인공지능

과제 #01

데이터셋(play_tennis, iris, adult)

분석 결과 14p

2019.11.20

2017204081 최수지

Python version : 3.7.3

사용 IDE : Anaconda _ Jupyter Notebook

1. tennisTest

[사용 데이터셋 : play_tennis.csv]

패키지 가져오기

In [1]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from IPython.display import Image
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import pydotplus
import os
```

In [1] : sklearn.metrics : 모델 평가에 사용되는 모듈 | classification_report : 주요 분류 측정 항목을 보여주는 보고서 모듈 | confusion_matrix : 오차 행렬 계산 모듈
train_test_split : 배열 및 행렬을 분할하는 모듈
DecisionTreeClassifier 의사결정 트리 분류 모듈
IPython.display import Image : 이미지 객체를 만들어 정보를 보여줌.
import pandas as pd | import numpy as np | import pydotplus | import os
: pandas, numpy 패키지 및 pydotplus, os 모듈을 가져옴.

데이터 가져오기 및 전처리

In [2]:
```
tennis_data = pd.read_csv('C:/Users/Susie/Desktop/play_tennis.csv')
tennis_data
```

Out[2]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

In [3]:
```
tennis_data.outlook = tennis_data.outlook.replace('Sunny', 0)
tennis_data.outlook = tennis_data.outlook.replace('Overcast', 1)
tennis_data.outlook = tennis_data.outlook.replace('Rain', 2)

tennis_data.temp = tennis_data.temp.replace('Hot', 3)
tennis_data.temp = tennis_data.temp.replace('Mild', 4)
tennis_data.temp = tennis_data.temp.replace('Cool', 5)

tennis_data.humidity = tennis_data.humidity.replace('High', 6)
tennis_data.humidity = tennis_data.humidity.replace('Normal', 7)

tennis_data.wind = tennis_data.wind.replace('Weak', 8)
tennis_data.wind = tennis_data.wind.replace('Strong', 9)

tennis_data.play = tennis_data.play.replace('no', 10)
tennis_data.play = tennis_data.play.replace('yes', 11)


tennis_data
```

Out[3]:

	day	outlook	temp	humidity	wind	play
0	D1	0	3	6	8	No
1	D2	0	3	6	9	No
2	D3	1	3	6	8	Yes
3	D4	2	4	6	8	Yes
4	D5	2	5	7	8	Yes
5	D6	2	5	7	9	No
6	D7	1	5	7	9	Yes
7	D8	0	4	6	8	No
8	D9	0	5	7	8	Yes
9	D10	2	4	7	8	Yes
10	D11	0	4	7	9	Yes
11	D12	1	4	6	9	Yes
12	D13	1	3	7	8	Yes
13	D14	2	4	6	9	No

In [3] : 변수 tennis_data의 (outlook, temp, humidity, wind, play)의 (Sunny, Overcast, Rain...) 등을 문자열 타입에서 숫자 타입으로 변환하여 저장.

속성과 클래스 분리

```
In [4]: X = np.array(pd.DataFrame(tennis_data, columns = ['outlook', 'temp', 'humidity', 'wind']))
        y = np.array(pd.DataFrame(tennis_data, columns=['play']))
        X_train, X_test, y_train, y_test = train_test_split(X,y)
```

In [4] : 컬럼(outlook, temp, humidity, wind)들과 컬럼(play)를 데이터프레임 형태로 추출한 뒤 배열형태로 변환하여 각각 변수X, 변수y에 저장.
train_test_split()을 사용하여 train과 test로 구분하여 임의의 개수로 각각 변수 X_train, X_test, y_train, y_test에 저장.

```
In [5]: X_train
Out[5]: array([[0, 3, 6, 9],
               [1, 4, 6, 9],
               [2, 4, 6, 9],
               [1, 3, 7, 8],
               [2, 5, 7, 9],
               [0, 5, 7, 8],
               [1, 3, 6, 8],
               [0, 4, 7, 9],
               [0, 3, 6, 8],
               [2, 4, 7, 8]], dtype=int64)
```

```
In [6]: X_test
Out[6]: array([[1, 5, 7, 9],
               [0, 4, 6, 8],
               [2, 4, 6, 8],
               [2, 5, 7, 8]], dtype=int64)
```

```
In [7]: y_train
Out[7]: array([['No'],
               ['Yes'],
               ['No'],
               ['Yes'],
               ['No'],
               ['Yes'],
               ['Yes'],
               ['Yes'],
               ['No'],
               ['Yes']], dtype=object)
```

```
In [8]: y_test
Out[8]: array([['Yes'],
               ['No'],
               ['Yes'],
               ['Yes']], dtype=object)
```

데이터 학습

```
In [9]: dt_clf = DecisionTreeClassifier(max_depth = 5,criterion = 'entropy')
        dt_clf = dt_clf.fit(X_train, y_train)
```

```
In [10]: dt_prediction = dt_clf.predict(X_test)
```

In [9] : 변수 dt_clf에 의사결정 트리 분류 모듈을 저장, max_depth는 5, 지표는 entropy로 설정. dt_clf의 함수 fit()에 변수 X_train, y_train을 입력하여 의사결정 트리 분류 모델 생성 및 저장.

In [10] : dt_clf의 함수 predict에 X_test 입력 및 예측 값을 dt_prediction에 저장.

의사결정 트리 그래프 만들기

In [11]:
```
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

In [12]:
```
feature_names = tennis_data.columns.tolist()
feature_names = feature_names[0:4]
```

In [13]:
```
target_name = np.array(['Play No', 'Play Yes'])
```

In [14]:
```
dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                   feature_names = feature_names,
                                   class_names = target_name,
                                   filled = True, rounded = True,
                                   special_characters = True)
```

In [15]:
```
dt_graph = pydotplus.graph_from_dot_data(dt_dot_data)
```

In [11] : 그래프를 생성할 수 있는 인터페이스 경로를 추가 설정. Graphviz2.39 소프트웨어의 bin 폴더가 있는 경로인 C:/Program Files (x86)/Graphviz2.38/bin/을 os.pathsep 함수를 이용하여 s.environ[“PATH”]에 동적으로 할당하여 저장

In [12] : 변수 tennis_data의 각 컬럼 이름을 list형태로 변환하여 변수 feature_names에 저장. 이후 변수 feature_names를 슬라이싱([0:4])하여 outlook, temp, humidity, wind의 칼럼 이름을 추출한 뒤 다시 변수 feature_names에 저장

In [13] : Target_class 값 ‘Play No’와 ‘Play Yes’를 배열형태로 변수 target_name에 저장

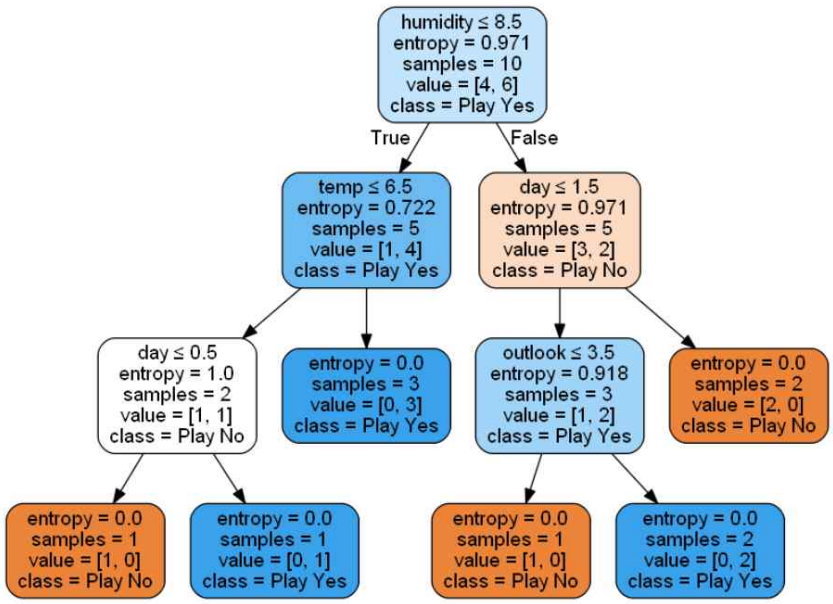
In [14] :

In [15] : dot 형식의 데이터로 정의된 그래프를 불러오는 grap_from_dot_data()함수에 변수 dt_dot_data입력하여 dt_graph에 저장.

