노트북: 첫 번째 노트북
만든 날짜: 2021-01-05 오후 6:24
URL: https://continuous-development.tistory.com/173?category=736685

ML,DL

[ML/DL] DecisionTree 구현 및 hyper parameter 설정

2020. 11. 3. 00:25 수정 삭제 공개

의사결정 트리(DecisionTree)

규칙을 학습을 통해 분류 규칙을 만드는 알고리즘 / 데이터의 어떤 기준을 바탕으로 규칙을 만드느냐가 성능을 결정하는 중요한 요소이다.

hyper parameter란?

기계 학습에서 하이퍼 파라미터는 학습 프로세스를 제어하는 데 사용되는 값을 갖는 매개 변수로서 가장 좋은 변수를 뜻한다. 여기서의 가장 좋은 변수는 사용자가 원하는 score 값이 높게 만드는 parameter를 뜻한다.

파라미터에 대한 설명

rootnode- 시작점

leaflood - 더 이상의 규칙을 정할 수 없는 것(결정된 클래스 값)

정보 균일도 측정 방법

정보 이득 방식 - 엔트로피는 데이터의 혼잡도를 의미한다. 엔트로피가 높다는 것은 혼잡도가 높다는 것이다.
지니계수 - 불평등지수 / 이 값이 0 이면 평등하다는 것을(분류가 잘됐다) 뜻한다. 이 값이 리프 노드가 된다.

이 두 가지를 통해 튜닝이 가능하다.

max_nodes

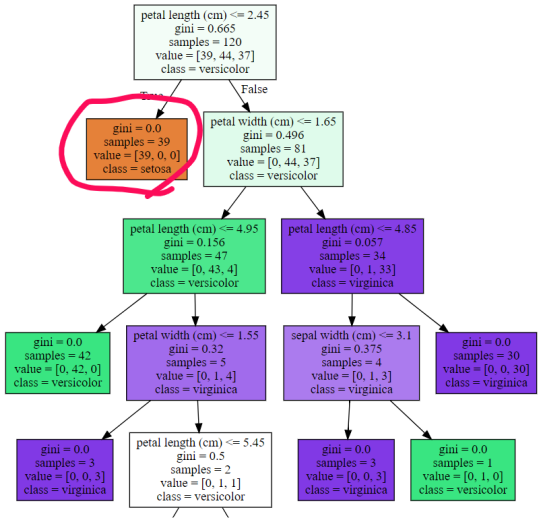
기존에는 노드는 지니계수가 0 이 될 때까지 계층을 만들어가지만 max_노드를 지정해 놓으면 지정해 놓은 층 이상으로 내려가지 않는다.

min_samples_split

해당 노드가 가지고 있는 최소한의 샘플의 개수를 나타낸다.

min sample_leaf

또 다른 노드를 만들 수 있는 최소한의 샘플 수 조건을 뜻한다.



해당 노드의 경우 gini0.0 이 되었다. 이건 평등하다는 뜻으로 해당 노드는 확정이 됐다 오른쪽 노드 같은 경우에는 아직 gini계수가 0이 안됐으므로 계속 나눈다.

또한 value를 봤을 때 최소 샘플 개수까지 내려가지 않았기 때문에 계속 트리를 만들게 된다.

구현

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder , OneHotEncoder , StandardScaler , MinMaxScaler , Binarizer
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score , roc_auc_score
from sklearn.metrics import confusion_matrix, precision_recall_curve , roc_curve

import pandas as pd
import numpy  as np
import matplotlib.pyplot as plt
import seaborn as sns

# import missingno as ms
%matplotlib inline
import warnings


warnings.filterwarnings('ignore')

from sklearn.datasets import load_iris
```

기본적인 함수들을 import 해주고 iris 데이터를 사용한다.

```
# 분류기 생성
dtc_iris = DecisionTreeClassifier(random_state=100)


# 데이터 로드 및 전처리
# 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=.2,random_state=100)


# 학습
dtc_iris.fit(X_train,y_train)
```

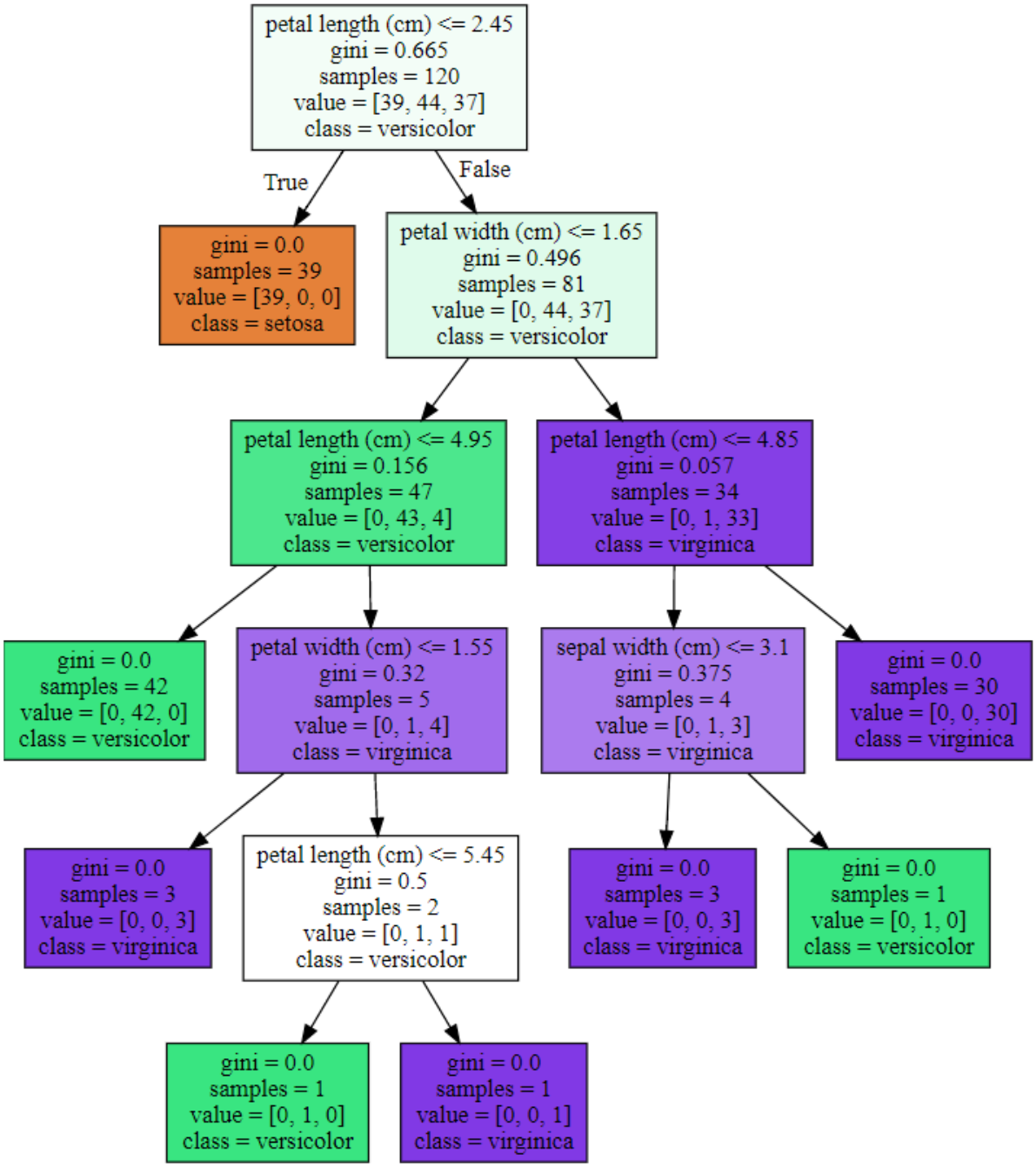
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=100,
            splitter='best')
```

```
from sklearn.tree import export_graphviz

# export_graphviz( )의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함
export_graphviz(dtc_iris, out_file="tree.dot", class_names = iris_data.target_names,
                feature_names = iris_data.feature_names, impurity=True, filled=True)


print('================max_depth의 제약이 없는 경우의 Decision Tree 시각화==================')
import graphviz

# 위에서 생성된 tree.dot 파일을 Graphiviz 가 읽어서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



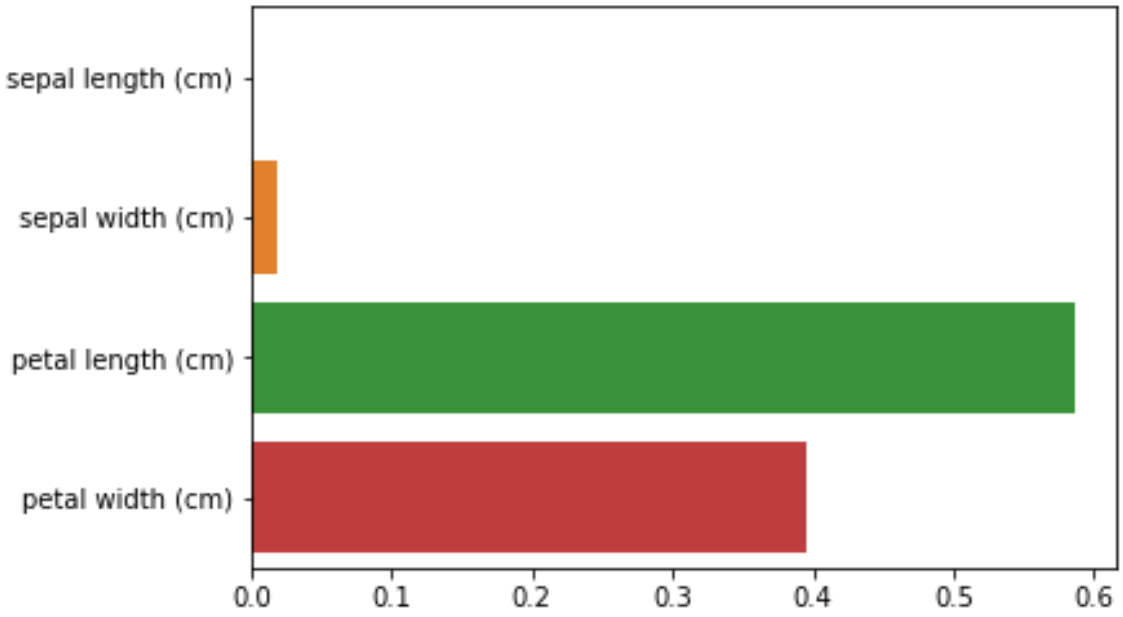

```
# 변수별 중요도
print('feature importance:',dtc_iris.feature_importances_)
```

```
feature importance: [0.         0.01880092 0.58591345 0.39528563]
```

feature_importances_ 명령어를 통해 변수별 중요도를 볼 수 있다.

```
sns.barplot(x=dtc_iris.feature_importances_,y=iris_data.feature_names)
```

```
sepal length (cm) 0.0
sepal width (cm) 0.018800919156047 63
petal length (cm) 0.5859134473686348
petal width (cm) 0.3952856334753 1753
```

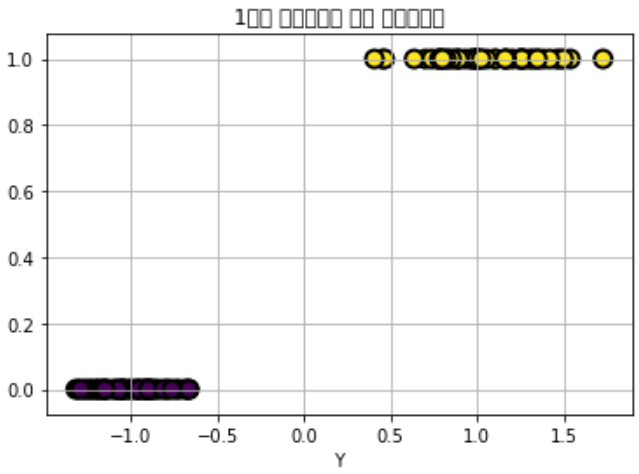


- 의사결정 트리의 단점

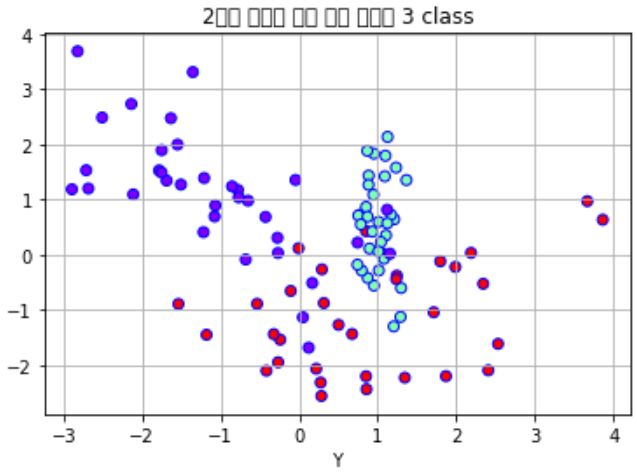
- 과적합(학습 데이터에서 완벽한 성능을 보이지만 테스트에서 성능이 좋지 않은 경우)
- 과적합을 위해서 분류용 가상의 데이터를 생성 make_classification()
가상용 데이터를 만들고 파라미터 값을 어떻게 바꾸냐에 따라 그래프가 어떻게 변경되는지를 볼 수 있다. 이 방법을 통해 하이퍼 파라미터를 맞춰 줄 수 있다.

