3주차 예습 과제

4.1 분류의 개요

- 지도학습: 명시적인 정답(=레이블)이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식
- 분류: 학습 데이터를 머신러닝 알고리즘으로 학습해 모델을 생성하고, 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측하는 것
- 분류를 구현할 수 있는 머신러닝 알고리즘:
  - 나이브 베이즈
  - 로지스틱 회귀
  - 결정 트리
  - 서포트 벡터 머신
  - 최소 근접 알고리즘
  - 신경망
  - 앙상블

- 앙상블: 서로 다른/또는 같은 알고리즘을 결합한 알고리즘
  - 정형 데이터의 예측 분석에서 높은 예측 성능을 가짐
  - 배깅: 높은 예측 성능, 빠름, 유연성 좋음 → ex)랜덤 포레스트
  - 부스팅: 수행 시간이 오래 걸리지만 수행 시간을 단축시킨 알고리즘이 등장 중임 → ex) 그래디언트 부스팅, XgBoost, LightGBM
- 앙상블의 기본 알고리즘으로는 일반적으로 결정 트리가 사용됨

  (+) 쉽고 유연, 사전 가공의 영향이 적음

  (-) 예측 성능을 향상시키기 위해서는 복잡한 구조를 가져야 함

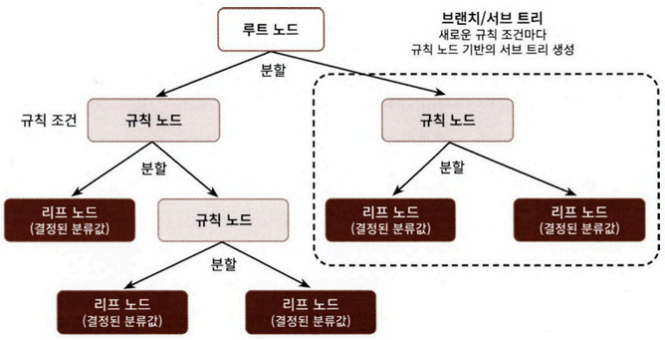
  (-) 오히려 과적합이 발생해 성능이 저하될 수도

  → 앙상블은 여러 개의 약한 학습기를 결합하기 때문에 적합함

4.2 결정 트리

- 데이터에 있는 규칙을 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것

- 규칙 노드(=결정 노드): 규칙 조건이 되는 것
- 리프 노드: 결정된 클래스 값
- 서브 트리: 새로운 규칙 조건마다 생성됨



- 규칙 노드는 균일도가 높은 데이터 세트를 먼저 선택하도록 조건을 만듦
- 정보의 균일도를 측정하는 방법:
  - 정보 이득 지수: 1-엔트로피 지수, 엔트로피는 데이터 집합의 혼잡도를 의미 ⇒ 정보 이득이 높은 속성을 기준으로 분할
  - 지니 계수: 낮을수록 데이터 균일도가 높음
    ⇒ 지니 계수가 낮은 속성을 기준으로 분할
  - DecisionTreeClassifier는 지니 계수로 데이터 세트를 분할함

(+): 균일도라는 명확한 룰을 기반으로 하여 알고리즘이 쉽고 직관적

(+): 각 피처의 스케일랑과 정규화 같은 전처리 작업이 필요 없음

(-): 과적합으로 정확도가 떨어짐 → 튜닝 필요

- 사이킷런의 결정 트리 알고리즘 구현 클래스:
  - DecisionTreeClassifier: 분류를 위한 클래스
  - DecisionTreeRegressor: 회귀를 위한 클래스

파라미터

- min_samples_split:
  - 노드를 분할하기 위한 최소한의 샘플 데이터 수 (default=2)
- min_samples_leaf:
  - 분할 시 양쪽 브랜치 노드에서 가져야 할 최소한의 샘플 데이터 수

    - 특정 클래스의 데이터가 극도로 작을 때 주의
- max_features:
    - 최대 피처 개수
    - int형: 대상 피처 개수
    - float형: 대상 피처의 퍼센트
    - sqrt/auto: $\sqrt{\text{전체피처개수}}$만큼으로 선정
    - None: 전체 피처 선정 (default)
- max_depth:
    - 트리의 최대 깊이 규정
    - None: 완벽히 분할될 때까지 깊이 증가시킴 (default)
- max_lead_nodes:
    - 말단 노드의 최대 개수

