머신러닝 과제 2

201724557 장수현

In [1]:

```
%load_ext watermark
%watermark -v -p numpy,scipy,sklearn,pandas,matplotlib
```

```
CPython 3.7.7
IPython 7.14.0

numpy 1.18.2
scipy 1.4.1
sklearn 0.0
pandas 1.0.3
matplotlib 3.2.1
```

In [2]:

```
import numpy as np
import pandas as pd
```

Iris data

In [3]:

```
from sklearn.datasets import load_iris
iris = load_iris()
iris.keys()
```

Out[3]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [4]:

```
iris['target_names']
```

Out[4]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [5]:

```
iris_pd = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= iris['feature_names']+['target'])
```

In [6]:

```
iris_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    float64
dtypes: float64(5)
```

```
memory usage: 6.0 KB
```

Split data

In [7]:

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(iris_pd, test_size=0.2, random_state=123)
```

In [8]:

```
print(train_set.info())
print(test_set.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 120 entries, 130 to 109
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  120 non-null    float64
 1   sepal width (cm)   120 non-null    float64
 2   petal length (cm)  120 non-null    float64
 3   petal width (cm)   120 non-null    float64
 4   target             120 non-null    float64
dtypes: float64(5)
memory usage: 5.6 KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30 entries, 72 to 4
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  30 non-null     float64
 1   sepal width (cm)   30 non-null     float64
 2   petal length (cm)  30 non-null     float64
 3   petal width (cm)   30 non-null     float64
 4   target             30 non-null     float64
dtypes: float64(5)
memory usage: 1.4 KB
None
```

In [9]:

```
train_set.head(10)
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
130	7.4	2.8	6.1	1.9	2.0
119	6.0	2.2	5.0	1.5	2.0
29	4.7	3.2	1.6	0.2	0.0
0	5.1	3.5	1.4	0.2	0.0
62	6.0	2.2	4.0	1.0	1.0
93	5.0	2.3	3.3	1.0	1.0
131	7.9	3.8	6.4	2.0	2.0
5	5.4	3.9	1.7	0.4	0.0
16	5.4	3.9	1.3	0.4	0.0
82	5.8	2.7	3.9	1.2	1.0

Case1 : criterion = 'entropy' & max_depth = 2

Decision tree training and visualization

In [10]:

```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [11]:

```
features = list(train_set.columns[:-1])
features
```

Out[11]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [12]:

```
X = train_set[features]
y = train_set['target']
```

In [13]:

```
tree_clf.fit(X,y)
```

Out[13]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [14]:

```
from sklearn.tree import export_graphviz
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [15]:

```
import graphviz
from IPython.display import display
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree', directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [16]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [17]:

```
y_pred = tree_clf.predict(X_test)
```

In [18]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  1 10]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.86      1.00      0.92         6
         2.0       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Case2 : criterion = 'entropy' & max_depth = 3

Decision tree training and visualization

In [19]:

```
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [20]:

```
features = list(train_set.columns[:-1])
features
```

Out[20]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [21]:

```
X = train_set[features]
y = train_set['target']
```

In [22]:

```
tree_clf.fit(X,y)
```

Out[22]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [23]:

```
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [24]:

```
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree', directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [25]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [26]:

```
y_pred = tree_clf.predict(X_test)
```

In [27]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  1 10]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.86      1.00      0.92         6
         2.0       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Case3 : criterion = 'entropy' & max_depth = 4

Decision tree training and visualization

In [28]:

```
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [29]:

```
features = list(train_set.columns[:-1])
features
```

Out[29]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [30]:

```
X = train_set[features]
y = train_set['target']
```

In [31]:

```
tree_clf.fit(X,y)
```

Out[31]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [32]:

```
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [33]:

```
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree',directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [34]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [35]:

```
y_pred = tree_clf.predict(X_test)
```

In [36]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  2  9]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.75      1.00      0.86         6
         2.0       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.92      0.94      0.92        30
weighted avg       0.95      0.93      0.93        30
```

Case4 : criterion = 'gini' & max_depth = 2

Decision tree training and visualization

In [37]:

```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=2)
```

In [38]:

```
features = list(train_set.columns[:-1])
features
```

Out[38]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [39]:

```
X = train_set[features]
y = train_set['target']
```

In [40]:

```
tree_clf.fit(X,y)
```

Out[40]:

```
DecisionTreeClassifier(max_depth=2)
```

In [41]:

```
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [42]:

```
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree', directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [43]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [44]:

```
y_pred = tree_clf.predict(X_test)
```

In [45]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  1 10]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.86      1.00      0.92         6
         2.0       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Case5 : criterion = 'gini' & max_depth = 3

Decision tree training and visualization

In [46]:

```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=3)
```

In [47]:

```
features = list(train_set.columns[:-1])
features
```

Out[47]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [48]:

```
X = train_set[features]
y = train_set['target']
```

