통계학과머신러닝PBL

유방암 진단 데이터 분석 리포트

(Breast Cancer Wisconsin Diagnostic Dataset)

목 차

Ⅰ. 문제정의

유방암 진단 사진으로부터 측정한 종양의 특정 값들을 사용하여 종양이 양성(benign)인지 악성(malignant)인지를 판별하고자 한다. 분석 데이터로는 위스콘신 대학교에서 제공한 유방암 진단 데이터를 사용한다.

Ⅱ. 데이터 설명

1. 분석 데이터

scikit-learn 유방암 진단 데이터

(**Breast cancer wisconsin diagonostic dataset**
https://scikit-learn.org/stable/datasets/toy_dataset.html#breast-cancer-wisconsin-diagnostic-dataset)

2. 데이터 속성

· Number of Instances : 569
· Number of Attributes : 30 numeric

3. 변수 설명

· 특성변수

radius (mean of distances from center to points on the perimeter)	반지름 (중심점에서 외벽까지 거리들의 평균)
texture (standard deviation of gray-scale values)	질감 (Gray-Sclae 값들의 표준편차)
perimeter	둘레
area	면적
smoothness (local variation in radius lengths)	매끄러움 (반지름 길이의 국소적 변화)
compactness (perimeter^2 / area - 1.0)	조그만 정도 (둘러^2 / 면적-1.0)
concavity (severity of concave portions of the contour)	오목함 (윤곽의 오목한 부분의 정도)

concave points (number of concave portions of the contour)	오목한 점 (윤곽의 오목한 부분 수)
symmetry	대칭
fractal dimension	프랙탈 차원

특정 변수들은 mean(평균), standard error(표준오차), worst(가장 큰값 3개의 평균)로 계산되어 각각 총 30개가 생성됨.

· 목적변수

target	목적 변수 (악성(malignant) : 0, 양성(benign) : 1)

Ⅲ. 데이터 탐색

1. 데이터 불러오기

```
import pandas as pd
from sklearn import datasets
load_df = datasets.load_breast_cancer()
```

· 데이터프레임을 다루기 위한 pandas와 데이터를 불러오기 위한 sklearn 라이브러리를 작성하고, load_df로 유방암 진단 데이터를 불러온다.

```
# array형식의 load_df를 DataFrame형태로 변경
data = pd.DataFrame(load_df.data)

# DataFrame 형태로 바뀐 data의 column을 지정
feature = pd.DataFrame(load_df.feature_names)
data.columns = feature[0]

# target 변수의 column 또한 지정
target = pd.DataFrame(load_df.target)
target.columns = ['target']

# data (columns 포함), target 컬럼으로 데이터 합침

# datad와 target을 열 방향으로 합친다.
# 최종 df 생성
df = pd.concat([data, target], axis=1)
df
```

· 최종 df를 만들기 위한 작업을 진행한다.

2. 데이터 탐색 과정

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	1

```
print(load_df.DESCR)
```

```
Output exceeds the size limit. Open the full output data in a text editor
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
...
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

```
print(df.shape)
df.head()
```

shape() 함수를 이용하여 행과 열의 개수를 알 수 있으며, head() 함수로는 상위 5행을 출력하여 데이터의 값을 부분적으로 확인할 수 있다.

```
(569, 31)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	...

5 rows × 31 columns

```
df.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

결측값을 확인해본 결과 없는 것으로 나타났다.

Ⅳ. 모델

· 분류 기법 : CART (Classification And Regression Tree) Algorithm

1. X, y 데이터 생성

```
X = df.drop('target', axis=1)
y = df['target']
```

모델링을 진행하기 앞서 목적변수 'target'을 제외한 나머지 열을 input X로 정의, 'target' 열을 label y로 정의

2. Train data, Test data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

전체 data를 training set과 test set으로 구분하는 과정으로써, train 데이터는 70%, test 데이터는 30%로 data의 비율을 유지하며 데이터를 만든다.

3. Classification Tree

3.1 Gini Index

3.1.0. Impurity: Gini Index & Max depth: 2

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

분류 나무 분석을 진행하기 위해 DecisionTreeClassifier를 호출하여 실행한다.
불순도 측도 기준은 지니 지수이며, 나무의 깊이는 2, random_state 값은 1로 지정하여 X_train과 y_train을 적합시킨다.
X_train을 통해 예측한 값을 y_train_pred, X_test를 통해 예측한 값을 y_test_pred로 할당하여 저장한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9623115577889447
0.9473684210526315
```

train data에 대한 정확도와 test data에 대한 정확도를 구한 결과 train set은 약 96.2%, test set은 약 94.7%의 정확도를 보여주고 있다. 이는 test data가 약 94.7%를 분류하였다는 것을 의미한다.

```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

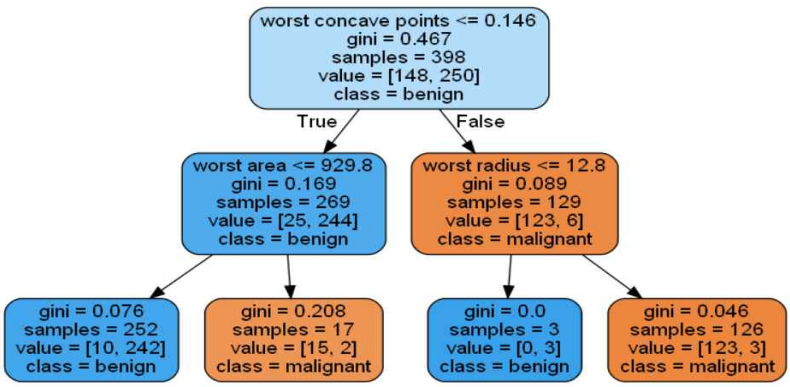
```
[[ 60   4]
 [  5 102]]
```

현재 악성(malignant)인 0은 64개, 양성(benign)인 1은 107개이다. 오차 행렬(confusion matrix)을 통해 분류 결과를 확인해보자면 악성은 4개, 양성은 5개의 값이 오분류 되었음을 알 수 있다.

다음으로는 분류 나무그림을 통해 확인한다. 악성과 양성 여부를 판별하기 위해 노드가 결정된 차례와 변수 선택, 그리고 노드값을 제공한다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree1.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 worst concave points를 기준으로 분류되었다. 종양의 오목한 점이 0.146보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 1 분류되었다. 다음 중간마디(internal node)는 worst area, worst radius로 나뉘었으며 각각 지니계수는 0.169, 0.089이다. 끝노드(terminal node)의 값 중 반지름 값(worst)이 12.8보다 작거나 같은 경우로 분류된 노드는 모두 양성(benign)으로 분류되었음을 확인할 수 있다. 추가적으로, 노드에 색이 모두 표시된 것을 보아 순도가 높게 나타났음을 알 수 있다.

3.1.1. Impurity: Gini Index & Max depth: 3

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

분류 나무 분석을 진행하기 위해 DecisionTreeClassifier를 호출하여 실행한다.
불순도 측도 기준은 지니 지수이며, 나무의 깊이는 3, random_state 값은 1로 지정하여 X_train과 y_train을 적합시킨다.
X_train을 통해 예측한 값을 y_train_pred, X_test를 통해 예측한 값을 y_test_pred로 할당하여 저장한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9698492462311558
0.9532163742690059
```

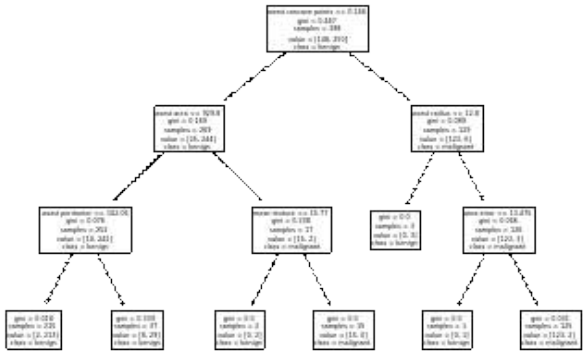
train data에 대한 정확도와 test data에 대한 정확도를 구한 결과 train set은 약 97%, test set은 약 95.3%의 정확도를 보여주고 있다. 이는 test data가 약 95.3%를 분류하였다는 것을 의미한다.

```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 60   4]
 [  4 103]]
```

현재 악성(malignant)인 0은 64개, 양성(benign)인 1은 107개이다. 오차 행렬(confusion matrix)을 통해 분류 결과를 확인해보자면 0과 1 모두 4개의 값이 오분류 되었음을 알 수 있다.

