Scikit-learn

Scikit-learn
개요

- 파이썬에서 가장 많이 사용되는 기계학습 라이브러리


```
pip install scikit-learn

conda install scikit-learn
```

Scikit-learn
개요

- 다양한 기계학습 알고리즘 지원
  - 선형 회귀
  - k-NN 알고리즘
  - 서포트 벡터머신
  - k-means

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import linear_model  # scikit-learn 모듈을 가져온다

data_home = 'https://github.com/dknife/ML/raw/main/data/'
lin_data = pd.read_csv(data_home+'pollution.csv')
```

선형 회귀
개요

- 선형 회귀는 임의의 변수 x(독립변수)와 이 변수에 따른 또 다른 변수 y(종속변수)의 상관관계를 모델링하는 기법
- 독립변수 x 와 종속변수 y의 관계를 함수식으로 설명하는 것
- 두 변수의 관계를 알아내거나 이를 이용하여 y가 없는 x값에 대하여 y를 예측하는데 사용되는 통계학의 기법
- sklearn 패키지의 datasets 서브패키지를 사용하여 회귀분석을 진행

Scikit-learn
개요

- 선형 회귀 모델의 입력 데이터는 2차원 배열로 구성
  - 각 행은 데이터 인스턴스
  - 데이터 인스턴스은 여러 개의 특징값을 가질 수 있음

```
x = lin_data['input'].to_numpy()
y = lin_data['pollution'].to_numpy()
x = x[:, np.newaxis]     # 선형 회귀 모델의 입력형식에 맞게 차원을 증가시킴
print(x)
```

```
[[0.24055707]
 [0.1597306 ]
 ...
 [0.00720486]
 [0.29029368]]
```

회귀문제

데이터를 잘 설명하는 함수를 찾아라

- 선형회귀 모델을 생성하는 함수로 입력벡터 X를 목표값 y에 최적화시키는(fitting) 모델을 생성함

```
X = [[164], [179], [162], [170]]          # 다중회귀에도 사용하도록 함
y = [53, 63, 55, 59]                      # y = f(X)의 결과
regr.fit(X, y)
```

14장_기계학습으로.ipynb

회귀문제

데이터를 잘 설명하는 함수를 찾아라

- 직선의 방정식의 기울기와 절편을 확인함

$$y = w_0 + w_1 X_1$$

```
coef = regr.coef_              # 직선의 기울기
intercept = regr.intercept_  # 직선의 절편
score = regr.score(X, y)      # 학습된 직선이 데이터를 얼마나 잘 따르나

print("y =", coef, "* X + ", intercept)
print("The score of this line for the data: ", score)
```

```
y = [0.55221745] * X +  -35.686695278969964
The score of this line for the data:  0.903203123105647
```

- 두개의 자료(키)를 통해 몸무게(y) 를 예측함

```
input_data = [ [180], [185] ]
```

```
result = regr.predict(input_data)
print(result)
```

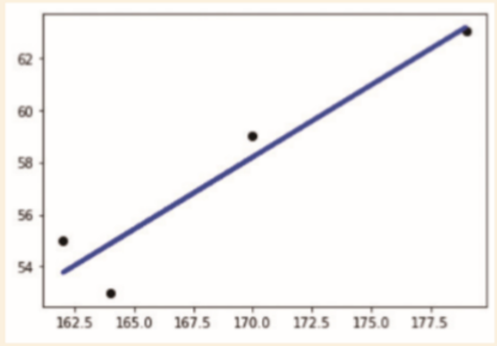
```
[63.71244635 66.47353362]
```

회귀문제

데이터를 잘 설명하는 함수를 찾아라

- 그래프로 그리기 위하여 matplotlib 라이브러리 사용

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model  # scikit-learn 모듈을 가져온다

regr = linear_model.LinearRegression()

X = [[164], [179], [162], [170]]  # 선형회귀의 입력은 2차원으로 만들어야 함
y = [53, 63, 55, 59]              # y = f(X)의 결과값
regr.fit(X, y)

# 학습 데이터와 y 값을 산포도로 그린다.
plt.scatter(X, y, color='black')

# 학습 데이터를 입력으로 하여 예측값을 계산한다.
y_pred = regr.predict(X)

# 학습 데이터와 예측값으로 선그래프로 그린다.
# 계산된 기울기와 y 절편을 가지는 직선이 그려진다
plt.plot(X, y_pred, color='blue', linewidth=3)
plt.show()
```

