AI Track

모델 실습

breast_cancer 데이터를 활용한 분류 모델 실습

KNN, Decision Tree

강연자 이름
김중앙

SeCurious

중앙대학교
산업보안학과 교육봉사동아리

23/01/01 Sun

목차

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

위스콘신 유방암 데이터 세트는 종양의 크기, 모양 등의 다양한 속성 값을 기반으로
해당 종양이 악성인지 양성인지 분류한 데이터 세트

01 cancer['data']..shape

행, 열 확인

02 cancer.keys()

데이터의 key 값 출력

03 cancer.feature_names

컬럼 이름 출력

04 cancer.target_names

타켓 이름 출력

데이터셋 분리

breast_cancer

01

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(cancer.data,
cancer.target, stratify=cancer.target, random_state=0)
```

random_state

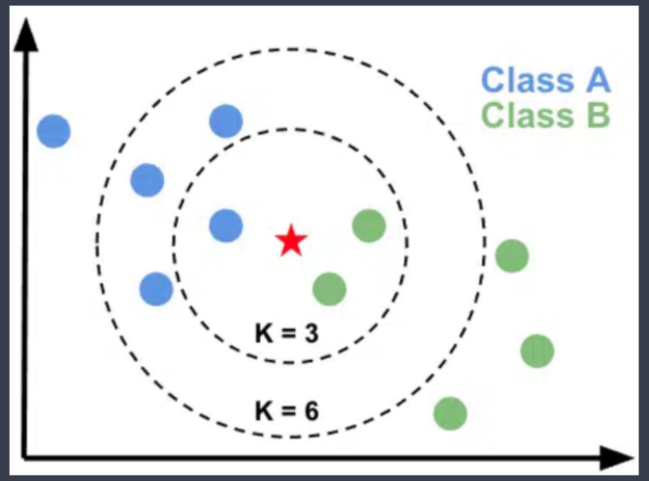
▶ 코드 실행 결과가 일정하도록 random 함수의 seed 값 지정

stratify

▶ train,test 데이터 중 한 데이터에 양성 데이터가 쏠려서 분배되는 것을 방지, 원본 비율 유지 가능

KNeighborsClassifier

어떤 데이터가 주어지면 그 주변의 데이터 k개를 살펴본 뒤 더 많은 데이터가 포함되어있는 범주로 분류하는 방식



k=3 인 경우

class B

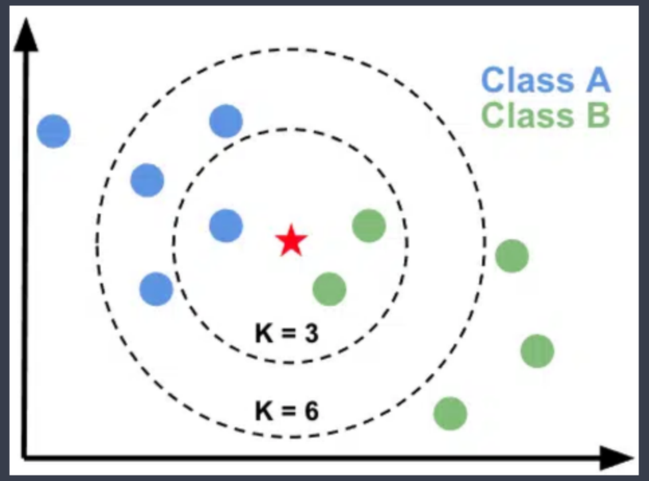
k=6 인 경우

class A

일반적으로 홀수로 지정

KNeighborsClassifier

사용 parameter: n_neighbors(k라고도 함)



k가 너무 크면 underfitting
신중한 모델 X, 예측 결과가 모두 비슷해진다

k가 너무 작으면 overfitting
이웃에 너무 민감해진다

KNeighborsClassifier

사용 parameter: n_neighbors(k라고도 함, 기본값=5)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
print("Accuracy on training set: {:.3f}".format(knn.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(knn.score(X_test, y_test)))
```

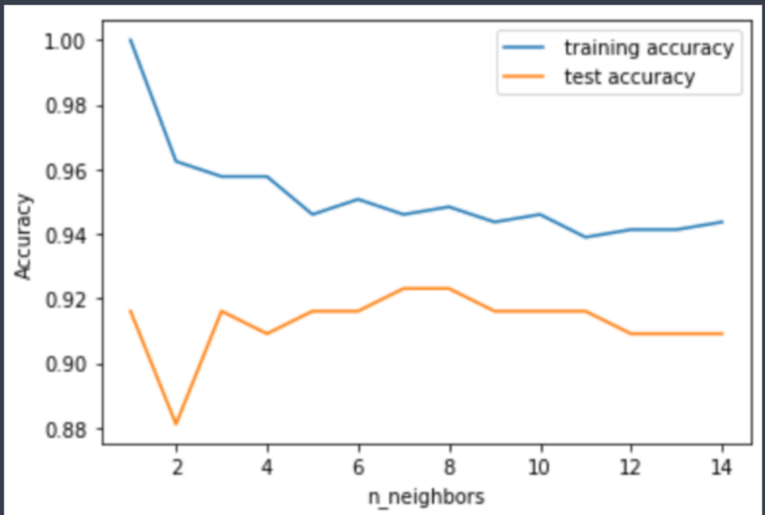
KNeighborsClassifier

parameter 사용
knn = KNeighborsClassifier(n_neighbors=사용자 지정)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
print("Accuracy on training set: {:.3f}".format(knn.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(knn.score(X_test, y_test)))
```

KNeighborsClassifier

적합한 parameter 값을 찾는 방법
for 문을 사용한 시각화



parameter 조정 결과를 한번에 시각화
train 데이터와 test 데이터에 대한 스코어를 변하는 parameter 값에 따라 표시

사용자가 원하는 범위와 값의 parameter 설정 가능
테스트 정확도가 가장 높은 parameter를 쉽게 확인 가능

```
training_accuracy = [] #리스트 생성: 스코어 저장 용도
test_accuracy = []  #이 코드를 사용할 때 항상 리스트를 초기화하는 것 명심!!

neighbors_settings = range(1, 15) #원하는 범위 지정
for n_neighbors in neighbors_settings: #지정한 범위 안에서 아래 코드 반복
    knn = KNeighborsClassifier(n_neighbors = n_neighbors) #모델 선언
    knn.fit( X_train, y_train)  #train 데이터 적용
    training_accuracy.append(knn.score( X_train, y_train)) #앞서 만든 리스트에 스코어를 기록, 추가
    test_accuracy.append(knn.score( X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label =" training accuracy")
plt.plot(neighbors_settings, test_accuracy, label =" test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

KNeighborsClassifier

장점

이해하기 쉽다

많은 조정이 필요하지 않다

빠르다

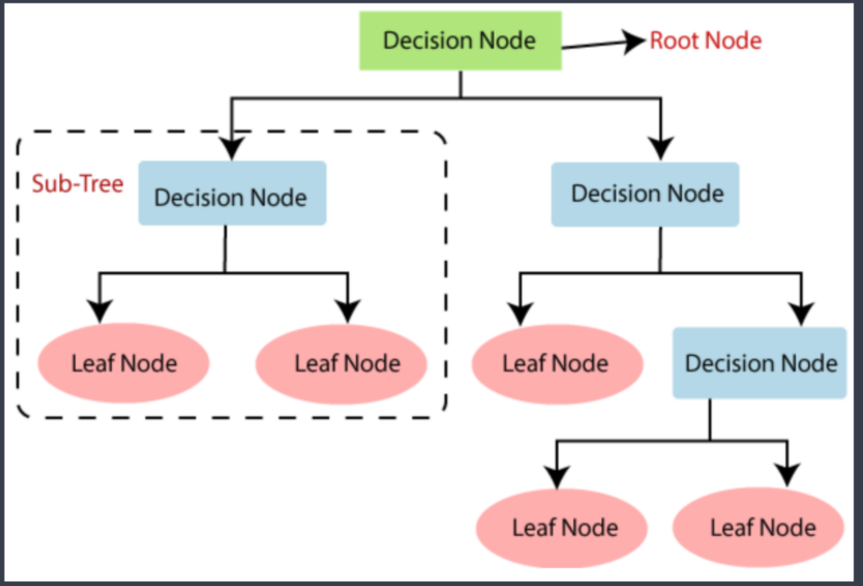
단점

데이터 세트의 크기가 크면 느리고 성능이 좋지 않다

따라서 실제로는 자주 사용하지 않는다

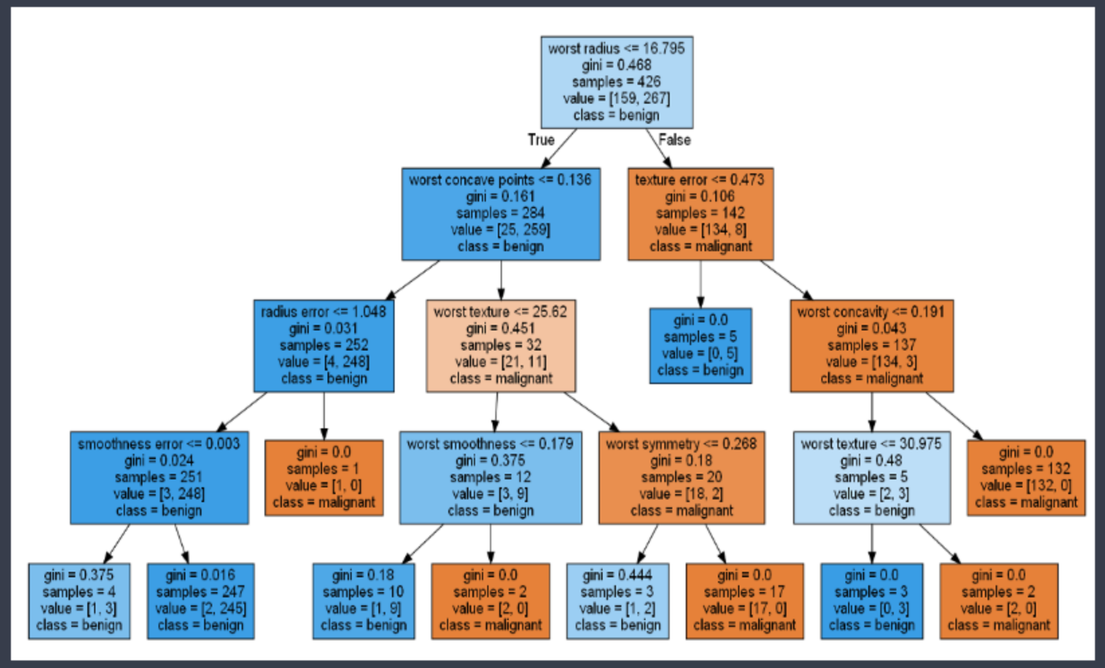
DecisionTreeClassifier

나무가 뒤집어있는 모양의 모델
특정 기준(질문)에 따라 데이터를 구분



Root (node): 시작 노드->가장 영향력이 큰, 분류를 확실히 해주는 칼럼
Leaf (node): 결정 경로에 의해 target 변수를 예측한 결과가 있는 마지막 노드
– Pure (leaf) node: 모두 동일 target 값을 가진 데이터 포인트를 포함하는 마지막 노드
Branch (or Edge or Line)
Depth: 트리의 깊이

DecisionTreeClassifier



각 노드에서 데이터 셋을 여러 하위 집합으로 나눈다
조건문이 있는 노드에서 데이터 셋을 참과 거짓으로 나누는 원리

DecisionTreeClassifier

실제 데이터셋에서는 leaf 노드를 모두 pure leaf 노드로 만들기 어렵기 때문에
(오래 걸리며, 깊이가 매우 깊어진다, train 데이터에 overfitting 되기 쉽다)
분할을 멈추기 위한 parameter 조정 필수

max_depth: 트리 최대 깊이
min_samples_split(기본값=2): 노드를 분할할 때 필요한 최소 샘플 수
min_samples_leaf(기본값=1): 리프노드에 있어야하는 최소 샘플 수
max_leaf_nodes: 리프노드의 최대 개수

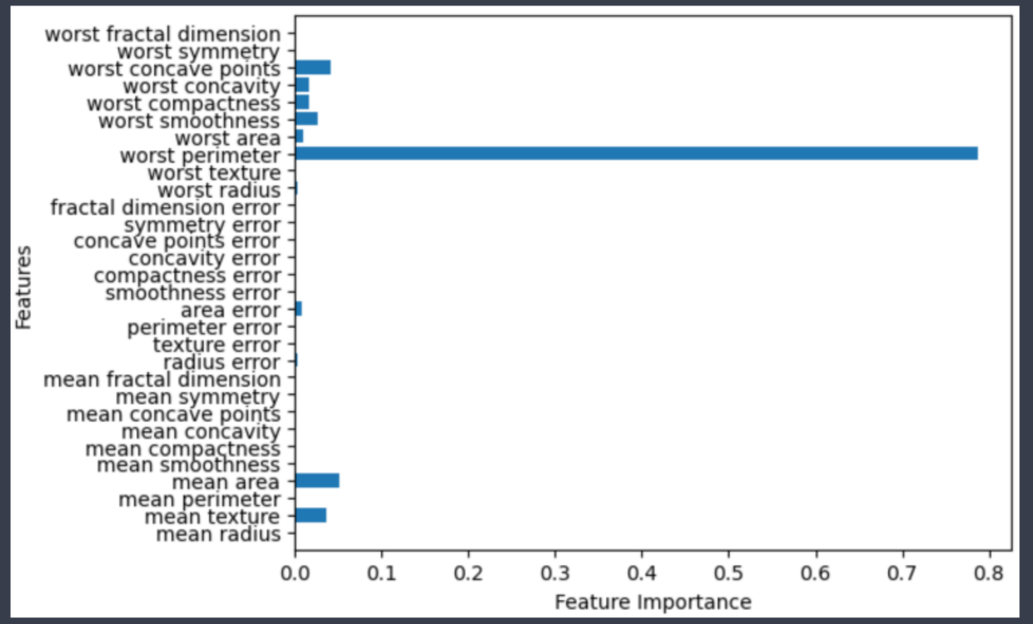
DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

DecisionTreeClassifier

feature_importances_

트리가 내리는 결정에 각 피처가 얼마나 중요한지 평가
0에서 1 사이의 숫자로 0은 전혀 사용되지 않은 피처, 1은 완벽하게 대상을 예측하는 피처를 의미
모든 피처의 중요도 합계는 항상 1
그러나 피처의 중요도가 낮다고 해서 유용하지 않다는 의미는 아님



```
import matplotlib.pyplot as plt
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(np.arange(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Feature Importance")
    plt.ylabel("Features")
    plt.ylim(-1, n_features)

plot_feature_importances_cancer(tree)
```

feature_importances_에는 각 피쳐의 중요도를 담고있음
어떤 피쳐가 타겟을 예측하는 데 많은 영향을 미쳤는지 알 수 있음

DecisionTreeClassifier

장점

쉽게 이해할 수 있다.

parameter 조정을 통해 높은 성능을 보인다.

단점

parameter 조정을 하더라도 overfitting되는 경향이 있고 일반화 성능이 좋지 않을 수 있다.

따라서 일반적으로 ensemble tree 모델을 사용한다.

Thank you

강의 열심히 들어주셔서 감사합니다 :)

강연자 이름
김중앙

SeCurious

중앙대학교
산업보안학과 교육봉사동아리

23/01/01 Sun