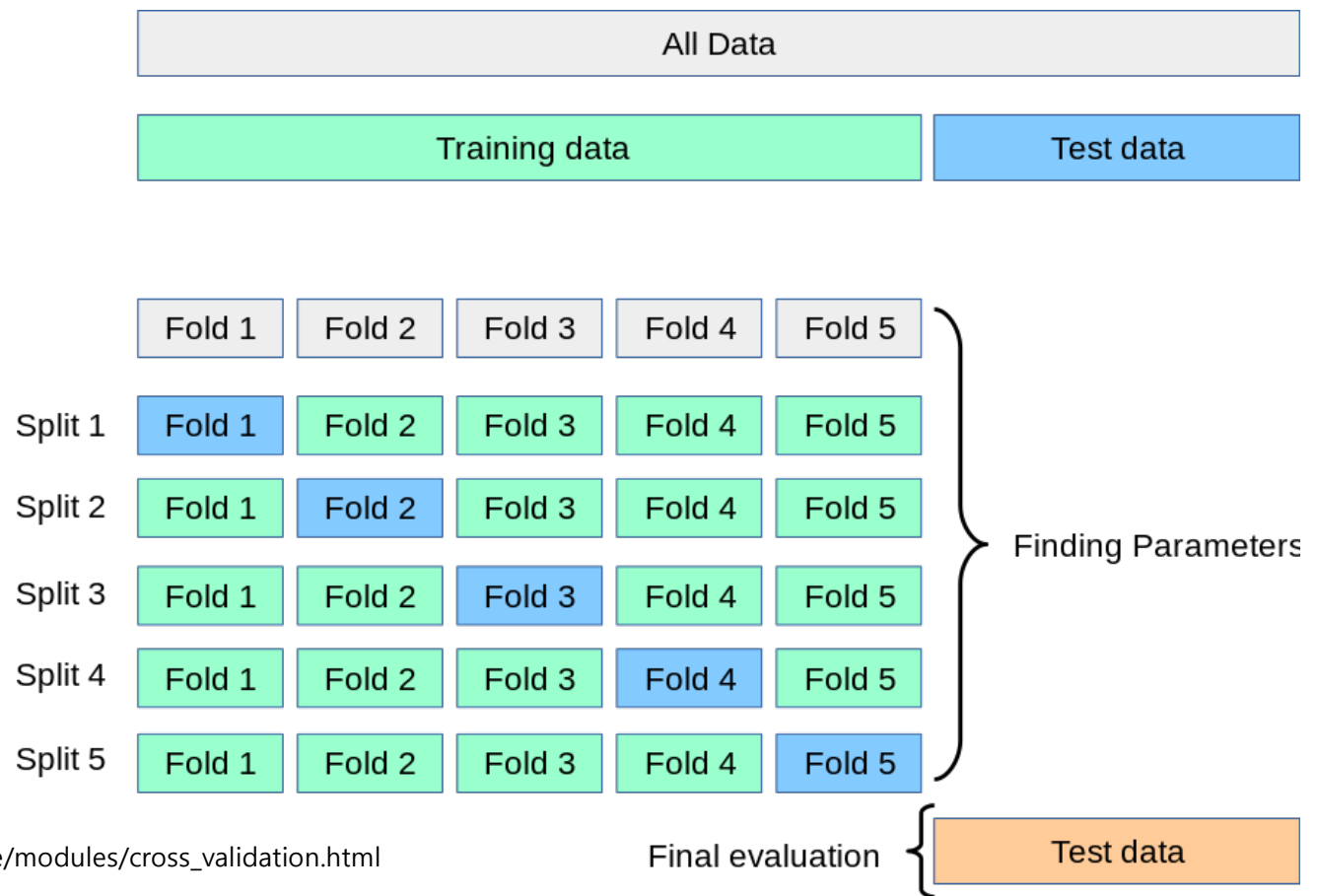


| 모델 평가방법과 오버피팅



명지대학교
MYONGJI UNIVERSITY

Cross Validation (교차검증)