In [49]:

```
tree_clf.fit(X,y)
```

Out[49]:

```
DecisionTreeClassifier(max_depth=3)
```

In [50]:

```
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [51]:

```
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree', directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [52]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [53]:

```
y_pred = tree_clf.predict(X_test)
```

In [54]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  2  9]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.75      1.00      0.86         6
         2.0       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.92      0.94      0.92        30
weighted avg       0.95      0.93      0.93        30
```

Case6 : criterion = 'gini' & max_depth = 4

Decision tree training and visualization

In [55]:

```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=4)
```

In [56]:

```
features = list(train_set.columns[:-1])
features
```

Out[56]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [57]:

```
X = train_set[features]
y = train_set['target']
```

In [58]:

```
tree_clf.fit(X,y)
```

Out[58]:

```
DecisionTreeClassifier(max_depth=4)
```

In [59]:

```
export_graphviz(
    tree_clf,
    out_file='iris_tree.dot',
    feature_names=features,
    rounded=True,
    filled=True
)
```

In [60]:

```
with open('iris_tree.dot') as f:
    dot_graph = f.read()
dot = graphviz.Source(dot_graph)
dot.format = 'png'
dot.render(filename='iris_tree', directory='./',cleanup=True)
display(dot)
```

Result of Decision Tree

In [61]:

```
X_test = test_set[features]
y_test = test_set['target']
```

In [62]:

```
y_pred = tree_clf.predict(X_test)
```

In [63]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0  0]
 [ 0  6  0]
 [ 0  2  9]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        13
         1.0       0.75      1.00      0.86         6
         2.0       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.92      0.94      0.92        30
weighted avg       0.95      0.93      0.93        30
```

과제 내용

1. DecisionTreeClassifier 함수의 entropy는 무엇인가?

entropy는 불순도 계산 방법으로, 의사결정트리 학습 API인 DecisionTreeClassifier()가 사용하는 분할 기준 중 하나이다. 불순도는 데이터의 순도를 측정하는 척도로, 0에 가까울수록 순도가 높음 즉, 완벽히 불리 되었음을 의미한다. 예를 들어 한 항아리 안에 빨간 공만 들어있다면 불순도는 0이고, 빨간공과 파란공이 섞여있다면 불순도 값이 커지게 된다. 불순도는 정보량 계산에 쓰이고 이후, 이 정보량에 따라 의사결정트리에서 branch할 node의 순서를 정해주게 된다. 주어진 노드 m에 대한 엔트로피 불순도의 공식은 다음과 같다.

$$ – \sum_{j=1}^{J} \frac{c_j}{n_m} \log \Big(\frac{c_j}{n_m} \Big) $$

c_j = 현재 노드의 클래스 j에 해당하는 데이터의 개체(행)의 수

n_m = 현재 노드의 총 행 수

J = 총 클래스 수

의사결정트리의 또 다른 분할 기준이 되는 불순도 계산 방법은 'gini'로, 마찬가지로 정보량 계산에 쓰인다. 지니 불순도의 공식은 다음과 같다.

$$ \sum_{j=1}^{J} p_j(1-p_j) $$

p_j = 집합에서 클래서 j를 지닌 요소를 선택할 확률

J = 총 클래스 수

사실 entropy와 gini 두 criterion에는 똑같이 순도를 계산하기 때문에 큰 차이는 없다. 실제로도 같은 max_depth를 가지는 entropy와 gini(case1&case4, case2&case5, case3&case6)를 비교해보면, 똑같거나 거의 비슷함을 알 수 있다. 다만 case3과 case6에서 차이가 나는데, entropy는 balanced tree를 만들고, gini는 unbalanced tree를 만는다는 차이에서 발생한 것이다. 속도는 entropy가 비교적 느리다.

2. DecisionTreeClassifier 함수의 max_depth는 무엇인가?

의사결정트리를 만들때에는 overfitting을 주의해야한다. overfitting이란 머신러닝을 할 때, branching을 너무 세세하게 하게되면, branch 케이스가 너무 많아져서 시간도 오래걸리고, 예외 케이스에까지 맞춰버리게 버리게 되는 것이다. 이는 비효율적이며, 머신러닝의 의미를 퇴색시킨다. 그래서 pruning 즉, 가지치기가 필요한데, 이것을 max_depth를 감소시키는 것으로 조절해야한다. 그렇다고, 너무 max_depth가 작으면 training data set과의 consistency가 커지게 되어 이또한 머신러닝의 의미를 퇴색시키게된다. 그래서 적절한 max_depth를 설정하는 것이 중요하다. 같은 entropy이고, max_depth만 달라지는 case1 ~ 3을 보면, depth가 가장 작은 case1에서 branch가 가장 적게 일어나고, depth가 가장 큰 case3에서 branch가 가장 많이 일어남을 확인할 수 있다.

In [ ]: