* WHY?

① Machine Learning 기본 목표 : Model이 처음보는 data에 대해서도 잘 예측하게 하고 싶다.

(training data 뿐만 아니라 진짜 세상의 data 생성 mechanism을 잘 모사하는 Model이 필요해!)

② 그럼 data가 이렇게 이렇게 생겼다고 치자.

= Data는 어떤 (알 수 없지만 존재한다고 가정되는) 확률 분포로부터 생성됐다고 가정.

이때 Model이 할 일? ┌ 그 확률분포를 최대한 잘 흉내내는 parameter를 찾자.
└ 즉, "내가 이 데이터를 실제로 만들어냈을 수 있다"고 말할 수 있는 Model을 찾아보자.

③ MLE Concept.

Data는 내가 선택한 Model로부터 생성됐고, 그 중 가장 가능성이 높은 parameter는 무엇인가?

= 여러 후보 parameter 중에서 이 dataset이 일어날 가능성(Likelihood)이 가장 큰 parameter θ를 고르자!

e.g) 동전을 던졌을때 앞면이 나올 확률 θ를 모른다고 가정,

→ 10번 던졌더니 9번이 앞, 1번이 뒷면 나왔다면, $\theta = 0.9$ 일때 이 data가 나올 가능성이 제일 높겠지?

그럼 우리는 "이 data가 가장 그럴듯하게 나올수 있는 θ인 0.9를 선택하는 것.

④ Next step.

MLE를 사용해서 확률문제인 Logistic Regression을 풀어보자. MLE 표현) $\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i | x_i, \theta)$

→ log를 씌워볼까? ∵ ① 곱 → 합 바꿈 : 계산을 쉽게하자

② 확률을 곱하면 값이 매우 작아짐 → underflow 방지.

→ 최소화 문제로 바꾸자 $\underset{\theta}{\operatorname{argmax}} \approx \underset{\theta}{\operatorname{argmin}} - (\cdots)$ $\hat{\theta} = \underset{\theta}{\operatorname{argmin}} - \sum_{i=1}^{n} \log P(y_i | x_i, \theta)$

⑤ 엥? 바꿔놓고 보니 Cross-Entropy Loss랑 형태가 똑같네?

⑤ 엥? 바꿔놓고 보니 Cross-Entropy Loss랑 형태가 똑같네?

- Cross Entropy (informatics 관점)

$CrossEntropy(p, q) = -\sum_{K} p(K) \log q(K)$: 두 확률분포 p(true), q(predict)에 대해 정의되는 값.

= 즉, 실제 label이 맞다고 했을때, Model이 그 label을 얼마나 "확신했는가"

- $p(K)$: 실제 분포
- $q(K)$: Model이 예측한 분포

Loss를 보는 이유? → 모델 학습시키기 위해서 (모델 parameter를 잘 조정해서 예측을 더 정확하게 하자)

- Cross Entropy Loss Model이 예측을 잘하면 loss는 0에 가까움, 예측을 못하면 loss가 커짐.

$CrossEntropy(p, q) = -\log q(y_i)$ → Loss 값을 어떻게 활용할까?

<최소화>하자! 모델이 점점 더 정답에 높은 확률을 주도록 parameter를 바꾸자.

= gradient descent $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \, loss$

해석) Loss 값이 작아지면, 모델의 예측이 더 정확해짐.

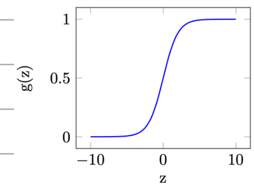
8 Logistic Regression → Classification ; x가 주어졌을때 결과 y를 예측할때 사용.

학률 예측 ... Binary classification : spam filtering ($y=1$ spam, $y=0$ no spam) 이메일 분류

• Formel : 확률적으로 접근해보자 → MLE ("현재 모델이 data를 얼마나 잘 설명하는가" 측정지표

- given : training set $\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$ where $x_i \in \mathbb{R}^d$, $y_i \in \{0,1,\cdots,K-1\}$
- goal : find classifier h
- Classification 1) generative : learns $P(x|y)$, $P(y)$ →
  2) discrimitive : describe $P(y|x)$ diretly → y 값을 직접 판별

8.1 Logistic Regression Model



• Problem State

우선 Binary 부터 시작해보자. Classification을 contnous-valued로 접근하면 성능 ↓↓↓

대신 classifier의 parameter form을 바꿔, [0,1] 구간만 출력하는 것으로 가능. sigmoid function

• Model

$$h_\theta(x) = g(\langle\theta, x\rangle) \text{ , where } g(x) = \frac{1}{1+e^{-z}}$$

→ discriminative model, $P(y|x)$를 직접 모델링.

→ 확률 출력이 가능 → 불확실성 표현 가능. decision threshold 조정 가능

Then, propability $P(y=1|x) = h_\theta(x)$,
$P(y=0|x) = 1 - h_\theta(x)$

→ 최종적) $\hat{y} = \begin{cases} 1 , & \text{if } \langle\theta,x\rangle > 0 \\ 0 , & \text{otherwise} \end{cases}$

8.2 Fitting a Model

• Goal : 모델의 parameter θ를 given dataset $D=\{(x_1,y_1),\cdots,(x_n,y_n)\}$에 대해 학습.

• Suppose x is deterministic, y generated at random & independently (확률적 발생)

$P(y_i=1 | x_i,\theta) = h_\theta(x_i)$, $P(y_i=0|x_i,\theta) = 1-h_\theta(x_i)$

• Maximum Likelyhood Estimate (MLE) → MLE가 D가 모델 h_θ에서 나올 확률을 최대화. = MLE의 θ 찾기.

$$L(\theta,D) = P(y_1,\cdots,y_n|x_1,\cdots,x_n,\theta) = \prod_{i=1}^{n} P(y_i|x_i,\theta) = \prod_{i=1}^{n} (h_\theta(x_i))^{y_i}(1-h_\theta(x_i))^{1-y_i}$$

↓ log

$$\log L(\theta) = \sum_{i=1}^{n} y_i \log h_\theta(x_i) + (1-y_i)\log(1-h_\theta(x_i))$$

이 함수를 최대화하자 = loss minimization problem!

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n}\left[y_i \log\frac{1}{h_\theta(x_i)} + (1-y_i)\log\frac{1}{1-h_\theta(x_i)}\right]$$

we note that in Regression, we started our discussion by fitting model by minimizing the loss between data predizted by the model & traning data, and we chose the quadratic loss function $\text{loss}(y,z) = (y-z)^2$

The theoretical justification was that learning with the quadratic loss corresponds to MLE under Gaufian noise

we could approach logistic regression

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \text{loss}(h_\theta(x_i), y_i)$$

chosen log-loss or negative cross entropy

$$\text{loss}(y,z) = -y\log(z) - (1-y)\log(1-z) = y\log\left(\frac{1}{z}\right) + (1-y)\log\left(\frac{1}{1-z}\right)$$

classification 문제에 적합한 확률기반의 loss function.

classfication error 보다 확률값을 고려. gradient 학습에 유리

8.3 Computing the Logistic Regression Estimate with gradient descent

• The negative log-likelyhood

$$-\log L(\theta, D) = \sum_{i=1}^{n} loss(g(\langle\theta, x_i\rangle), y_i) \text{ with } loss(g(\langle\theta, x_i\rangle), y_i) = -y_i \log(g(\langle\theta, x_i\rangle)) - (1-y_i)\log(1-g(\langle\theta, x_i\rangle))$$

↑ it is convex. Thus, we can approximate the logistic regression estimate $\hat{\theta}$ with gradient descent.

1) $\theta^{k+1} = \theta^k - \alpha \nabla(-\log L(\theta, D))$ ** In practice, use SGD ∵ computational reason.

$$= \theta^k - \alpha \sum_{i=1}^{n} \nabla loss(g(\langle\theta, x_i\rangle), y_i)$$

2) Compute Gradient of Loss ** Logistic function 에서는 도함수(미분) 사용 가능!

$$\boxed{g(z) = \frac{1}{1+e^{z}}}$$

$$g'(z) = \frac{-1}{(1+e^{-z})^2} e^{-z}(-1) = \frac{1}{(1+e^{-z})}\left(1 - \frac{1}{(1+e^{-z})}\right) = g(z)(1-g(z))$$

$g'(z) = g(z)(1-g(z))$

$$\Rightarrow \frac{\partial}{\partial\theta_j} loss(y, h_\theta(x)) = -\left(y \frac{1}{g(\langle\theta,x\rangle)} - (1-y)\frac{1}{1-g(\langle\theta,x\rangle)}\right)\frac{\partial}{\partial\theta_j} g(\langle\theta, x\rangle)$$

$$= -\left(\frac{y - g(\langle\theta,x\rangle)}{g(\langle\theta,x\rangle)(1-g(\langle\theta,x\rangle))}\right) g(\langle\theta,x\rangle)(1-g(\langle\theta,x\rangle))\frac{\partial}{\partial\theta_j}\langle\theta,x\rangle$$

$$= (g(\langle\theta,x\rangle) - y)x_j$$

$$\nabla^2 loss(y, h_\theta(x)) = \underbrace{g(\langle\theta,x\rangle)(1-g(\langle\theta,x\rangle))}_{\geq 0}\underbrace{x_j x_i}_{\text{postive-semidefint}}$$

neg. log-likelihood 도 결국 볼록(convex) 함수의 합이므로 GD는 최소화 함수로 수렴함.

⇒ Hessian is positive semidefinit, thus, loss is convex in θ.

• Sigmoid function : Binary Classification 에서 사용.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

• 왜 사용? 1) 확률 해석 가능 $\sigma(z) = P(y|x)$ 로 해석 가능
2) 미분 가능 → Gradient Descent 에 사용 가능
3) 값을 0-1 사이로 정규화 → 0~1 ≈ 확률처럼 사용 가능

• e.g) $z = -0.5 \rightarrow \sigma(-0.5) = \frac{1}{1+e^{0.5}} = \frac{1}{1+\sqrt{e}} = 0.37 \ldots$

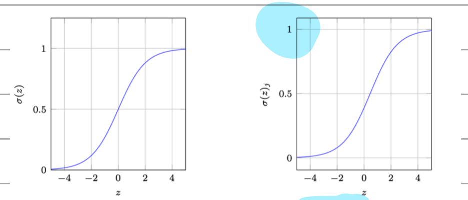
$z = 0 \rightarrow \sigma(0) = \frac{1}{1+1} = 0.5$

• intuitive)
참(True)일 확률을 부드럽게 출력하자. 예측 점수가 매우 크면 거의 1 ≙ 참일 확률 높다. $z \approx 0$ 이면 불확실함 $\sigma(0) = 0.5$
"얼마나 확신하냐?" 함수.

8.4 Multiclass Logistic Regression

지금까지 Binary 봤으니 이제 multi class를 보자. Key idea : One-hot Encoding

- Problem State : case with $K > 2$.
- One-Hot Encoding : $y \in \{0,1\}^K$, $[y]_k = 1$, if $y=k$.
  $[y]_j = 0$, for all $y \neq k$.
  e.g) $y=2$ 일때, $y = [0\ 0\ 1\ 0 \cdots 0]^T$
- we associate a parameter vector $\theta_k \in \mathbb{R}^d$ with each class.
  - → Input $x \in \mathbb{R}^d$ 에 대해 각 class별 score 계산. $z_k = \langle \theta_k, x \rangle$
  - → 모든 class의 score를 모은 vector $z = [z_1\ z_2 \cdots z_K]^T$
  - → Generalization of Logistic Function $[\sigma(z)]_j = \dfrac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$



(a) Sigmoid activation function. (b) Softmax activation function.

Figure 1: Sigmoid and Softmax activation functions

- Softmax function 입력된 ℝ 벡터를 확률분포로 변환하는 함수.

$$[\sigma(z)]_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

왜 사용? Binary Classification 에서 Sigmoid 사용됨. Class가 2개일때 → Sigmoid가 확률처럼 잘 작동함.
But, class가 3개 이상일때 Sigmoid는 확률분포 전체를 표현하지 못함. 이때 Softmax는 모든 클래스의 score를 한번에 고려해 각 클래스가 속할 확률을 동시에 출력

e.g) input vector $z = [2.0\ \ 1.0\ \ 0.1]$ 일때.

$$\text{softmax}(z_1) = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} \approx 0.659\cdots$$

마찬가지로 $\text{softmax}(z_2) \approx 0.242\cdots$

$\text{softmax}(z_3) \approx 0.09\cdots$

전체 결과 $\text{softmax}(z) \approx [0.66\ \ 0.24\ \ 0.1]$

⇒ 첫번째 클래스에 속할 확률이 66%로 가장 높다.

- intuitive)
  "누가 더 큰 점수를 받았나?" 높은 점수 = 높은 확률 / 낮은 점수 = 낮은 확률

- Multiclass classification parameter θ를 MLE로 학습해보자.

$$\Rightarrow \hat{\theta} = \arg\max_{\theta} \mathcal{L}(\theta, D) \stackrel{*}{=} \arg\max_{\theta} \prod_{i=1}^{n} P(y_i \mid x_i, \theta)$$

* Likelihood Definition: 각 sample x_i 에 대해 정답 y_i 가 나올 확률을 모델이 얼마나 잘 설명하는가.

$$\stackrel{*}{=} \arg\max_{\theta} \prod_{i=1}^{n} \prod_{i=1}^{K} P(y_i = k \mid x_i, \theta_k)^{\mathbb{1}\{y_i = k\}}$$

* One-Hot Encoding : 각 데이터 포인트에 대해 "정답 클래스에 해당하는 확률만" 곱해지도록 표현.

$$\stackrel{*}{=} \arg\min_{\theta} -\sum_{i=1}^{n} \sum_{i=1}^{K} \mathbb{1}\{y_i = k\} \log P(y_i = k \mid x_i, \theta_k)$$

* Log-Likelihood로 표현 최대값 구하는 문제를 로그 씌워 최솟값 구하는 문제로 & 곱 → 합 바꿔서 계산 편하게

$$\stackrel{*}{=} \arg\min_{\theta} -\sum_{i=1}^{n} \sum_{i=1}^{K} \mathbb{1}\{y_i = k\} \log \left(\frac{e^{\langle \theta_j, x_i \rangle}}{\sum_{j=1}^{K} e^{\langle \theta_j, x_i \rangle}} \right)$$

* Softmax를 통해 $P(y_i = k \mid x_i)$를 계산해서 대입하자

이 수식이 cross-entropy loss와 완전 동일 구조.

즉, $\text{Loss} = -\sum^{n} \sum^{K} y_{i,k} \log \hat{p}_{i,k}$

loss 수식을 최소화하면 모델이 정답 클래스에 높은 확률을 주도록 학습.

전체 과정은 convex optimization problem 이고, 따라서 gradient descent로 해를 구할수 있다.