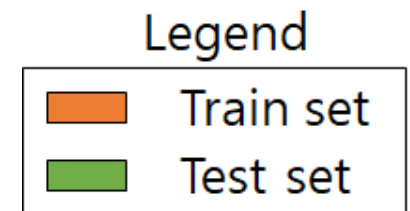
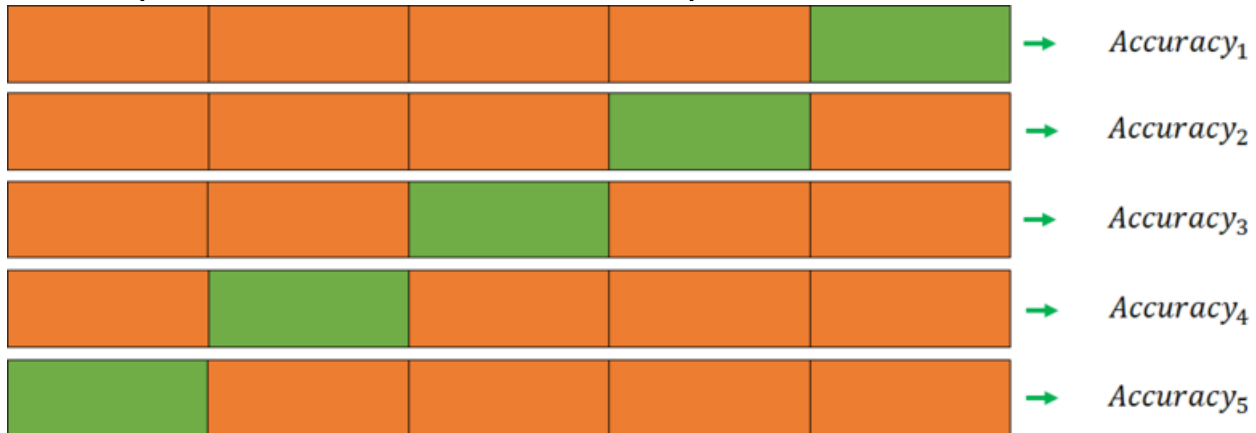


Cross Validation (교차검증)

홀드아웃 방법(Holdout method)



k-겹 교차 검증(k-fold cross validation)

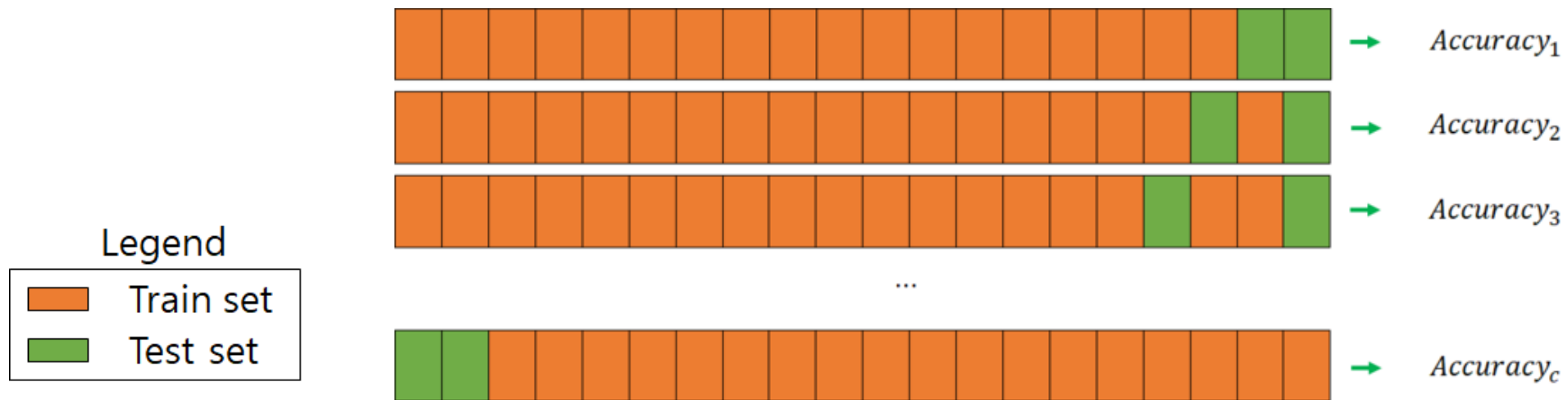


$$Accuracy = Average(Accuracy_1, \dots, Accuracy_k)$$

Cross Validation (교차검증)

리브-p-아웃 교차 검증(Leave-p-out cross validation)