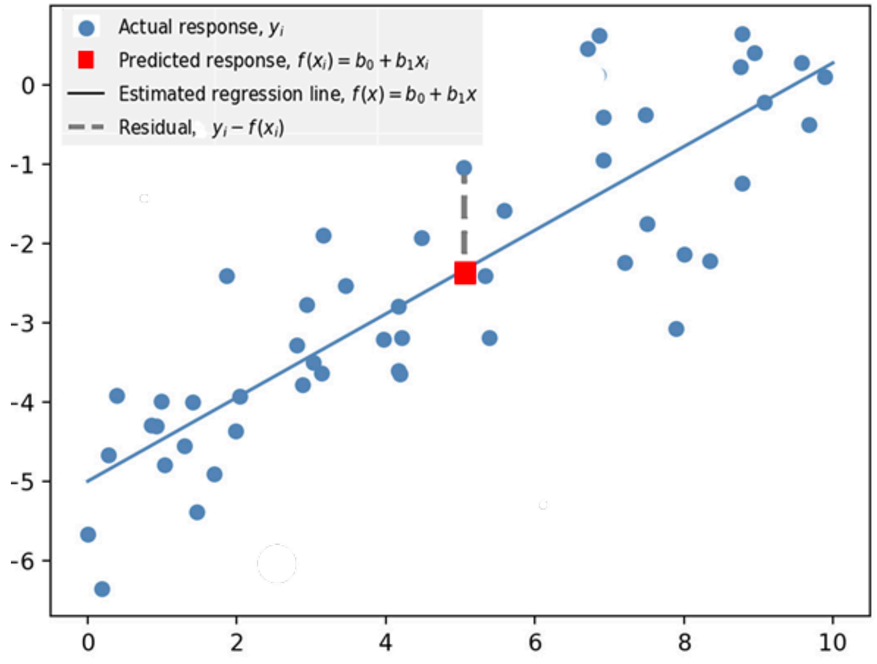


Scikit-learn

fit() 메소드

- 선형회귀를 위한 학습

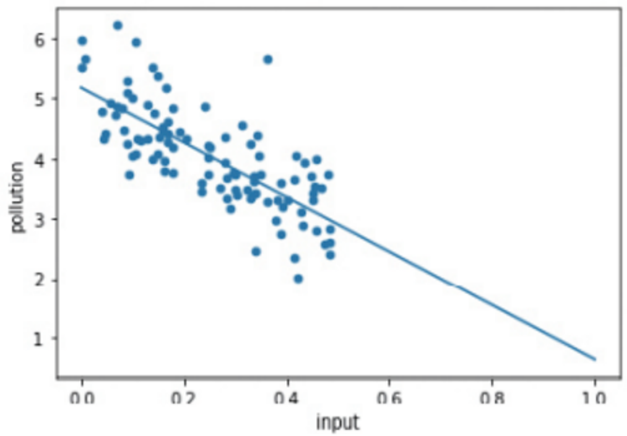
```
regr = linear_model.LinearRegression()
regr.fit(x, y)    # 선형 회귀 모델에 데이터를 넣어 학습을 진행함
```



Scikit-learn
fit() 메소드

- 입력으로 0과 1을 주고 이에 해당하는 출력값을 예측함

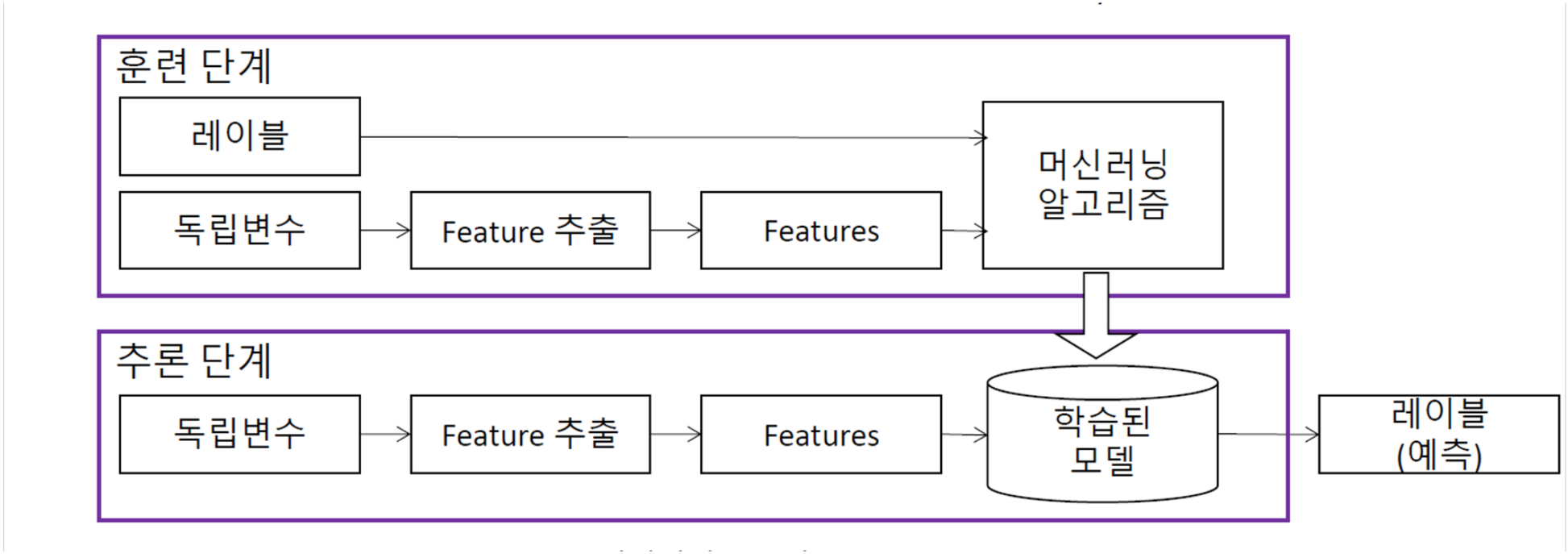
```
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')
y_pred = regr.predict([[0], [1]])
plt.plot([0, 1], y_pred)    # x 구간을 0에서 1 사이로 두자
```



회귀문제

보스턴 지역 집값 예측

- 학습과 추론(테스트)



회귀문제

보스턴 지역 집값 예측

- sklearn.datasets 라이브러리에서 load_boston 모듈을 사용함
  - x 변수 13, y 변수 1

- from sklearn.datasets import load_boston
- boston = load_boston()
- dir(boston)
- ['DESCR', 'data', 'feature_names', 'filename', 'target']

회귀문제

보스턴 지역 집값 예측

- 독립변수 행렬을 X로, 종속변수 벡터를 y로 구성, 종속변수의 이름은 MEDV로 지정

```
X = pd.DataFrame(boston.data, columns=boston.feature_names)

y = pd.DataFrame(boston.target, columns=[“MEDV"])
```

- 독립변수과 종속변수 데이터프레임을 하나의 데이터프레임 df 로 구성

```
df = pd.concat([X, y], axis=1)

df.tail()
```

회귀문제

보스턴 지역 집값 예측

- 독립변수와 종속변수의 관계를 스캐터플롯(scatter plot)으로 살펴봄
  - 종속변수인 집값(MEDV)과 방 개수(RM), 노후화 정도(AGE)와 어떤 관계를 확인
  - 방 개수가 증가할 수록 집값은 증가하는 경향이 있음
  - 노후화 정도와 집값은 관계가 없음

회귀문제

보스턴 지역 집값 예측

- 특징 선택

```
import numpy as np

X = pd.DataFrame(np.c_[df["LSTAT"], df["RM"]], columns=["LSTAT", "RM"])

y = df[“price”]
```

- 훈련과 테스트 데이터 분류

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=5)
```

회귀문제

보스턴 지역 집값 예측

- 학습모델 생성

```
from sklearn.linear_model import LinearRegression


model = LinearRegression()

model.fit(x_train, y_train)
```

회귀문제

보스턴 지역 집값 예측

- 학습모델 평가

```
from sklearn.metrics import mean_squared_error, r2_score

y_train_predict = model.predict(x_train)

rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))

r2 = r2_score(y_train, y_train_predict)

print("train set")

print("price is {}".format(rmse))

print("R2 Score is {}".format(r2))
```

회귀문제

보스턴 지역 집값 예측

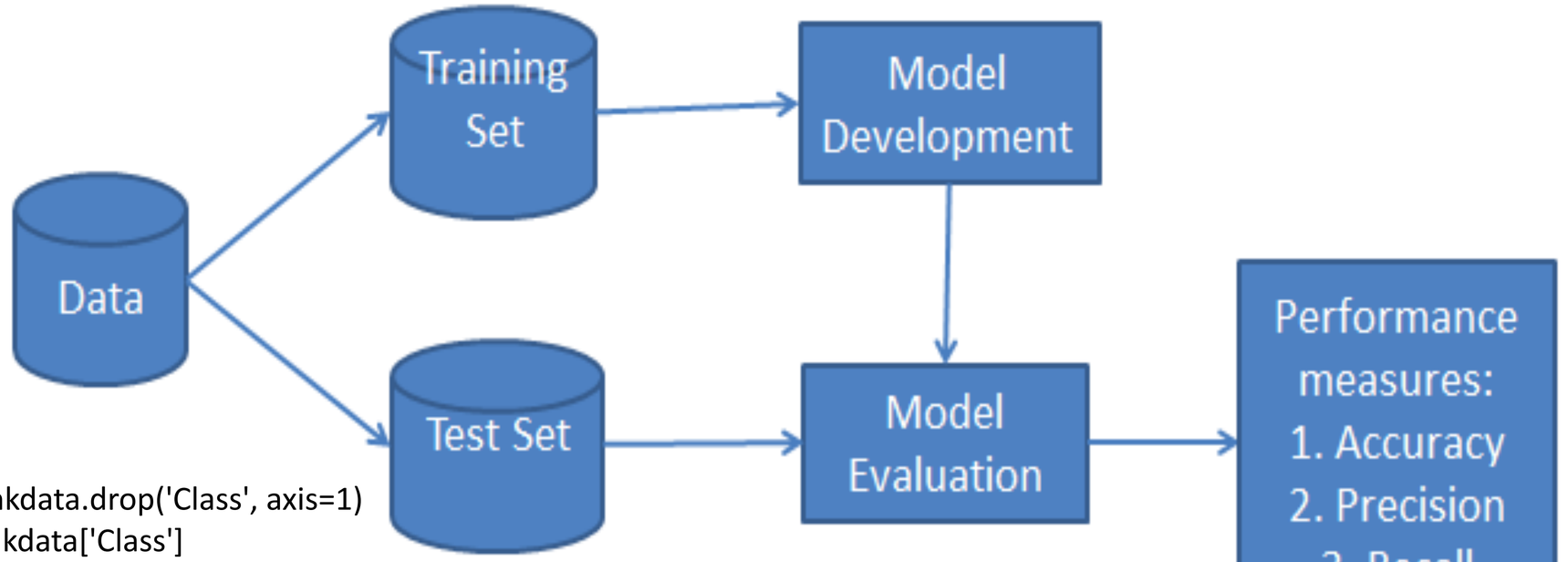
- 학습모델 평가

```
y_train_predict = model.predict(x_test)

rmse = (np.sqrt(mean_squared_error(y_test, y_train_predict)))

r2 = r2_score(y_test, y_train_predict)

print("train set")

print("price is {}".format(rmse))

print("R2 Score is {}".format(r2))
```

분류

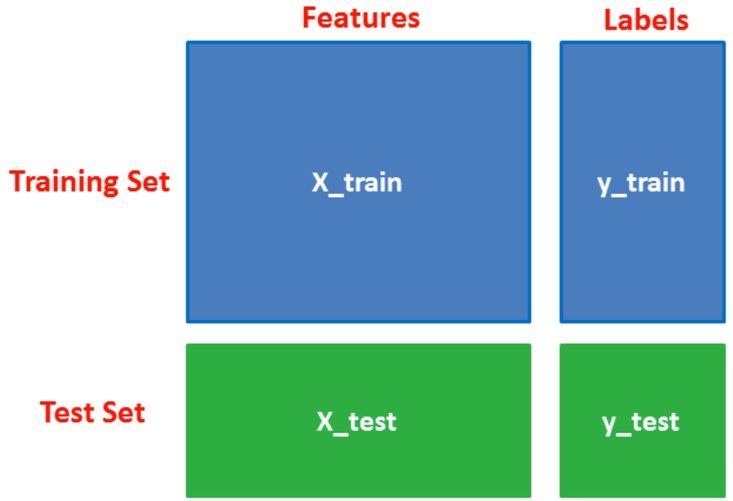
- 데이터를 어떤 기준에 따라 나누는 기법
- 이미지를 입력했을 때 이미지가 개인지 고양이 인지 분류하는 것
- y 값은 이산형 값을 갖으며, 분류 형태에 따라 이진분류와 다중 분류로 나눔

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)



X = bankdata.drop('Class', axis=1)
y = bankdata['Class']

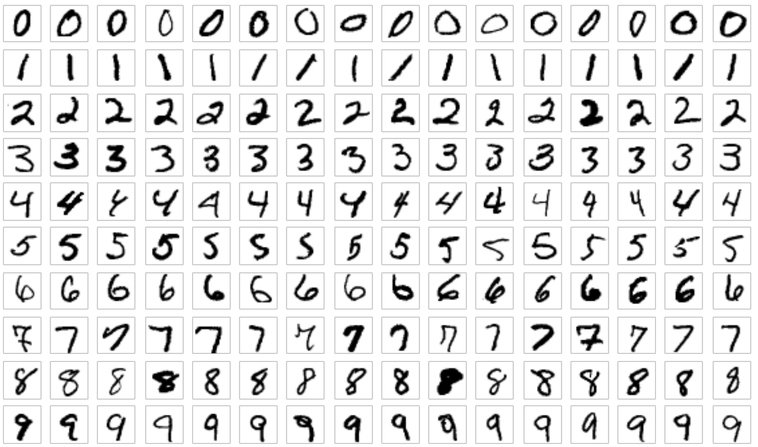
```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
```



분류

MNIST

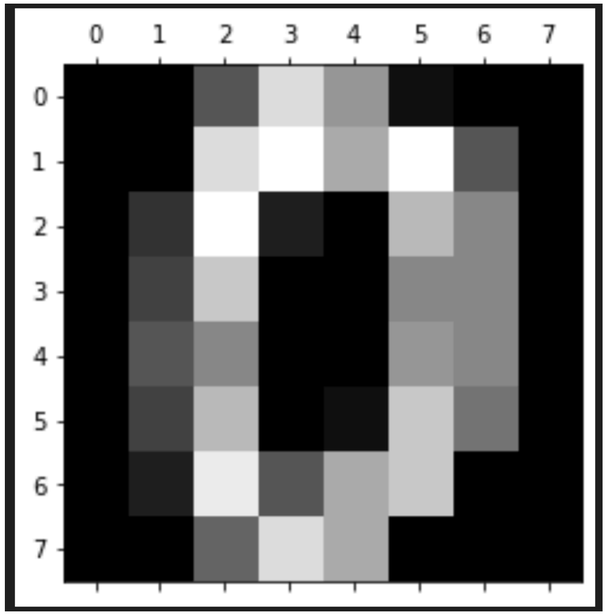
- 손글씨를 숫자로 인식하는 이미지 분류 문제
- MNIST 데이터셋을 사용하여 이미지를 학습한 후 숫자로 분류
- 입력자료는 24* 24 이진 이미지를 사용



분류

데이터 로드

- from sklearn.datasets import load_digits
- digits = load_digits()
- print(digits.data.shape)
- (1797, 64)
- digits.keys()
- dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', ‘DESCR'])



분류

데이터 셋을 데이터프레임으로 변환

- # feature_names 와 target을 레코드로 갖는 데이터프레임 생성
- df = pd.DataFrame(data=digits.data, columns=digits.feature_names)
- x_data = df
- df['target'] = digits.target
- y_data = digits.target

분류

데이터 분할 (train_test_split)

- from sklearn.model_selection import train_test_split
- x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3)

분류

Classifier

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
estimator = SVC(kernel='linear', C=1.0) #linear SVM
estimator.fit(x_train, y_train)
y_predict = estimator.predict(x_train)
score = accuracy_score(y_train, y_predict)
print(score) #1.0

y_predict = estimator.predict(x_test)
score = accuracy_score(y_test, y_predict)
print(score) #1.0
```

*분류

Performance Evaluation

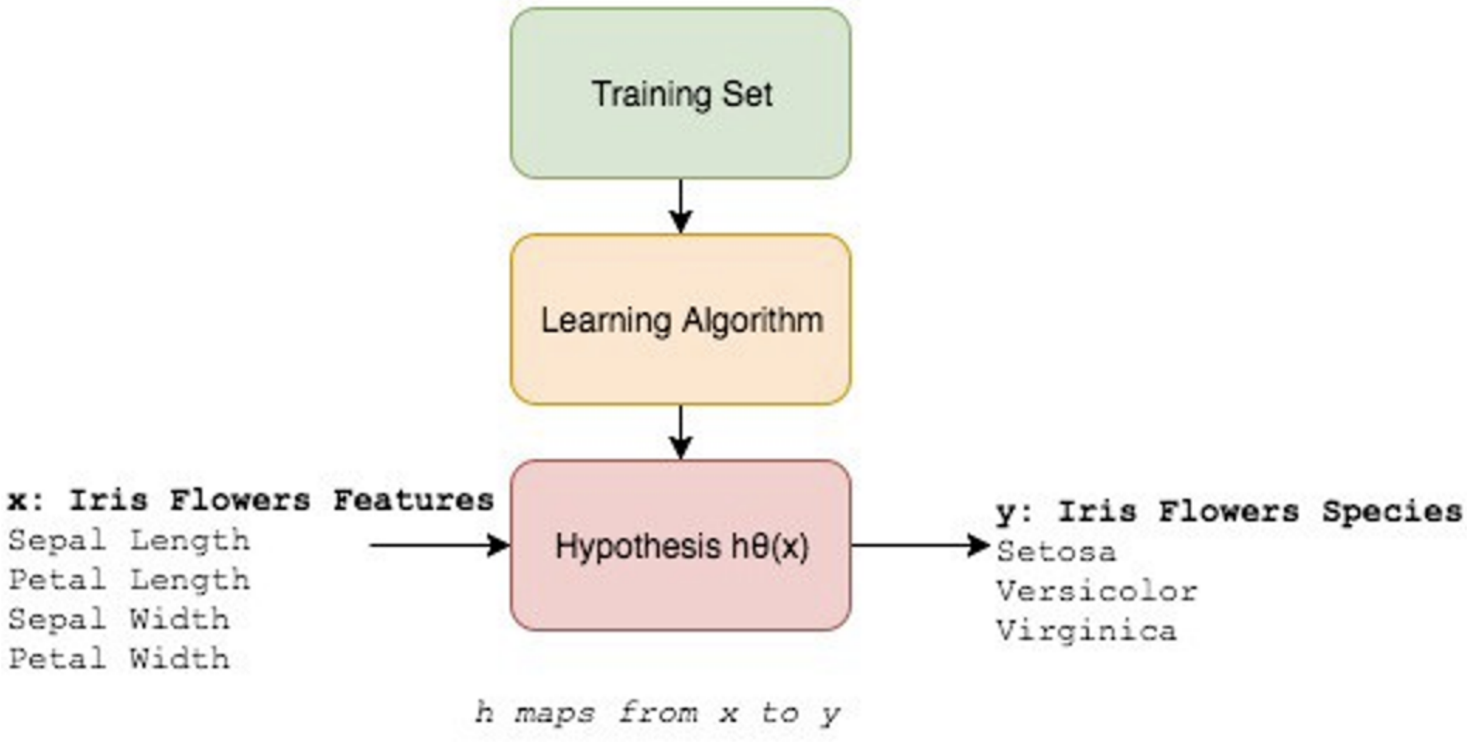
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       147
           1       0.99      1.00      1.00       128

   micro avg       1.00      1.00      1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

분류

아이리스 데이터셋

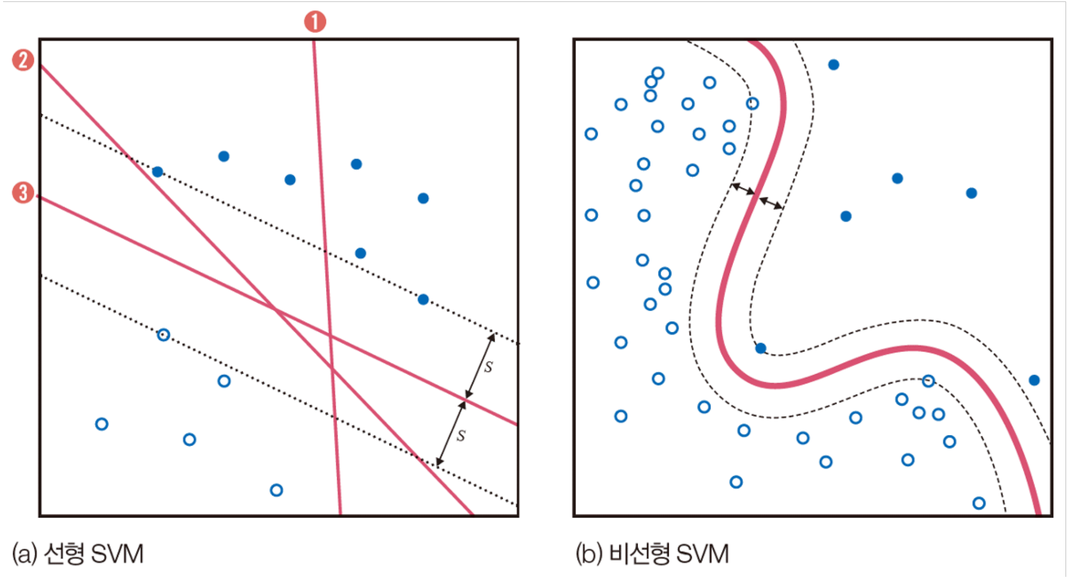
- 분류함수 h(X)를 data[X,y] 학습으로 구성, 입력 X로 y를 분류 함



분류
적용예시

- 분류하기

```
from sklearn import svm

s=svm.SVC(gamma=0.1,C=10) # svm 분류 모델 SVC 객체 생성하고
s.fit(data.data,data.target) # iris 데이터로 학습

# 101번째와 51번째 샘플을 변형하여 새로운 데이터 생성
new_d=[[6.4,3.2,6.0,2.5],[7.1,3.1,4.7,1.35]]
res=s.predict(new_d)
print("새로운 2개 샘플의 부류는", res)
```



(a) 선형 SVM

(b) 비선형 SVM

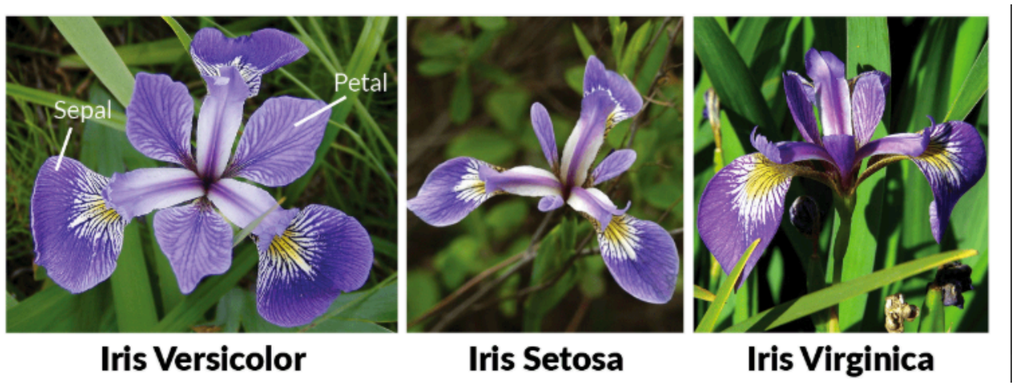
- 새로운 2개 샘플의 부류는 [2 1]

분류

데이터 로드

- 아이리스 데이터셋

```
from sklearn.datasets import load_iris

from matplotlib import pyplot as plt


data = load_iris()
```



객체 data의 DESCR 변수를 출력하시오

분류

데이터 로드

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

```
 :Summary Statistics:

 ============== ==== ==== ======= ===== ====================
                 Min  Max   Mean    SD   Class Correlation
 ============== ==== ==== ======= ===== ====================
 sepal length:   4.3  7.9   5.84   0.83    0.7826
 sepal width:    2.0  4.4   3.05   0.43   -0.4194
 petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
 petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
 ============== ==== ==== ======= ===== ====================
```

분류

데이터 구조 파악하기

- 3개 종, 4개 특징(feature)

features = data['data']

feature_names = data['feature_names']

target = data[‘target’]

features[: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]])

- 배열 features 를 특징 벡터라고 하며, 배열 target 은 레이블이라고 함

분류

아이리스 데이터셋

```
print(data["target"].shape)  # 자료 수

unique, counts = np.unique(target, return_counts=True)

print ("unique : ", unique)  # 레이블 값

print ("counts : ", counts)  # 레이블 별 수
```

```
(150,)

unique :  [0 1 2]

counts :  [50 50 50]
```

분류
레이블

- 레이블 (또는 타켓) 얻기

target = data ['target']

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, 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, 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,

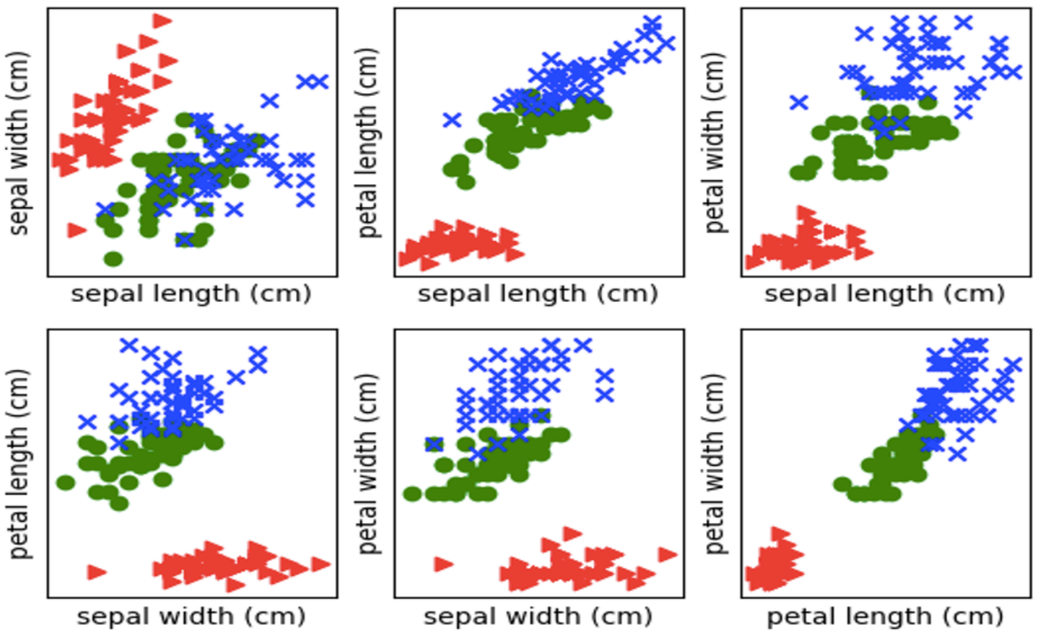
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

분류

산포도 시각화

```
for t in range(3):
    if t == 0:
        c='r'
        marker='>'
    elif t == 1:
        c='g'
        marker='o'
    elif t == 2:
        c='b'
        marker='x'
    plt.scatter(features[target == t, 0],    # sepal length
                features[target == t, 1],    # sepal width
                marker = marker,
                c = c)
    plt.xlabel("sepal length")
    plt.ylabel("sepal width")
```



분류

데이터 셋을 데이터프레임으로 변환

```
import pandas as pd # 데이터 프레임으로 변환을 위해 임포트

import numpy as np # 고수학 연산을 위해 임포트


# feature_names 와 target을 레코드로 갖는 데이터프레임 생성

df = pd.DataFrame(data=data.data, columns=iris.feature_names)

df['target'] = iris.target


# 0.0, 1.0, 2.0으로 표현된 label을 문자열로 매핑

df['target'] = df['target'].map({0:"setosa", 1:"versicolor", 2:"virginica"})

print(df)
```

분류

데이터 분할 (train_test_split)

- sklearn.model_selection의 train_test_split은 학습과 검증 (혹은 테스트) 셋으로 구분함
- 학습 (Train) / 검증 (Validation or Test) 세트로 나누며, 검증 세트로 과대 적합여부를 모니터링

```
from sklearn.model_selection import train_test_split

x = df.iloc[:, :4]

x.head()
```

분류

데이터 분할 (train_test_split)

- 훈련 집합과 테스트 집합
  - 훈련 집합: 기계 학습 모델을 학습하는데 쓰는 데이터로서 특징 벡터와 레이블 정보를 모두 제공
  - 테스트 집합: 학습을 마친 모델의 성능을 측정하는데 쓰는 데이터로서 예측할 때는 특징 벡터 정보만 제공하고, 예측 결과를 가지고 정확률을 측정할 때 레이블 정보를 사용

분류

데이터 분할 (train_test_split)

- sklearn.model_selection의 train_test_split은 학습과 검증 (혹은 테스트) 셋으로 구분함
- 학습 (Train) / 검증 (Validation or Test) 세트로 나누며, 검증 세트로 과대 적합여부를 모니터링

```
from sklearn.model_selection import train_test_split

x = df.iloc[:, :4]

x.head()

x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.2, random_state=30)
```

분류

데이터 분할 (train_test_split)

- 주어진 데이터를 적절한 비율로 훈련, 검증, 테스트 집합으로 나누어 씀
  - 모델 선택 포함: 훈련/검증/테스트 집합으로 나눔
  - 모델 선택 제외: 훈련/테스트 집합으로 나눔

(a) 모델 선택 포함

(b) 모델 선택 제외

분류

데이터 분할 (train_test_split)

x_train.shape, x_test.shape

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

y.shape

(150,)

y_train.shape, y_test.shape

((120,), (30,))

분류

데이터 분할 (train_test_split)

- 훈련/테스트 집합 나누기의 한계
  - 우연히 높은 정확률 또는 낮은 정확률 발생 가능성
- k-겹 교차 검증 k-fold cross validation
  - 훈련 집합을 k개의 부분집합으로 나누어 사용
  - 한 개를 남겨두고 k-1개로 학습
  - 남겨둔 것으로 성능 측정
  - k개의 성능을 평균하여 신뢰도 높임

(a) 모델 선택 포함

(b) 모델 선택 제외

그림 3-12 k-겹 교차 검증(k=5인 경우)

분류

데이터 분할 (train_test_split)

- k-겹 교차 검증 k-fold cross validation

```
digit=datasets.load_digits()

s=svm.SVC(gamma=0.001)

accuracies=cross_val_score(s,digit.data,digit.target,cv=5) # 5-겹 교차 검증

print(accuracies)

print("정확률(평균)=%0.3f, 표준편차 =%0.3f"%(accuracies.mean()*100,accuracies.std()))
```

분류

규칙 기반 vs. 고전적 기계 학습 vs. 딥러닝

- 규칙 기반 방법
  - 분류하는 규칙을 사람이 구현. 예)“꽃잎의 길이가 a보다 크고, 꽃잎의 너비가 b보다 작으면 Setosa”라는 규칙에서 a와 b를 사람이 결정해 줌
  - 큰 데이터셋에서는 불가능하고, 데이터가 바뀌면 처음부터 새로 작업해야 하는 비효율성
- 기계 학습 방법 (수업에서 다룸)
  - 특징 벡터를 추출하고 레이블을 붙이는 과정은 규칙 기반과 동일(수작업 특징hand-crafted feature)
  - 규칙 만드는 일은 기계학습 모델을 이용하여 자동으로 수행
- 딥러닝 방법
  - 레이블을 붙이는 과정은 기계 학습과 동일
  - 특징 벡터를 학습이 자동으로 알아냄. 특징 학습feature learning 또는 표현 학습representation learning을 한다고 말함

