

Machine Learning with Python

Life is too short, You need Python



실습 내용

- 머신러닝 모델링을 위한 코딩은 무조건 할 수 있어야 합니다.
- 코딩 내용을 자세히 알지 못해도 무작정 코딩을 진행해봅니다.
- Iris 데이터를 대상으로 모델링을 진행합니다.
- DecisionTree 알고리즘을 사용합니다.
- 다양한 방법으로 모델 성능을 평가합니다.

1.환경 준비

- 기본 라이브러리와 대상 데이터를 가져와 이후 과정을 준비합니다.



```
In [1]: # 라이브러리 불러오기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

```
warnings.filterwarnings(action='ignore')
%config InlineBackend.figure_format = 'retina'
```

```
In [2]: # 데이터 읽어오기
path = 'https://raw.githubusercontent.com/Jangrae/csv/master/iris.csv'
data = pd.read_csv(path)
```

2.데이터 이해

- 분석할 데이터를 **충분히 이해**할 수 있도록 다양한 **탐색** 과정을 수행합니다.



```
In [3]: # 상위 몇 개 행 확인
data.head()
```

```
Out[3]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [4]: # 하위 몇 개 행 확인
data.tail()
```

```
Out[4]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
In [5]: # 변수 확인
data.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    150 non-null   float64
1   Sepal.Width     150 non-null   float64
2   Petal.Length    150 non-null   float64
3   Petal.Width     150 non-null   float64
4   Species         150 non-null   object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
In [6]: # 기술통계 확인
data.describe()
```

```
Out[6]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
In [7]: # 상관관계 확인
data.corr(numeric_only=True)
```

```
Out[7]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.000000	-0.117570	0.871754	0.817941
Sepal.Width	-0.117570	1.000000	-0.428440	-0.366126
Petal.Length	0.871754	-0.428440	1.000000	0.962865
Petal.Width	0.817941	-0.366126	0.962865	1.000000

3.데이터 준비

- 전처리 과정을 통해 머신러닝 알고리즘에 사용할 수 있는 형태의 데이터를 준비합니다.



1) x, y 분리

- 우선 target 변수를 명확히 지정합니다.
- target을 제외한 나머지 변수들 데이터는 x로 선언합니다.
- target 변수 데이터는 y로 선언합니다.
- 이 결과로 만들어진 x는 데이터프레임, y는 시리즈가 됩니다.
- 이후 모든 작업은 x, y를 대상으로 진행합니다.

```
In [8]: # target 확인
target = 'Species'

# 데이터 분리
x = data.drop(target, axis=1)
y = data.loc[:, target]
```

2) 학습용, 평가용 데이터 분리

- 학습용, 평가용 데이터를 적절한 비율로 분리합니다.
- 반복 실행 시 동일한 결과를 얻기 위해 random_state 옵션을 지정합니다.

```
In [9]: # 모듈 불러오기
from sklearn.model_selection import train_test_split

# 7:3으로 분리
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
```

4.모델링

- 본격적으로 모델을 선언하고 학습하고 평가하는 과정을 진행합니다.
- 우선 회귀 문제인지 분류 문제인지 명확히 구분합니다.



- 회귀 문제 인가요? 분류 문제 인가요?
- 회귀인지 분류인지에 따라 사용할 알고리즘과 평가 방법이 달라집니다.
- 우선 다음 알고리즘을 사용합니다.
 - 알고리즘: DecisionTreeClassifier

```
In [10]: # 1단계: 불러오기
from sklearn.tree import DecisionTreeClassifier
```

```
In [11]: # 2단계: 선언하기
model = DecisionTreeClassifier()
```

```
In [12]: # 3단계: 학습하기
model.fit(x_train, y_train)
```

Out[12]:

▼ DecisionTreeClassifier

DecisionTreeClassifier()

In [13]:

```
# 4단계: 예측하기
y_pred = model.predict(x_test)
```

5.분류 성능 평가

- 다양한 성능 지표로 분류 모델 성능을 평가합니다.

1) Confusion Matrix

In [14]:

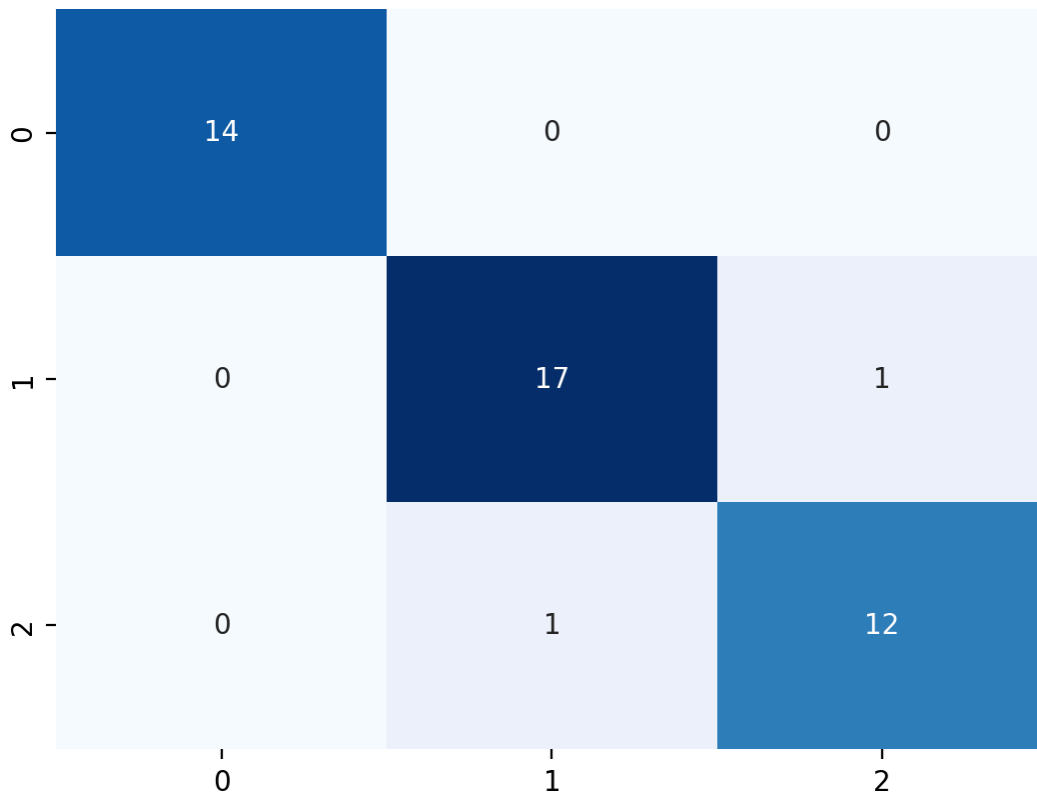
```
# 모듈 불러오기
from sklearn.metrics import confusion_matrix

# 성능 평가
print('Confusion Matrix', confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix [[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
```

In [17]:

```
# 혼동행렬 시각화
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cbar=False, cmap='Blues')
plt.show()
```



2) Accuracy

```
In [18]: # 모듈 불러오기
from sklearn.metrics import accuracy_score

# 성능 평가
print('Accuracy:', accuracy_score(y_test, y_pred))
```

Accuracy: 0.9555555555555556

3) Precision

```
In [19]: # 모듈 불러오기
from sklearn.metrics import precision_score

# 성능 평가
print('Precision:', precision_score(y_test, y_pred, average=None))
```

Precision: [1. 0.94444444 0.92307692]

4) Recall

```
In [20]: # 모듈 불러오기
from sklearn.metrics import recall_score

# 성능 평가
print('Recall:', recall_score(y_test, y_pred, average=None))
```

Recall: [1. 0.94444444 0.92307692]

5) F1-Score

```
In [21]: # 모듈 불러오기
from sklearn.metrics import f1_score

# 성능 평가
print('F1:', f1_score(y_test, y_pred, average=None))
```

F1: [1. 0.94444444 0.92307692]

6) Classification Report

```
In [23]: # 모듈 불러오기
from sklearn.metrics import classification_report

# 성능 평가
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	14
versicolor	0.94	0.94	0.94	18
virginica	0.92	0.92	0.92	13
accuracy			0.96	45
macro avg	0.96	0.96	0.96	45
weighted avg	0.96	0.96	0.96	45

```
In [24]: # 참조
print('학습성능', model.score(x_train, y_train))
```

```
print('평가성능', model.score(x_test, y_test))
```

학습성능 1.0

평가성능 0.9555555555555556

In []: