Classification

4. Classification.ipynb

Cross entropy

Measures the differences between the true probability p_i and the predicted probability q_i

$$H(p,q) = -\sum_i p_i ln(q_i)$$

Use Excel to verify

Excel formula: =LN(x)

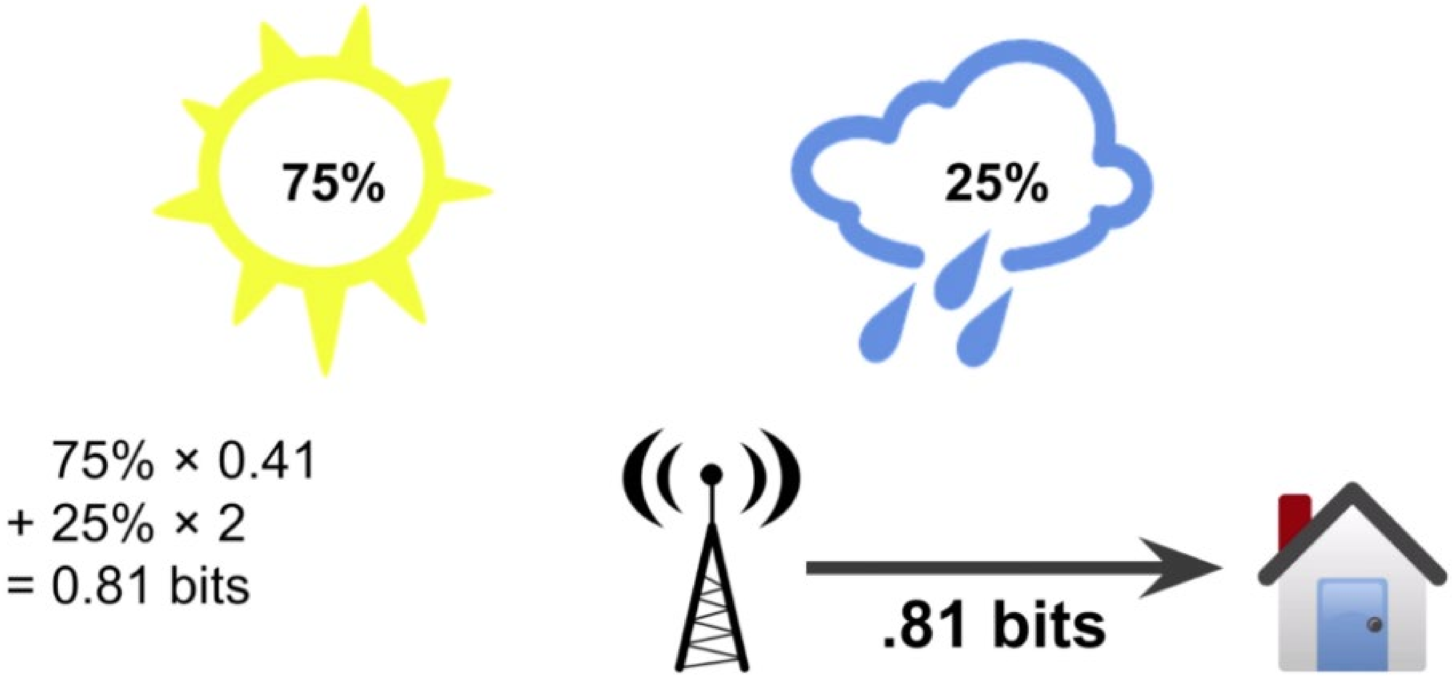
動物	實際機率分佈	預測機率分佈	Entropy
Cat	0%	2%	0% * -log(2%) = 0
Dog	0%	30%	0% * -log(30%) = 0
Fox	0%	45%	0% * -log(45%) = 0
Cow	0%	0%	0% * -log(0%) = 0
Red Panda	100%	25%	100% * -log(25%) = 1.386
Bear	0%	5%	0% * -log(5%) = 0
Dolphin	0%	0%	0% * -log(0%) = 0
總計: cross-entropy = 1.386			

https://r23456999.medium.com/%E4%BD%95%E8%AC%82-cross-entropy-%E4%BA%A4%E5%8F%89%E7%86%B5-b6d4cef9189d