시각화 및 정확도 평가

```
In [16]: Image(dt_graph.create_png())
```

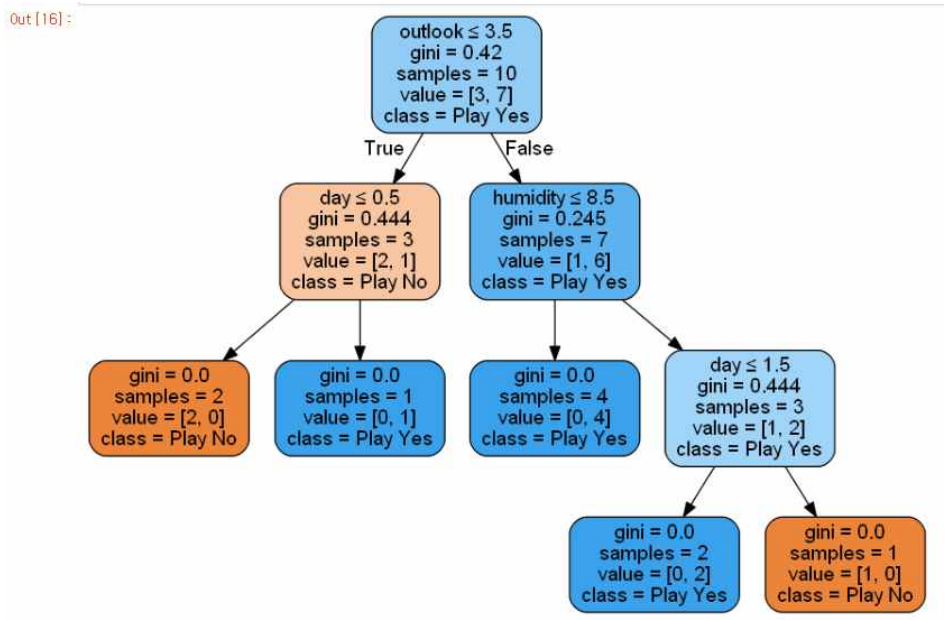
Out[16]:




```
In [17]: print('Accuracy: %.2f' % accuracy_score(y_test, dt_prediction))
Accuracy: 1.00
```

가장 이상적인 결과가 나온 경우는 max_depth : 5 | criterion : entropy로 설정한 경우

Out[16]:




```
In [17]: print('Accuracy: %.2f' % accuracy_score(y_test, dt_prediction))
Accuracy: 0.75
```

criterion : gini 인 경우

총 20번을 진행 해 본 결과 criterion와는 무관하게 정확도가 0.25 / 0.5 / 0.75 / 1.0 의 분포를 보임. 또한 max_depth를 과도하게 적게 설정(1)하는 경우 정확도가 0.25 / 0.5를 맴돌음.

2. irisTest

[사용 데이터셋 : 기본 데이터셋 iris]

패키지 및 데이터 가져오기

```
In [1]: from sklearn import tree
        from sklearn import datasets
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import accuracy_score
        from sklearn.tree import export_graphviz
        from IPython.display import Image
        import numpy as np
        import pydotplus

        iris = datasets.load_iris()
```

In [1] : tennisTest와 유사, sklearn.preprocessing import StandardScaler 은 평균이 0, 표준편차가 1이 되도록 가공하는 모듈이다.

클래스 분리

```
In [2]: X = [[0, 0], [1, 1]]
        Y = [0, 1]

In [3]: clf = tree.DecisionTreeClassifier()
        clf = clf.fit(X, Y)
        clf.predict([[2., 2.]])
Out[3]: array([1])

In [4]: X = iris.data[:, [2, 3]]
        y = iris.target

In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

In [6]: sc = StandardScaler()
        sc.fit(X_train)
Out[6]: StandardScaler(copy=True, with_mean=True, with_std=True)

In [7]: X_train_std = sc.transform(X_train)
        X_test_std = sc.transform(X_test)
```

tennisTest와 과정은 동일하다.

In [5] : 여기서는 train_test_split() 함수 내부에 test_size = 0.3과 random_state = 0을 설정하였다.

데이터 학습 및 의사결정 트리 만들기

```
In [8]: iris_tree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
        iris_tree.fit(X_train, y_train)
        y_pred_tr = iris_tree.predict(X_test)
```

