들기
vo_clf = VotingClassigier(estimator = [('LR', lr_clf),
('KNN', knn_clf)], voting='soft')
```

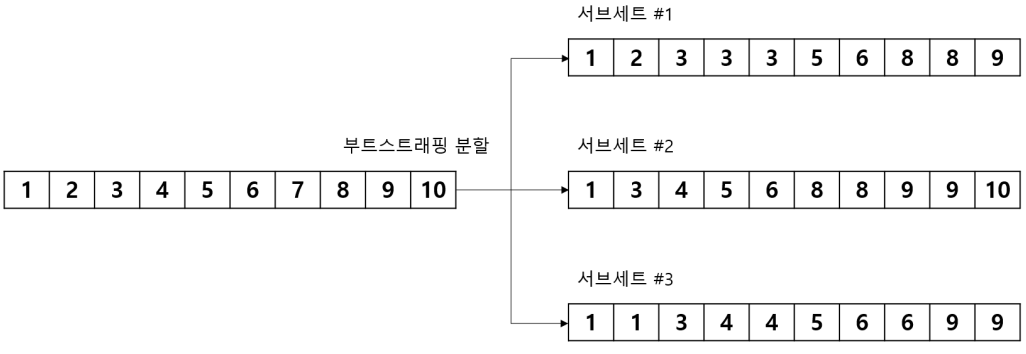
4. 랜덤 포레스트

랜덤 포레스트 개요 및 실습

- 랜덤 포레스트는 배깅의 대표적인 알고리즘
- 기반 알고리즘 : 결정 트리 → 결정 트리의 장점을 그대로 가지고 감
- 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤, 최종적으로 모든 분류기가 보팅을 통해 예측 결정



- 소프트 보팅 방식을 통해 최종 결정
- 랜덤 포레스트는 개별적인 분류기의 기반 알고리즘은 결정 트리이지만, 개별 트리가 학습하는 데이터 세트는 전체 데이터의 일부가 중첩되게 샘플링된 데이터 세트 (데이터셋들 중 조금씩은 같은 데이터 사용)
- 부트스트래핑 분할 방식 : 여러 개의 데이터 세트를 중첩되게 분리하는 방식



< 부트스트래핑 샘플링 방식 >

→ 데이터가 중첩된 개별 데이터 세트에 결정 트리 분류기를 각각 적용하는 것이 랜덤 포레스트라고 할 수 있음

랜덤 포레스트 하이퍼 파라미터 및 튜닝

- n_estimators : 랜덤 포레스트에서 결정 트리의 개수 지정, 많이 설정할수록 좋은 성능이지만, 무조건적인 성능 증가는 아님
- max_features : 최적의 분할을 위해 고려할 최대 피처 개수 (단, 기본값은 None가 아닌 auto, 즉 sqrt와 같음)
- 그 외의 파라미터는 결정 트리와 같음
- 결정 트리와 마찬가지로 feature*importance*를 통해 알고리즘이 선택한 피처의 중요도를 알 수 있음.