```
from sklearn.datasets import make_classification
plt.title('1개의 독립변수를 가진 가상데이터')
X,y = make_classification(n_features=1,
                          n_informative =1,
                          n_redundant=0,
                          n_clusters_per_class=1,
                          random_state=100)
```

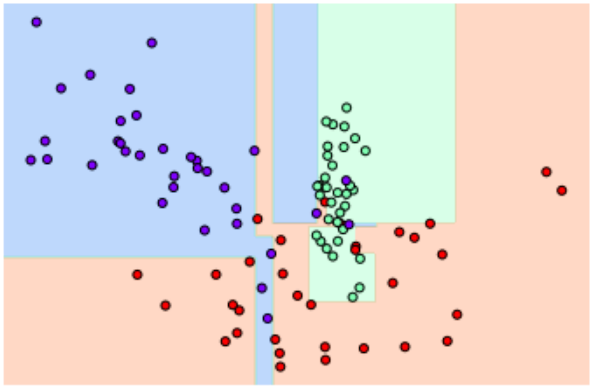
```
plt.scatter(X,y, marker='o', c=y, s=100, edgecolor='k',linewidth=2)
plt.xlabel('X')
plt.xlabel('Y')
plt.grid()
plt.show()
```



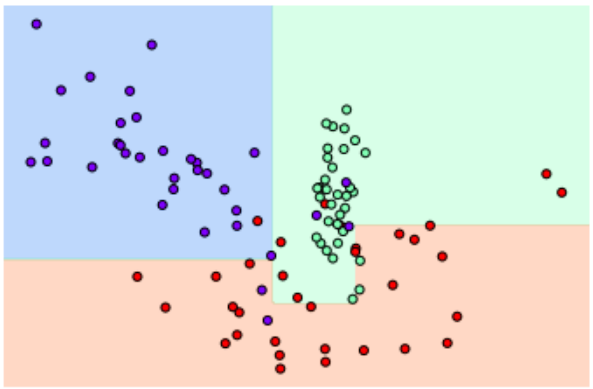

```
from sklearn.datasets import make_classification
plt.title('2개의 피처를 가진 가상 데이터 3 class')
X_features,y_labels = make_classification(n_features=2,
                    n_informative =2,
                    n_redundant=0,
                     n_classes=3,
                    n_clusters_per_class=1,
                    random_state=0)
plt.scatter(X_features[:,0],X_features[:,1], marker='o', c=y_labels, edgecolor='b',cmap='rainbow')
plt.xlabel('X')
plt.xlabel('Y')
plt.grid()
plt.show()
```



```
#튜닝을 통한 과적합 해결
sample_dtc_model = DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(sample_dtc_model,X_features,y_labels)
```



```
# max_depth 줄여서 트리의 깊이를 제한
# min_samples_split 높여서 데이터가 분할하는데 필요한 샘플데이터의 수를 높인다.
# min_samples_leaf 높여서 말단 노드가 되는데 필요한 심플데이터의 수를 높인다.
# max_featrues 높여서 분할을하는데 고려하는 피처의 수 제한
sample_dtc_model = DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels)
visualize_boundary(sample_dtc_model,X_features,y_labels)
```



UCI HAR 결정 트리 실습

```
# 데이터셋을 구성하는 함수 설정

def har_dataset():

    # 각 데이터 파일들은 공백으로 분리되어 있으므로 read_csv에서 공백문자를 sep으로 할당

    feature_name_df = pd.read_csv('../data/features.txt', sep='\s+',
                                  header=None, names=['column_index', 'column_name'])
    # 데이터프레임에 피처명을 컬럼으로 뷰여하기 위해 리스트 객체로 다시 반환

    feature_name = feature_name_df.iloc[:, 1].values.tolist()

