[Scikit-Learn]

1. scikit-learn이란?

- 머신러닝 알고리즘을 구현한 오픈소스 라이브러리 중 가장 유명한 라이브러리 중 하나
- 일관되고 간결한 라이브러리라는 장점이 있으며, 문서화가 잘 되어있음
- 데이터 셋은 Numpy 배열, Pandas DataFrame, Scipy 희소행렬, scikit-leaen bunch 를 사용 가능

2. scikit_learn 알고리즘

1) 분류 (classification) 알고리즘

(1) 로지스틱 회귀 (Logistic Regression) : 선형 회귀를 이진 분류에 적용한 모델로, 확률 기반의 분류를 수행

(2) K-최근접 이웃 (K-Nearest Neighbors) : 주어진 데이터의 가장 가까운 이웃들을 기반으로 분류를 수행하는 모델

(3) 나이브 베이즈 (Naive Bayes) : 베이즈 정리를 기반으로 한 확률적 분류 모델

(4) 신경망 (Neural Networks) : 인공 신경망을 사용하여 복잡한 비선형 분류 문제를 처리하는 모델

2) 회귀 (Regression) 알고리즘

(1) 선형 회귀 (Linear Regression) : 선형 모델을 사용하여 연속적인 값을 예측하는 회귀 알고리즘

(2) 의사결정 트리 (Decision Trees) : 트리 기반의 회귀 알고리즘으로, 데이터를 반복적으로 분할하여 예측을 수행

3) 클러스터링 (clustering)

(1) K-평균 클러스터링 (K-Means Clustering) : 지정한 클러스터의 개수(K)에 따라 데이터를 그룹화하는 알고리즘

(2) 가우시안 혼합 모델 (Gaussian Mixture Models) : 여러 개의 가우시안 분포를 혼합하여 복잡한 데이터 분포를 모델링하는 알고리즘

3. 데이터 구조

[번치(Bunch)]

- scikit-learn에서 사용되는 특별한 데이터 구조

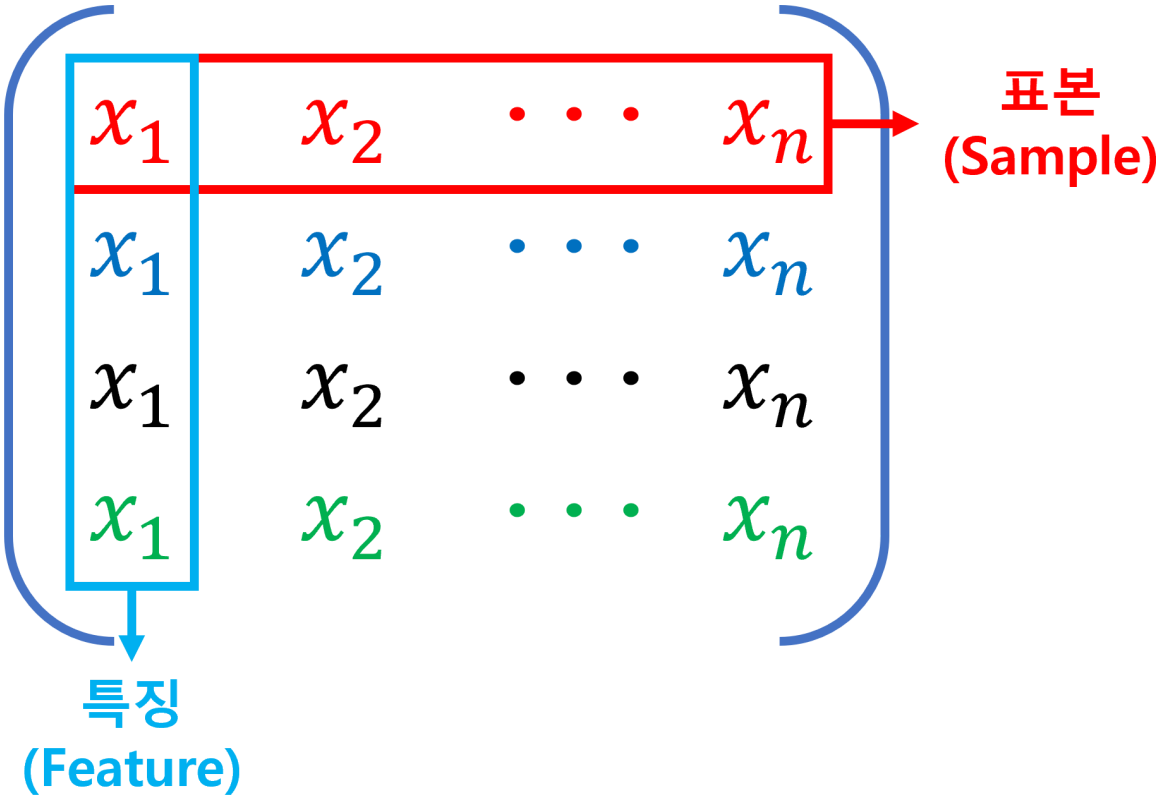
- 번치는 딕셔너리(Dictionary)와 유사한 형태를 가지며, 특성 데이터와 타겟 데이터를 각각의 키로 저장

