성능평가 방법

iris 데이터에셋에 대해 KNN과 나이브 베이즈를 통해 각각 분류 모델을 구현하고 두 모델의 하드웨어적 성능을 비교

#01. 패키지 참조

```
import warnings
warnings.filterwarnings('ignore')

import os
import numpy as np
import seaborn as sb
from matplotlib import pyplot as plt
from pandas import read_excel, pivot_table
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
# GaussianNB - 연속형 변수, CategoricalNB - 범주형 변수, MultinomialNB - 다항분포형 변수
from sklearn.naive_bayes import GaussianNB
```

#02. 데이터 가져오기

```
origin = read_excel("https://data.hossam.kr/G02/iris.xlsx")
origin.head()
```

	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

```
x = origin.drop('Species', axis=1)
y = origin['Species']
x.shape, y.shape
```

```
((150, 4), (150,))
```

#03. KNN 데이터 분류

`%%timeit` 명령이 상단에 명시된 블록은 기본적으로 100회 실행한다.

각 회차마다의 실행시간에 대한 평균을 마지막에 출력한다.

`-r회차 -n회차` 옵션을 추가하면 반복 회차를 직접 조절할 수 있다.

- `-r` : 몇 번 loop 돌 것인지를 설정
- `-n` : 각 loop당 몇 번 실행할 것인지를 설정

1. 100회 반복 후 평균 실행 시간 측정

```
#%%timeit   # ⟵ 실습에서는 주석 해제하세요.

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)
y_pred = knn.predict(x)
score = accuracy_score(y.values, y_pred)
print("정확도: %.2f%%" % (score*100))
```

```
정확도: 96.00%
```

2. 1회만 실행 후 시간 측정

```
%%timeit -r1 -n1

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)
y_pred = knn.predict(x)
score = accuracy_score(y.values, y_pred)
print("정확도: %.2f%%" % (score*100))
```

```
정확도: 96.00%
16 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

#04. 나이브 베이즈 모델

클래스가 주어졌을 때 독립변수의 조건부 확률에 조건부 독립 가정을 추가한 분류기

각 사건이 독립이라는 가정

ex) 스팸메일 분류기

> https://scikit-learn.org/stable/modules/naive_bayes.html

1회만 실행 후 시간 측정

```
%%timeit -r1 -n1

nb = GaussianNB()
nb.fit(x, y)
y_pred = nb.predict(x)
score = accuracy_score(y.values, y_pred)
print("정확도: %.2f%%" % (score*100))
```

```
정확도: 96.00%
6.27 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

#05. 학습과 예측에 대한 실행시간 비교

1. 학습 시간 비교

```
nb = GaussianNB()
%timeit -r1 -n1 nb.fit(x, y)

knn = KNeighborsClassifier(n_neighbors=3)
%timeit -r1 -n1 knn.fit(x, y)
```

```
2.38 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
1.61 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

> knn이 학습에 걸리는 시간은 짧다.

2. 예측 시간 비교

```
nb = GaussianNB()
nb.fit(x, y)
%timeit -r1 -n1 nb.predict(x)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)
%timeit -r1 -n1 knn.predict(x)
```

```
1.75 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
11.3 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

#06. 사용되는 메모리 계산

`memory_profiler` 패키지 설치가 필요하다.

코드 블록 상단에 `%%memit` 명령을 명시하면 해당 블록이 실행되면서 소비되는 메모리 용량을 산정한다.

외부 라이브러리 로딩

```
%load_ext memory_profiler
```

```
%%memit

nb = GaussianNB()
nb.fit(x, y)
y_pred = nb.predict(x)
```

```
score = accuracy_score(y.values, y_pred)
print("정확도: %.2f%%" % (score*100))
```

```
정확도: 96.00%
peak memory: 249.21 MiB, increment: 0.02 MiB
```

```
%%memit

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)
y_pred = knn.predict(x)
score = accuracy_score(y.values, y_pred)
print("정확도: %.2f%%" % (score*100))
```

```
정확도: 96.00%
peak memory: 249.20 MiB, increment: 0.00 MiB
```