Chương 6
Logistic Regression

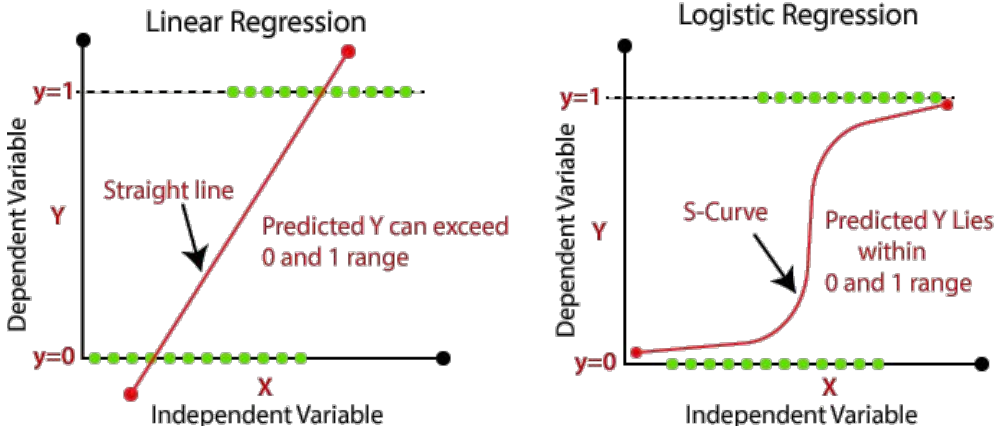
TS. Phạm Tuấn

ptuan@ute.udn.vn

Giới thiệu

Logistic regression là thuật toán **học có giám sát** rất phổ biến trong Trí tuệ nhân tạo. Khác với Linear regression, Logistic regression sẽ dự đoán các nhãn rời rạc cho các các giá trị đầu vào.

Logistic regression là một **Discriminative Classifiers** thay vì **Generative Classifiers**.



Giới thiệu

Nhiệm vụ của Logistic regression là nhận vào các điểm dữ liệu và đưa ra dự đoán xác suất của một nhãn tương ứng với điểm dữ liệu.

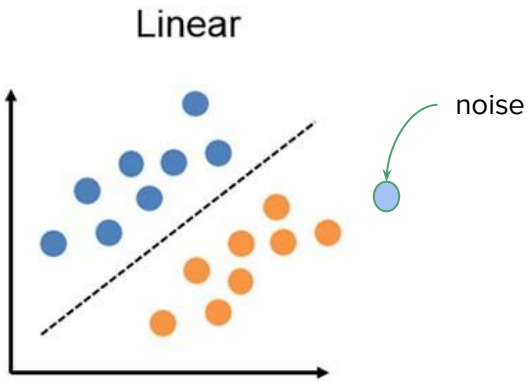
Dữ liệu đầu vào **Nhãn**

Date	Location	MinTemp	MaxTemp	Rainfall	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Temp9am	Temp3pm	RainToday	RainTomorrow
01/12/2008	Albury	13.4	22.9	0.6	71	22	1007.7	1007.1	16.9	21.8	No	No
02/12/2008	Albury	7.4	25.1	0	44	25	1010.6	1007.8	17.2	24.3	No	No
03/12/2008	Albury	12.9	25.7	0	38	30	1007.6	1008.7	21	23.2	No	No
04/12/2008	Albury	9.2	28	0	45	16	1017.6	1012.8	18.1	26.5	No	No
05/12/2008	Albury	17.5	32.3	1	82	33	1010.8	1006	17.8	29.7	No	No
06/12/2008	Albury	14.6	29.7	0.2	55	23	1009.2	1005.4	20.6	28.9	No	No
07/12/2008	Albury	14.3	25	0	49	19	1009.6	1008.2	18.1	24.6	No	No

Linear regression cho phân loại ?

Ở chương trước, chúng ta đã áp dụng Linear regression cho bài toán hồi quy, tuy nhiên Linear regression có những nhược điểm mà không thể dùng cho bài toán phân loại:

- Nhạy cảm với nhiễu => kết quả phân loại sai
- Giá trị đầu ra liên tục và từ (-∞, +∞) => khó khăn chọn ngưỡng để phân loại



Logistic function

Chúng ta đặt hàm *z* với:

$$z = wx + b$$

Với *x* là dữ liệu đầu vào, *w* được gọi là trọng số, và *b* là bias. Ở phương trình trên không có điều kiện ràng buộc nào để *z* nằm trong khoảng từ 0 đến 1. Giá trị của z có thể (-∞, +∞). **Chú ý *w* là tham số/biến số quan trọng mà chúng ta phải cần tìm**.