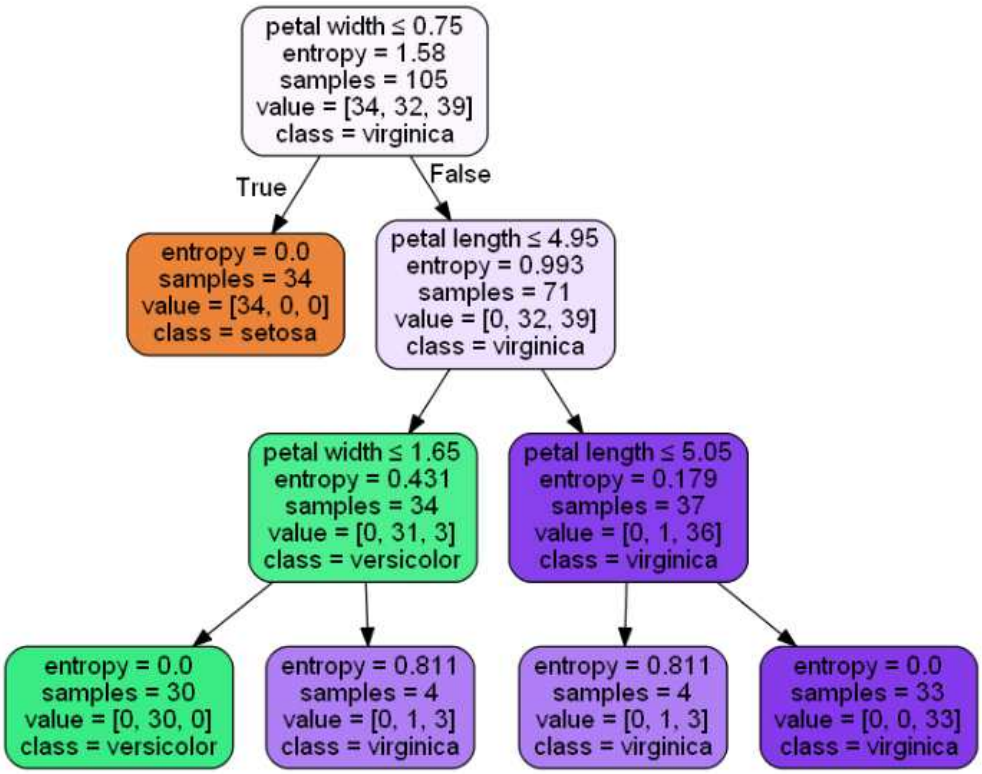
```
In [9]: dot_data = export_graphviz(iris_tree, out_file=None, feature_names=['petal length', 'petal width'],
                                   class_names=iris.target_names, filled=True, rounded=True, special_characters=True)
        graph = pydotplus.graph_from_dot_data(dot_data)
```

이번에는 DecisionTreeClassifier() 함수에 random_state = 0 이란 값을 하나 더 추가하여 넣었다. random_state에 정수 값을 입력하면 숫자를 random하게 생성할 때 사용되는 seed 숫자가 된다.

시각화 및 정확도 평가

```
In [10]: Image(graph.create_png())
```

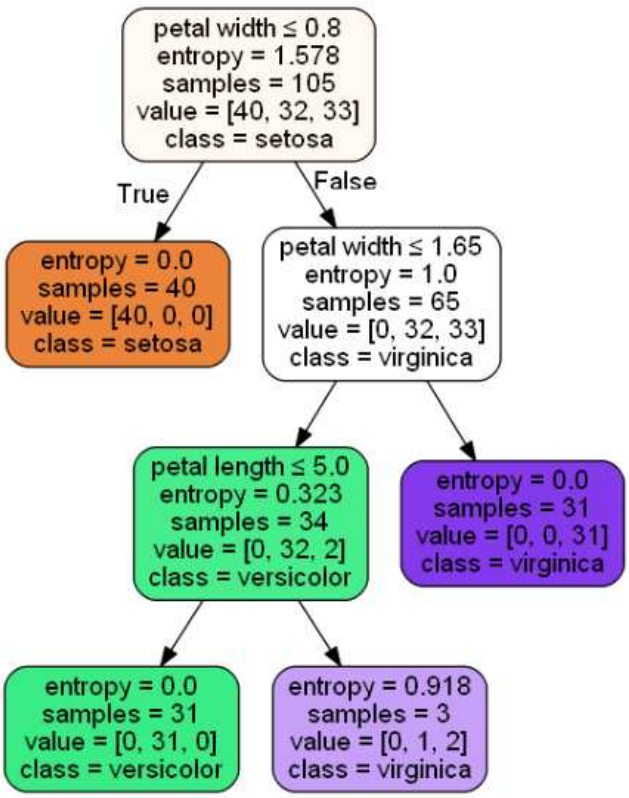
Out[10]:




```
In [11]: print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_tr))
Accuracy: 0.98
```

```
In [10]: Image(graph.create_png())
```

Out[10]:

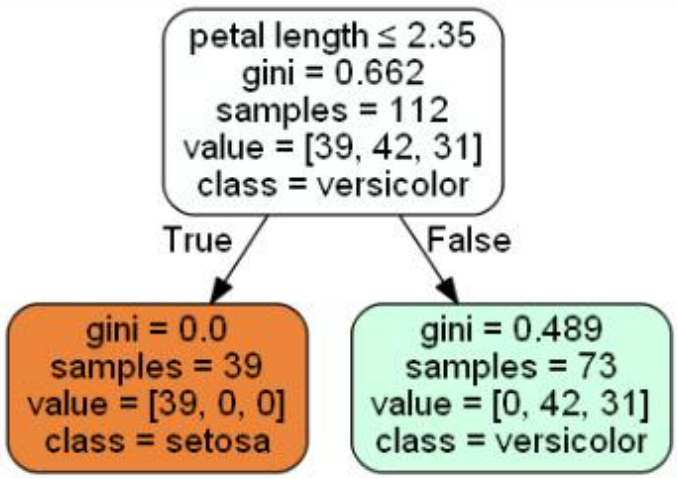



```
In [11]: print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_tr))
Accuracy: 0.93
```

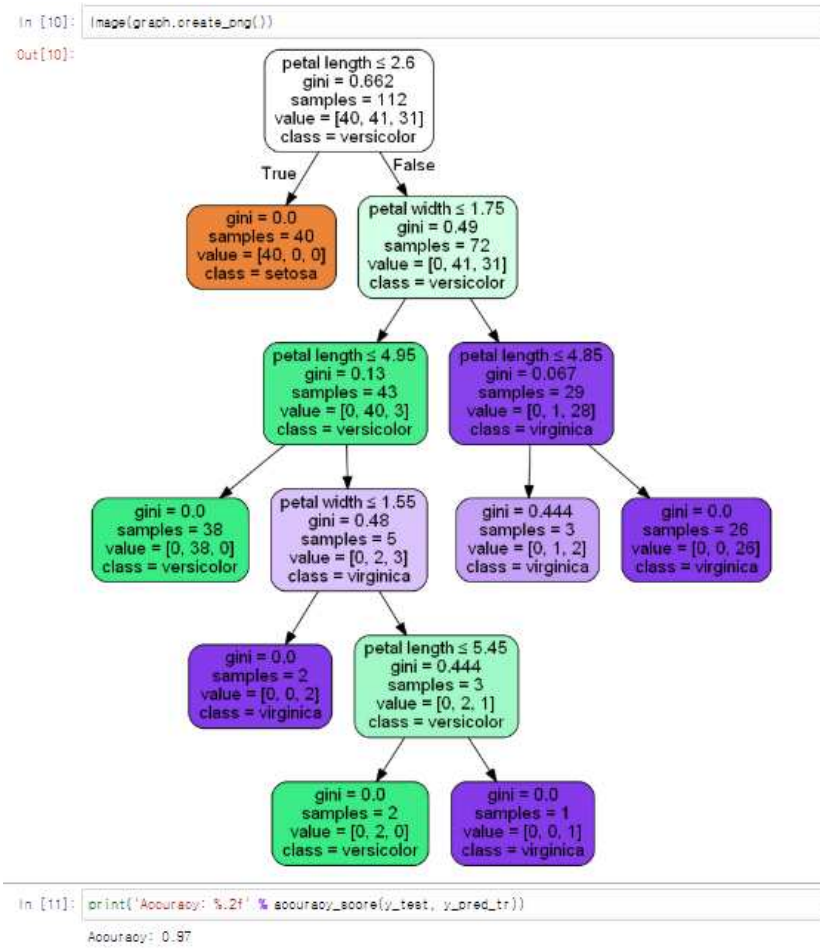
위 2가지는 criterion을 엔트로피로 설정한 경우

```
In [10]: Image(graph.create_png())
```

Out[10]:




```
In [11]: print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_tr))
Accuracy: 0.50
```



위 2가지는 criterion을 gini로 설정한 경우

max_depth | criterion(gini, entropy) | random_state | test size를 20번 조정 해 본 결과 정확도가 최소 0.50(max_depth = 1)에서 최대 0.98(max_depth = 3, criterion = entropy, random_state = 0, test_size = 0.3)사이의 분포를 보였다.