- 데이터셋의 특성 데이터, 타겟 데이터, 특성 이름, 타겟 값의 이름 등을 저장

In [ ]:
```python
from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data  # 특성 데이터
y = iris.target  # 타겟 데이터

# 번치 객체 속성 확인

#print(iris.feature_names)  # 특성 이름
#print(iris.data)           # 특성 데이터
print(iris.target_names)   # 레이블 이름
print(iris.target)         # 레이블 데이터
#print(iris.DESCR)          # 데이터셋 설명
```

4. 데이터 표현 방식

1) 특징 행렬 (Feature Matrix)

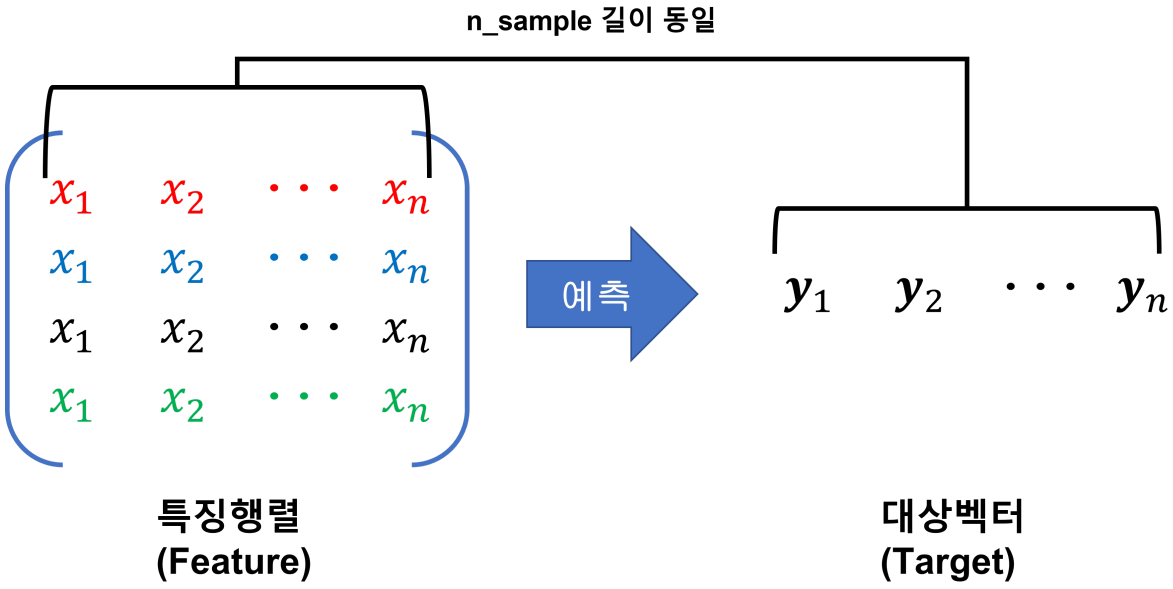


- Sample (표본)은 데이터셋이 설명하는 개별 객체를 의미
- Feature (특징)은 표본을 연속적인 수치값, 부울값, 이산값으로 표현하는 개별 관측치 의미
- 표본은 행렬의 행, 특징은 행렬의 열로 표현
- n_samples : 행의 개수, n_features : 열의 개수
- 특징행렬은 변수 X에 저장
- [n_samples, n_features] 형태의 2차원 배열 구조

2) 대상 벡터 (Target Vector)