분류

Classifier

```
from sklearn.tree import DecisionTreeClassifier #Decision Tree
models = []
models.append(("LR", LogisticRegression()))
models.append(("DT", DecisionTreeClassifier()))
from sklearn.metrics import accuracy_score

# 모델 학습 및 정확도 분석
for name, model in models:
    model.fit(x_data, y_data.values.ravel())
    y_pred = model.predict(x_data)
    print(name, "'s Accuracy is ", accuracy_score(y_data, y_pred))
```

분류

Classifier

```
from sklearn import model_selection
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=7, shuffle=True)
    cv_results = model_selection.cross_val_score(model, x_data, y_data.values.ravel(), cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
fig = plt.figure()
fig.suptitle('Classifier Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

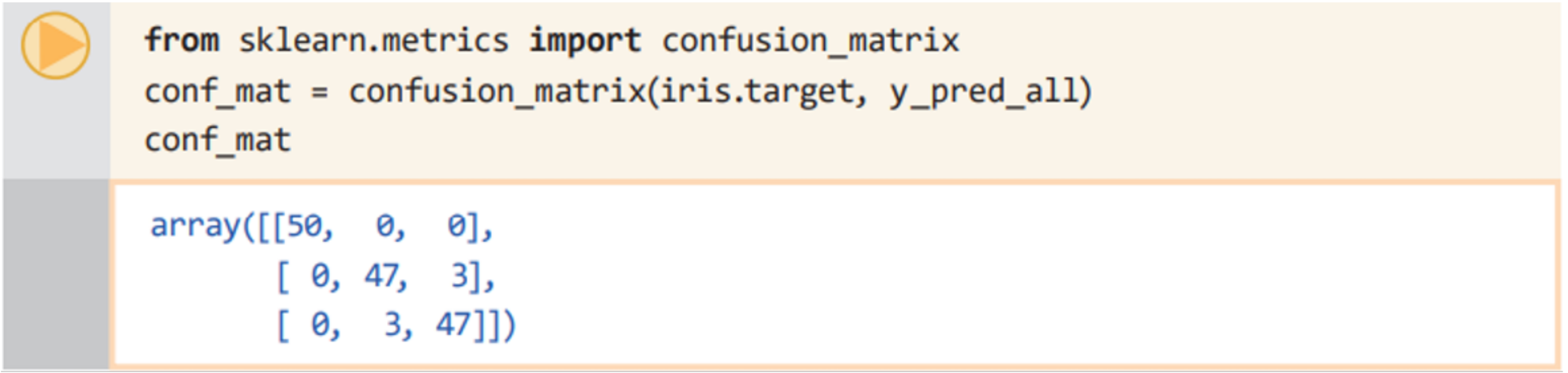
분류
성능측정

- 객관적인 성능 측정의 중요성
  - 모델 선택할 때 중요
  - 현장 설치 여부 결정할 때 중요
- 일반화 generalization 능력
  - 학습에 사용하지 않았던 새로운 데이터에 대한 성능
  - 가장 확실한 방법은 실제 현장에 설치하고 성능 측정, 비용 때문에 실제 적용 어려움
  - 주어진 데이터를 분할하여 사용하는 지혜 필요

분류

혼동 행렬과 성능 측정 기준

- 시각화 결과를 혼동 행렬 confusion matrix 으로 판단할 수 있음



```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(iris.target, y_pred_all)
conf_mat
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  3, 47]])
```

분류

혼동 행렬과 성능 측정 기준

		참값(그라운드 트루스)					
		부류 1	부류 2	…	부류 j	…	부류 c
예측한 부류	부류 1	n_{11}	n_{12}		n_{1j}		n_{1c}
	부류 2	n_{21}	n_{22}		n_{2j}		n_{2c}
	…						
	부류 i	n_{i1}	n_{i2}		n_{ij}		n_{ic}
	…						
	부류 c	n_{c1}	n_{c2}		n_{cj}		n_{cc}

(a) 부류가 c개인 경우

		그라운드 트루스	
		긍정	부정
예측값	긍정	TP	FP
	부정	FN	TN

(b) 부류가 2개인 경우

- 혼동 행렬confusion matrix
  - 분류 별로 옳은 분류와 틀린 분류의 개수를 기록한 행렬
  - n_{ij}는 모델이 i 라고 예측했는데 실제 분류는 j인 샘플의 개수
- 이진 분류에서 긍정positive과 부정negative
  - 검출하고자 하는 것이 긍정(환자가 긍정이고 정상인이 부정, 불량품이 긍정이고 정상이 부정)
  - 참 긍정(TP), 거짓 부정(FN), 거짓 긍정(FP), 참 부정(TN)의 네 경우

분류

혼동 행렬과 성능 측정 기준

- 성능 측정 기준
  - 정확률accuracy
    - $$\text{정확률} = \frac{\text{맞힌 샘플 수}}{\text{전체 샘플 수}} = \frac{\text{대각선 샘플 수}}{\text{전체 샘플 수}} \qquad (3.2)$$
  - 정밀도 precision와 재현률 recall
    - $$\text{정밀도} = \frac{TP}{TP+FP}, \ \text{재현율} = \frac{TP}{TP+FN} \qquad (3.4)$$

```
[[73.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 66.  0.  0.  0.  0.  0.  0.  4.  0.]
 [ 0.  0. 76.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 66.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 74.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 74.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. 66.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 76.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 74.  1.]
 [ 0.  0.  0.  0.  1.  1.  0.  1.  0. 65.]]
테스트 집합에 대한 정확률은 98.74826147426981 %입니다
```