머신러닝의 이해

1 Scikit-Learn API를 소개하고 사용법을 확인한다. 3

2 회귀, 분류와 같은 머신러닝 모델을 직접 생성해보고 절차를 살펴본다. 16

3 머신러닝 데이터를 훈련용과 테스트용으로 나누는 목적과 방법을 이해한다. 23

Scikit-Learn 라이브러리

- 머신러닝 알고리즘을 구현한 오픈소스 라이브러리 중
  가장 유명한 라이브러리 중 하나
- 일관되고 간결한 API가 강점이며, 문서화가 잘되어 있음
- 알고리즘은 파이썬 클래스로 구현되고, 데이터 셋은 Numpy 배열,
  Pandas DataFrame, SciPy 희소행렬을 사용할 수 있음

Scikit-Learn의 데이터 표현 방식

특징 행렬 (Feature Matrix)

- 표본(sample)은 데이터셋이 설명하는 개별 객체를 나타냄
- 특징(feature)은 각 표본을 연속적인 수치값, 부울값, 이산값으로 표현하는 개별 관측치를 의미
- 표본 ➡ 행렬의 행
- 행의 개수 ➡ n_samples
- 특징(feature) ➡ 행렬의 열
- 열의 개수 ➡ n_features
- 관례적으로 특징행렬은 변수 X에 저장
- [n_samples, n_features] 형태의 2차원 배열 구조를 사용
  (주로 Numpy 배열, Pandas DataFrame, SciPy 희소행렬을 사용)

Scikit-Learn의 데이터 표현 방식

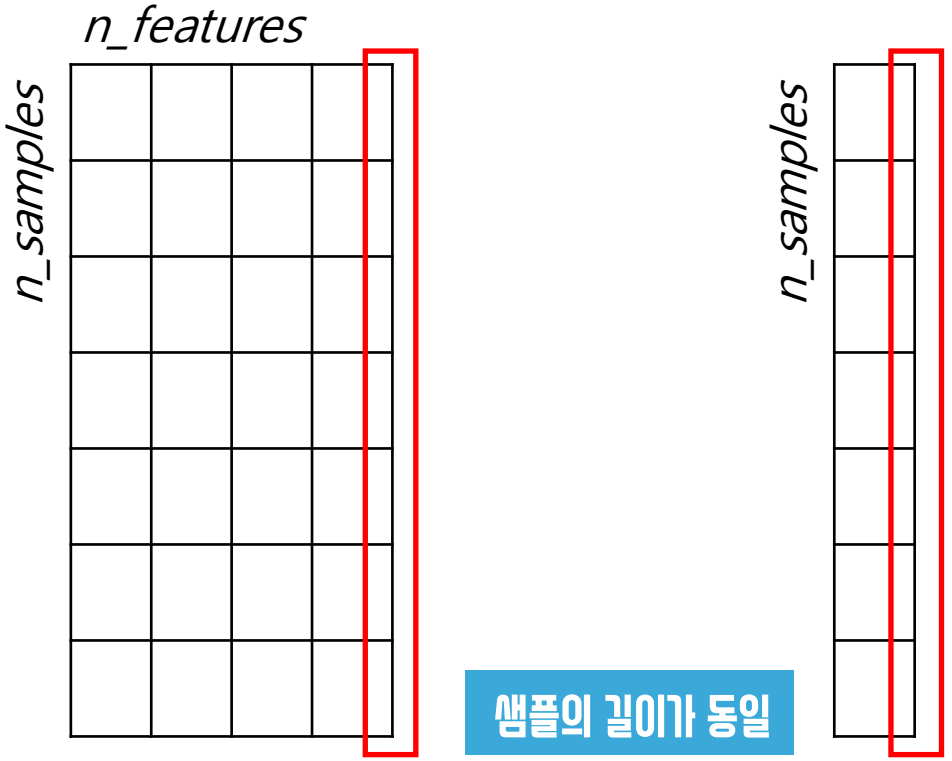
대상 벡터 (Target Vector)

- 연속적인 수치값, 이산 클래스/레이블을 가짐
- 길이 ➡ n_samples
- 관례적으로 대상벡터는 변수 y에 저장
- 1차원 배열 구조를 사용 (주로 Numpy 배열, Pandas Series를 사용)
- 특징 행렬로부터 예측하고자 하는 값의 벡터
- 종속 변수, 출력 변수, 결과 변수, 반응 변수 라고도 함

Scikit-Learn의 데이터 표현 방식

특징행렬과 대상벡터의 데이터 레이아웃

- 특징행렬 (X)
- 대상벡터 (y)



Scikit-Learn의 데이터 표현 방식

Numpy배열을이용한특징행렬(X),대상벡터(y)의생성

```
1  import numpy as np
2
3  rs = np.random.RandomState(10)
4  x = 10 * rs.rand(5)
5  y = 2 * x - 1 * rs.rand(5)
6  x.shape, y.shape
```

Seed값 고정

난수 5개 생성

```
((5,), (5,))
```

```
1  X = x.reshape(-1, 1)
2  X.shape
```

```
(5, 1)
```

Scikit-Learn의 데이터 표현 방식

Pandas DataFrame을 이용한 특징 행렬(X), 대상 벡터(y)의 생성

```
1  import seaborn as sns
2  iris = sns.load_dataset("iris")
3  iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

Scikit-Learn의 데이터 표현 방식

```
1  iris.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

Feature Matrix와 Target Vector를 나누는 작업 필요

Scikit-Learn의 데이터 표현 방식

```
1  X = iris.drop("species", axis=1)
2  X.shape
```

"species" 열이 삭제된
X 데이터프레임(Feature Matrix) 생성

```
(150, 4)
```

```
1  y = iris["species"]
2  y.shape
```

"species" 열을 추출하여 y Series(Target Vector) 생성

```
(150,)
```

Scikit-Learn의 데이터 표현 방식

Bunch 객체를 이용한 특징 행렬(X), 대상 벡터(y)의 생성

```
1  from sklearn.datasets import load_iris
2
3  iris = load_iris()
4  type(iris)
```

```
sklearn.utils.Bunch
```

```
1  iris.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

Feature Matrix

Target Vector

Feature Matrix 열의 이름

Scikit-Learn의 데이터 표현 방식

```
1  iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Feature Matrix 열의 이름

Scikit-Learn의 데이터 표현 방식

```
1  iris.data[:5]
```

```
array([[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]])
```

```
1  iris.data.shape
```

```
(150, 4)
```

Scikit-Learn의 데이터 표현 방식

```
1  iris.target
```

```
array([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, 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])
```

```
1  iris.target.shape
```

```
(150,)
```

```
1  iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

특징벡터 내 속성값들

Scikit-Learn의 데이터 표현 방식

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

특징행렬

타겟벡터

Scikit-Learn Estimator API 기본 활용 절차

1. 데이터 준비
2. 모델 클래스 선택
3. 모델 인스턴스 생성과 하이퍼파라미터 선택
4. 특징 행렬과 대상 벡터 준비
5. 모델을 데이터에 적합
6. 새로운 데이터를 이용해 예측
7. 모델 평가

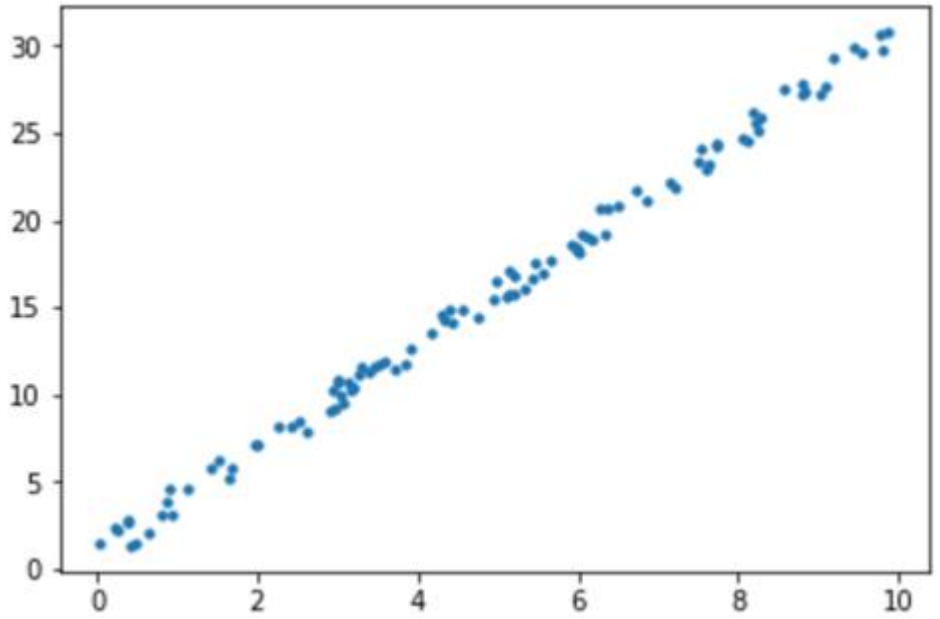
Scikit-Learn Estimator API 기본 활용 절차

1 데이터 준비

```
1  import numpy as np
2  import matplotlib.pyplot as plt
```

```
1  rs = np.random.RandomState(10)
2  x = 10 * rs.rand(100)
3  y = 3 * x + 2 * rs.rand(100)
```

```
1  plt.scatter(x, y, s=10);
```



Scikit-Learn Estimator API 기본 활용 절차

2 모델 클래스 선택

3 모델 인스턴스 생성과 하이퍼파라미터 선택

입력데이터(x), 출력데이터(y)가 모두 연속형 수치 데이터이므로 그에 맞는 분석 모델을 선택

```
1  from sklearn.linear_model import LinearRegression
2  regr = LinearRegression()
```

선형회귀 객체(인스턴스) 생성 - 디폴트

```
1  from sklearn.linear_model import LinearRegression
2  regr = LinearRegression(fit_intercept=True)
```

선형회귀 객체(인스턴스) 생성 -
(fit_intercept=True라는 하이퍼파라미터를 제공)

Scikit-Learn Estimator API 기본 활용 절차

4 특징 행렬과 대상 벡터 준비

```
1  X = x.reshape(-1, 1)
2  X.shape, y.shape
```

```
((100, 1), (100,))
```

Scikit-Learn Estimator API 기본 활용 절차

5 모델을 데이터에 적합

```
1 regr.fit(X, y)
```

X, y에 맞는 선형회귀 모델을 적합(모델 생성)

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
1 regr.coef_
```

모델의 기울기

```
array([2.9855087])
```

```
1 regr.intercept_
```

모델의 y 절편

```
0.9878534341975644
```

Scikit-Learn Estimator API 기본 활용 절차

6 새로운 데이터를 이용해 예측

```
1  x_new = np.linspace(-1, 11, num=100)
```

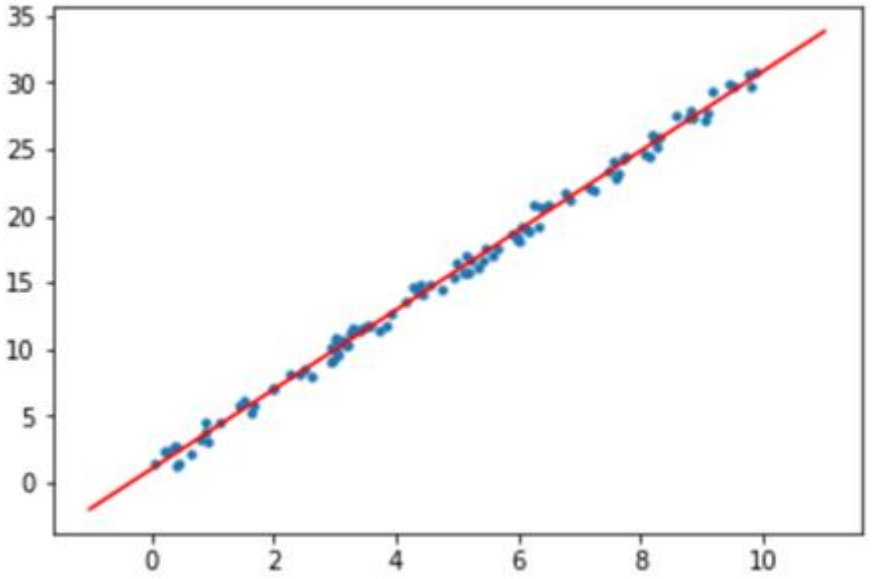
```
1  X_new = x_new.reshape(-1, 1)
2  X_new.shape
```

```
(100, 1)
```

새로 입력된 X_new에 대한 모델 예측값(y_pred) 생성

```
1  y_pred = regr.predict(X_new)
```

```
1  plt.plot(x_new, y_pred, c="red")
2  plt.scatter(x, y, s=10);
```



Scikit-Learn Estimator API 기본 활용 절차

7 모델 평가

```
1  from sklearn.metrics import mean_squared_error
2
3  rmse = np.sqrt(mean_squared_error(y, y_pred))
4  rmse
```

RMSE(평균제곱오차) 구하기

y : 실제값
y_pred : 예측값

```
13.708237122486333
```

훈련 데이터와 테스트 데이터

iris데이터

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	6.5	2.8	4.6	1.5	1
1	5.7	2.5	5.0	2.0	2
2	7.7	3.0	6.1	2.3	2
3	5.0	3.6	1.4	0.2	0
4	6.4	3.2	5.3	2.3	2

'setosa', 'versicolor', 'virginica’

훈련 데이터와 테스트 데이터

정확도가정말1.0인가?

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

```
1  from sklearn.neighbors import KNeighborsClassifier
2  knn = KNeighborsClassifier(n_neighbors=1)
```

분류 알고리즘 라이브러리

```
1  knn.fit(X, y)
```

훈련데이터 X와 y로 모델 적합

```
1  y_pred = knn.predict(X)
```

훈련데이터 X로 y_pred값 예측

```
1  np.mean(y == y_pred)
```

실제값 y와 예측값 y_pred값을 비교하여 정확도 측정

```
1.0
```

훈련 데이터와 테스트 데이터

훈련 데이터와 테스트 데이터의 분리

- 머신러닝 모델을 만들 때 사용한 데이터는 모델의 성능측정용으로 사용하지 않음
  ➡ 일반화 문제

- 훈련 데이터
  : 머신러닝 모델을 만들 목적으로 사용

- 테스트 데이터
  : 머신러닝 모델이 잘 작동하는지를 측정할 목적으로 사용

- scikit-learn의 train_test_split 함수를 주로 사용
  (기본적으로 훈련용 75%, 테스트용 25% 구성)

훈련 데이터와 테스트 데이터

```
1  from sklearn.model_selection import train_test_split
2
3  X_train, X_test, y_train, y_test = train_test_split(
4      X, y, test_size=0.2, random_state=25)
```

데이터 재현을 위한 seed값

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

```
((120, 4), (120,), (30, 4), (30,))
```

```
1  from sklearn.neighbors import KNeighborsClassifier
2  knn = KNeighborsClassifier(n_neighbors=1)
```

```
1  knn.fit(X_train, y_train)
```

훈련 데이터로 모델 적합

훈련 데이터와 테스트 데이터

테스트 데이터를 이용한 모델의 성능 측정

```
1  y_pred = knn.predict(X_test)
2  y_pred
```

테스트 데이터로 예측 수행

```
array([0, 2, 2, 1, 2, 1, 2, 0, 1, 1, 0, 0, 0, 1, 0, 1, 2, 2, 1, 1, 1, 1,
       1, 0, 0, 2, 1, 2, 2, 0])
```

```
1  np.mean(y_test == y_pred)
```

테스트 데이터의 실제값과 예측값 비교를 통해 정확도 계산

```
0.9
```

훈련 데이터와 테스트 데이터

```
1  knn.score(X_test, y_test)
```

모델 정확도 계산

```
0.9
```

```
1  from sklearn.metrics import accuracy_score
2
3  accuracy_score(y_test, y_pred)
```

모델 정확도 계산

```
0.9
```

하이퍼파라미터의 선택

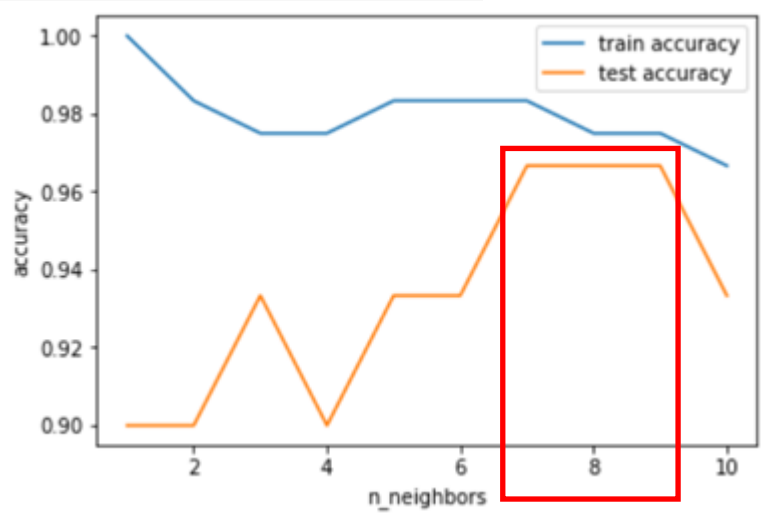
하이퍼파라미터의 선택

```
1  train_accuracy = []
2  test_accuracy = []
3  neighbors = range(1, 11)
4  for n in neighbors:
5      knn = KNeighborsClassifier(n_neighbors=n)
6      knn.fit(X_train, y_train)
7      train_accuracy.append(knn.score(X_train, y_train))
8      test_accuracy.append(knn.score(X_test, y_test))
```

Neighbors를 1~11까지 번갈아가며 설정해서 모델 적합 및 정확도 계산

하이퍼파라미터의 선택

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3
4  plt.plot(neighbors, train_accuracy, label="train accuracy")
5  plt.plot(neighbors, test_accuracy, label="test accuracy")
6  plt.xlabel("n_neighbors")
7  plt.ylabel("accuracy")
8  plt.legend();
```



파라미터를 변경해가면서 가장 높은 정확도를
보이는 모델 생성을 위한 하이퍼파라미터 선택 가능

주요 정리

1. scikit-learn은 데이터를 담고 있는 2차원 구조의 특징 행렬(X)와 레이블을 담고 있는 1차원 구조의 대상 벡터(y)를 사용하도록 설계되어 있습니다.

2. scikit-learn의 Estimator는 공통 인터페이스로 fit, predict, score 메서드를 제공합니다.

3. 회귀 문제는 선형 알고리즘이 구현되어 있는 LinearRegressor를 사용할 수 있고, 분류 문제는 분류 알고리즘이 구현되어 있는 KNeighborsClassifier를 사용할 수 있습니다.

4. scikit-learn의 train_test_split 함수를 이용해 훈련 데이터와 테스트 데이터를 나누고, 모델의 성능은 테스트 데이터를 이용해 측정합니다.