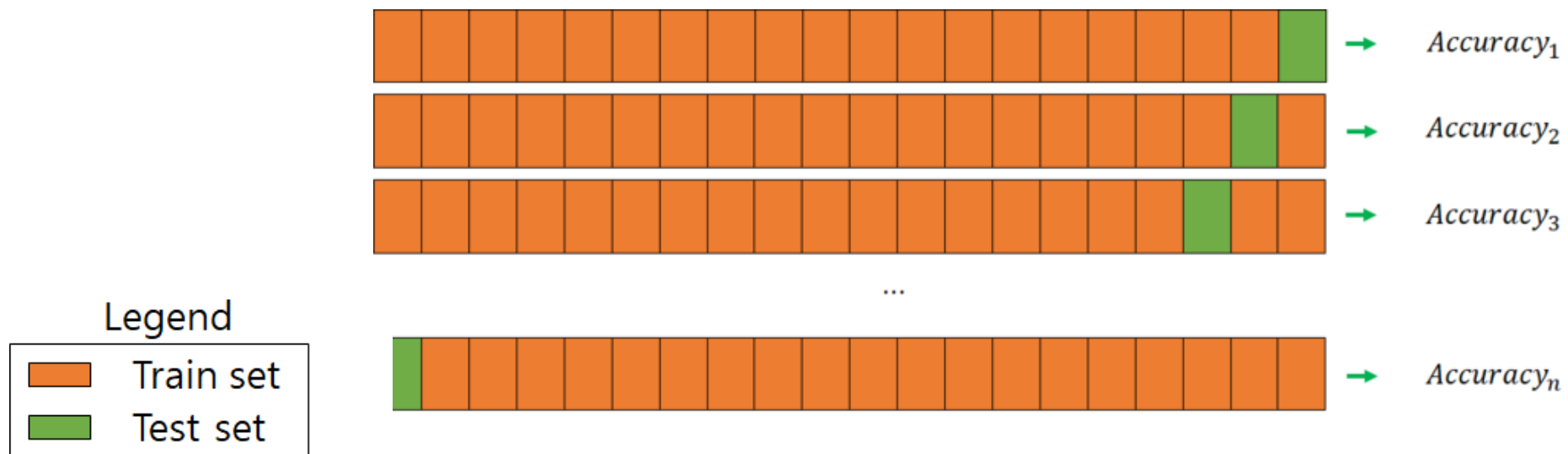
- 전체 데이터(서로 다른 데이터 샘플들) 중에서 p 개의 샘플을 선택하여 그것을 모델 검증에 사용하는 방법



$Accuracy = Average(Accuracy_1, \dots, Accuracy_c),$
where $c = \# \text{ of combinations of data sample}$

Cross Validation (교차검증)

리브-원-아웃 교차 검증(Leave-one-out cross validation)



$$Accuracy = Average(Accuracy_1, \dots, Accuracy_n), \quad \text{where } n = \# \text{ of data sample}$$

정확도 (Accuracy)

정확도

■ $390 + 490 / 1000 = 0.88$

	음성(예측)	양성(예측)
음성(실제)	390	110
양성(실제)	10	490

음성 400

양성 600

$900 / 1000 = 0.9$

	음성(예측)	양성(예측)
음성(실제)	900	0
양성(실제)	100	0

음성 1000

양성 0

정확도가 높으면 항상 좋은 모델?

실제로 좋은
모델????

양성을 예측 하지 못함

정확도 (Accuracy)

- 불균형한(Imbalanced) 데이터에서는 Accuracy를 사용할 경우 잘못된 성능 예측 결과
- 모델 성능 평가를 위해
 - Accuracy, Precision, Recall과 이를 이용한 F1 Score 등을 주로 사용

Confusion matrix

분류 결과		실제 정답	
		True	False
		True	False
	True	True Positive	False Positive
	False	False Negative	True Negative

Confusion matrix- Precision

TP/TP+FP

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

Confusion matrix- Recall

TP/TP+FN

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

Confusion matrix

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

label = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2]
predict = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 0]

print(confusion_matrix(label, predict))
```

- 0번 클래스의 경우 11개 중 11개를 0으로, 0개를 1로, 0개를 2로 예측하였습니다.
- 1번 클래스의 경우 4개 중 3개를 0으로, 1개를 1로, 0개를 2로 예측하였습니다.
- 2번 클래스의 경우 1개를 0으로, 1개를 1로, 2개를 2로 예측하였습니다

Confusion matrix

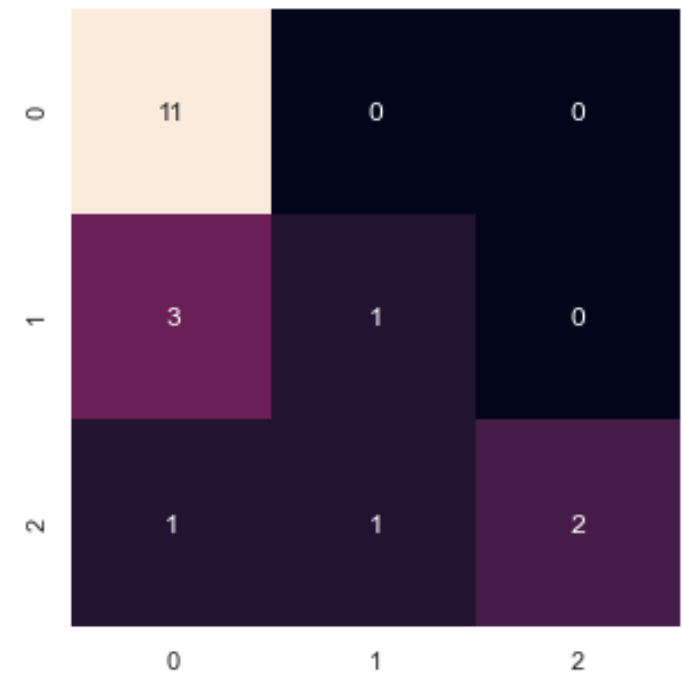
```
import seaborn as sns
```

```
label = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2]  
predict = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2,  
0]
```

```
cmatrix = confusion_matrix(label, predict)
```

```
sns.set(rc={'figure.figsize':(6,5)})
```

```
sns.heatmap(data=cmatrix, annot=True)
```



선형회귀모델

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

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

```
# 당뇨병 데이터셋 가져오기
```

```
diabetes = datasets.load_diabetes()
```

```
diabetes_X = diabetes.data[:, np.newaxis, 2]
```

```
# 배열 크기
```

```
diabetes_X.shape
```

```
# 훈련 세트, 테스트 세트 데이터 분리
```

```
diabetes_X_train = diabetes_X[:-20]
```

```
diabetes_X_test = diabetes_X[-20:]
```

```
# 훈련 세트, 테스트 세트 타겟값 분리
```

```
diabetes_y_train = diabetes.target[:-20]
```

```
diabetes_y_test = diabetes.target[-20:]
```

```
# 선형회귀 모델 생성
```

```
regr = LinearRegression()
```

```
# 훈련 세트를 사용해서 모델 훈련
```

```
regr.fit(diabetes_X_train, diabetes_y_train)
```

```
# scikit-learn은 훈련데이터에서 유도된 속성은 항상 끝에 밑줄을 붙입니다.
```

```
# regr.coef_, regr.intercept_ 처럼 밑줄을 붙임으로써 사용자가 지정한 변수와 구분할 수 있습니다.
```

```
# 계수(가중치)
```

```
print('Coefficients: ', regr.coef_)
```

```
i
```

```
# 편향(절편)
```

```
print('Intercept: ', regr.intercept_)
```

```
# 평균 제곱근 편차
```

```
print("Mean squared error: %.2f" % np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))
```

```
# 훈련데이터 성능
```

```
print('TrainSet score: %.2f' % regr.score(diabetes_X_train, diabetes_y_train))
```

```
# 테스트데이터 성능
```

```
print('TestSet score: %.2f' % regr.score(diabetes_X_test, diabetes_y_test))
```

```
# 도표 결과
```

```
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
```

```
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test), color='blue',  
         linewidth=3)
```

```
plt.xticks()
```

```
plt.yticks()
```

```
plt.show()
```

퍼셉트론

```
# Load required libraries
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Load the iris dataset
iris = datasets.load_iris()

# Create our X and y data
X = iris.data
y = iris.target

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

sc = StandardScaler() #StandardScaler를 X_train 데이터가 mean = 0, variance = 1로 만들 수 있도록 학습
sc.fit(X_train)

# Apply the scaler to the X training data
X_train_std = sc.transform(X_train)

# Apply the SAME scaler to the X test data
X_test_std = sc.transform(X_test)

# Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a learning rate of 0.1
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)

# Train the perceptron
ppn.fit(X_train_std, y_train)

y_pred = ppn.predict(X_test_std)

print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

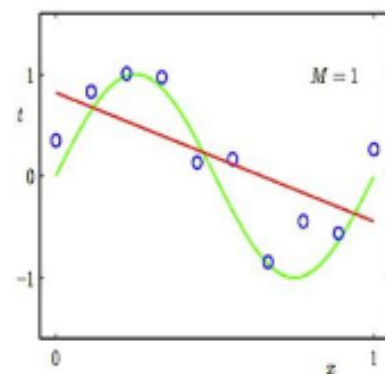
| 과적합



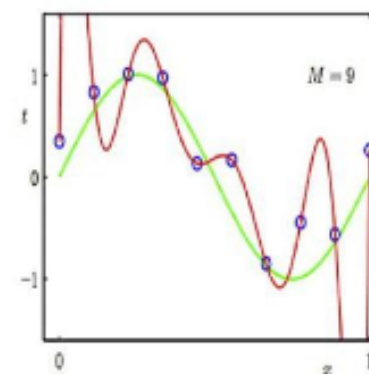
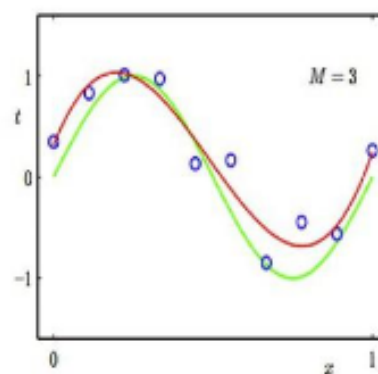
명지대학교
MYONGJI UNIVERSITY

과적합(Overfitting)

Regression:

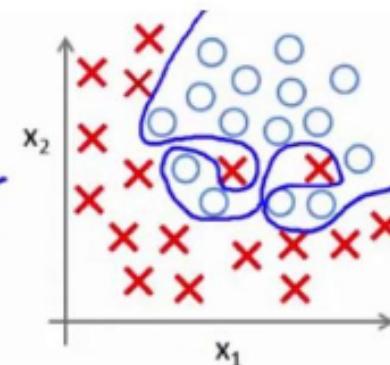
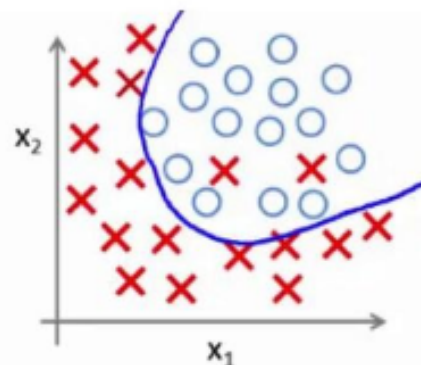
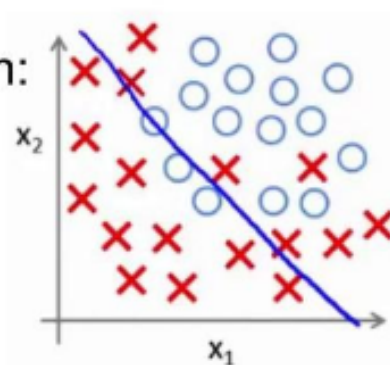


predictor too inflexible:
cannot capture pattern



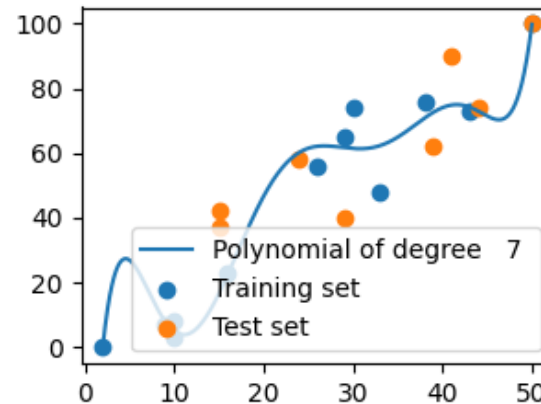
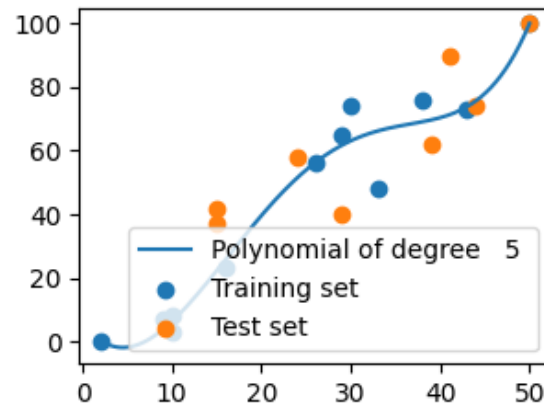
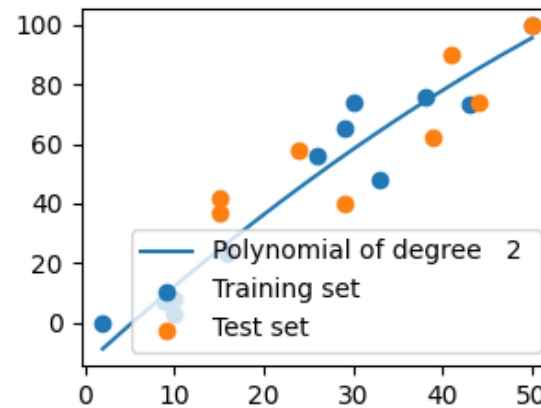
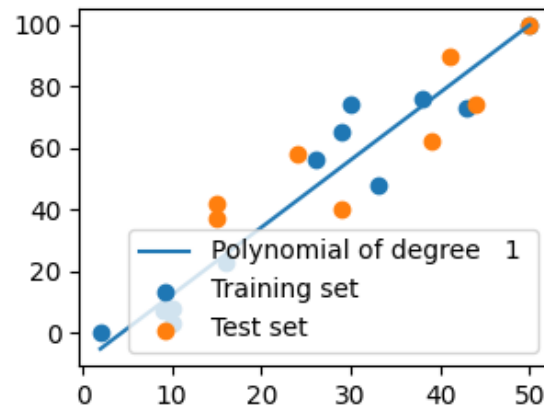
predictor too flexible:
fits noise in the data

Classification:



Copyright © 2014 Victor Lavrenko

과적합(Overfitting)



과적합(Overfitting)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import pandas as pd
#np.random.seed(0)
#x = np.arange(-1,1,0.1)
#y = -x**2 + np.random.normal(0,0.05,len(x))

students = {'hours': [29, 9, 10, 38, 16, 26, 50, 10, 30, 33, 43, 2, 39, 15, 44, 29, 41, 15,
24, 50],
            'test_results': [65, 7, 8, 76, 23, 56, 100, 3, 74, 48, 73, 0, 62, 37, 74, 40, 90, 4
2, 58, 100]}
student_data = pd.DataFrame(data=students)
x = student_data.hours
y = student_data.test_results
```

```
X_train = x[0:12]
y_train = y[0:12]
X_test = x[12:]
y_test = y[12:]

def polynomial_fit(degree = 1):
    return np.poly1d(np.polyfit(X_train,y_train,degree))

def plot_polyfit(degree=1):
    p = polynomial_fit(degree)
    plt.scatter(X_train, y_train, label="Training set")
    plt.scatter(X_test, y_test, label="Test set")
    curve_x = np.arange(min(x), max(x), 0.01)

    plt.plot(curve_x, p(curve_x), label="Polynomial of degree {}".format(degree))
    #plt.xlim((-1, 1))
    #plt.ylim((-1, np.max(y) + 0.1))

    print(r2_score(y, p(x)))


    plt.legend()
    plt.plot()

plt.subplot(2,2,1)
plot_polyfit(1)

plt.subplot(2,2,2)
plot_polyfit(2)

plt.subplot(2,2,3)
plot_polyfit(5)

plt.subplot(2,2,4)
plot_polyfit(7)
plt.show()
```



이미지 및 코드 출처

<https://github.com/ExcelsiorCJH/Hands-On-ML/>

<https://github.com/rickiepark/handson-ml2>