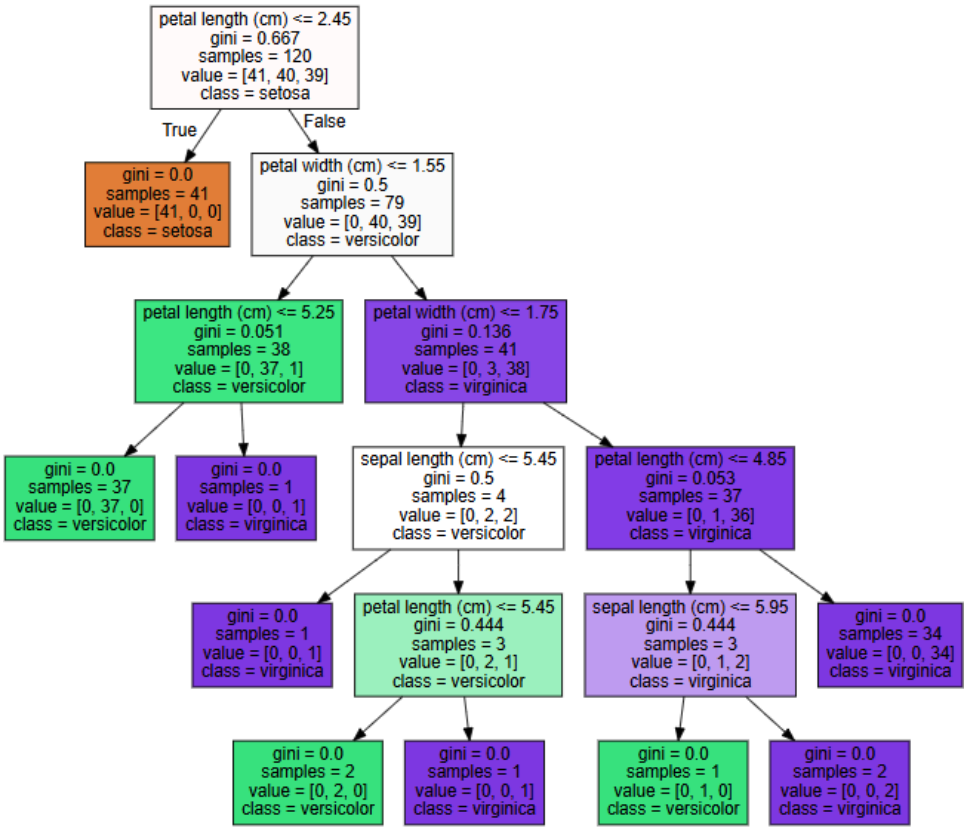
결정 트리 모델의 시각화

- Graphiz 패키지: 원래 그래프 기반의 dot 파일로 기술된 다양한 이미지를 시각화
- export_graphviz(): 결정 트리를 Graphviz 형식(.dot 파일)으로 변환해주는 함수

  → (학습이 완료된 estimator, 피처 이름 리스트, 레이블 이름 리스트)를 입력하면 학습된 결정 트리 규칙을 트리 형태로 시각화해줌

```
# classifier 생성
dt_clf=DecisionTreeClassifier(random_state=156)

# 데이터 로딩, 학습/테스트 데이터 세트 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=11)

# 학습
dt_clf.fit(X_train,y_train)

# tree.dot 파일을 생성
from sklearn.tree import export_graphviz
```

```
export_graphviz(dt_clf, out_file='tree.dot',class_names=iris_data.target_na
mes, feature_names=iris_data.feature_names,impurity=True,filled=True)

# 생성된 tree.dot 파일을 Graphiz가 읽어서 시각화
import graphviz
with open('tree.dot') as f:
    dot_graph=f.read()
graphviz.Source(dot_graph)
```



- 맨 윗줄 조건: 해당 노드에서 데이터를 분할하는 기준 (조건이 없으면 리프 노드)
- gini: 지니 계수
- samples: 현재 노드에 속한 데이터 건수
- value=[ ]: 각 클래스 별 데이터 수
- class: 가장 많은 클래스 값
- 색깔: 레이블 값에 따라 정해짐, 지니 계수가 낮을 수록 색이 짙어짐

