GRID NOTEBOOK

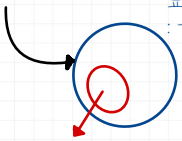
Linear Regression & Gradient Descent

Date

1. Static Review

1) 평균, 분산, 표준 편차

실험(Experiment)
: 결과물을 만드는 행위



평균 (mean)
: 기댓값(Expectation), μ

$E(X) = \sum X_i P(X)$: Discrete Data
$\int XP(X)$: Continaous Data

분산(Variance)
: Data들의 산포된 정도 , σ^2

$\sigma^2 = E(X^2) - [E(X)]^2$

표준 편차(Standard Deviation)

$\sigma = \sqrt{\text{분산}}$

예제) X	1	2	3
P	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$E(X)$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{3}{8}$
$E(X^2)$	$\frac{1}{8}$	1	$\frac{9}{8}$

$\rightarrow \mu = \frac{12}{8}$ $\sigma^2 = \frac{23}{16}$ $\mu = \frac{144}{8}$

2) Random Variable

: 랜덤 변수, 이름에 현혹 X -> 랜덤 변수 = 함수 (Function)

Random Variabe
- Discrete RV
  -> 이산적 Data : PMF
- Continuous RV
  -> 연속적 Data : CDF, PDF

PMF (Probability Mess Function)
-> 확률질량함수

Ex) x = 1, 2, 3
$P(X=x) = \frac{x}{6}$

$P(X==1) = \frac{1}{6}$
$P(X==2) = \frac{2}{6}$
$P(X==3) = \frac{3}{6}$

CDF (Continuous Distribution Function)
: 누적 분포 함수
PDF (Probability Density Function)
: 확률 밀도 함수

$P(X < a) = F(a)$
$P(X < b) = F(b)$
$P(a < X < b) = F(b) - F(a)$

CDF
$P(X \leq x) = F(x)$

Date

3) Gaussian Distribution

→ Normal Distribution

$N(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(x-\mu)^2\right)$

= Gaussian PDF

$\Phi(x;\mu,\sigma^2) \triangleq \int_{-\infty}^{x} N(z|\mu,\sigma^2)dz$

4) Joint Probability

$P(A,B) = P(A \cap B)$

$= f_{xy} = P(X=x, Y=y)$

= A,B가 동시에 일어날 확률

5) Conditional Probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

: Given B에 대한 A의 확률

6) Bayes Rule

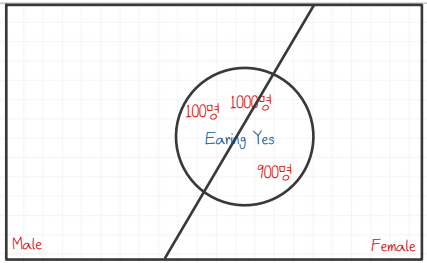
$$\underset{\text{Posterior}}{P(\Theta|X)} = \frac{\overset{\text{Likelihood}}{P(X|\theta)}\overset{\text{Prior}}{P(\Theta)}}{P(X)}$$

$$P(X=x|Y=y) = \frac{P(X=x)P(X=x|Y=y)}{\sum P(X=x')P(Y=y'|X=x')}$$

$$\underset{\text{Posterior}}{P(\Theta|X)} \propto \underset{\text{Likelihood}}{P(X|\Theta)}\underset{\text{Prior}}{P(\Theta)}$$

Ex)

10000명



$$\frac{P(남,귀)}{P(귀)} = P(남|귀) = \frac{P(귀|남)P(남)}{P(귀)}$$

$P(귀) = P(귀|남)P(남) + P(남|귀)P(귀)$

나중에 분모 고려 X

: posterior확률마다 분모는 동일하기에 고려 안함.

	귀	남	Total
남	100	6900	7000
여	900	2100	3000
Total	1000	9000	10000

2. Machine Learning 의 시작

1) Generative Classifier

$P(\Theta|X) \propto P(X|\Theta)P(\Theta)$

Posterior Likelihood Prior

Posterior Maximization: MAP(Maximum a posterior)

Likelihood Maximization: MLE(Maximum likelihood estimation)

(Likelihood가 training set일 가능성 많음(알고있을 가능성 큼))

2) Machine Learning의 분류

Supervised Learning VS Unsupervised Learning

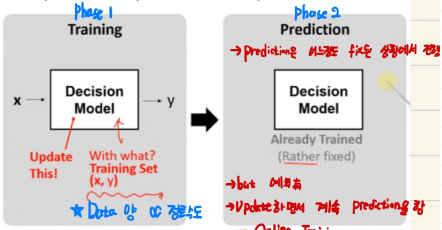
정답 있음 정답 없음

Classification VS Regression

이산 data 분류 연속 data 분류

In Supervised Learning,

- The operations are performed in two phases



Machine Learning

data를 Y(알고 싶은 것), X(알고 있는 것)으로 나눈다

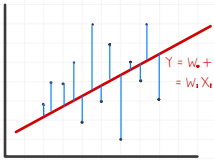
Y = f(X) + n 관계라 할 때

f(Decision Model) 를 추정하는 것이 machine learning이다.

Programming (X -> Y)

X로부터 Y를 도출

3. Linear Regression



1) Machine Learning 에서 matrix 를 사용하는 이유

Input feature: Data의 특징들을 수치화해서 카테고리화 시킴

➔ This 4D data can be expressed with a vector

Feature Extraction

특징추출(수치화)

$$x = \begin{bmatrix} 8.1 \\ 2.8 \\ 5.0 \\ 1.1 \end{bmatrix}$$

input feature

Data의 추출해서 수치화한 특성 : Vector

Feature를 여러 개 뽑아내면 무조건 행렬로 처리해야 함!!

Linear Regression

Date

2) Weight & Design Matrix

Weight = 가중치, 비중

각각의 input feature에 대해서 영향력/ 가중치를 평가

Ex) $\underline{X}$ = 4D = $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ $\underline{W}$ = 키 = $\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$

$\underline{W}^T\underline{X} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$

$= \underline{W} \cdot \underline{x}$

Weight 종류가 여러개

$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

3) Design Matrix

$\underline{x}_1 = \begin{bmatrix} 1.1 \\ 1.2 \\ 1.3 \\ 1.4 \end{bmatrix}$ $\underline{x}_2 = \begin{bmatrix} 2.1 \\ 2.2 \\ 2.3 \\ 2.4 \end{bmatrix}$ $\underline{x}_3 = \begin{bmatrix} 3.1 \\ 3.2 \\ 3.3 \\ 3.4 \end{bmatrix}$ $\underline{x}_4 = \begin{bmatrix} 4.1 \\ 4.2 \\ 4.3 \\ 4.4 \end{bmatrix}$

Design Matrix = $X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \underline{x}_3^T \\ \underline{x}_4^T \end{bmatrix} = \begin{bmatrix} 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ 3.1 & 3.2 & 3.3 & 3.4 \\ 4.1 & 4.2 & 4.3 & 4.4 \end{bmatrix}$

$X = N \times D$

(Data의 차원수 / Data 수)

4) Model parameter

Machine learning을 할 때, 가장 중요한 것?

Model 설정!!!!!!

Weight = 중요도.

Weight를 input data에 곱하면??

Vector 내적!!!!

$\hat{y} = \underline{W}^T\underline{x} = W_0 + W_1x_1 + W_2x_2 + W_3x_3 + \cdots + W_Dx_D \Rightarrow$

- $\hat{y}$: Predicted value (Predicted value = 예측 값)
- D : Feature (Dimension) 수
- x_i : i 번째 input feature
- W_j : j 번째 model parameter

내적

$\therefore \hat{y} = h_\theta(x) = \underline{W}^T\underline{x}$

Θ=W = Model parameter vector

=> 우리가 결정해야하는 model

(Called 'Hyperparameter')

5) Residual Sum of Squares

-> 정확성을 판단하는 방법이 뭘까?

-> Cost function

-> Rss(W) = Cost Func = $\sum_{i=1}^{N}(y_i - \underline{W}^T\underline{x}_i)^2$

-> RSS = 0 이면 Best!!

6) Mean square Error

-> Square Error의 평균!! 즉, RSS(W)의 평균

-> MSE(W) = $\frac{1}{N}RSS(W)$

$= \frac{1}{N}(y_i - \underline{W}^T\underline{x}_i)^2$

Date

Finding optimal weight

4. Linear regression – Analytical way

1) Ordinary Least Squares

$$MSE(W) = \sum_{i=1}^{N}(y_i - W^T x_i)^2 \cdot \frac{1}{N} \longrightarrow \nabla MSE = -\frac{2}{N}\sum_{i=1}^{N}(y_i - W^T x_i)\cdot x_i = 0$$

$$\therefore \sum (W^T x_i)\cdot x_i = \sum y_i x_i \quad ; \text{Vector의 표현}$$

↳ 이걸 행렬로 어떻게 표현하지?

Idea 1) Design matrix

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \vdots & \\ — & x_N & — \end{bmatrix}$$

→ N×D 행렬

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Idea 2) (스칼라) = 스칼라

$W^T x_i$ = 벡터 내적 = 스칼라

$$\therefore (W^T x_i)^T = (W^T x_i) = x_i^T W$$

$$(W^T x_i) x_i = (x_i^T W) x_i = (x_i^T x_i) W$$

Idea 3) Concatenation

In python : vstack, hstack, dstack, column_stack

→ 이게 Design matrix!!!!

$$\# \sum (W^T x_i) x_i = \sum y_i x_i$$

$$\sum (x_i^T W) x_i = \sum y_i x_i$$

$$\sum (x_i^T x_i) W = \sum y_i x_i$$

$$\downarrow$$

$$X^T X W = X^T y$$

By inverse matrix

* Ordinary Least Squares

$W = W^* = W_{OLS}$

$$W_{OLS} = (X^T X)^{-1} X^T y$$

Stack in python(Concatenation)

```
import numpy as np
import pandas as pd

a = [1,2,3,4,5]
b = [6,7,8,9,10]
c = [11,12,13,14,15]


Row1 = np.r_[a,b,c]
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

```
Row2 = np.hstack([a,b,c])
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

```
Column1 = np.c_[a,b,c]
```

```
[[ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]
 [ 5 10 15]]
```

```
Column2 = np.dstack([a,b,c])
```

```
[[[ 1  6 11]
  [ 2  7 12]
  [ 3  8 13]
  [ 4  9 14]
  [ 5 10 15]]]
```

```
print(Row1)
print(Row2)
print(Column1)
print(Column2)
```

Is There any Numerical way?

* Numerical Way 가 왜 필요해?

→ 차원이 커지면 Computational Complexity up !!

→ 계산 복잡도 up !!

→ So, Closed-form Solution으로 풀기 불가능!!

Date

5. Linear Regression – Numerical way

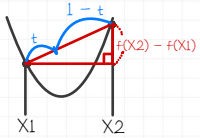
차원(N)이 커지면 Computational Complexty up!!!
즉 계산 복잡도 매우 큼!!
많은 경우 Analytical 한 방법으로 풀기 불가능

1) Numerical Way Solution
* Gradient Descent
* Newton – Raphson method
* Least-Squares Solution

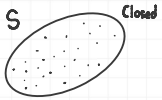
2) Gradient Descent

Convex 개념

* 볼록함수



* feasible Set

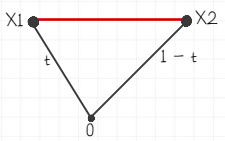


S 안에 존재하는 모든 X1,X2에 대해
tX1 + (1−t)X2 ⊂ S

Convexity ≜ 모든 X1, X2 ⊂ 정의역 , t = [0,1]
∴f(tX1 + (1−t)X2) ≦ tf(X1) + (1−t)f(X2)

내분점

내분점의 함숫값



* 즉, 구간 내에서 함수보다 위에 있다!!

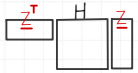
Multi Variable function

: function은 scalar & Input = Vector , Output = Scalar

Scalar를 Vector로 미분 = Vector (Jacobian) → Vector를 Vector로 미분 = Matrix (Hessian)

Z = Vector , H = Hessian

$z^T H z$ =



D x 1 D x D 1 x D = Scalar
= 1x1

Convex가 왜 중요해?
: feasible set에서 convex하면
Local min = Global min

Gradient: Vector로 미분
* ∇: Gradient = Gradient의 방향 = 가파른 방향
* But, 내가 원하는 것 = 미분이 0이 되는 Local Point 지점

그럼 그 반대 방향으로 가면
∇MSE가 최소가 되겠네?!!

Date

Gradient Descent Algorithm

* MLE 관점: P(W|D)

-> Model parameter를 update

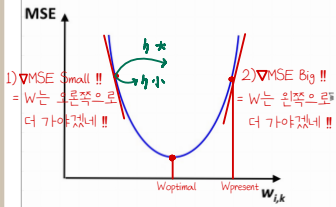
$MSE(w) = \sum_{i=1}^{N} \frac{1}{N}(y_i - w^T x_i)^2$: (진짜 Data의 y좌표 - 예측 Data의 y좌표)의 제곱의 평균 - RSS의 평균

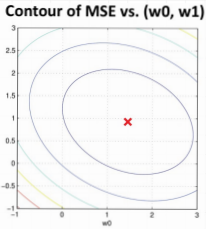
$\nabla MSE(w) = \sum_{i=1}^{N} \frac{-2}{N}(y_i - w^T x_i)x_i$

Gradient Discent

$W_{next} = W_{present} - \eta \nabla MSE(W)$

§ Gradient Descent는 초기값이 중요!! §



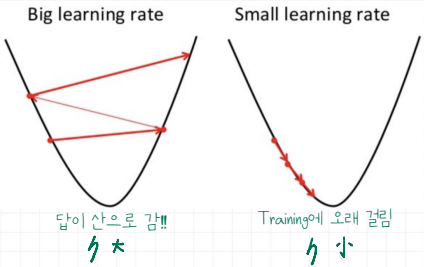


만약 Wpresent ≠ Woptimal 이면

W를 optimal W 쪽으로 update해야겠네!!!

(∇MSE의 반대방향 !!)

Learning Rate의 중요성 !!



Data Set으로 update 하는 방법 !!

-> MAP = P(D|W)

Step 1. Set up the initial value

Step 2. Xnext = Xpresent - (Learning rate) x (Gradient MSE)

$x_{next} = x_{present} - \eta \nabla MSE$

Step 3. Iteration

Date

Step 1) Initial value 설정하기

* Gradient Descent는 초기값이 매우 중요
* 방향 = $-\nabla$ = − Gradient = $-\frac{\partial f}{\partial x^T}$
* 얼마나? = Step size = Learning rate

Step 2)

* Scalar를 Vector로 미분

Objective Function : RSS = $\| y - Ax \|_2^2$ A: Weight matrix

$f(a,b) = df = \frac{\partial f}{\partial a}da + \frac{\partial f}{\partial b}db$

→ $f_{(a,b)}$ 변환당 = $\begin{bmatrix} \frac{\partial f}{\partial a} & \frac{\partial f}{\partial b} \end{bmatrix} \begin{bmatrix} da \\ db \end{bmatrix}$

$= \boxed{\frac{\partial f}{\partial x^T} \cdot dx}$

→ df를 구한 다음에 dx 앞에 있는게 Gradient

$\therefore df = \lim_{dx \to 0} \frac{f(a+dx) - f(a)}{dx}$ → first Order만 미분.

$\Rightarrow dx = 0 \Longleftrightarrow \frac{dx^2}{dx} = 0$

$f = (y - Ax)^T (y - Ax)$

$df = \left[y - A(x+dx) \right]^T \left[y - A(x+dx) \right] - (y - Ax)^T (y - Ax)$

$= \left[(y - Ax) - Adx \right]^T \left[(y - Ax) - Adx \right] - (y - Ax)^T (y - Ax)$

$= (y - Ax)^T (y - Ax) - (Adx)^T (y - Ax) - (y - Ax)^T Adx + (Adx)^T Adx - (y - Ax)^T (y - Ax)$

$= -(Adx)^T (y - Ax) - (y - Ax)^T Adx$

$= -dx^T A^T (y - Ax) - (y - Ax)^T A dx$

dx^T · A^T · $(y - Ax)$ = Scalar

$= \left[-dx^T A^T (y - Ax) \right]^T - (y - Ax)^T A dx$

$= dx^T A^T (y - Ax) - (y - Ax)^T A dx$

$= -(y - Ax)^T A dx - (y - Ax)^T A dx$

$= -2(y - Ax)^T A dx$

$= -2(y - Ax)^T A \cdot dx$; $\frac{\partial f}{\partial x} \cdot dx = -2(y - Ax)^T A dx$

$\therefore \frac{\partial f}{\partial x^T} = -2(y - Ax)^T A = \nabla MSE$

* X가 Gradient와 반대방향으로 가면 되겠네!!
* $+2(y - Ax)^T A$ 방향!!

$\therefore x_{next} = x_{present} + h \cdot 2(y - Ax)^T A$

* 곱의 미분으로도 풀수 있다!!

$(fg)' = $ 앞미그뒤 + 앞그뒤미

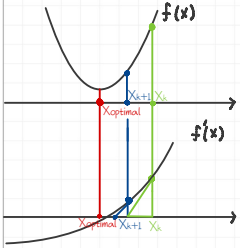
$= f'g + fg'$

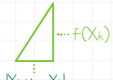
→ $f = (y - Ax)^T (y - Ax)$

$df = \underbrace{(-Adx)^T (y - Ax)}_{\text{Scalar}} + \underbrace{(y - Ax)^T (-Adx)}_{\text{Transpose}} = -2(y - Ax)^T A dx$

Date

3) Newton – Raphson Method

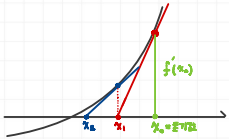




Newton –Raphson method는 Basically " Zero– Finding "

* Let Objective function = f(X) = ($\underline{y} - A\underline{x}$)^T($\underline{y} - A\underline{x}$)
* Gradient Descent의 결과 : Scalar를 Vector로 미분

$x_{k+1} = x_k + 2(\underline{y} - A\underline{x})^T A \cdot h$



접선의 기울기 = 미분 –> 미분 역수 x f(Xo) = t

$\cdot |x_{next} - x_{present}| = |\Delta x|$

$\cdot \frac{\Delta y}{\Delta x} = f'(x_{present}),\ \Delta y = f'(x_{present})$

$$X_{next} = X_{present} - \frac{f(X_{present})}{f'(X_{present})}$$

Gradient Descent 에서 Scalar를 Vector로 미분 –> Jacobian (Vector)

Newton–Raphson Method

* Scalar를 Vector로 두번 미분 = Vector를 Vector로 미분 = Hessian (Matrix)
* 두번 미분하고 나눈것은 역행렬을 곱한 것과 같다.

$$\frac{\partial}{\partial \underline{x}^T}\left(\frac{\partial f}{\partial \underline{x}^T}\right) = \frac{\partial}{\partial \underline{x}^T}\left[2(\underline{y} - A\underline{x})^T A\right]$$

= Scalar

$\cdot$ [Scalar]^T = Scalar

$$\frac{\partial}{\partial \underline{x}^T}\left[2(\underline{y} - A\underline{x})^T\right]^T = \frac{\partial}{\partial \underline{x}^T}\left[2A^T(\underline{y} - A\underline{x})\right] = 2A^TA = \text{Hessian Matrix}$$

$\cdot$ [Hessian]$^{-1} = \frac{1}{f''}$

$\therefore\ x_{next} = x_{present} - (2A^TA)^{-1}2A^T(\underline{y} - A\underline{x}_{present})$

4) Least Squares Solution

만약에 Weight matrix A가 역행렬이 있다면? invertable 하다면?

$(2A^TA)^{-1}2A^T(\underline{y} - A\underline{x})$

$= \frac{1}{2}\cdot\frac{1}{2}\cdot A^{-1}A^{-T}2A^T(\underline{y} - A\underline{x})$

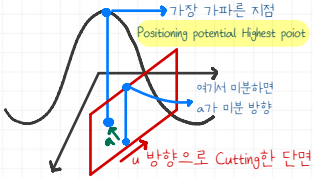
$\rightarrow \underline{x}_{next} = \underline{x}_{present} + A^{-1}(\underline{y} - A\underline{x}_{present})$

$= x_{present} + A^{-1}\underline{y} - \underline{x}_{present}$

$\therefore \underline{x}_{next} = A^{-1}\underline{y}$

Date

6. 방향도함수와 Gradient



Directional Derivative

$$\lim \frac{f(x + h\vec{u}) - f(x)}{h} = \nabla f \cdot \underline{u}$$

$\vec{u}$ 방향으로의 방향 미분 |u| = 1

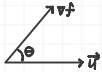
* $f(\underline{x}) = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = f\left(\begin{bmatrix} x_1 + hu_1 \\ x_2 + hu_2 \end{bmatrix}\right)$

→ 이러면 x1 , x2 는 h의 함수가됨 (u1 , u2 는 내가 정한 방향 : 상수)

$$\frac{df}{dh} = \frac{df}{dx_1} \cdot \frac{dx_1}{dh} + \frac{df}{dx_2} \cdot \frac{dx_2}{dh} = \nabla f \cdot \underline{u}$$

따라서 Gradient = $\vec{u}$ 방향으로의 미분

* 그럼 $\vec{u}$를 바꿔 가면서 어느 방향으로 제일 크니? |u| = 1



$\nabla f \cdot \underline{u}$ = 최대 얼마지?

$\nabla f \cdot \underline{u} = |\nabla f||\underline{u}|\cos\theta$

$\therefore \theta = 0°$일 때!!

즉, Gradient 방향 = $\vec{u}$ 방향일때 가장 가파름 !!

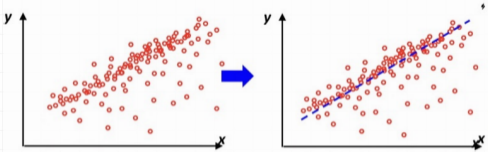
7. Linear Regression 보충

Phase 1. Training에서 MLE를 써보자 !!

$W_{next} = W_{present} - \eta \cdot \nabla MSE$: P(W|D)

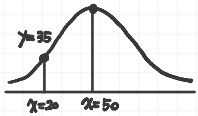
-> MAP 임!!

Phase 2. MLE??



MLE = P(W|D)

* 파란색 선을 찾을때 MLE를 쓴다!!
* Linear Regression이 좀더 meaningful 하려면 Probability 써야 함 !!



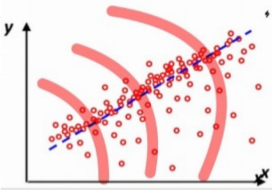
ㄱ: "X = 20 에서 Y = 35" 이거보단

ㄴ: "X = 20일때 평균에서 얼마만큼 떨어졌고 확률이 얼마야 !!"

@ ㄴ이 훨씬 meaningful !!

Linear Regression에 Gaussian Model 적용

가정: 표준편차는 fix이고 x에 따라 평균만 change



$P(y \mid \underline{x}, \underline{\theta}) = N(y \mid \underline{W}^T\underline{x}, \sigma^2)$

* 계산 편이: log 사용(ln)

$\underline{\theta} \triangleq \underset{\theta}{\arg\max} \ln P(D \mid \underline{\theta})$

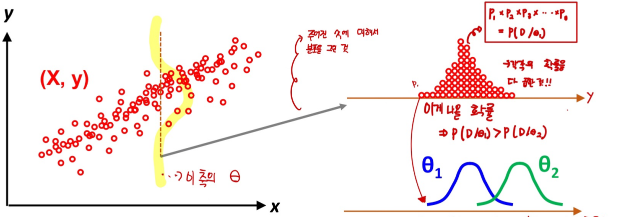
$\therefore \ell(\underline{\theta}) = \text{log-likelihood} \triangleq \log P(D \mid \theta) = \sum_{i}^{N} \log P(y_i \mid \underline{x}_i, \underline{\theta})$

Meaning of p(D/θ)? $P(\theta/D) \propto P(D/\theta)P(\theta)$

이거 구하기 힘드니 → 이걸 최대화

↳ θ가 달라도 Uniform

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2)$$



➔ Can you compare $p(D/\theta_1)$ and $p(D/\theta_2)$?
Note that probability is probability ∝ PDF

➔ Can you relate this with Number Game case? Occam's razor?

↳ 이거는 엄밀히 말해서 확률 X, 면적으로 적분해야 확률

⇒ But, PDF가 클수록 확률이 더 높다고 예상 가능

6

MLE 계산유도

* Analitic한 방법으로 구한 (MAP) wOLS 와 결과 같음

But, why MLE?

-> P(D|W)는 구하기 어려워

-> Model은 눈으로만 보면 알기 어려움 ; 계산 필요

$P(W|D) \propto P(D|W)P(W)$

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \mathbf{w}^T\mathbf{x}_i)^2\right)\right]$$
$$= \frac{-1}{2\sigma^2} RSS(\mathbf{w}) - \frac{N}{2}\log(2\pi\sigma^2)$$

$\therefore \ell(\underline{\theta})$의 최대 = RSS(w)의 최소

→ $(\underline{y} - A\underline{x})^2$을 최소화 = MLE를 최대화

Minimizing MSE approach for Comparing it w/ MLE or MAP

Minimizing MSE it to minimize the following

$$\text{MSE}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2$$ = Error

* MAP is to maximize p(W|D)
* MLE is to maximize p(D|W)

For now, we can roughly say that decreasing error is equivalent to increase probability

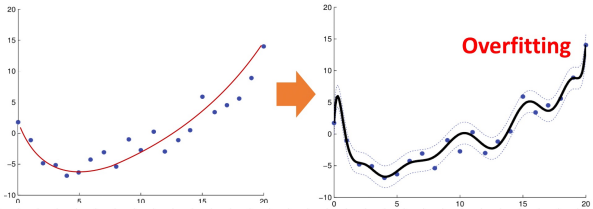
(MLE와 MSE는 방식은 달라도 결과는 같다!!)

Date

8. Overfitting

: Dataset에 매몰되어 Model을 정함

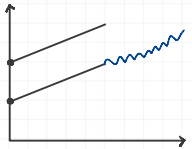
-> Solution 1) Prior 2) Abundent Data 3) Regularization



P(W|D) = P(D|W)P(W)

* 만약 MLE를 위해서 P(D|W)만 maximize하면 overfitting 생길 수 있음
* data 수가 적으면 특히 많이 발생
* Prior 를 이용하거나 더 많은 data를 fitting!!

Performing MAP



$W_0 + (W_1x_1 + W_2x_2 + W_3x_3 + W_4x_4 + \cdots)$;

W_0 W^Tx

W_0는 Zero mean 이면 안됨

-> W_0는 data가 X에 대해 중심점을 잡아주는 것이야 !!

$p(D|\boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}|w_0 + {}^{w}\boldsymbol{x}, \boldsymbol{\sigma^2})$ and $p(\mathbf{w}) = \prod_j \mathcal{N}(w_j|0, \tau^2)$

- This leads to maximize log likelihood:

$$\underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \mathcal{N}(y_i|w_0 + \mathbf{w}^T\mathbf{x}_i, \sigma^2) + \sum_{j=1}^{D} \log \mathcal{N}(w_j|0, \tau^2)$$

이것만 최대화하면 Overfitting !!

* 목표는 전체 최대화 !!

-> 기존이랑 Different , why?

-> 기존에는 P(W) = Prior 가 uniform distribution

- That is equivalent to minimize

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + \mathbf{w}^T\mathbf{x}_i))^2 + \lambda||\mathbf{w}||_2^2$$

where

$$\lambda \triangleq \sigma^2/\tau^2 \text{ and } ||\mathbf{w}||_2^2 = \textstyle\sum_j w_j^2 = \mathbf{w}^T\mathbf{w}$$

Date

- That is equivalent to minimize

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + \mathbf{w}^T\mathbf{x}_i))^2 + \lambda||\mathbf{w}||_2^2$$

where

$$\lambda \triangleq \sigma^2/\tau^2 \text{ and } ||\mathbf{w}||_2^2 = \sum_j w_j^2 = \mathbf{w}^T\mathbf{w}$$

* Wo 는 offset -> Wo는 regularized 안됨: 함수의 높이에만 영향을줌
* 다시 말해 bias는 regularized X -> complexity에 영향X
* lambda = Regularizer = λ

Optimal w with Regularization

- The resultant optimal ***w*** is

$$\hat{\mathbf{w}}_{ridge} = (\lambda\mathbf{I}_D + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

* Ridge Regression = Penalized Least Squares
* Prior를 uniform distribution 이 아닌, Gaussian prior term으로 더해주면 이를 Regularization 또는 Weight decay라고 한다.
* Regularization의 가장 큰 목적은 OVERFITTING을 줄이는 것!!!!!!!

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import math as m

plt.close("all")

dfLoad = pd.read_csv("https://raw.githubusercontent.com/hanwoolJeong/lectureD
                     , sep ="\s+")

xxRaw = dfLoad["xx"]
yyRaw = dfLoad["yy"]
yyRawNP =np.array(yyRaw)
plt.plot(xxRaw, yyRaw, ".r")

# # 이 상황에서 linear regression모델 찾아보겠다이말이야
Ndata = len(xxRaw)
X = np.c_[np.ones([100, 1]), xxRaw]
# wOLS =  X.T.dot(X)  행렬곱까지

# wOLS =  np.linalg.inv(X.T.dot(X)) 역행
wOLS =  np.linalg.inv(X.T.dot(X)).dot(X.T).dot(yyRawNP.reshape(Ndata,1))
# wOLS 구해짐
xPredict = np.linspace(0,2,num=101)
xPredictPadding = np.c_[np.ones([101,1]), xPredict]
yPredict = wOLS.T.dot(xPredictPadding.T)

# plt.plot(xPredict.reshape(1,101), yPredict, "b.")
# 직선 만들기
eta = 0.1
Niteration = 10
# 20번돌렸을때는 차이가 얼마 안나서 200번으로 바꿈(Niteration) , print page 27, 28

wGD = np.array([0,0]).reshape(2,1)
for iteration in np.arange(Niteration):
    gradients = -(2/Ndata)*(X.T.dot(yyRawNP.reshape(Ndata,1)-X.dot(wGD)))
    wGD = wGD - gradients*eta
    print(iteration)
    yGD = wGD.T.dot(xPredictPadding.T)
    plt.plot(xPredict.reshape(1,101), yGD, "b.")
```