직장인을 위한 파이썬 데이터 분석

# Scikit-learn Cheat Sheet

## Scikit-learn

**Scikit-learn 은 파이썬 대표 머신러닝 라이브러리**입니다. 머신러닝에서 사용될 수 있는 전처리, 학습, 예측, 평가 단계를 모두 제공합니다.

### 라이브러리 로딩

```
from sklearn.datasets import load_iris                  데이터 로딩
from sklearn.preprocessing import StandardScaler        전처리 함수
from neighbors import KNeighborsClassifier              학습 모델
from sklearn.cross_validation import train_test_split   학습 / 테스트 데이터 분리
from sklearn.metrics import accuracy_score              평가 지표
```

### 예제 사용법

```
iris = load_iris()                                                  데이터 로딩
X, y = iris.data[:, :2], iris.target                                X, Y데이터 지정
X_train, X_test, y_train, y_test = train_test_split(X, y)           학습 / 테스트 분할
scaler = StandardScaler().fit(X_train)                              표준화
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=5)                           학습 모델 생성
knn.fit(X_train, y_train)                                           모델 학습
y_pred = knn.predict(X_test)                                        모델 예측
accuracy_score(y_test, y_pred)                                      평가
```

## 학습 / 테스트 데이터 분할

```
import numpy as np                                  데이터 로딩
X = np.random.random((10,5))                        (기본형 - numpy)
y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
X[X < 0.7] = 0
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

## 데이터 전처리

### 표준화 (Standardization)

```
from sklearn.preprocessing import StandardScaler    라이브러리 로딩
scaler = StandardScaler().fit(X_train)              표준화 스케일러 정의
standardized_X = scaler.transform(X_train)          표준화 스케일러 적용
```

### 정규화 (Normalization)

```
from sklearn.preprocessing import Normalizer        라이브러리 로딩
scaler = Normalizer().fit(X_train)                  정규화 스케일러 정의
normalized_X = scaler.transform(X_train)            정규화 스케일러 적용
```

### 범주형 변수의 수치화 (Encoding)

```
from sklearn.preprocessing import LabelEncoder      라이브러리 로딩
enc = LabelEncoder()                                라벨 인코더 정의
y = enc.fit_transform(y)                            라벨 인코더 적용
```

### 결측값 처리

```
from sklearn.preprocessing import Imputer                                   라이브러리 로딩
imp = Imputer(missing_values=0, strategy='mean', axis=0)                    대체기 (Inputer) 정의
imp.fit_transform(X_train)                                                  대체기 (Inputer) 적용
```

## 학습 모델 생성

### 지도 학습 (Supervised) 모델

```
from sklearn.linear_model import LinearRegression      회귀 모델 (선형 회귀)
lr = LinearRegression(normalize=True)

from sklearn.svm import SVC                            분류 모델 (SVM)
svc = SVC(kernel='linear')

from sklearn import neighbors                          분류 모델 (KNN)
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### 비지도 학습 (Unsupervised) 모델

```
from sklearn.decomposition import PCA                  차원 축소 (PCA)
pca = PCA(n_components=0.95)

from sklearn.cluster import KMeans                     분류 모델 (Kmeans)
k_means = KMeans(n_clusters=3, random_state=0)
```

## 모델 학습

### 지도 학습 (Supervised)

```
lr.fit(X, y)                                           모델 학습 (X : 문제, Y : 정답)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
```

### 비지도 학습 (Unsupervised)

```
k_means.fit(X_train)                                   모델 학습 (X : 문제)
pca_model = pca.fit_transform(X_train)
```

## 모델 예측

### 지도 예측 (Unsupervised)

```
y_pred = svc.predict(np.random.random((2,5)))          정답 라벨(Label) 예측
y_pred = lr.predict(X_test)
y_pred = knn.predict_proba(X_test))                    정답 확률(Probability) 예측
```

### 비지도 예측 (Unsupervised)

```
y_pred = k_means.predict(X_test)                       군집 (Cluster) 예측
```

## 모델 평가

### 분류 모델 평가

**정확도 (Accuracy)**

```
from sklearn.metrics import accuracy_score             정확도 측정
accuracy_score(y_test, y_pred)                         (실제값, 예측값 맞춘 비율)
```

**오차 행렬 (Confusion Matrix)**

```
from sklearn.metrics import confusion_matrix           오차 행렬 생성
print(confusion_matrix(y_test, y_pred)))
```

### 회귀 모델 평가

**MAE (Mean Absolute Error)**

```
from sklearn.metrics import mean_absolute_error        평균 절대 오차
y_true = [3, -0.5, 2])                                 (평균적으로 나타나는 오차)
mean_absolute_error(y_true, y_pred))
```

**MSE (Mean Squared Error)**

```
from sklearn.metrics import mean_squared_error         평균 제곱 오차
mean_squared_error(y_test, y_pred))                    (오차의 넓이)
```

**$R^2$ (R-Squared)**

```
from sklearn.metrics import r2_score                   적합도
r2_score(y_true, y_pred))
```

### 교차 검증 (Cross-Validation)

```
from sklearn.cross_validation import cross_val_score
print(cross_val_score(knn, X_train, y_train, cv=4))    분류 교차 검증 (4 분할)
print(cross_val_score(lr, X, y, cv=2))                 회귀 교차 검증 (2 분할)
```

## 모델 튜닝

### 그리드 탐색 (GridSearchCV)

```
from sklearn.grid_search import GridSearchCV                    그리드 조합 전체를 탐색
params = {"n_neighbors": np.arange(1,3),                        그리드 생성
          "metric": ["euclidean", "cityblock"]}
grid = GridSearchCV(estimator=knn,param_grid=params)            그리드 탐색 모델 정의
grid.fit(X_train, y_train)                                      그리드 탐색
print(grid.best_score_)                                         최적 파라미터 점수
print(grid.best_params)                                         최적 파라미터 출력
```

### 무작위 탐색 (RandomizedSearchCV)

```
from sklearn.grid_search import RandomizedSearchCV              무작위 조합으로 탐색
params = {"n_neighbors": range(1,5),                            그리드 생성
          "weights": ["uniform", "distance"]}
rsearch = RandomizedSearchCV(estimator=knn,                     무작위 탐색 모델 정의
                             param_distributions=params,
                             cv=4,                              교차 검증 데이터 분할 수
                             n_iter=8,                          탐색 횟수
                             random_state=5)
rsearch.fit(X_train, y_train)                                   무작위 탐색
print(rsearch.best_params)                                      최적 파라미터 출력
```