- dt_clf.feature_importances_: ndarray 형태로 피처 별 중요도 반환

- make_classification(): 분류를 위한 테스트용 데이터 생성

```
'''
피처 2개, 중복되는 특성 수 0개, 클래스 분류에 이용되는 특성 2개,
클래스 3가지, 클래스 별 클러스터 수 1개의 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)
```

```
# DecisionTreeClassifier를 이용한 분류 수행 (디폴트 하이퍼 파라미터)
dt_clf = DecisionTreeClassifier(random_state=156)
dt_clf.fit(X_trainz y_train)
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('결정 트리 예측 정확도:{0:.4f}'.format(accuracy))
print('기본 하이퍼 파라미터:',dt_clf.get_params())
```

```
# GridSearchCV를 이용해 하이퍼 파라미터 튜닝
params={'max_depth':[8,12,16,20],'min_samples_split':[16,24]}
grid_cv=GridSearchCV(dt_clf,param_grid=params,scoring='accuracy',cv=
5,verbose=1)
grid_cv.fit(X_train,y_train)
print('최고 평균 정확도 수치:{0:.4f}'.format(grid_cv.best_score_))
print('최적 하이퍼 파라미터:',grid_cv.best_params_)

best_df_clf=grid_cv.best_estimator_
pred1=best_df_clf.predict(X_test)
accuracy1=accuracy_score(y_test,pred1)
print('결정 트리 예측 정확도:{0:.4f}'.format(accuracy1))
```

4.3 앙상블 학습

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

1. 보팅: **서로 다른** 알고리즘을 가진 분류기를 결합하여 투표를 통해 최종 예측 결과를 결정

a. 하드 보팅: 다수결 원칙

b. 소프트 보팅: 레이블 값 결정 **확률을 평균**해서 확률이 가장 높은 레이블 값을 선정
→ 일반적으로 예측 성능이 더 좋음

2. 배깅: **같은 유형**의 알고리즘을 가진 분류기를 **데이터 샘플링을 서로 다르게** 해서 학습시킨 후 투표를 통해 최종 예측 결과를 결정
(ex. 랜덤 포레스트 알고리즘)
   - 부트스트래핑 분할 방식으로 개별 Classifier에게 데이터를 샘플링함
   - **중첩을 허용**하면서 랜덤 샘플을 뽑는 것
3. 부스팅: 여러 개의 분류기가 순차적으로 학습을 수행, 앞에서 학습한 분류기가 틀린 데이터를 맞출 수 있도록 다음 분류기에서는 **가중치를 부여**하면서 학습함
(ex. 그래디언트 부스트, XGBoost, LightGBM)
4. 스태킹: 여러 모델의 예측 결괏값을 다시 학습 데이터로 만들어서 다른 모델(메타 모델)로 재학습시킴

- 사이킷런에서는 VotingClassifier 클래스를 통해 보팅 앙상블을 지원
  - estimators: 리스트 값 입력, 사용될 여러 Classifier 객체들을 ('종류', 객체이름)의 형식으로 입력받음
  - voting: 'hard' / 'soft' (default='hard')

```
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

# 위스콘신 유방암 데이터 세트 생성
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,te
st_size=0.2, random_state=156)

# 로지스틱 회귀와 KNN 객체 생성
lr_clf = LogisticRegression(solver='liblinear')
knn_clf = KNeighborsClassifier(n_neighbors=8)
```

```
# 보팅 분류기 생성
vo_clf = VotingClassifier(estimators=[('LR', lr_clf), ('KNN', knn_clf)], voting
='soft')

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

4.4 랜덤 포레스트

- 여러 개의 결정 트리 분류기가 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 보팅을 통해 예측 결정
- 앙상블 알고리즘 중 비교적 빠른 수행 속도 + 다양한 영역에서 높은 예측 성능
- 부트스트래핑 분할 방식: 서브세트의 데이터 건수는 전체 데이터 선수와 동일하지만 개별 데이터가 중첩되게 만들어짐
- 사이킷런은 RandomForestClassifier 클래스로 랜덤 포레스트 기반 분류를 지원

```
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))
```

하이퍼 파라미터 및 튜닝

- n_estimators: 결정 트리의 개수 지정 (default=10)
- max_features: 결정 트리에 사용되는 파라미터와 똑같이 피처 개수이지만 default= auto = sqrt
- 결정 트리에서 과적합 개선을 위한 파라미터가 똑같이 적용됨

```
# GridSeachCV로 최적 파라미터 추출
params={'max_depth':[8,16,24],'min_samples_leaf':[1,6,12],'min_samples_sp
lit':[2,8,16]}

rt_clf=RandomForestClassifier(n_estimators=100,random_state=0,n_jobs=-
```

```
1)
grid_cv=GridSearchCV(rt_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_))

# 피처 중요도 시각화
ftr_importances_values=rf_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 Top 20')
sns.barplot(x=ftr_top20,y=ftr_top20.index)
plt.show()
```