    # 학습 피처 데이터세트와 테스트 피처 데이터를 데이터프레임으로 로딩
    # 컬럼명은 feature_name 적용
    X_train = pd.read_csv('../data/train/X_train.txt', sep='\s+', names=feature_name)
    X_test = pd.read_csv('../data/test/X_test.txt', sep='\s+', names=feature_name)


    # 학습 레이블과 테스트 레이블 데이터를 데이터 프레임으로 로딩, 컬럼명은 action으로 부여

    y_train = pd.read_csv('../data/train/y_train.txt', sep='\s+', names=['action'])
    y_test = pd.read_csv('../data/test/y_test.txt', sep='\s+', names=['action'])

    # 로드된 학습/테스트용 데이터프레임을 모두 반환
    return X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = har_dataset()
```

- 튜닝없이 예측 즉, 디폴트로 예측한 결과

```
har_dtc = DecisionTreeClassifier(random_state=120)
print('parameter: ',har_dtc.get_params()) # 내가 튜닝할 수 있는 하이퍼 파라미터가 나온다.
```

```
parameter:  {'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min
_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_lea
f': 0.0, 'presort': False, 'random_state': 120, 'splitter': 'best'}
```

```
har_dtc.fit(X_train,y_train)
y_pred = har_dtc.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print('Decision Tree 예측 값:',accuracy)
```

Decision Tree 예측 값: 0.8598574821852731

```
# max_depth가 정확도에 주는 영향
params = {
    'max_depth' : [6,8,10,12,16,20,24]
```

```
}

grid_cv = GridSearchCV(har_dtc, param_grid = params, scoring='accuracy', cv=5)
grid_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=120,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': [6, 8, 10, 12, 16, 20, 24]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

print('최고 평균 정확도 수치: ', grid_cv.best_score_)

print('최고 하이퍼 수치: ', grid_cv.best_params_)

```
# max_depth가 크면 과적합으로 성능이 하락하는지 살펴보자
max_depths = [6,10,12,16,20,24]

for depth in max_depths:
    har_dtc = DecisionTreeClassifier(max_depth=depth,random_state=100)
    har_dtc.fit(X_train,y_train)
    y_pred = har_dtc.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print('max_dept:{}, Decisionm Treee 예 값{}:'.format(depth,accuracy))
```

```
max_dept:6, Decisionm Treee 예 값0.8537495758398371:
max_dept:10, Decisionm Treee 예 값0.8673227010519172:
max_dept:12, Decisionm Treee 예 값0.8561248727519511:
max_dept:16, Decisionm Treee 예 값0.8544282321004412:
max_dept:20, Decisionm Treee 예 값0.8506956226671191:
max_dept:24, Decisionm Treee 예 값0.8506956226671191:
```

```
# 해당 파라미터를 적용해서 예측 수행
best_params_dtc = grid_cv.best_estimator_
print(best_params)
best_pred = best_params_dtc.predict(X_test)
accuracy = accuracy_score(y_test,best_pred)
print('예측 정확도 : ', accuracy)
```

```
sionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
        max_features=None, max_leaf_nodes=None,
        min_impurity_decrease=0.0, min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, presort=False, random_state=120,
        splitter='best')
 정확도 :  0.8557855446216491
```

```
# best_params_dtc.feature_importances_
# 피처 중요도를 시각화 top 20
feature_importance = pd.Series(best_params_dtc.feature_importances_, index=X_train.columns)

# featuer_importance
feature_top20 = feature_importance.sort_values(ascending=False)[:20]

# featuer_top20
plt.figure(figsize=(15,5))
plt.title('feture importance')
sns.barplot(x=feature_top20, y=feature_top20.index)
plt.show()
```


'ML,DL' 카테고리의 다른 글□

[ML/DL] 앙상블 학습 (Ensemble Learning): 1. bagging(배깅)이란?□

[ML/DL] 앙상블 학습 (Ensemble Learning): bagging,voting,boosting□

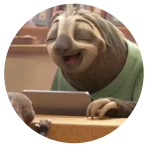
[ML/DL] DecisionTree 구현 및 hyper parameter 설정□

[ML/DL] python 으로 구현하는 ROC곡선과 AUC□

[ML/DL] 정밀도와 재현율의 트레이드 오프 정의와 구현□

[ML/DL] python 을 통한 분류(classification) 성능평가지표 사용법(Accuracy,Precision,Recall,F1 Sc...

DecisionTree hyper parameter　python DecisionTree　python make_classification　파이썬 DecisionTree



혼자 끄적끄적하는 블로그 입니다.