인공지능

HW1

2017803094 최지훈
소프트웨어학부
인공지능
박병준 교수님
월6, 수5 강의

첫 번째 dataSet

●작성한 소스코드의 구동 환경

아나콘다의 버전은 4.7.12고의 Python 3.7.4에서 작성하고 사용 IDE는 jupyter Notebook입니다.

●작성한 코드에 대한 설명

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

우선 필요한 데이터세트에 대한 load, 그리고 필요한 라이브러리들을 불러오고 있습니다.

```
#Dot to png
import pydot
```

dot파일에서 png파일로 바꾸기 위해 필요한 것을 불러오고 있습니다.

```
#샘플 데이터 로드(유방암 데이터 세트)
cancer = load_breast_cancer()
```

유방암 데이터를 읽어들입니다.

```
#훈련, 테스트 데이터 셔플
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

데이터를 읽은 후 훈련세트와 테스트 세트로 나누고 있습니다.

```
#의사결정 트리 선언
dTreeAll = DecisionTreeClassifier(random_state=0)

#훈련 (모든 리프 노드 사용)
dTreeAll.fit(X_train, y_train)

#점수 출력
print("Train Set Score1 : {:.3f}".format(dTreeAll.score(X_train, y_train)))
print("Test  Set Score1 : {:.3f}".format(dTreeAll.score(X_test, y_test)))
```

모든 리프노드가 순수노드라서 훈련 정확도가 처음에 100프로가 나오고 테스트 정확도는 93.7% 정도가 나옵니다.

```
#의사결정 트리 선언(트리 깊이 제한)
dTreeLimit = DecisionTreeClassifier(max_depth=4, random_state=0)
```

결정 트리의 깊이를 제한하지 않으면 트리가 무한정 깊어질 수도 있어서 위험하기에

max_depth로 제한하여 트리의 성장을 막으려고 한 것입니다.

```
#훈련 (가지치기 : 리프노드 깊이 제한)
dTreeLimit.fit(X_train, y_train)

#점수 출력
print("Train Set Score2 : {:.3f}".format(dTreeLimit.score(X_train, y_train)))
print("Test  Set Score2 : {:.3f}".format(dTreeLimit.score(X_test, y_test)))
```

가지치기를 한 뒤 훈련 세트의 정확도는 98.8%, 테스트 세트 정확도는 95.1%로 나왔습니다.

```
export_graphviz(dTreeLimit, out_file="decisionTree1.dot", class_names =[
"malignant","benign"], feature_names = cancer.feature_names, impurity=False, filled=True)
```

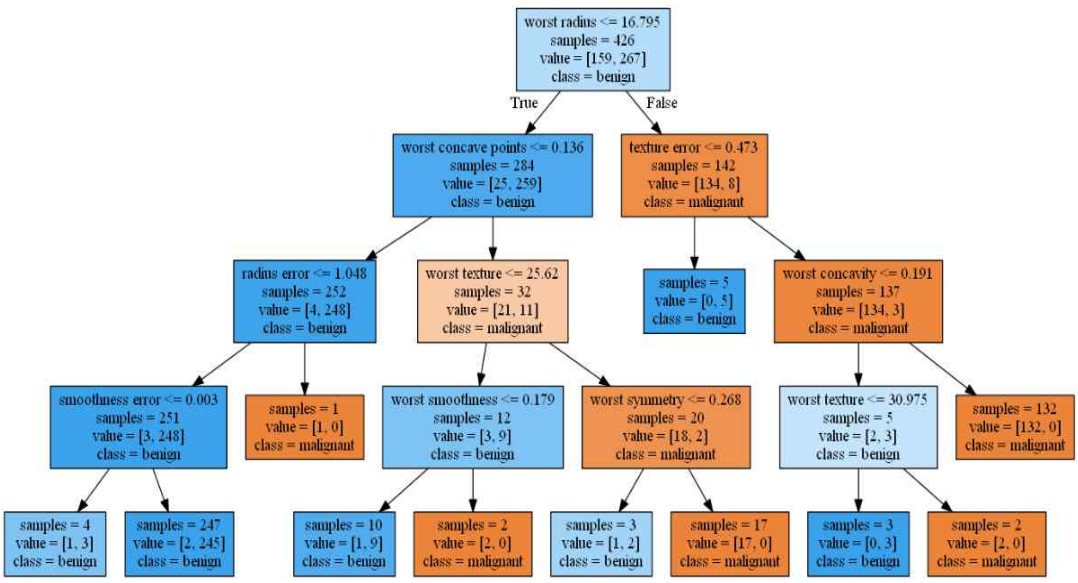
결정 트리를 시각화하기 위해서 이렇게 사용하고 있습니다. export_graphviz()함수를 사용해서 그래프 정의를 decisionTree1.dot로 출력하고 있습니다.
filled=True로 각 노드에서 다수의 클래스를 색으로 표현하고, class_names는 각 클래스를 위한 레이블이고 feature_names는 특성이름으로 분기 조건을 표시한 것입니다. 그리고 Impurity는 각 노드의 impurity 값을 보여주는 것입니다.

```
#Encoding 중요
(graph,) = pydot.graph_from_dot_file('decisionTree1.dot', encoding='utf8')
```

실험 결과 로그를 보여주기 위해서 dot 파일을 만들고 있습니다.

```
#Dot 파일을 Png 이미지로 저장
graph.write_png('decisionTree1.png')
```

●트리 시각화 출력물



●각 실험 결과 로그 캡쳐

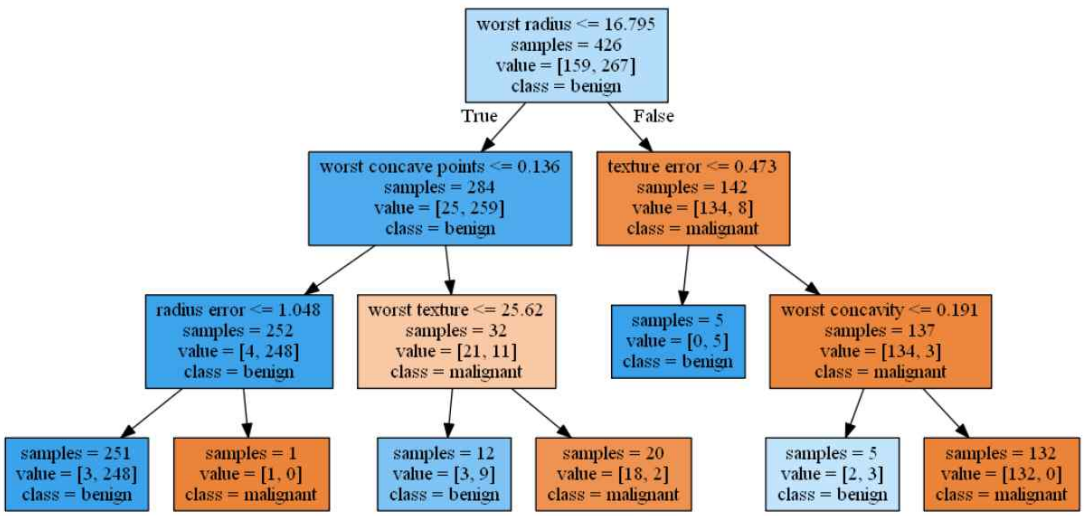
```
digraph Tree {

node [shape=box, style="filled", color="black"] ;

0 [label="worst radius <= 16.795₩nsamples = 426₩nvalue = [159, 267]₩nclass = benign", fillcolor="#afd7f4"] ;

1 [label="worst concave points <= 0.136₩nsamples = 284₩nvalue = [25, 259]₩nclass = benign", fillcolor="#4ca6e8"] ;

0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;

2 [label="radius error <= 1.048₩nsamples = 252₩nvalue = [4, 248]₩nclass = benign", fillcolor="#3c9fe5"] ;

1 -> 2 ;

3 [label="smoothness error <= 0.003₩nsamples = 251₩nvalue = [3, 248]₩nclass = benign", fillcolor="#3b9ee5"] ;

2 -> 3 ;

4 [label="samples = 4₩nvalue = [1, 3]₩nclass = benign", fillcolor="#7bbeee"] ;

3 -> 4 ;

5 [label="samples = 247₩nvalue = [2, 245]₩nclass = benign", fillcolor="#3b9ee5"] ;

3 -> 5 ;

6 [label="samples = 1₩nvalue = [1, 0]₩nclass = malignant", fillcolor="#e58139"] ;

2 -> 6 ;

