Scikit-Learn

2021

0. 머신 러닝이란?

❖ 인공지능의 분류

0. 머신 러닝이란?

❖ 초창기 정의

"Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort. 컴퓨터가 경험을 통해 학습할 수 있도록 프로그래밍할 수 있다면, 세세하게 프로그래밍해야 하는 번거로움에서 벗어날 수 있다[Samuel1959]."

❖ 현대적 정의

"A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. 어떤 컴퓨터 프로그램이 *T*라는 작업을 수행한다. 이 프로그램의 성능을 *P*라는 척도로 평가했을 때 경험 *E*를 통해 성능이 개선된다면 이 프로그램은 학습을 한다고 말할 수 있다[Mitchell1997(2쪽)]."

"Programming computers to optimize a performance criterion using example data or past experience 사례 데이터, 즉 과거 경험을 이용하여 성능 기준을 최적화하도록 프로그래밍하는 작업[Alpaydin2010]"

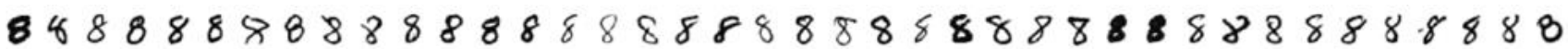
"Computational methods using experience to improve performance or to make accurate predictions 성능을 개선하거나 정확하게 예측하기 위해 경험을 이용하는 계산학 방법들[Mohri2012]"

❖ 지식 기반

- 예) 8 이란?

  "구멍이 2개이고 중간 부분이 홀쭉하며, 맨 위와 아래가 둥근 모양이라면 8이다"

- 한계
  - 단추를 "가운데 구멍이 몇 개 있는 물체"라고 규정하면 많은 오류 발생



- 사람은 변화가 심한 장면을 아주 쉽게 인식하지만, 왜 그렇게 인식하는지 서술하지는 못함
- 초창기 지식 기반에서 현재는 데이터 중심의 접근 방식이 대세

0. 머신 러닝이란?

❖ 학습 방법 종류

1. 사이킷런이란?

❖ 특징

- 파이썬 머신러닝 라이브러리 중 가장 많이 사용

  → 다른 머신러닝 패키지도 사이킷런 스타일의 API를 지향
- 매우 다양한 알고리즘과 개발을 위한 편리한 프레임워크와 API를 제공
- 오랜 기간 실전 환경에서 검증된 성숙한 라이브러리

  → Anaconda 설치시 기본 설치됨

❖ 학습 과정

- 전처리 작업
- 학습과 테스트 데이터로 분리
- 머신러닝 알고리즘을 적용해 학습
- 학습된 모델을 기반으로 테스트 데이터에 대한 예측 수행
- 예측값과 실제값을 비교해 머신러닝 모델에 대한 평가 수행

2. 아이리스 품종 예측하기

❖ 지도학습 > 분류

- 붓꽃 데이터 세트(4개의 피쳐) → 붓꽃의 품종(3가지) 예측

Feature	Label, Class
Sepal length Sepal width Petal length Petal width	Setosa Versicolor Virginica

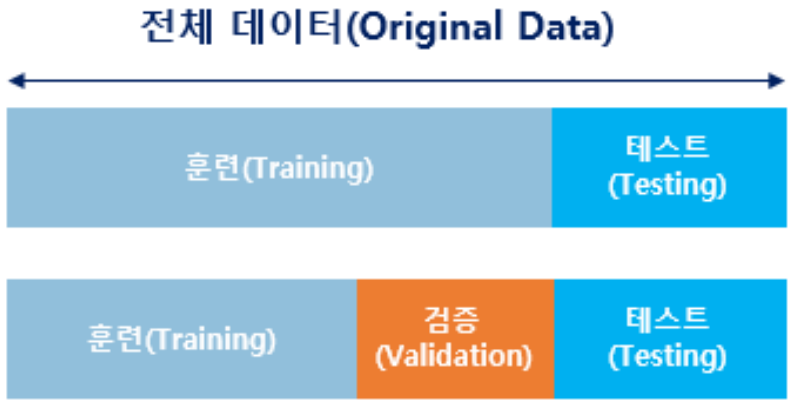
- 결정 트리, 서포트 벡터 머신 활용

2. 아이리스 품종 예측하기

1) 데이터 세트 로딩

```
from sklearn.datasets import load_iris
iris = load_iris()
iris_data = iris.data       # 피쳐 이름은 feature_names
iris_label = iris.target    # 레이블 이름은 target_names
                            # 데이터에 대한 설명은 DESCR
```

2) 학습용 데이터와 테스트용 데이터 분리




```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
    = train_test_split(iris_data, iris_label, test_size=0.2, random_state=11)
```

3) 결정 트리 알고리즘 사용

```
from sklearn.tree import DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(random_state=11)      # 객체 생성
```

4) 학습 수행

```
dt_clf.fit(X_train, y_train)
```

5) 예측 수행

```
pred = dt_clf.predict(X_test)

from sklearn.metrics import accuracy_score
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

6) 서포트 벡터 머신 알고리즘 사용

```
from sklearn.svm import SVC
sv_clf = SVC(random_state=11)       # 객체 생성
```

7) 학습 수행

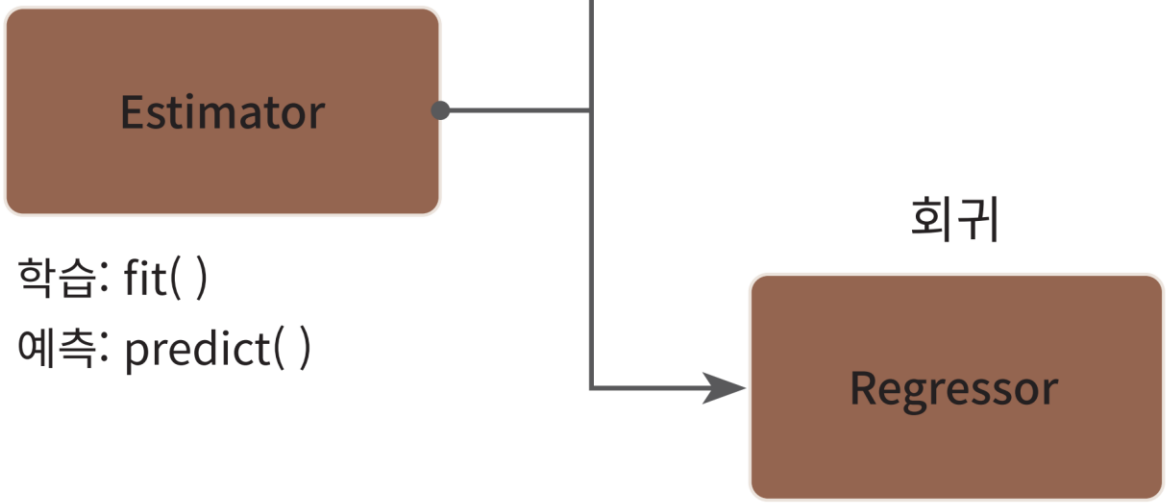
```
sv_clf.fit(X_train, y_train)
```

8) 예측 수행

```
pred = sv_clf.predict(X_test)

from sklearn.metrics import accuracy_score
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

❖ Estimator



분류 구현 클래스

DecisionTreeClassifier
RandomForestClassifier
GradientBoostingClassifer
GaussianNB
SVC

회귀 구현 클래스

LinearRegression
Ridge
Lasso
RandomForestRegressor
GradientBoostingRegressor

❖ 주요 모듈

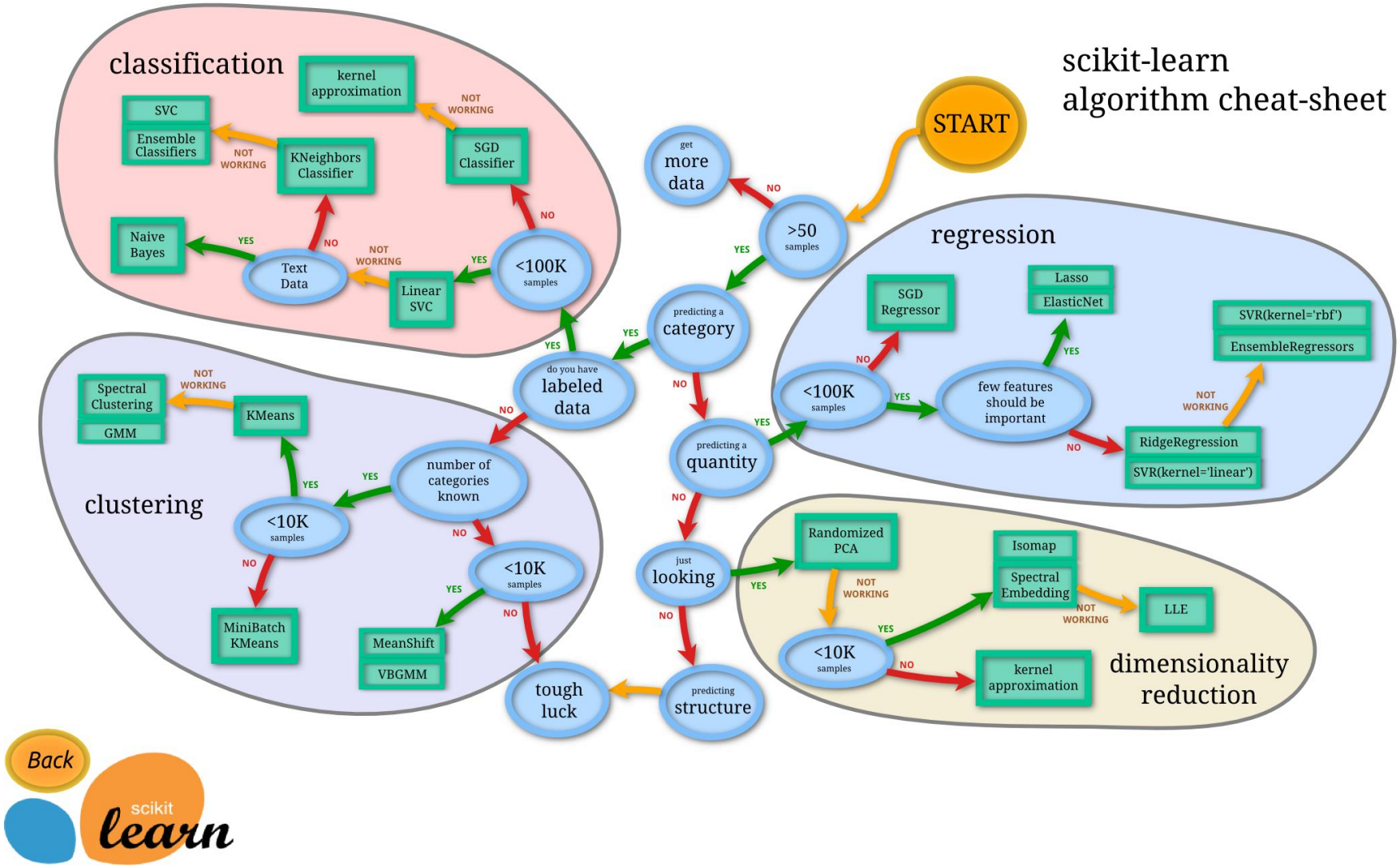
분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업을 수행하는 다양한 기능 제공

피처 처리	sklearn.feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는 데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tf-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 sklearn.feature_extraction.text 모듈에, 이미지 데이터의 피처 추출은 sklearn.feature_extraction.image 모듈에 지원 API가 있음.
피처 처리 & 차원 축소	sklearn.decomposition	차원 축소와 관련한 알고리즘을 지원하는 모듈임. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있음
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
	sklearn.ensemble	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공

3. 사이킷런 기반 프레임워크

ML 알고리즘	sklearn.linear_model	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	sklearn.naive_bayes	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등.
	sklearn.neighbors	최근접 이웃 알고리즘 제공. K-NN 등
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사 결정 트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

❖ 알고리즘 치트 시트(Cheat Sheet)



❖ 내장된 데이터 세트

API 명	설명
datasets.load_boston()	회귀 용도이며, 미국 보스턴의 집 피처들과 가격에 대한 데이터 세트
datasets.load_breast_cancer()	분류 용도이며, 위스콘신 유방암 피처들과 악성/음성 레이블 데이터 세트
datasets.load_diabetes()	회귀 용도이며, 당뇨 데이터 세트
datasets.load_digits()	분류 용도이며, 0에서 9까지 숫자의 이미지 픽셀 데이터 세트
datasets.load_iris()	분류 용도이며, 붓꽃에 대한 피처를 가진 데이터 세트

분류와 클러스터링을 위한 표본 데이터 생성기

API 명	설명
datasets.make_classifications( )	분류를 위한 데이터 세트를 만듭니다. 특히 높은 상관도, 불필요한 속성 등의 노이즈 효과를 위한 데이터를 무작위로 생성해 줍니다.
datasets.make_blobs( )	클러스터링을 위한 데이터 세트를 무작위로 생성해 줍니다. 군집 지정 개수에 따라 여러 가지 클러스터링을 위한 데이터 세트를 쉽게 만들어 줍니다.

4. Model Selection 모듈

❖ 학습/테스트 데이터 세트 분리 – `train_test_split()`

- test_size: 디폴트는 0.25
- train_size
- shuffle: 데이터를 분리하기 전에 데이터를 미리 섞을지를 결정. 디폴트는 True
- random_state
- 반환 값: 튜플 형태
  - 학습용 데이터의 피쳐 데이터 세트(X_train)
  - 테스트용 데이터의 피쳐 데이터 세트(X_test)
  - 학습용 데이터의 레이블 데이터 세트(y_train)
  - 테스트용 데이터의 레이블 데이터 세트(y_test)

4. Model Selection 모듈

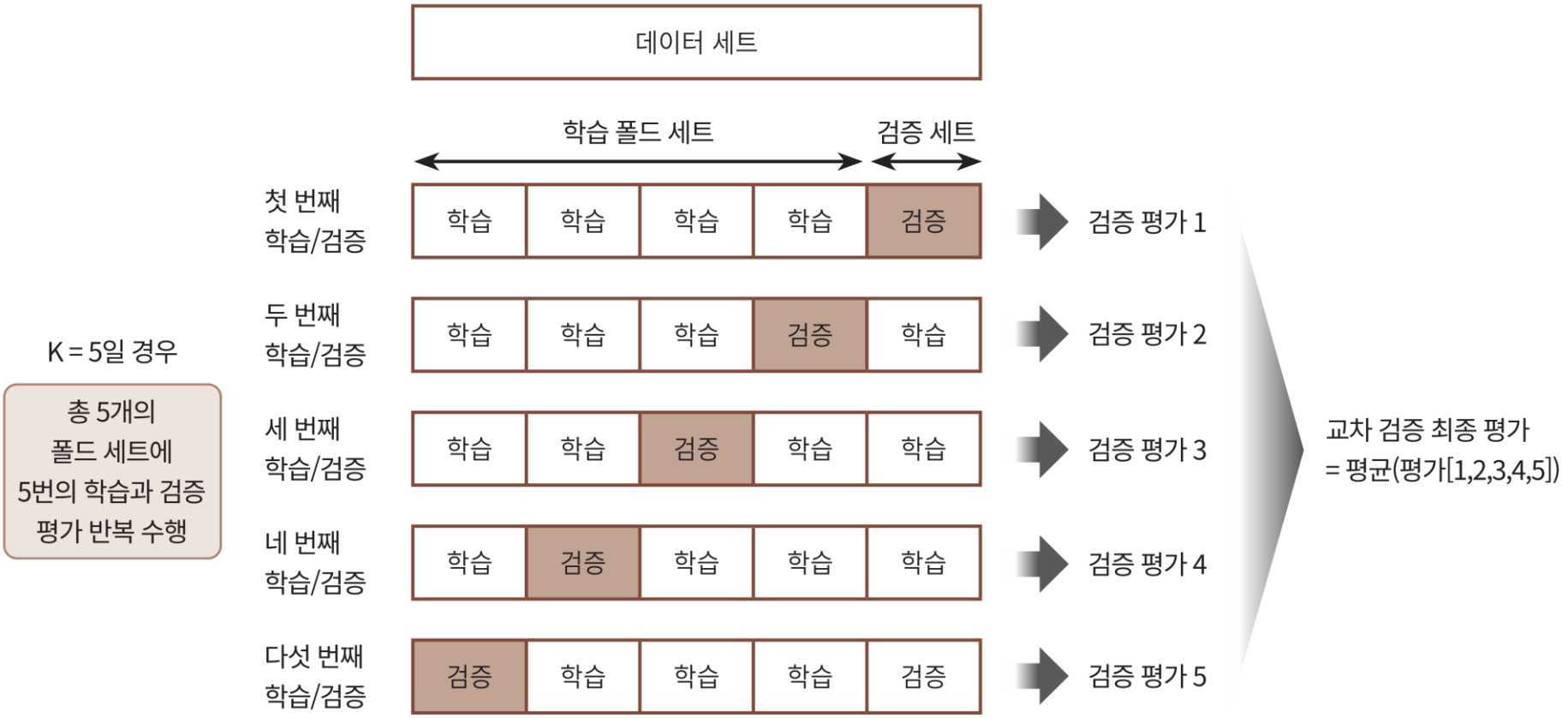
❖ 교차 검증

- 데이터의 편중을 막기 위함
- 과적합을 피하고 성능 저하를 막기 위함
- 교차 검증 기반으로 1차 평가 → 최종적으로 테스트 데이터 세트에 적용

4. Model Selection 모듈

❖ K 폴드 교차 검증

- 가장 보편적으로 사용되는 교차 검증 기법



4. Model Selection 모듈

```
from sklearn.model_selection import KFold
import numpy as np
kfold = KFold(n_splits=5)
cv_accuracy = []
features, label = iris.data, iris.target
dt_clf = DecisionTreeClassifier(random_state=156)
print('붓꽃 데이터 세트 크기:', features.shape[0])

n_iter = 0
for train_index, test_index in kfold.split(features):
    # kfold.split( )으로 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]
    #학습 및 예측
    dt_clf.fit(X_train , y_train)
    pred = dt_clf.predict(X_test)
    n_iter += 1
    # 반복 시 마다 정확도 측정
    accuracy = np.round(accuracy_score(y_test, pred), 4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]
    print(f'\n#{n_iter} 교차 검증 정확도 :{accuracy}, 학습 데이터 크기: {train_size}, 검증 데이터
크기: {test_size}')
    print(f'#{n_iter} 검증 세트 인덱스:{test_index}')
    cv_accuracy.append(accuracy)

print('## 평균 검증 정확도:', np.mean(cv_accuracy))
```

❖ Stratified K 폴드

- 불균형한 분포를 가진 레이블(결정 클래스) 데이터 집합을 위한 K 폴드 방식

```
import pandas as pd
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

kfold = KFold(n_splits=3)
n_iter = 0
for train_index, test_index in kfold.split(iris_df):
    n_iter += 1
    label_train= iris_df['label'].iloc[train_index]
    label_test= iris_df['label'].iloc[test_index]
    print(f'## 교차 검증: {n_iter}')
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())

from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=3)

n_iter = 0
for train_index, test_index in skf.split(iris_df, iris_df['label']):
    n_iter += 1
    label_train= iris_df['label'].iloc[train_index]
    label_test= iris_df['label'].iloc[test_index]
    print('## 교차 검증: {n_iter}')
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())
```

❖ 편리한 교차 검증 방법 – cross_val_score()

- K 폴드 방식
  - ① 폴드 세트를 설정
  - ② for 루프에서 반복으로 학습 및 테스트 데이터의 인덱스 추출
  - ③ 반복적으로 학습과 예측을 수행하고 예측 성능을 반환
- cross_val_score() 는 위 과정을 한꺼번에 수행해줌

```
from sklearn.model_selection import cross_val_score

# 성능 지표는 정확도(accuracy), 교차 검증 세트는 3개
scores = cross_val_score(dt_clf, iris_data, iris_label, scoring='accuracy', cv=3)
print('교차 검증별 정확도:', np.round(scores, 4))
print('평균 검증 정확도:', np.round(np.mean(scores), 4))
```

❖ 교차 검증과 최적 하이퍼 파라미터 튜닝을 한번에 – `GridSearchCV`

- 하이퍼 파라미터
  - Classifier나 Regressor에 사용되는 하이퍼 파라미터
  - 예측 성능 개선에 도움을 줌
- GridSearchCV의 파라미터
  - estimator: classifier, regressor, pipeline
  - param_grid: 튜닝을 위해 파라미터명과 사용될 여러가지 파라미터값을 지정
  - scoring: 예측 성능을 측정할 평가 방법
  - cv: 교차 검증을 위한 학습/테스트 세트의 갯수
  - refit: 최적의 하이퍼 파라미터를 찾은 뒤, 입력된 estimator 객체를 재학습
    디폴트는 True

❖ GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# 데이터를 로딩하고 학습 데이터와 테스트 데이터 분리
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label,
                                                    test_size=0.2, random_state=121)
dtree = DecisionTreeClassifier()

### parameter 들을 dictionary 형태로 설정
parameters = {'max_depth':[2,3,4], 'min_samples_split':[2,3]}

# param_grid의 파라미터들을 3개의 train, test set으로 나누어서 테스트 수행 설정.
grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)

# 붓꽃 Train 데이터로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .
grid_dtree.fit(X_train, y_train)

# GridSearchCV 결과 추출하여 DataFrame으로 변환
scores_df = pd.DataFrame(grid_dtree.cv_results_)
scores_df[['params', 'mean_test_score', 'rank_test_score',
           'split0_test_score', 'split1_test_score', 'split2_test_score']]
```

❖ GridSearchCV

```
print('GridSearchCV 최적 파라미터:', grid_dtree.best_params_)
print(f'GridSearchCV 최고 정확도: {grid_dtree.best_score_:.4f}')

# GridSearchCV의 refit으로 이미 학습이 된 estimator 반환
estimator = grid_dtree.best_estimator_

# GridSearchCV의 best_estimator_는 이미 최적 하이퍼 파라미터로 학습이 됨
pred = estimator.predict(X_test)
print(f'테스트 데이터 세트 정확도: {accuracy_score(y_test, pred):.4f}')
```

❖ 데이터 전처리

- ML 알고리즘은 데이터에 기반하고 있음
- Garbage In, Garbage Out
- 결손값(NaN, Null)은 허용 안됨
  - 결손값의 비율이 작을 때 – 평균값, 중앙값, 최빈값 등으로 대체
  - 결손값의 비율이 클 때 – 해당 피쳐는 드롭
  - 결손값의 비율이 어중간할 때
- 문자열 값은 입력값으로 허용 안됨
  - 인코딩을 통해 숫자로 변환되어야 함
  - 카테고리형 – 코드 값으로 표현
  - 텍스트형 – 벡터화(CountVectorizer, TfidfVectorizer)

❖ 데이터 인코딩

▪ 레이블 인코딩

원본 데이터

상품 분류	가격
TV	1,000,000
냉장고	1,500,000
전자렌지	200,000
컴퓨터	800,000
선풍기	100,000
선풍기	100,000
믹서	50,000
믹서	50,000

상품 분류를 레이블 인코딩한 데이터

상품 분류	가격
0	1,000,000
1	1,500,000
4	200,000
5	800,000
3	100,000
3	100,000
2	50,000
2	50,000

```
from sklearn.preprocessing import LabelEncoder

items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# LabelEncoder를 객체로 생성한 후, fit( ) 과 transform( )으로 label 인코딩 수행.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:', labels)

encoder.inverse_transform([4, 5, 2, 0, 1, 1, 3, 3])
```

❖ 데이터 인코딩

▪ 원-핫(One-Hot) 인코딩

원본 데이터

상품 분류
TV
냉장고
전자렌지
컴퓨터
선풍기
선풍기
믹서
믹서

원-핫 인코딩

상품분류_ TV	상품분류_ 냉장고	상품분류_ 믹서	상품분류_ 선풍기	상품분류_ 전자렌지	상품분류_ 컴퓨터
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

❖ 데이터 인코딩

▪ 원-핫(One-Hot) 인코딩

```
from sklearn.preprocessing import OneHotEncoder

items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# 먼저 숫자값으로 변환을 위해 LabelEncoder로 변환
encoder = LabelEncoder()
labels = encoder.fit_transform(items)
labels.shape

# 2차원 데이터로 변환
labels = labels.reshape(-1,1)
labels.shape

# 원-핫 인코딩을 적용
oh_encoder = OneHotEncoder()
oh_encoder.fit(labels)
oh_labels = oh_encoder.transform(labels)
print('원-핫 인코딩 데이터\n', oh_labels.toarray())
print('원-핫 인코딩 데이터 차원\n', oh_labels.shape)
```

❖ 피쳐 스케일링과 정규화

- 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
- 표준화(Standardization)
  - 평균이 0이고, 표준편차가 1인 가우시안 정규분포로 변환
  - StandardScaler
- 정규화(Normalization)
  - 최소 0 ~ 최대 1 값으로 변환
  - MinMaxScaler

❖ StandardScaler

```
# 붓꽃 데이터 셋을 로딩하고 DataFrame으로 변환
iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)

print('feature 들의 평균 값\n', iris_df.mean())
print('feature 들의 표준편차 값\n', iris_df.std())

from sklearn.preprocessing import StandardScaler

# StandardScaler객체 생성
scaler = StandardScaler()
# StandardScaler 로 데이터 셋 변환. fit( ) 과 transform( ) 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature 들의 평균 값')
print(iris_df_scaled.mean())
print('feature 들의 표준편차 값')
print(iris_df_scaled.std())
```

❖ MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler

# MinMaxScaler객체 생성
scaler = MinMaxScaler()
# MinMaxScaler 로 데이터 셋 변환. fit() 과 transform() 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

# transform()시 변환된 데이터 셋이 numpy ndarry로 반환되고 이를 DataFrame으로 변환
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature들의 최소 값')
print(iris_df_scaled.min())
print('feature들의 최대 값')
print(iris_df_scaled.max())
```

6. 타이타닉 생존자 예측

❖ 캐글 제공 타이타닉 탑승자 데이터 기반

1) 캐글 데이터(https://www.kaggle.com/c/titanic/data) 다운로드
2) 데이터 확인
3) 결손치 처리
4) 문자열 처리
5) Survived 속성을 y로 나머지 피쳐를 X로 만듦
6) 학습/테스트 데이터 세트 분리
7) 적용할 알고리즘 선정:
    - 결정 트리 – DecisionTreeClassifier
    - 랜덤 포레스트 – RandomForestClassifier
    - 로지스틱 회귀 – LogisticRegression
8) 학습, 예측 및 평가
9) 교차 검증
10) GridSearchCV를 통해 최적 파라미터 도출