3. loserTest

[사용 데이터셋 : adult.csv]

패키지 가져오기

In [1]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from IPython.display import Image
import pandas as pd
import numpy as np
import pydotplus
import os
```

데이터 전처리

In [2]:
```
dataset = pd.read_csv('C:/Users/Susie/Desktop/adult.csv', header = None, index_col = False
                      ,names=['age','workclass','fnlwgt','education','education.num','marital.status','occupation',
                              'relationship','race','sex','capital.gain','capital.loss','hours.per.week','native.country','income'
])
dataset.columns
```

Out[2]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education.num',
       'marital.status', 'occupation', 'relationship', 'race', 'sex',
       'capital.gain', 'capital.loss', 'hours.per.week', 'native.country',
       'income'],
      dtype='object')
```

In [3]:
```
dataset = dataset[['age','workclass','education','race','sex',
                   'occupation','hours.per.week', 'income']]
dataset = dataset.dropna()
display(dataset.head())
print(dataset.sex.value_counts())
```

	age	workclass	education	race	sex	occupation	hours.per.week	income
1	82	Private	HS-grad	White	Female	Exec-managerial	18	<=50K
3	54	Private	7th-8th	White	Female	Machine-op-inspct	40	<=50K
4	41	Private	Some-college	White	Female	Prof-specialty	40	<=50K
5	34	Private	HS-grad	White	Female	Other-service	45	<=50K
6	38	Private	10th	White	Male	Adm-clerical	40	<=50K

```
Male      20788
Female     9930
Name: sex, dtype: int64
```

In [4]:
```
data_dummies = pd.get_dummies(dataset)
print("get_dummies 후의 특성: \n",list(data_dummies.columns),"\n")
```

```
get_dummies 후의 특성:
 ['age', 'hours.per.week', 'workclass_Federal-gov', 'workclass_Local-gov', 'workclass_Private', 'workclass_Self-emp-inc', 'workclass_Self-emp-not-inc', 'workclass_State-gov', 'workclass_Without-pay', 'education_10th', 'education_11th', 'education_12th', 'education_1st-4th', 'education_5th-6th', 'education_7th-8th', 'education_9th', 'education_Assoc-acdm', 'education_Assoc-voc', 'education_Bachelors', 'education_Doctorate', 'education_HS-grad', 'education_Masters', 'education_Preschool', 'education_Prof-school', 'education_Some-college', 'race_Amer-Indian-Eskimo', 'race_Asian-Pac-Islander', 'race_Black', 'race_Other', 'race_White', 'sex_Female', 'sex_Male', 'occupation_Adm-clerical', 'occupation_Armed-Forces', 'occupation_Craft-repair', 'occupation_Exec-managerial', 'occupation_Farming-fishing', 'occupation_Handlers-cleaners', 'occupation_Machine-op-inspct', 'occupation_Other-service', 'occupation_Priv-house-serv', 'occupation_Prof-specialty', 'occupation_Protective-serv', 'occupation_Sales', 'occupation_Tech-support', 'occupation_Transport-moving', 'income_<=50K', 'income_>50K']
```

