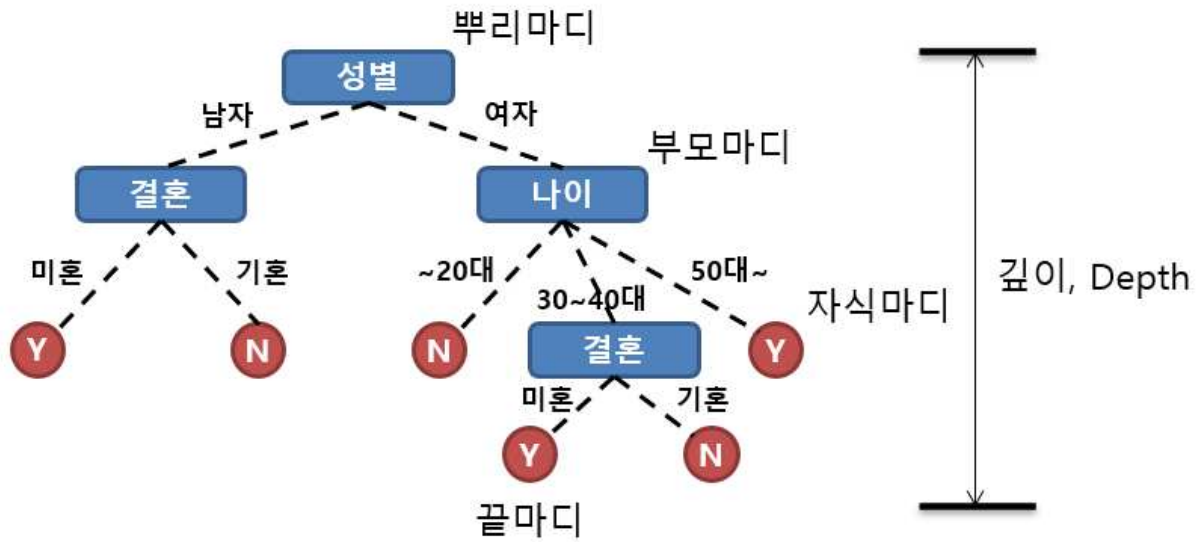
13. Decision Tree

1. 의사결정나무분석의 이해

○ 의사결정나무의 특징

- 의사결정나무는 지도학습 기법으로 각 변수의 영역을 반복적으로 분할함으로써 전체 영역에서의 규칙을 생성함.
- 의사결정나무의 예측력은 다른 지도학습기법들에 비해 대체로 떨어지나 생성된 규칙들이 if-then 형식으로 표현되어 해석력이 좋은 방법임.
- 또한, 분류작업이 용이하고, 연속형 변수와 범주형 변수를 모두 취급할 수 있으며, 모형에 대한 가정이 필요없는 비모수적 방법이라는 장점이 있음.
- 그리고 가장 설명력이 있는 변수에 대하여 최초로 분리가 일어나는 특징을 가지고 있음.



○ 기본구성요소

- 뿌리마디: 시작되는 마디로 전체 자료를 포함함.
- 자식마디: 하나의 마디로부터 분리되어 나간 2개 이상의 마디들을 말함.
- 부모마디: 주어진 마디의 상위마디를 말함.
- 끝마디: 자식마디가 없는 마디를 말함.
- 중간마디: 부모마디와 자식마디가 모두 있는 마디를 말함.

- 가지: 뿌리마디로부터 끝마디까지 연결된 마디들을 말함.
- 깊이: 뿌리마디부터 끝마디까지의 중간마디들의 수를 말함.

○ 의사결정나무의 형성

- 의사결정나무의 형성과정은 크게 성장, 가지치기, 타당성 평가, 해석 및 예측으로 이루어짐.
- 성장 단계는 각 마디에서 적절한 최적의 분리규칙을 찾아서 나무를 성장시키는 과정으로서 적절한 정지규칙을 만족하면 중단함.
- 가지치기 단계는 오차를 크게 할 위험이 높거나 부적절한 추론규칙을 가지고 있는 가지 또는 불필요한 가지를 제거함.
- 타당성 평가 단계에서는 시험자료를 이용하여 의사결정나무를 평가하게 됨.
- 해석 및 예측 단계에서는 구축된 나무모형을 해석하고 예측모형을 설정한 후 예측에 적용함.

- Train 자료 (x_i,y_i), $i=1,2,...,n$, $x_i=(x_{i1},...,x_{ip})$로 나타낸다고 함. 전체 영역을 M개의 영역 $R_1,...,R_M$으로 나누고 각 영역에서 상수값 c_m으로 예측하는 다음과 같은 나무모형을 생각해보도록 함.

$$f(x)=\sum_{m=1}^{M} c_m I(x\in R_m)$$

여기서 c_m과 R_m은 불순도의 측도를 이용하여 값을 정하게 되며, 회귀나무에서는 흔히 오차제곱합 $Q_m(T)=\sum_{i=1}^{n}(y_i-f(x_i))^2$를 그 측도로서 사용함.

- 주어진 분리변수 x_j가 연속형인 경우 분리점을 s라고 하면 두 영역 $R_1(j,s)=\{x:x_j\le s\}$, $R_2(j,s)=\{x:x_j>s\}$를 정의할 수 있음.
- 분리변수가 범주형인 경우에는 전체 범주를 두 개의 부분집합으로 나눔. 그러면 분리기준을 정하는 것은 다음의 최정화 문제로 볼 수 있음.

$$\min\left(\min\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min\sum_{x_i\in R_2(j,s)}(y_1-c_2)^2\right)$$

- 주어진 j와 s에 대하여 최소값을 갖는 해는 $\hat{c_1}$과 $\hat{c_2}$로서 각 $R_1(j,s)$와 $R_2(j,s)$에 속하는 자료의 y_i값들의 평균으로 주어짐.
- 또한, 분리변수가 주어지면 분리점 s는 어렵지 않게 찾을 수 있으므로

적절한 최적화를 통하여 최적 분리기준 (j,s)를 찾을 수 있음. 일단 최적 분리를 찾은 후에는 두 영역에 대하여 동일한 과정을 반복하게 됨.

- 그러면 언제까지 나무모형을 성장시킬 것인지에 대해 생각해야 함.
- 너무 큰 나무모형은 자료를 과대적합하고, 반대로 너무작은 나무모형은 자료를 과소적합할 위험이 있음.
- 즉, 의사결정나무에서는 나무의 크기를 모형의 복잡도로 볼 수 있으며, 최적의 나무 크기는 자료로부터 추정하게 됨.
- 일반적으로 사용되는 방법은 마디에 속하는 자료가 일정수 이하일 때 분할을 정지함.

○ 불순도 측도

- 출력변수가 범주형인 분류나무는 지니지수, 엔트로피지수, 카이제곱통계량 등을 불순도의 측도로 사용하여 나무를 성장시키게 됨. 파이썬에서 제공하는 함수는 지니지수와 엔트로피지수를 통한 불순도 측도를 사용하여 나무를 성장시키게 됨.
- 지니지수는

$$\begin{aligned}&2(P(Left\text{에서 } Y=1)P(Left\text{에서 } Y=0)P(Left)\\&+P(Right\text{에서 } Y=1)P(Right\text{에서 } Y=0)P(Right))\end{aligned}$$

로 정의되며, 지니지수가 최소가 되는 분리를 선택함.

- 엔트로피지수는

엔트로피지수 = 엔트로피(Left) P(Left) + 엔트로피(Right) P(Right)

이고, 엔트로피는

$$\begin{aligned}\text{엔트로피}(Left) = &-P(Left\text{에서 } Y=1)\log_2 P(Left\text{에서 } Y=1)\\&-P(Left\text{에서 } Y=0)\log_2 P(Left\text{에서 } Y=0)\end{aligned}$$

로 정의됨.

2. sklearn을 활용한 의사결정나무분석

- 의사결정나무분석을 실시하기 위해 먼저 sklearn 모듈을 활용한 의사결정나무분석에 대해서 살펴보도록 함.

- 의사결정나무분석은 로지스틱회귀분석, k 근방 분류기법과 마찬가지로 분류를 위한 분석으로, 의사결정나무분석을 통해 출력변수의 값을 분류하여 예측함.
- 따라서 지난 시간 사용하였던 분석방법과 마찬가지로 주어진 데이터를 100% 활용하여 의사결정나무분석을 실시한 후 새롭게 데이터를 모형에 적용하였을 때 예측 성능이 떨어지는 경우가 많음.
- 이러한 현상을 해소하기 위하여 의사결정나무분석 역시 모형을 만드는 Train 데이터와 검정을 하는 Test 데이터를 일정 비율로 나누어 Test 데이터로 모형을 평가하는 데 이를 교차 검증을 이용함.
- 주어진 데이터를 Train 데이터와 Test 데이터로 분할은 지난 로지스틱회귀분석, k 근방 분류기법과 마찬가지로 sklearn.model_selection 모듈의 train_test_split() 함수를 이용하여 실시할 수 있음.

```
from sklearn.model_selection import train_test_split
train 객체명, test 객체명 = train_test_split(데이터셋, test_size=비율,
random_state=번호, stratify = 출력변수)
```

- 의사결정나무분석은 로지스틱회귀분석 및 k 근방 분류기법과 마찬가지로 분류 방법이기 때문에 stratify 옵션을 이용하여 예측의 대상이 되는 출력변수에 대하여 층화추출을 통해 Train 데이터와 Test 데이터를 분류함.
- 그 외 train_test_split() 함수의 사용방법은 지난 로지스틱회귀분석을 참고하시기 바람.

- 그럼 분석에 사용되는 Train 데이터를 활용하여 의사결정나무분석을 실시하는 방법을 알아보도록 함.
- 분석에 사용되는 Train 데이터를 활용하여 의사결정나무분석 실시는 sklearn.tree 모듈의 DecisionTreeClassifier() 함수를 이용하여 실시할 수 있음.

```
from sklearn.tree import DecisionTreeClassifier
DecisionTreeClassifier(criterion= 'gini'/'entropy', max_depth=최대깊이,
min_sample_split=분기 최소 데이터수, min_sample_leap=마디 최소데이터수,
random_state=초기난수값).fit(입력변수, 출력변수)
```

- DecisionTreeClassifier().fit() 함수에 분석에 사용되는 Train 데이터의 입력변수와 출력변수를 입력함.
- 그리고 criterion 옵션을 통해 불순도를 측정하는 지표인 gini와 entropy를 지정할 수 있음. 이 때 gini방법이 기본으로 사용되어짐.
- 그리고 max_depth 옵션을 통해 의사결정나무의 최대깊이를 지정할 수 있음. max_depth 옵션을 통해 의사결정나무가 과도하게 성장하는 것을 막을 수 있음. max_depth 옵션은 None이 기본적으로 지정되기 때문에 max_depth 옵션을 지정하지 않으면 성장할 수 있는 최대 나무 크기로 나무가 성장하게 됨.
- 그리고 min_sample_split 옵션은 분기 최소 데이터수를 지정하는 옵션임. 즉, 노드를 분리할 때 필요한 최소의 데이터수를 지정하는 옵션으로 기본적으로 2의 값이 지정되어 있음.
- 그리고 min_sample_leap 옵션은 마디 최소 데이터수로 마디가 분리되었을 때 최소한 마디에서 가지고 있어야 하는 데이터 수를 지정하는 옵션으로 기본적으로 1의 값이 지정되어 있음.

- 위 명령어를 사용하면 다른 분석들과 마찬가지로 아무런 결과가 출력되지 않고, 분석이 실시되었다는 의미로 DecisionTreeClassifier()이라는 메시지만 출력됨.
- 이는 R과 같은 다른 프로그램들과 마찬가지로 분석이 수행된 결과를 다른 객체에 할당한 후 이를 다양한 함수에 활용하여 결과를 확인할 수 있음.

- 따라서 DecisionTreeClassifier().fit() 함수를 실행시킨 결과를 객체에 할당하여 다음과 같은 함수를 활용하여 의사결정나무분석의 예측값을 확인할 수 있음.

정분류률: 의사결정나무 객체**.score**(입력변수, 출력변수)
출력변수 예측값: 의사결정나무 객체**.predict**(독립변수)
출력변수 확률값: 의사결정나무 객체**.predict_proba**(독립변수)
입력변수의 importance: 의사결정나무 객체**.feature_importances_**

- 의사결정나무분석의 분류정확도를 확인하기 위한 정분류률은 의사결정나무분석의 결과를 할당한 객체를 활용하여 score() 함수를 사용하여 확인할 수 있음. 이 때 score() 함수에 정분류률 계산을 위한 입력변수와 출

력변수를 차례로 입력하면 모형의 정분류율을 확인할 수 있음. 만약 Test 데이터의 입력변수와 출력변수를 입력하면 생성된 의사결정나무분석에 Test 데이터를 적용하였을 때의 정분류율을 확인하여 모형을 평가할 수 있음.

- 그리고 의사결정나무분석에 의해 예측된 출력변수의 값을 확인하기 위해서는 의사결정나무분석의 결과를 할당한 객체를 활용하여 predict() 함수를 사용하여 확인할 수 있음. 이 때 predict() 함수에 출력변수의 예측값을 계산하기 위한 입력변수를 입력하면 출력변수의 예측값을 확인할 수 있음. 만약 Test 데이터의 입력변수를 입력하면 생성된 의사결정나무분석에 Test 데이터를 적용하였을 때의 출력변수의 예측값을 확인할 수 있음.
- 또한, 의사결정나무분석에 의해 예측된 출력변수의 값이 1이 될 확률을 확인하기 위해서는 의사결정나무분석의 결과를 할당한 객체를 활용하여 predict_proba() 함수를 사용하여 확인할 수 있음.
- 그리고 의사결정나무분석에 사용된 입력변수들의 중요도(importance)를 출력하기 위해 feature_importances_ 함수를 사용할 수 있고 이를 통해 입력변수의 순서대로 중요도가 출력되어 확인할 수 있음.

- 그리고 의사결정나무분석 모형 평가에 사용되는 분류표, 정분류율, 평가지표를 확인하기 위해 sklearn.metrics 모듈의 confusion_matrix() 함수, accuracy_score() 함수, classification_report() 함수를 이용하여 실시할 수 있음.

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
분류표: confusion_matrix(출력변수 예측값, 출력변수)
정분류율: accuracy_score(출력변수, 출력변수 예측값)
분류 예측력 평가 지표: classification_report(출력변수, 출력변수 예측값)
```

- 의사결정나무분석을 평가하기 위해 실제 출력변수의 값과 예측값을 이원분할표의 형태로 나타내기 위해 confusion_matrix() 함수를 사용할 수 있음. 이 때, 이원분할표의 행 위치와 열 위치를 차례로 입력하여 분류표의 행과 열에 위치할 값을 지정할 수 있음.
- confusion_matrix() 함수에 의해 나타난 분류표 상의 정분류율을 계산하기 위해 accuracy_score() 함수를 사용할 수 있음.이 때, 출력변수와

출력변수 예측값을 차례로 입력함.

- 의사결정나무분석을 평가하기 위한 지표인 presicion, recall, f1_score를 계산하기 위해 classification_report() 함수를 사용할 수 있음. 이때, 출력변수와 출력변수 예측값을 차례로 입력함.

- 그리고 의사결정나무분석 결과를 그림으로 그리기 위해 sklearn.tree 모듈의 export_graphviz() 함수, pydotplus 모듈의 graph_from_dot_data() 함수, IPython.display 모듈의 Image() 함수를 이용하여 그릴 수 있음.

```
from sklearn.tree import export_graphviz
객체명1 = export_graphviz(의사결정나무 객체, filled=마디 채우기, rounded=마디 끝모양, class_names=출력변수 클래스 이름, feature_names=입력변수 이름)
from pydotplus import graph_from_dot_data
객체명2 = graph_from_dot_data(객체명1)
from IPython.display import Image
그래프 출력: Image(객체명2.create_png())
```

- 먼저 의사결정나무분석 결과의 그림의 내용을 설정하기 위해 export_graphviz() 함수를 사용함. export_graphviz() 함수에 먼저 의사결정나무 결과를 할당한 객체를 입력하고, filled 옵션과 rounded 옵션등을 이용하여 마디의 형태를 설정해 줌.
- 그런 뒤 class_names 옵션에 출력변수의 클래스 이름을 설정하여 그림에서 쉽게 확인할 수 있게하고, feature_names 옵션에 입력변수 이름을 차례로 입력하여 역시나 그림에서 쉽게 입력변수의 이름을 확인할 수 있게 함.
- 이렇게 설정한 의사결정나무 그림의 내용을 객체에 저장하고, 이 객체를 graph_from_dot_data() 함수에 대입하여 그림을 그려줌.
- 하지만 graph_from_dot_data() 함수를 실행하면 콘솔화면에서 바로 확인할 수 없고 write_png() 함수를 통해 결과를 컴퓨터에 저장하거나 Image() 함수를 통해 결과를 콘솔화면에 출력하여 의사결정나무 그림을 확인할 수 있음.

- seaborn 모듈의 iris 데이터에서, species를 예측하기 위해

sepal_length, sepal_width, petal_length, petal_width를 사용한 의사결정나무분석을 실시하고자 함.

- info() 함수를 활용하여 자료를 확인한 결과 분석에 사용되는 모든 변수에서 결측자료가 없는 것을 확인할 수 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('iris')

In [3]: df.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
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [4]: from sklearn.model_selection import train_test_split

In [5]: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [6]: y_train=train['species']
   ...: X_train=train[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
   ...: y_test=test['species']
   ...: X_test=test[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- Train 데이터의 입력변수 자료인 X_train과 Train 데이터의 출력변수 자료인 y_train을 각각 DecisionTreeClassifier().fit() 함수에 대입하여 의사결정나무분석을 실시하였음.

```
In [7]: from sklearn.tree import DecisionTreeClassifier

In [8]: result = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=1).fit(X_train, y_train)
```

- 이 때 불순도 지표를 gini로 사용하였고, 최대깊이를 3, 분기 최소 데이터수와 끝마디 최소 데이터수는 기본값을 사용하였음.
- DecisionTreeClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 result라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측

확률값을 계산하였음.

```
In [9]: result.predict_proba(X_train)[:5]
Out[9]:
array([[1.     , 0.     , 0.     ],
       [1.     , 0.     , 0.     ],
       [0.     , 0.03125, 0.96875],
       [0.     , 0.03125, 0.96875],
       [1.     , 0.     , 0.     ]])
```

- predict_prob() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 각 클래스별 확률을 계산한 결과를 확인할 수 있음. 이 때 각 클래스별 확률에서 가장 높은 확률을 가지는 클래스를 예측값으로 분류하게 됨.

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 Train 데이터의 출력변수 예측값을 y_train_pred 객체에 할당하여 의사결정나무분석을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [11]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [12]: confusion_matrix(y_train_pred, y_train)
Out[12]:
array([[35,  0,  0],
       [ 0, 34,  4],
       [ 0,  1, 31]], dtype=int64)

In [13]: accuracy_score(y_train_pred, y_train)
Out[13]: 0.9523809523809523

In [14]: dtc_report = classification_report(y_train, y_train_pred)

In [15]: print(dtc_report)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        35
  versicolor       0.89      0.97      0.93        35
   virginica       0.97      0.89      0.93        35

    accuracy                           0.95       105
   macro avg       0.95      0.95      0.95       105
weighted avg       0.95      0.95      0.95       105
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 setosa, versicolor, virginica를 각각 35개, 34개, 31개 데이터에 대해서 제대로 분류하여 정분류율는 0.95238로 나타남. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 0.89, 0.97, 0.93으로 나타

났고, species가 virginica인 경우 precision, recall, f1-score는 각각 0.97, 0.89, 0.93으로 나타남.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test를 대입하여 Test 데이터의 종속변수 예측값을 y_test_pred 객체에 할당하여 의사결정나무분석을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

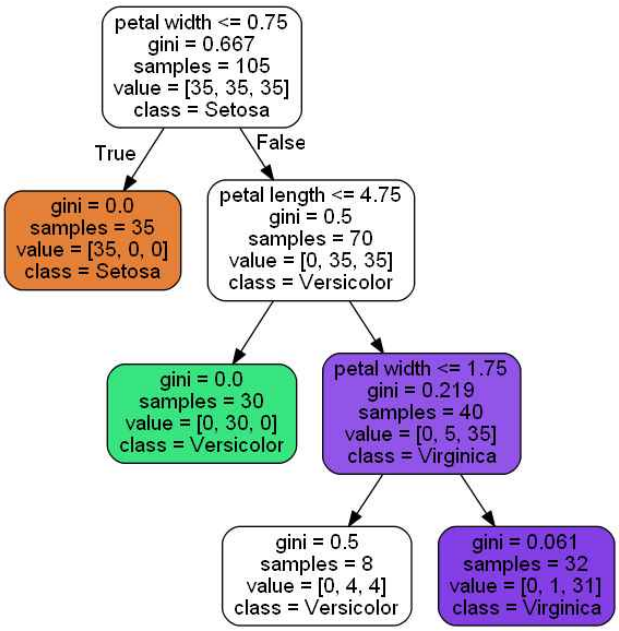
```
In [16]: y_test_pred = result.predict(X_test)

In [17]: from sklearn.metrics import accuracy_score, confusion_matrix

In [18]: confusion_matrix(y_test_pred, y_test)
Out[18]:
array([[15,  0,  0],
       [ 0, 15,  1],
       [ 0,  0, 14]], dtype=int64)

In [19]: accuracy_score(y_test_pred, y_test)
Out[19]: 0.9777777777777777

In [20]: dtc_report1 = classification_report(y_test, y_test_pred)

In [21]: print(dtc_report1)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 setosa, versicolor, virginica를 각각 15개, 15개, 14개 데이터에 대해서 제대로 분류하여 정분류율는 0.97777로 나타남. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 0.94, 1.00, 0.97로 나타났고, species가 virginica인 경우 precision, recall, f1-score는 각각 1.00, 0.93, 0.97로 나타남.
- 이는 Train 데이터를 활용하여 나타난 결과와 큰 차이가 존재하지 않는다는 것을 확인할 수 있음.

- 다음으로 의사결정나무분석의 결과를 그림으로 나타내기 위하여 export_graphviz() 함수, graph_from_dot_data() 함수, Image() 함수를 사용하였음.

```
In [22]: from pydotplus import graph_from_dot_data

In [23]: from sklearn.tree import export_graphviz

In [24]: dot_data = export_graphviz(result, filled=True, rounded=True,
    ...:                            class_names=['Setosa', 'Versicolor', 'Virginica'],
    ...:                            feature_names=['sepal length', 'sepal width', 'petal length', 'petal width'])

In [25]: graph = graph_from_dot_data(dot_data)

In [26]: from IPython.display import Image

In [27]: Image(graph.create_png())
```

- export_graphviz() 함수에 의사결정나무분석 결과 객체를 입력하고, class_names에 출력변수의 클래스 이름인 Setosa, Versicolor, Virginica를 입력하고, 입력변수의 이름인 sepal length, sepal width, petal length, petal width를 입력하였음.
- export_graphviz() 함수의 결과를 dot_data라는 객체에 할당하여 graph_from_dot_data() 함수에 대입하여 Image() 함수를 통해 의사결정나무 그림을 출력하였음.



- 그 결과 petal width가 가정 먼저 나타난 분류 기준이었고, 0.75를 기준으로 petal width가 0.75보다 작으면 Setosa로 분류되고 0.75보다 크면 Versicolor로 분류되는 것을 확인할 수 있음. petal width가 0.75보다 큰 마디는 추가적으로 petal length에 의해 분류되어 나무가 성장하는 것을 알 수 있음.

```
In [31]: result.feature_importances_
Out[31]: array([0.        , 0.        , 0.4097561, 0.5902439])
```

- feture_importances_ 함수를 통해 petal width의 중요가 0.59024로 가장 높은 것을 알 수 있고, 그 다음으로 petal length의 중요도가 0.40975인 것을 확인할 수 있음.

3. 의사결정나무분석의 실습

※ seaborn 모듈의 penguins 데이터에서, species를 예측하기 위해 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g을 사용한 의사결정나무분석을 실시하세요.

- info() 함수를 활용하여 자료를 확인한 결과 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g에서 결측자료가 존재하는 것을 확인할 수 있음. 따라서 dropna() 함수를 이용하여 결측값을 제거할 필요가 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('penguins')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB

In [4]: df = df.dropna(subset=['species', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
   ...:                how='any', axis=0)
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [5]: from sklearn.model_selection import train_test_split

In [6]: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [7]: y_train=train['species']
   ...: X_train=train[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
   ...: y_test=test['species']
   ...: X_test=test[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- Train 데이터의 입력변수 자료인 X_train과 Train 데이터의 출력변수 자료인 y_train을 각각 DecisionTreeClassifier().fit() 함수에 대입하여 의사결정나무분석을 실시하였음.

```
In [8]: from sklearn.tree import DecisionTreeClassifier

In [9]: result = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=1).fit(X_train, y_train)
```

- 이 때 불순도 지표를 gini로 사용하였고, 최대깊이를 3, 분기 최소 데이터수와 끝마디 최소 데이터수는 기본값을 사용하였음.
- DecisionTreeClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 result라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측확률값을 계산하였음.

```
In [10]: result.predict_proba(X_train)[:5]
Out[10]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

- predict_prob() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 각 클래스별 확률을 계산한 결과를 확인할 수 있음. 이 때 각 클래스별 확률에서 가장 높은 확률을 가지는 클래스를 예측값으로 분류하게 됨.

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 Train 데이터의 출력변수 예측값을 y_train_pred 객체에 할당하여 의사결정나무분석을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [11]: y_train_pred = result.predict(X_train)

In [12]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [13]: confusion_matrix(y_train_pred, y_train)
Out[13]:
array([[106,   5,   1],
       [  0,  42,   0],
       [  0,   0,  85]], dtype=int64)

In [14]: accuracy_score(y_train_pred, y_train)
Out[14]: 0.9748953974895398

In [15]: dtc_report = classification_report(y_train, y_train_pred)

In [16]: print(dtc_report)
              precision    recall  f1-score   support

      Adelie       0.95      1.00      0.97       106
   Chinstrap       1.00      0.89      0.94        47
      Gentoo       1.00      0.99      0.99        86

    accuracy                           0.97       239
   macro avg       0.98      0.96      0.97       239
weighted avg       0.98      0.97      0.97       239
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 106개, 42개, 85개 데이터에 대해서 제대로 분류하여 정분류율는 0.974895로 나타남. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.95, 1.00, 0.97로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 1.00, 0.89, 0.94로 나타났고, species가 Gentoo인 경우 precision, recall, f1-score는 각각 1.00, 0.99, 0.99로 나타남.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test를 대입하여 Test 데이터의 종속변수 예측값을 y_test_pred 객체에 할당하여 의사결정나무분석을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

```
In [17]: y_test_pred = result.predict(X_test)

In [18]: from sklearn.metrics import accuracy_score, confusion_matrix

In [19]: confusion_matrix(y_test_pred, y_test)
Out[19]:
array([[44,  5,  1],
       [ 1, 16,  0],
       [ 0,  0, 36]], dtype=int64)

In [20]: accuracy_score(y_test_pred, y_test)
Out[20]: 0.9320388349514563

In [21]: dtc_report1 = classification_report(y_test, y_test_pred)

In [22]: print(dtc_report1)
              precision    recall  f1-score   support

      Adelie       0.88      0.98      0.93        45
   Chinstrap       0.94      0.76      0.84        21
      Gentoo       1.00      0.97      0.99        37

    accuracy                           0.93       103
   macro avg       0.94      0.90      0.92       103
weighted avg       0.94      0.93      0.93       103
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 44개, 16개, 36개 데이터에 대해서 제대로 분류하여 정분류율는 0.9320388로 나타남. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.88, 0.98, 0.93로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 0.94, 0.76, 0.84로 나타났고, species가 Gentoo인 경우 precision, recall, f1-score는 각각 1.00, 0.97, 0.99로 나타남.
- 이는 Train 데이터를 활용하여 나타난 결과와 큰 차이가 존재하지 않는다는 것을 확인할 수 있음.

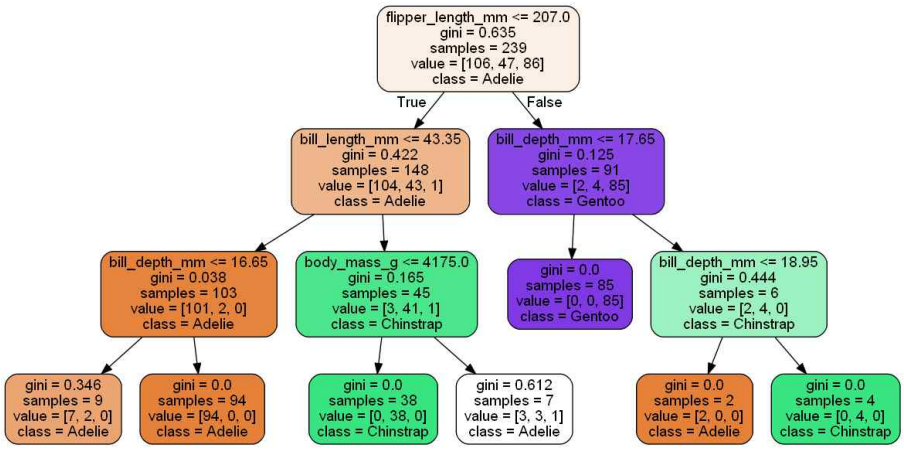
- 다음으로 의사결정나무분석의 결과를 그림으로 나타내기 위하여 export_graphviz() 함수, graph_from_dot_data() 함수, Image() 함수를 사용하였음.

```
In [23]: from pydotplus import graph_from_dot_data

In [24]: from sklearn.tree import export_graphviz

In [25]: dot_data = export_graphviz(result, filled=True, rounded=True,
    ...:                            class_names=['Adelie', 'Chinstrap', 'Gentoo'],
    ...:                            feature_names=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'])

In [26]: graph = graph_from_dot_data(dot_data)

In [27]: from IPython.display import Image

In [28]: Image(graph.create_png())
```

- export_graphviz() 함수에 의사결정나무분석 결과 객체를 입력하고, class_names에 출력변수의 클래스 이름인 Adelie, Chinstrap, Gentoo를 입력하고, 입력변수의 이름인 bill_length_mm, bill_depth_mm,

flipper_length_mm, body_mass_g를 입력하였음.

- export_graphviz() 함수의 결과를 dot_data라는 객체에 할당하여 graph_from_dot_data() 함수에 대입하여 Image() 함수를 통해 의사결정나무 그림을 출력하였음.



- 그 결과 flipper length mm가 가정 먼저 나타난 분류 기준이었고, 207을 기준으로 flipper length mm가 0.75보다 작으면 Adelie로 분류되고 207보다 크면 Gentoo로 분류되는 것을 확인할 수 있음. 그리고 bill length mm에 따라서 추가적으로 분류되어 나무가 성장하는 것을 알 수 있음.

```
In [29]: result.feature_importances_
Out[29]: array([0.35369487, 0.08445732, 0.54012719, 0.02172062])
```

- feture_importances_ 함수를 통해 flipper length mm의 중요가 0.540127로 가장 높은 것을 알 수 있고, 그 다음으로 bill length mm의 중요도가 0.35369, bill depth mm의 중요도가 0.084457, body mass g의 중요도가 0.021720인 것을 확인할 수 있음.