1. 붓꽃의 품종 분류

(1) 데이터 적재

-scikit-learn 의 데이터셋 모듈에 포함되어있다.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("iris_dataset의 키 : \n{}".format(iris_dataset.keys()))

#iris_dataset의 키 :
#dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'fi

print(iris_dataset.DESCR)

print(iris_dataset['DESCR'][:200]+"\n...") #데이터셋 설명 앞부분만(200글자)
```

- 예측하려는 붓꽃 품종의 이름을 가지고 있는 key : target_names

```
format("타겟의 이름: {}".format(iris_dataset['target_names']))

#"타겟의 이름: ['setosa' 'versicolor' 'virginica']"
```

- 특성을 설명하는 문자열 리스트 : feature_names

```
format("특성의 이름 : {}".format(iris_dataset['feature_names']))

#"특성의 이름 : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)
                'petal width (cm)']"
```

> 실제 데이터(target, data)중 data는
> 꽃잎의 길이와 폭, 꽃받침의 길이와 폭을 수치 값으로 가지고 있는 Numpy 배열

```
print("data의 타입 : {}".format(type(iris_dataset['data'])))

#data의 타입 : <class 'numpy.ndarray'>
```

```
print("data의 크기 : {}".format(iris_dataset['data'].shape))

# data의 크기 : (150, 4)
```

> 배열의 행은 개개의 꽃, 열은 각 꽃의 측정치

이 배열은 150개의 붓꽃 데이터를 가지고 있으며, 각 붓꽃마다 4개의 측정치를 가지고 있음.

머신러닝에서 각 아이템은 샘플이라 하고 속성은 특성이라고 부름.
그러므로 data배열의 크기는 150x4가 됨
이는 scikit-learn의 스타일이며 항상 데이터가 이런 구조일거라 가정하고

```
print("data의 처음 다섯 행 : \n{}".format(iris_dataset.data[:5]))

#data의 처음 다섯 행 :
#[[5.1 3.5 1.4 0.2]
# [4.9 3.  1.4 0.2]
# [4.7 3.2 1.3 0.2]
# [4.6 3.1 1.5 0.2]
# [5.  3.6 1.4 0.2]]

# - 1열 : 꽃받침의 길이
# - 2열 : 꽃받침의 폭
# - 3열 : 꽃잎의 길이
# - 4열 : 꽃잎의 폭
```

- target 배열 : 샘플 붓꽃의 품종을 담은 Numpy 배열

```
print("data의 타입 : {}".format(type(iris_dataset.target)))
#data의 타입 : <class 'numpy.ndarray'>

print("data의 타입 : {}".format(iris_dataset.target.shape))
# 1차원 배열 (150,)
```

```
print("타겟 : \n{}".format(iris_dataset.target))
# 0:setosa, 1:versicolor, 2:virginica
#타겟 :
#[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
# 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
# 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
# 2 2]
```

---

(2) 훈련데이터와 테스트 데이터

- 머신러닝 모델을 만들 때 사용하는 훈련데이터와 모델이 얼마나 잘 작동하는지 측정하는 테스트데이터로 나눈다.
- scikit-learn 은 데이터셋을 섞어서 나눠주는 train_test_split 함수 제공
  훈련 세트 : 75%, 테스트세트 : 25%
- scikit-learn 에서 데이터는 대문자 X로 표시하는 레이블은 소문자 y로 표기한다.
  이는 수학에서 함수의 입력을 x, 출력을 y로 나타내는 표준공식 f(x) = y에서 유래된 것이다.
- 수학의 표기 방식을 따르되 데이터는 2차원 배열(행렬)이므로 대문자X를,
  타깃은 1차원 배열(벡터)이므로 소문자y를 사용

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris_dataset.data,
                                                    iris_dataset.target,
                                                    random_state=0)
```

> train 데이터와, test 데이터로 나누기 전에 무작위로 섞어주지 않으면
> 순서대로 나누어 지기 때문에 y_test(테스트레이블) 값이 모두 2가 나오게 된다.

세 클래스(품종) 중 하나만 포함한 테스트 세트를 사용하면
모델이 얼마나 잘 일반화 되었는지 알 수 없다.

테스트 세트는 모든 클래스의 데이터를 포함하도록 잘 섞어야 한다.
random_state =0은 이 함수를 여러번 실행해도 같은 랜덤값이 리턴된다.

```
print("X_train 크기 : {}".format(X_train.shape)) # (112, 4)
print("y_train 크기 : {}".format(y_train.shape)) # (112,)
print("X_test 크기 : {}".format(X_test.shape)) # (38, 4)
print("y_test 크기 : {}".format(y_test.shape)) # (38,)
```

(3) 데이터 살펴보기

- 머신러닝 모델을 만들기 전에 머신러닝 없이도 풀 수 있는 문제가 아닌지,
  혹은 필요한 정보가 누락되어 있는지 데이터를 조사해 보는것이 좋다.
- 실제 데이터에는 일관성이 없거나 이상한 값이 들어가 있는 경우가 종종 있다.

산점도 행렬을 통해 데이터의 특성을 찾아보자

- 산점도 : 여러 변수로 이루어진 자료에서 두 변수끼리 짝을 지어 작성된 산점도를 행렬 형태로 배열

```
# X_train 데이터를 사용해서 데이터 프레임을 만든다.
iris_dataframe = pd.DataFrame(X_train,
                              columns = iris_dataset.feature_names)

iris_dataframe.head()

pd.plotting.scatter_matrix(iris_dataframe, c=y_train,
                           figsize=(15,15),
                           marker='o', hist_kwds={'bins':20},
                           s=60,alpha=.8)
```

> 세 클래스가 꽃잎과 꽃받침의 측정값에 따라
> 비교적 잘 구분되어 있는 것을 볼 수 있다.
> 클래스 구분을 위한 머신러닝 기법을 사용하면 잘 구분 될 것이다.

(4) K- 최근접 이웃(k_nearest neighbors, k-nn) 알고리즘을 이용한 머신러닝

- 훈련 데이터를 통해 모델이 만들어지고
  새로운 데이터가 들어오면 가까운 훈련 데이터 포인트를 찾아 분류한다.
- scikit-learn의 모든 머신러닝 모델은
  Estimator라는 파이썬 클래스로 각각 구현되어 있다.

- k-최근접 이웃 분류 알고리즘은
  neighbors 모듈 아래 KNeighborsClassifier 클래스에 구현되어 있다.
- 모델을 사용하기 위해 클래스로부터 객체를 만들고 parameter를 설정한다.
- 가장 중요한 이웃의 개수를 1로 지정하고 모델을 만들어보자.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 1)
# - 훈련 데이터 셋으로부터 모델을 만들기 위해 fit 메서드 사용
knn.fit(X_train, y_train)

# - fit 메서드는 knn 객체 자체를 변환시키면서 반환 시킨다.
# KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
#                      metric_params=None, n_jobs=None, n_neighbors=1, p=2,
#                      weights='uniform')
```

(5) 예측하기

- 위에서 만든 모델을 사용해서 새 데이터에 대한 예측을 만들 수 있다.
- 야생에서 꽃받침의 길이는 3cm, 폭은 4.2cm,
  꽃잎의 길이는 0.8cm, 폭은 0.4cm인 붓꽃을 찾았다고 가정하고 이 붓꽃의 품종을 찾아보자
- 측정값은 numpy배열로 만드는데,
  하나의 붓꽃 샘플(1) 에 4가지 특성(4)이 있으므로 1 by 4 배열을 만들어야 한다.
  (붓꽃 데이터가 이렇게 되어있기 때문에 이렇게 만든다.)
- 붓꽃 하나의 측정값은 2차원 numpy 배열에 행으로 들어가므로,
  scikit-learn은 항상 데이터가 2차원 배열일 것으로 예상

```
X_new = np.array([[3, 4.2, 0.8, 0.4]]) # 1차원 배열로 만들면 오류가 난다.
X_new

print("X_new.shape : {}".format(X_new.shape))
# X_new.shape : (1, 4)

prediction = knn.predict(X_new)
print("예측 : {}".format(prediction)) # [0] --> 0은 세토사

print("예측한 붓꽃의 이름 : {}".format(iris_dataset['target_names'][predictio
# 예측한 붓꽃의 이름 : ['setosa']
```

```
# - 하나의 입력, 특성을 가진 값이 아니기 때문에
# 아래와 같이 벡터형태로 나타내면 에러가 난다.
X_new2 = np.array([3, 4.2, 0.8, 0.4])
X_new2prediction = knn.predict(X_new2)
#ValueError: Expected 2D array, got 1D array instead:
#array=[3.  4.2 0.8 0.4].
#Reshape your data either using array.reshape(-1, 1)
#if your data has a single feature or array.reshape(1, -1) if it contains a
```

> X_new = np.array([[3, 4.2, 0.8, 0.4]]) 이와 같은 형태의 배열이어야 한다.

(6) 모델 평가

- 앞에서 만든 테스트 셋을 가지고
  현재 만든 학습 모델이 잘 만들어 졌는지 확인해보자

```
y_pred = knn.predict(X_test)
# 만들어진 학습 모델을 가지고 테스트 데이터의 붓꽃품종을 예측한다.

y_pred
# 테스트 데이터의 예측 값
#array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
#       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])

y_pred == y_test
# 예측 품종과 실제 품종이 같으면 true
#array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
#        True,  True,  True,  True,  True,  True,  True,  True,  True,
#        True,  True,  True,  True,  True,  True,  True,  True,  True,
#        True,  True,  True,  True,  True,  True,  True,  True,  True,
#        True, False])
```

테스트 세트의 정확도

```
# y_pred = knn.predict(X_test)
print("테스트 세트의 정확도 : {:.4f}% ".format(np.mean(y_pred==y_test)*100))
#테스트 세트의 정확도 : 97.3684%

# knn 객체의 score 메서드 사용
```

```
print("테스트 세트의 정확도 : {:.4f} %".format(knn.score(X_test, y_test)*100))
#테스트 세트의 정확도 : 97.3684 %

# sklearn.metrics 의 accuracy_score 사용
from sklearn import metrics

# y_pred = knn.predict(X_test)
print("테스트 세트의 정확도 : {:.4f} %".format(metrics.accuracy_score(y_test,
#테스트 세트의 정확도 : 97.3684 %
```

(7) k값 변경

```
accuracy_set = []
k_set = [1,3,5,7,9,11]

for k in k_set:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    accuracy_set.append(accuracy)

from pprint import pprint
pprint(accuracy_set)
#[0.9736842105263158,
# 0.9736842105263158,
# 0.9736842105263158,
# 0.9736842105263158,
# 0.9736842105263158,
# 0.9736842105263158]

max(accuracy_set)
#0.9736842105263158
```