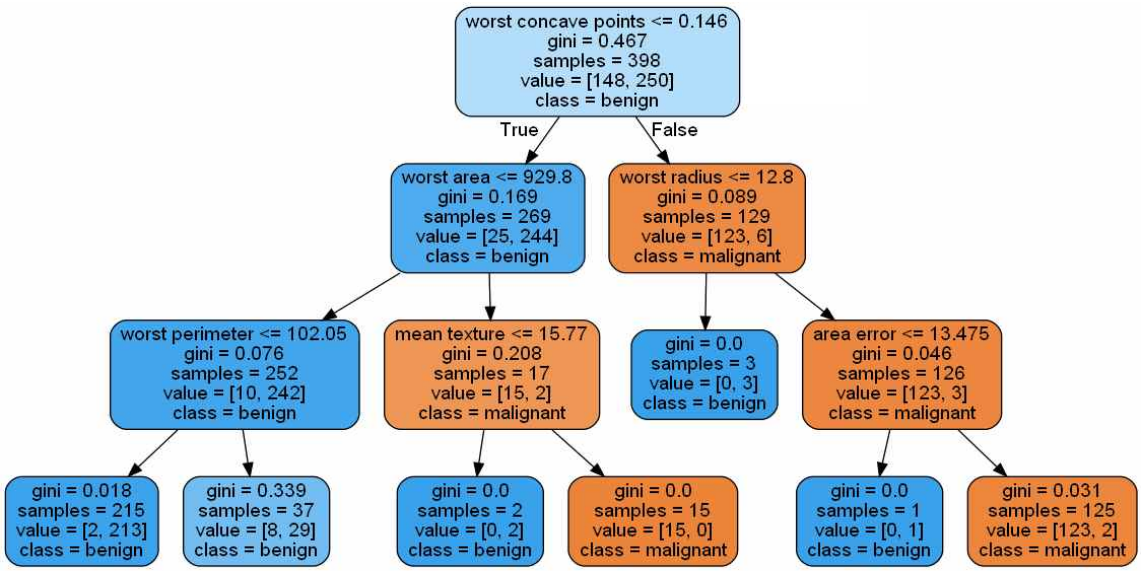
```
tree.plot_tree(result, class_names=load_df.target_names, feature_names=data.columns)
```



분류 나무 그림을 보다 선명하고, 색감을 통해 판단하기 쉽도록 새로운 코드를 통해 구현하고자 한다. (불순도 여부와 깊이에 따른 다른 결과 모두 아래의 코드를 사용한다.)

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree1.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 worst concave points를 기준으로 분류되었다. 종양의 오목한 점이 0.146보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 분류되었다. 다음 중간마디(internal node)는 worst area, worst radius로 나뉘었으며 각각 지

니계수는 0.169, 0.089이다.
둘레(worst 값)에 의해 분류된 끝노드(terminal node)는 모두 양성(benign)으로 분류되었으며, 질감(mean)은 15.77보다 작으면 양성(benign), 15.77값보다 크면 악성(malignant)로 분류되었다. 또한, 면적(standard error)이 13.475보다 작거나 같으면 완벽히 양성(benign)을 분류하였고, 13.475보다 크면 악성(malignant)로 분류되었다.
추가적으로, 노드에 색이 모두 표시되었는데 지니계수의 값이 모두 0.5 이하로 순도가 높게 나타났음을 알 수 있다.

3.1.2. Impurity: Gini Index & Max depth: 4

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

앞선 코드와 같이 DecisionTreeClassifier를 호출하여 실행하며, 불순도 측도 기준은 지니 지수로 같으나, 최대 깊이를 4로 설정하여 X_train과 y_train 적합을 진행한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9849246231155779
0.9415204678362573
```

train data의 정확도는 약 98.5%, test set은 약 94.1%의 정확도를 보여준다. 즉, test data가 약 94.1% 분류하였다는 것을 의미한다. max_depth가 3일 때보다 조금 낮은 정확도를 보이고 있다.

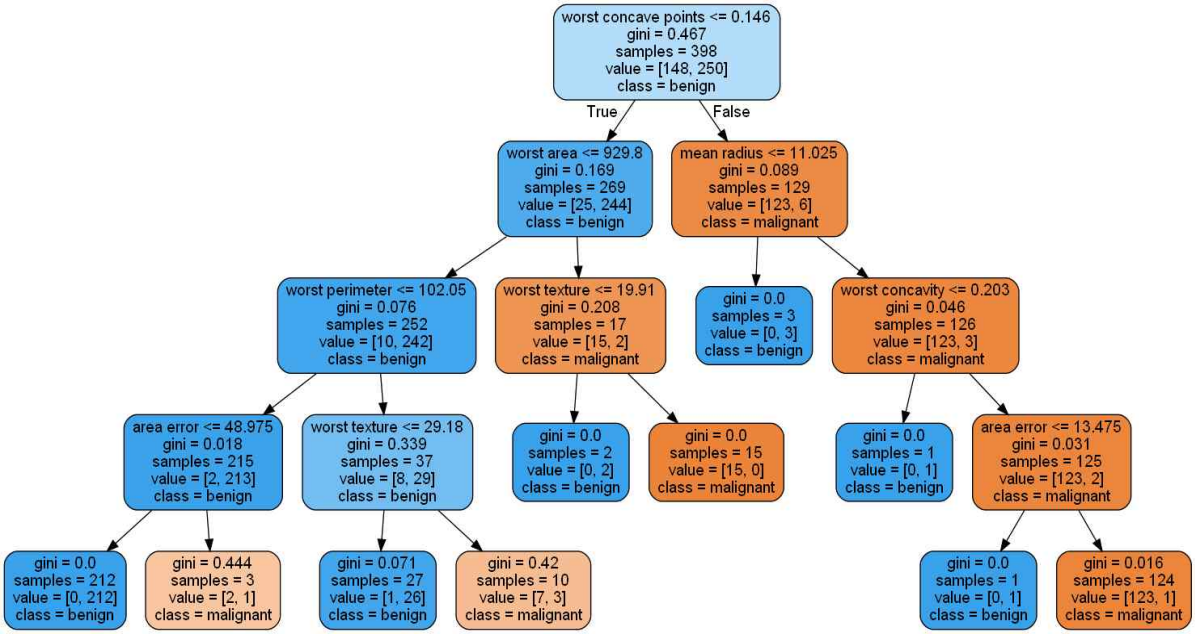
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 60   4]
 [  6 101]]
```

오차 행렬(confusion matrix)을 통한 분류 결과 0 값은 4개의 오분류가 나타났으며, 1 값은 6개의 오분류가 나타났다. 이 결과 또한 max_depth가 3일 때보다 적합이 낮게 나왔음을 확인할 수 있다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree2.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 worst concave points를 기준으로 분류되었다. 종양의 오목한 점이 0.146보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 분류되었다. 다음 중간마디(internal node)는 worst area, mean radius로 나뉘었으며 각각 지니계수는 0.169, 0.089이다. 최대 나무 깊이가 3일 때에 비해 증가한 모습을 확인할 수 있으며, terminal node의 지니계수 값이 전체적으로 증가하였다.

3.1.3. Impurity: Gini Index & Max depth: 5

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

DecisionTreeClassifier를 호출하여 모델링을 진행한다.
불순도는 지니 지수, 최대 깊이는 5, random state는 1로 설정하여 적합한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9974874371859297
0.9415204678362573
```

train data의 정확도는 약 99.7%, test set은 약 94.1%의 정확도를 보여준다. 즉, test data가 약 94.1% 분류하였다는 것을 의미한다. max_depth가 4일때와 거의 비슷한 값을 갖는다.

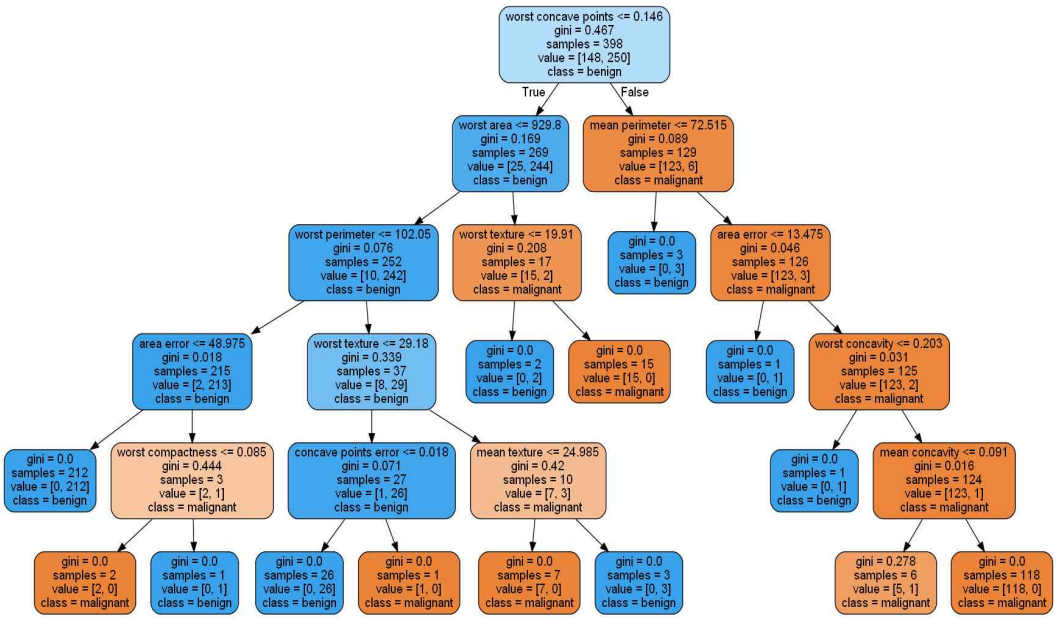
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 60   4]
 [  6 101]]
```

오차 행렬(confusion matrix)을 통한 분류 결과 0 값은 4개의 오분류가 나타났으며, 1 값은 6개의 오분류가 나타났다. 최대 깊이를 4로 진행했을 때와 같은 오차 행렬 값을 갖는다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree3.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



불순도 기준을 지니 지수로 설정하고, 최대 나무 깊이를 5로 했을 때의 의사결정나무는 이와 같다. 뿌리마디(root node)는 worst concave points를 기준으로 분류되었다. 종양의 오목한 점이 0.146보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 분류된 것은 최대 나무 깊이가 3, 4일때와 동일하다.

3.2 Cross Entropy

3.2.0. Impurity: Cross Entropy & Max depth: 2

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

DecisionTreeClassifier를 호출하여 모델링을 진행한다. 위의 코드에서 진행한 형식은 거의 같으나, 불순도 측도 기준이 지니 지수에서 엔트로피로 변경되었다는 것이다. 엔트로피 값과 최대 나무 깊이를 2으로 두고 적합을 진행한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9221105527638191
0.8947368421052632
```

train data의 정확도는 약 92.2%, test set은 약 89.5%의 정확도를 보여준다. 즉, test data가 약 89.5% 분류하였다는 것을 의미한다.

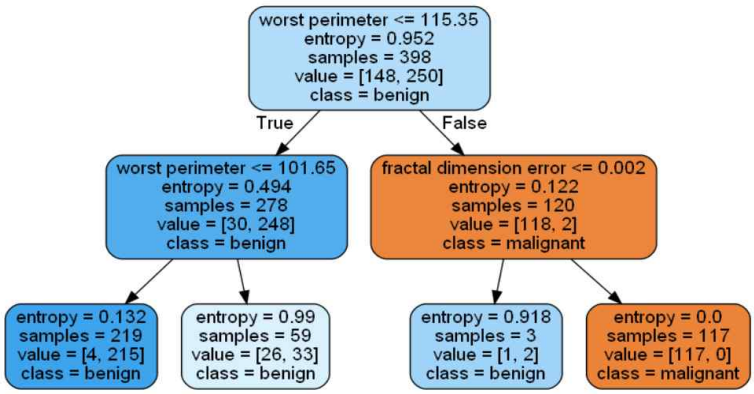
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 47  17]
 [  1 106]]
```

오차 행렬(confusion matrix)을 통한 분류 결과를 확인했을 때 0 값은 17개의 오분류가 나타났으며, 1 값은 1개의 오분류가 나타났다. 오분류율이 크게 나타났음을 확인할 수 있다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 둘레 값(worst) 기준으로 분류되었다. 둘레가 115.35보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로분류되었다. 다음 중간마디(internal node)는 둘레 값(worst), 프랙탈 차원값(standard error)으로 나뉘었으며 각각 지니계수는 0.494, 0.122으로 나타났다.

3.2.1. Impurity: Cross Entropy & Max depth: 3

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

불순도 기준은 엔토로피 값, 최대 나무 깊이는 3으로 두고 적합시킨다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9748743718592965
0.9473684210526315
```

train data의 정확도는 약 97.5%, test set은 약 94.7%의 정확도를 보여준다. 즉, test data가 약 94.7% 분류하였다는 것을 의미한다.

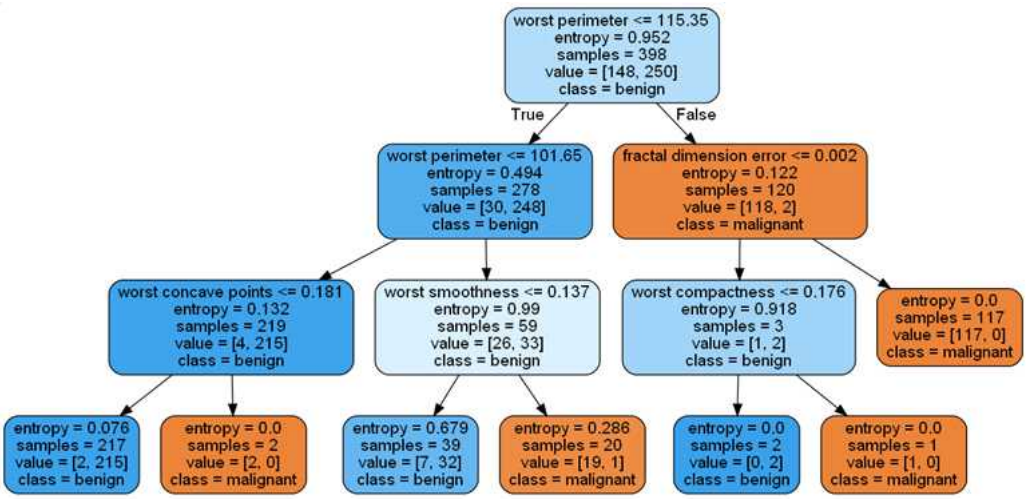
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 58   6]
 [  3 104]]
```

오차 행렬(confusion matrix)을 통한 분류 결과 0 값은 6개의 오분류가 나타났으며, 1 값은 3개의 오분류가 나타났다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 둘레 값(worst) 기준으로 분류되었다. 둘레가 115.35보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 분류되었다. 다음 중간마디(internal node)는 둘레 값(worst), 프랙탈 차원값(standard error)으로 나뉘었으며 각각 지니계수는 0.494, 0.122이다. 끝노드(terminal node)는 convave points(worst), smoothness(worst), compactness(worst) 값으로 악성과 양성을 분류하게 된다.

3.2.2. Impurity: Cross Entropy & Max depth: 4

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

불순도 측도 기준은 위와 같이 엔트로피로 지정하고, 최대 나무 깊이는 4, random_state는 1로 두고 적합을 진행한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9824120603015075
0.9473684210526315
```

train data의 정확도는 약 98.2%, test set은 약 94.7%의 정확도를 보여준다. 즉, test data가 약 94.7% 분류하였다는 것을 의미한다. 최대 나무 깊이를 3으로 설정하였을 때보다 train data의 정확도는 높게 나타났으며, test data 정확도는 같다.

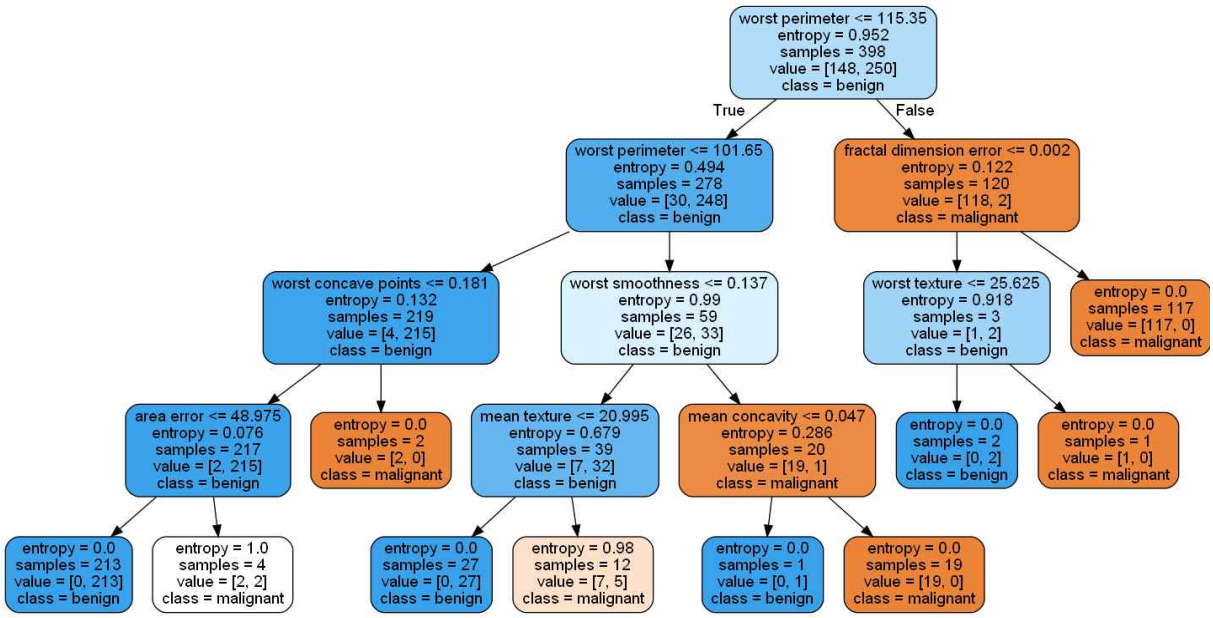
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 60   4]
 [  5 102]]
```

오차 행렬(confusion matrix)을 통한 분류 결과 0 값은 4개의 오분류가 나타났으며, 1 값은 5개의 오분류가 나타났다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



뿌리마디(root node)는 둘레 값(worst) 기준으로 분류되었다. 둘레가 115.35보다 작거나 같으면 양성(benign)으로 분류되었고, 큰 값일때는 악성(malignant)으로 분류되었다. 중간마디와 끝노드의 결정 값은 max_depth=3일때와는 다르게 나타나는 것을 확인할 수 있다.

3.2.3. Impurity: Cross Entropy & Max depth: 5

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=1)
result = dtc.fit(X_train, y_train)  # 적합
y_train_pred = dtc.predict(X_train) # Training accuracy
y_test_pred = dtc.predict(X_test)   # Test accuracy
```

불순도 측도 기준은 엔트로피, 최대 나무 깊이는 5, random_state는 1로 적합을 진행한다.

```
# Accuracy score
from sklearn import metrics
print(metrics.accuracy_score(y_train, y_train_pred))
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
0.9949748743718593
0.9473684210526315
```

train data의 정확도는 약 99.4%, test data는 약 94.7%의 정확도를 보여준다. 즉, test data가 약 94.7% 분류하였다는 것을 의미한다. 최대 나무 깊이를 4로 설정하였을 때보다 train data의 정확도는 높게 나타났으며, test data 정확도는 같다.

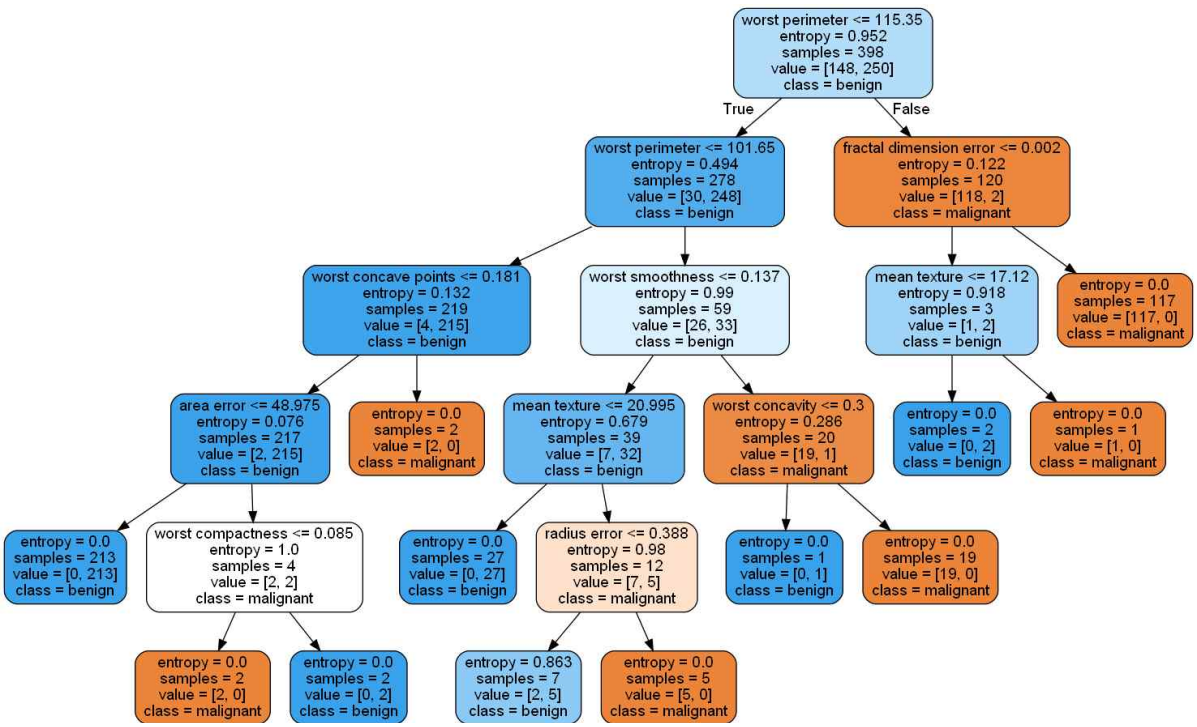
```
# Confusion matrix
print(metrics.confusion_matrix(y_test, y_test_pred))
```

```
[[ 58   6]
 [  3 104]]
```

오차 행렬(confusion matrix)을 통한 분류 결과 0 값은 6개의 오분류가 나타났으며, 1 값은 3개의 오분류가 나타났다.

```
# Graphic
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True, class_names=load_df.target_names,
feature_names=data.columns, out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree4.png')

from IPython.display import Image
Image(graph.create_png())  # Console 창에 결과 그림 출력
```



불순도 기준을 엔트로피로 설정하고, 최대 나무 깊이를 5로 했을 때의 의사결정나무는 이와 같다. 끝노드(terminal node)의 값이 대부분 엔트로피 값이 0으로 적합된 모습을 확인할 수 있다. 하나를 제외하고 모두 분류가 된 상황으로써, 과적합을 나타낸다고 할 수 있다. 또한, 엔트로피값이 1인 node는 색이 나타나지 않았다.

Ⅴ. 평가

CART 방법에서 불순도를 기준으로 Gini Index와 Cross Entropy로 각각 적용하고, 최대 나무 깊이를 2부터 5까지 증가시킴으로써 정확도, 오차 행렬, 의사결정나무 그래프를 확인해보았다.

- Gini Index의 train set과 test set의 정확도

Max_Depth / Data	Train set	Test set
2	96.2	94.7
3	97	95.3
4	98.5	94.1
5	99.7	94.1

-Cross Entropy의 train set과 test set의 정확도

Max_Depth / Data	Train set	Test set
2	92.2	89.5
3	97.5	94.7
4	98.2	94.7
5	99.4	94.7

Y train set의 정확도는 gini index와 cross entropy 모두 최대 나무 깊이가 증가할수록 증가하는 추세를 띤다.
Y test set의 정확도는 불순도 기준이 gini index인 경우, max_depth를 2에서 3으로 변경하면 정확도도 함께 증가하는 모습을 확인할 수 있다. (94.7% -> 95.3%) 하지만, max_depth를 4 혹은 5로 증가시켰을 때는 94.1%로 다시 감소하게 된다.
불순도 기준이 cross entropy 기준인 경우, max_depth=2에서 3으로 증가할 때 정확도도 증가한다. 하지만, 깊이를 4, 5로 넓혀갔을 때는 더 이상 증가하지 않고 94.7%로 일정한 모습을 확인할 수 있다.
이처럼 Gini index와 Cross entropy 기준 모두 최대 나무 깊이가 3일 때 가장 좋은 정확도를 보인다. max_depth를 증가시키면 오히려 정확도가 떨어지거나, 일정하게 지속되는 모습을 확인할 수 있다. 이는 과적합(over_fitting) 되었다는 것을 의미한다. 따라서, gini

index와 cross entropy 모두 최대 나무 깊이를 3으로 설정하는 것이 가장 좋은 설정값이며, 이때 gini index의 정확도가 cross entropy 값보다 높게 나타남을 확인할 수 있다.