In [5]:
```
dataset.workclass = dataset.workclass.replace('Private',1)
dataset.workclass = dataset.workclass.replace('Self-emp-inc',2)
dataset.workclass = dataset.workclass.replace('Self-emp-not-inc',3)
dataset.workclass = dataset.workclass.replace('State-gov',4)
dataset.workclass = dataset.workclass.replace('Local-gov',4)
dataset.workclass = dataset.workclass.replace('Federal-gov',4)
dataset.workclass = dataset.workclass.replace('Without-pay',0)

dataset.education = dataset.education.replace('HS-grad',10)
dataset.education = dataset.education.replace('10th',5)
dataset.education = dataset.education.replace('11th',5)
dataset.education = dataset.education.replace('12th',5)
dataset.education = dataset.education.replace('1st-4th',5)
dataset.education = dataset.education.replace('5th-6th',5)
dataset.education = dataset.education.replace('7th-8th',5)
dataset.education = dataset.education.replace('9th',5)
dataset.education = dataset.education.replace('Assoc-acdm',6)
dataset.education = dataset.education.replace('Assoc-voc',6)
dataset.education = dataset.education.replace('Bachelors',7)
dataset.education = dataset.education.replace('Doctorate',8)
dataset.education = dataset.education.replace('Masters',9)
dataset.education = dataset.education.replace('Preschool',11)
dataset.education = dataset.education.replace('Prof-school',12)
dataset.education = dataset.education.replace('Some-college',13)

dataset.race = dataset.race.replace('Amer-Indian-Eskimo',14)
dataset.race = dataset.race.replace('Asian-Pac-Islander',14)
dataset.race = dataset.race.replace('Black',15)
dataset.race = dataset.race.replace('Other',16)
dataset.race = dataset.race.replace('White',17)

dataset.sex = dataset.sex.replace('Female',18)
dataset.sex = dataset.sex.replace('Male',19)

dataset.occupation = dataset.occupation.replace('Adm-clerical',20)
dataset.occupation = dataset.occupation.replace('Armed-Forces',21)
dataset.occupation = dataset.occupation.replace('Craft-repair',22)
dataset.occupation = dataset.occupation.replace('Exec-managerial',23)
dataset.occupation = dataset.occupation.replace('Farming-fishing',24)
dataset.occupation = dataset.occupation.replace('Handlers-cleaners',25)
dataset.occupation = dataset.occupation.replace('Machine-op-inspct',26)
dataset.occupation = dataset.occupation.replace('Other-service',27)
dataset.occupation = dataset.occupation.replace('Priv-house-serv',28)
dataset.occupation = dataset.occupation.replace('Prof-specialty',29)
dataset.occupation = dataset.occupation.replace('Protective-serv',30)
dataset.occupation = dataset.occupation.replace('Sales',31)
dataset.occupation = dataset.occupation.replace('Tech-support',32)
dataset.occupation = dataset.occupation.replace('Transport-moving',33)

dataset.income = dataset.income.replace('<=50K',99)
dataset.income = dataset.income.replace('>50K',100)
```

In [6]:
```
X = np.array(pd.DataFrame(dataset,
                          columns=['age','workclass','education','race','sex',
                   'occupation','hours.per.week']))
y = np.array(pd.DataFrame(dataset, columns=['income']))
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

```
In [10]: X_train
```

```
Out[10]: array([[25,  1,  9, ..., 19, 23, 40],
                [38,  3,  5, ..., 19, 22, 40],
                [37,  1, 10, ..., 19, 20, 40],
                ...,
                [60,  1,  5, ..., 19, 25, 40],
                [20,  1, 13, ..., 19, 33, 32],
                [38,  1, 10, ..., 18, 20, 35]], dtype=int64)
```

```
In [12]: y_train
```

```
Out[12]: array([[99],
                [99],
                [99],
                ...,
                [99],
                [99],
                [99]], dtype=int64)
```

```
In [11]: X_test
```

```
Out[11]: array([[32,  1,  5, ..., 19, 27, 40],
                [28,  1, 10, ..., 19, 26, 48],
                [50,  1, 12, ..., 19, 29, 60],
                ...,
                [55,  4, 13, ..., 18, 20, 40],
                [18,  1,  5, ..., 18, 31, 20],
                [45,  1,  6, ..., 19, 26, 40]], dtype=int64)
```

```
In [13]: y_test
```

```
Out[13]: array([[ 99],
                [ 99],
                [100],
                ...,
                [100],
                [ 99],
                [100]], dtype=int64)
```

데이터 학습

```
In [14]: dt_clf = DecisionTreeClassifier(max_depth = 3,random_state = 1)
         dt_clf = dt_clf.fit(X_train, y_train)
         dt_prediction = dt_clf.predict(X_test)
```

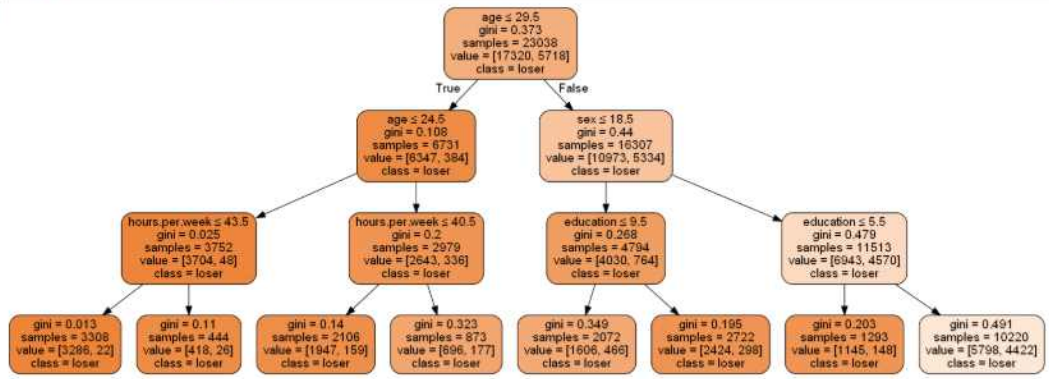
의사결정 트리 그래프 만들기