Entropy

More information → more uncertain → larger entropy

$$Entropy = -\sum_i p_i log_2(p_i)$$



https://r23456999.medium.com/%E4%BD%95%E8%AC%82-cross-entropy-%E4%BA%A4%E5%8F%89%E7%86%B5-b6d4cef9189d

Softmax

```
In [15]: print(tensorY.shape,"\n", tensorY)

torch.Size([5, 2])
 tensor([[-0.0180,  0.0855],
        [-0.0244,  0.0741],
        [-0.0187,  0.0850],
        [-0.0258,  0.0687],
        [-0.0267,  0.0617]], device='cuda:0', grad_fn=<

In [16]: # apply softmax
         tensorY = torch.softmax(tensorY, 1)
         print(tensorY.shape,"\n", tensorY)

torch.Size([5, 2])
 tensor([[0.4742, 0.5258],
        [0.4754, 0.5246],
        [0.4741, 0.5259],
        [0.4764, 0.5236],
        [0.4779, 0.5221]], device='cuda:0', grad_fn=<So

In [17]: MaxOfEachRow = torch.max(tensorY, 1)
         print(MaxOfEachRow)

torch.return_types.max(
values=tensor([0.5258, 0.5246, 0.5259, 0.5236, 0.5221],
       grad_fn=<MaxBackward0>),
indices=tensor([1, 1, 1, 1, 1], device='cuda:0'))
```

torch.softmax(tensor, 1)

$$\frac{e^{y_1}}{e^{y_1} + e^{y_2}} \qquad \frac{e^{y_2}}{e^{y_1} + e^{y_2}}$$

0: ↓

1: →

torch.max(tentor, 1)

Torch.max

```
tensor([[0.4742, 0.5258],
        [0.4754, 0.5246],
        [0.4741, 0.5259],
        [0.4764, 0.5236],
        [0.4779, 0.5221]],
```

torch.max(tentor, 1)[1]

[1]: The 2nd item of torch.max results

```
In [17]: MaxOfEachRow = torch.max(tensorY, 1)
         print(MaxOfEachRow)
```

```
torch.return_types.max(
values=tensor([0.5258, 0.5246, 0.5259, 0.5236, 0.5221], device='c
       grad_fn=<MaxBackward0>),
indices=tensor([1, 1, 1, 1, 1], device='cuda:0'))
```

```
In [18]: MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
         print(MaxIdxOfEachRow)
```

```
tensor([1, 1, 1, 1, 1], device='cuda:0')
```

```
In [19]: correct = 0
         MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
         for i in range(batchY_hat.shape[0]):
           print(int(MaxIdxOfEachRow[i]), int(batchY_hat[i]), end="==>")
           if (int(MaxIdxOfEachRow[i]) == int(batchY_hat[i])):
             print("correct")
             correct += 1
           else:
             print("wrong")
         print(correct)
         accuracy = correct/batchY_hat.shape[0]
         print("%.2f" % accuracy)
```

```
1 0==>wrong
1 0==>wrong
1 0==>wrong
1 1==>correct
1 1==>correct
2
0.40
```

Bayesian's rule to understand $y_1 = p(C_1|x)$

$$y_1 = P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

Generative Model $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

Probabilistic generative model

$$f_{\mu^1,\Sigma^1}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^1|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1)\right\}$$

P(C1)
= 79 / (79 + 61) =0.56

$$\boxed{\mu^1 = \begin{bmatrix} 75.0 \\ 71.3 \end{bmatrix} \quad \Sigma^1 = \begin{bmatrix} 874 & 327 \\ 327 & 929 \end{bmatrix}}$$

Assuming x^n are sampled from a Gaussian distribution, then we can use maximum likelihood to find the best Gaussian distribution behind them.

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$f_{\mu^2,\Sigma^2}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^2|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2)\right\}$$

P(C2)
= 61 / (79 + 61)
=0.44

$$\boxed{\mu^2 = \begin{bmatrix} 55.6 \\ 59.8 \end{bmatrix} \quad \Sigma^2 = \begin{bmatrix} 847 & 422 \\ 422 & 685 \end{bmatrix}}$$