- 특징행렬을 통해 얻고 싶은 값
- n_samples : 길이
- 대상벡터는 변수 y에 저장
- 1차원 배열 구조

3) 특징행렬 - 대상벡터



- 특징행렬을 학습 통해 대상벡터 예측
- 특징행렬과 대상벡터의 길이는 동일

5. Scikit-Learn을 이용한 머신러닝 순서

6. Scikit_Learn을 이용한 머신러닝 실습 - 선형회귀

1). 데이터 준비

- 학습데이터 X, y 준비

(1) Numpy의 ndarray를 이용한 데이터 생성

In [ ]:
```
import numpy as np
import random

np_x = np.random.uniform(1, 5, size = 100)
np_y = 10 * np_x + 3

# 1차원 배열 구조를 2차원 배열구조로 reshape

np_X = np_x.reshape(-1, 1)

np_X.shape, np_y.shape

np_x
```

(2) Pandas의 DataFrame 이용한 데이터 생성

In [ ]:
```
import pandas as pd

# pandas
# 1차원 : series
# 2차원 : DataFrame

pd_X = pd.DataFrame(np_x, columns=['number'])
pd_y = pd.Series(10 * pd_X['number'] + 3 * random.random())

pd_X.shape, pd_y.shape
```

2) 모델 생성

- From 모델 import 모델명
- 모델변수 이름 = 모델명

(1) sklearn.linear_model

- LinearRegression : 가장 기본적인 선형 회귀 모델로, 입력 변수와 출력 변수 간의 선형 관계를 학습
- Lasso : 선형 회귀에 L1 정규화를 추가한 모델입니다. 이 모델은 가중치를 0으로 만들어 특성 선택에 활용
- SGDRegressor : 확률적 경사 하강법을 사용하여 선형 회귀 모델을 훈련시키는 모델입니다. 큰 규모의 데이터셋에 적합하며, 반복적으로 샘플을 사용하여 모델을 업데이트

(2) LinearRegression (선형 회귀)

$$y = \underbrace{\beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots}_{\text{Independent}} + \underbrace{\beta_n}_{\text{Intercept}}$$

- Scikit-Learn에서 제공하는 선형 회귀(Linear Regression) 모델을 구현한 클래스
- 연속적인 타킷을 예측하는 알고리즘
- 하이퍼파라미터
  - fit_intercept (기본값: True): 모델이 절편(intercept)을 학습할지 여부를 결정
  - copy_X (기본값: True): 입력 변수(X)의 복사 여부를 결정
  - n_jobs (기본값: None): 모델 훈련에 사용되는 작업 수를 결정
  - normalize (기본값: False): 입력 변수(X)가 정규화(normalize)되어야 하는지 여부를 결정

In [ ]:
```
from sklearn.linear_model import LinearRegression

#선형회귀
np_regr = LinearRegression(fit_intercept = True)

pd_regr = LinearRegression(fit_intercept = True)
```

3) 모델 학습

- fit(X, y) 로 학습데이터 모델에 학습

In [ ]:
```
# numpy-array regression 모델 훈련
np_regr.fit(np_X,np_y)

print(np_regr.coef_ , np_regr.intercept_)

# oandas-DataFrame 모델 훈련
pd_regr.fit(pd_X,pd_y)

print(pd_regr.coef_ , pd_regr.intercept_)
```

4) 모델 예측

- fit(테스트데이터)를 통해 y값 예측

In [ ]:
```
# Test 데이터 생성


test_x = np.random.uniform(5,10, size = 100)
#test_x = np.arange(11,21)

test_X = test_x.reshape(-1,1)
test_y = 10 * test_x + 3
```

In [ ]:
```
# numpy-array

np_pred_y = np_regr.predict(test_X)

print(np_pred_y)

# pandas DataFrame

pd_pred_y = pd_regr.predict(test_X)

print(pd_pred_y)
```

In [ ]:
```
# 그래프로 결과 확인
import matplotlib.pyplot as plt

plt.figure(1)
plt.title('numpy grape')
plt.scatter(np_X, np_y, c='blue', s = 1)
plt.scatter(test_X, np_pred_y, c='red', s = 1)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()

plt.figure(2)
plt.title('pandas grape')
plt.scatter(pd_X, pd_y, c='blue', s = 1)
plt.scatter(test_X, pd_pred_y, c='red', s = 1)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()

# 그래프 출력
plt.show()
```

5) 모델 평가

- 평균 제곱근 오차(RMSE, Root Mean Squared Error) : 평균 제곱 오차를 계산하여 회귀 모델의 예측 정확도를 평가

$$RMSE = \sqrt{\frac{1}{W}\sum_{i=1}^{N} w_i u_i^2}$$

W = 모집단의 총 가중치

N = 관찰 값 수

wi = i번째 관찰 가중치

In [ ]:
```
from sklearn.metrics import mean_squared_error

# numpy 모델 평가
np_rmse = np.sqrt(mean_squared_error(test_y, np_pred_y))
print("ndarray RMSE : ", np_rmse)

# pandas 모델 평가
pd_rmse = np.sqrt(mean_squared_error(test_y, pd_pred_y))
print("DataFrame RMSE : ", np_rmse)
```

7. Scikit_Learn을 이용한 머신러닝 실습 - iris

1) sklearn.datasets 란?