```
In [15]: os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
         feature_names = dataset.columns.tolist()
         feature_names = feature_names[0:7]
         target_name = np.array(['loser', 'winner'])
         dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                            feature_names = feature_names,
                                            class_names = target_name,
                                            filled = True, rounded = True,
                                            special_characters = True)
```

시각화 및 정확도 평가

```
In [20]: dt_graph = pydotplus.graph_from_dot_data(dt_dot_data)
         Image(dt_graph.create_png())
```

Out[20]:




```
In [21]: print('Accuracy: %.2f' % accuracy_score(y_test, dt_prediction))
```

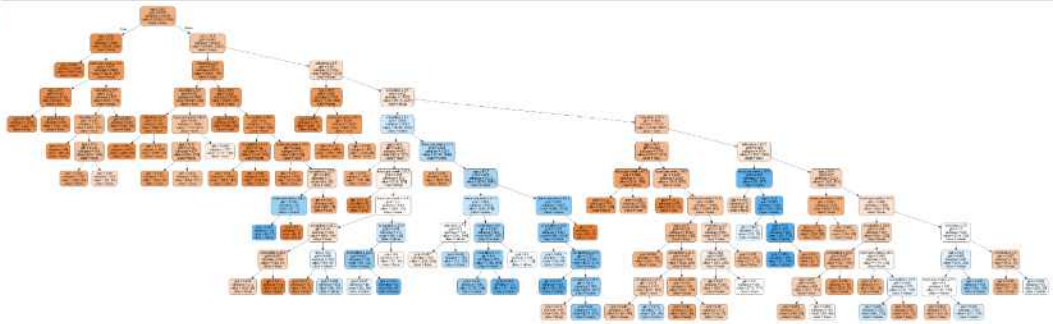
```
Accuracy: 0.75
```

```
In [13]: dt_clf = DecisionTreeClassifier(max_depth = 10,random_state = 0, max_leaf_nodes = 70)
         dt_clf = dt_clf.fit(X_train, y_train)
         dt_prediction = dt_clf.predict(X_test)

In [14]: os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
         feature_names = dataset.columns.tolist()
         feature_names = feature_names[0:7]
         target_name = np.array(['loser', 'winner'])
         dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                            feature_names = feature_names,
                                            class_names = target_name,
                                            filled = True, rounded = True,
                                            special_characters = True)

In [15]: dt_graph = pydotplus.graph_from_dot_data(dt_dot_data)
         Image(dt_graph.create_png())

Out[15]:
```



```
In [16]: print('Accuracy: %.2f' % accuracy_score(y_test, dt_prediction))
         Accuracy: 0.81
```

tennisTest 와는 다르게 max_leaf_nodes를 설정하여 제한을 두었다.

분석

tennisTest에서는 max_depth, criterion의 설정과는 거의 무관하게 정확도가 랜덤한 분포를 보였다.
(엔트로피는 1에 가까울수록, 지니(0.0~0.5)는 0에 가까울수록 잘 분류(질서도가 높음)된 상태)
지니와 엔트로피가 0.0이 아닌 경우에는 수치가 낮고 높음_(위)과는 상관없이 부모 노드에서부터 자식 노드로 뻗어나간다는 것을 알게 되었다.
데이터셋의 전처리 과정 상에서는 문제가 없다고 판단되어 데이터 자체가 적합하지 않은 데이터인 것 같다.
irisTest을 통하여 20번의 기계학습을 거친 후 gini와 entropy는 거의 유사한 분포를 보인다는 것을 알게 되었다. 또한 random_state 값을 고정해두면 다른 수치를 변경하지 않는 이상 트리 모양 및 정확도가 변하지 않는 다는 것을 알게 되었다.
loserTest에서는 횟수를 여러번 반복해도 최소 0.75에서 최대 0.81 사이로 큰 차이를 보이지 않았다. 또한 max_depth가 1이였을 때에도 0.75였던 것을 보아 노드 수가 증가하면 정확도가 좀 더 높아질 수는 있으나, 그것 또한 확률일 뿐 오히려 노드 수가 별 차이가 없으면 경우에 따라 역전할 수 있다는 것을 알았다.