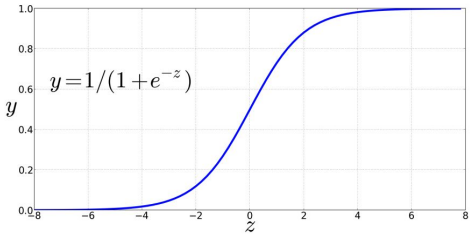
Để tạo một phép tính xác suất cũng như ràng buộc *z* nằm trong khoảng từ 0 đến 1, chúng ta dùng hàm sigmoid function như sau:

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

Logistic function

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

Hàm sigmoid đó có đồ thị như sau:



Hàm Logistic này sẽ nhận số thực và cho giá trị đầu ra nằm ở khoảng (0, 1). Chúng ta nhận thấy rằng hàm này gần như tuyến tính tại z=0.

Hàm mất mát (phương pháp 1)

Ta xét hàm mất mát tại 2 điểm đặc biệt như sau: Tại nhãn 0 và 1,

Tại nhãn 1, ta muốn hàm mất mát có giá trị càng nhỏ khi hàm logistic gần đến 1.

Tại nhãn 0, ta muốn hàm mất mát có giá trị càng nhỏ khi hàm logistic gần đến 0.

Chúng ta có 2 hàm số thoả mãn điều kiện trên:

$$loss(\sigma(z), y) = -\log(\sigma(z))$$

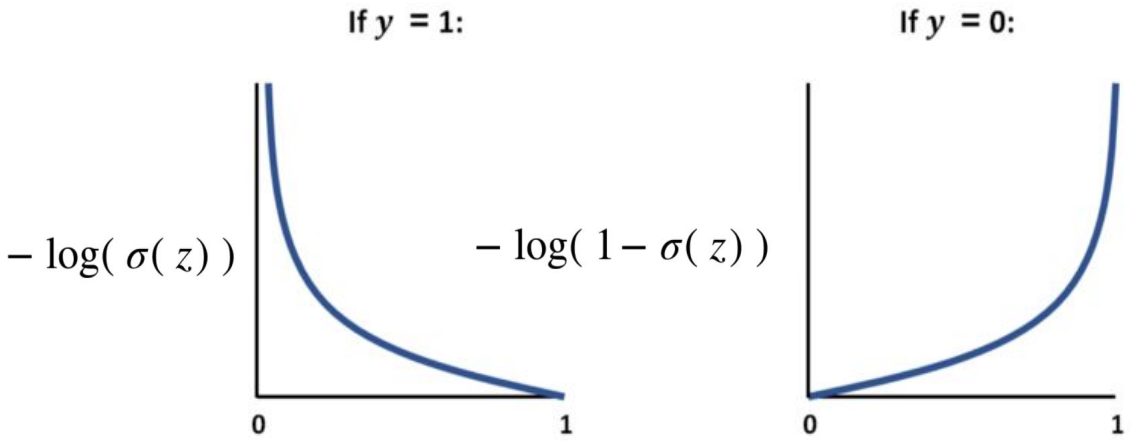
Khi y = 1

$$loss(\sigma(z), y) = -\log(1 - \sigma(z))$$

Khi y = 0

Hàm mất mát (phương pháp 1)

Trục đồ thị của hai hàm số trên như sau:



Hàm mất mát (phương pháp 1)

Chúng ta dùng một trick ở đây để kết hợp hai hàm mất mát đặc biệt ở trên để tạo ra hàm mất mát tổng quát:

$$loss(\sigma(z), y) = -y\log(\sigma(z)) - (1-y)\log(1-\sigma(z))$$

Hàm số trên được viết gọn thành như sau:

$$loss(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Hàm mất mát (phương pháp 2)

Chúng ta đưa một biến ***x*, tức là giá trị đầu vào**. Chúng ta sẽ luôn mong muốn **xác suất dự đoán vào nhãn chân trị *y* luôn luôn là lớn nhất** => maximize($p(y|x)$).

Do đây là dự đoán nhị phân, cho nên chúng ta có:

$$p(y|x) = \begin{cases} \hat{y} & if \ \ y = 1 \\ 1 - \hat{y} & if \ \ y = 0 \end{cases}$$

Để biển diễn hàm trên với một hàm duy nhất, chúng ta viết lại như sau:

$$p(y|x) = \hat{y}^{y}(1 - \hat{y})^{1-y}$$

Ref: LOGISTIC REGRESSION Url: https://web.stanford.edu/~jurafsky/slp3/5.pdf

Hàm mất mát (phương pháp 2)

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

Từ phương trình trên ta lấy log cả 2 bên:

$$\log(p(y|x)) = y\log\hat{y} + (1-y)\log(1-\hat{y})$$

Ở đây, chúng ta đang cố gắng maximize($p(y|x)$), tương đương với việc maximize($\log(p(y|x))$). Tuy nhiên, hàm loss là hàm minimize, cho nên ta biến đổi như sau để có hàm loss:

$$loss(\hat{y}, y) = -\log(p(y|x)) = -\left[y\log\hat{y} + (1-y)\log(1-\hat{y})\right]$$

Chúng ta thấy rằng giá trị hàm loss sẽ nhỏ nếu mô hình dự đoán gần đúng, và lớn nếu mô hình còn đang lưỡng lự.

Hàm loss ở trên được gọi là **Cross entropy**.

Đạo hàm của hàm mất mát

$$L(\hat{y}, y) = -\log(p(y|x)) = -\left[y\log\hat{y} + (1-y)\log(1-\hat{y})\right]$$

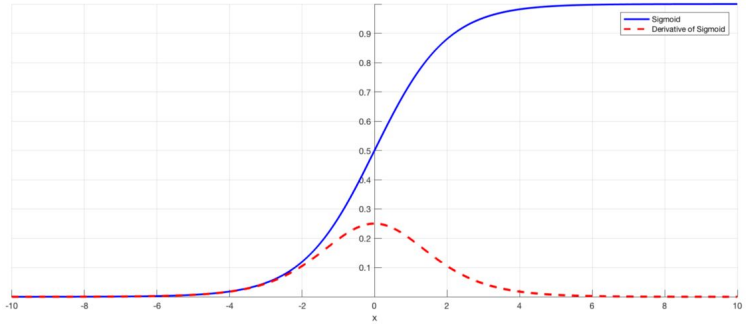
Ta dùng **Chain rule** để tính đạo hàm *L* theo w:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial w} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} = \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial w} = \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \frac{\partial \hat{y}}{\partial z} x$$

Đạo hàm của hàm mất mát



Thật may mắn, đạo hàm của hàm sigmoid chính là mẫu số của đạo hàm:

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$$

Do đó hàm mất mát sẽ từ $\frac{\partial L}{\partial w} = \left(\frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \right) \frac{\partial \hat{y}}{\partial z} x$ chuyển thành: $\frac{\partial L}{\partial w} = (\hat{y} - y)\, x$

An example

Let's have an example. Suppose we are doing binary sentiment classification on movie review text, and we would like to know whether to assign the sentiment class + or − to a review document doc. We'll represent each input observation by the 6 features $x_1 ... x_6$ of the input shown in the following table;

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

An example

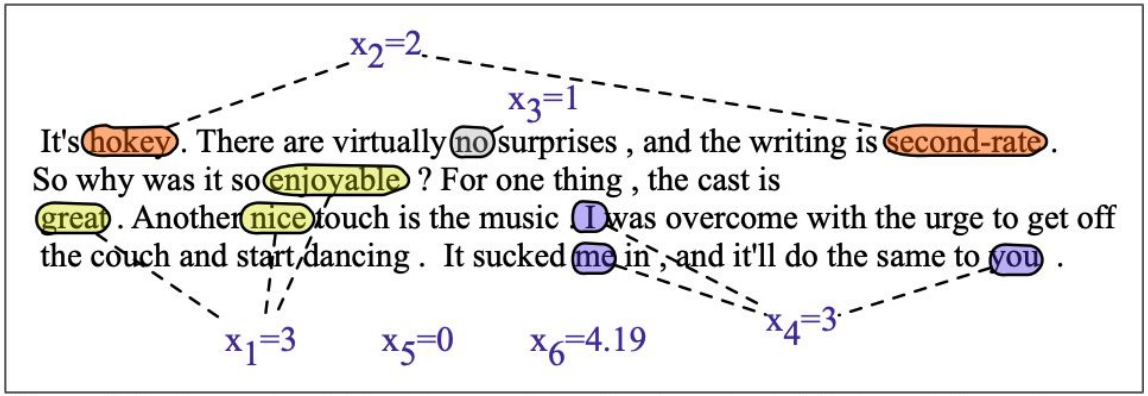


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Let's assume for the moment that we've already learned a real-valued weight for each of these features, and that the 6 weights corresponding to the 6 features are [2.5,−5.0,−1.2,0.5,2.0,0.7], while b = 0.1

An example

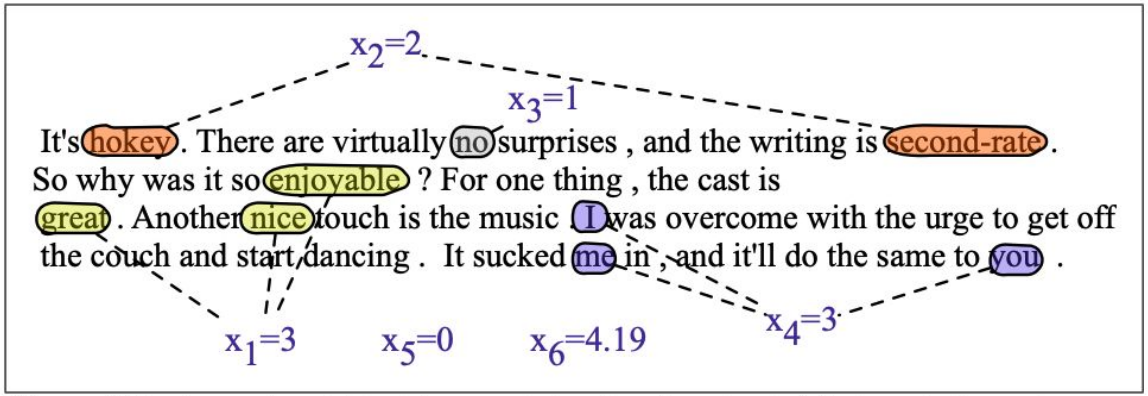


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Let's assume for the moment that we've already learned a real-valued weight for each of these features, and that the 6 weights corresponding to the 6 features are [2.5,−5.0,−1.2,0.5,2.0,0.7], while b = 0.1.

The weight w_1, for example indicates how important a feature the number of positive lexicon words (great, nice, enjoyable, etc.) is to a positive sentiment decision, while w_2 tells us the importance of negative lexicon words.

An example

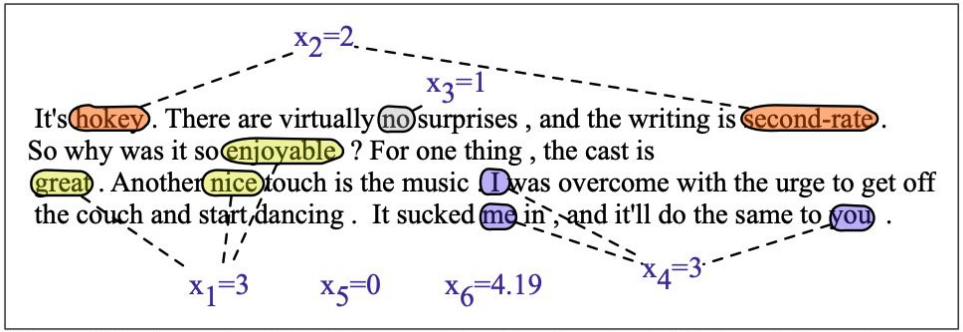


Figure 5.2 A sample mini test document showing the extracted features in the vector x.