7 [label="worst texture <= 25.62₩nsamples = 32₩nvalue = [21, 11]₩nclass = malignant", fillcolor="#f3c3a1"] ;

1 -> 7 ;

8 [label="worst smoothness <= 0.179₩nsamples = 12₩nvalue = [3, 9]₩nclass = benign", fillcolor="#7bbeee"] ;

7 -> 8 ;

9 [label="samples = 10₩nvalue = [1, 9]₩nclass = benign", fillcolor="#4fa8e8"] ;

8 -> 9 ;

10 [label="samples = 2₩nvalue = [2, 0]₩nclass = malignant", fillcolor="#e58139"] ;

8 -> 10 ;

11 [label="worst symmetry <= 0.268₩nsamples = 20₩nvalue = [18, 2]₩nclass = malignant", fillcolor="#e88f4f"] ;

7 -> 11 ;

12 [label="samples = 3₩nvalue = [1, 2]₩nclass = benign", fillcolor="#9ccef2"] ;

11 -> 12 ;

13 [label="samples = 17₩nvalue = [17, 0]₩nclass = malignant", fillcolor="#e58139"] ;

11 -> 13 ;

14 [label="texture error <= 0.473₩nsamples = 142₩nvalue = [134, 8]₩nclass =
```

```
malignant", fillcolor="#e78945"] ;

0 -> 14 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;

15 [label="samples = 5Wnvalue = [0, 5]Wnclass = benign", fillcolor="#399de5"] ;

14 -> 15 ;

16 [label="worst concavity <= 0.191Wnsamples = 137Wnvalue = [134, 3]Wnclass =
malignant", fillcolor="#e6843d"] ;

14 -> 16 ;

17 [label="worst texture <= 30.975Wnsamples = 5Wnvalue = [2, 3]Wnclass = benign",
fillcolor="#bddef6"] ;

16 -> 17 ;

18 [label="samples = 3Wnvalue = [0, 3]Wnclass = benign", fillcolor="#399de5"] ;

17 -> 18 ;

19 [label="samples = 2Wnvalue = [2, 0]Wnclass = malignant", fillcolor="#e58139"] ;

17 -> 19 ;

20 [label="samples = 132Wnvalue = [132, 0]Wnclass = malignant", fillcolor="#e58139"]
;

16 -> 20 ;

}
```

각 노드가 어떠한 성질을 가지고 있고, 분기에 따라서 노드가 이동함을 보여주고 있습니다. 트리 시각화 결과물을 그냥 텍스트화한 것과 같습니다.

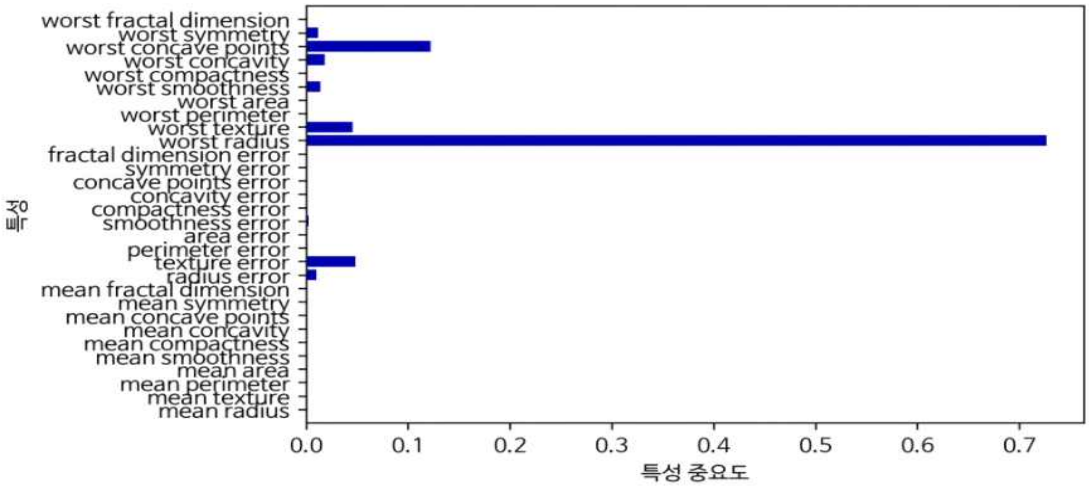
●실험 결과 분석

시각화로 인해 알고리즘의 예측이 어떻게 이루어지는지 볼 수 있습니다.



이는 트리의 깊이 제한을 3으로 했을 경우의 모습입니다. 4로 했을 때보다 훨씬 간결해 보이는 모습입니다. 깊이가 4만 되어도 트리가 매우 장황해져서 더 깊어지면 한눈에 보기가 어려울 것 같습니다. 그럼에도 불구하고 많은 수의 데이터가 흐르는 경로 조사에 유용한 편

입니다.
benign은 양성, malignant는 악성을 판단해 줍니다.



첫 번째 노드에서 사용한 특성이 가장 중요한 특성으로 나타납니다. 어떤 특성의 feature_importance 값이 낮다고 해서 유용하지 않다는 뜻이 아니고, 단지 이 트리가 그 특성을 선택하지 않았을 뿐입니다. 오히려 다른 특성이 동일한 정보를 지니고 있어서일 수 있습니다. 특성의 중요도가 어떤 단일 클래스를 지지하는 것은 아님에 유의해야할 것 같습니다.

두 번째 dataSet

●작성한 코드에 대한 설명

