



3강: Locally Weighted & Logistic Regression

Probabilistic Interpretation of Loss Function

[통데마에서 배웠던 Linear Regression](#)

[Linear Regression 파이썬으로 구현하기](#)

[Locally Weighted linear regression](#)

[Parameteric vs. Nonparametric](#)

[Underfitting vs Overfitting](#)

[LWR algorithm](#)

[참고용 코드-Gaussian Basis Function](#)

[Classification - Binary Cases](#)

[Logistic Regression](#)

[Logistic Regression 파이썬으로 구현하기](#)

[Newton's Method](#)

[Perceptron Learning Algorithms](#)

Probabilistic Interpretation of Loss Function

왜 **regression** 문제에서 **loss function**은 **least-squares cost function**인가요 ?

그냥 절대값으로 해도 싫어요

알아보자고 휘비고

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

- $\epsilon^{(i)}$ 는 error term $\epsilon^{(i)} \sim N(0, \sigma^2)$; independently and identically distributed (IID)
- error term captures either **unmodeled effects** or **random noise**

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

(\because by Central Limit Theorem most error distributions are Gaussian)

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$y^{(i)}|x^{(i)}; \theta \sim N(\theta^T x^{(i)}, \sigma^2)$$

이를 이용하여 **likelihood function**을 적어보면 다음과 같다

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X; \theta) \\ = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

이 때, 우리는 $L(\theta)$ 를 최대화하는 θ 를 선택해야한다 (= making data as high probability as possible)

$L'(\theta) = 0$ 을 만족하는 θ 를 구하는데, 미분의 편의성을 위해 **log likelihood** $l(\theta)$ 를 대신 사용한다.

$$l(\theta) = \log L(\theta) \\ = \log \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ = \log \sum_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ = n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2$$

위의 수식을 통해 $l(\theta)$ 를 최대화하기 위해서는 $\frac{1}{2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2$ 를 최소화해야함을 알 수 있다.

데이터마이닝 HW 문제 기억하시나요?

Consider the linear regression model $y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_p x_{pi} + \epsilon_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i, i = 1, \dots, n$. When $\epsilon_i \sim^{i.i.d} N(0, \sigma^2)$ and σ^2 is known, show that the least square estimator (LSE), $\hat{\boldsymbol{\beta}}$, is the maximum likelihood estimator (MLE).

The likelihood function of $\boldsymbol{\beta}$ is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2}{2\sigma^2}\right] \\ = (2\pi\sigma^2)^{-n/2} \exp\left[-\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2\right].$$

MLE of $\boldsymbol{\beta}$ can be obtained by maximizing the log-likelihood function of $\boldsymbol{\beta}$.

$$\begin{aligned} \arg \max_{\boldsymbol{\beta}} l(\boldsymbol{\beta}) &= \arg \max_{\boldsymbol{\beta}} -\frac{n}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 \\ &\equiv \arg \max_{\boldsymbol{\beta}} -\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 \\ &\equiv \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 \\ &\equiv \arg \min_{\boldsymbol{\beta}} RSS \end{aligned}$$

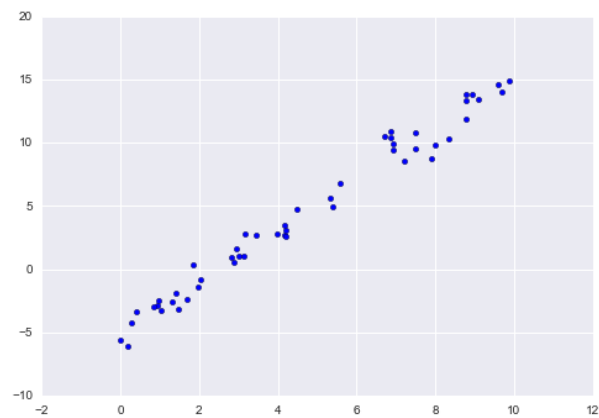
Since LSE is $\boldsymbol{\beta}$ minimizing RSS , LSE is MLE.

통데마에서 배웠던 Linear Regression

Linear Regression 파이썬으로 구현하기

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np

rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y);
```



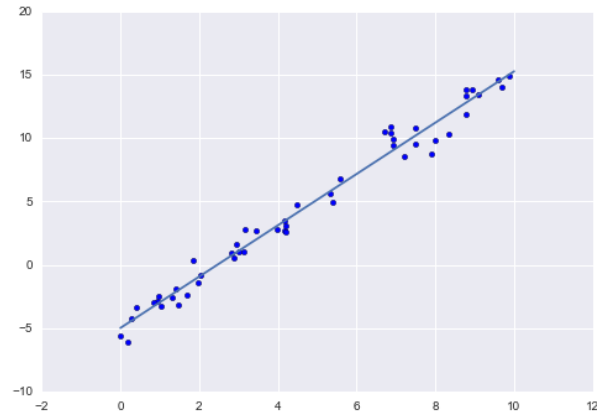
Scikit-Learn의 `LinearRegression` 함수를 이용해 fit해보자

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

model.fit(x[:, np.newaxis], y)

xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.plot(xfit, yfit);
```



```
print("Model slope:      ", model.coef_[0])
print("Model intercept:", model.intercept_)
```

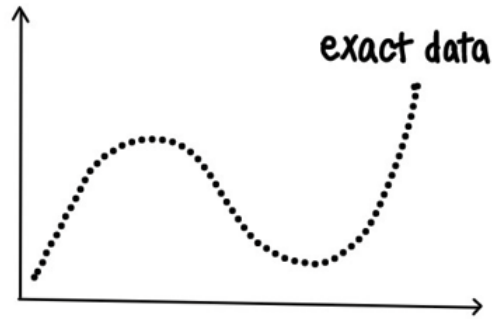
```
# Model slope:      2.02720881036
# Model intercept: -4.99857708555
```

Locally Weighted linear regression

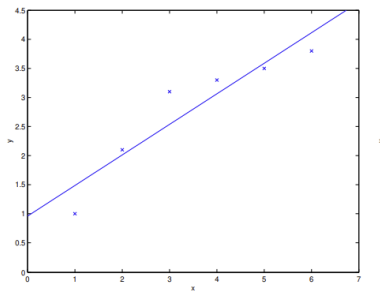
Parameteric vs. Nonparametric

1. **parametric learning algorithm** : fit fixed set of parameters θ_i to data
(예측에 있어서는 데이터를 통해 적합한 모수를 이용)
2. **nonparametric learning algorithm** (궁금한 사람 링크 들어가기)
: 주어진 데이터(샘플)을 그 자체를 이용해 새로운 값
 x_0 이 주어졌을 때의 분포를 추정
예) locally weighted linear regression
" the amount of data/ parameters, you need to keep grows , in this case grows linearly with the size of the data"

Underfitting vs Overfitting

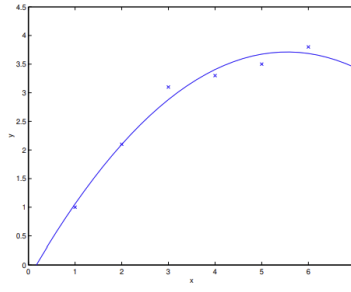


아무리 봐도 데이터는 직선 상에 분포하고 있지는 않다.
선형회귀를 통한 적합은 좋지 않을 것이다.



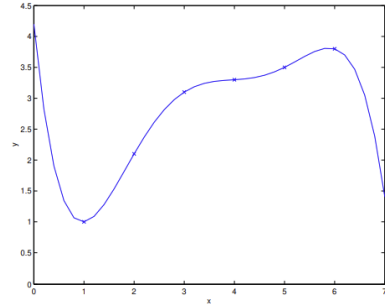
$$y = \theta_0 + \theta_1 x$$

- high bias, low variance
- simple model
- **underfitting**



$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

- obtain a slight better prediction than previous one

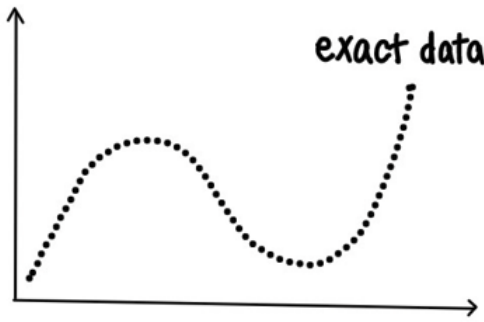


- high variance, low bias
- complexity ↑
- **overfitting**

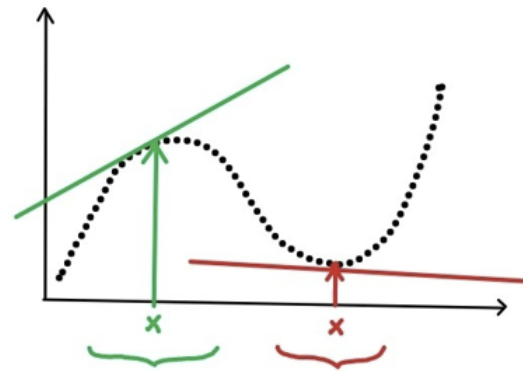
To ensure good performance of a learning algorithm?

: The choice of features is important to ensuring good performance

LWR algorithm



해당 데이터의 경우
선형회귀를 통한 적합은 좋지 않을 것이다. (underfitting)
⇒
Locally weighted regressions



main idea
x축 비슷한 위치에 있다 → 비슷한 값을 가질 것이다.

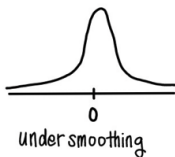
1. Fit θ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$ (where $w^{(i)}$ is non-negative valued weights)
2. Output $\theta^T x$

usually, fairly standard choice for the weights is

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

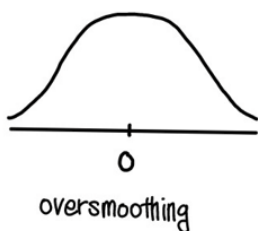
- 가우시안 분포 같지만 아니다. 적분하면 1이 아니다.
- $|x^{(i)} - x|$ is small , $w^{(i)} \approx 1$: 거리가 가까우면 높은 weight
- $|x^{(i)} - x|$ is large , $w^{(i)} \approx 0$: 거리가 멀어진다면 낮은 weight를 가짐
- τ : hyperparameter (**bandwidth**) ← model의 **complexity**를 결정

① when τ is small



- $|x^{(i)} - x|$ 가 유의미
- wiggly structures

② when τ is big



- $|x^{(i)} - x|$ 가 유의미하지 않음
- bumps smoothed out

앤드류 응 씨는 이걸 when we have a relatively **low dimensional dataset** ($n \approx 2$ or 3) 일 때 사용한다고 함

if we have a lot of data → computationally expensive

What happens if we need to infer the value of h **outside of the scope of the dataset?**

→ We can still use the algorithm but results may not be very good

참고용 코드-Gaussian Basis Function

```
from sklearn.base import BaseEstimator, TransformerMixin

class GaussianFeatures(BaseEstimator, TransformerMixin):
    """Uniformly spaced Gaussian features for one-dimensional input"""

    def __init__(self, N, width_factor=2.0):
        self.N = N
        self.width_factor = width_factor

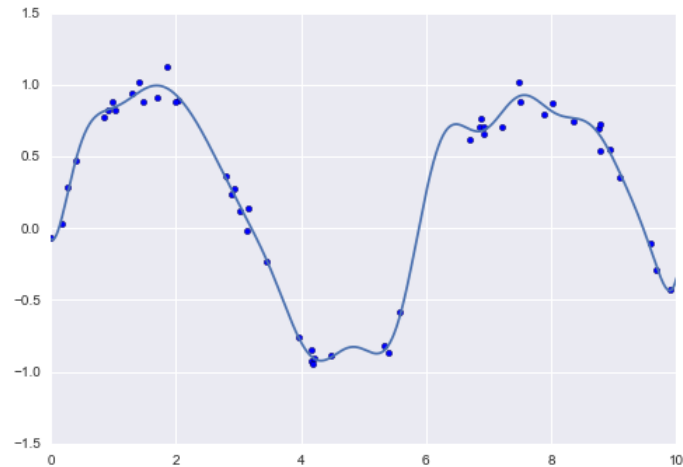
    @staticmethod
    def _gauss_basis(x, y, width, axis=None):
        arg = (x - y) / width
        return np.exp(-0.5 * np.sum(arg ** 2, axis))

    def fit(self, X, y=None):
        # create N centers spread along the data range
        self.centers_ = np.linspace(X.min(), X.max(), self.N)
        self.width_ = self.width_factor * (self.centers_[1] - self.centers_[0])
        return self

    def transform(self, X):
        return self._gauss_basis(X[:, :, np.newaxis], self.centers_,
                                  self.width_, axis=1)

gauss_model = make_pipeline(GaussianFeatures(20),
                             LinearRegression())
gauss_model.fit(x[:, np.newaxis], y)
yfit = gauss_model.predict(xfit[:, np.newaxis])
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.xlim(0, 10);
```



Classification - Binary Cases

: problem in which y can take on only two values

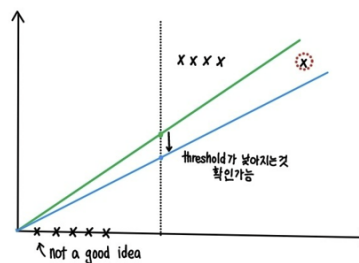
label : the corresponding $y^{(i)}$ given $x^{(i)}$

- **negative class (-)** : $y=0$ 일 때
- **positive class(+)** : $y=1$ 일 때

Logistic Regression

잉.... 통계 싫어... 그냥 이거도 선형회귀로 분류하면 안 되나요?—

안 됩니다

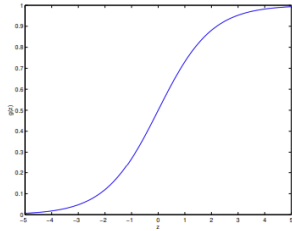


선형회귀로 분류시 데이터가 주어짐에 따라 threshold가 변화하는 것을 확인 가능하다.

Hypothesis Function

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ ← **sigmoid function (logistic function)**



시그모이드 함수

- $z \rightarrow \infty, g(z) \rightarrow 1$
- $z \rightarrow -\infty, g(z) \rightarrow 0$
- $\theta^T x$: 1보다 더 클 수도 있고 0보다 더 작을 수 있음
→ 보정 필요 (← 시그모이드 함수 $g(z)$ 를 통해서)

$$\theta^T x = \theta_0 + \sum_{j=1}^d \theta_j x_j$$

- $g(\theta^T x)$: output values only between 0&1

Useful Property of the derivative of the sigmoid function

미분.... 미분을 하자

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)) \end{aligned}$$

Probabilistic Approach - Likelihood

$$\begin{aligned} P(y = 1|x; \theta) &= h_{\theta}(x) \\ P(y = 0|x; \theta) &= 1 - h_{\theta}(x) \end{aligned}$$

라고 가정하면 $p(y|x; \theta)$ 는 다음과 같이 표현할 수 있다

$$p(y|x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

- $y=1$ 을 대입 시 $h(x)$
- $y=0$ 을 대입 시 $1 - h(x)$ 이

이를 활용하여 Likelihood 함수를 써보면

$$\begin{aligned} L(\theta) &= p(\vec{y}|\vec{X}; \theta) \\ &= \prod_{i=1}^n p(y^i|x^{(i)}; \theta) \\ &= \prod_{i=1}^n (h_{\theta}(x^{(i)}))^y (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

전처럼 log likelihood를 이용해서 최대화해보자

$$l(\theta) = \log L(\theta) \\ = \sum_{i=1}^n y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

이를 어떻게 최대화할 수 있을까? 우리에게는 **gradient ascent**가 있다!

$$\theta_j := \theta_j + \alpha \cdot \frac{\partial}{\partial \theta_j} l(\theta)$$

(\because maximize해야하니까 ascent(+))이다.)

벡터 notation으로 표현하면 다음과 같다.

$$\theta := \theta + \alpha \nabla_{\theta} l(\theta)$$

$$\begin{aligned} \frac{\partial}{\partial \theta_j} l(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_{\theta}(x)) x_j \end{aligned}$$

이 때, $g'(z) = g(z)(1 - g(z))$ 임을 고려하면 우리의 **stochastic gradient ascent rule**은 다음과 같다

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

어...? 이거 어디서 많이 본건데? 저기 우리 어디서 만난적 있지 않나요...?

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

바로 이 rule을 LMS update rule이라고 하는 것이다~~

똑 같아요

♬ 풀잎 동요마을 원곡: 박재홍

1. 무 엇 이 무 엇 이 똑 같 은 가
2. 무 엇 이 무 엇 이 똑 같 은 가

것 가 락 두 짝 이 똑 같 아 요
울 가 락 네 짝 이 똑 같 아 요

물론 로지스틱 회귀의 경우 $h_{\theta}(x^{(i)})$ 가 $\theta^T x$ 의 non-linear한 function이다.

Logistic Regression 파이썬으로 구현하기

```

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(features, labels)

print("Model slope:    ", model.coef_[0])
print("Model intercept:", model.intercept_)

model.predict(features) # 1또는 0으로 구성된 벡터를 반환해줌
model.predict_proba(features) # 각 샘플에 대한 확률을 0에서 1사이의 값을 돌려줌

```

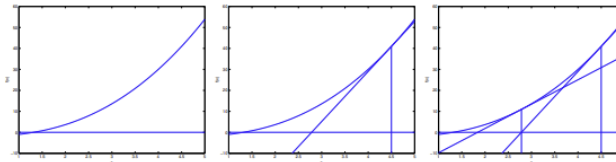
그런데 실제로 로지스틱 회귀 알고리즘을 사용하기 전에는 반드시 데이터를 정규화(Normalization) 해줘야 한다.

Newton's Method

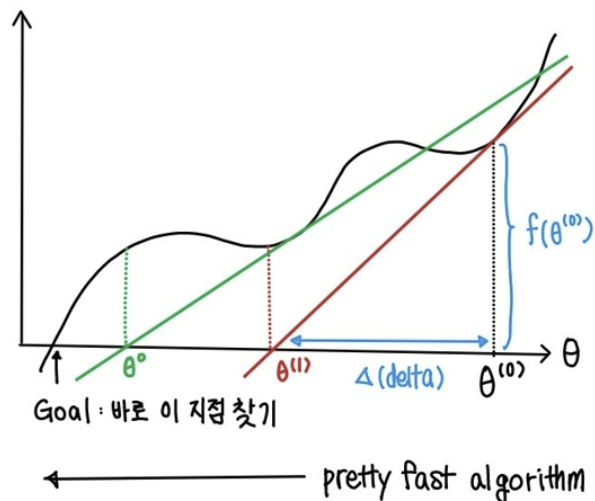
아 현기증 나 이래서 어느 세월에 수렴합니까

그런 널 위해 준비했어....☆☆☆☆☆☆☆☆☆☆

- Newton's method allows us *to take bigger jumps*



- When having f , want to find θ such that



위의 그림은 Newton's Method를 이용해 함수 f 의 해를 찾아나가는 방식을 간단하게 그린것이다.
수식으로 표현해보자면 다음과 같다.

$$\begin{aligned}\theta^{(1)} &:= \theta^0 - \Delta \\ f'(\theta^0) &= \frac{f(\theta^{(0)})}{\Delta} \\ \Delta &= \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}\end{aligned}$$

To generalize this,

$$\begin{aligned}\theta^{(t+1)} &:= \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})} \\ \text{let } f(\theta) &= l'(\theta) \\ \theta^{(t+1)} &:= \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}\end{aligned}$$

In vector Notation ,when θ is a vector(R^{n+1}),

$$\theta := \theta - H^{-1} \nabla_{\theta} l(\theta)$$

- $\nabla_{\theta} l(\theta)$: partial derivatives of $l(\theta)$ with respect to the θ_i 's.
- H is the Hessian Matrix($R^{(n+1) \times (n+1)}$)

$$H_{ij} = \frac{\partial^2}{\partial \theta_i \partial \theta_j}$$

- Newton's method enjoys a property called **quadratic convergence**

if one iteration Newton's method has 0.01 error

→ two iteration 0.0001 error

→ third iteration 0.00000001 error

→ **Extremely Rapid Convergence**

- 차원이 커진다면 Newton's method is much more expensive

Perceptron Learning Algorithms

로지스틱 회귀가 정확히 0 또는 1의 값을 출력할 수 있게 조금만 수정한다면 ,

threshold function

g 의 정의를 다음과 같이 할 수 있다.

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$


$h_{\theta}(x) = g(\theta^T x)$ 라 할 때, update rule이 다음과 같다면,

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_{\theta}(x^{(i)}))x_j^{(i)}$$

우린 이것 **perceptron learning algorithm**이라 부르기로 했어요

딥러닝이란? (feat 퍼셉트론)

퍼셉트론과 딥러닝

 <https://80000coding.oopy.io/2470b1c3-c280-4ad6-88ce-b649c2ba2196>