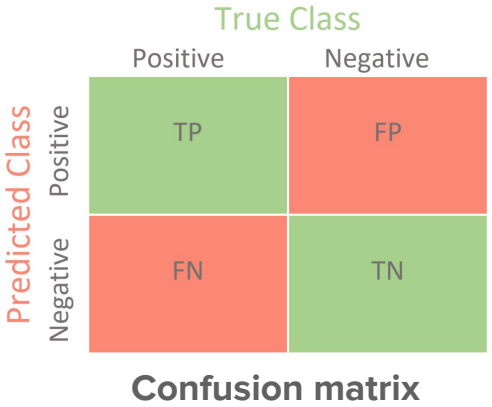
Xác suất dự đoán của bình luận đó là tích cực

Given these 6 features and the input review x, $P(+|x)$ and $P(-|x)$ can be computed using Eq. 5.5:

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70 \qquad (5.7) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

$$
p(y|x) = \begin{cases} \hat{y} & \text{if } y = 1 \\ 1 - \hat{y} & \text{if } y = 0 \end{cases}
$$

Đánh giá bài toán phân loại



Accuracy: (TP+TN)/(TP+TN+FP+FN)

Precision of positive class = TP/(TP+FP)

Recall of positive class= TP/(TP+FN)

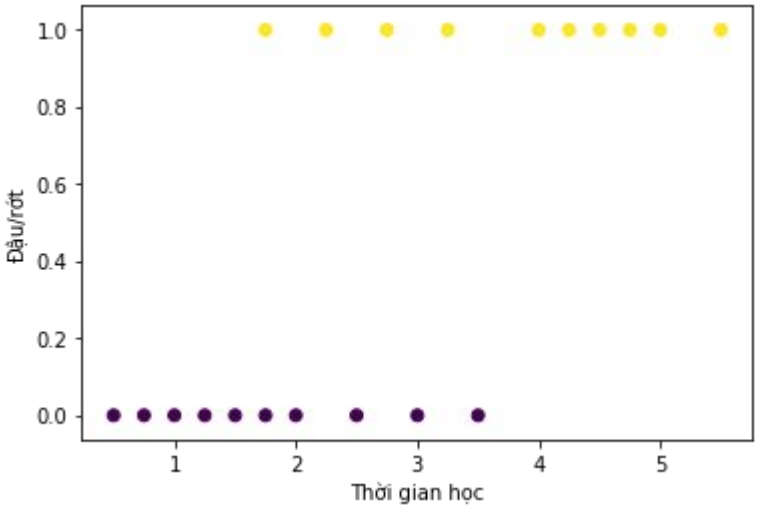
$$F_1 - score = 2\frac{Precision \times Recall}{Precision + Recall}$$

Ví dụ Python

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)

X = np.array([[0.50, 0.75, 1.00, 1.25, 1.50,
1.75, 1.75, 2.00, 2.25, 2.50,
            2.75, 3.00, 3.25, 3.50, 4.00, 4.25,
4.50, 4.75, 5.00, 5.50]])
y = np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 1, 1, 1, 1, 1])

plt.scatter(X, y, c = y)
plt.xlabel('Thời gian học')
plt.ylabel('Đậu/rớt')
plt.show()
```



Ví dụ Python

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0).fit(X.reshape(-1,1), y)
clf.predict_proba(np.array([0.75]).reshape(-1,1))
```

```
array([[0.90704455, 0.09295545]])
```

```
from sklearn.metrics import confusion_matrix,
plot_confusion_matrix
y_pre = clf.predict(X.reshape(-1,1))
cm = confusion_matrix(y, y_pre)
cm
```

```
array([[8, 2],
       [2, 8]])
```

Ví dụ Python

```
def sigmoid(s):
    return 1/(1 + np.exp(-s))

def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
    w = [w_init]
    it = 0
    N = X.shape[1]
    d = X.shape[0]
    count = 0
    check_w_after = 20
    while count < max_count:
        # mix data
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = y[i]
            zi = sigmoid(np.dot(w[-1].T, xi))
            w_new = w[-1] + eta*(yi - zi)*xi
            count += 1
            # stopping criteria
            if count%check_w_after == 0:
                if np.linalg.norm(w_new - w[-check_w_after]) < tol:
                    return w
            w.append(w_new)
    return w
X = np.concatenate((np.ones((1, X.shape[1])), X), axis = 0)
eta = .05
d = X.shape[0]
w_init = np.random.randn(d, 1)
w = logistic_sigmoid_regression(X, y, w_init, eta)
print(w[-1])
```

```
[[-4.092695  ]
 [ 1.55277242]]
```

```
print(sigmoid(np.dot(w[-1].T, [1, 0.75])))
```

```
[0.05078108]
```

Any question ?