사이킷런(scikit-learn) 시작

scikit-learn 특징

- 다양한 머신러닝 알고리즘을 구현한 파이썬 라이브러리
- 심플하고 일관성 있는 API, 유용한 온라인 문서, 풍부한 예제
- 머신러닝을 위한 쉽고 효율적인 개발 라이브러리 제공
- 다양한 머신러닝 관련 알고리즘과 개발을 위한 프레임워크와 API 제공
- 많은 사람들이 사용하며 다양한 환경에서 검증된 라이브러리

scikit-learn 주요 모듈

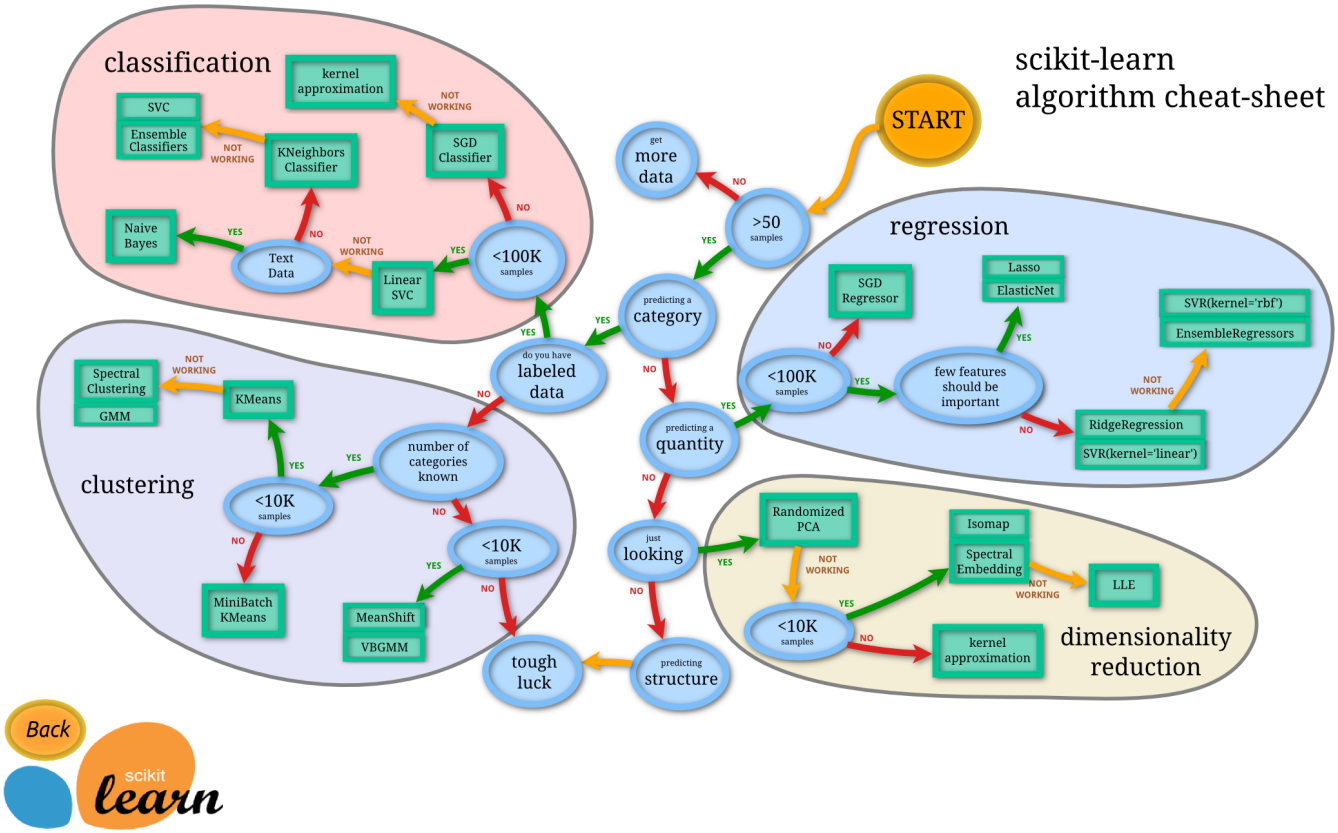
모듈	설명
`sklearn.datasets`	내장된 예제 데이터 세트
`sklearn.preprocessing`	다양한 데이터 전처리 기능 제공 (변환, 정규화, 스케일링 등)
`sklearn.feature_selection`	특징(feature)를 선택할 수 있는 기능 제공
`sklearn.feature_extraction`	특징(feature) 추출에 사용
`sklearn.decomposition`	차원 축소 관련 알고리즘 지원 (PCA, NMF, Truncated SVD 등)
`sklearn.model_selection`	교차 검증을 위해 데이터를 학습/테스트용으로 분리, 최적 파라미터를 추출하는 API 제공 (GridSearch
`sklearn.metrics`	분류, 회귀, 클러스터링, Pairwise에 대한 다양한 성능 측정 방법 제공 (Accuracy, Precision, Recall, RO
`sklearn.pipeline`	특징 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 묶어서 실행할 수 있는 유틸리티 제공
`sklearn.linear_model`	선형 회귀, 릿지(Ridge), 라쏘(Lasso), 로지스틱 회귀 등 회귀 관련 알고리즘과 SGD(Stochastic Gradie
`sklearn.svm`	서포트 벡터 머신 알고리즘 제공
`sklearn.neighbors`	최근접 이웃 알고리즘 제공 (k-NN 등)
`sklearn.naive_bayes`	나이브 베이즈 알고리즘 제공 (가우시안 NB, 다항 분포 NB 등)
`sklearn.tree`	의사 결정 트리 알고리즘 제공
`sklearn.ensemble`	앙상블 알고리즘 제공 (Random Forest, AdaBoost, GradientBoost 등)
`sklearn.cluster`	비지도 클러스터링 알고리즘 제공 (k-Means, 계층형 클러스터링, DBSCAN 등)

`estimator` API

- 일관성: 모든 객체는 일관된 문서를 갖춘 제한된 메서드 집합에서 비롯된 공통 인터페이스 공유
- 검사(inspection): 모든 지정된 파라미터 값은 공개 속성으로 노출
- 제한된 객체 계층 구조
    - 알고리즘만 파이썬 클래스에 의해 표현
    - 데이터 세트는 표준 포맷(NumPy 배열, Pandas DataFrame, Scipy 희소 행렬)으로 표현
    - 매개변수명은 표준 파이썬 문자열 사용
- 구성: 많은 머신러닝 작업은 기본 알고리즘의 시퀀스로 나타낼 수 있으며, Scikit-Learn은 가능한 곳이라면 어디서든 이 방식을 사용
- 합리적인 기본값: 모델이 사용자 지정 파라미터를 필요로 할 때 라이브러리가 적절한 기본값을 정의

API 사용 방법

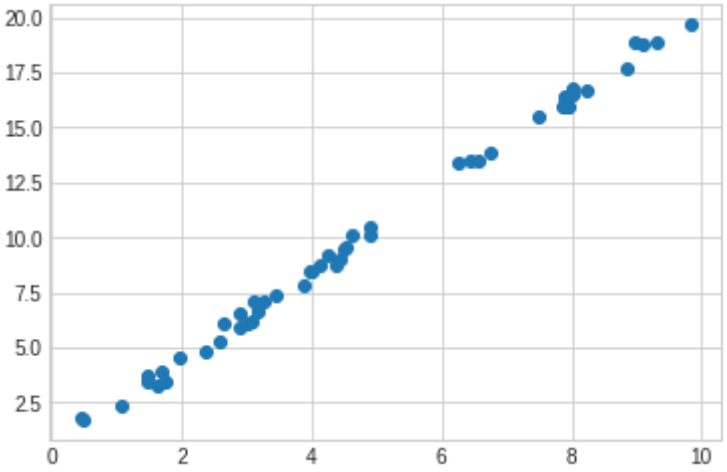
1. Scikit-Learn으로부터 적절한 `estimator` 클래스를 임포트해서 모델의 클래스 선택
2. 클래스를 원하는 값으로 인스턴스화해서 모델의 하이퍼파라미터 선택
3. 데이터를 특징 배열과 대상 벡터로 배치
4. 모델 인스턴스의 `fit()` 메서드를 호출해 모델을 데이터에 적합
5. 모델을 새 데이터에 대해서 적용
    - 지도 학습: 대체로 `predict()` 메서드를 사용해 알려지지 않은 데이터에 대한 레이블 예측
    - 비지도 학습: 대체로 `transform()`이나 `predict()` 메서드를 사용해 데이터의 속성을 변환하거나 추론



API 사용 예제

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use(['seaborn-whitegrid'])
```

```
x = 10 * np.random.rand(50)
y = 2 * x + np.random.rand(50)
plt.scatter(x, y);
```




```
# 1. 적절한 estimator 클래스를 임포트해서 모델의 클래스 선택
from sklearn.linear_model import LinearRegression
```

```
# 2. 클래스를 원하는 값으로 인스턴스화해서 모델의 하이퍼파라미터 선택
model = LinearRegression(fit_intercept=True)
model
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
# 3. 데이터를 특징 배열과 대상 벡터로 배치
X = x[:, np.newaxis]
X
```

```
array([[4.61034553],
       [6.44487273],
       [1.46984046],
       [0.49352057],
       [3.16977189],
       [6.74337995],
       [8.02147322],
       [3.25829698],
       [4.52430905],
       [8.8490915 ],
       [4.43080995],
       [7.93061092],
       [4.00616424],
       [4.23668778],
       [1.74460893],
       [2.89708882],
       [4.88903762],
       [4.36674552],
       [8.01017019],
       [3.96960264],
       [9.1080991 ],
       [3.07549833],
       [4.50222874],
       [3.00721738],
       [2.64173705],
       [3.45755969],
       [8.21759261],
       [2.35602665],
       [9.83287066],
       [7.4790058 ],
       [2.57170237],
       [7.9384558 ],
       [8.97809519],
       [3.0941904 ],
       [7.88537167],
       [1.07334597],
       [1.97640888],
       [7.87369839],
       [9.29955918],
       [0.4418047 ],
       [1.63388869],
       [6.25188701],
       [2.89970748],
       [3.88544976],
       [4.1232166 ],
       [6.57717785],
       [7.87860913],
       [1.47078438],
```

```
[4.90638523],
[1.68696803]])
```

```
# 4. 모델 인스턴스의 fit() 메서드를 호출해 모델을 데이터에 적합
model.fit(X, y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
model.coef_
```

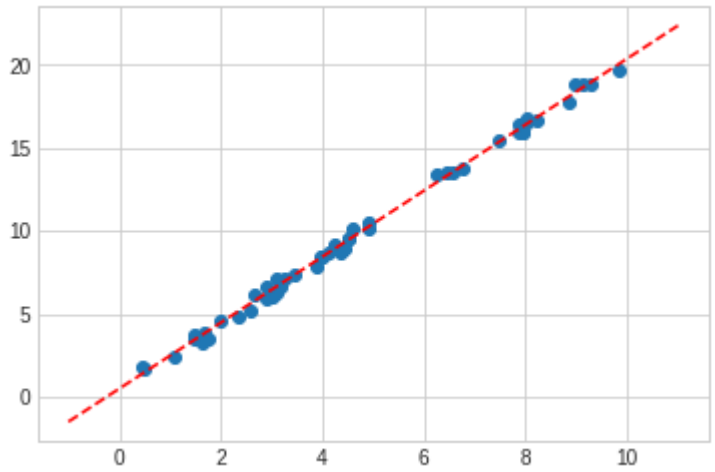
```
array([1.98875012])
```

