Data-intensive space engineering

Lecture 3

Carlos Sanmiguel Vila

Logistic regression

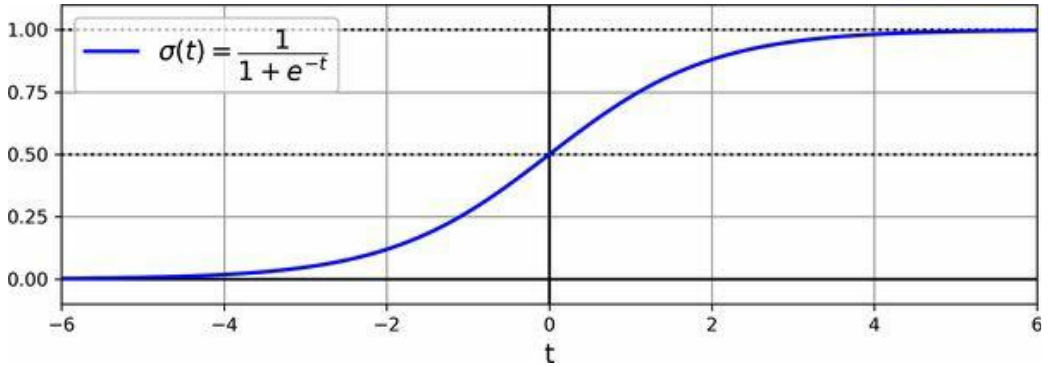
Prediction

- Logistic regression is used to estimate the probability that an instance belongs to a particular class (out of two classes, e.g., an email is spam or not)
- If the estimated probability is greater than 50%, the model predicts that the instance belongs to the positive class (labeled “1”) otherwise, the model predicts that the instance belongs to the negative class (labeled “0”)
- Like linear regression, logistic regression computes a weighted sum of the input features
- **Difference**: It applies the **logistic function σ** to the output result

$$\hat{p} = h(x) = \sigma(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d)$$

Logistic function

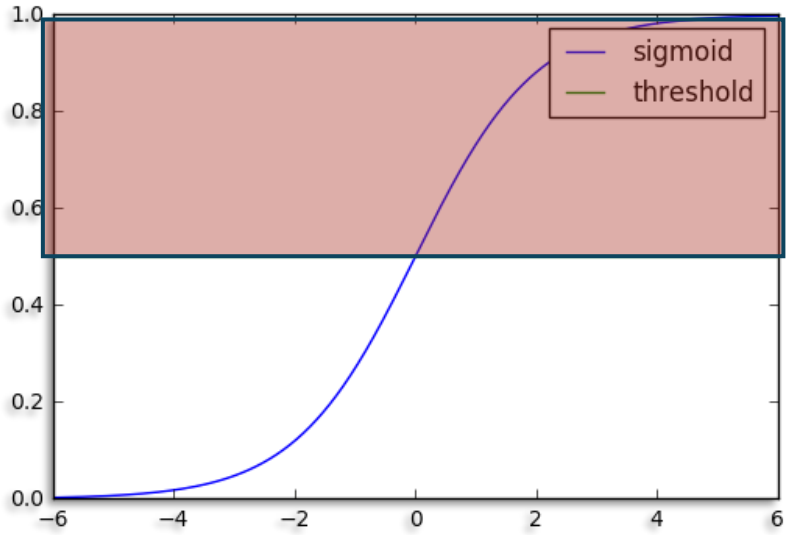
- The logistic or sigmoid function outputs a number between and as follows:



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

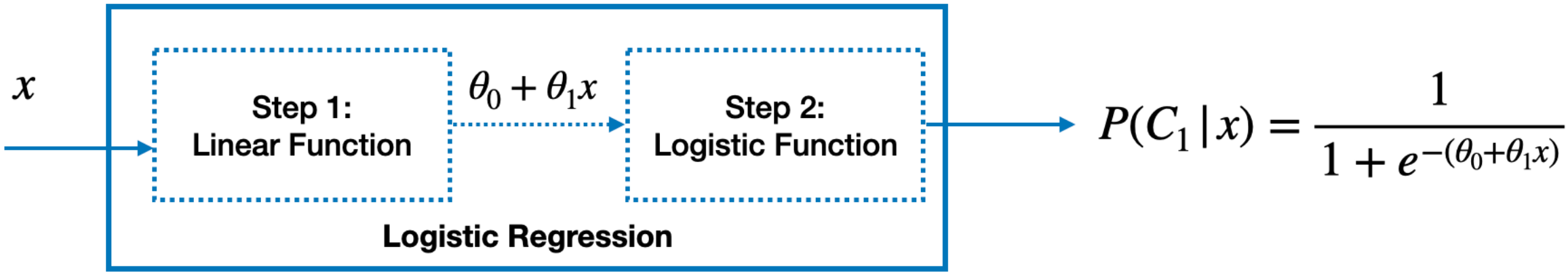
- Using a 50% threshold probability

$\hat{y} = 0$ if $\hat{p} < 0.5$ and if $\hat{p} > 0.5$ $\hat{y} = 1$



Logistic function

- Assuming a two-class classification problem C_1, C_2



- We want to find $P(C_i|x)$

$$P(C_1|x) + P(C_2|x) = 1 \rightarrow P(C_1|x) = 1 - P(C_2|x)$$

Decision boundary

Consider, $\hat{p} = h(x) = \sigma(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2)$

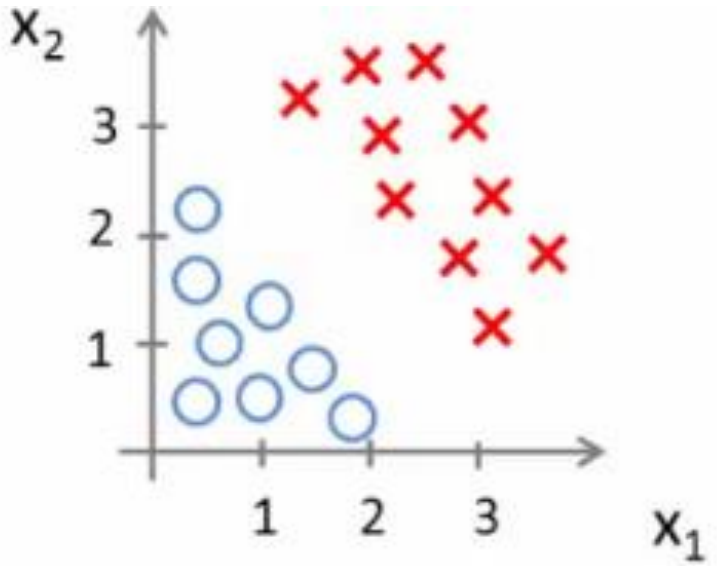
For example, $\theta = [-3,1,2]$

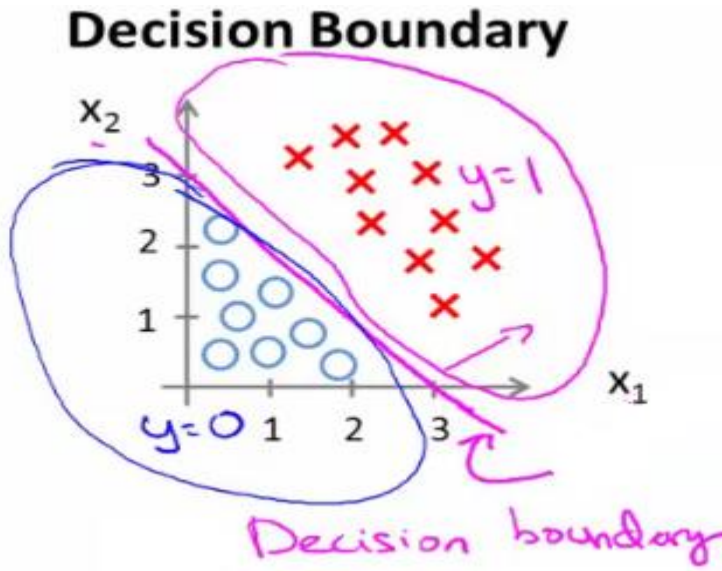
•What does this mean?

We predict $\hat{y}$ = 1 if $-3x_0 + 1x_1 + 1x_2 >= 0$

•We can also re-write this as If (x_1 + x_2 >= 3) then we predict $\hat{y}$ = 1

- If we plot x_1 + x_2 = 3 we graphically plot our **decision boundary**





Decision boundary

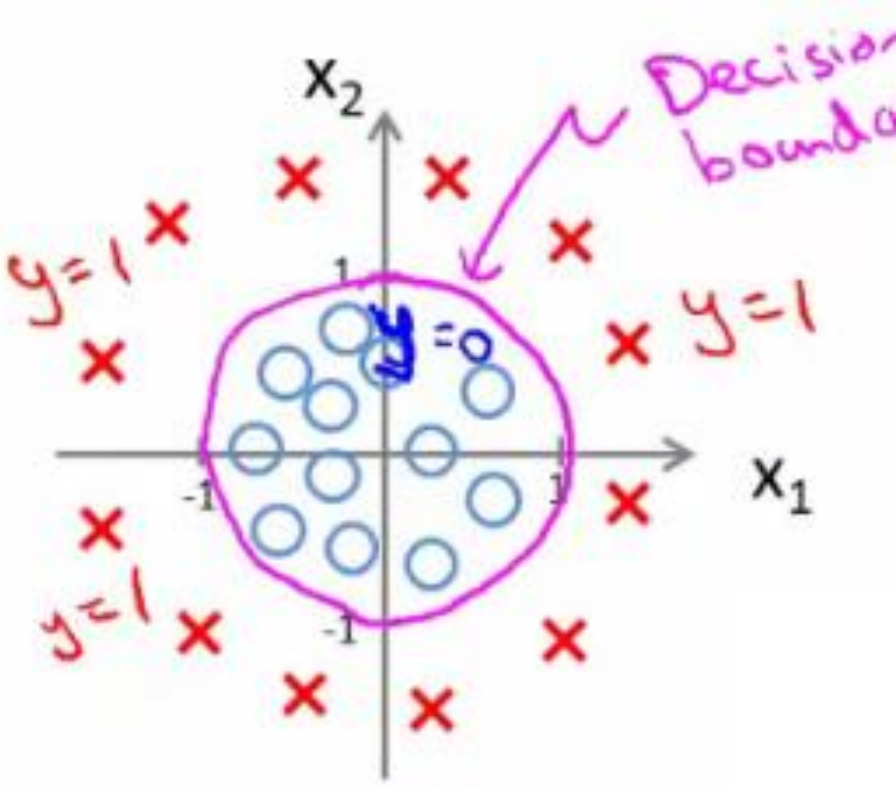
Using non-linear functions,

$$\hat{p} = h(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

For example, $\theta = [-1,0,0,1,1]$

We predict $\hat{y}$ = 1 if:

- $-1 + 1{x_1}^2 + {x_2}^2 >= 0$
- Or ${x_1}^2 + {x_2}^2 = 1$



Training and cost function

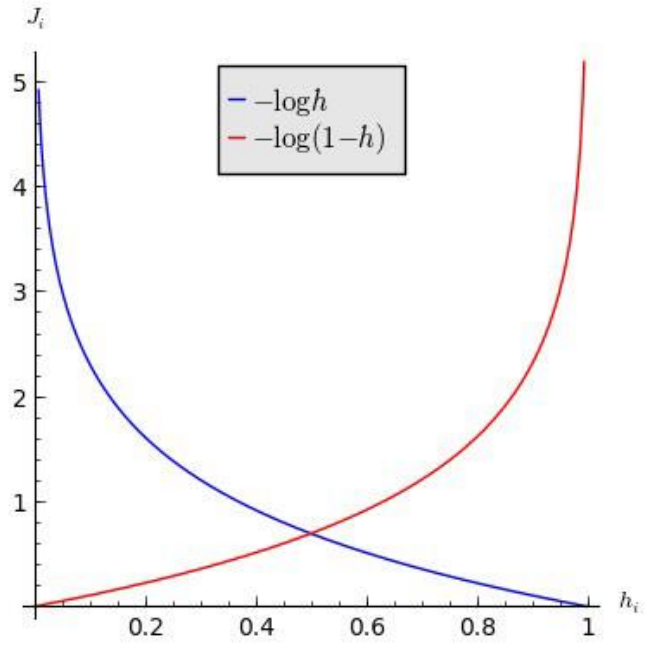
- To perform the training, we need a cost or loss function other than MSE for logistic regression: MSE with sigmoid is non-convex and gradient descent will get trapped in local minima

- In general terms $J(\theta)$ can be expressed as

$$J(\theta) = \frac{1}{n}\sum_{i=0}^{n} cost(h(x^{(i)}), y^{(i)})$$

- For the logistic regression we can use

$$cost(h(x), y) = \begin{cases} -\log(h(x)) \; if \; y = 1 \\ -\log(1 - h(x)) \; if \; y = 0 \end{cases}$$



Training and cost function

- Note that for a two-class problem $y \in \{0,1\}$

$$cost(h(x), y) = -y\ log\big(h(x)\big) - (1-y)\log(1-h(x))$$

- The cost function over the whole training set

$$J(\theta) = -\frac{1}{n}\sum_{i=0}^{n}[y^{(i)}\log\Big(h\big(x^i\big)\Big) + \big(1-y^i\big)\log(1-h(x^i))]$$

- The bad news is that there is no known closed-form solution, but the good news is that this cost function is convex, and therefore we can use Gradient Descent to find the global minimum

Logistic regression

- The log loss was not just pulled out of a hat. It can be shown mathematically (using Bayesian inference) that minimizing this loss will result in the model with the *maximum likelihood* of being optimal, assuming that the instances follow a Gaussian distribution around the mean of their class. When you use the log loss, this is the implicit assumption you are making. The more wrong this assumption is, the more biased the model will be.

- Similarly, when we used the MSE to train linear regression models, we were implicitly assuming that the data was purely linear, plus some Gaussian noise. So, if the data is not linear (e.g., if it's quadratic) or if the noise is not Gaussian (e.g., if outliers are not exponentially rare), then the model will be biased.

Softmax regression

- The logistic regression model can be generalized to support multiple classes
- Let K be the number of classes and compute a score $s_k(x)$ for each $k \in \{1, \dots, K\}$

$$s_k(x) = \mathrm{x}^{\mathrm{T}} \theta^k$$

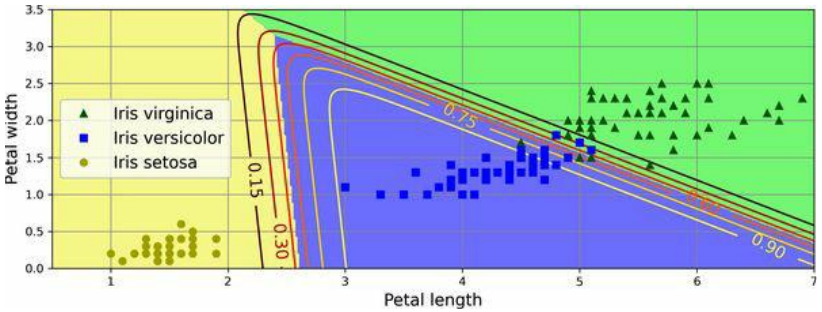
- Estimate the probability that the instance x belongs to class k

$$\hat{p}_k = \sigma\big(s(x)\big)_K = \frac{\exp\big(s_k(x)\big)}{\sum_{j=1}^{k} \exp(s_j(x))}$$

Softmax regression

- Softmax regression predicts the class with the highest estimated probability

$$\hat{p} = argmax\ \sigma\big(s(x)\big)_k = argmax\ s_k(x) = argmax\ x^T\theta^T$$



- • Cross entropy cost function

$$J(\theta) = -\frac{1}{n}\sum_{i=0}^{n}\sum_{j=1}^{K} y_k^{(i)}\log(\hat{p}_k^{(i)})$$

- where $y_k^{(i)} \in \{0,1\}$ is the target probability

Confusion Matrix

- Confusion matrix:
  - Defines the performance of a classification model
  - Represented as a table

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive) b: FN (false negative)

c: FP (false positive) d: TN (true negative)

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm_df=pd.DataFrame(cm, index=['Luxury','Economy'], columns=['Luxury','Economy'])
cm_df
```

	Luxury	Economy
Luxury	51	2
Economy	7	58

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred, normalize='true')
cm_df=pd.DataFrame(cm, index=['Luxury','Economy'], columns=['Luxury','Economy'])
cm_df
```

Normalized on rows

	Luxury	Economy
Luxury	0.962264	0.037736
Economy	0.107692	0.892308

Compute Accuracy from Confusion Matrix

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

e.g., The percentage of emails that are being labeled correctly

Issue of Using Accuracy

- Does not deal with imbalance issue well
  - E.g., detecting spam emails where 99% are normal and 1% are spams
  - Given a model that labels every email as normal, the model has 99% accuracy – but is that good? Think about detection of dangerous asteroids
- Lots of classification problems where the classes are skewed (more records from one class than another)
  - Credit card fraud
  - Covid test

Alternative Measures

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a	b
	Class=No	c	d

Need to specify what is the ”true” label

$$\text{Precision (p)} = \frac{a}{a+c}$$

e.g., The percentage of emails labeled as “spam” that are actually spam

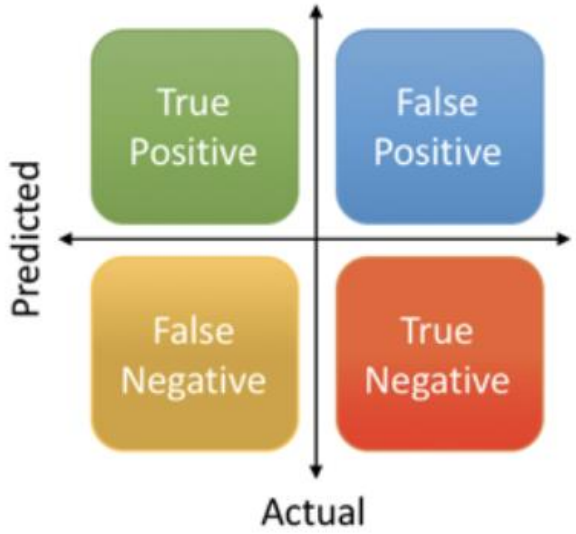
$$\text{Recall (r)} = \frac{a}{a+b}$$

e.g., The percentage of spam emails are labeled as “spam”

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

Combines Precision and Recall

Precision vs Recall



$$Precision = \frac{True\ Postive}{True\ Positive + False\ Postiive}$$

$$Recall = \frac{True\ Postive}{True\ Positive + False\ Negative}$$

- Precision is sensitive to False Positive
- Recall is sensitive to False Negative

Which one is more affordable, False Positive or False Negative?

- Detect if an asteroid is dangerous or not - High recall
- Choose stocks to invest – High precision

Precision, Recall, F1-score

```
from sklearn import metrics #Import scikit learn metrics module for accuracy calculation
```

```
print("Precision is: ", metrics.precision_score(y_test, y_pred, pos_label="Luxury"))
```

```
Precision is:  0.8405797101449275
```

```
print("Recall is: ", metrics.recall_score(y_test, y_pred, pos_label="Luxury"))
```

```
Recall is:  0.8923076923076924
```

```
print("Recall is: ", metrics.f1_score(y_test, y_pred, pos_label="Luxury"))
```

```
Recall is:  0.8656716417910447
```