


ML_02. 회귀 모델

🕒 생성일	@2022년 6월 6일 오후 7:59
📁 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

내용구성

선형 회귀

- 핸즈온 : 4장 모델 훈련 파트 일부 내용 요약
- An Introduction to Statistical Learning (ISLR) : Chapter 3. Linear Regression

다항회귀

- 핸즈온 : 4장 일부 내용
- ISLR : Chapter 3

K-최근접 이웃 회귀 (KNN)

- 혼공 머신 코드
- ISLR : Chapter 3

로지스틱 회귀 & 소프트맥스 회귀

- 핸즈온 : 4장 일부 내용
- 로지스틱 회귀의 경우, 사실상 분류(Classification) 모델에 속하기 때문에 이론적으로 더 자세히 다음 문서에서 다룰 예정이다.

참고한 자료에 따라 내용이 섞여 있습니다.

핸즈온 4장 : 선형 회귀

일반적으로 선형 회귀 모델은 다음의 식에서처럼 입력 특성의 가중치 합과 편향이라는 상수를 더해 예측을 만든다.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- \hat{y} 은 예측값입니다.
- n 은 특성의 수입니다.
- x_i 는 i 번째 특성값입니다.
- θ_j 는 j 번째 모델 파라미터입니다(편향 θ_0 과 특성의 가중치 $\theta_1, \theta_2, \dots, \theta_n$ 을 포함합니다).

벡터 형태로 나타내면 다음과 같다.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- $\boldsymbol{\theta}$ 는 편향 θ_0 과 θ_1 에서 θ_n 까지의 특성 가중치를 담은 모델의 파라미터 벡터입니다.
- \mathbf{x} 는 x_0 에서 x_n 까지 담은 샘플의 **특성 벡터**입니다. x_0 는 항상 1입니다.¹
- $\boldsymbol{\theta} \cdot \mathbf{x}$ 는 벡터 $\boldsymbol{\theta}$ 와 \mathbf{x} 의 점곱입니다. 이는 $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ 와 같습니다.
- $h_{\boldsymbol{\theta}}$ 는 모델 파라미터 $\boldsymbol{\theta}$ 를 사용한 가설^{hypothesis} 함수입니다.

회귀에 사용되는 성능 측정 지표는 RMSE(root-mean square error), MAE(mean-absolute error) , MSE(mean square error) 가 있으며 MSE 를 주로 사용한다. MSE 를 위의 벡터 모델에 적용한 수식은 다음과 같다.

$$\text{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^m (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

사이킷런에서 선형 회귀를 수행하는 것은 간단하다.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X,y)

# 가중치와 편향 각각은 모델 속성에 저장되어 있다.
# lin_reg.coef_, lin_reg.intercept_
```

ISLR : Chapter 3.

About RSS : residual sum of squares

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

$RSS = e_1^2 + e_2^2 + \dots + e_n^2$, 이 수식은 잔차들의 제곱의 합을 의미한다. RSS 를 위의 y 예측 수식으로 풀면 다음과 같다.

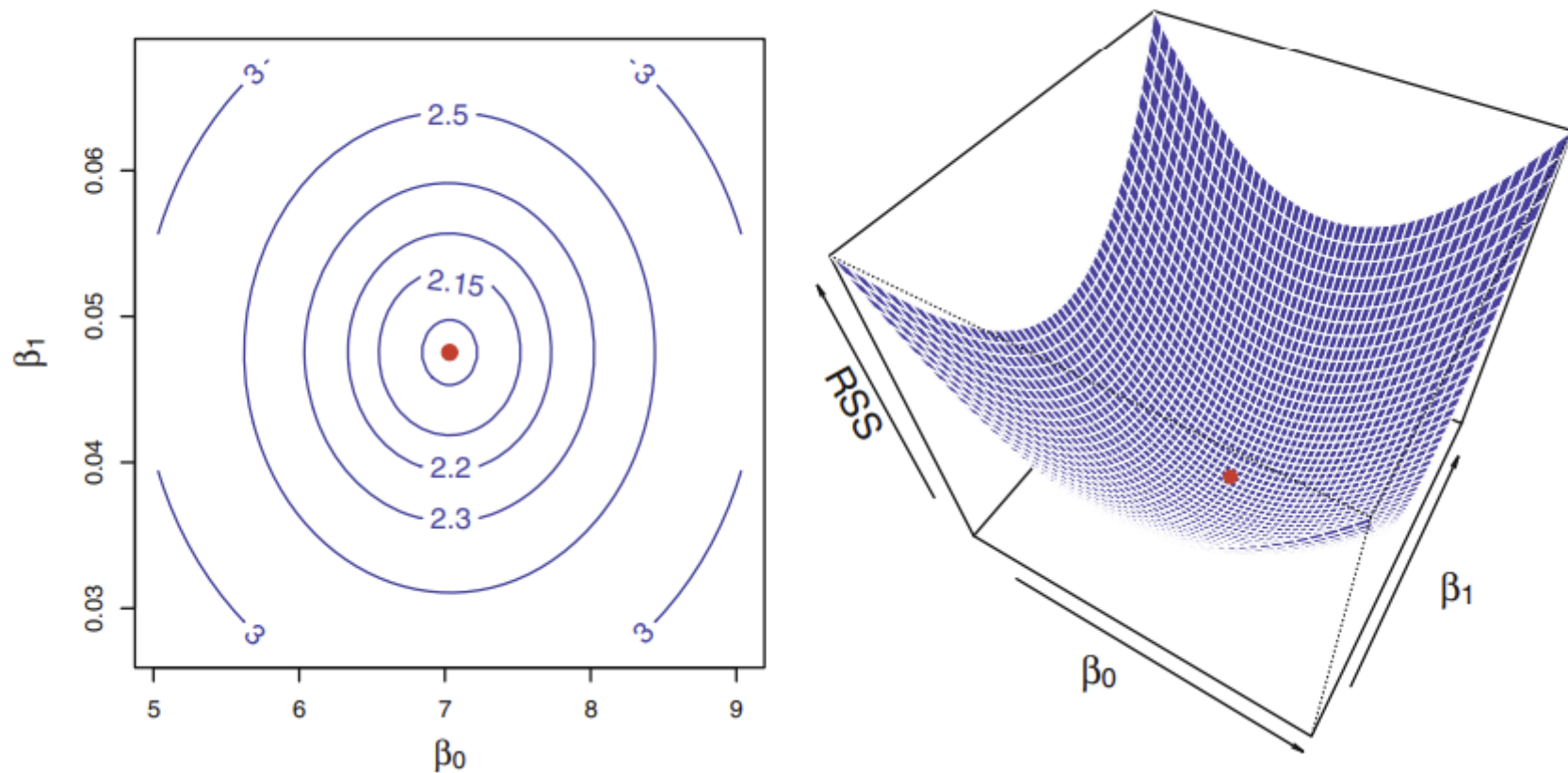
$$RSS = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \dots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2.$$

least squares approach (최소자승법) 에 의해 도출되는 각 계수의 값은 다음과 같다.(계산 과정에서 각 요소에 대한 편미분 방식 사용.)

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2},$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

그래프로 시각화 하면 다음 이미지와 같다. 붉은 점이 global minimum 을 의미한다.



least squares approach 방식에 따라 도출된 β_0 추정값과 β_1 추정값은 샘플 평균을 사용한다. 이것이 의미하는 것은, 선형 회귀에서 우리가 모르는 값인 β_0, β_1 을 추정할 때, 모회귀 모형에서의 값으로 정의하는 것이 합리적이라는 것이다.

예를 들어 수 없이 많은 데이터들의 평균을 모를 때, 우리는 일부 n 개의 샘플만을 표본으로 뽑아 평균을 추출하고 그것을 모평균으로 추정하며 합리적인 방식으로 생각한다. 이와 유사한 방식이지만, 반대로, 일부 샘플의 평균을 모를 때 모평균으로 추정하는 것이다.

Assessing the accuracy of the model

선형 회귀 모델에 대한 정량적인 평가는 두 개의 수치를 이용한다. 'residual standard error; RSE' & R^2 계수. RSE 는 다음과 같이 계산한다.

$$RSE = \sqrt{\frac{1}{n-2}RSS} = \sqrt{\frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2}. \quad (3.15)$$

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

RSE 에서 (n-2) 는 자유도를 의미한다.

RSE는 주어진 데이터에 대해 모델이 얼마나 부정확한지 보여주는 지표이다.

앞선 RSE 가 absolute measure of lack of fit of the model 이라면, R^2 계수는 alternative measure of fit 을 의미한다. 분산의 비율의 형태로 나타나는데, 수식은 다음과 같다.

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} \quad (3.17)$$

$$TSS = \sum (y_i - \bar{y})^2$$

RSS 실제값과 추정값 사이의 error, 즉 ‘잔차’ 의 제곱 합이고, TSS 는 실제값과 모평균 사이의 error, 즉 ‘오차’ 의 제곱 합을 의미한다.

R^2 계수는 X를 사용해 설명되는 Y 의 분산 정도를 비율로 나타낸다고 할 수 있다. (원문에서는 R^2 measures the proportion of variability in Y that can be explained using X. 라고 설명.)

헨즈온 : 다항 회귀(polynomial regression)

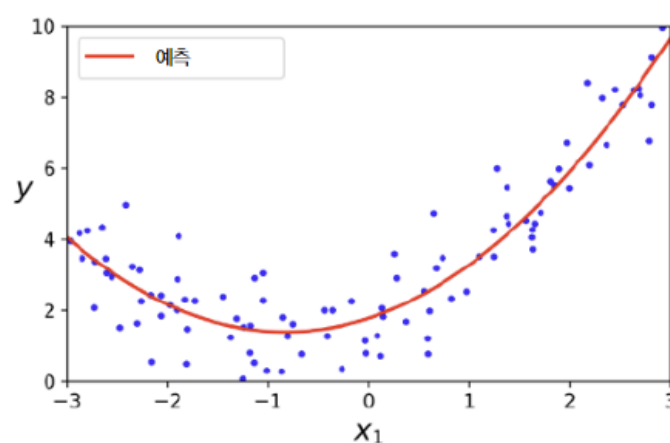
다항 회귀는 각 특성의 거듭제곱을 새로운 특성으로 추가하고, 이 확장된 특성을 포함한 데이터셋에 선형 모델을 훈련시키는 방식이다. 즉, 비선형 데이터를 학습하는데 선형 모델을 사용하는 것이다.

다항 회귀 모델은 특별한 모델 식이 있는 것이 아니라 차수가 늘어난 데이터를 활용하는 방식이다.

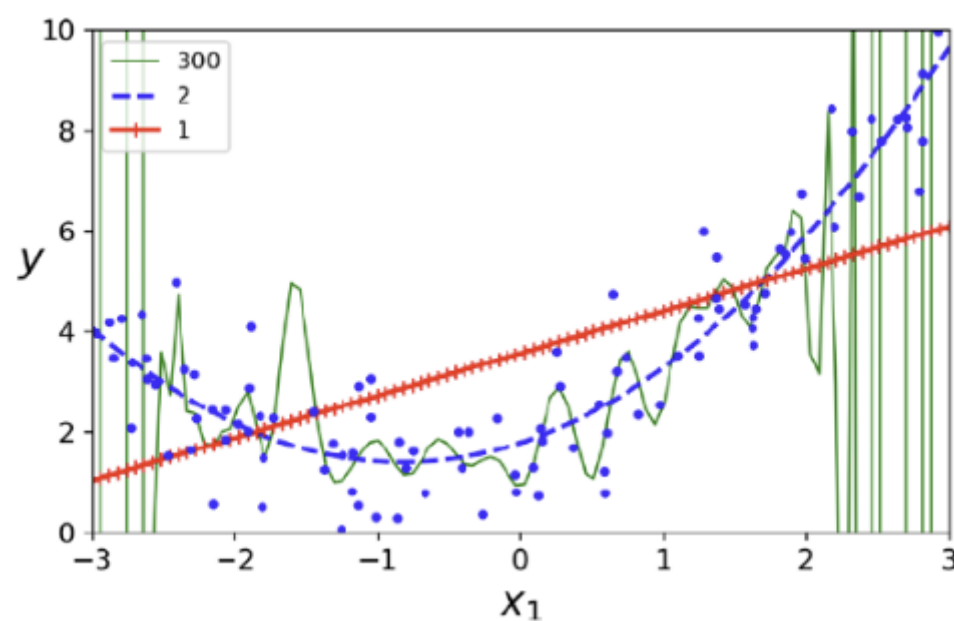
```
# 데이터의 차수를 2차로 높이는 과정
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
# X_poly는 이제 원래 특성 x와 이 특성의 제곱을 포함한다.

# 선형 회귀 모델 적용
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```

prediction 그래프를 그려보면 다음과 같을 것이다.



다음의 그림은 고차 다항 회귀 모델이 1차, 2차, 300차 일 때, 비교를 위한 prediction 그래프이다.



차수가 높다고 좋은 모델이 아니라, 오히려 예측을 더 못하게 된다. 위 그림에서 2차 모델이 데이터를 가장 잘 대표하고 있으며 예측값도 일반화할 만하다. 이러한 성능 평가를 위해 학습 곡선을 살펴보는 것이 모델 평가에 큰 도움이 된다.

학습 곡선

이 그래프는 훈련 세트와 검증 세트의 모델 성능을 훈련 세트 크기 또는 훈련 반복의 함수로 나타낸다. 다음 코드는 `X_train` 크기만큼 훈련을 진행하면서 훈련 세트에 대한 모델 성능과 검증 세트에 대한 모델 성능을 기록한다.

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model, X, y):
    # split dataset train / valid
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=10)

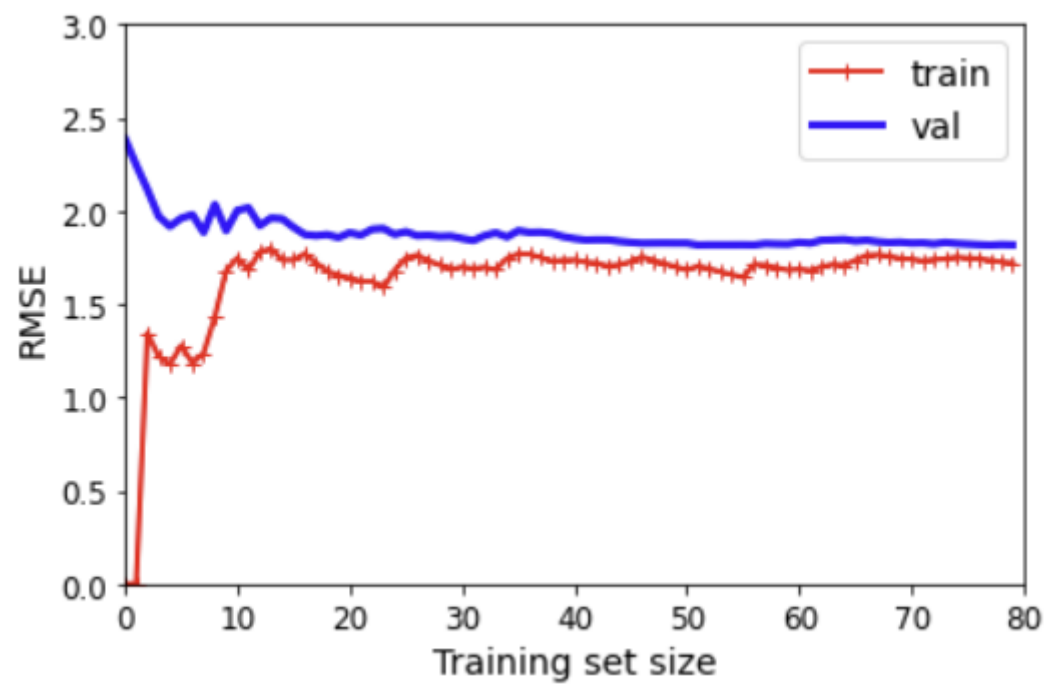
    train_errors, val_errors = [], []
    for m in range(1, len(X_train) + 1):
        # 훈련 데이터 크기만큼 훈련 진행
        # 입력으로 들어가는 데이터는 1씩 점차 증가
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])

        # valid dataset predict (훈련시키지 않고 바로 예측.)
        y_val_predict = model.predict(X_val)

        # mse 측정 및 리스트 업데이트
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))

    # 모델 훈련 후, plt.show() 로 그래프 나타내기
    lin_reg = LinearRegression()
    plot_learning_curves(lin_reg, X, y)
    plt.show()
```

위 코드의 실행 결과는 아래 이미지와 같다.

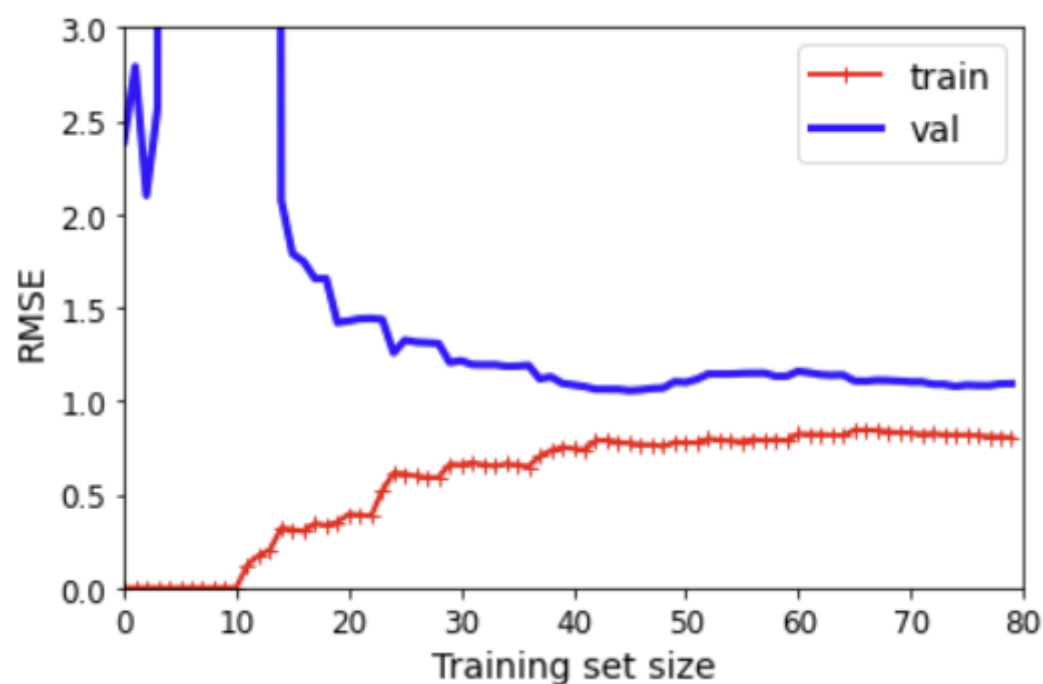


위 그래프를 분석하면, 두 곡선이 수평한 구간을 만들고 꽤 높은 오차에서 매우 가까이 근접해 있는 것으로 보인다. 이러한 경우는 과소적합 모델의 전형적인 모습이다.

10차 다항 함수의 학습 곡선을 그려보면 어떨까? 다음 코드는 10차 데이터가 선형 회귀 모델에 학습되고 학습 곡선 그래프를 만들어낸다.

```
from sklearn.pipeline import Pipeline
# Pipeline 클래스는 객체를 순서에 맞게 묶어준다.
polynomial_regression = Pipeline([
    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
    ("lin_reg", LinearRegression()),
])

plot_learning_curves(polynomial_regression, X, y)
plt.show()
```



위 그래프를 분석하면, 일단 훈련 데이터의 오차가 선형 회귀 모델보다 낮으며, valid dataset 보다 train dataset 훈련에서 오차가 약 0.5 정도 낮게 일관되게 측정된다. 이는 모델이 train dataset 에 과대적합 되었음을 의미한다.

ISLR : Chapter 3 다항 선형회귀

단순 선형 회귀 모델을 각 독립변수 별로 생성하고 결과를 종합적으로 해석하는 방식은 만족스럽지 않다. 첫 번째 이유는, 각 독립변수에 의한 결과를 하나로 합치는 방식이 불분명하기 때문이다. 두 번째로, 각각의 단순 선형 회귀 모델들은 서로의 독립변수에 의한 상관관계가 결과에 미

치는 영향을 무시하기 때문이다. 따라서 다음의 방식을 제안한다. (Suppose that we have p distinct predictors.)

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon, \quad (3.19)$$

당연히 각각의 계수들에 대한 적합한 추정값을 찾는 것이 목적이 된다.

이 때도 RSS 를 사용할 수 있다.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p. \quad (3.21)$$

$$\begin{aligned} \text{RSS} &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2. \end{aligned} \quad (3.22)$$

몇 가지 중요한 질문들

다항 선형 회귀를 적용하려 할 때, 생각해야 할 몇 가지 질문이 있다. (원문에서는 다음과 같이 제시된다.)

1. Is at least one of the predictors X_1, X_2, \dots, X_p useful in predicting the response?
2. Do all the predictors help to explain Y , or is only a subset of the predictors useful?
3. How well does the model fit the data?
4. Given a set of predictor values, what response value should we predict, and how accurate is our prediction?
- 5.

혼공머신 : k-최근접 이웃 회귀

회귀는 클래스 중 하나로 분류하는 것이 아니라 임의의 어떤 숫자를 예측하는 문제이다. k-최근접 이웃 회귀는 예측하려는 샘플에 가장 가까운 샘플 k 개를 선택한다. 샘플의 타깃은 어떤 클래스가 아니라 임의의 수치이며, 수치들의 평균을 구하여 샘플의 타깃을 예측할 수 있다.

아래는 농어(perch) 데이터를 임의의 생성해서 k-최근접 이웃 회귀 모델에 훈련시키는 일련의 과정이다.

```
import numpy as np
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
    21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
    23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
    27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
    39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5,
    44.0])
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
    115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
```



```

150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
1000.0])

# 훈련 세트와 테스트 세트 분리
# 디폴트로 0.8:0.2 로 분리된다.
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    perch_length, perch_weight, random_state=42)

'''
현재 훈련 데이터는 1차원 배열이므로 훈련에 들어가기 전 2차원 배열의
column 벡터로 변환한다. reshape(-1, 1) 로 지정해서 열기준의 shape 조성.
-1 은 열 기준 shape 조성 시 나머지 원소로 행을 채우라는 의미이다.
'''

train_input = train_input.reshape(-1, 1)
test_input = test_input.reshape(-1, 1)

# 모델 객체 생성 및 fit
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor()
knr.fit(train_input, train_target)

# 성능 평가 - MAE(평균 절댓값 오차 사용)
from sklearn.metrics import mean_absolute_error
test_prediction = knr.predict(test_input)
mae = mean_absolute_error(test_target, test_prediction)
print(mae)

```

ISLR : Chapter 3. K-최근접 이웃 알고리즘

Comparison of Linear Regression with K-Nearest Neighbors

선형 회귀는 파라메트릭 접근방식이며 이 방식은 모델을 데이터에 fit 하는 것이 편하다는 장점이 있다. 하지만 단점은, 실제 데이터를 나타내는 $f(x)$ 에 대해 강한 추정을 만든다는 것이다. 실제 $f(x)$ 에서 X 와 Y 가 선형 관계가 아니라면, 선형 회귀는 데이터에 대해 'poor fit' 하게 된다.

non-parametric 방식은 $f(x)$ 에 대한 명시적인 form을 가지지 않기 때문에 regression을 수행하는데 좀 더 유연한 접근법이라고 할 수 있다. 단 순하면서도 잘 알려진 non-parametric 방식에 K-최근접 이웃 회귀(K-nearest neighbors regresssion; KNN regression) 이 있다.

k- 최근접 이웃 회귀는 KNN 분류와 상당히 유사하다. k 값과, 예측 포인트 x_0 가 주어졌을 때, KNN 회귀는 우선 x_0 와 가까운 K 개의 이웃을 식별하며 각 이웃에 대한 특성값을 확인한다. 이 특성값들의 평균을 이용해 x_0 의 값을 수치로서 예측한다.

이 때 적절한 k 값을 찾는 것에 중요하며 값은 bias-variance tradeoff 에 의존한다. k 값이 작다면 flexible 하며 편향은 낮지만, 분산이 크게 된다. 이에 대한 이유를 원문에서는 다음과 같이 설명한다.

This variance is due to the fact that the prediction in a given region is entirely dependent on just one observation.”

대조적으로, k값이 크면, 편향은 크게 나타날 수 있어도 분산은 작게 된다. k 값을 크게 잡았을 때 데이터를 나타내는 $f(x)$ 의 구조적인 특징을 못 알아챌 수 도 있게 된다.

Larger values of K provided a smoother and less variable, and the smoothing may cause bias by masking some of the structure in $f(x)$.

적당한 k 값을 구하는 방법은 test error rates 를 추정하는 방식이 있는데 이것은 나중에 설명하도록 하겠다.

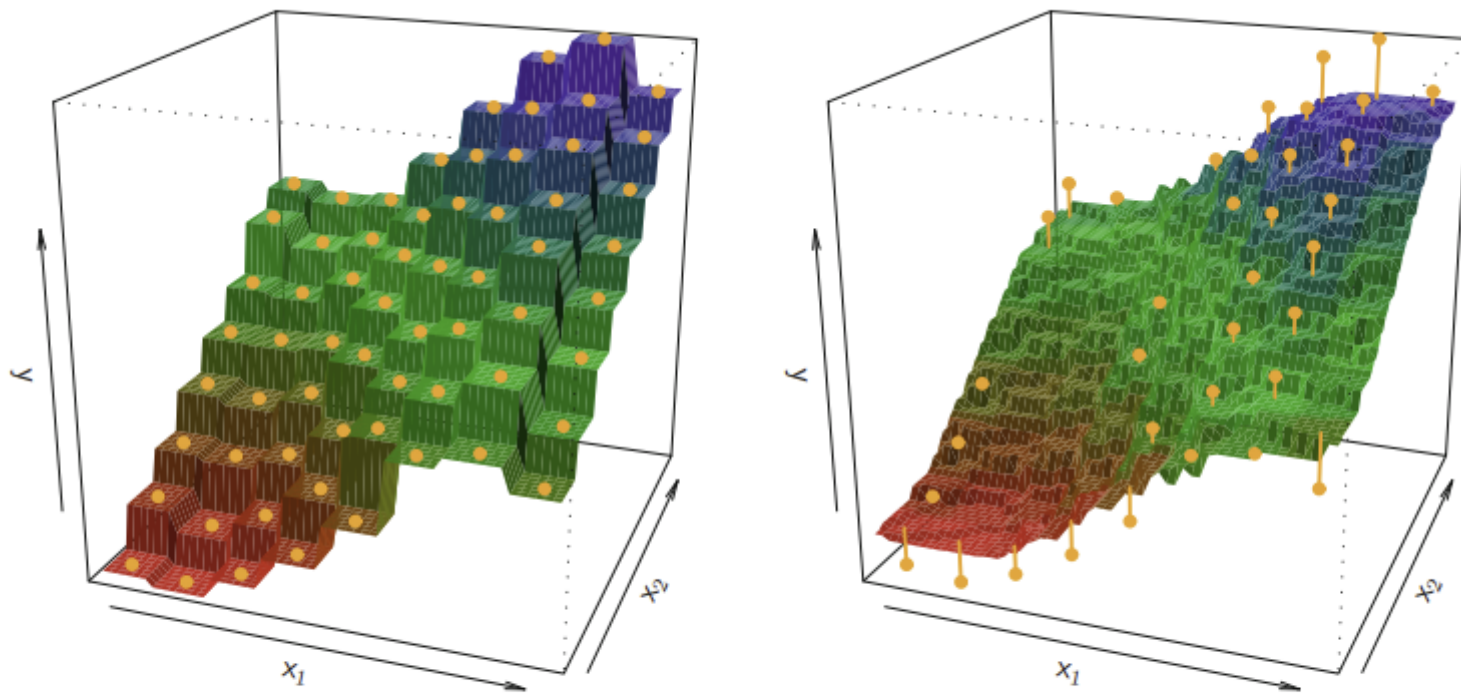


FIGURE 3.16. Plots of $\hat{f}(X)$ using KNN regression on a two-dimensional data set with 64 observations (orange dots). Left: $K = 1$ results in a rough step function fit. Right: $K = 9$ produces a much smoother fit.

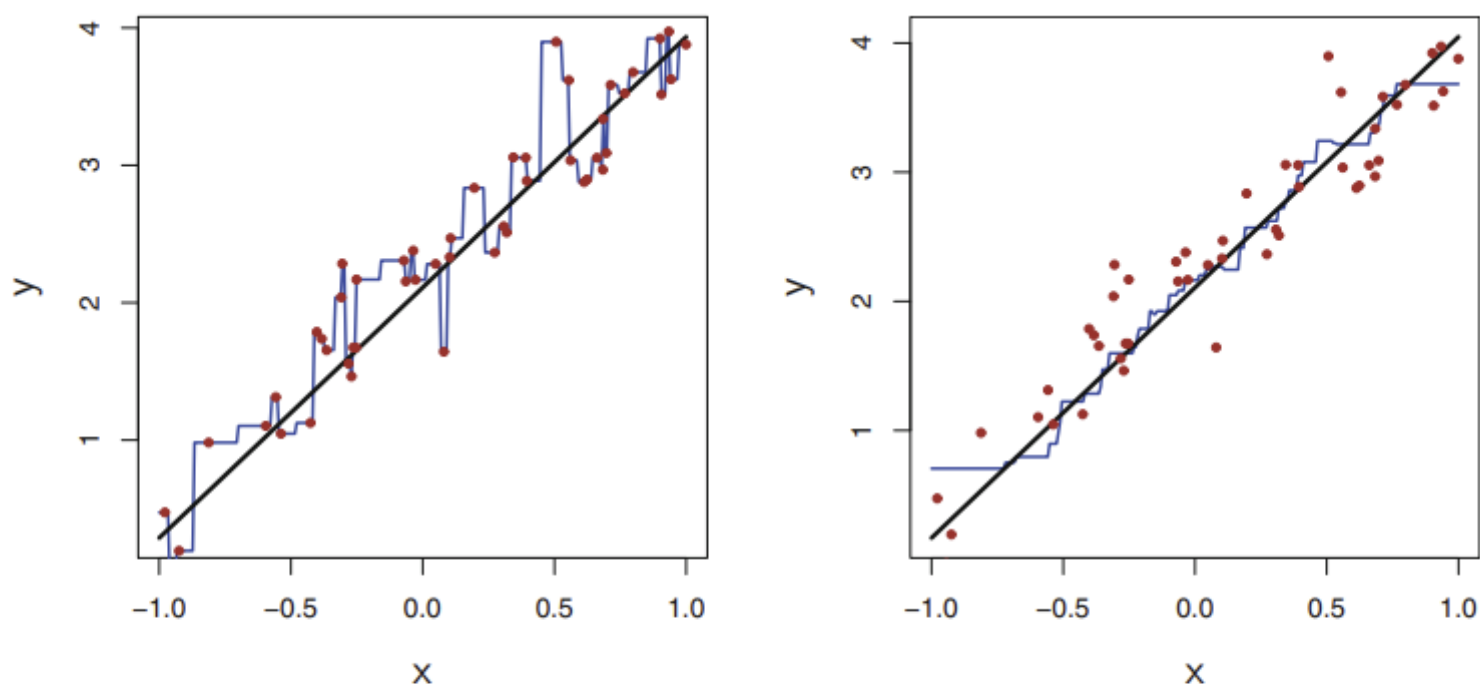


FIGURE 3.17. Plots of $\hat{f}(X)$ using KNN regression on a one-dimensional data set with 100 observations. The true relationship is given by the black solid line. Left: The blue curve corresponds to $K = 1$ and interpolates (i.e. passes directly through) the training data. Right: The blue curve corresponds to $K = 9$, and represents a smoother fit.

Figure 3.17은 1차원 선형 회귀 모델에서 발생한 데이터 샘플을 보여준다. 검은 선이 실제 데이터 관계를 나타내는 $f(x)$ 이다. 파란 커브는 각 $k=1, k=9$ 일 때이다. 이 경우 $k=1$ 일 때는 예측의 분산이 꽤 크고, $k=9$ 일 때 smooth 하며 실제 $f(x)$ 에 더 근사한다.

하지만 이렇게 선형 관계에 있는 데이터에서 knn 과 같은 non-parametric 방식을 적용하기에는 어려움이 있다. 아래 이미지에서 녹색 그래프는 KNN에 대한 MSE (평균 제곱 오차) 를 계산한 결과이며, 왼쪽 그래프에서 파란 라인은 선형 회귀 이며 데이터를 잘 설명한다고 볼 수 있다.

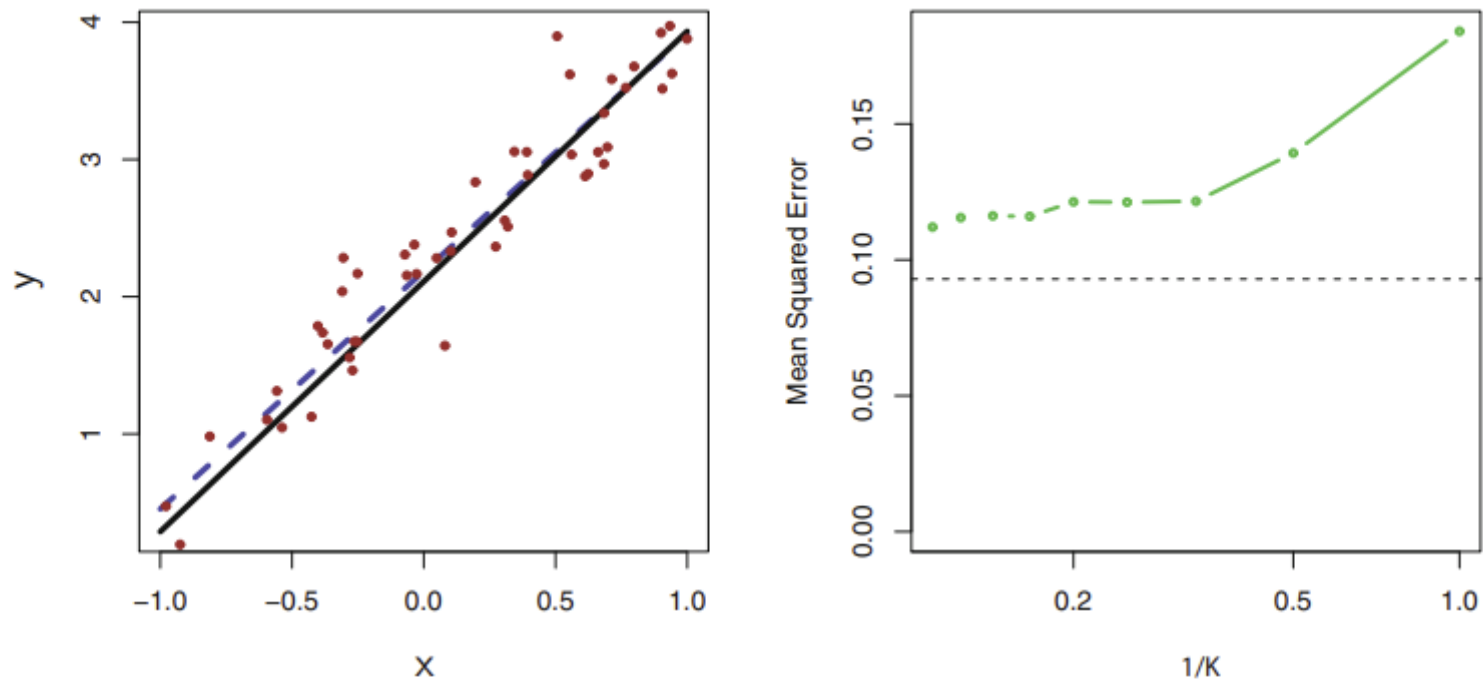


FIGURE 3.18. The same data set shown in Figure 3.17 is investigated further. Left: The blue dashed line is the least squares fit to the data. Since $f(X)$ is in fact linear (displayed as the black line), the least squares regression line provides a very good estimate of $f(X)$. Right: The dashed horizontal line represents the least squares test set MSE, while the green solid line corresponds to the MSE for KNN as a function of $1/K$ (on the log scale). Linear regression achieves a lower test MSE than does KNN regression, since $f(X)$ is in fact linear. For KNN regression, the best results occur with a very large value of K , corresponding to a small value of $1/K$.

반면, X와 Y 가 선형 관계에서 멀어진다면, non-parametric 은 효과가 있다. k 값이 어느 정도 크기라면, knn이 선형 회귀보다 데이터 관계를 더 잘 설명하게 된다.

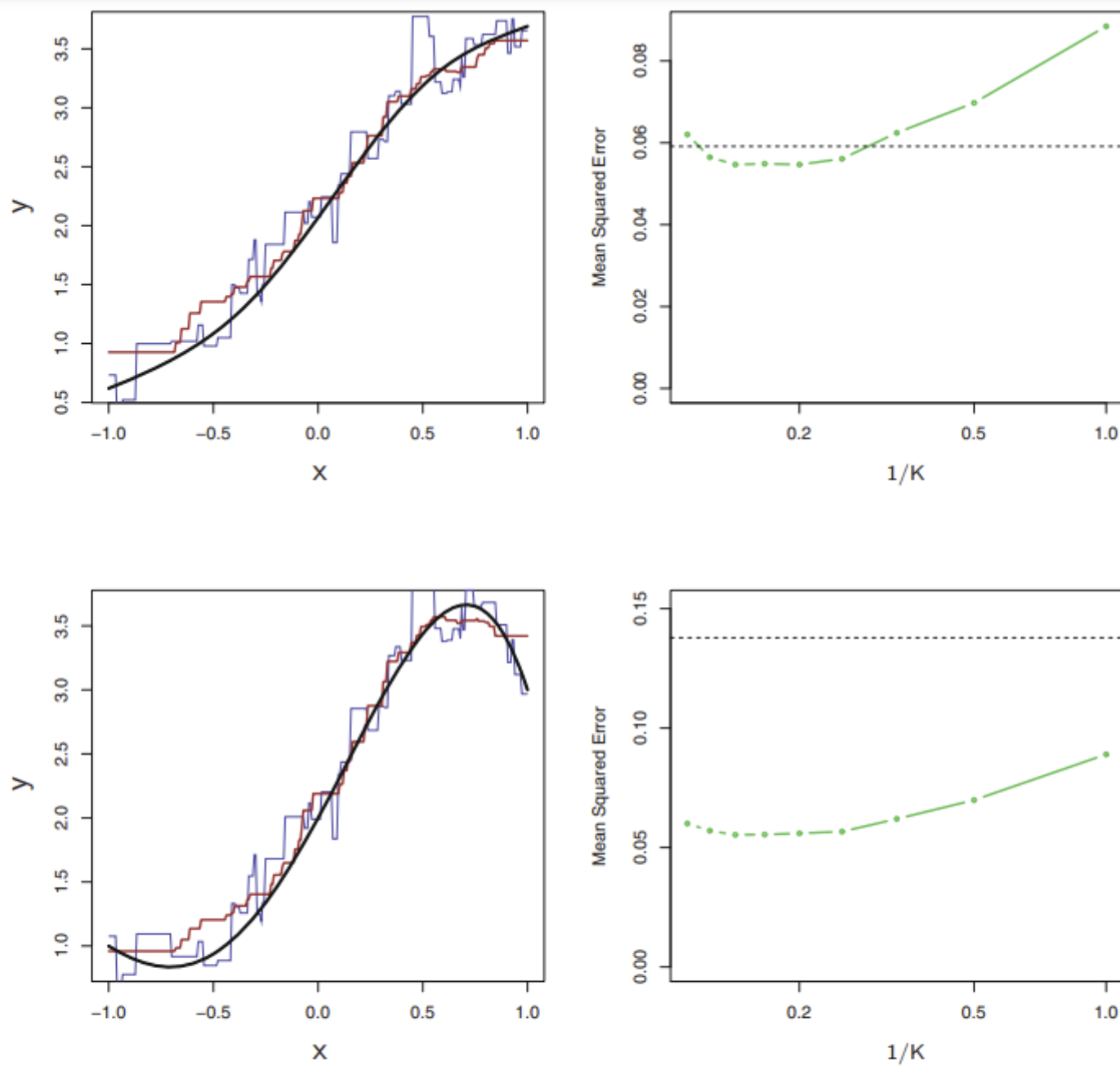
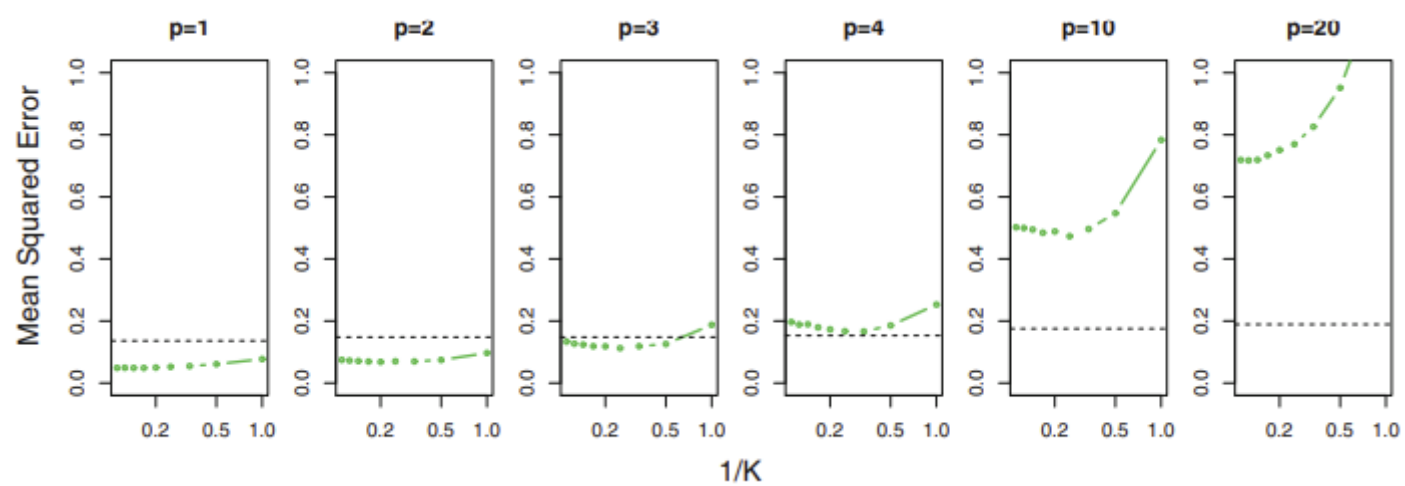


FIGURE 3.19. Top Left: In a setting with a slightly non-linear relationship between X and Y (solid black line), the KNN fits with $K = 1$ (blue) and $K = 9$ (red) are displayed. Top Right: For the slightly non-linear data, the test set MSE for least squares regression (horizontal black) and KNN with various values of $1/K$ (green) are displayed. Bottom Left and Bottom Right: As in the top panel, but with a strongly non-linear relationship between X and Y .



In reality, even when the true relationship is highly non-linear, KNN may still provide inferior results to linear regression.

헨즈온 : 로지스틱 회귀(Logistic regression)

로지스틱 회귀는 샘플이 특정 클래스에 속할 확률을 추정하는데 사용된다. 추정 확률이 50% 가 넘으면 모델은 그 샘플이 해당 클래스에 속한다고 예측한다. (레이블이 '1' 인 양성 클래스) binary classification 이라고도 불린다.

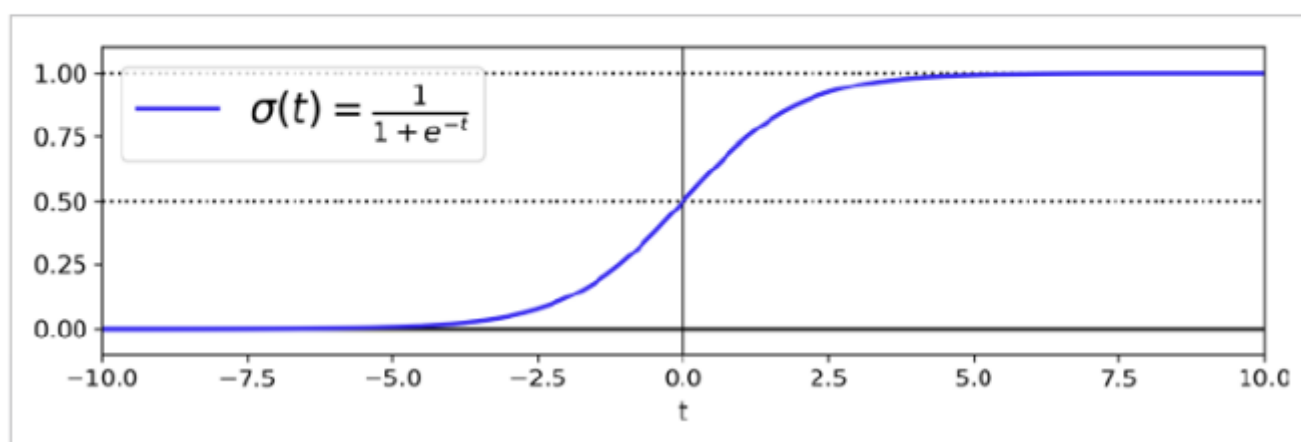
로지스틱 회귀는 입력 특성의 가중치 합을 계산하고 편향을 더하지만, 바로 결과를 출력하지 않고 결과값의 로지스틱을 출력한다.

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

예측 확률값이 0.5 미만이면 0(음성 클래스), 0.5 이상이면 1(양성클래스) 이다.

로지스틱 함수는 0-1 사이 값을 출력하는 시그모이드 함수이다.

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



로지스틱 회귀 모델의 비용 함수(=손실함수)

로지스틱 회귀 모델의 훈련 목적은 양성 샘플에 대해서는 높은 확률을 추정하고 음성 샘플에 대해서는 낮은 확률을 추정하는 모델의 파라미터 벡터 θ 를 찾는 것이다.

$$c(\theta) = \begin{cases} -\log(\hat{p}) & y = 1 \text{ 일 때} \\ -\log(1 - \hat{p}) & y = 0 \text{ 일 때} \end{cases}$$

위 수식은 하나의 훈련 샘플 x 에 대해 나타낸 비용 함수이다. t 가 0에 가까워지면 $-\log(t)$ 가 매우 커지므로 타당하다 할 수 있다. 모델이 양성 샘플을 0에 가까운 확률로 추정하면 비용이 크게 증가할 것이다. 반대로 음성 샘플을 1에 가까운 확률로 추정해도 비용이 증가한다. 다음 수식은 모든 훈련 샘플에 대한 비용함수이며, 이를 로그손실이라고 부른다.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)})]$$

이 비용함수는 convex 함수 이므로 경사 하강법이 global minimum 을 찾는 것을 보장한다. (→ 경사하강법 파트에서 자세히 설명된다.)

다음 편미분은 j번째 파라미터 θ_j 에 대한 편미분 결과이다. 로지스틱 비용 함수의 편도함수라고 부른다.

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

각 샘플에 대해 예측 오차를 계산하고 j 번째 특성값(= θ_j) 를 곱해서 모든 훈련 샘플에 대해 평균을 낸다. 모든 편도함수를 포함한 gradient vector 를 만들면 배치 경사 하강법 알고리즘을 사용할 수 있다.

소프트맥스 회귀(softmax regression)

로지스틱 회귀 모델은 직접 다중 클래스를 지원하도록 일반화 될 수 있으며 이러한 모델을 소프트맥스 회귀, 또는 다항 로지스틱 회귀 (multinomial logistic regression) 이라고 한다.

샘플 \mathbf{x} 가 주어지면 먼저 소프트맥스 회귀 모델이 각 클래스 k 에 대한 점수 $s_k(\mathbf{x})$ 를 계산하고, 그 점수에 소프트맥스 함수를 적용하여 각 클래스의 확률을 추정한다. 다음 식은 클래스 k 에 대한 소프트맥스 score 함수이다.

$$s_k(\mathbf{x}) = \left(\boldsymbol{\theta}^{(k)} \right)^T \mathbf{x}$$

각 클래스는 자신만의 파라미터 벡터 $\boldsymbol{\theta}^{(k)}$ 가 있으며 이 벡터들은 파라미터 $\boldsymbol{\theta}$ 에 행 벡터로 저장된다.

샘플 \mathbf{x} 에 대해 소프트맥스 score(각 클래스의 점수) 가 계산되면, 소프트맥스 함수를 통과시켜 클래스 k 에 속할 예측 확률을 추정할 수 있다. 소프트맥스 함수는 각 소프트맥스 score 에 지수 함수를 적용한 후 정규화한다.

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^K \exp(s_j(\mathbf{x}))}$$

- K 는 클래스 수입니다.
- $\mathbf{s}(\mathbf{x})$ 는 샘플 \mathbf{x} 에 대한 각 클래스의 점수를 담은 벡터입니다.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ 는 샘플 \mathbf{x} 에 대한 각 클래스의 점수가 주어졌을 때 이 샘플이 클래스 k 에 속할 추정 확률입니다.

소프트맥스 회귀 분류기는 추정 확률이 가장 높은 클래스를 선택한다. 이러한 점에서 회귀 모델이 분류에 빈번하게 사용된다는 것을 알 수 있다.

$$\hat{y} = \operatorname{argmax}_k \sigma(\mathbf{s}(\mathbf{x}))_k = \operatorname{argmax}_k s_k(\mathbf{x}) = \operatorname{argmax}_k \left(\left(\boldsymbol{\theta}^{(k)} \right)^T \mathbf{x} \right)$$

argmax 연산은 함수를 최대화하는 변수의 값을 반환한다.(numpy 라이브러리에서 자주 활용.) 이 식에서는 추정 확률이 최대인 k값을 반환한다.

소프트맥스 회귀 모델의 비용 함수(=손실함수)

소프트맥스 회귀 역시 로지스틱 회귀와 마찬가지로 모델이 타깃 클래스에 대해서는 높은 확률을 추정하도록 만드는 것이 목적이다. 크로스 엔트로피(cross entropy) 비용 함수를 최소화 하는 것은 타깃 클래스에 대해 낮은 확률을 예측하는 모델을 억제하므로 이 목적에 부합한다.

(*크로스 엔트로피는 추정된 클래스의 확률이 타깃 클래스에 얼마나 잘 맞는지 측정하는 용도로 또한 사용된다.)

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)})$$

- 이 식에서 $y_k^{(i)}$ 는 i 번째 샘플이 클래스 k 에 속할 타깃 확률입니다. 일반적으로 샘플이 클래스에 속하는지 아닌지에 따라 1 또는 0이 됩니다.

다음은 클래스 k 에 대한 크로스 엔트로피의 gradient vector 이다.

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^m (\hat{p}_k^{(i)} - y_k^{(i)}) \mathbf{x}^{(i)}$$

소프트맥스 또한 경사 하강법을 사용해 파라미터 행렬 Θ 를 찾을 수 있다.

이상이 핸드온 4장 에서 설명하는 회귀 모델 내용이다.