(1) 데이터 셋이란?

- sklearn.dataset 안에는 빌트인 (built-in) 데이터 셋들이 존재
- bunch 타입으로 key-value 형식으로 구성되어 있으며, 사전(dict)형 타입과 유사한 구조

(2) 데이터 셋 종류

- load_boston: 보스톤 집값 데이터
- load_iris: 아이리스 붓꽃 데이터
- load_diabetes: 당뇨병 환자 데이터
- load_digits: 손글씨 데이터
- load_linnerud: multi-output regression 용 데이터
- load_wine: 와인 데이터
- load_breast_cancer: 위스콘신 유방암 환자 데이터

2) IRIS 데이터란?

- Sklearn에서 제공하고 있는 데이터 셋
- 150개의 분꽃 데이터
- Bunch 데이터 타입

    ⦿ data: 특성 데이터를 담고 있는 NumPy 배열

    ⦿ target: 레이블 또는 타겟 변수를 담고 있는 NumPy 배열

    ⦿ feature_names: 각 특성의 이름을 담고 있는 리스트

    ⦿ target_names: 각 레이블의 이름을 담고 있는 리스트

	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
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa

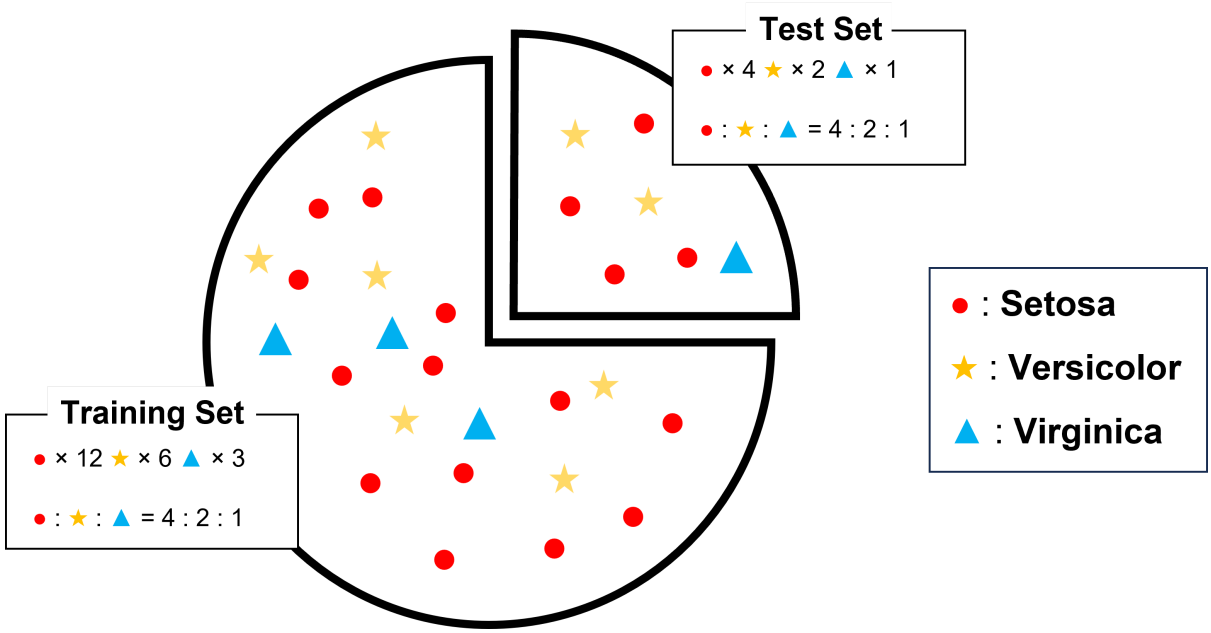
Feature Matrix

Target Vector

In [ ]:
```
from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data
y = iris.target
```

2) 훈련 데이터와 테스트 데이터 분류

- Training Set (훈련 세트) : 머신러닝 모델을 만들 목적으로 사용, 모델이 학습할 데이터
- Test Set (테스트 세트) : 훈련된 모델의 성능을 테스트하기 위해 사용할 데이터



1. 훈련세트의 데이터가 테스트 데이터보다 많아야 한다.
2. 훈련 세트와 테스트 세트가 동일한 비율의 분포를 가지고 있어야 한다.
3. 훈련 세트와 테스트 세트는 중복되는 데이터가 최대한 없어야 한다.
4. scikit-learn의 train_test_split 함수를 자주 사용 (일반적으로 훈련용-75%, 테스트용 25% 구성)

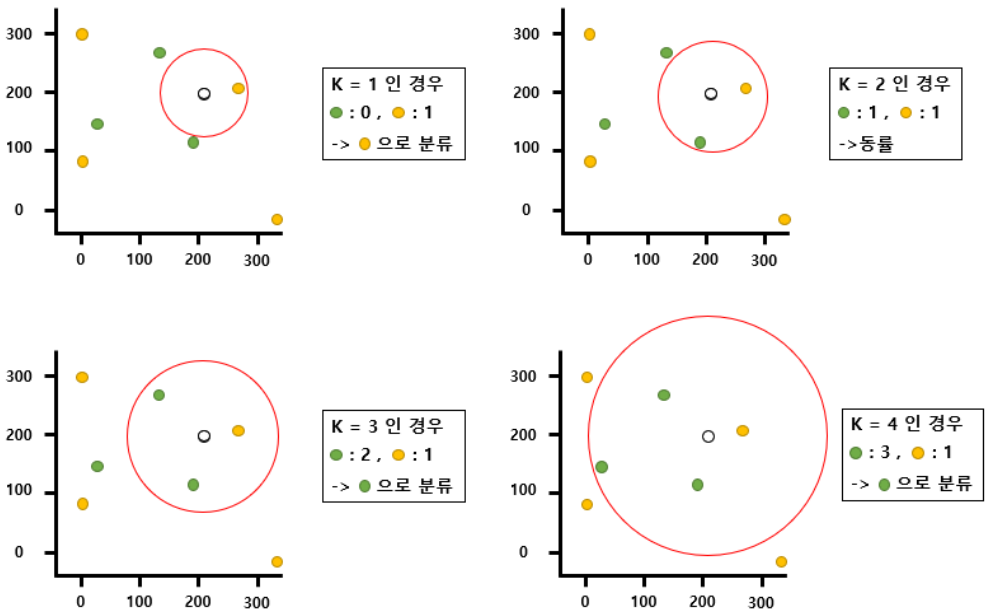
[train_test_split]

- 데이터셋을 학습 데이터와 테스트 데이터로 나누기 위해 사용되는 함수
- test_size: 테스트 데이터의 비율을 지정하는 매개변수
- train_size: 학습 데이터의 비율을 지정하는 매개변수
- random_state: 난수 발생 시드를 지정하는 매개변수, 일정한 난수 시퀀스를 생성하여 데이터를 섞는 데 사용

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

# train_data - 80%, test_data - 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=25

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

4) K-최근접 이웃(K-Nearest Neighbor)을 활용한 분류

- 분류 알고리즘 중 하나
- 주어진 데이터 포인트의 근처에 있는 k개의 가장 가까운 이웃 데이터 포인트들을 기반으로 예측
- 유사한 특성을 가진 데이터는 유사한 범주에 속하는 경향이 있다는 가정하에 사용
- n_neighbors : 근접한 이웃의 수



In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# 분류알고리즘
knn = KNeighborsClassifier(n_neighbors = 1)

# 학습 데이터로 모델 구축
knn.fit(X_train, y_train)

# 테스트 데이터로 예측
y_pred = knn.predict(X_test)

y_pred
```

In [ ]:
```
# 정확도 측정
# 타겟 데이터와 예상값 비교
# 타켓 데이터와 테스트 데이터가 같이 때문에 정확도 100%
np.mean(y_test == y_pred)
```

In [ ]:
```
# score 주어진 테스트데이터셋에 대해 정확도 계산하는 메서드
knn.score(X_test, y_test)
```

In [ ]:

```
# iris 결과 비교

result = pd.DataFrame(X_test, columns = iris.feature_names)
result['species'] = y_test
result['pred species'] = y_pred

result
```