If $P(C_1|x) > 0.5$

Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

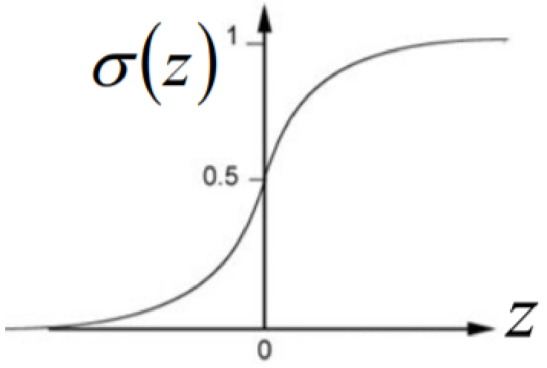
Represent $y_1 = p(C_1|x)$ as a sigmoid function of z

$$\boxed{y_1 = P(C_1|x)} = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$= \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} = \frac{1}{1 + exp(-z)} \boxed{= \sigma(z)}$$

Sigmoid function

$$\boxed{z = ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}}$$



Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

Represent $y_1 = p(C_1|x)$ as a sigmoid function of z

$$P(C_1|x) = \sigma(z)$$

Assuming the covariance matrices of the two classes are the same

$$z = \cancel{ln\frac{|\Sigma^2|^{1/2}}{|\Sigma^1|^{1/2}}} \cancel{-\frac{1}{2}x^T(\Sigma^1)^{-1}x} + (\mu^1)^T(\Sigma^1)^{-1}x - \frac{1}{2}(\mu^1)^T(\Sigma^1)^{-1}\mu^1$$
$$\cancel{+\frac{1}{2}x^T(\Sigma^2)^{-1}x} - (\mu^2)^T(\Sigma^2)^{-1}x + \frac{1}{2}(\mu^2)^T(\Sigma^2)^{-1}\mu^2 + ln\frac{N_1}{N_2}$$

$$\Sigma_1 = \Sigma_2 = \Sigma$$

$$z = \underbrace{(\mu^1 - \mu^2)^T\Sigma^{-1}}_{\boldsymbol{w}^T}x \underbrace{- \frac{1}{2}(\mu^1)^T\Sigma^{-1}\mu^1 + \frac{1}{2}(\mu^2)^T\Sigma^{-1}\mu^2 + ln\frac{N_1}{N_2}}_{\text{b}}$$

$$\boxed{y_1 = P(C_1|x) = \sigma(w \cdot x + b)}$$

How about directly find ***w*** and b?

In generative model, we estimate $N_1, N_2, \mu^1, \mu^2, \Sigma$

Then we have ***w*** and b

Logistic regression

If we use gradient decent to find optimal w and b for the posterior probability $y_1 = p(C_1|x) = \sigma(w \cdot x + b)$, then the problem becomes logistic regression.

We want to find $P_{w,b}(C_1|x)$

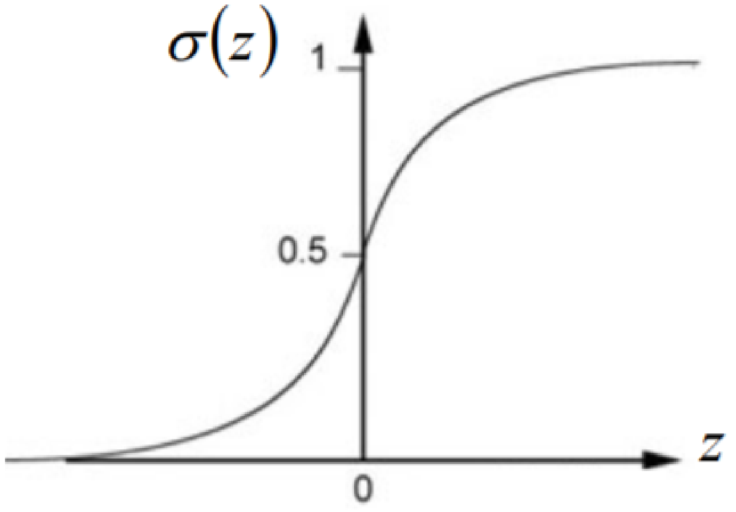
If $P_{w,b}(C_1|x) \geq 0.5$, output C_1

Otherwise, output C_2

$$P_{w,b}(C_1|x) = \sigma(z)$$

$$z = w \cdot x + b$$

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



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Logistic regression vs regression

Logistic Regression

$$f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$$

Output: between 0 and 1

Linear Regression

$$f_{w,b}(x) = \sum_i w_i x_i + b$$

Output: any value

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Maximum likelihood

Assuming the training data is generated from $y_1 = P_{w,b}(C_1 | x) = \sigma(w \cdot x +$

Training Data	x^1	x^2	x^3	……	x^N
	C_1	C_1	C_2		C_1

max $\quad L(w,b) = f_{w,b}(x^1) f_{w,b}(x^2) \left(1 - f_{w,b}(x^3)\right) \cdots f_{w,b}(x^N)$

min $\quad -lnL(w,b) = -lnf_{w,b}(x^1) - lnf_{w,b}(x^2) - ln\left(1 - f_{w,b}(x^3)\right) \cdots$

$\hat{y}^n$: 1 for class 1, 0 for class 2

$$= \sum_n -\underbrace{\left[\hat{y}^n lnf_{w,b}(x^n) + (1 - \hat{y}^n) ln\left(1 - f_{w,b}(x^n)\right)\right]}_{\text{Cross entropy between two Bernoulli distribution}}$$

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Loss function

Training data: $(x^n, \hat{y}^n)$

$\boxed{\hat{y}^n\text{: 1 for class 1, 0 for class 2}}$

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

Training data: $(x^n, \hat{y}^n)$

$\hat{y}^n$: a real number

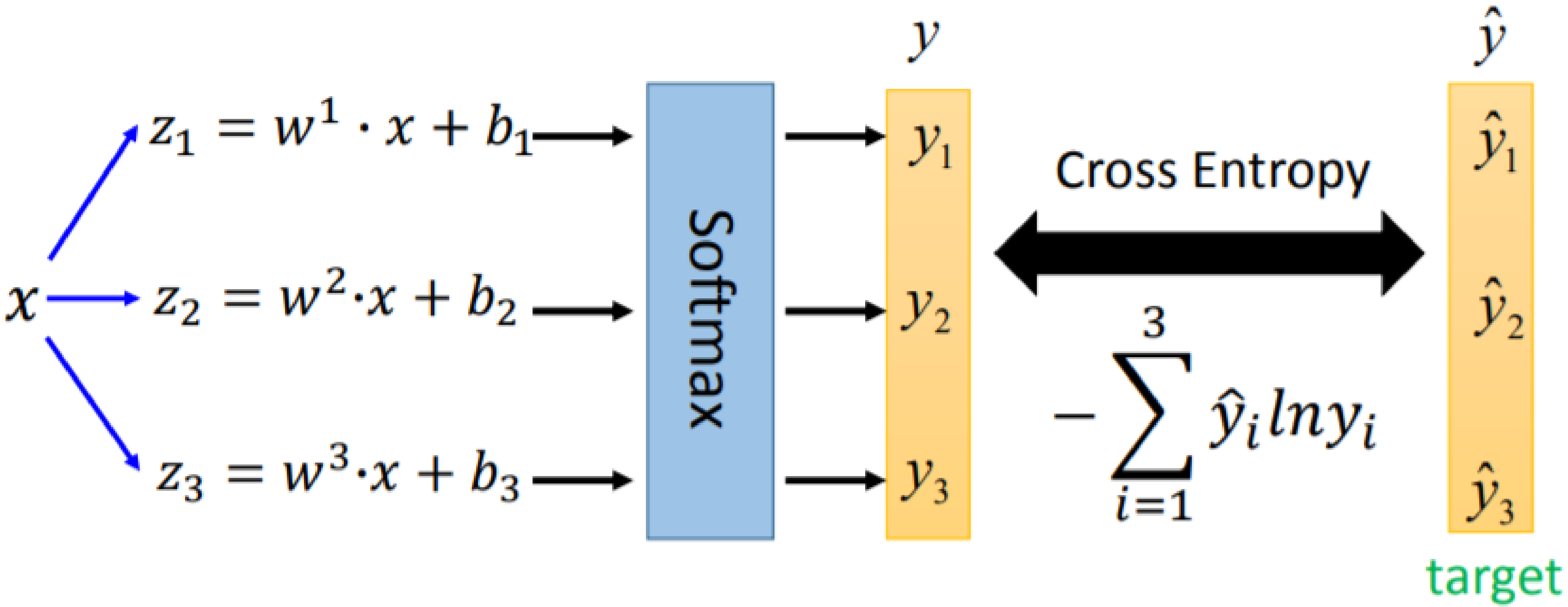
$$L(f) = \frac{1}{2}\sum_n (f(x^n) - \hat{y}^n)^2$$

Cross entropy:

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n ln f(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Multi-class classification



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA