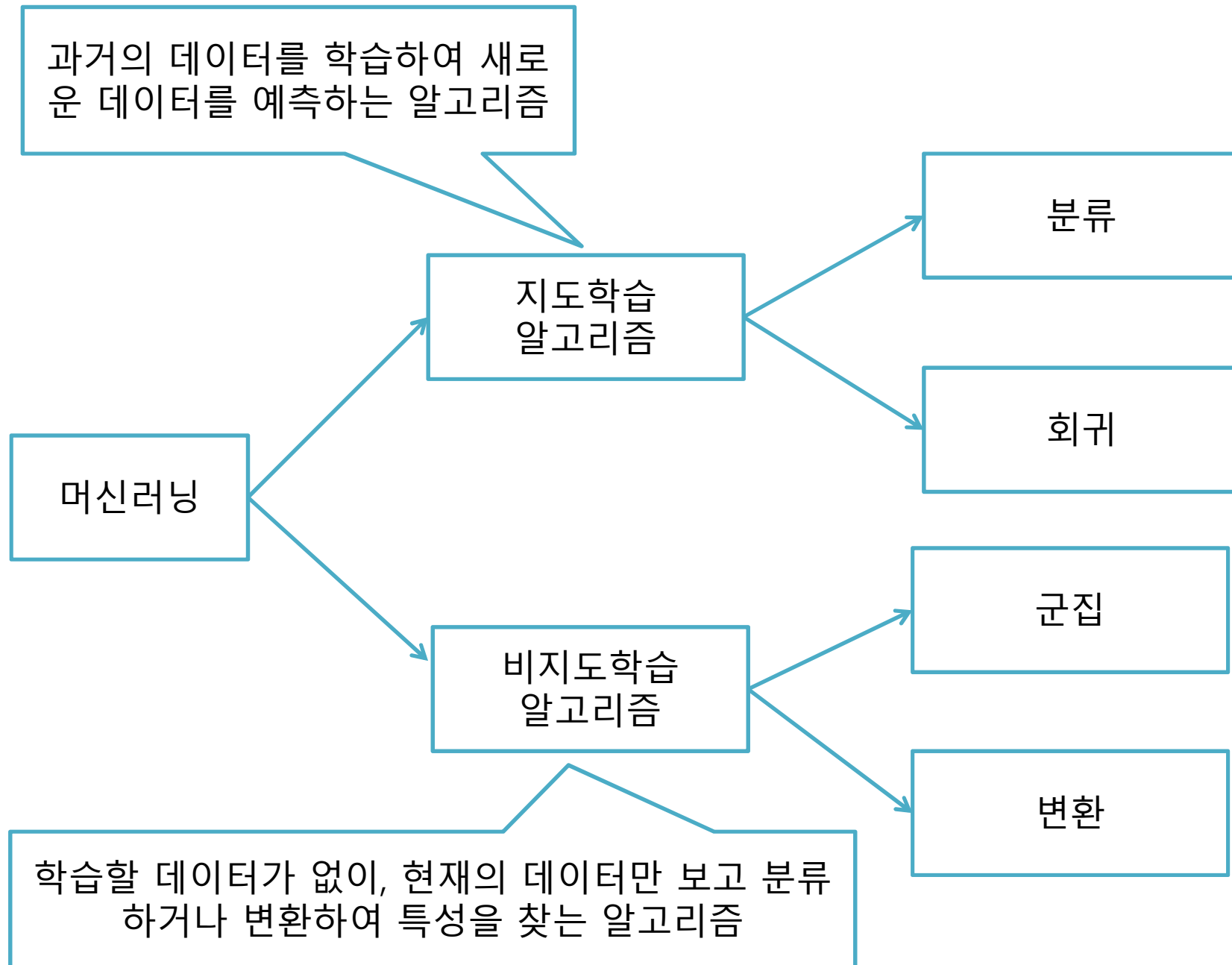


파이썬 머신러닝 1

2017010715 허지혜

목차

1. 머신러닝 종류
2. 과대적합
3. 일반화
- 4.
5. Knn
6. 코드 예시

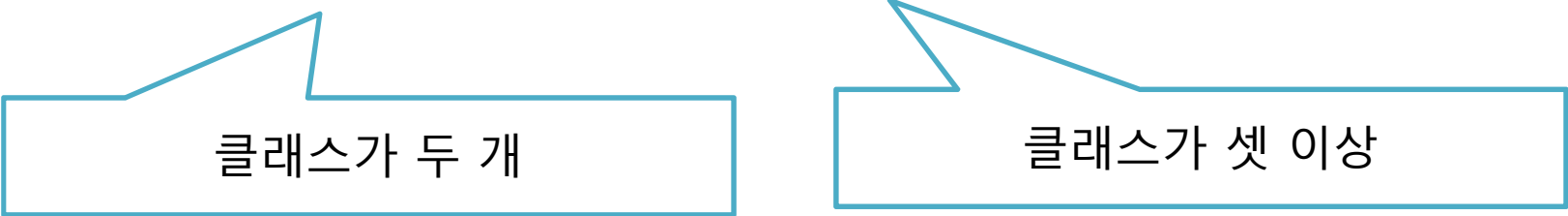


1-(1)지도학습의 종류

- 분류

가능성 있는 몇몇 후보 중 하나를 선정
(항목선택)

⇒이진 분류와 다중 분류로 나뉘짐



클래스가 두 개

클래스가 셋 이상

클래스 = 레이블 = 분류할 종류

- 회귀

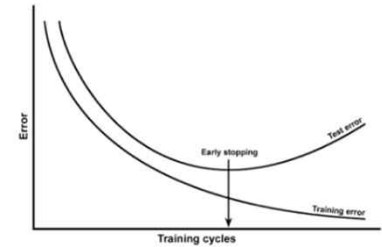
특정 숫자를 예측(값 예측)

2. 과대적합 문제

- 과대적합이란?

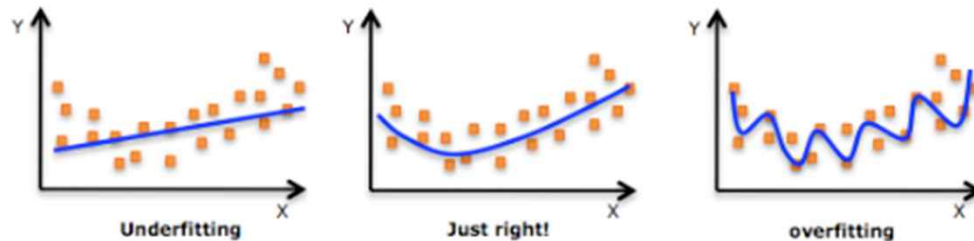
학습데이터를 과하게 잘 학습한 것

과하게 학습을 진행해 학습되지 않은 데이터가 들어오게 되면 분류를 하지 못하게 됨



< Fig. 1. 머신러닝에서의 과대적합 예시 >

학습 데이터에 대하여 오차가 감소하는데, 실제 데이터에 대해 오차가 증가하는 지점이 존재한다
갑자기 치솟는 부분이 과대적합이 발생했다고 볼 수 있다



과소적합은 반대로
규칙이 너무 단순하게
만들어진 모델이라고 한다

3. 일반화

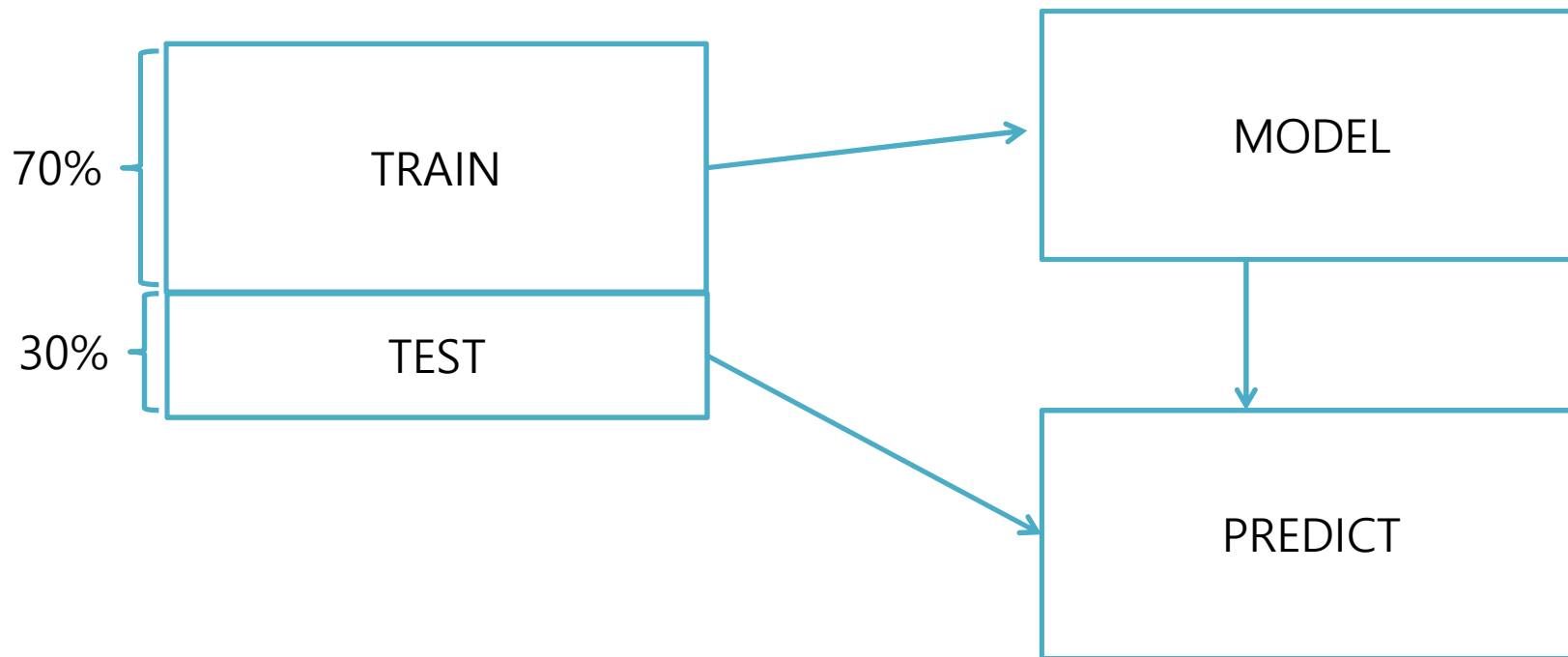
머신러닝에서 중요한 점은
데이터를 잘 맞추게 하는 것이다.
이런 모델을 일반화 되었다라고 한다.

샘플이 많고 다양성을 갖춘 경우 과대적합 없이
복잡한 모델을 만들 수 있다.

따라서 미래에 나타날 패턴과 같거나 유사한 샘플이 많이 있어야 새로운 데이터를 잘 맞춘다.

4. 훈련데이터와 테스트데이터

전체 데이터셋을 임의로 섞어 약 70% 만으로 모델을 만듦
이 모델이 30%의 테스트데이터를 잘 예측하면 모델이 일반화 되었다 라고 함
파이썬에는 전체 데이터셋을 훈련데이터와 테스트 데이터로 구분해주는
패키지가 있음



5. KNN

KNN(K-Nearest Neighbors)

알고리즘을 예측할 데이터와 가장 가까운 데이터를 찾아서 분류한다.

K는 참고할 이웃의 숫자(보통 홀수)

<장점>

이해하기 쉬운 알고리즘이다
결과가 쉽게 납득이 간다
조정할 옵션이 많이 필요 없어 간단

<단점>

계산 속도가 느린편
독립변수가 많으면 잘 작동안함
독립변수의 내용이 비슷한 값의 범위를
갖도록 전처리 해야함

6. 실습

In [7]: #데이터 로딩

```
import os
import pandas as pd
```

os.chdir("c:/pytest") 디렉토리 위치 변경하기

```
dataset = pd.read_csv('knn.csv')
```

```
print(dataset.head())
```

```
print(dataset.shape)
```

```
X=dataset.iloc[:,0:2]
```

```
Y=dataset.iloc[:,2]
```

```

      x1    x2    y
0  9.96  4.600  1.0
1 11.00 -0.168  0.0
2 11.50  5.210  1.0
3  8.69  1.540  0.0
4  8.11  4.290  0.0
(26, 3)

```

Operating system, 특정 디렉토리 내 파일 보

상위 행 가져오기

데이터의 행, 열 크기 확인하기

모든 행에 대하여 2번열 전까지만 가져와 X 변수에 넣기

모든 행에 대하여 2번열만 가져와 Y 변수에 넣기

	A	B	C	
1	x1	x2	y	
2	9.96E+00	4.60E+00	1.00E+00	
3	1.10E+01	-1.68E-01	0.00E+00	
4	1.15E+01	5.21E+00	1.00E+00	
5	8.69E+00	1.54E+00	0.00E+00	
6	8.11E+00	4.29E+00	0.00E+00	
7	8.31E+00	4.81E+00	1.00E+00	
8	1.19E+01	4.65E+00	1.00E+00	
9	9.67E+00	-2.03E-01	0.00E+00	
10	8.35E+00	5.13E+00	1.00E+00	
11	8.67E+00	4.48E+00	1.00E+00	
12	9.18E+00	5.09E+00	1.00E+00	
13	1.02E+01	2.46E+00	1.00E+00	
14	8.69E+00	1.49E+00	0.00E+00	
15	8.92E+00	-6.40E-01	0.00E+00	
16	9.29E+00	4.85E+00	1.00E+00	
17	9.26E+00	5.13E+00	1.00E+00	
18	8.90E+00	4.85E+00	1.00E+00	
19	8.18E+00	1.30E+00	0.00E+00	
20	8.73E+00	2.49E+00	0.00E+00	
21	9.32E+00	5.10E+00	1.00E+00	
22	1.01E+01	9.91E-01	0.00E+00	
23	9.50E+00	-2.64E-01	0.00E+00	
24	8.34E+00	1.64E+00	0.00E+00	
25	9.50E+00	1.94E+00	0.00E+00	
26	9.15E+00	5.50E+00	1.00E+00	
27	1.16E+01	1.34E+00	0.00E+00	
28				
29				

In [9]:

#데이터 확인

`print(X.head())`

`print(Y.head())`

`print(X.shape)`

`print(Y.shape)`

X에 대한 상위행

Y에 대한 상위행

X에 대한 2차원 배열, (26,2)

Y에 대한 일차원 배열, (26,)

	x1	x2
0	9.96	4.600
1	11.00	-0.168
2	11.50	5.210
3	8.69	1.540
4	8.11	4.290

Name: y, dtype: float64

(26, 2)

(26,)

파이썬에서 머신러닝을 할 수 있게 하는 패키지

```
In [10]: #데이터 분리
#훈련 데이터와 테스트 데이터로 분리하고 확인한다
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0)
print(X_train.head())
print(X_test.head())
```

	x1	x2
13	8.92	-0.640
18	8.73	2.490
19	9.32	5.100
16	8.00	4.850
1	11.00	-0.168
	x1	x2
2	11.50	5.210
20	10.10	0.991
14	9.49	4.330
17	8.18	1.300
5	8.31	4.810

여러 번 실행해도 똑같은 값을
얻기 위해 고정
난수 발생을 위한 seed의 인자값

훈련세트와 테스트세트로 분리

- 머신러닝은 훈련 과정과 테스트 과정 두가지로 분리된다.
- 훈련세트를 가지고 예측모델을 훈련시킨 다음에 테스트세트로 훈련 성과를 판단한다.
- sklearn.model_selection.train_test_split() 함수를 사용하면 편리하게 나눌 수 있다.
- train_test_split() 함수는 기본값으로 훈련세트를 75%, 테스트세트를 25% 로 나눈다.

```
sklearn.model_selection.train_test_split(array*, test_size, train_size, random_state,  
shuffle, stratify)[source]
```

X랑 Y를 무작위로 섞어서
70% => X_train, Y_train
30% => X_test, Y_test

```
In [12]: #모델 만들기
#knn 분류기 객체를 생성하고 훈련데이터로 모델을 학습시킨다.
#분류기 객체 만들기
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)  기본값은 3

#훈련데이터를 모델에 넣어 학습시키기
clf.fit(X_train,Y_train)
```

```
Out[12]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                             weights='uniform')
```

```
weights='uniform')
```

In [15]:

```
#데이터 예측
```

```
#테스트 데이터를 넣어 예측하고 정확도를 산출
```

```
result = clf.predict(X_test)  테스트 데이터를 넣어서 예측한 값을 결과 변수에 저장  
print(result)
```

```
print(clf.score(X_test,Y_test))
```

테스트 세트의 정확도 = 85.7%

```
print(round(clf.score(X_test,Y_test),3))
```

```
[1. 0. 1. 0. 1. 0. 0.]
```

```
0.8571428571428571
```

```
0.857
```

0.857

In [17]: # K 숫자를 변경해보기

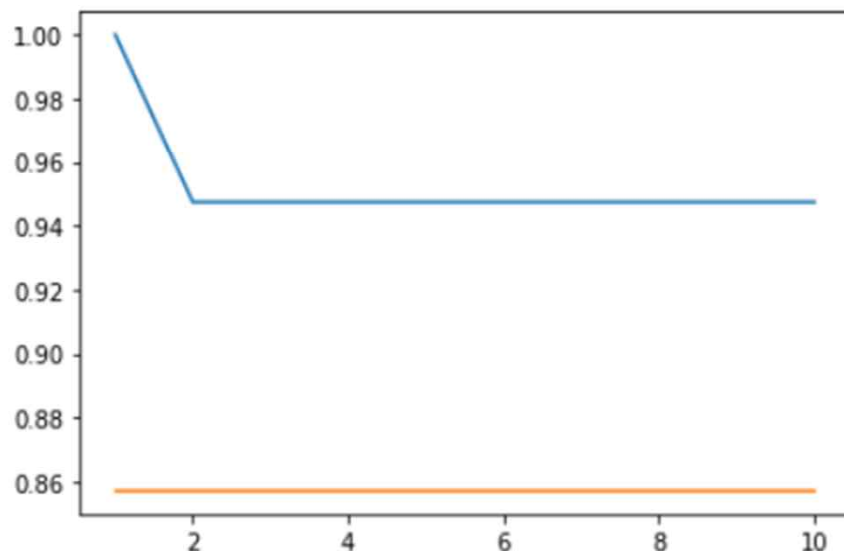
```
import matplotlib.pyplot as plt
train_accuracy = []
test_accuracy = []
n_neighbors_settings = range(1, 11)    1에서 10까지 n_neighbors에 적용

for n_neighbors in n_neighbors_settings :
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, Y_train)
    train_accuracy.append(clf.score(X_train, Y_train))
    test_accuracy.append(clf.score(X_test, Y_test))

plt.plot(n_neighbors_settings, train_accuracy, label="Traing Acuuracy")
plt.plot(n_neighbors_settings, test_accuracy, label="Test Acuuracy")
plt.show()
```

훈련 데이터 : 파랑

테스트 데이터 : 주황



결론

K=1일때 훈련데이터의 정확도가

100%인것은 과대적합

KNN=3 이상이 적당한 값이라고 봄

KNN 회귀

KNN은 숫자를 예측하는 회귀 상황에서도 쓰일 수 있다.

K개의 이웃이 가진 값의 평균을 구함

R² score

R square 점수는 회귀모델이 자료를 얼마나 잘 설명하는지를 말한다

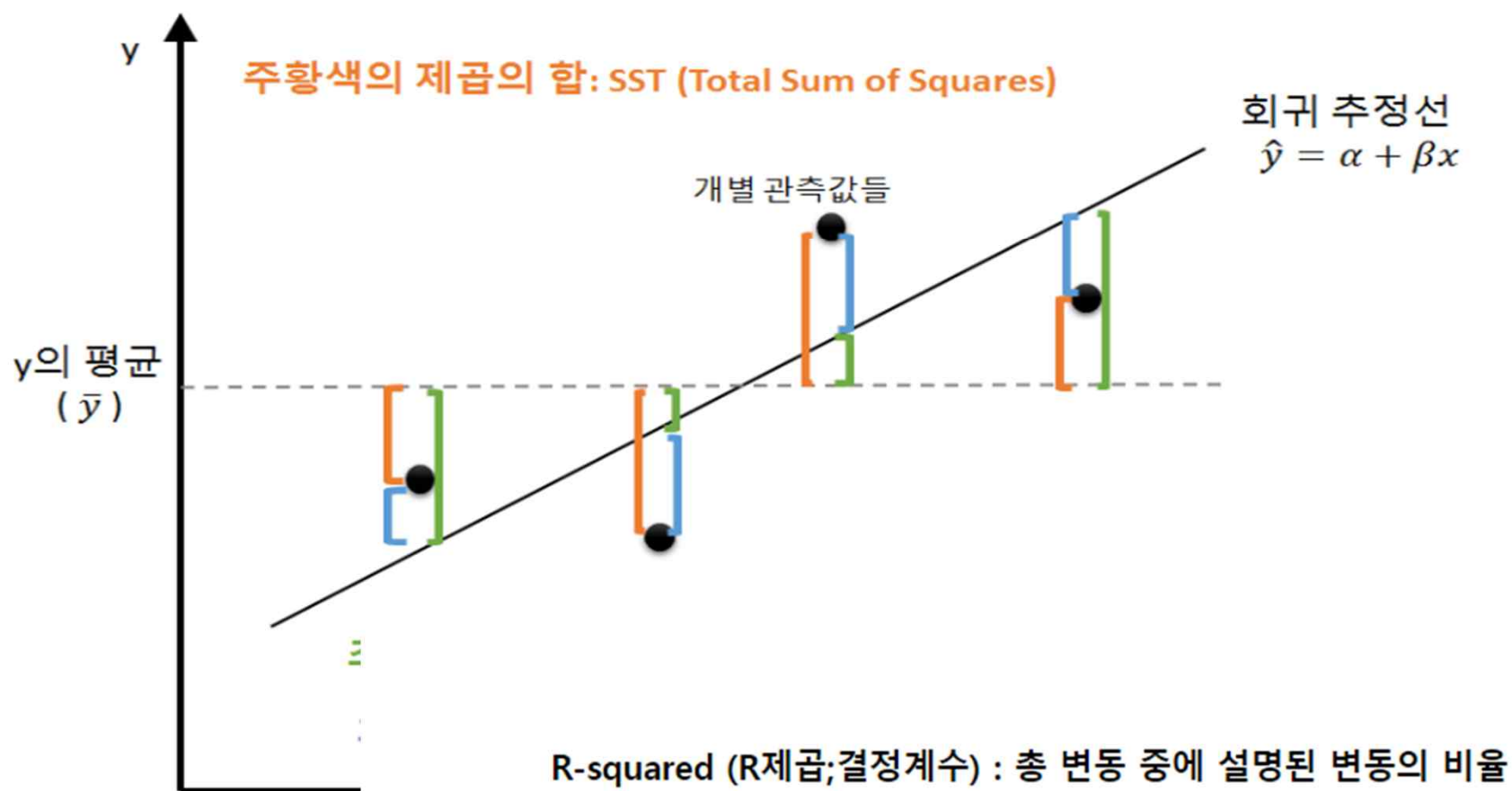
SST = 기본 차이값(분산)

총 변동 Total SS : total sum of squares (SST) : $SST = \sum_{i=1}^n (y_i - \bar{y})^2$: 개별 y의 편차제곱의 합

SSE = 예측의 차이값

설명된 변동 Model SS : explained sum of squares (SSE) : $SSE = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$: 회귀식 추정 y의 편차제곱의 합

SSR = SST - SSE



$$\frac{\sum (\hat{Y}_i - \bar{Y})^2}{\sum (Y_i - \bar{Y})^2} = r^2$$

$$R^2 \equiv \frac{R}{SST} = 1 - \frac{E}{SST}$$

끝