```
model.intercept_
```

```
0.4828069631628722
```

```
# 5. 모델을 새 데이터에 대해서 적용
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit, '--r');
```



예제 데이터 세트

분류 또는 회귀용 데이터 세트

API	설명
datasets.load_boston()	미국 보스턴의 집에 대한 특징과 가격 데이터 (회귀용)
datasets.load_iris()	붓꽃에 대한 특징을 가진 데이터 (분류용)
datasets.load_diabetes()	당뇨 데이터 (회귀용)
datasets.load_wine()	와인에 대한 특징을 가진 데이터 (분류용)

API	설명
datasets.load_breast_cancer()	위스콘신 유방암 특징들과 악성/음성 레이블 데이터 (분류용)

온라인 데이터 세트

- 데이터 크기가 커서 온라인에서 데이터를 다운로드 한 후에 불러오는 예제 데이터 세트

API	설명
fetch_california_housing()	캘리포니아 주택 가격 데이터
fetch_covtype()	회귀 분석용 토지 조사 데이터
fetch_20newsgroups()	뉴스 그룹 텍스트 데이터
fetch_olivetti_faces()	얼굴 이미지 데이터
fetch_lfw_people()	얼굴 이미지 데이터
fetch_lfw_paris()	얼굴 이미지 데이터
fetch_rcv1()	로이터 뉴스 말뭉치 데이터
fetch_mldata()	ML 웹사이트에서 다운로드

분류와 클러스터링을 위한 표본 데이터 생성

API	설명
datasets.make_classifications()	분류를 위한 데이터 세트 생성. 높은 상관도, 불필요한 속성 등의 노이즈를 고려한 데이터를 무작
datasets.make_blobs()	클러스터링을 위한 데이터 세트 생성. 군집 지정 개수에 따라 여러 가지 클러스터링을 위한 데이

예제 데이터 세트 구조

- 일반적으로 딕셔너리 형태로 구성
- data: 특징 데이터 세트
- target: 분류용은 레이블 값, 회귀용은 숫자 결과값 데이터
- target_names: 개별 레이블의 이름 (분류용)
- feature_names: 특징 이름
- DESCR: 데이터 세트에 대한 설명과 각 특징 설명

```
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
print(diabetes.keys())
```

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

```
print(diabetes.data)
```

```
[[ 0.03807591  0.05068012  0.06169621 ... -0.00259226  0.01990842
  -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 ... -0.03949338 -0.06832974
  -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 ... -0.00259226  0.00286377
  -0.02593034]
```

```
 ...
 [ 0.04170844  0.05068012 -0.01590626 ... -0.01107952 -0.04687948
   0.01549073]
 [-0.04547248 -0.04464164  0.03906215 ...  0.02655962  0.04452837
  -0.02593034]
 [-0.04547248 -0.04464164 -0.0730303  ... -0.03949338 -0.00421986
   0.00306441]]
```

```
print(diabetes.target)
```

```
[151.  75. 141. 206. 135.  97. 138.  63. 110. 310. 101.  69. 179. 185.
 118. 171. 166. 144.  97. 168.  68.  49.  68. 245. 184. 202. 137.  85.
 131. 283. 129.  59. 341.  87.  65. 102. 265. 276. 252.  90. 100.  55.
  61.  92. 259.  53. 190. 142.  75. 142. 155. 225.  59. 104. 182. 128.
  52.  37. 170. 170.  61. 144.  52. 128.  71. 163. 150.  97. 160. 178.
  48. 270. 202. 111.  85.  42. 170. 200. 252. 113. 143.  51.  52. 210.
  65. 141.  55. 134.  42. 111.  98. 164.  48.  96.  90. 162. 150. 279.
  92.  83. 128. 102. 302. 198.  95.  53. 134. 144. 232.  81. 104.  59.
 246. 297. 258. 229. 275. 281. 179. 200. 200. 173. 180.  84. 121. 161.
  99. 109. 115. 268. 274. 158. 107.  83. 103. 272.  85. 280. 336. 281.
 118. 317. 235.  60. 174. 259. 178. 128.  96. 126. 288.  88. 292.  71.
 197. 186.  25.  84.  96. 195.  53. 217. 172. 131. 214.  59.  70. 220.
 268. 152.  47.  74. 295. 101. 151. 127. 237. 225.  81. 151. 107.  64.
 138. 185. 265. 101. 137. 143. 141.  79. 292. 178.  91. 116.  86. 122.
  72. 129. 142.  90. 158.  39. 196. 222. 277.  99. 196. 202. 155.  77.
 191.  70.  73.  49.  65. 263. 248. 296. 214. 185.  78.  93. 252. 150.
  77. 208.  77. 108. 160.  53. 220. 154. 259.  90. 246. 124.  67.  72.
 257. 262. 275. 177.  71.  47. 187. 125.  78.  51. 258. 215. 303. 243.
  91. 150. 310. 153. 346.  63.  89.  50.  39. 103. 308. 116. 145.  74.
  45. 115. 264.  87. 202. 127. 182. 241.  66.  94. 283.  64. 102. 200.
 265.  94. 230. 181. 156. 233.  60. 219.  80.  68. 332. 248.  84. 200.
  55.  85.  89.  31. 129.  83. 275.  65. 198. 236. 253. 124.  44. 172.
 114. 142. 109. 180. 144. 163. 147.  97. 220. 190. 109. 191. 122. 230.
 242. 248. 249. 192. 131. 237.  78. 135. 244. 199. 270. 164.  72.  96.
 306.  91. 214.  95. 216. 263. 178. 113. 200. 139. 139.  88. 148.  88.
 243.  71.  77. 109. 272.  60.  54. 221.  90. 311. 281. 182. 321.  58.
 262. 206. 233. 242. 123. 167.  63. 197.  71. 168. 140. 217. 121. 235.
 245.  40.  52. 104. 132.  88.  69. 219.  72. 201. 110.  51. 277.  63.
 118.  69. 273. 258.  43. 198. 242. 232. 175.  93. 168. 275. 293. 281.
  72. 140. 189. 181. 209. 136. 261. 113. 131. 174. 257.  55.  84.  42.
 146. 212. 233.  91. 111. 152. 120.  67. 310.  94. 183.  66. 173.  72.
  49.  64.  48. 178. 104. 132. 220.  57.]
```

```
print(diabetes.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442
```

```
        :Number of Attributes: First 10 columns are numeric predictive values

        :Target: Column 11 is a quantitative measure of disease progression one year after baseline

        :Attribute Information:
            - Age
            - Sex
            - Body mass index
            - Average blood pressure
            - S1
            - S2
            - S3
            - S4
            - S5
            - S6

        Note: Each of these 10 feature variables have been mean centered and scaled by the standard d

        Source URL:
        https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

        For more information see:
        Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regres
        (https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

```
print(diabetes.feature_names)
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
print(diabetes.data_filename)
print(diabetes.target_filename)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/datasets/data/diabetes_data.csv.gz
/usr/local/lib/python3.6/dist-packages/sklearn/datasets/data/diabetes_target.csv.gz
```

`model_selection` 모듈

- 학습용 데이터와 테스트 데이터로 분리
- 교차 검증 분할 및 평가
- Estimator의 하이퍼 파라미터 튜닝을 위한 다양한 함수와 클래스 제공

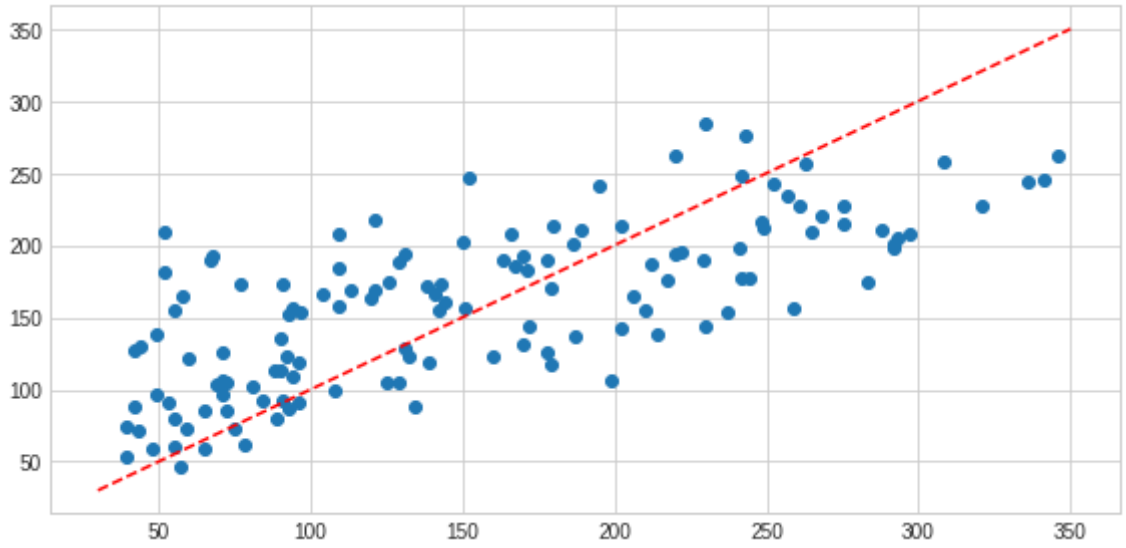
`train_test_split()`: 학습/테스트 데이터 세트 분리

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.3)

model = LinearRegression()
```

```
model.fit(X_train, y_train)

print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.5066329772801276
평가 데이터 점수: 0.518254941974408
```

```
import matplotlib.pyplot as plt

predicted = model.predict(X_test)
expected = y_test
plt.figure(figsize=(8, 4))
plt.scatter(expected, predicted)
plt.plot([30, 350], [30, 350], '--r')
plt.tight_layout()
```



`cross_val_score()`: 교차 검증

```
from sklearn.model_selection import cross_val_score, cross_validate

scores = cross_val_score(model, diabetes.data, diabetes.target, cv=5)

print("교차 검증 정확도: {}".format(scores))
print("교차 검증 정확도: {} +/- {}".format(np.mean(scores), np.std(scores)))
```

```
교차 검증 정확도: [0.42955643 0.52259828 0.4826784  0.42650827 0.55024923]
교차 검증 정확도: 0.48231812211149394 +/- 0.049266197765632194
```

`GridSearchCV`: 교차 검증과 최적 하이퍼 파라미터 찾기

- 훈련 단계에서 학습한 파라미터에 영향을 받아서 최상의 파라미터를 찾는 일은 항상 어려운 문제
- 다양한 모델의 훈련 과정을 자동화하고, 교차 검사를 사용해 최적 값을 제공하는 도구 필요

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
import pandas as pd

alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
param_grid = dict(alpha=alpha)

gs = GridSearchCV(estimator=Ridge(), param_grid=param_grid, cv=10)
result = gs.fit(diabetes.data, diabetes.target)

print("최적 점수: {}".format(result.best_score_))
print("최적 파라미터: {}".format(result.best_params_))
print(gs.best_estimator_)
pd.DataFrame(result.cv_results_)
```

```
최적 점수: 0.4633240541517593
최적 파라미터: {'alpha': 0.1}
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	par
0	0.001146	0.000855	0.000783	0.000387	0.001	{'alp 0.0
1	0.000549	0.000074	0.000504	0.000031	0.01	{'alp 0.
2	0.000501	0.000023	0.000488	0.000017	0.1	{'alp (
3	0.000535	0.000031	0.000537	0.000087	1	{'alp
4	0.000564	0.000049	0.000529	0.000071	10	{'alp
5	0.000538	0.000033	0.000502	0.000033	100	{'alp 1
6	0.000581	0.000072	0.000537	0.000091	1000	{'alp 10

- `multiprocessing`을 이용한 `GridSearchCV`

```
import multiprocessing
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()

param_grid = [ { 'penalty': ['l1', 'l2'],
                 'C': [1.5, 2.0, 2.5, 3.0, 3.5] } ]

gs = GridSearchCV(estimator=LogisticRegression(), param_grid=param_grid,
                  scoring='accuracy', cv=10, n_jobs=multiprocessing.cpu_count())
```

```
result = gs.fit(iris.data, iris.target)

print("최적 점수: {}".format(result.best_score_))
print("최적 파라미터: {}".format(result.best_params_))
print(gs.best_estimator_)
pd.DataFrame(result.cv_results_)
```

```
최적 점수: 0.9800000000000001
최적 파라미터: {'C': 2.5, 'penalty': 'l2'}
LogisticRegression(C=2.5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
```

`preprocessing` 데이터 전처리 모듈

- 데이터의 특징 스케일링(feature scaling)을 위한 방법으로 표준화(Standardization)와 정규화(Normalization) 사용
- 표준화 방법

$$x_i^{'} = \frac{x_i - mean(x)}{stdev(x)}$$

- 정규화 방법

$$x_i^{'} = \frac{x_i - min(x)}{max(x) - min(x)}$$

- scikit-learn에서는 개별 벡터 크기를 맞추는 형태로 정규화

`StandardScaler` : 표준화 클래스

```
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_df)
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_scaled.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	-1.690315e-15	-1.842970e-15	-1.698641e-15	-1.409243e-15
std	1.003350e+00	1.003350e+00	1.003350e+00	1.003350e+00
min	-1.870024e+00	-2.433947e+00	-1.567576e+00	-1.447076e+00
25%	-9.006812e-01	-5.923730e-01	-1.226552e+00	-1.183812e+00
50%	-5.250608e-02	-1.319795e-01	3.364776e-01	1.325097e-01
75%	6.745011e-01	5.586108e-01	7.627583e-01	7.906707e-01

```
X_train, X_test, y_train, y_test = train_test_split(iris_df_scaled, iris.target, test_size=0.3)

model = LogisticRegression()
model.fit(X_train, y_train)

print("훈련 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
훈련 데이터 점수: 0.9809523809523809
평가 데이터 점수: 0.9111111111111111
```

`MinMaxScaler`: 정규화 클래스

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
iris_scaled = scaler.fit_transform(iris_df)
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_scaled.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	0.428704	0.440556	0.467458	0.458056
std	0.230018	0.181611	0.299203	0.317599
min	0.000000	0.000000	0.000000	0.000000
25%	0.222222	0.333333	0.101695	0.083333
50%	0.416667	0.416667	0.567797	0.500000
75%	0.583333	0.541667	0.694915	0.708333
max	1.000000	1.000000	1.000000	1.000000

```
X_train, X_test, y_train, y_test = train_test_split(iris_df_scaled, iris.target, test_size=0.3)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)

print("훈련 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
훈련 데이터 점수: 0.9333333333333333
평가 데이터 점수: 0.9555555555555556
```

성능 평가 지표

정확도(Accuracy)

- 정확도는 전체 예측 데이터 건수 중 예측 결과가 동일한 데이터 건수로 계산
- scikit-learn에서는 `accuracy_score` 함수를 제공

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_redundant=0, n_clusters_per_class=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = LogisticRegression()
model.fit(X_train, y_train)

print("훈련 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))

predict = model.predict(X_test)
print("정확도: {}".format(accuracy_score(y_test, predict)))
```

```
훈련 데이터 점수: 0.9485714285714286
평가 데이터 점수: 0.9533333333333334
정확도: 0.9533333333333334
```

오차 행렬(Confusion Matrix)

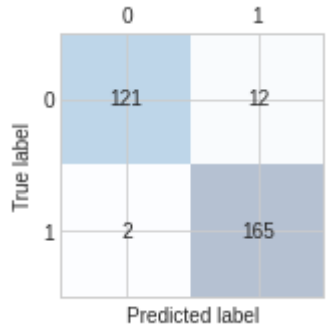
- True Negative: 예측값을 Negative 값 0으로 예측했고, 실제 값도 Negative 값 0
- False Positive: 예측값을 Positive 값 1로 예측했는데, 실제 값은 Negative 값 0
- False Negative: 예측값을 Negative 값 0으로 예측했는데, 실제 값은 Positive 값 1
- True Positive: 예측값을 Positive 값 1로 예측했고, 실제 값도 Positive 값 1

```
from sklearn.metrics import confusion_matrix

confmat = confusion_matrix(y_true=y_test, y_pred=predict)
print(confmat)
```

```
print(confmat)
```

```
[[121  12]
 [  2 165]]
```

```
fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
  for j in range(confmat.shape[1]):
    ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.tight_layout()
plt.show()
```



정밀도(Precision)와 재현율(Recall)

- 정밀도 = TP / (FP + TP)
- 재현율 = TP / (FN + TP)
- 정확도 = (TN + TP) / (TN + FP + FN + TP)
- 오류율 = (FN + FP) / (TN + FP + FN + TP)

```
from sklearn.metrics import precision_score, recall_score

precision = precision_score(y_test, predict)
recall = recall_score(y_test, predict)

print("정밀도: {}".format(precision))
print("재현율: {}".format(recall))
```

```
정밀도: 0.9322033898305084
재현율: 0.9880239520958084
```

F1 Score(F-measure)

- 정밀도와 재현율을 결합한 지표
- 정밀도와 재현율이 어느 한쪽으로 치우치지 않을 때 높은 값을 가짐

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, predict)

print("F1 Score: {}".format(f1))
```

```
F1 Score: 0.9593023255813954
```

ROC 곡선과 AUC

- ROC 곡선은 FPR(False Positive Rate)이 변할 때 TPR(True Positive Rate)이 어떻게 변하는지 나타내는 곡선
  - TPR(True Positive Rate): TP / (FN + TP), 재현율
  - TNR(True Negative Rate): TN / (FP + TN)
  - FPR(False Positive Rate): FP / (FP + TN), 1 - TNR
- AUC(Area Under Curve) 값은 ROC 곡선 밑에 면적을 구한 값 (1이 가까울수록 좋은 값)

```
from sklearn.metrics import roc_curve

pred_proba_class1 = model.predict_proba(X_test)[:, 1]
fprs, tprs, thresholds = roc_curve(y_test, pred_proba_class1)

plt.plot(fprs, tprs, label='ROC')
plt.plot([0, 1], [0, 1], '--k', label='Random')
start, end = plt.xlim()
plt.xticks(np.round(np.arange(start, end, 0.1), 2))
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('FPR(1-Sensitivity)')
plt.ylabel('TPR(Recall)')
plt.legend();
```

```
from sklearn.metrics import roc_auc_score

roc_auc = roc_auc_score(y_test, predict)

print("ROC AUC Score: {}".format(roc_auc))
```

```
ROC AUC Score: 0.948899194093017
```

참고문헌

- scikit-learn 사이트: https://scikit-learn.org/
- Jake VanderPlas, "Python Data Science Handbook", O'Reilly
- Sebastian Raschka, Vahid Mirjalili, "Python Machine Learning", Packt
- Giuseppe Bonaccorso, "Machine Learning Algorithm", Packt
- Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems", O'Reilly