```
from sklearn import tree
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

붓꽃 데이터에 대하여 DecisionTreeClassifier을 훈련시키고 있습니다.

```
clf.predict([[2., 2.]])
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
```

```
import pydot
```

위 코드와 마찬가지로 각종 필요한 라이브러리들을 import하고 있습니다.

```
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
```

dataSet을 가져오고 target을 정했습니다. 타겟 데이터는 setosa, versicolor, virginica의 세 가지 붓꽃 종입니다.

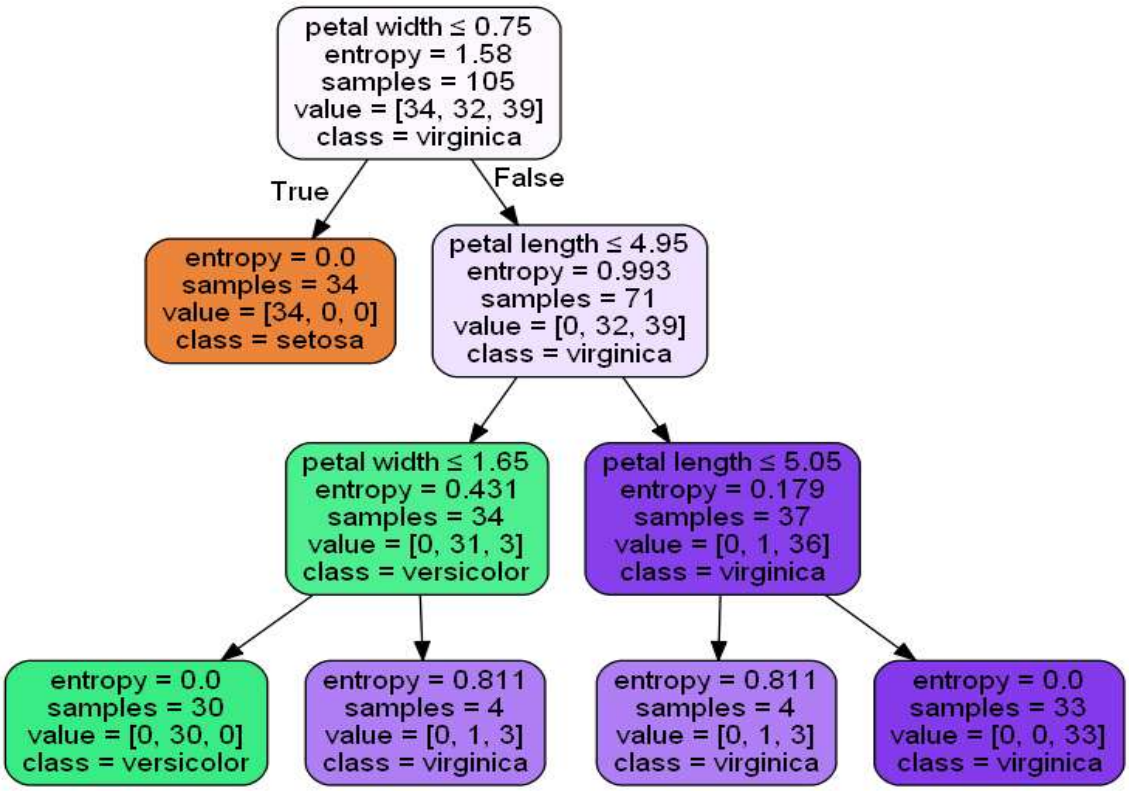
```
# 자동으로 데이터셋을 분리해주는 함수
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# 데이터 표준화 작업
sc = StandardScaler()
sc.fit(X_train)

# 표준화된 데이터셋
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

iris_tree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
iris_tree.fit(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score

y_pred_tr = iris_tree.predict(X_test)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_tr))

from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

dot_data = export_graphviz(iris_tree, out_file='iris_tree1.dot', feature_names=['petal length', 'petal width'],
                           class_names=iris.target_names, filled=True, rounded=True, special_characters=True)
```

dot 파일 추출 작업입니다. 이는 결정트리를 시각화 하는 것입니다.

```
#Encoding 중요
(graph,) = pydot.graph_from_dot_file('iris_tree1.dot', encoding='utf8')

#Dot 파일을 Png 이미지로 저장
graph.write_png('iris_tree1.png')
```

●트리 시각화 출력물



●각 실험 결과 로그 캡쳐

```
digraph Tree {

node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;

edge [fontname=helvetica] ;

0 [label=<petal width &le; 0.75<br/>entropy = 1.58<br/>samples = 105<br/>value = [34, 32, 39]<br/>class = virginica>, fillcolor="#f6f1fd"] ;

1 [label=<entropy = 0.0<br/>samples = 34<br/>value = [34, 0, 0]<br/>class = setosa>, fillcolor="#e58139"] ;

0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;

2 [label=<petal length &le; 4.95<br/>entropy = 0.993<br/>samples = 71<br/>value = [0, 32, 39]<br/>class = virginica>, fillcolor="#e8dbfa"] ;

0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;

3 [label=<petal width &le; 1.65<br/>entropy = 0.431<br/>samples = 34<br/>value = [0, 31, 3]<br/>class = versicolor>, fillcolor="#4ce88d"] ;

2 -> 3 ;

4 [label=<entropy = 0.0<br/>samples = 30<br/>value = [0, 30, 0]<br/>class = versicolor>, fillcolor="#39e581"] ;

3 -> 4 ;
```

```
5 [label=<entropy = 0.811<br/>samples = 4<br/>value = [0, 1, 3]<br/>class = virginica>, fillcolor="#ab7bee"] ;

3 -> 5 ;

6 [label=<petal length &le; 5.05<br/>entropy = 0.179<br/>samples = 37<br/>value = [0, 1, 36]<br/>class = virginica>, fillcolor="#843ee6"] ;

2 -> 6 ;

7 [label=<entropy = 0.811<br/>samples = 4<br/>value = [0, 1, 3]<br/>class = virginica>, fillcolor="#ab7bee"] ;

6 -> 7 ;

8 [label=<entropy = 0.0<br/>samples = 33<br/>value = [0, 0, 33]<br/>class = virginica>, fillcolor="#8139e5"] ;

6 -> 8 ;

}
```

●실험 결과 분석

루트 노드는 깊이가 0인 맨 꼭대기 노드이고, 왼쪽의 자식 노드는 깊이가 1이고 더 이상 깊어지지 않기에 이 경우 리프노드라고 합니다. 자식을 가지지 않는 노드입니다. entropy는 불순도이며 그 노드에 해당하는 샘플 수를 samples에 넣었습니다. 그리고 value=[a,b,c]는 해당 노드의 클래스별 샘플 수입니다. 그러므로 a,b,c를 더해서 sample의 수가 나오는 것입니다.

이를 분류할 때 CART 훈련 알고리즘을 사용했는데 CART란 Classification and Regression Tree입니다. 훈련세트를 하나의 특성 k를 임계 값 tk를 사용해서 subset으로 나눈 것입니다. 예를 들어서 가장 위에 있는 노드인 petal width <= 0.75와 같은 것입니다.

k와 tk를 고르는 방법은 가장 순수한 subset으로 나눌 수 있는 k,tk의 짝을 찾아서 그걸로 나눕니다.

이러한 방식으로 subset을 나누고 max_depth까지 이를 반복합니다.

최적의 트리를 찾는 문제는 O(exp(m))입니다. 여기서 m은 샘플 수입니다.