Classification

Motivation

Previously, we learned about linear regression, which predicts a number. This week, you learn about classification where your output variable y can take on only one of a small handful of possible values instead of any number in an infinite range of numbers.

It turns out that linear regression is not a good algorithm for classification problems.

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values

"binary classification"

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values

"binary classification"

class = category

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes
	false	true
	0	1

useful for classification

y can only be one of two values

"binary classification"

class = category

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values

"binary classification"

class = category

false true

0 1

useful for classification

"negative class"

"positive class"

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values

"binary classification"

class = category

false true

0 1

useful for classification

"negative class"

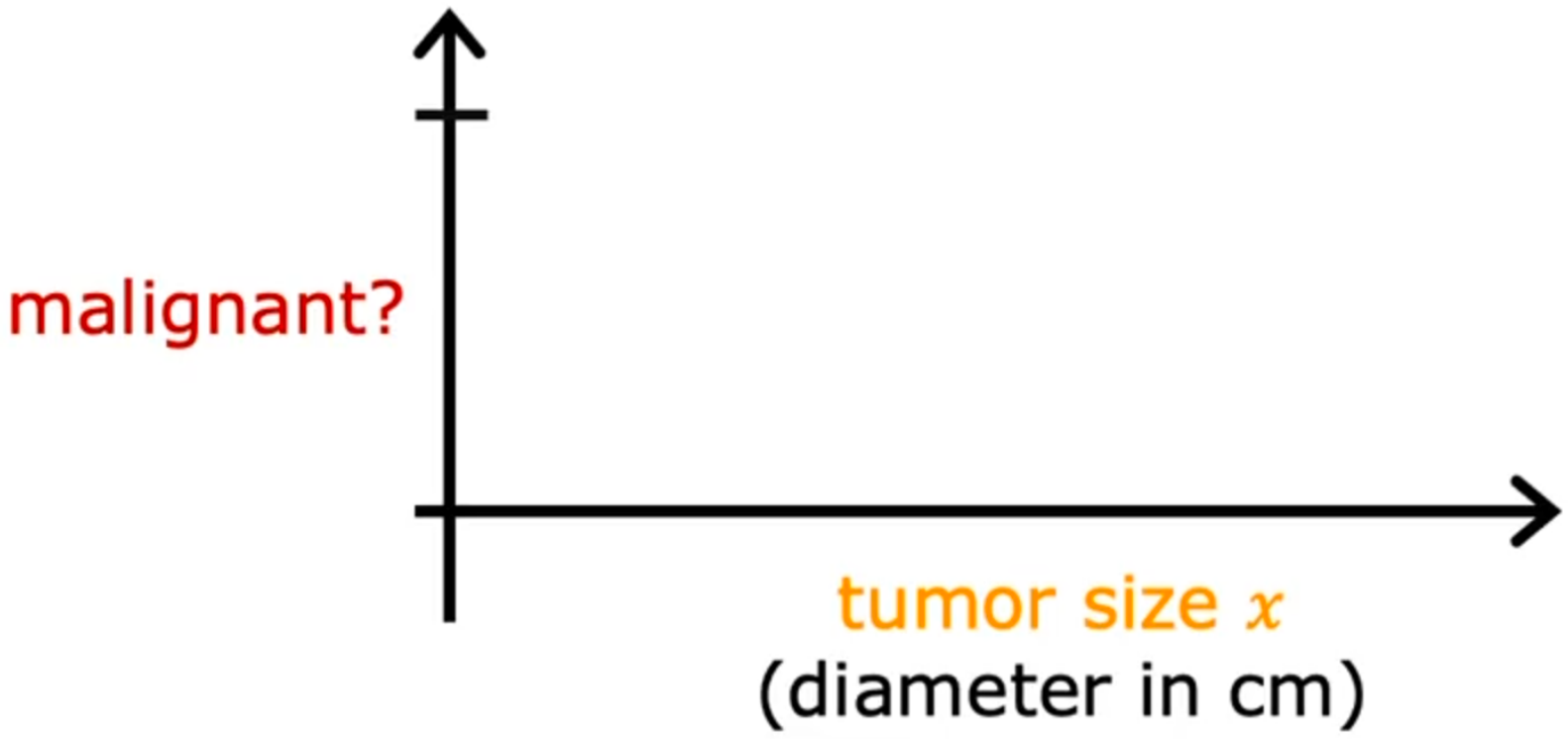
≠ "bad"

absence

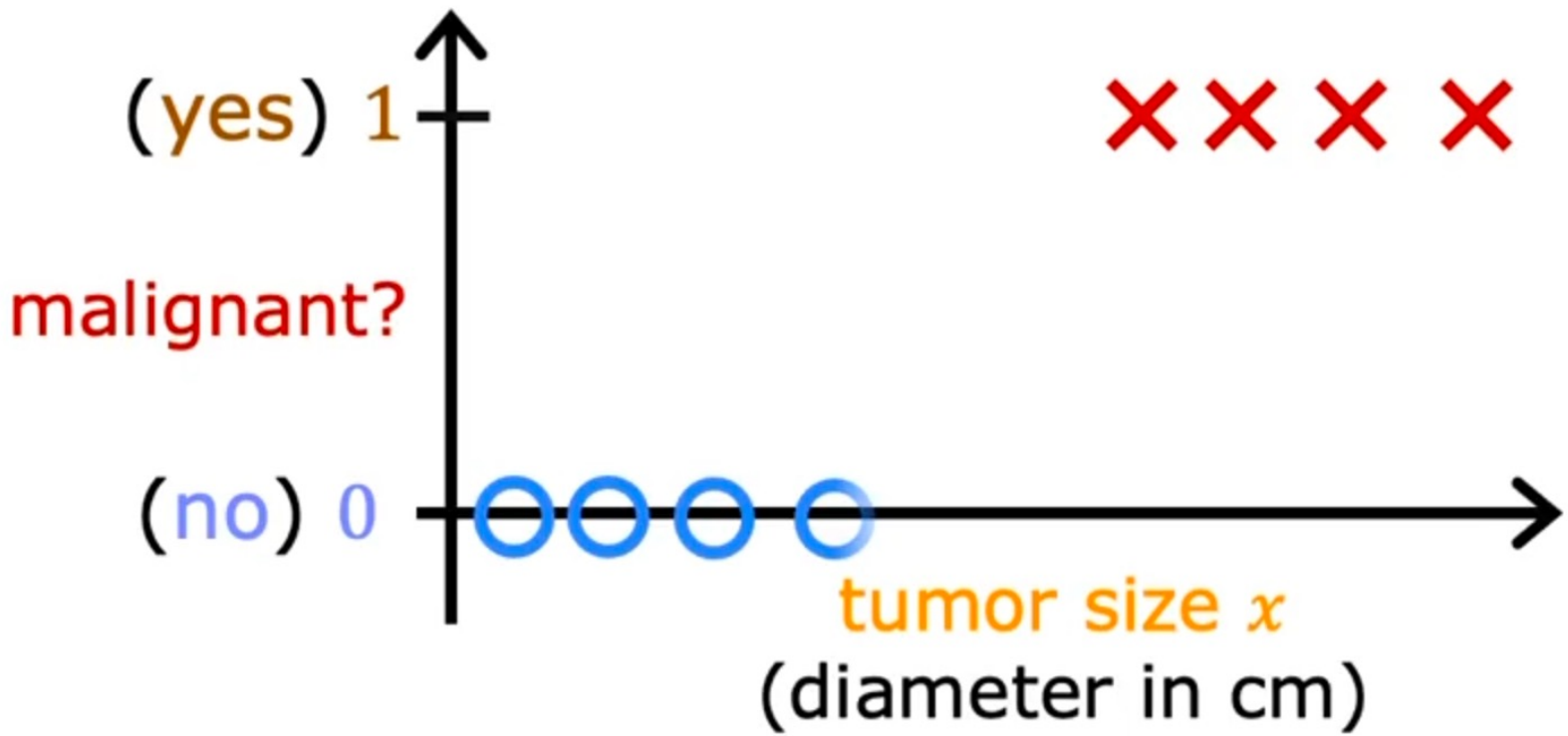
"positive class"

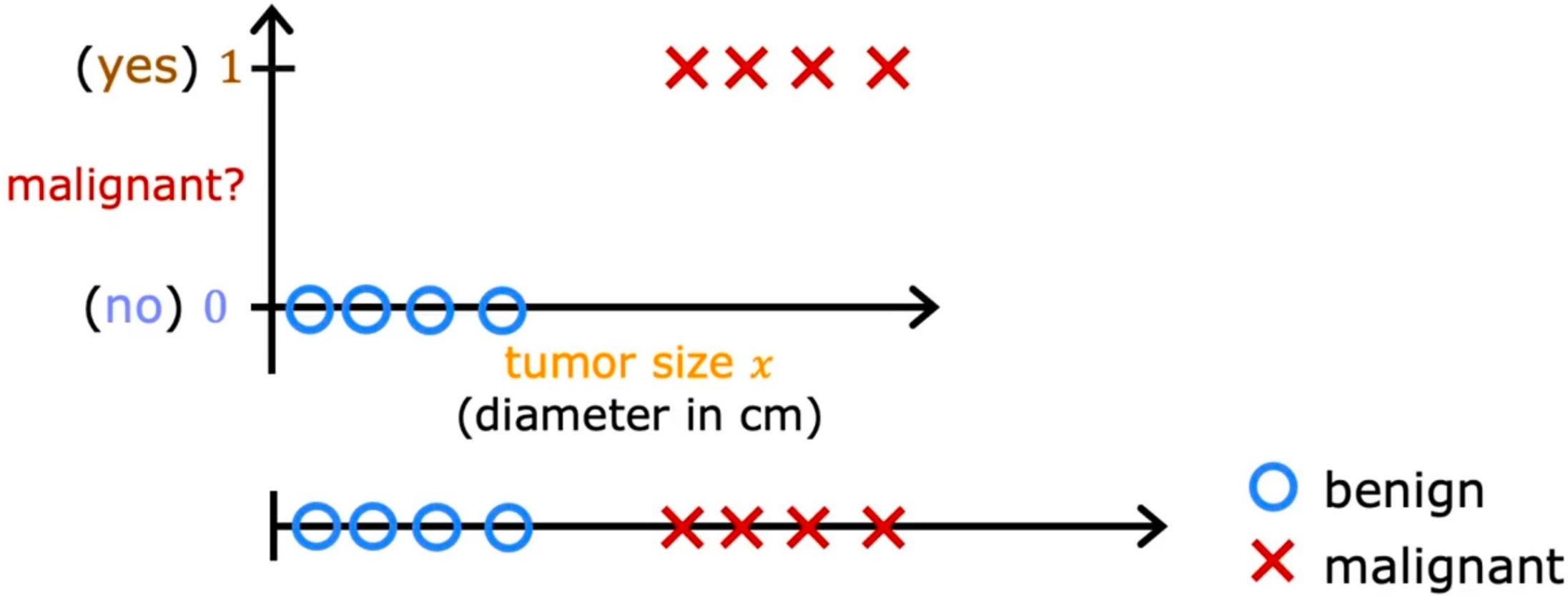
≠ "good"

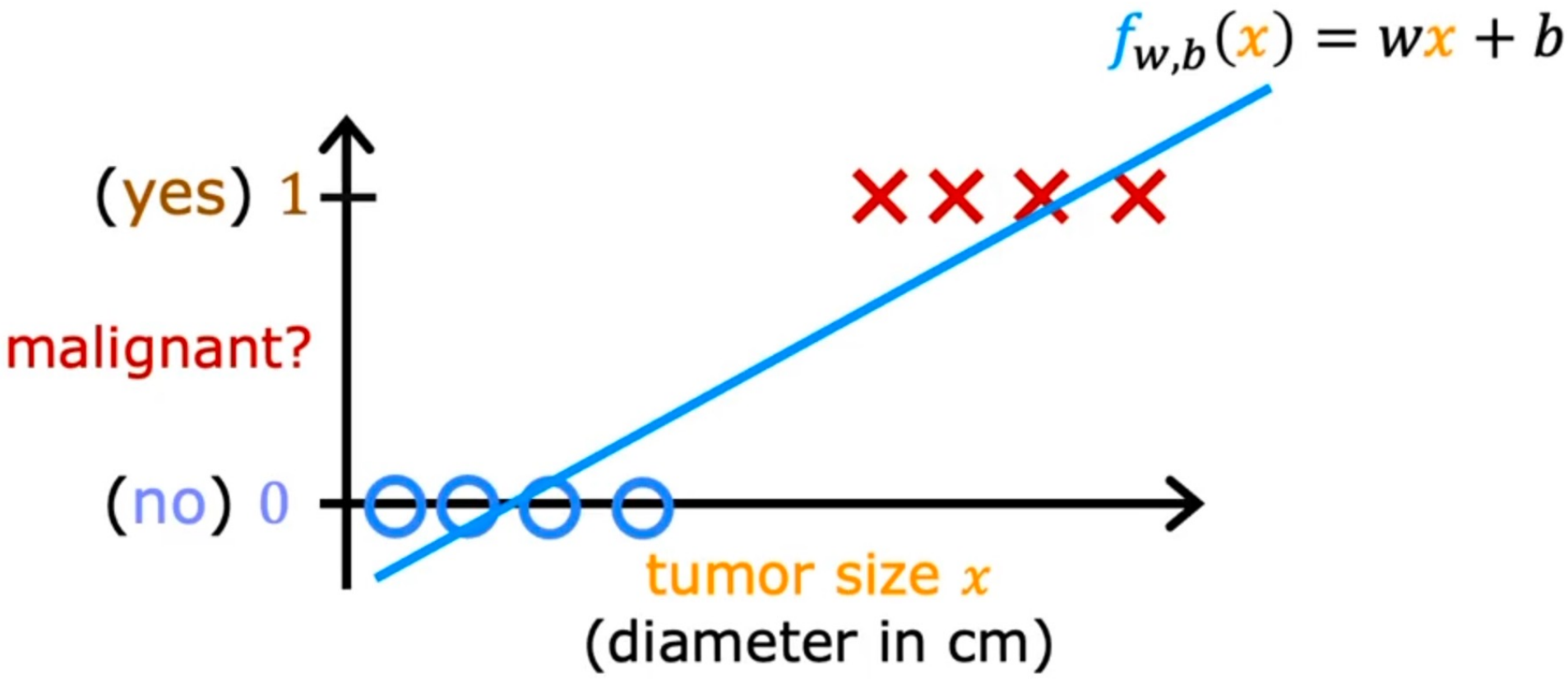
presence



Here's the example of a training set for classifying if the tumor is malignant. A class one, positive class, yes class or benign, class zero or negative class. I plotted both the tumor size on the horizontal axis as well as the label Y on the vertical axis.

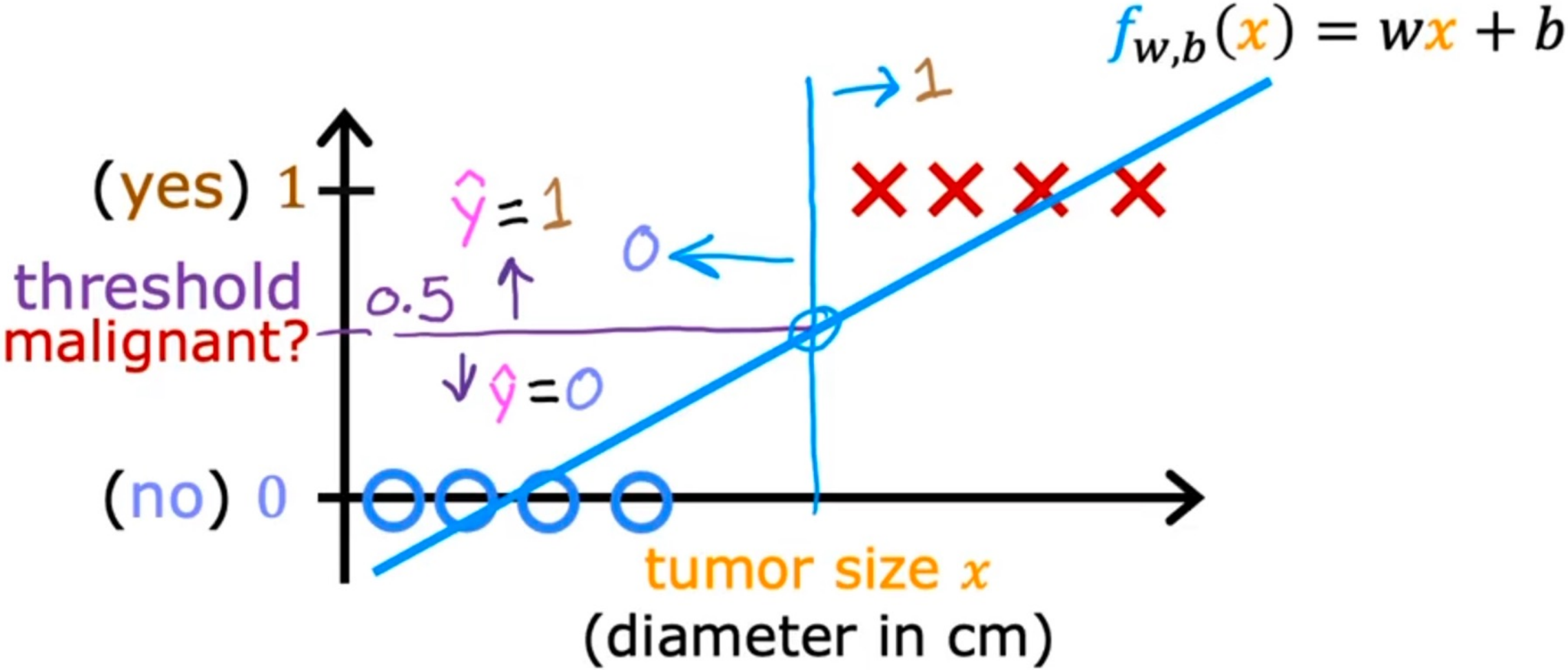

(yes) 1
malignant?
(no) 0
tumor size x
(diameter in cm)


(yes) 1
malignant?
(no) 0
tumor size x
(diameter in cm)
benign
malignant



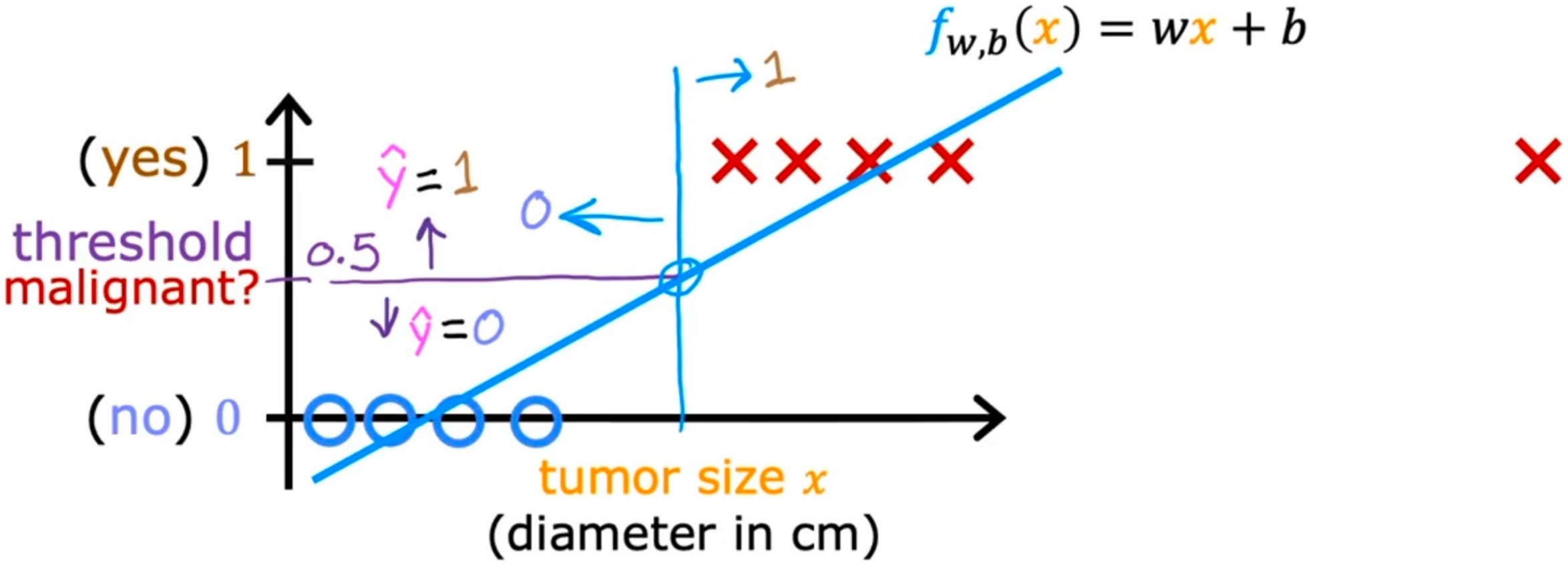
Now, one thing you could try on this training set is to apply the Linear regression and try to fit a straight line to the data. If you do that, maybe the straight line.

Linear regression predicts not just the values zero and one. But all numbers between zero and one or even less than zero or greater than one. But here we want to predict categories.



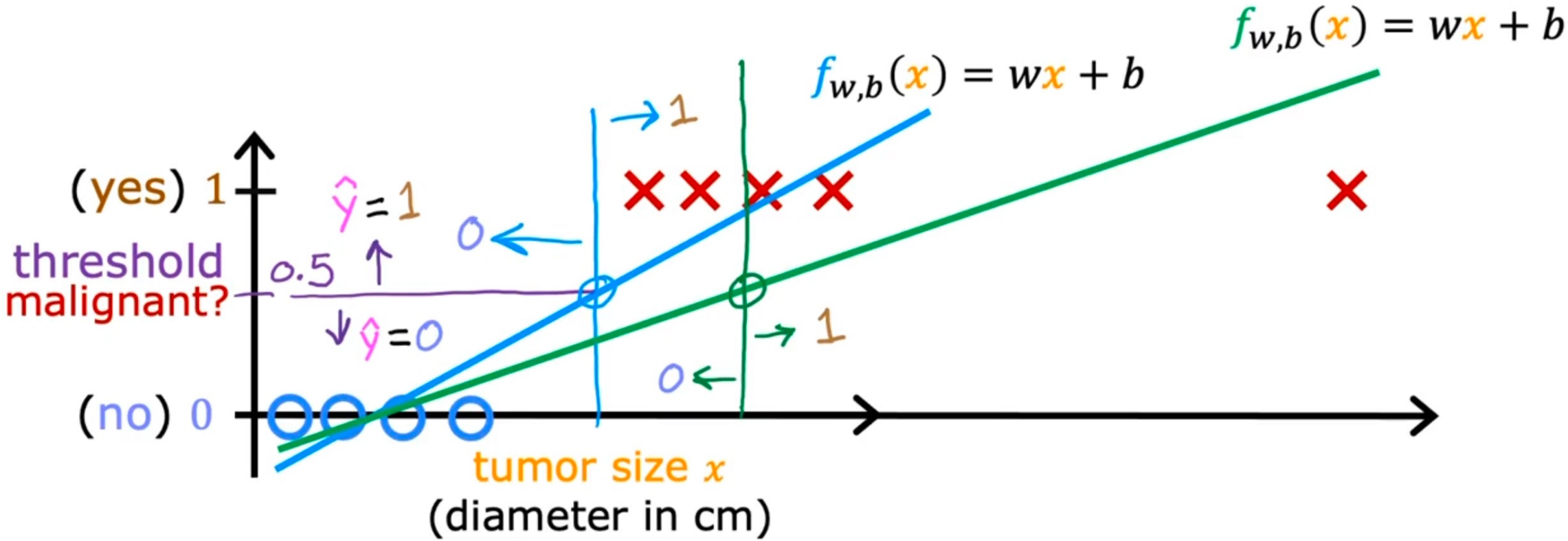
if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



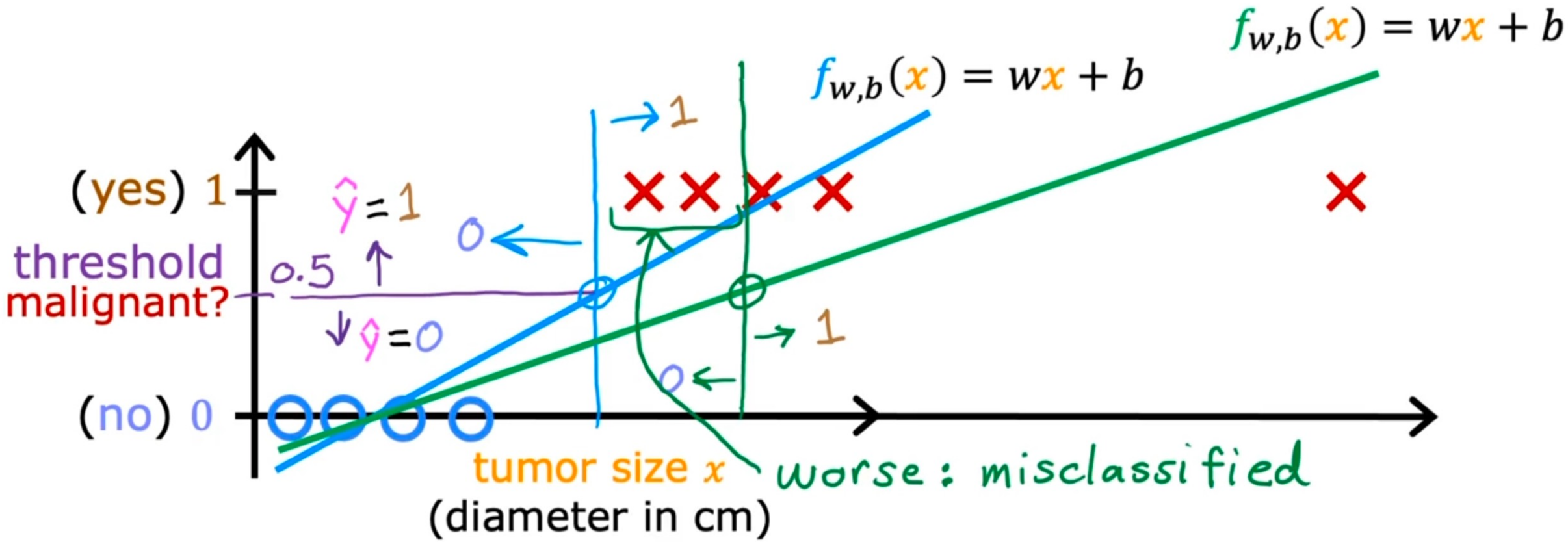
if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



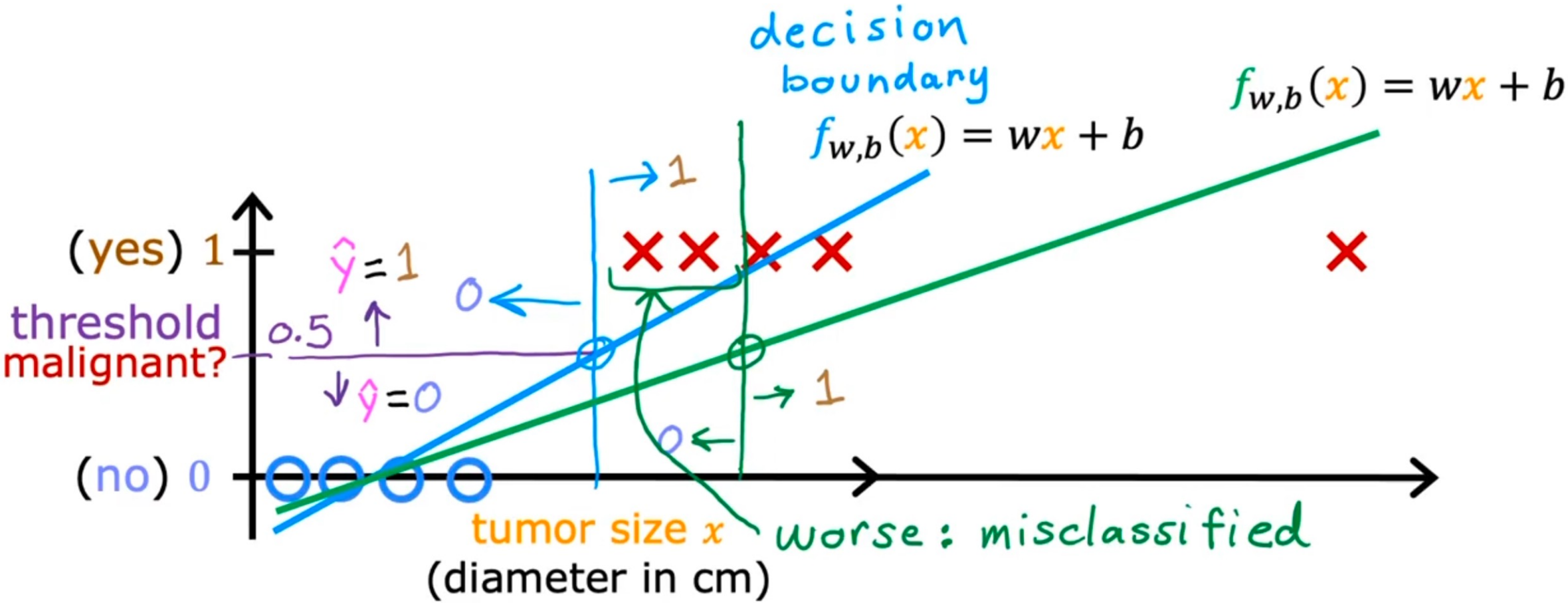
if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



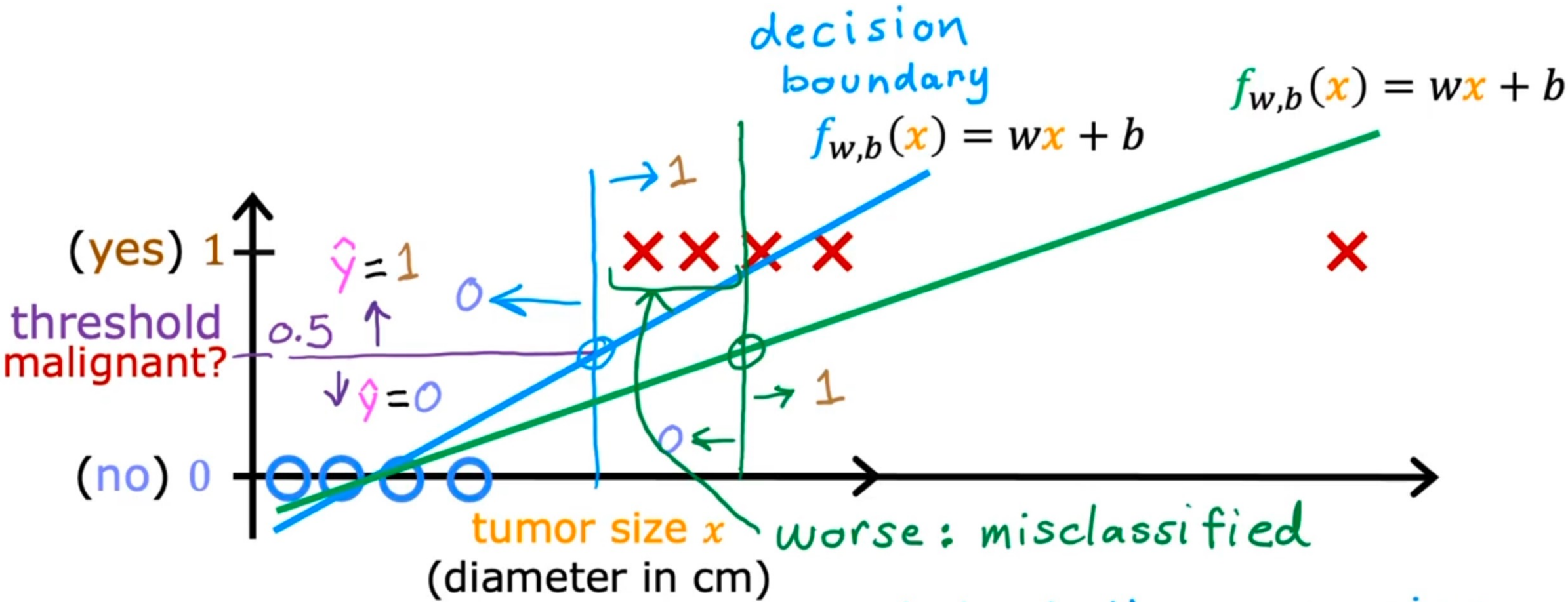
if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$

next: logistic regression

classification

jupyter Classification Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

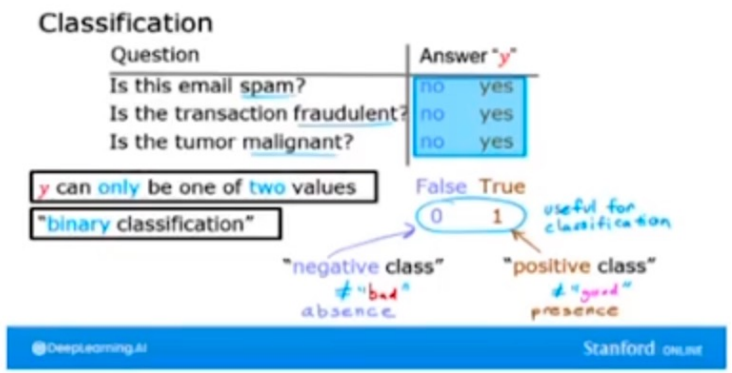
Run Code

Optional Lab: Classification using Logistic Regression

In this lab, you will contrast regression and classification.

In [1]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
from lab_utils_common import dlc, plot_data
from plt_one_addpt_onclick import plt_one_addpt_onclick
plt.style.use('./deeplearning.mplstyle')
```

Classification Problems

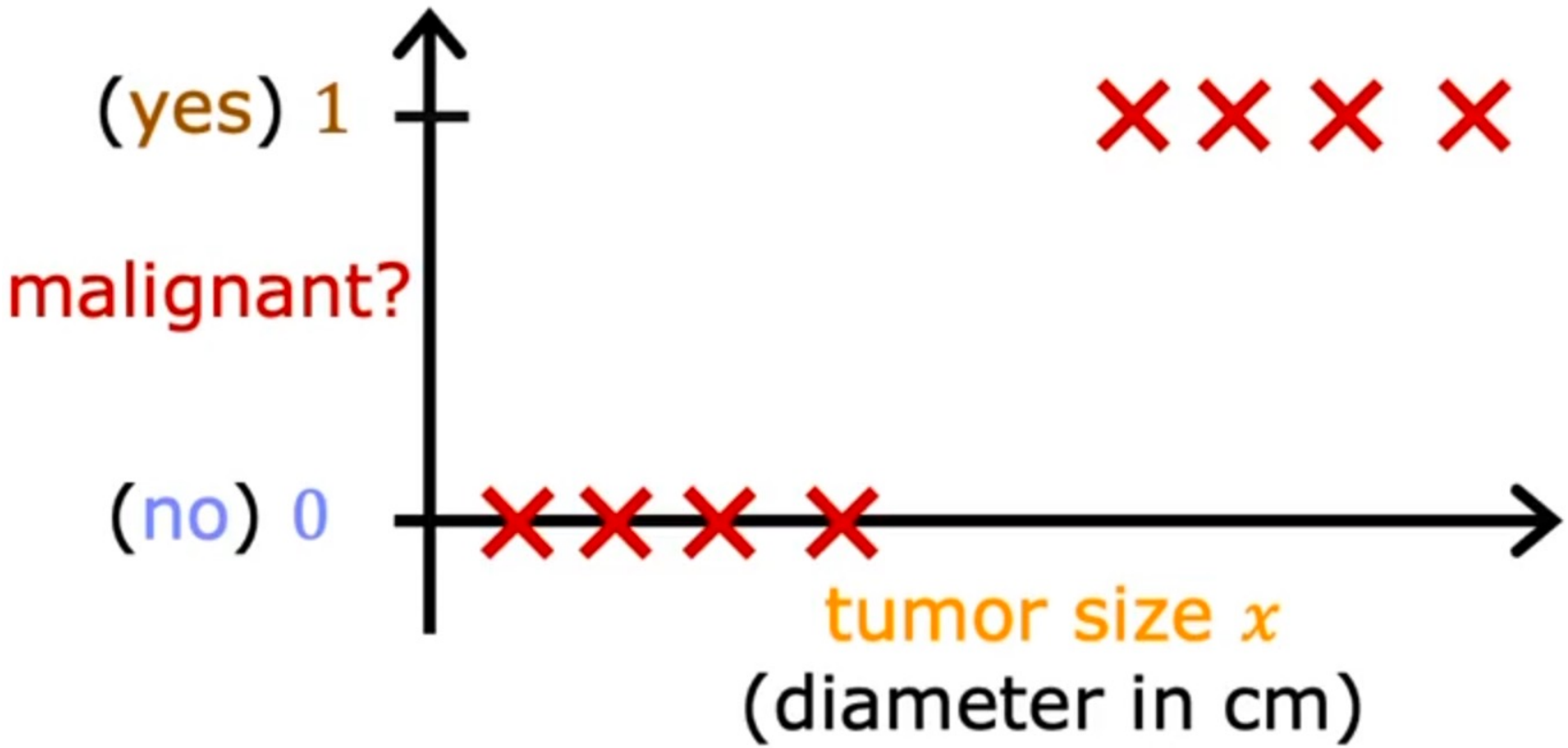


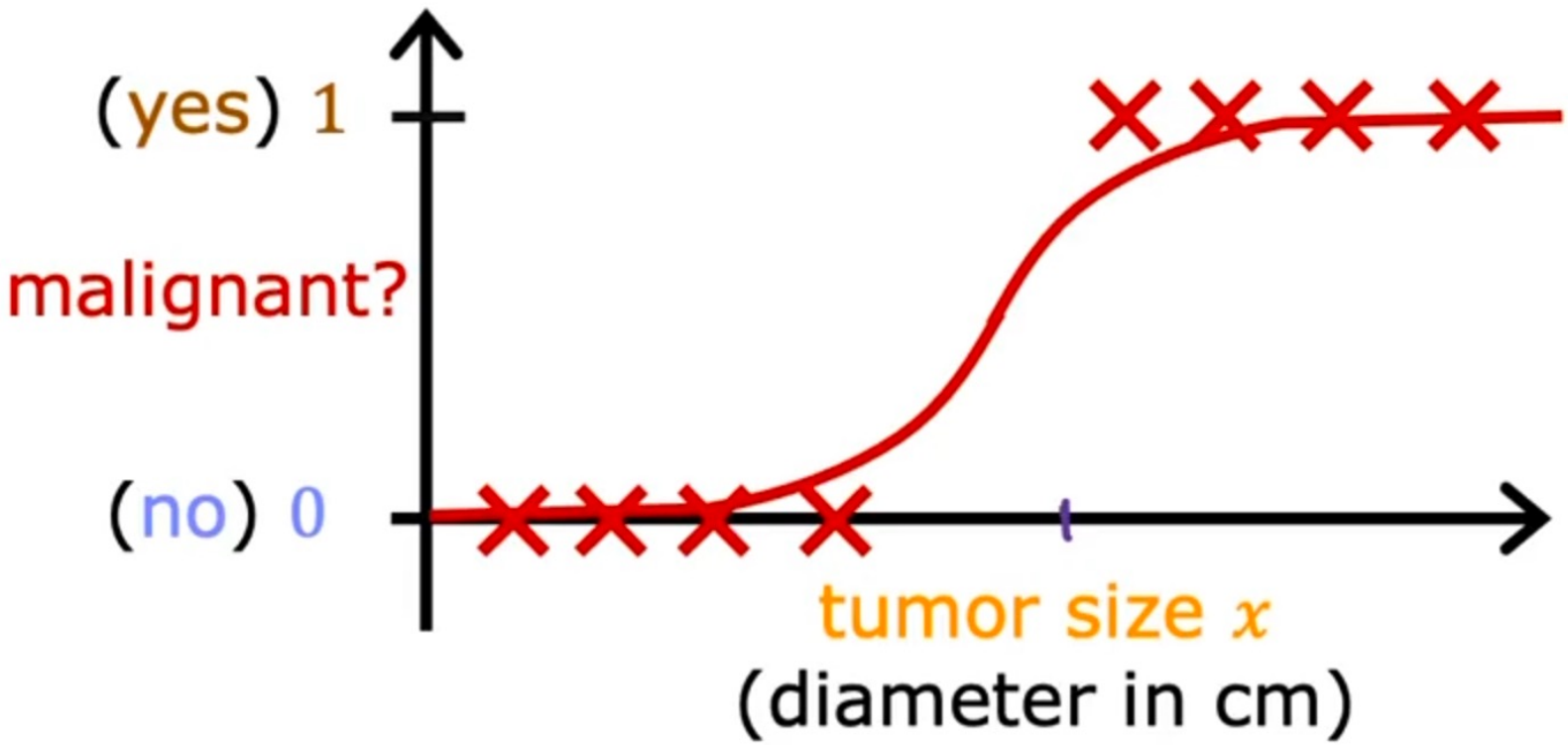
Examples of classification problems are things like: identifying email as Spam or Not Spam or determining if a tumor is malignant or benign. In particular, these are examples of *binary* classification where there are two possible outcomes. Outcomes can be described in pairs of 'positive'/'negative' such as 'yes'/'no, 'true'/'false' or '1'/'0'.

Plots of classification data sets often use symbols to indicate the outcome of an example. In the plots below, 'X' is used to represent the positive values while 'O' represents negative outcomes.

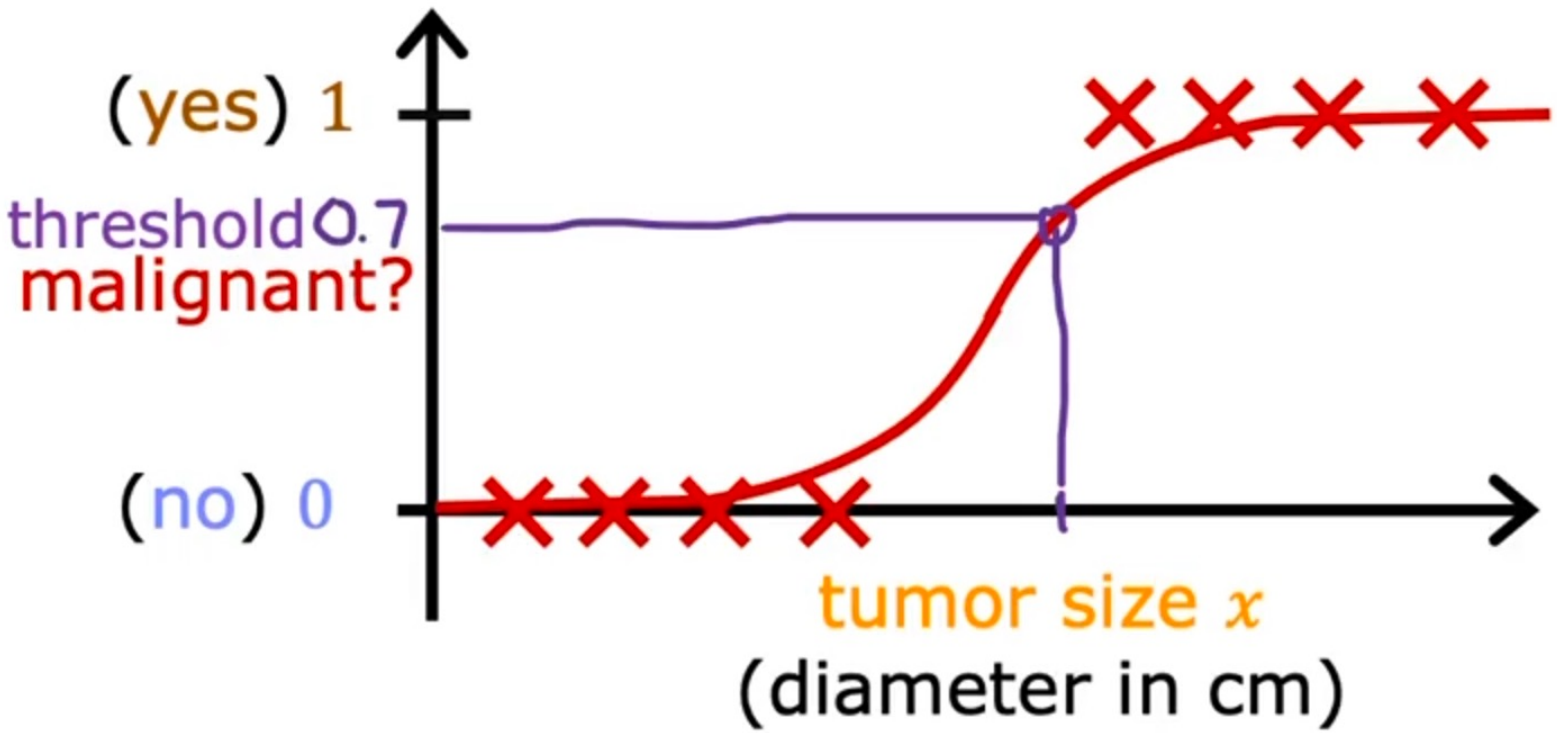
Classification

Logistic Regression

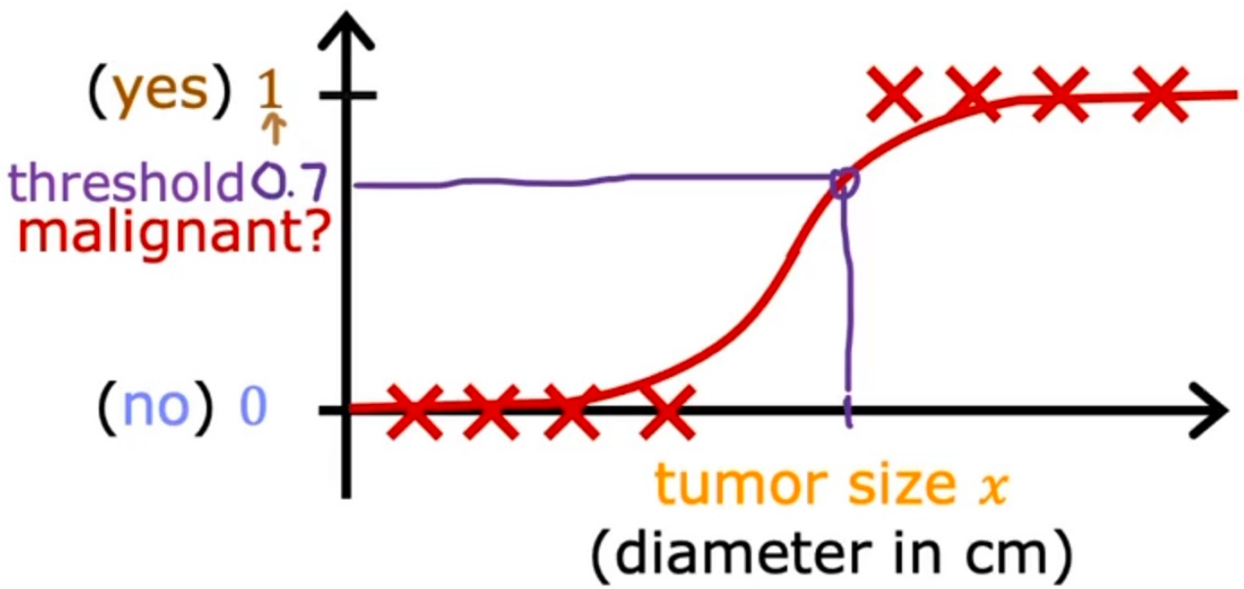

(yes) 1
malignant?
(no) 0
tumor size x
(diameter in cm)



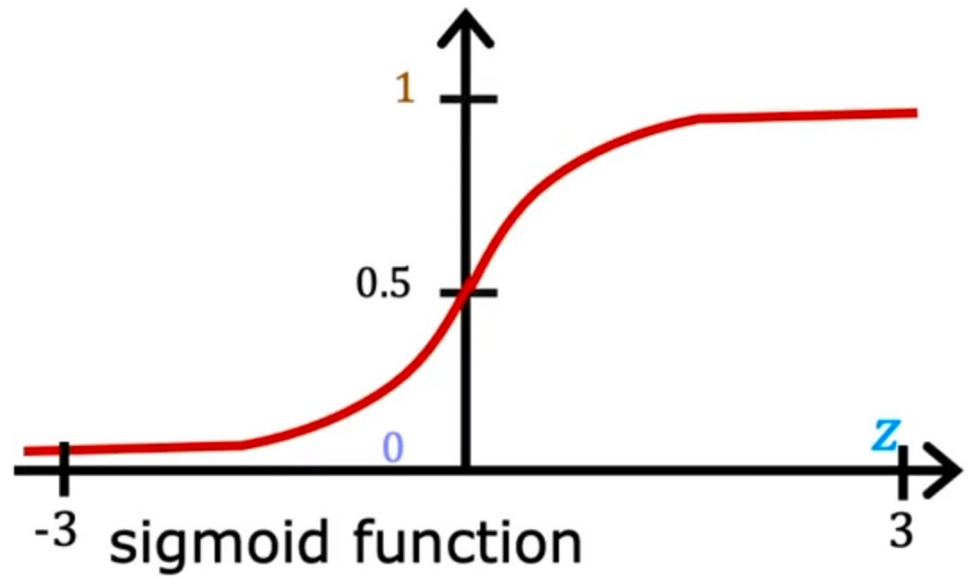
In contrast, what logistic regression we end up doing is fit a curve that looks like this, S-shaped curve to this dataset.



For this example, if a patient comes in with a tumor of this size, which I'm showing on the x-axis, then the algorithm will output 0.7 suggesting that is closer or maybe more likely to be malignant and benign.

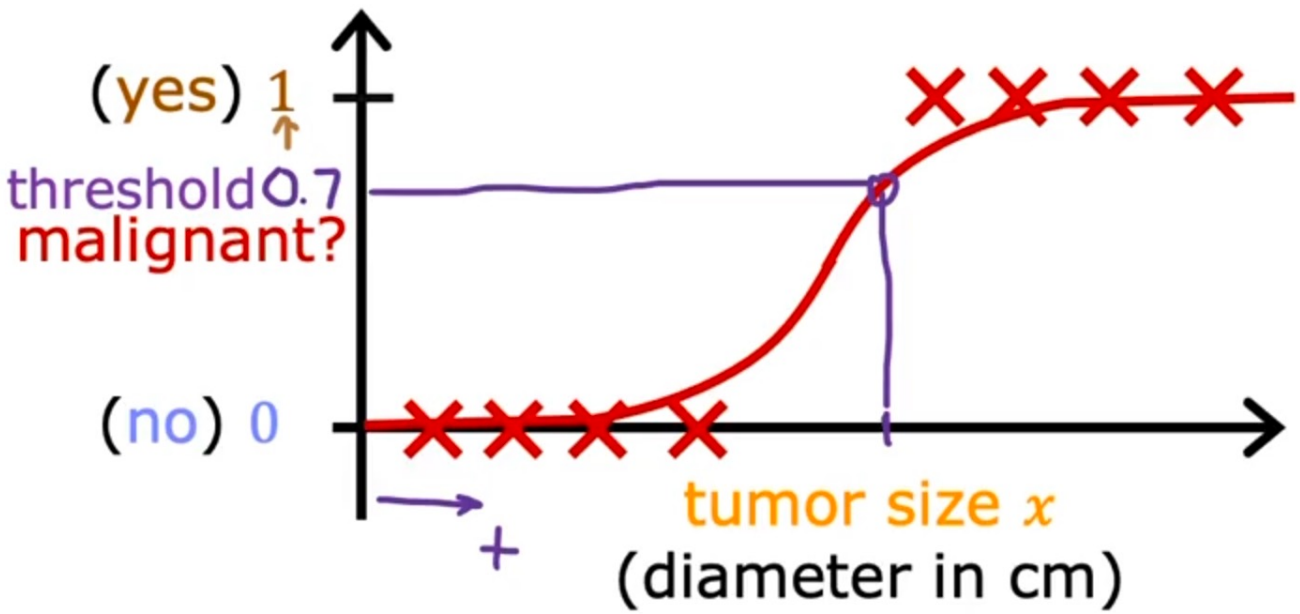


Want outputs between 0 and 1

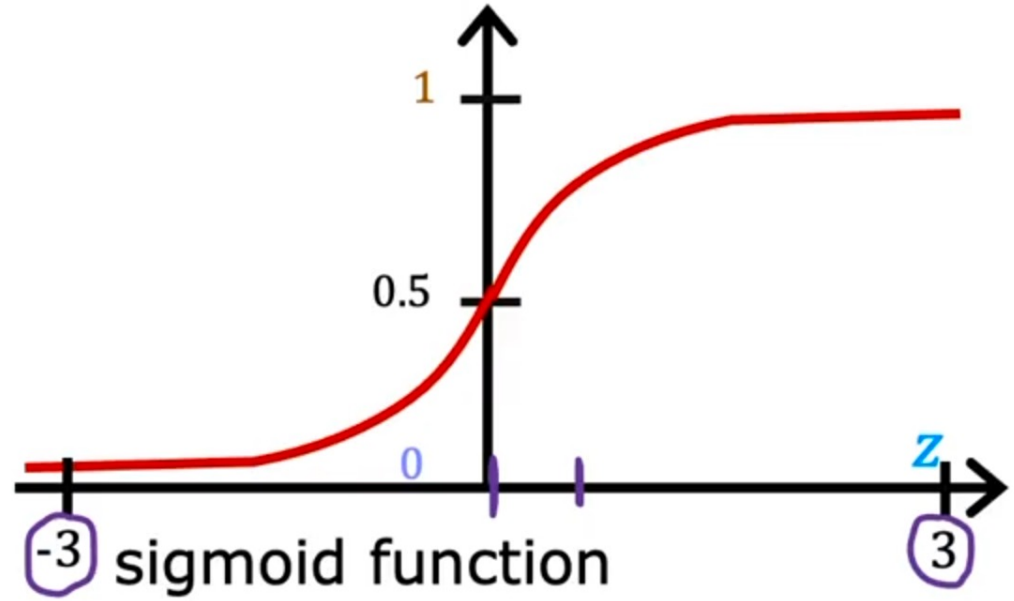


sigmoid function

logistic function



Want outputs between 0 and 1

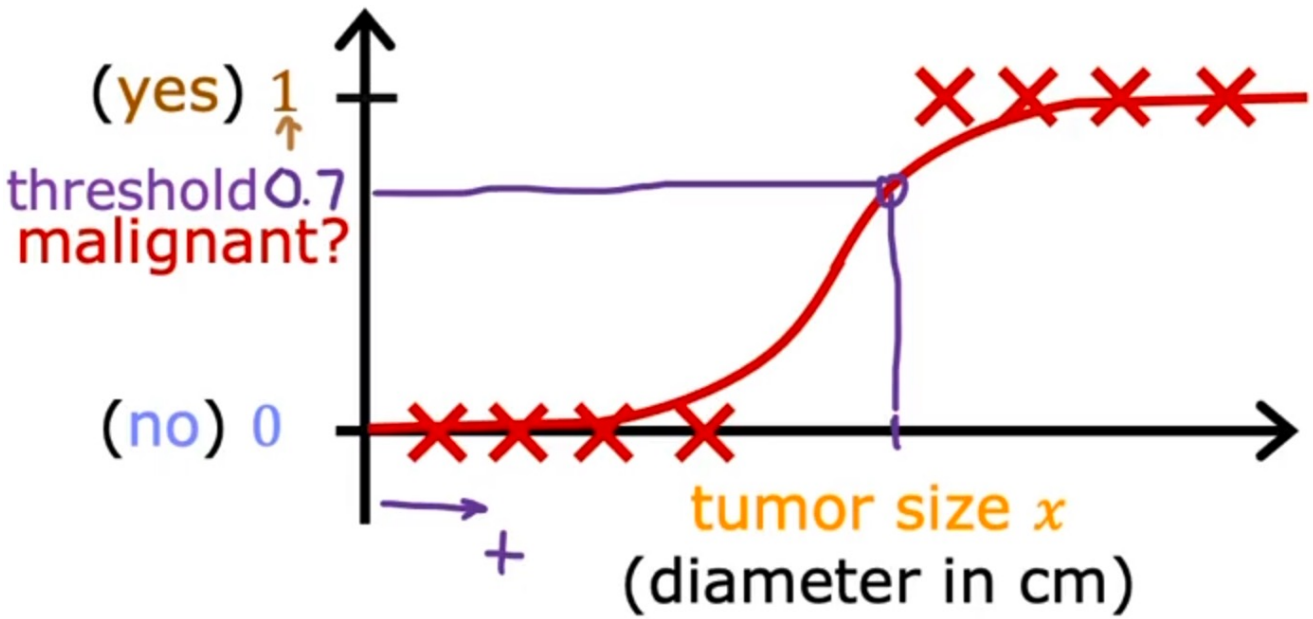


sigmoid function

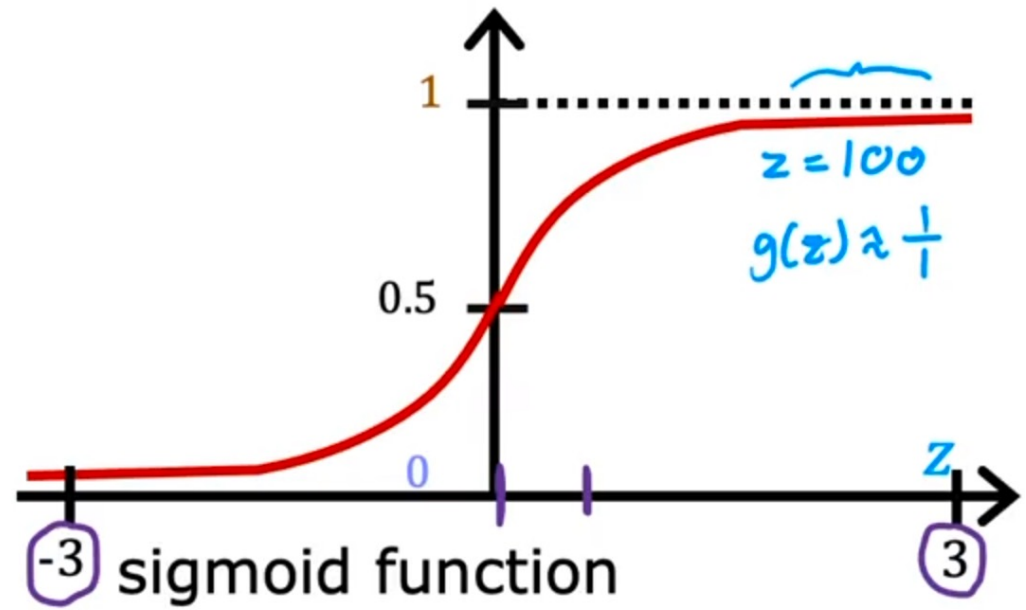
logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$



Want outputs between 0 and 1

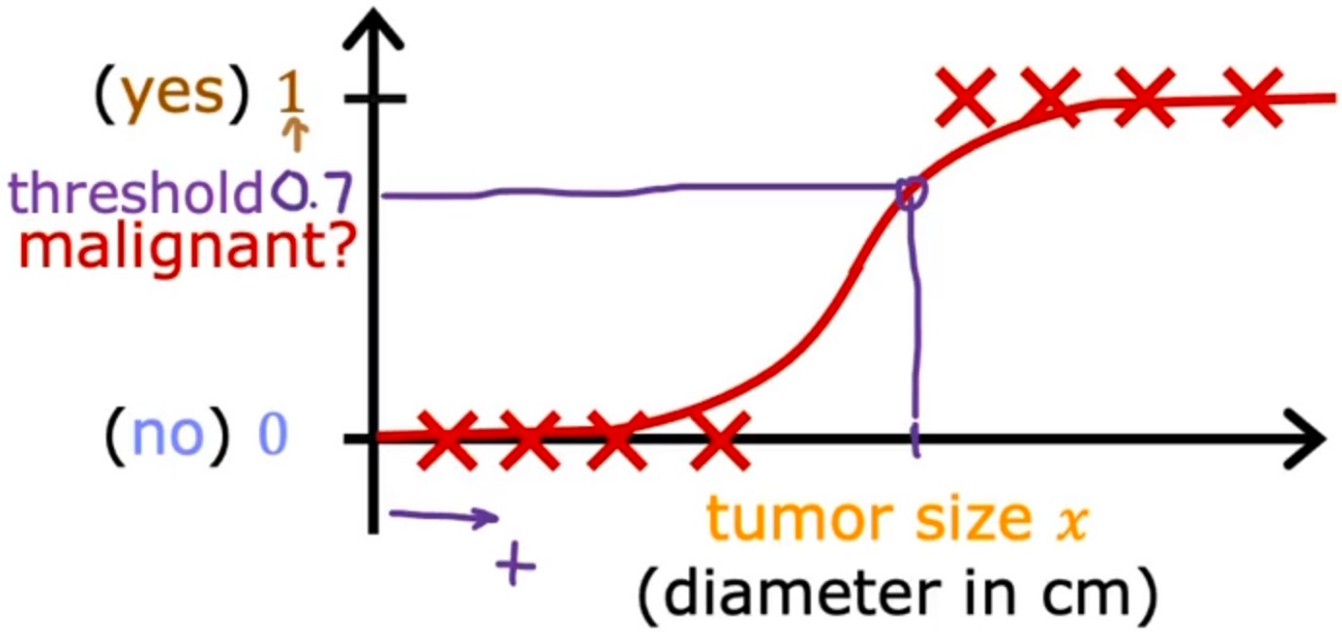


sigmoid function

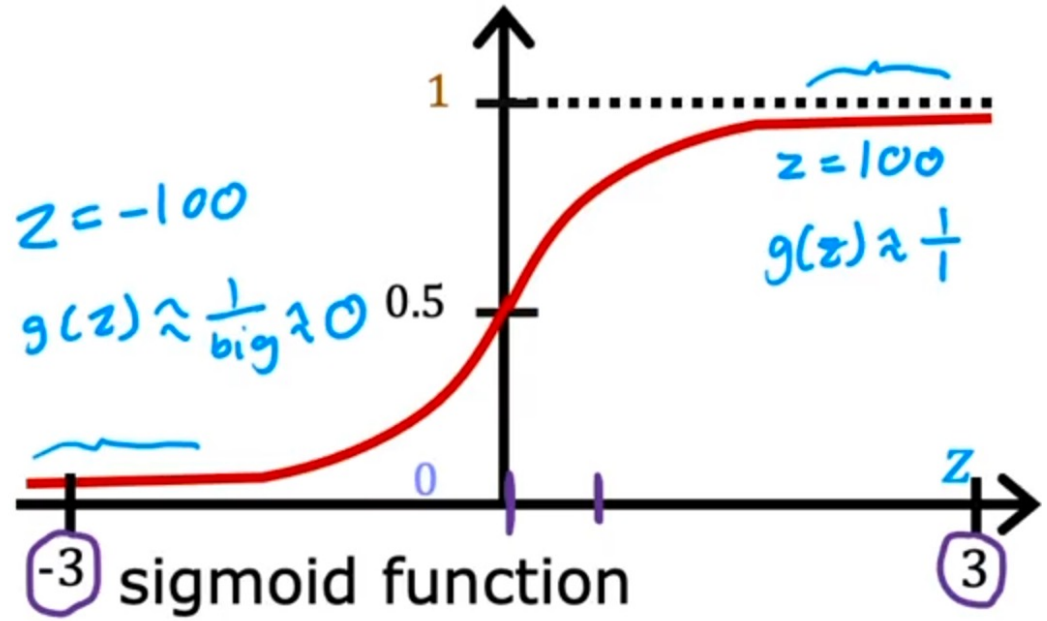
logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$



Want outputs between 0 and 1

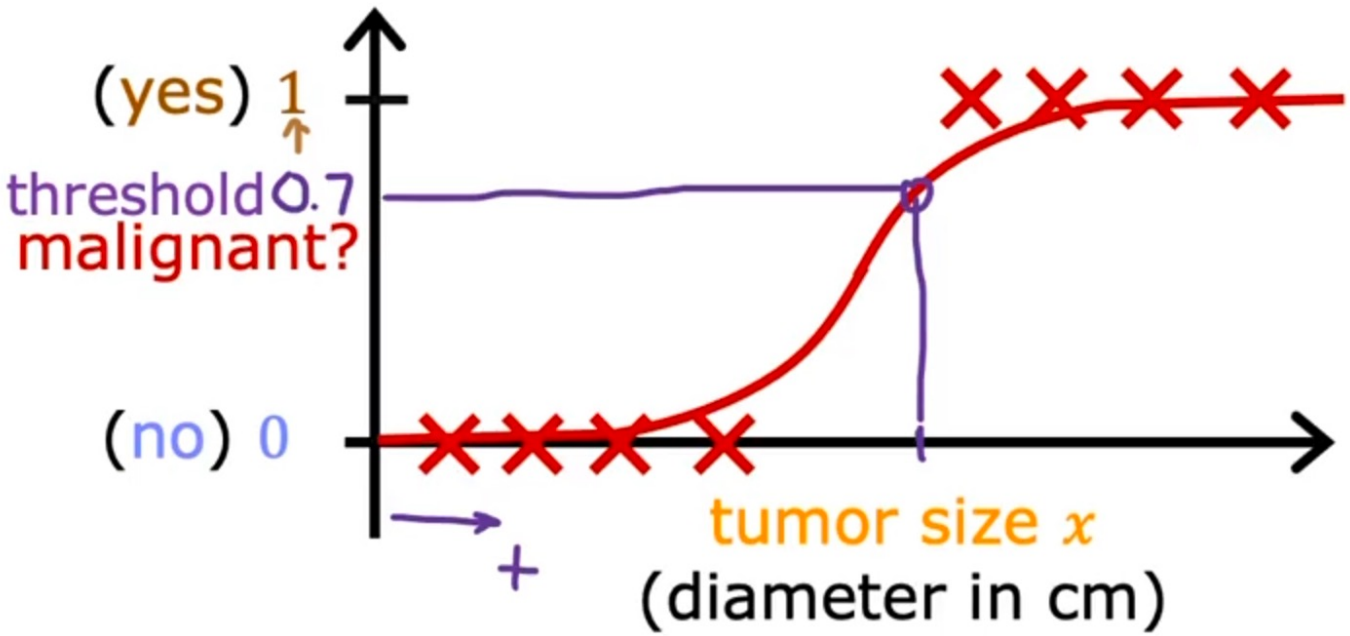


sigmoid function

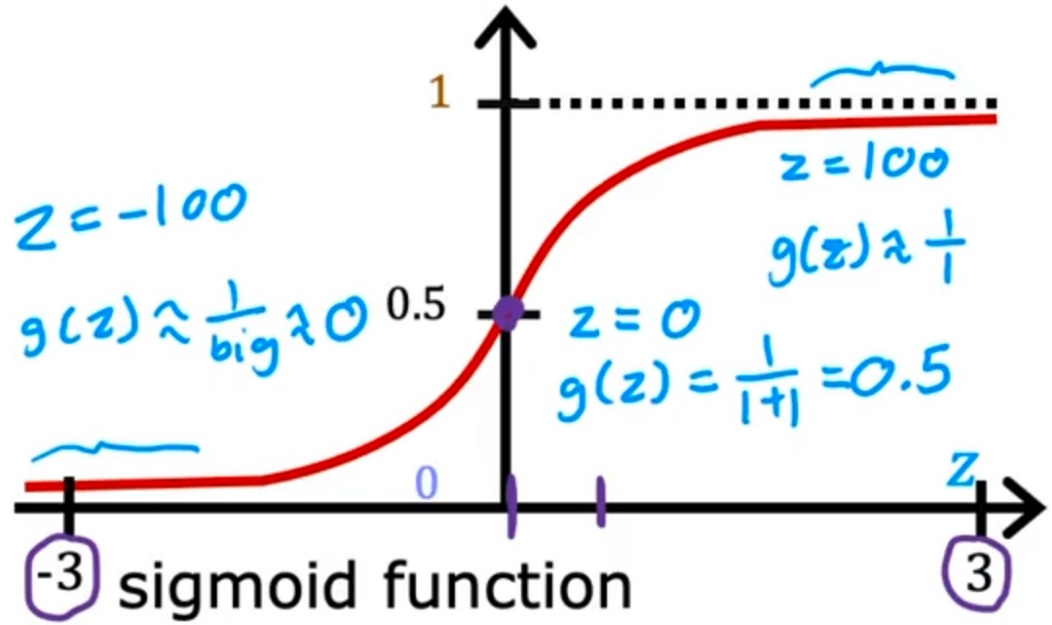
logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$



Want outputs between 0 and 1



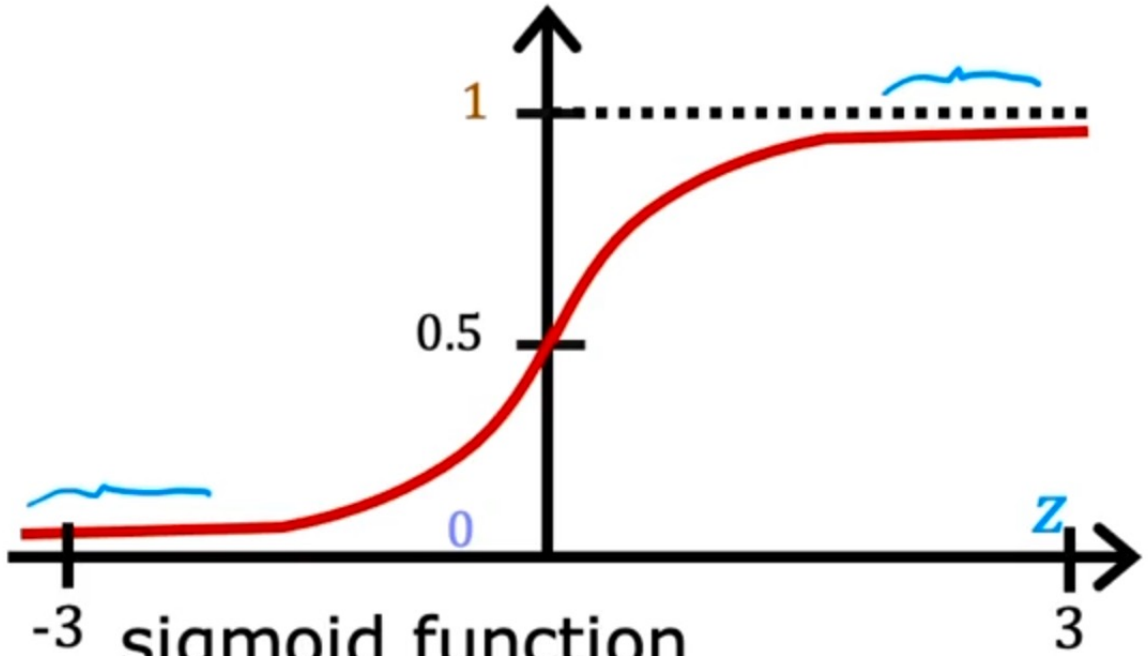
sigmoid function

logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

Want outputs between 0 and 1



sigmoid function

logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

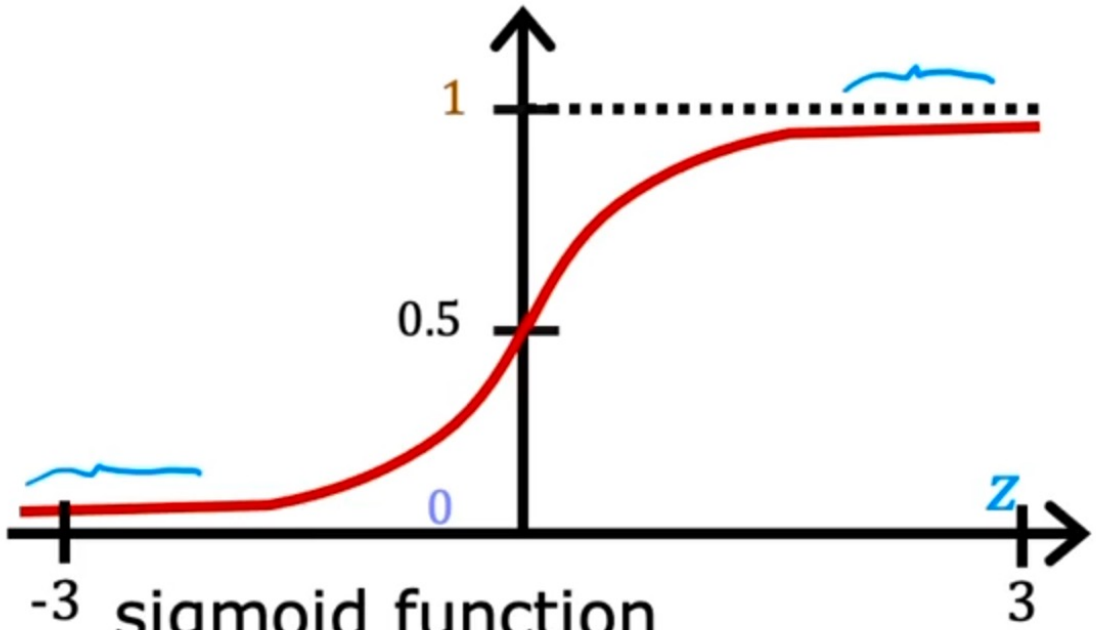
$$f_{\vec{w},b}(\vec{x})$$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

Now, let's use this to build up to the logistic regression algorithm. We're going to do this in two steps. In the first step, I hope you remember that a straight line function, like a linear regression function can be defined as w. product of x plus b.

Let's store this value in a variable which I'm going to call z, and this will turn out to be the same z

Want outputs between 0 and 1



sigmoid function

logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

$$f_{\vec{w},b}(\vec{x})$$

$$z = \vec{w} \cdot \vec{x} + b$$

$$\downarrow$$

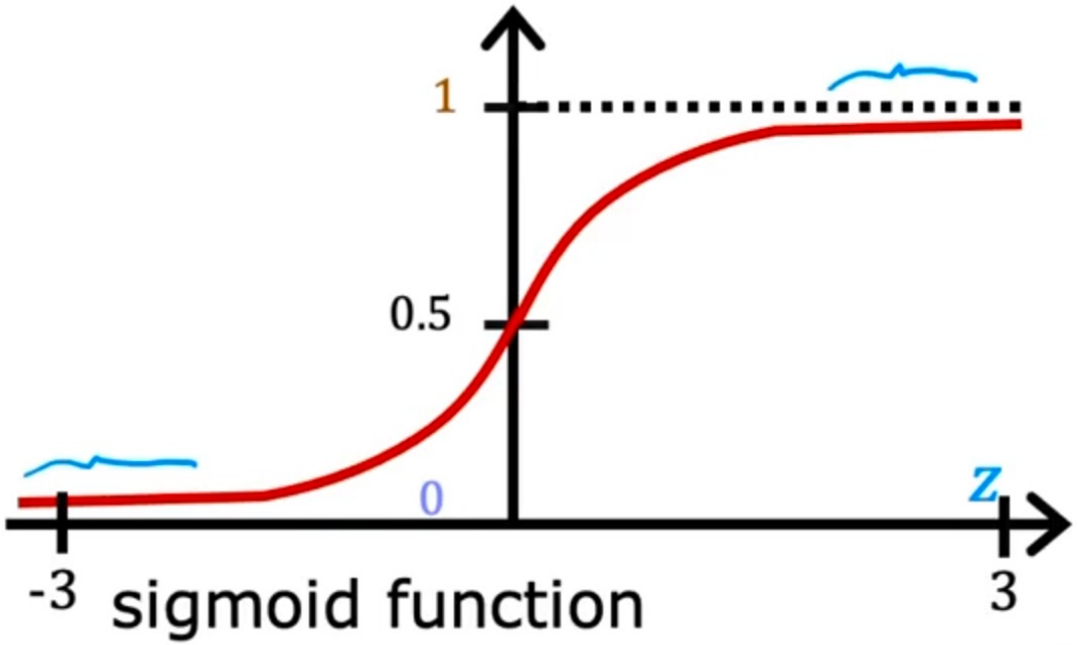
$$z$$

$$\downarrow$$

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

The next step then is to take this value of z and pass it to the Sigmoid function, also called the logistic function, g.

Want outputs between 0 and 1



sigmoid function

logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

$$f_{\vec{w},b}(\vec{x})$$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

"logistic regression"

This is the logistic regression model, and what it does is it inputs feature or set of features X and outputs a number between 0 and 1.

Interpretation of logistic regression output

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

"probability" that class is 1

Example:

x is "tumor size"

y is 0 (not malignant)

or 1 (malignant)

$f_{\vec{w},b}(\vec{x}) = 0.7$

70% chance that y is 1

For example, in this application, where x is the tumor size and y is either 0 or 1, if you have a patient come in and she has a tumor of a certain size x, and if based on this input x, the model I'll plus 0.7, then what that means is that the model is predicting or the model thinks there's a 70 percent chance that the true label y would be equal to 1 for this patient.

In other words, the model is telling us that it thinks the patient has a 70 percent chance of the tumor turning out to be malignant.

Interpretation of logistic regression output

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

"probability" that class is 1

Example:

x is "tumor size"

y is 0 (not malignant)

or 1 (malignant)

$f_{\vec{w},b}(\vec{x}) = 0.7$

70% chance that y is 1

$$P(y = 0) + P(y = 1) = 1$$

That's why if the chance of y being 1 is 0.7 or 70 percent chance, then the chance of it being 0 has got to be 0.3 or 30 percent chance.

Interpretation of logistic regression output

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

"probability" that class is 1

P(1)

$$f_{\vec{w},b}(\vec{x}) = P(y = 1|\vec{x}; \vec{w}, b)$$

Probability that y is 1,
given input $\vec{x}$, parameters $\vec{w}, b$

Example:
x is "tumor size"
y is 0 (not malignant)
or 1 (malignant)

$$f_{\vec{w},b}(\vec{x}) = 0.7$$

70% chance that y is 1

$$P(y = 0) + P(y = 1) = 1$$

For a long time, a lot of Internet advertising was actually driven by basically a slight variation of logistic regression. This was very lucrative for some large companies, and this is basically the algorithm that decided what ad was shown to you and many others on some large websites.

jupyter C1_W3_Lab02_Sigmoid_function_Soln Last Checkpoint: 11/04/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

Optional Lab: Logistic Regression

In this ungraded lab, you will

- explore the sigmoid function (also known as the logistic function)
- explore logistic regression which uses the sigmoid function

Tools

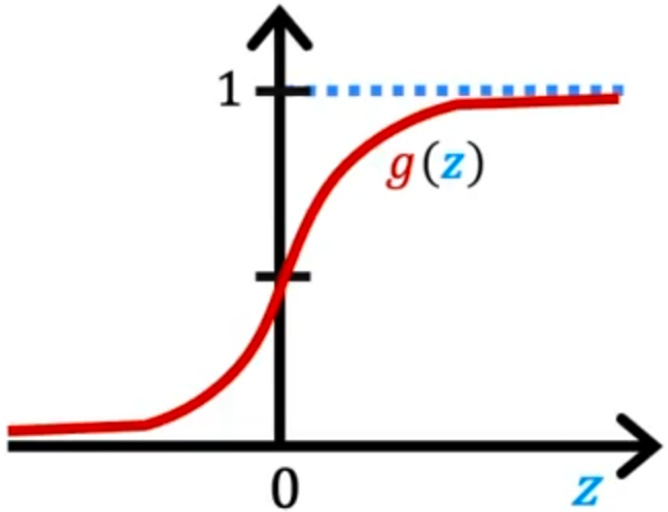
In this lab you will make use of:

- NumPy, a popular library for scientific computing
- Matplotlib, a popular library for plotting data

In [ ]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
plt.rcParams['font.size'] = 8
from plt_one_addpt_onclick import plt_one_addpt_onclick
from lab_utils_common import draw_vthresh
```

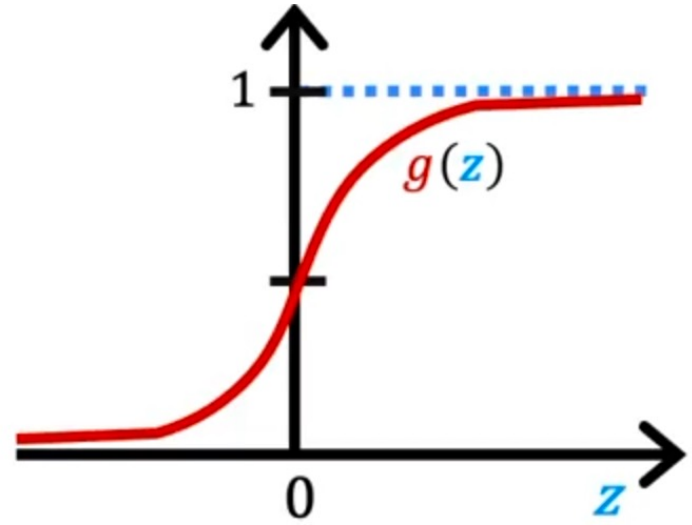
Classification

Decision Boundary



$f_{\vec{w},b}(\vec{x})$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$



$f_{\vec{w},b}(\vec{x})$

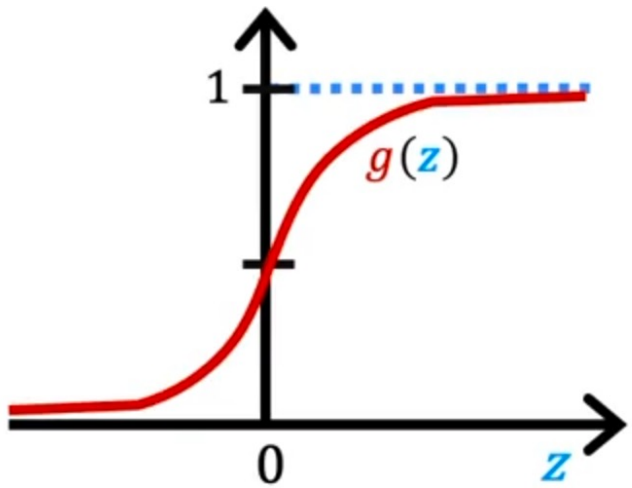
$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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



$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$f_{\vec{w},b}(\vec{x})$

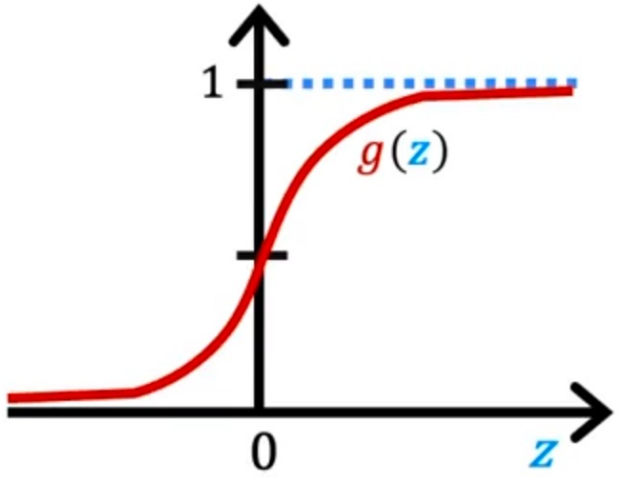
$$\boxed{z = \vec{w}\cdot\vec{x}+b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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



$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y=1|x;\vec{w},b) \quad 0.7 \quad 0.3$$

$f_{\vec{w},b}(\vec{x})$

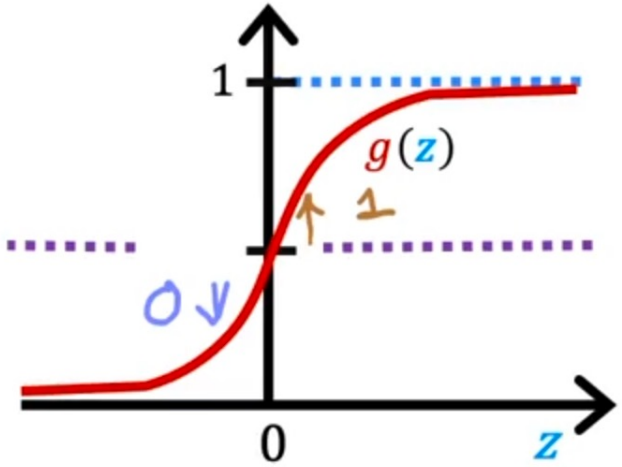
$$\boxed{z = \vec{w}\cdot\vec{x}+b}$$

$\downarrow$

z

$\downarrow$

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



$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y=1|x;\vec{w},b) \quad 0.7 \quad 0.3$$

0 or 1?

$$f_{\vec{w},b}(\vec{x})$$

$$\boxed{z = \vec{w}\cdot\vec{x}+b}$$

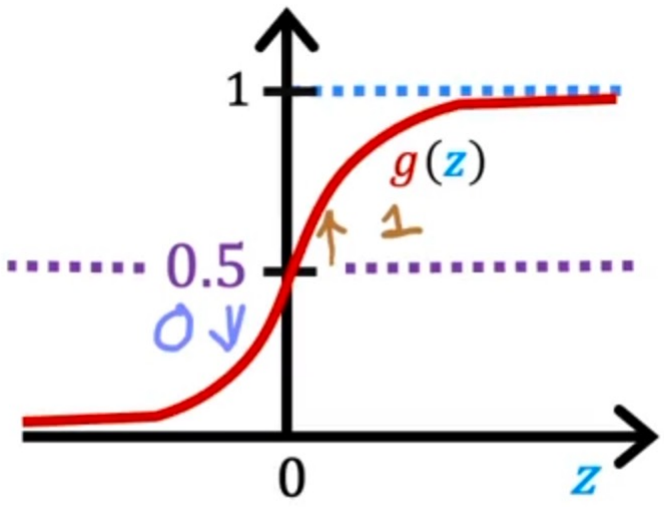
$$\downarrow$$

$$z$$

$$\downarrow$$

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

Is the value of y going to be zero or one? Well, one thing you might do is set a threshold above which you predict y is one, or you set y hat to prediction to be equal to one and below which you might say y hat, my prediction is going to be equal to zero.



$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y=1|x;\vec{w},b) \quad 0.7 \quad 0.3$$

0 or 1?

Is $f_{\vec{w},b}(\vec{x}) \geq \overbrace{0.5}^{\text{threshold}}$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

$f_{\vec{w},b}(\vec{x})$

$$\boxed{z = \vec{w}\cdot\vec{x}+b}$$

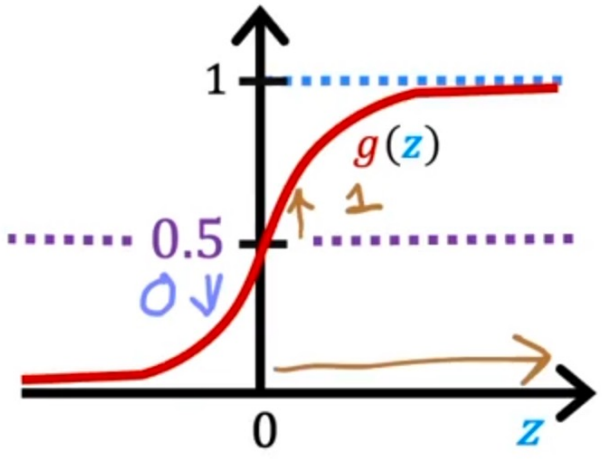
$$\downarrow$$

$$z$$

$$\downarrow$$

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

A common choice would be to pick a threshold of 0.5 so that if f of x is greater than or equal to 0.5, then predict y is one. We write that prediction as y hat equals 1, or if f of x is less than 0.5, then predict y is 0, or in other words, the prediction y hat is equal to 0.



$$f_{\vec{w},b}(\vec{x})$$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y = 1 | x; \vec{w}, b) \quad 0.7 \quad 0.3$$

0 or 1?

threshold

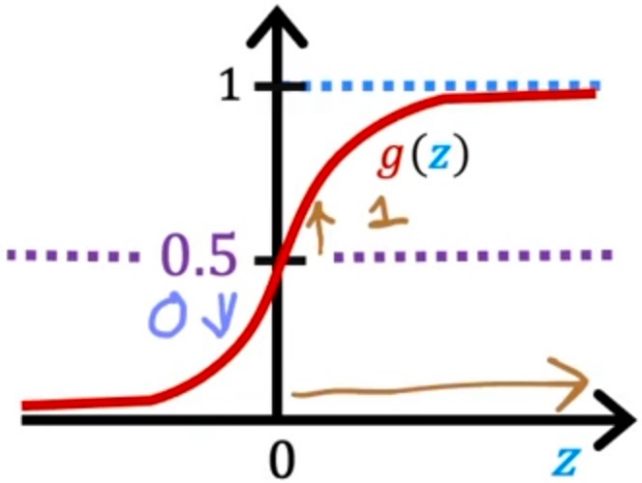
Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

$$g(z) \geq 0.5$$

In other words, when is f of x greater than or equal to 0.5. We'll recall that f of x is just equal to g of z. So f is greater than or equal to 0.5 whenever g of z is greater than or equal to 0.5. But when is g of z greater than or equal to 0.5? Well, here's a Sigmoid function over here. So g of z is greater than or equal to 0.5 whenever z is greater than or equal to 0. That is whenever z is on the right half of this axis. Finally, when is z greater than or equal to zero? Well, z is equal to w.x plus b, so z is greater than or equal to zero whenever w.x plus b is greater than or equal to zero.



$f_{\vec{w},b}(\vec{x})$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y = 1 | x; \vec{w}, b) \quad 0.7 \quad 0.3$$

0 or 1?

Is $f_{\vec{w},b}(\vec{x}) \geq \overbrace{0.5}^{\text{threshold}}$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

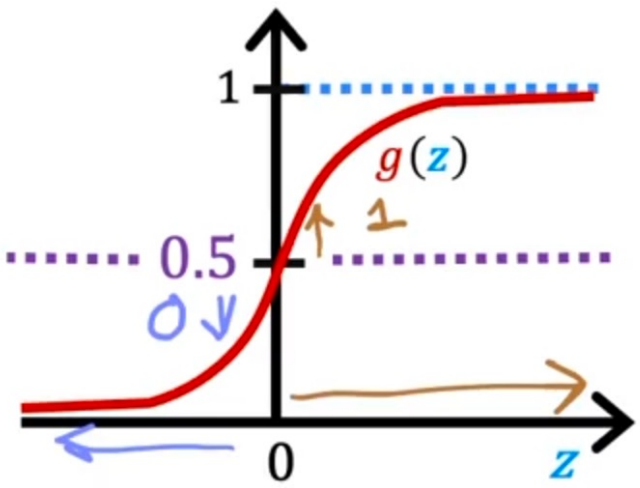
When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

$$g(z) \geq 0.5$$

$$z \geq 0$$

$$\vec{w} \cdot \vec{x} + b \geq 0$$

$$\hat{y} = 1$$



$$f_{\vec{w},b}(\vec{x})$$

$$\boxed{z = \vec{w} \cdot \vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y = 1 | x; \vec{w}, b) \quad 0.7 \quad 0.3$$

0 or 1?

Is $f_{\vec{w},b}(\vec{x}) \geq \overbrace{0.5}^{\text{threshold}}$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

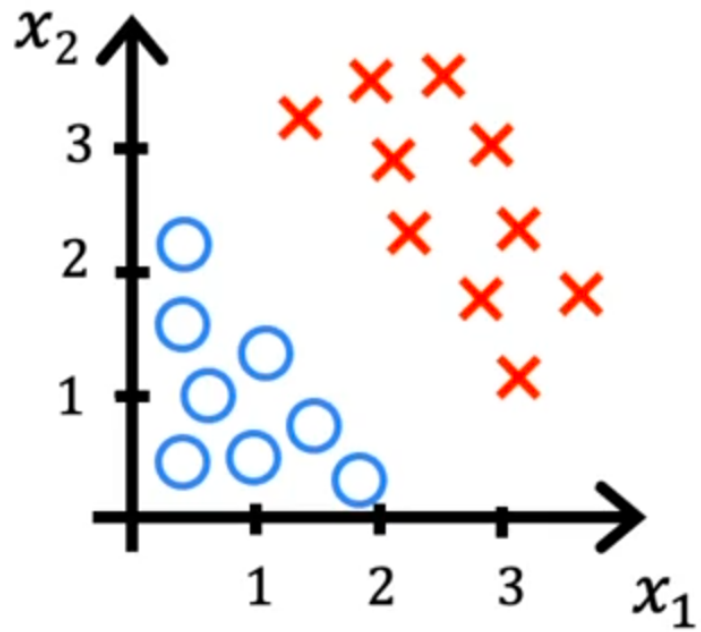
$$g(z) \geq 0.5$$

$$z \geq 0$$

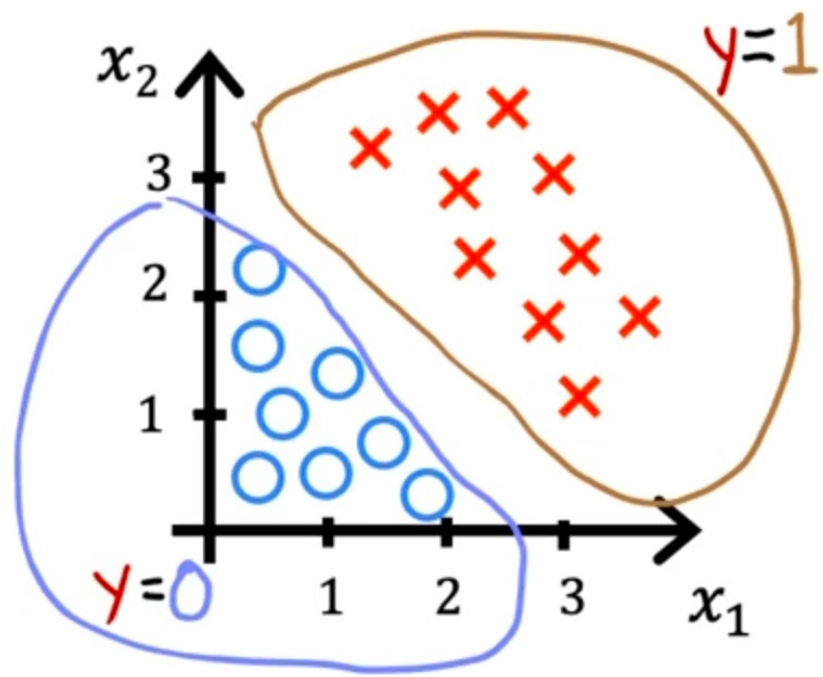
$$\vec{w} \cdot \vec{x} + b \geq 0 \qquad \vec{w} \cdot \vec{x} + b < 0$$

$$\hat{y} = 1 \qquad \hat{y} = 0$$

Decision boundary

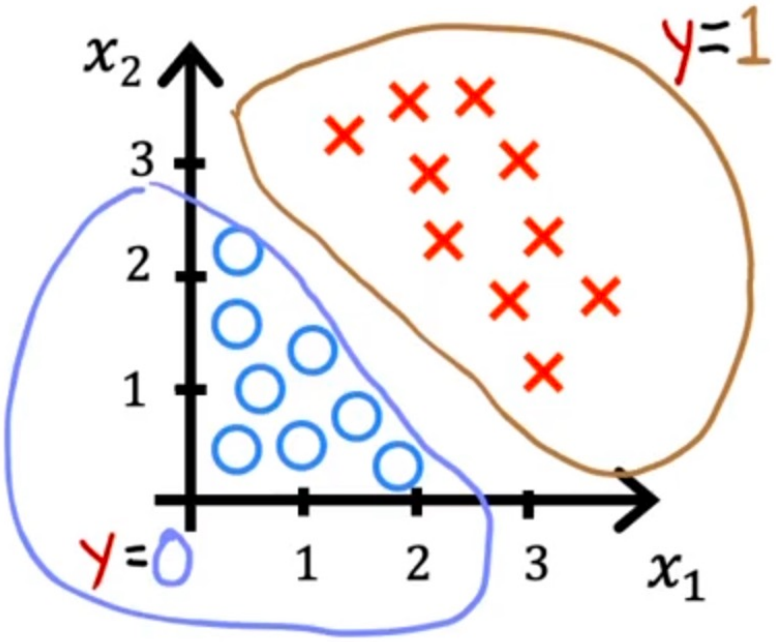


Decision boundary



Decision boundary

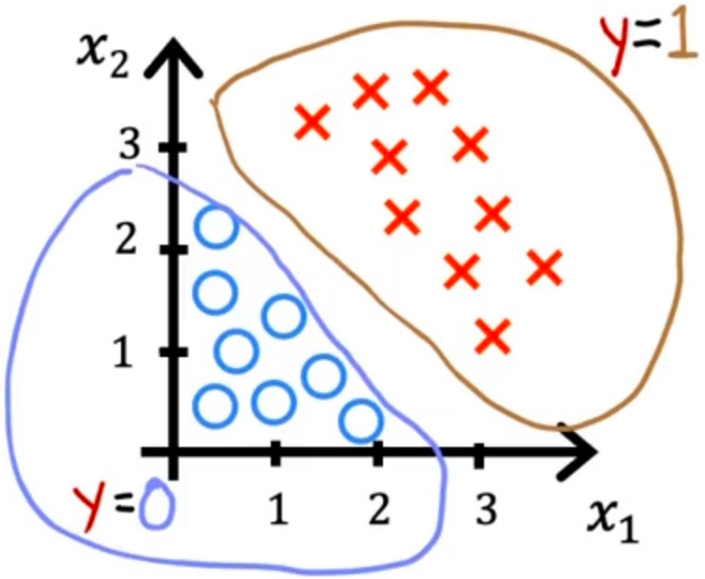
$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{-3}{b})$$



Decision boundary

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underbrace{w_1 x_1}_{1} + \underbrace{w_2 x_2}_{1} + \underbrace{b}_{-3})$$

Decision boundary $z = \vec{w} \cdot \vec{x} + b = 0$



To figure that out, there's a very interesting line to look at, which is when wx plus bis exactly equal to 0. It turns out that this line is also called the decision boundary because that's the line where you're just almost neutral about whether y is 0 or y is 1.

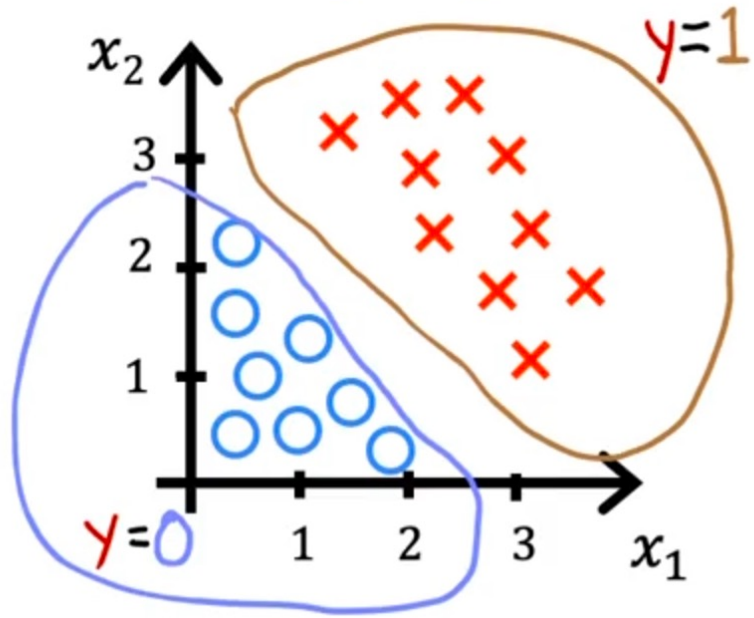
Decision boundary

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{-3}{b})$$

Decision boundary $z = \vec{w} \cdot \vec{x} + b = 0$

$$z = x_1 + x_2 - 3 = 0$$

$$x_1 + x_2 = 3$$



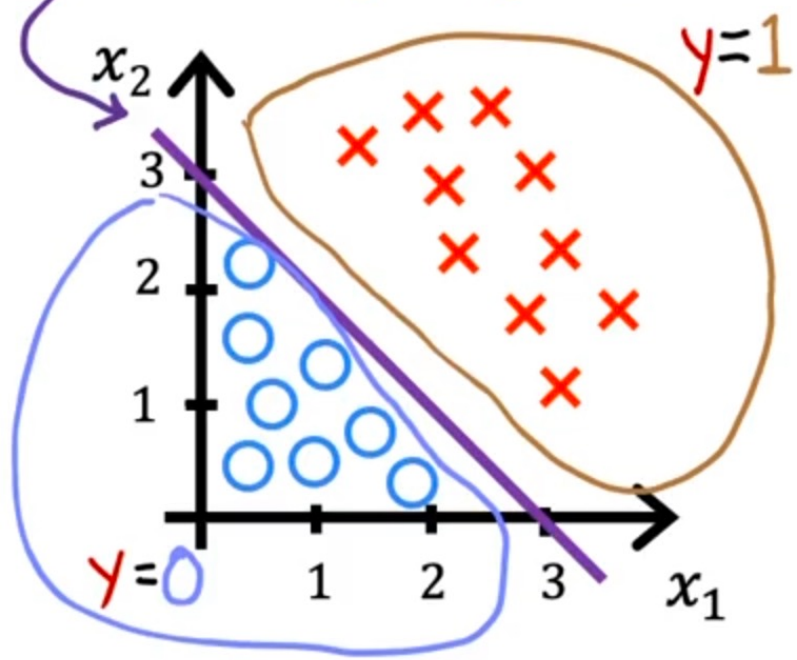
Decision boundary

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{-3}{b})$$

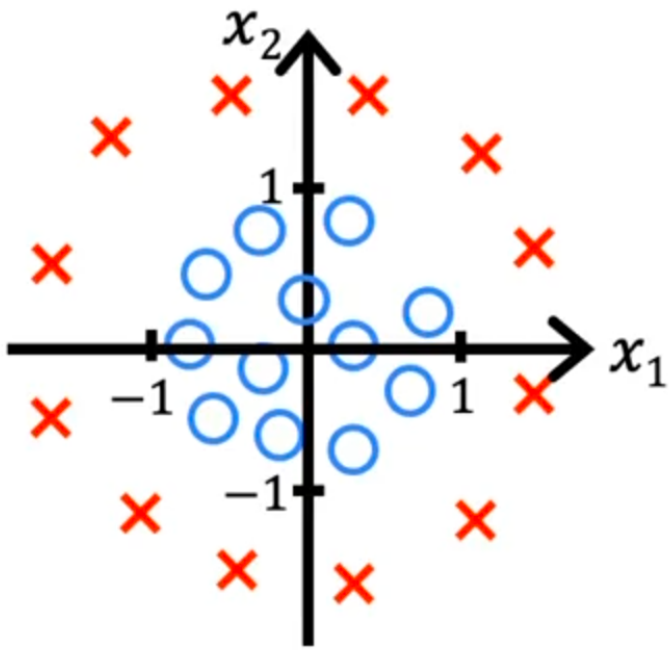
Decision boundary $z = \vec{w} \cdot \vec{x} + b = 0$

$$z = x_1 + x_2 - 3 = 0$$

$$x_1 + x_2 = 3$$

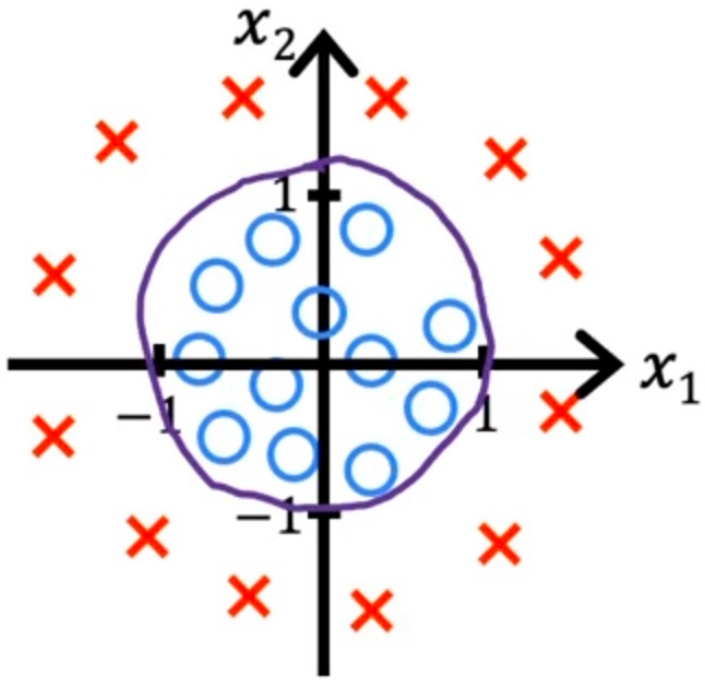


Non-linear decision boundaries



$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{w_1 x_1^2 + w_2 x_2^2 + b}^{z})$$

Non-linear decision boundaries



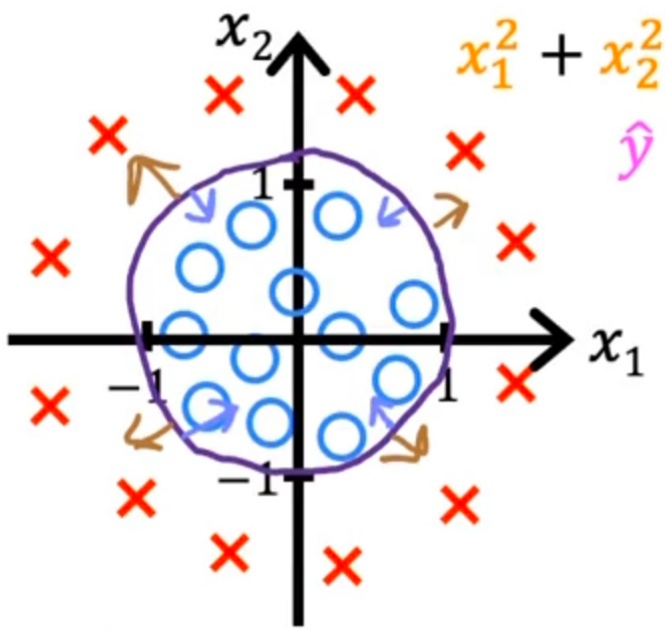
$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{\underset{1}{w_1} x_1^2 + \underset{1}{w_2} x_2^2 + \underset{-1}{b}}^{z})$$

decision boundary

$$z = x_1^2 + x_2^2 - 1 = 0$$

$$x_1^2 + x_2^2 = 1$$

Non-linear decision boundaries



$x_1^2 + x_2^2 \geq 1$

$\hat{y} = 1$

$x_1^2 + x_2^2 < 1$

$\hat{y} = 0$

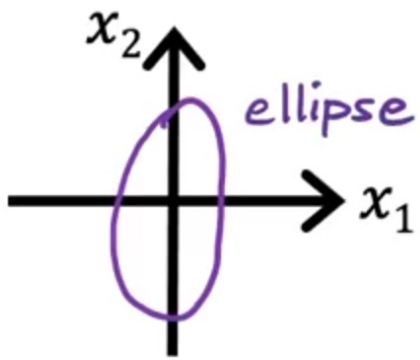
$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{\underset{1}{w_1} x_1^2 + \underset{1}{w_2} x_2^2 + \underset{-1}{b}}^{z})$$

decision boundary

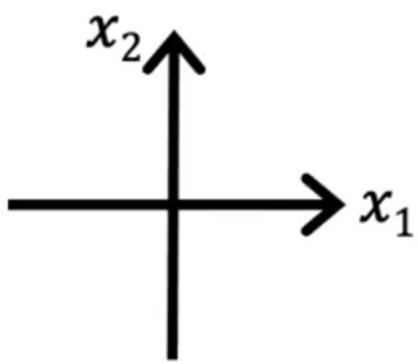
$$z = x_1^2 + x_2^2 - 1 = 0$$

$$x_1^2 + x_2^2 = 1$$

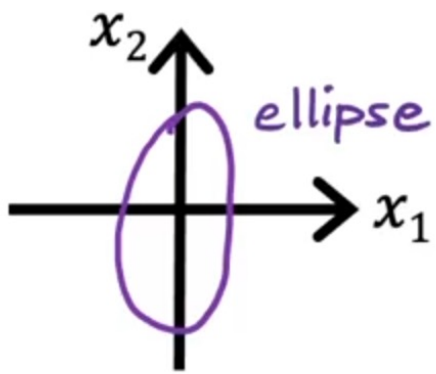
Non-linear decision boundaries

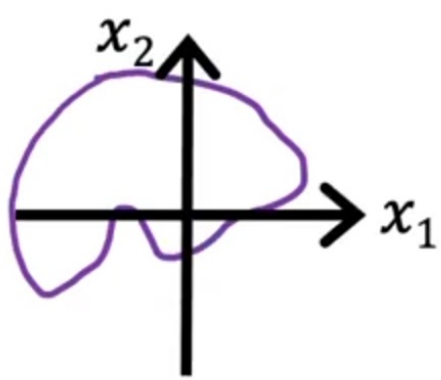


$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2$$



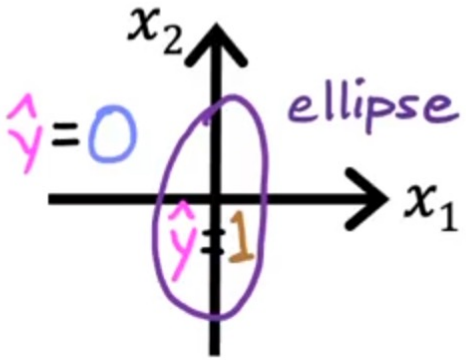
Non-linear decision boundaries



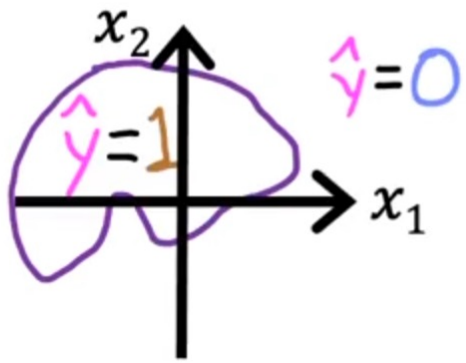


$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2 + w_6 x_1^3 + \cdots + b)$$

Non-linear decision boundaries



$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2 + w_5x_2^2 + w_6x_1^3 + \cdots + b)$$



In other words, logistic regression can learn to fit pretty complex data. Although if you were to not include any of these higher-order polynomials, so if the only features you use are x_1, x_2, x_3, and so on, then the decision boundary for logistic regression will always be linear, will always be a straight line.

jupyter Decision Boundary Last Checkpoint: 11/04/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Optional Lab: Logistic Regression, Decision Boundary

Goals

In this lab you will:

- plot the decision boundary for a logistic regression model, which will give you a better sense of what the model is predicting.

In [ ]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
plt.rcParams['font.size'] = 8
from lab_utils_common import plot_data, dlc, sigmoid, draw_vthresh
```

Dataset

Let's suppose you have following training dataset

- The input variable `X` is a numpy array which has 6 training examples, each with two features

Cost Function

Cost Function for Logistic Regression

Training set

	tumor size (cm)	…	patient's age	malignant?
$i=1$	10		52	1
⋮	2		73	0
	5		55	0
	12		49	1
$i=m$	…		…	…

$i = 1, \ldots, m$ ← training examples

$j = 1, \ldots, n$ ← features

The cost function gives you a way to measure how well a specific set of parameters fits the training data. Thereby gives you a way to try to choose better parameters. In this lecture, we'll look at how the squared error cost function is not an ideal cost function for logistic regression. We'll take a look at a different cost function that can help us choose better parameters for logistic regression.

Training set

	tumor size (cm) x_1	…	patient's age x_n	malignant? y
$i=1$	10		52	1
⋮	2		73	0
	5		55	0
	12		49	1
$i=m$	…		…	…

$i = 1, \ldots, m$ ← training examples

$j = 1, \ldots, n$ ← features

target y is 0 or 1

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

Training set

	tumor size (cm) x_1	…	patient's age x_n	malignant? y
$i=1$	10		52	1
⋮	2		73	0
	5		55	0
	12		49	1
$i=m$	…		…	…

$i = 1, \ldots, m$ ← training examples

$j = 1, \ldots, n$ ← features

target y is 0 or 1

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

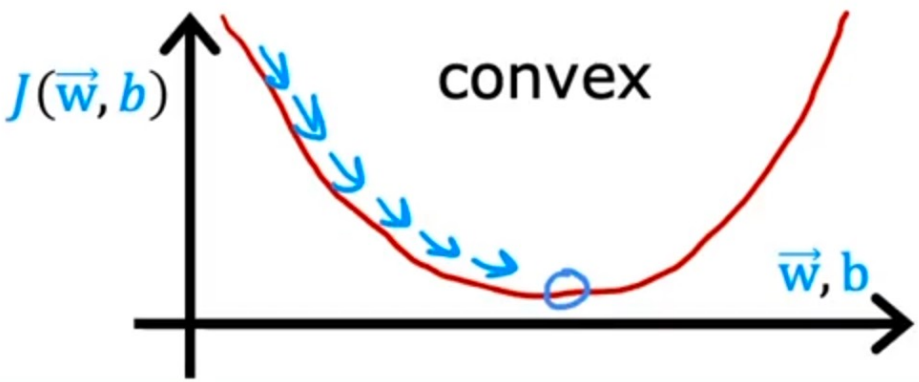
How to choose $\vec{w} = [w_1 \quad w_2 \quad \cdots \quad w_n]$ and b?

Squared error cost

$$J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)}\right)^2$$

linear regression

$$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b$$

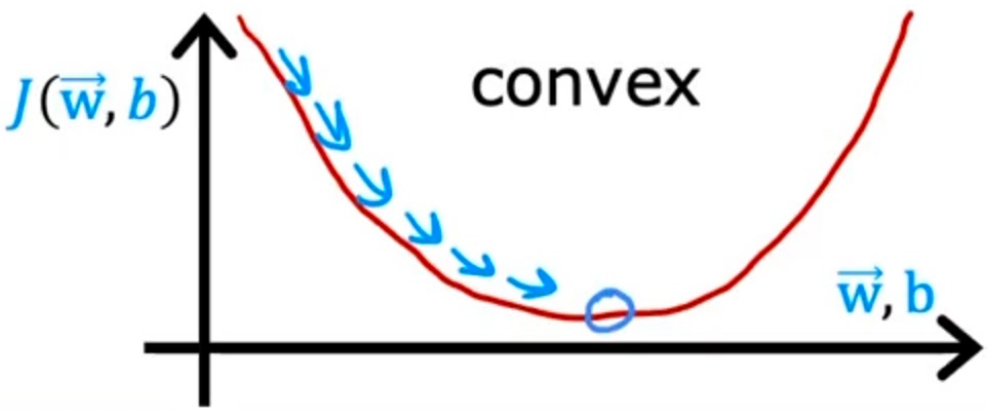


Squared error cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2$$

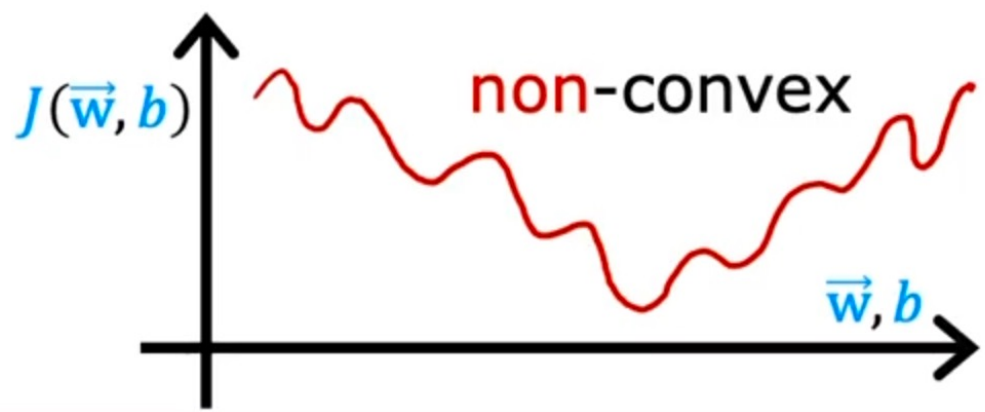
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

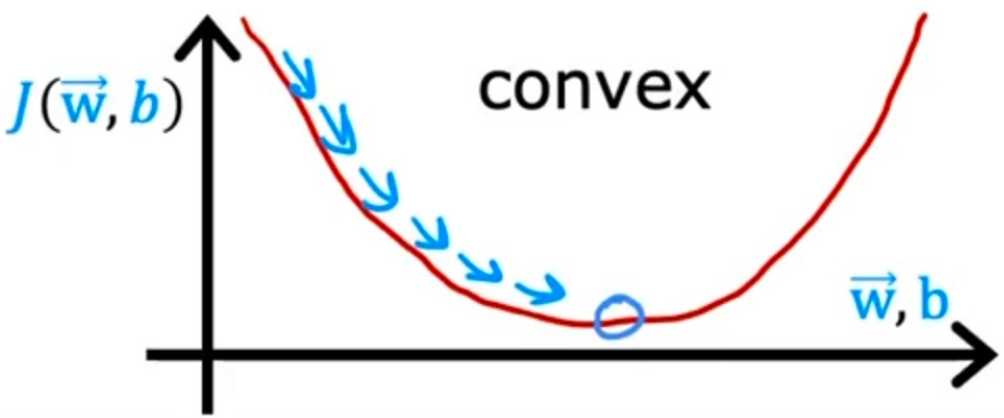


Squared error cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2$$

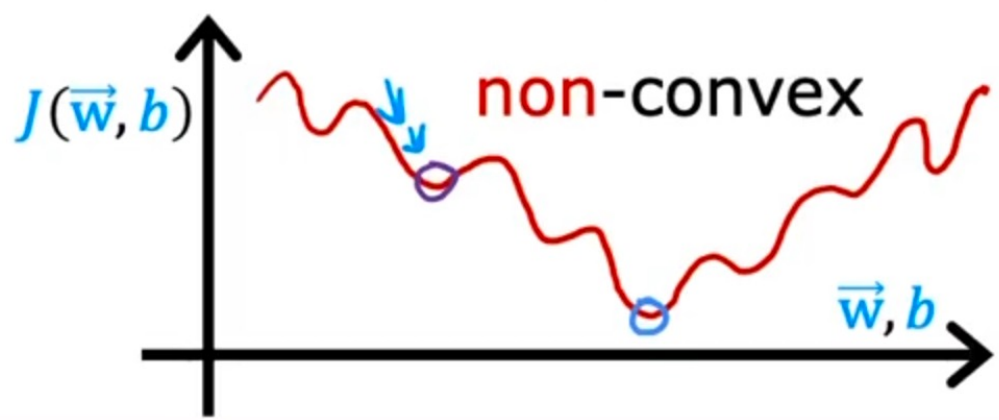
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$



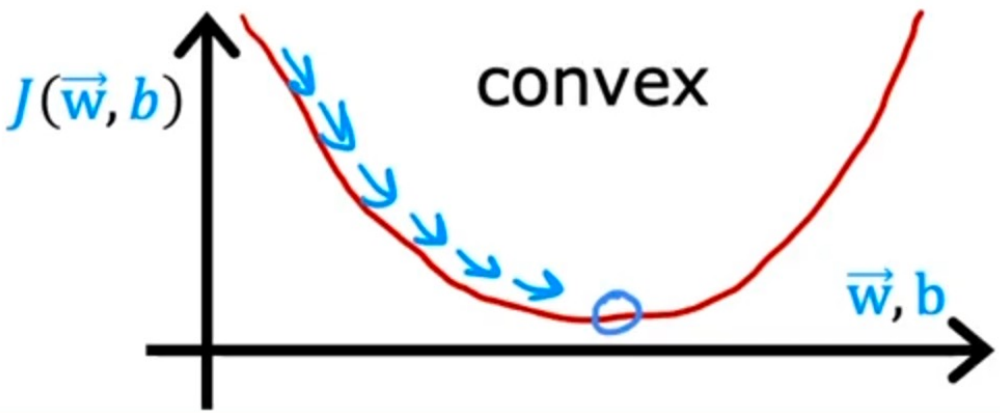
Squared error cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} \boxed{\frac{1}{2}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2}$$

loss $\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)}$

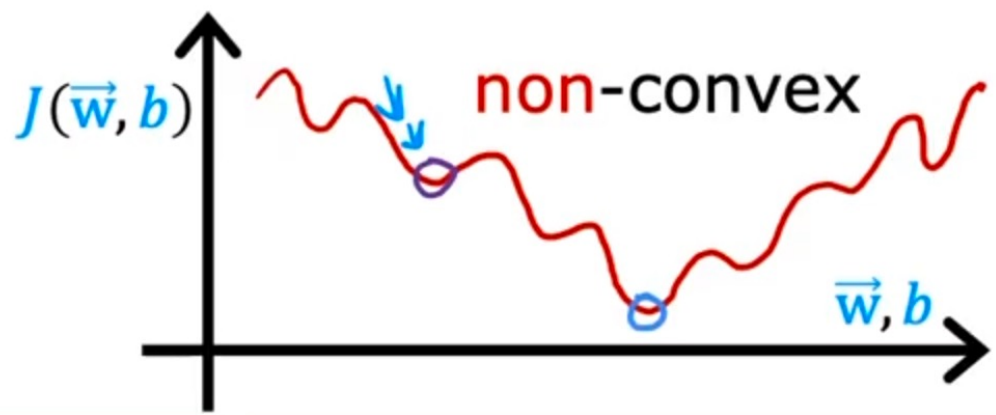
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

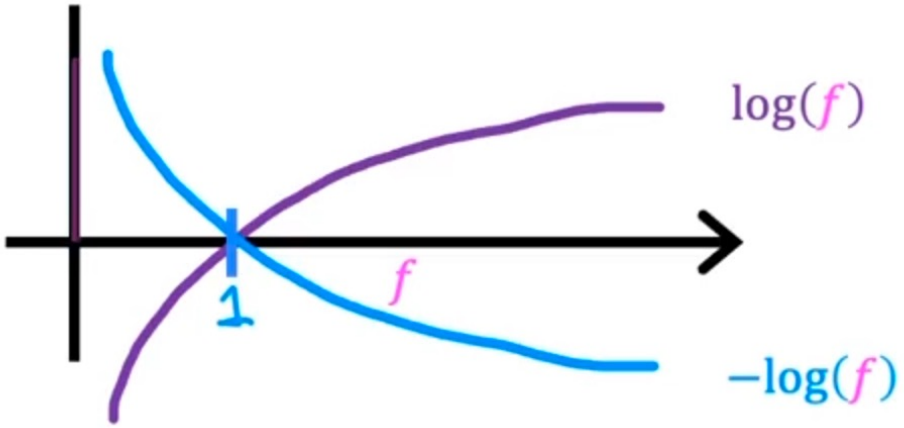


Logistic loss function

$$L\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$



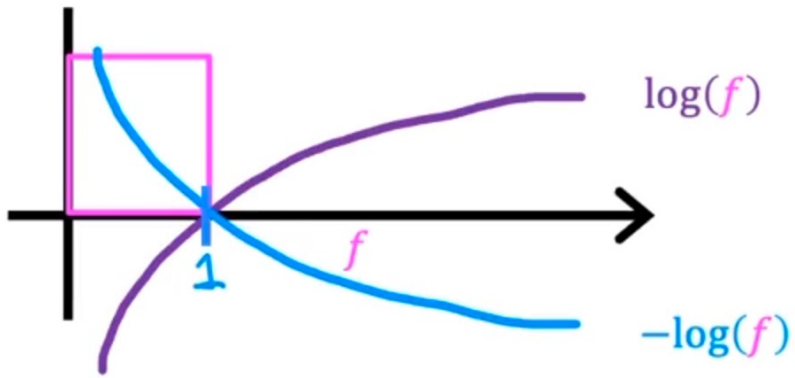
If you plot log of f, it looks like this curve here, where f here is on the horizontal axis. A plot of a negative of the log of f looks like this, where we just flip the curve along the horizontal axis.

Notice that it intersects the horizontal axis at f equals 1 and continues downward from there.

Now, f is the output of logistic regression. Thus, f is always between zero and one because the output of logistic regression is always between zero and one.

Logistic loss function

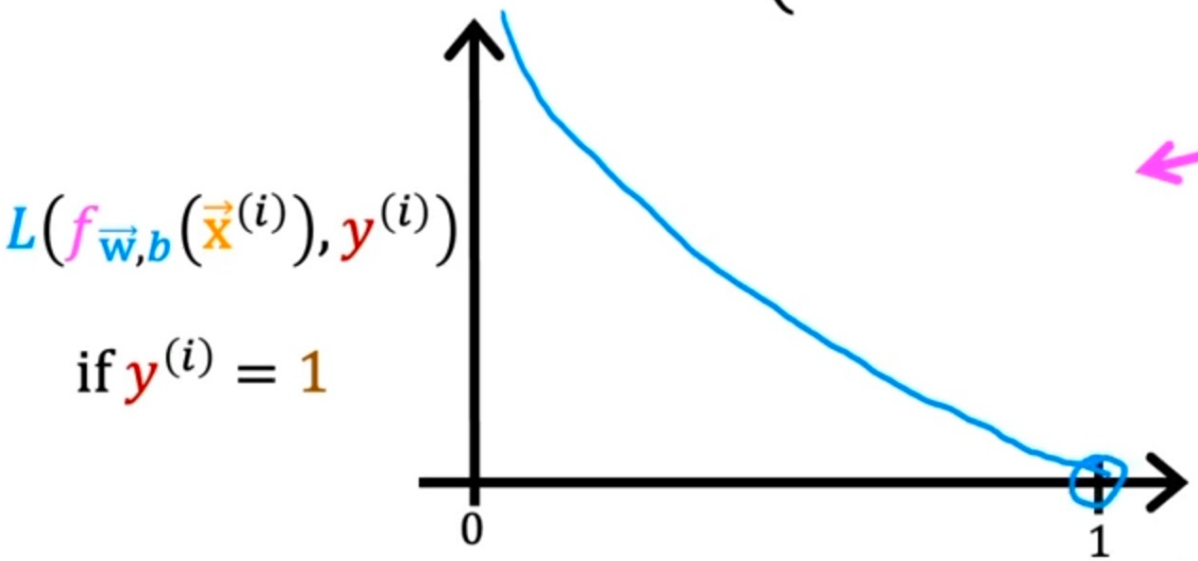
$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

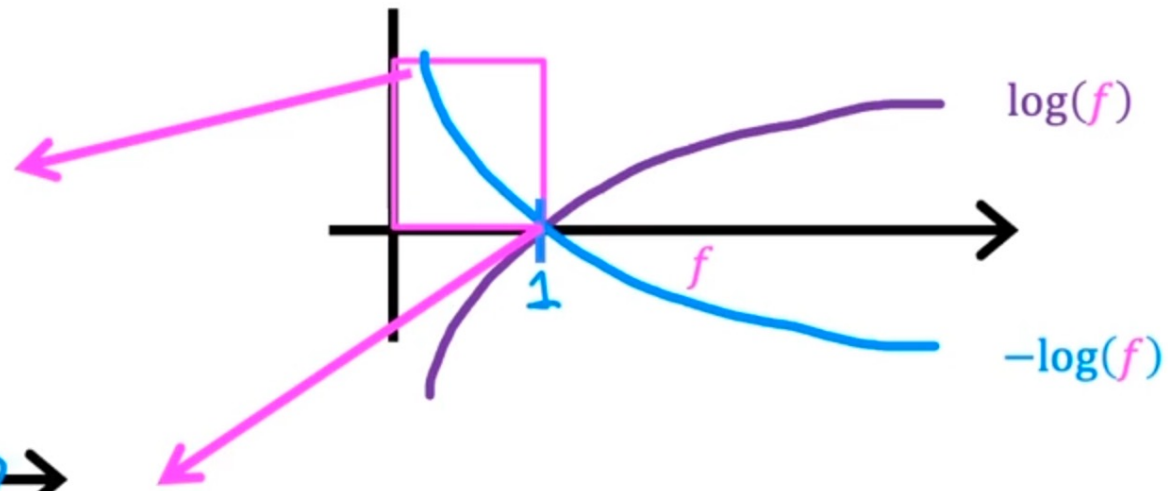


The only part of the function that's relevant is therefore this part over here, corresponding to f between 0 and 1.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$





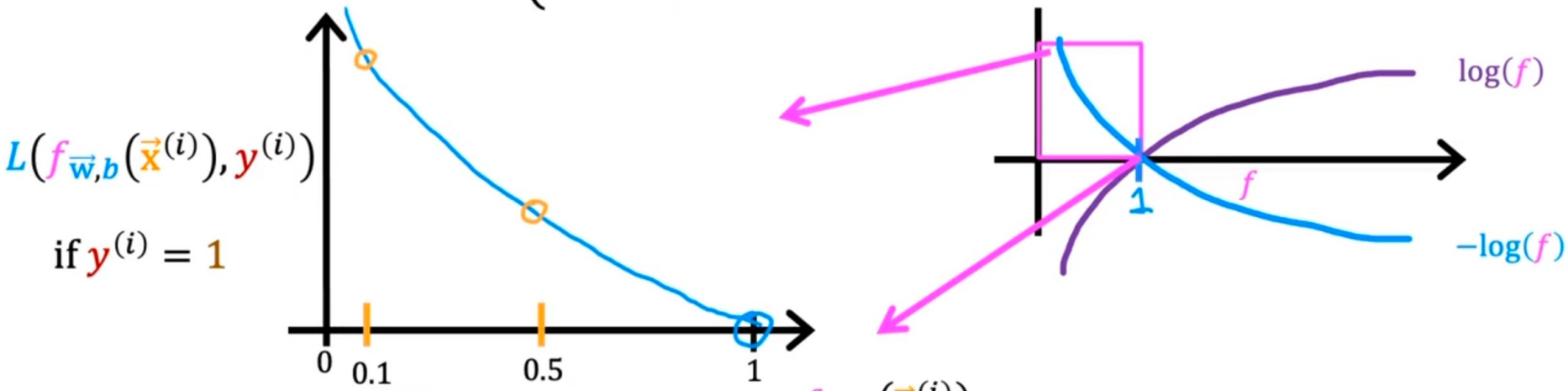
As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to 0$!!

$f_{\vec{w},b}\left(\vec{x}^{(i)}\right)$

Let's zoom in and take a closer look at this part of the graph. If the algorithm predicts a probability close to 1 and the true label is 1, then the loss is very small. It's pretty much 0 because you're very close to the right answer.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$



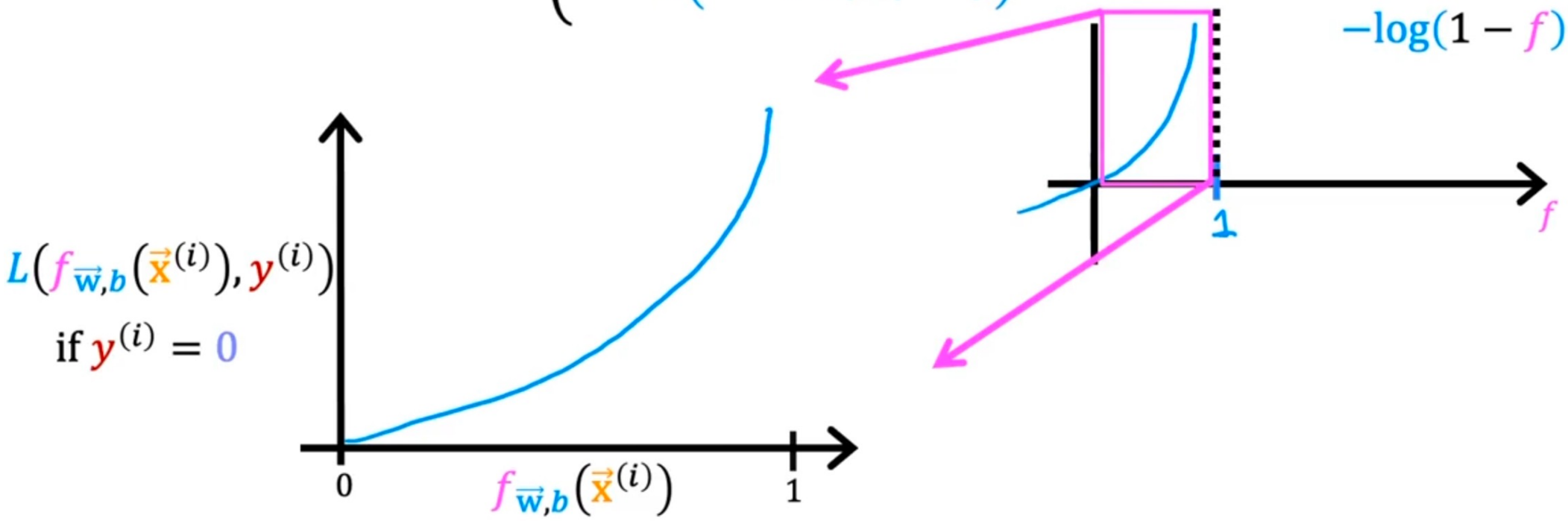
As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to \infty$

Now continue with the example of the true label y being 1, say everything is a malignant tumor. If the algorithm predicts 0.5, then the loss is at this point here, which is a bit higher but not that high. Whereas in contrast, if the algorithm were to have outputs at 0.1 if it thinks that there is only a 10 percent chance of the tumor being malignant but y really is 1. If really is malignant, then the loss is this much higher value over here. When y is equal to 1, the loss function incentivizes or nurtures, or helps push the algorithm to make more accurate predictions because the loss is lowest, when it predicts values close to 1.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

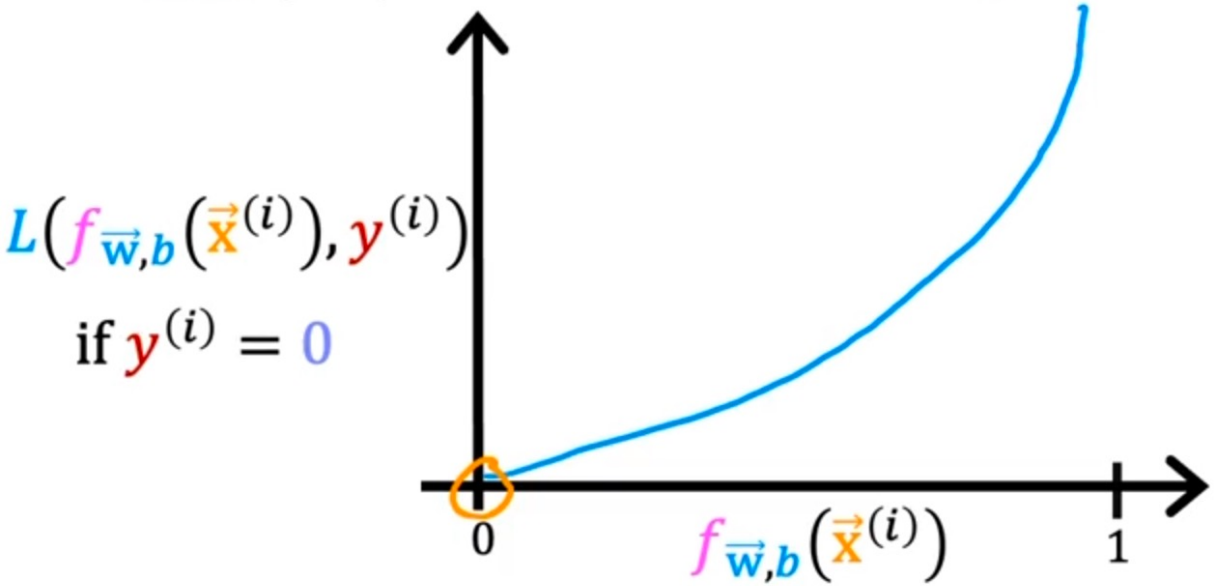


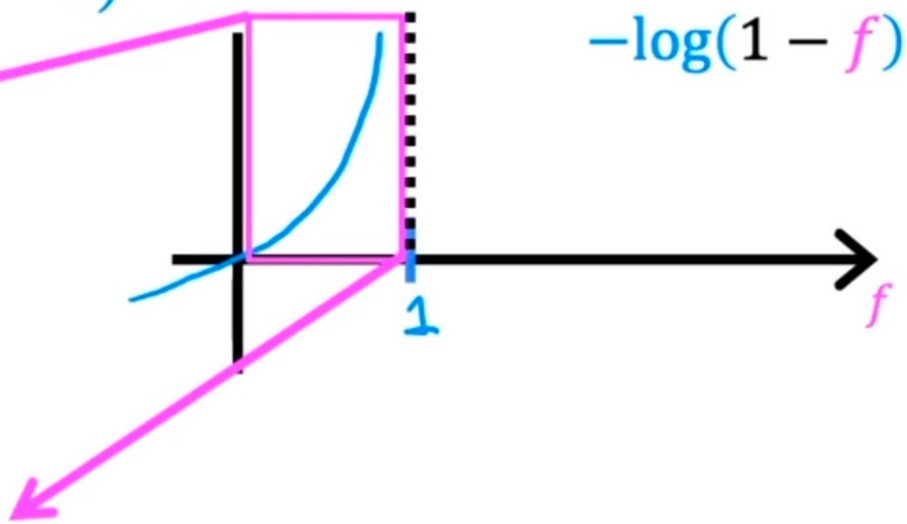
On this slide, let's look at the second part of the loss function corresponding to when y is equal to 0.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to 0$



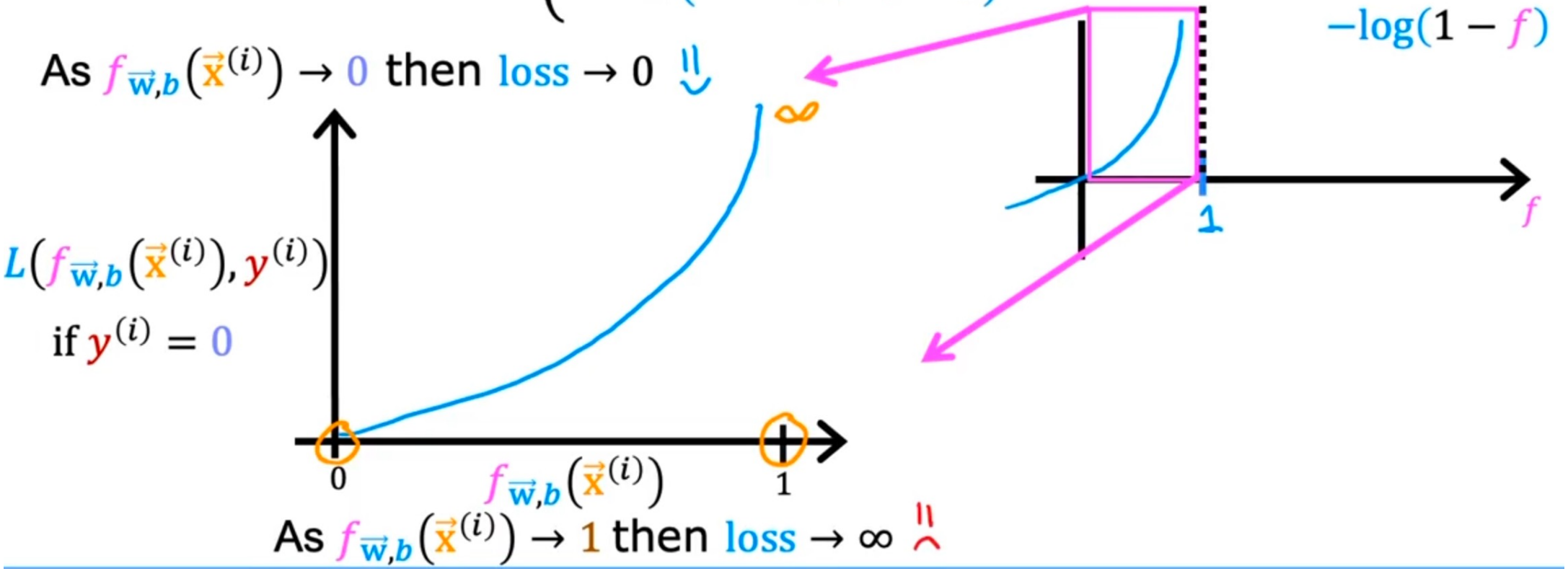


In this plot, corresponding to y equals 0, the vertical axis shows the value of the loss for different values of f of x. When f is 0 or very close to 0, the loss is also going to be very small which means that if the true label is 0 and the model's prediction is very close to 0, well, you nearly got it right so the loss is appropriately very close to 0.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to 0$



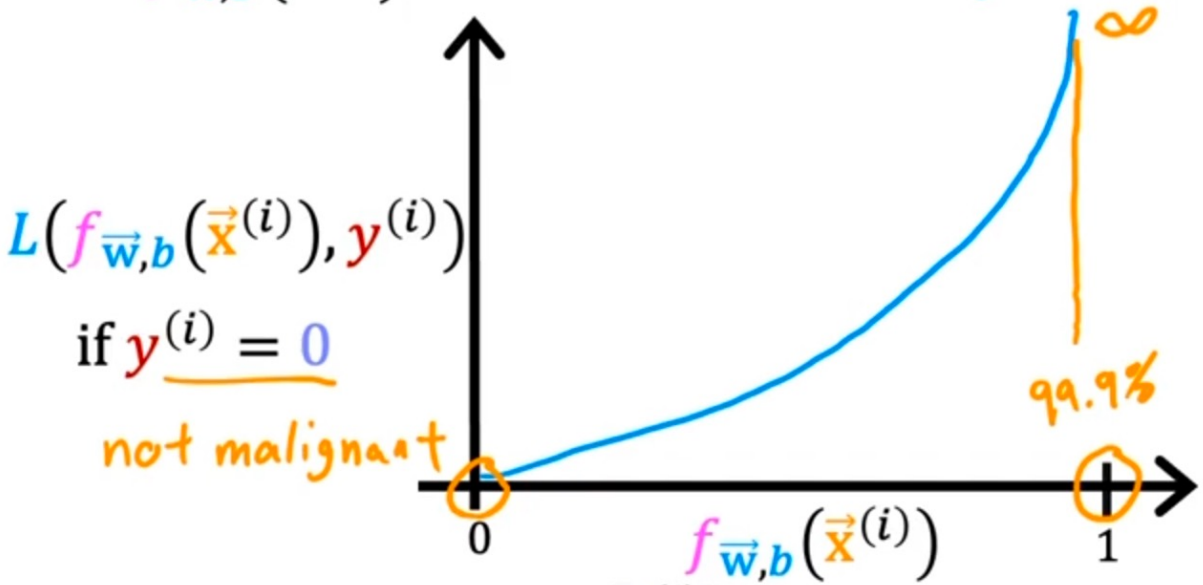
As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to \infty$

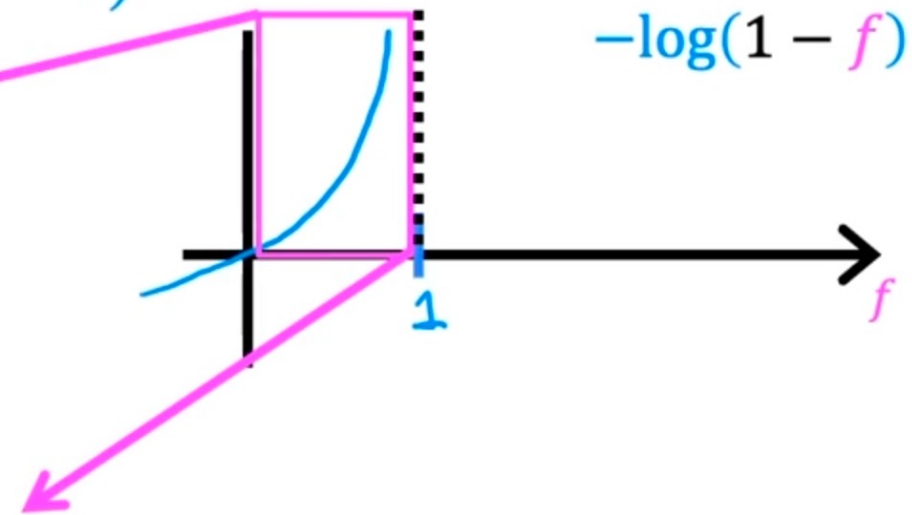
The larger the value of f of x gets, the bigger the loss because the prediction is further from the true label 0. In fact, as that prediction approaches 1, the loss actually approaches infinity.

Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to 0$





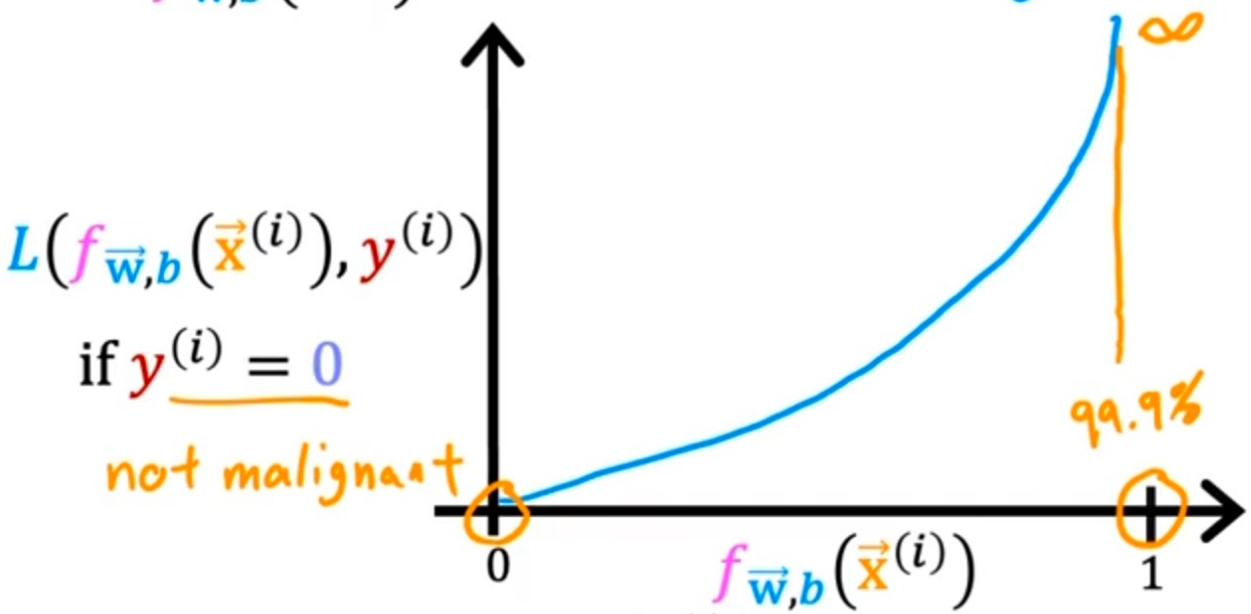
As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to \infty$

Going back to the tumor prediction example just says if the model predicts that the patient's tumor is almost certain to be malignant, say, 99.9 percent chance of malignancy, that turns out to actually not be malignant, so y equals 0 then we penalize the model with a very high loss.

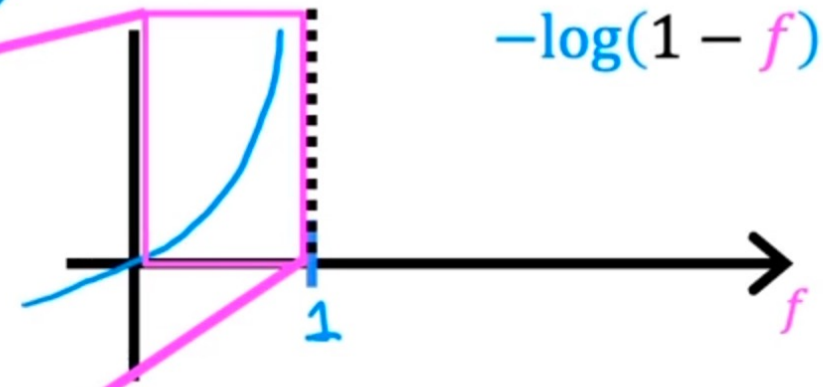
Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to 0$



As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to \infty$



The further prediction $f_{\vec{w},b}\left(\vec{x}^{(i)}\right)$ is from target $y^{(i)}$, the higher the loss.

Cost

cost

$$J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m} L\big(\underbrace{f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}), y^{(i)}}_{\text{loss}}\big)$$

$$= \begin{cases} -\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

convex
↳ can reach a global minimum

the squared error cost function doesn't work well for logistic regression. We also defined the loss for a single training example and came up with a new definition for the loss function for logistic regression. It turns out that with this choice of loss function, the overall cost function will be convex and thus you can reliably use gradient descent to take you to the global minimum.

Cost

cost

$$J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m} L\big(\underbrace{f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}), y^{(i)}}_{\text{loss}}\big)$$

$$= \begin{cases} -\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

convex
↳ can reach a global minimum

find w, b that minimize cost J

The cost function is a function of the entire training set and is, therefore, the average or 1 over m times the sum of the loss function on the individual training examples. The cost on a certain set of parameters, w and b, is equal to 1 over m times the sum of all the training examples of the loss on the training examples. If you can find the value of the parameters, w and b, that minimizes this, then you'd have a pretty good set of values for the parameters w and b for logistic regression.

jupyter Logistic Loss Last Checkpoint: 11/04/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

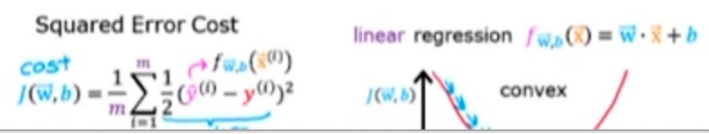
Optional Lab: Logistic Regression, Logistic Loss

In this ungraded lab, you will:

- explore the reason the squared error loss is not appropriate for logistic regression
- explore the logistic loss function

In [ ]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
plt.rcParams['font.size'] = 8
from plt_logistic_loss import  plt_logistic_cost, plt_two_logistic_loss_curves, plt_simple_example
from plt_logistic_loss import soup_bowl, plt_logistic_squared_error
```

Squared error for logistic regression?



In linear regression, the course utilized a squared error cost to measure how well the model predicted the training data.
Recall for **Linear Regression** the you used the **squared error cost function**: The equation for the squared error cost with one variable is:

Cost Function

Simplified Cost Function

Simplified loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}$$

Because we're still working on a binary classification problem, y is either zero or one. Because y is either zero or one and cannot take on any value other than zero or one, we'll be able to come up with a simpler way to write this loss function.

Simplified loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} \boxed{-\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)} & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\underbrace{y^{(i)}}_{1}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) \cancel{-\underbrace{\left(1 - y^{(i)}\right)}_{\underbrace{(1-1)}_{0}}\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}}$$

if $y^{(i)} = 1$:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \underbrace{-1 \log\left(f(\vec{x})\right)}$$

Simplified loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ \boxed{-\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)} & \text{if } y^{(i)} = 0 \end{cases}$$

$$\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\underbrace{\cancel{y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}}_{0} - \underbrace{\left(1 - y^{(i)}\right)}_{(1-0)}\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}$$

if $y^{(i)} = 1$:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -1 \log\left(f(\vec{x})\right)$$

if $y^{(i)} = 0$:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \underbrace{-(1-0)\log\left(1-f(\vec{x})\right)}$$

Simplified cost function

loss

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\, y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)\right]$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)\right]$$

Simplified cost function

loss

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\, y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)\right]$$

convex
(single global minimum)

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)\right]$$

maximum likelihood
(don't worry about it!)

jupyter Cost Function Last Checkpoint: 11/04/2021 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

Optional Lab: Cost Function for Logistic Regression

Goals

In this lab you will:

- examine the implementation and utilize the cost function for logistic regression.

In [*]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
plt.rcParams['font.size'] = 8
from lab_utils_common import  plot_data, sigmoid, dlc
```

Dataset

Let's start with the same dataset as was used in the decision boundary lab.

In [ ]:
```
import numpy as np
```

Gradient Descent

Gradient Descent Implementation

Training logistic regression

Find $\vec{w}, b$

Given new $\vec{x}$, output $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

$$P(y = 1|\vec{x}; \vec{w}, b)$$

To fit the parameters of a logistic regression model, we're going to try to find the values of the parameters w and b that minimize the cost function J of w and b, and we'll again apply gradient descent to do this. Let's take a look at how. In this, we'll focus on how to find a good choice of the parameters w and b. After you've done so, if you give the model a new input, x, say a new patients at the hospital with a certain tumor size and age, then these are diagnosis. The model can then make a prediction, or it can try to estimate the probability that the label y is one.

Gradient descent

cost

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right]$$

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

}

Gradient descent

cost

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right]$$

repeat {

$j = 1 \ldots n$

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$\frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

}

Gradient descent

cost

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right]$$

repeat {

$j = 1 \ldots n$

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

}

$$\frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$$\frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

Gradient descent

cost

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right]$$

repeat {

$j = 1 \ldots n$

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

} simultaneous updates

$$\frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

$$\frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right) \right]$$

} simultaneous updates

Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)} \right) \right]$$

} simultaneous updates

Linear regression $\quad f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b$

Logistic regression $\quad f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \dfrac{1}{1 + e^{-(\vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b)}}$

Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)} \right) \right]$$

} simultaneous updates

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

Linear regression $f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b$

Logistic regression $f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \dfrac{1}{1 + e^{-(\vec{\mathrm{w}} \cdot \vec{\mathrm{x}} + b)}}$

jupyter Gradient Descent Last Checkpoint: 11/04/2021 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Optional Lab: Gradient Descent for Logistic Regression

Goals

In this lab you will:

- explore the gradient descent update code for logistic regression.
- explore gradient descent on a familiar data set

In [ ]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
plt.rcParams['font.size'] = 8
import copy, math
from lab_utils_common import  dlc, plot_data, plt_tumor_data, sigmoid, compute_cost_logistic
from plt_quad_logistic import plt_quad_logistic, plt_prob
```

Data set

Let's start with the same two feature data set used in the decision boundary lab.

In [ ]:
```
X_train = np.array([[0.5, 1.5], [1,1], [1.5, 0.5], [3, 0.5], [2, 2], [1, 2.5]])
y_train = np.array([0, 0, 0, 1, 1, 1])
```

As before, we'll use a helper function to plot this data. The data points with label $y = 1$ are shown as red crosses, while the data points with label $y = 0$ are shown as black circles.

In [ ]:
```
fig,ax = plt.subplots(1,1,figsize=(4,4))
plot_data(X_train, y_train, ax)

ax.axis([0, 4, 0, 3.5])
ax.set_ylabel('$x_1$', fontsize=12)
ax.set_xlabel('$x_0$', fontsize=12)
plt.show()
```

jupyter Gradient Descent with Scikit Learn Last Checkpoint: 11/04/2021 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

Ungraded Lab: Logistic Regression using Scikit-Learn

Goals

In this lab you will:

- Train a logistic regression model using scikit-learn.

Dataset

Let's start with the same dataset as before.

In [ ]:
```
import numpy as np

X = np.array([[0.5, 1.5], [1,1], [1.5, 0.5], [3, 0.5], [2, 2], [1, 2.5]])
y = np.array([0, 0, 0, 1, 1, 1])
```

Fit the model

The code below imports the logistic regression model from scikit-learn. You can fit this model on the training data by calling `fit` function.

In [ ]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()
lr_model.fit(X, y)
```

Make Predictions

You can see the predictions made by this model by calling the `predict` function.

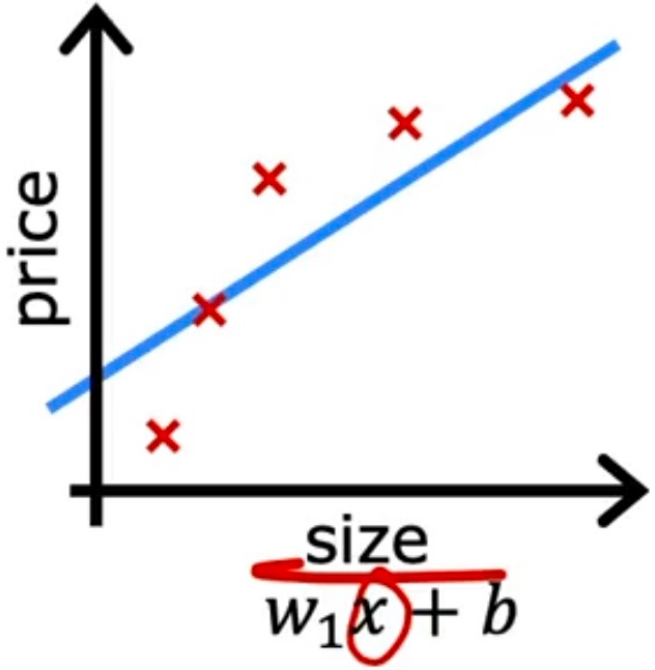
In [ ]:
```
y_pred = lr_model.predict(X)

print("Prediction on training set:", y_pred)
```

Regularization to Reduce Overfitting

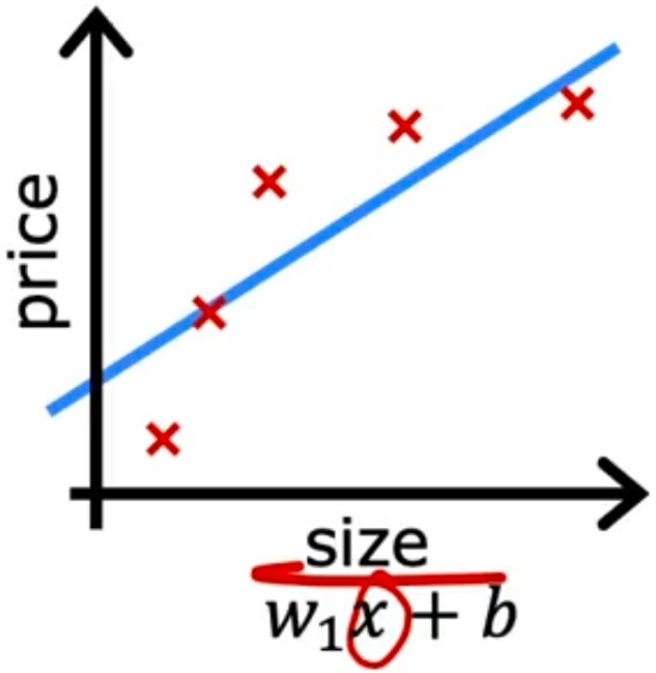
The Problem of Overfitting

Regression example



Suppose your data-set looks like this, with the input feature x being the size of the house, and the value, y that you're trying to predict the price of the house. One thing you could do is fit a linear function to this data. If you do that, you get a straight line fit to the data that maybe looks like this. But this isn't a very good model. Looking at the data, it seems pretty clear that as the size of the house increases, the housing process flattened out. This algorithm does not fit the training data very well. The technical term for this is the model is underfitting the training data.

Regression example



underfit

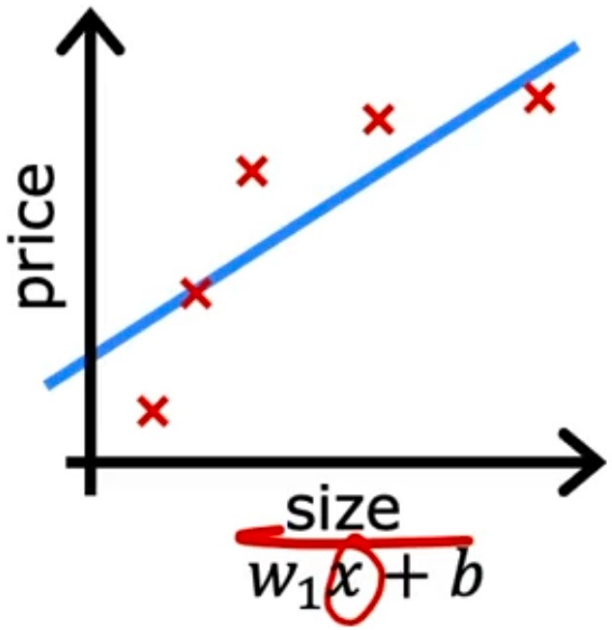
- Does not fit the training set well

high bias

The algorithm has underfit the data, meaning that it’s just not even able to fit the training set that well. There's a clear pattern in the training data that the algorithm is just unable to capture.

Another way to think of this form of bias is as if the learning algorithm has a very strong preconception, or we say a very strong bias, that the housing prices are going to be a completely linear function of the size despite data to the contrary. This preconception that the data is linear causes it to fit a straight line that fits the data poorly, leading it to underfitted data.

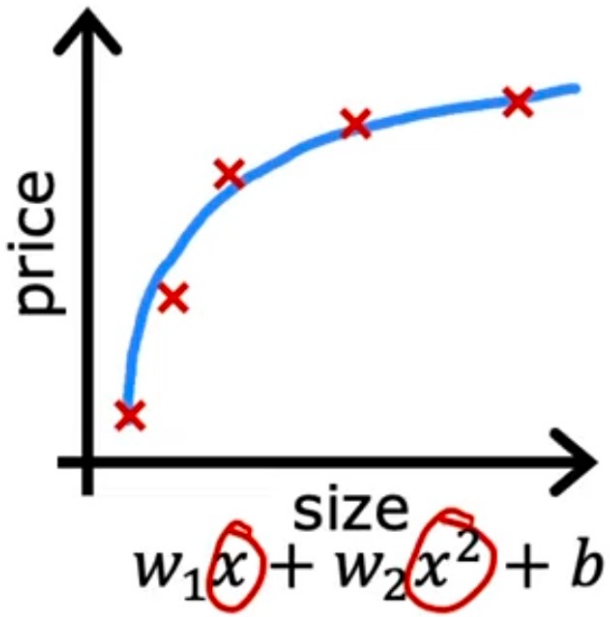
Regression example



underfit

- Does not fit the training set well

high bias

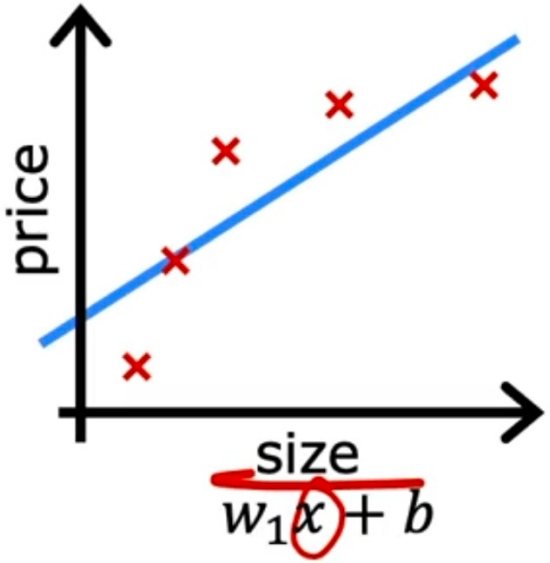


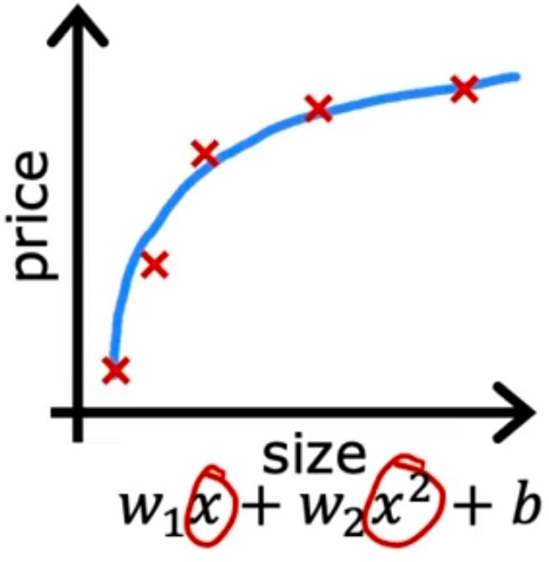
- Fits training set pretty well

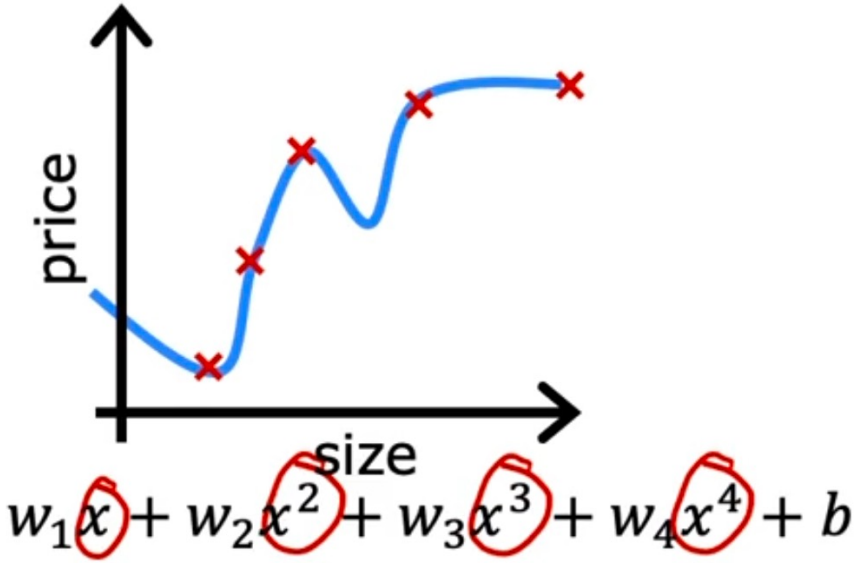
The idea that you want your learning algorithm to do well, even on examples that are not on the training set, that's called **generalization**.

Technically we say that you want your learning algorithm to generalize well, which means to make good predictions even on brand new examples that it has never seen before.

Regression example







underfit

- Does not fit the training set well

high bias

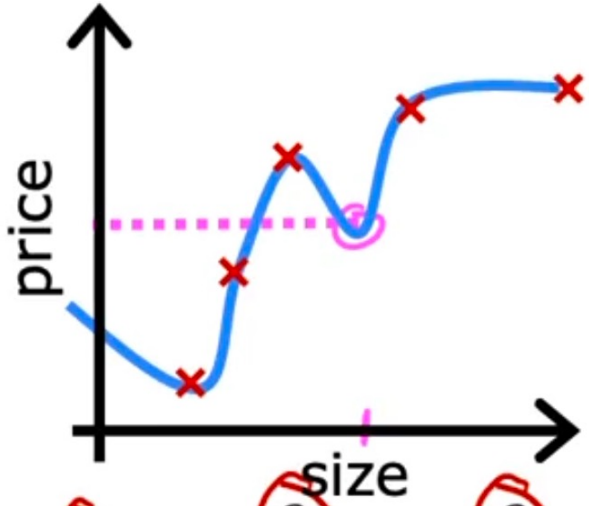
- Fits training set pretty well

generalization

- Fits the training set extremely well

In fact, you'd be able to choose parameters that will result in the cost function being exactly equal to zero because the errors are zero on all five training examples. But this is a very wiggly curve, its going up and down all over the place.

Regression example



$$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

overfit

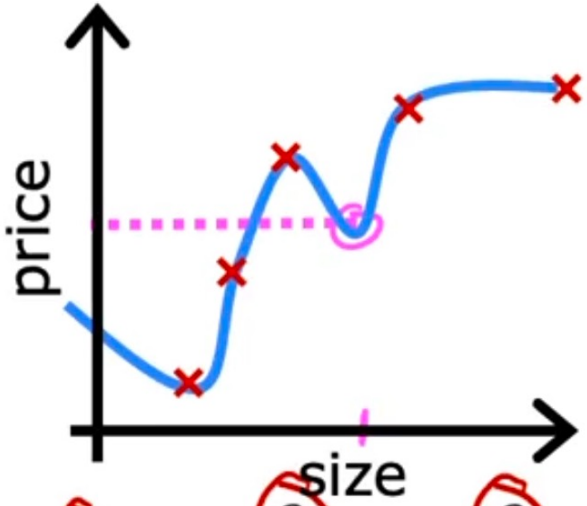
- Fits the training set extremely well

high variance

If you have this whole size right here, the model would predict that this house is cheaper than houses that are smaller than it. We don't think that this is a particularly good model for predicting housing prices. The technical term is that we'll say this model has overfit the data, or this model has an overfitting problem. Because even though it fits the training set very well, it has fit the data almost too well, hence is overfit.

It does not look like this model will generalize to new examples that's never seen before. Another term for this is that the algorithm has high variance. In machine learning, many people will use the terms over-fit and high-variance almost interchangeably.

Regression example



$$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

overfit

- Fits the training set extremely well

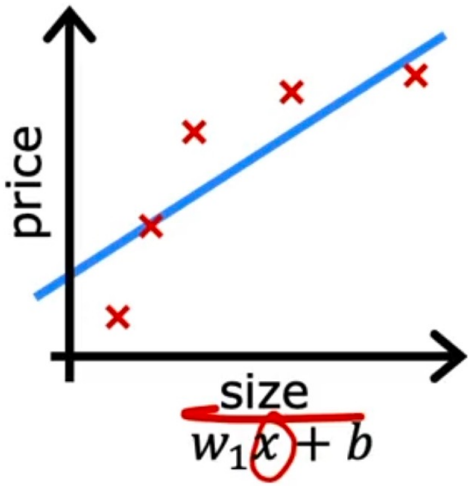
high variance

The intuition behind overfitting or high-variance is that the algorithm is trying very hard to fit every single training example.

It turns out that if your training set were just even a little bit different, say one house was priced just a little bit more little bit less, then the function that the algorithm fits could end up being totally different.

If two different machine learning engineers were to fit this fourth-order polynomial model, to just slightly different datasets, they couldn't end up with totally different predictions or highly variable predictions. That's why we say the algorithm has high variance.

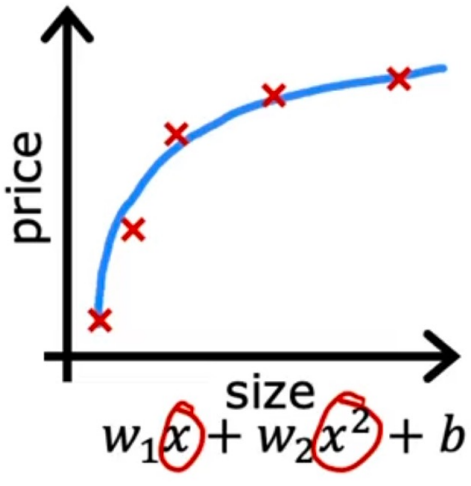
Regression example



underfit

- Does not fit the training set well

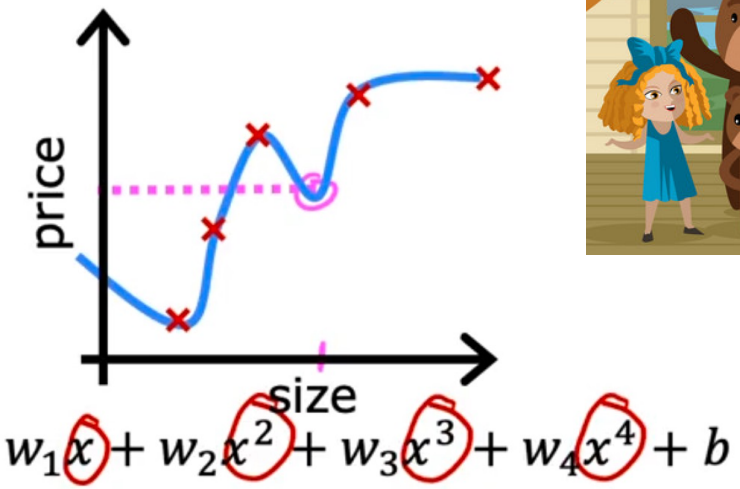
high bias



just right

- Fits training set pretty well

generalization



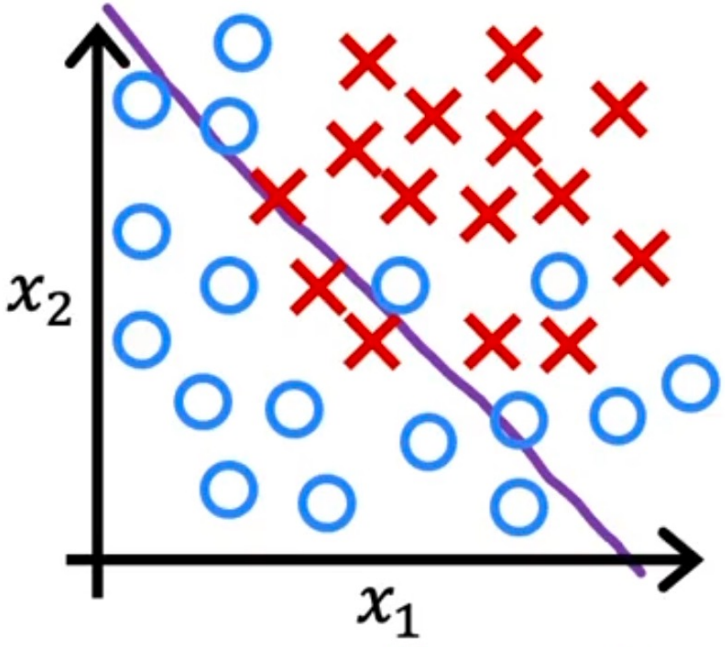
overfit

- Fits the training set extremely well

high variance

Contrasting this rightmost model with the one in the middle for the same house, it seems, the middle model gives them much more reasonable prediction for price. There isn't really a name for this case in the middle, but I'm just going to call this just right, because it is neither underfit nor overfit. You can say that the goal machine learning is to find a model that hopefully is neither underfitting nor overfitting. In other words, hopefully, a model that has neither high bias nor high variance.

Classification



$$z = w_1 x_1 + w_2 x_2 + b$$

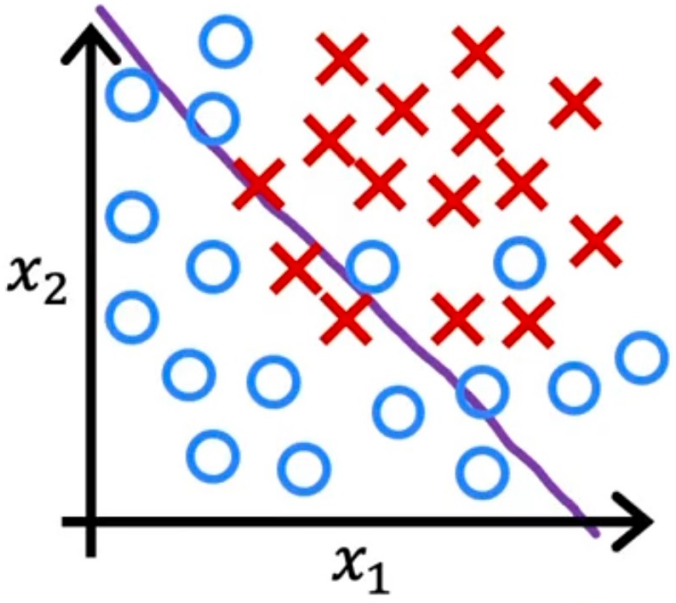
$$f_{\vec{w},b}(\vec{x}) = g(z)$$

g is the sigmoid function

underfit high bias

Here's a classification example with two features, x_1 and x_2, where x_1 is maybe the tumor size and x_2 is the age of patient. We're trying to classify if a tumor is malignant or benign, as denoted by these crosses and circles, one thing you could do is fit a logistic regression model. Just a simple model like this, where as usual, g is the sigmoid function and this term here inside is z. If you do that, you end up with a straight line as the decision boundary. This is the line where z is equal to zero that separates the positive and negative examples. This straight line doesn't look terrible. It looks okay, but it doesn't look like a very good fit to the data either. This is an example of underfitting or of high bias.

Classification

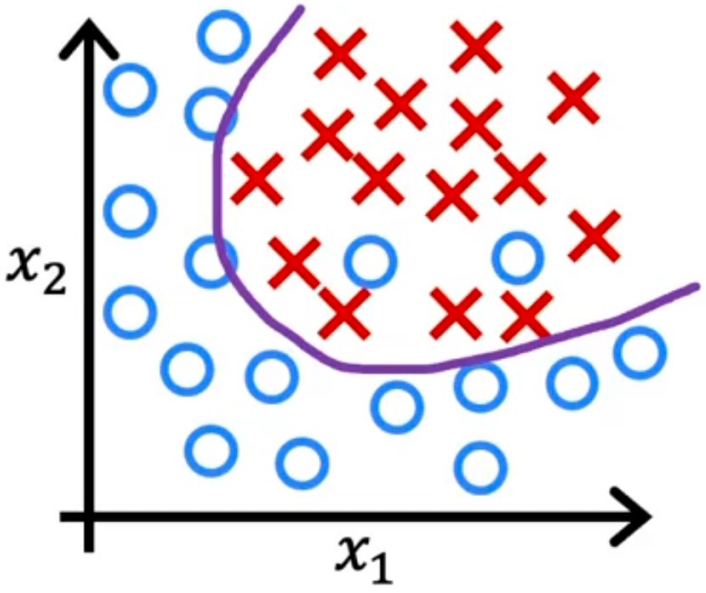


$$z = w_1 x_1 + w_2 x_2 + b$$

$$f_{\vec{w},b}(\vec{x}) = g(z)$$

g is the sigmoid function

underfit high bias



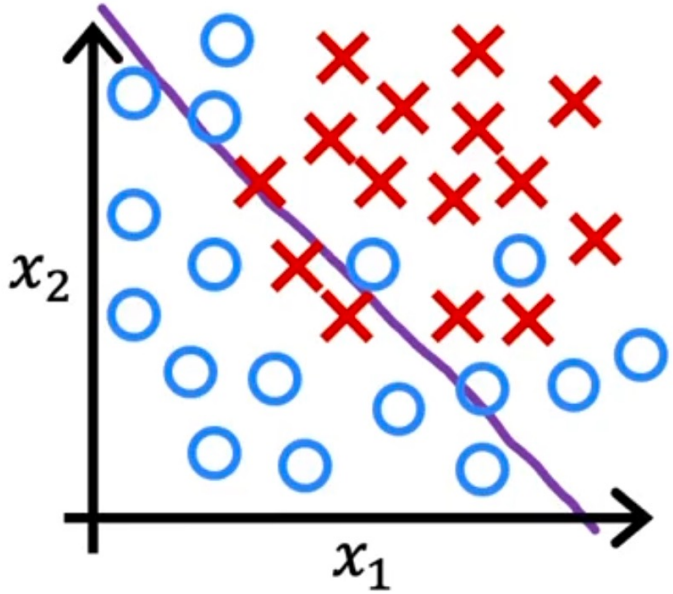
$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$$

just right

Let's look at another example. If you were to add to your features these quadratic terms, then z becomes this new term in the middle and the decision boundary, that is where z equals zero can look more like this, more like an ellipse or part of an ellipse.

This is a pretty good fit to the data, even though it does not perfectly classify every single training example in the training set. Notice how some of these crosses get classified among the circles. But this model looks pretty good. I'm going to call it just right. It looks like this generalized pretty well to new patients.

Classification

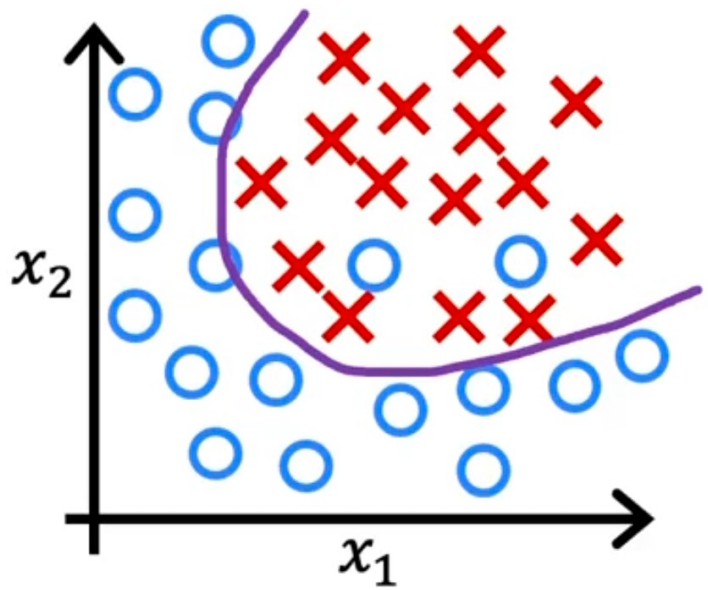


$$z = w_1 x_1 + w_2 x_2 + b$$

$$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = g(z)$$

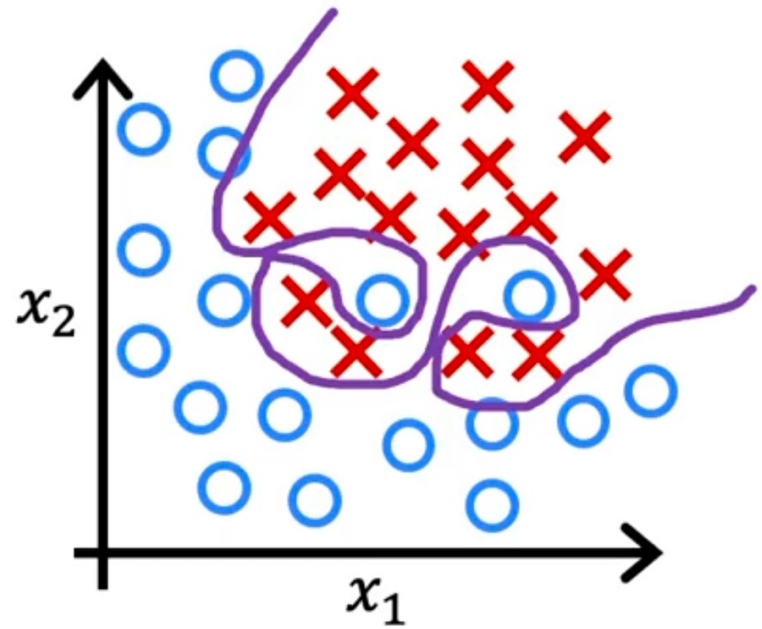
g is the sigmoid function

underfit high bias



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$$

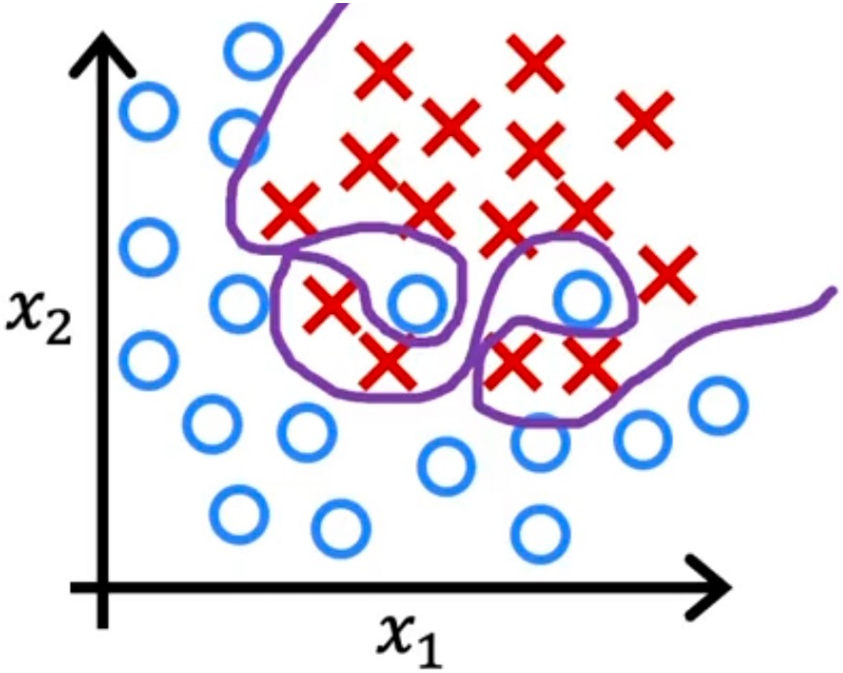
just right



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 + \cdots + b$$

overfit

Classification



$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + w_6x_1^3x_2 + \cdots + b$$

overfit

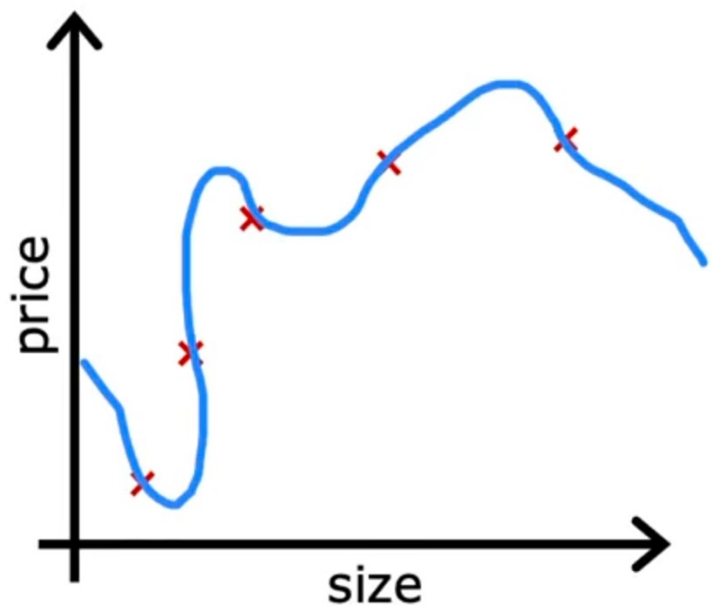
Finally, at the other extreme, if you were to fit a very high order polynomial with many features like these, then the model may try really hard and contoured or twist itself to find a decision boundary that fits your training data perfectly. Having all these higher-order polynomial features allows the algorithm to choose this really over the complex decision boundary. If the features are tumor size in age, and you're trying to classify tumors as malignant or benign, then this doesn't really look like a very good model for making predictions.

Once again, this
is an instance of overfitting and high variance because its model, despite doing very well on the training set, doesn't look like it'll generalize well to new examples.

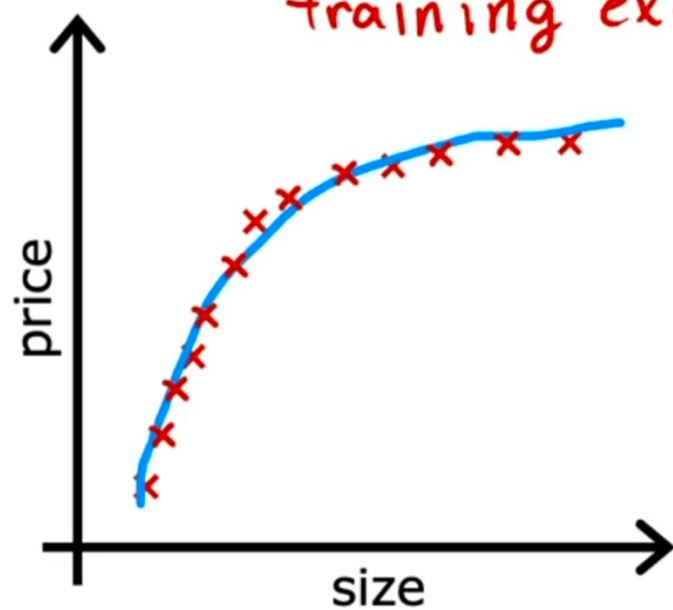
Regularization to Reduce Overfitting

Addressing Overfitting

Collect more training examples



overfit



collect more training examples

Select features to include/exclude

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price

Previousluy, our models features included the size x, as well as the size squared, and this x squared, and x cubed and x^4 and so on. These were a lot of polynomial features. In that case, one way to reduce overfitting is to just not use so many of these polynomial features.

Select features to include/exclude

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

all features

+

insufficient data

↓

overfit

Select features to include/exclude

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

all features

+

insufficient data

↓

overfit

selected features

size
bedrooms
age
just right
feature selection

Now instead of using all 100 features, if we were to pick just a subset of the most useful ones, maybe size, bedrooms, and the age of the house. If you think those are the most relevant features, then using just that smallest subset of features, you may find that your model no longer overfits as badly. Choosing the most appropriate set of features to use is sometimes also called feature selection.

Select features to include/exclude

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

all features

+

insufficient data

↓

overfit

selected features

size
bedrooms
age
just right
feature selection

disadvantage

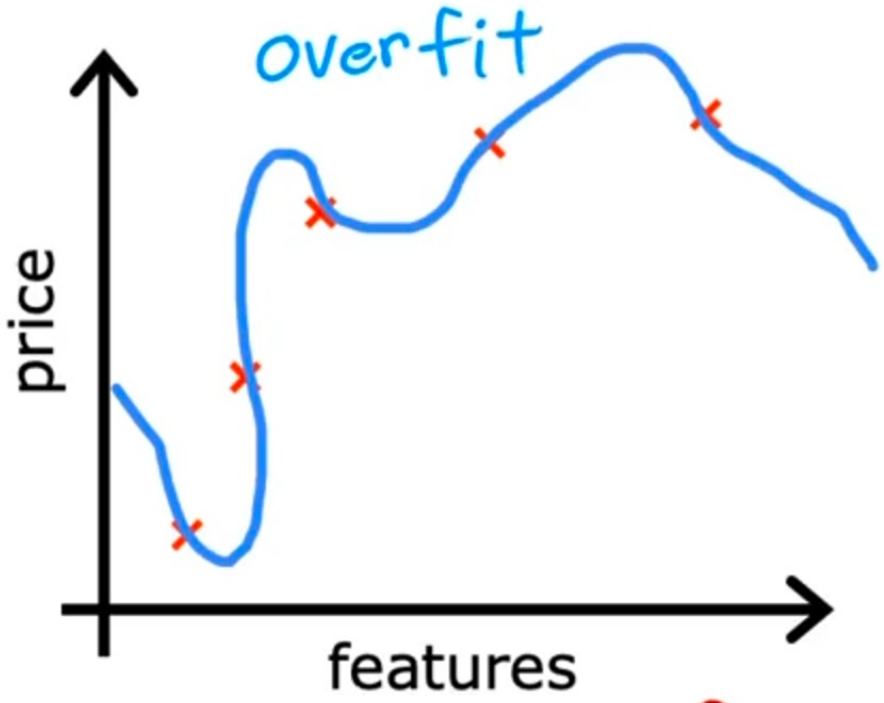
↓

useful features could be lost

Algorithms are available for automatically choosing the most appropriate set of features to use for our prediction task.

Regularization

Reduce the size of parameters w_j



$$f(x) = 28x - 385x^2 + 39x^3 - \underbrace{\cancel{174}x^4}_{0} + 100$$

large values for w_j

eliminate feature

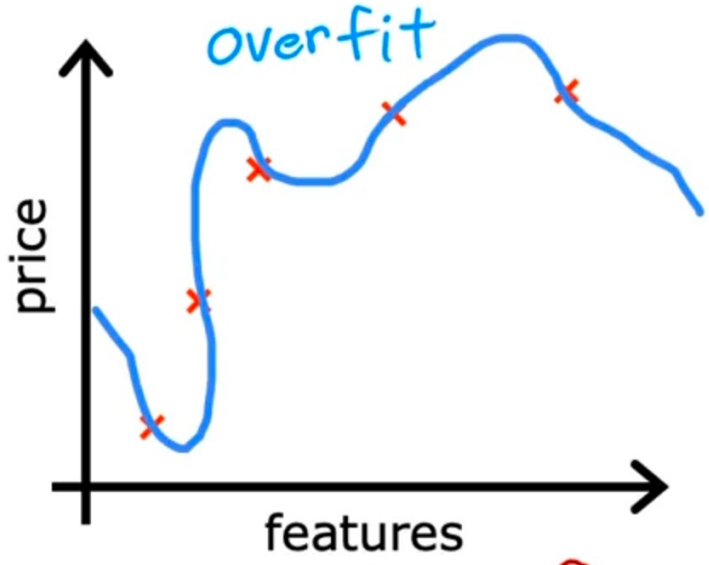
If you look at an overfit model, here's a model using polynomial features: x, x squared, x cubed, and so on. You find that the parameters are often relatively large.

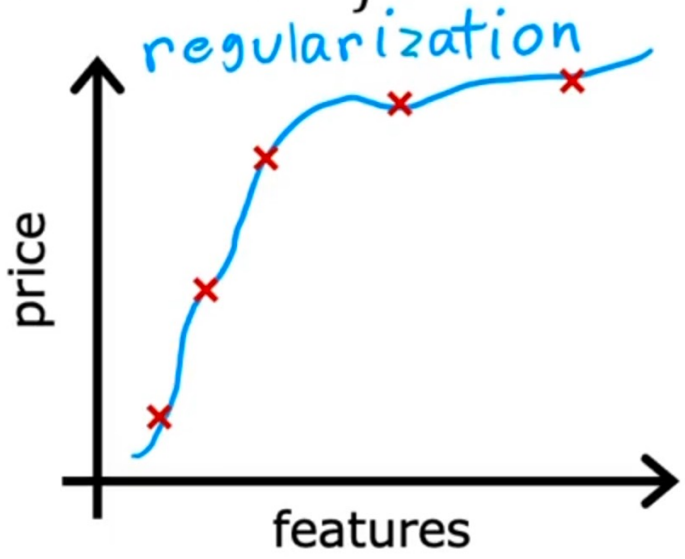
Now if you were to eliminate some of these features, say, if you were to eliminate the feature x4, that corresponds to setting this parameter to 0. So setting a parameter to 0 is equivalent to eliminating a feature, which is what we saw on the previous slide.

It turns out that regularization is a way to more gently reduce the impacts of some of the features without doing something as harsh as eliminating it outright.

Regularization

Reduce the size of parameters w_j





$$f(x) = 28x - 385x^2 + 39x^3 - \underbrace{\cancel{174}x^4}_{0} + 100$$

large values for w_j

eliminate feature

$$f(x) = 13x - 0.23x^2 + \underbrace{0.000014}x^3 - \underbrace{0.0001}\,x^4 + 10$$

small values for w_j

What regularization does is encourage the learning algorithm to shrink the values of the parameters without necessarily demanding that the parameter is set to exactly 0. It turns out that even if you fit a higher order polynomial like this, so long as you can get the algorithm to use smaller parameter values: w1, w2, w3, w4. You end up with a curve that ends up fitting the training data much better. So what regularization does, is it lets you keep all of your features, but they just prevents the features from having an overly large effect, which is what sometimes can cause overfitting.

Addressing overfitting

Options

1. Collect more data

2. Select features
   – Feature selection

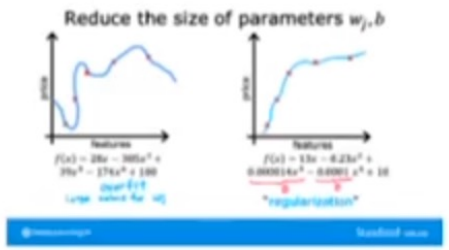
3. Reduce size of parameters
   – “Regularization”

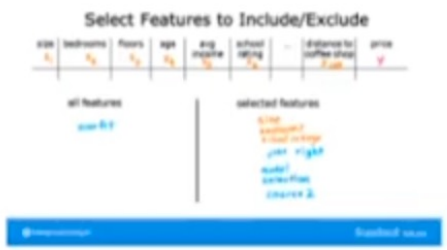
jupyter Overfitting Last Checkpoint: 3 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Ungraded Lab: Overfitting





Goals

In this lab, you will explore:

- the situations where overfitting can occur
- some of the solutions

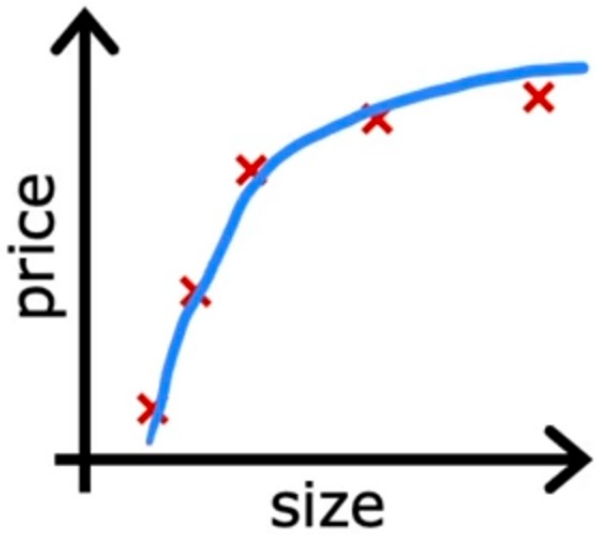
In [ ]:
```
%matplotlib widget
import matplotlib.pyplot as plt
from ipywidgets import Output
from plt_overfit import overfit_example, output
plt.style.use('./deeplearning.mplstyle')
```

Overfitting

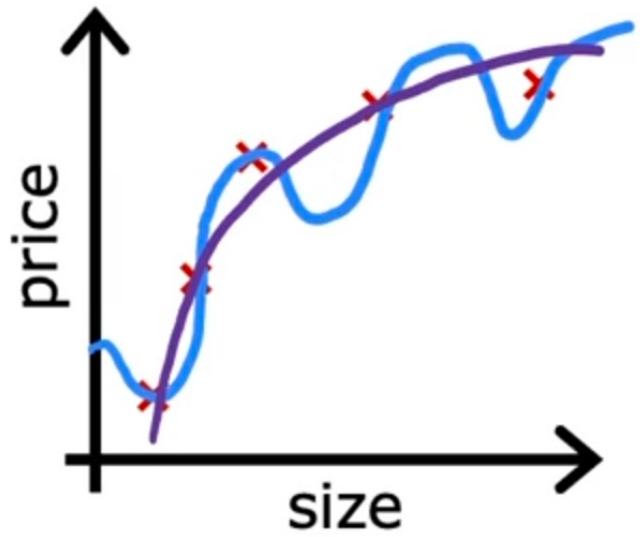
Regularization to Reduce Overfitting

Cost Function with Regularization

Intuition



$w_1 x + w_2 x^2 + b$

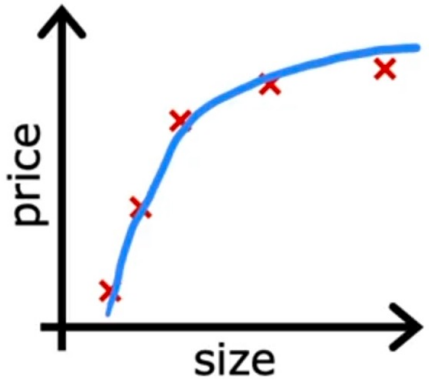


$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

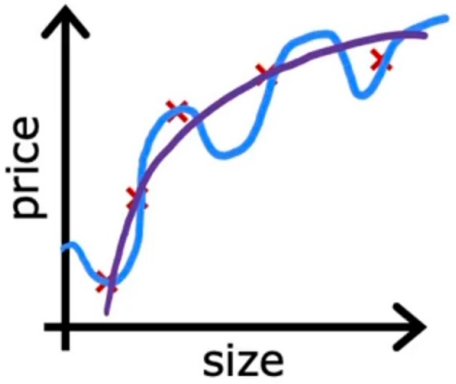
make w_3, w_4 really small (≈ 0)

$$\min_{\vec{\mathrm{w}},b} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)^2 + 1000\, w_3^2 + 1000\, w_4^2$$

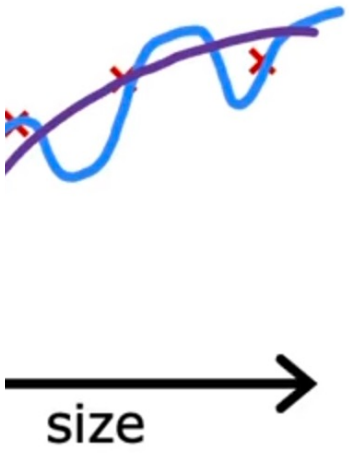
Intuition



$$w_1x + w_2x^2 + b$$



$$w_1x + w_2x^2 + \underbrace{\cancel{w_3}}_{\approx 0}x^3 + \underbrace{\cancel{w_4}}_{\approx 0}x^4 + b$$



$$^2 + w_3x^3 + w_4x^4 + b$$

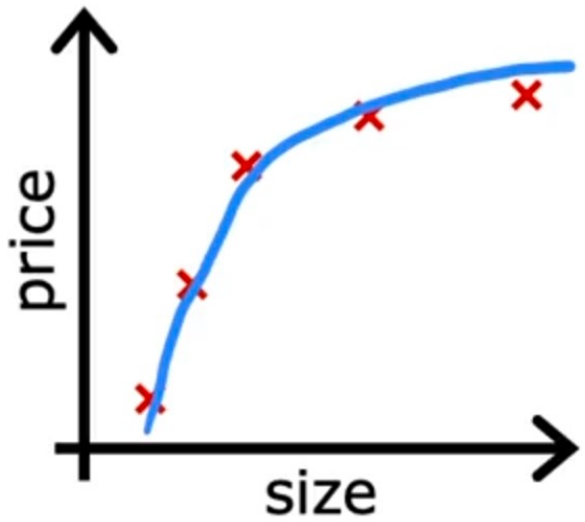
make w_3, w_4 really small (≈ 0)

$$\min_{\vec{w},b} \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + 1000\,\underbrace{w_3^2}_{0.001} + 1000\,\underbrace{w_4^2}_{0.002}$$

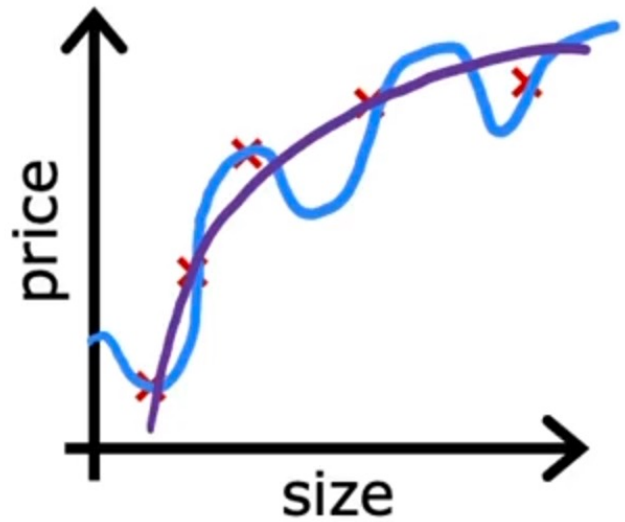
$$\min_{\vec{w},b} \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + 1000\,\underbrace{w_3^2}_{0.001} + 1000\,\underbrace{w_4^2}_{0.002}$$

if you want to minimize this function, the only way to make this new cost function small is if W3 and W4 are both small.

Intuition



$$w_1 x + w_2 x^2 + b$$



$$w_1 x + w_2 x^2 + \cancel{\underbrace{w_3}_{\approx 0} x^3} + \cancel{\underbrace{w_4}_{\approx 0} x^4} + b$$

make w_3, w_4 really small (≈ 0)

$$\min_{\vec{\mathrm{w}},b} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)^2 + \cancel{1000 \underbrace{w_3^2}_{0.001}} + \cancel{1000 \underbrace{w_4^2}_{0.002}}$$

Regularization

small values $w_1, w_2, \cdots, w_n, b$

simpler model

less likely to overfit

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right)^2$$

Regularization

small values $w_1, w_2, \cdots, w_n, b$

simpler model

less likely to overfit

$w_3 \approx 0$

$w_4 \approx 0$

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

n features

The way regularization is typically implemented is to penalize all of the features or more precisely, you penalize all the WJ parameters and it's possible to show that this will usually result in fitting a smoother simpler, less weekly function that's less prone to overfitting.

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right)^2$$

Regularization

small values $w_1, w_2, \cdots, w_n, b$

simpler model

less likely to overfit

$w_3 \approx 0$

$w_4 \approx 0$

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

$n = 100$

$w_1, w_1, w_2, \cdots, w_{100}, b$

n features

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

"lambda"

regularization parameter

$\lambda > 0$

Regularization

small values $w_1, w_2, \cdots, w_n, b$

simpler model

less likely to overfit

$w_3 \approx 0$

$w_4 \approx 0$

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

$n = 100$

$w_1, w_1, w_2, \cdots, w_{100}, b$

n features

regularization term

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

"lambda"

regularization parameter $\lambda > 0$

Regularization

small values $w_1, w_2, \cdots, w_n, b$

simpler model

less likely to overfit

$w_3 \approx 0$

$w_4 \approx 0$

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

$n = 100$

$w_1, w_1, w_2, \cdots, w_{100}, b$

n features

regularization term

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 + \frac{\lambda}{2m}b^2\right]$$

can include or exclude b

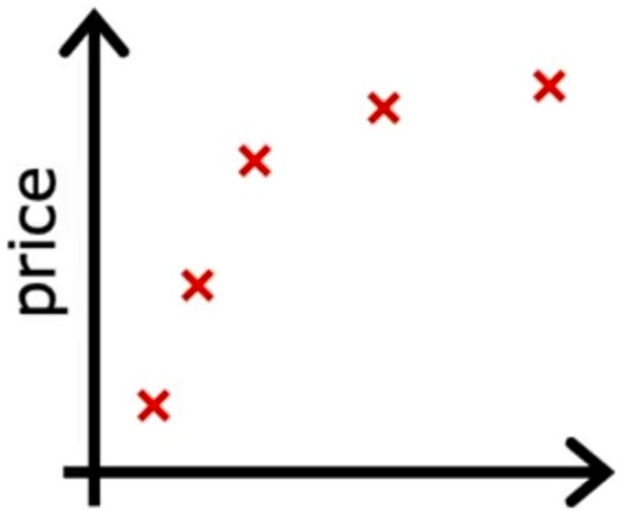
"lambda"

regularization parameter

$\lambda > 0$

Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\overbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}}\right]$$



$$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

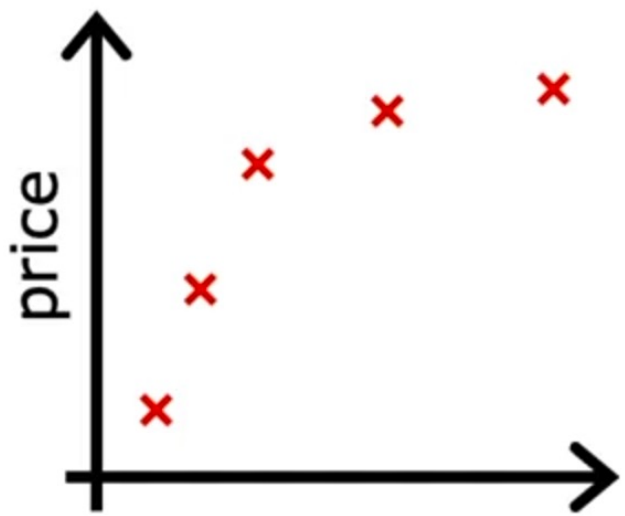
Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\overbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}}\right]$$

fit data

Keep w_j small

λ balances both goals



$$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

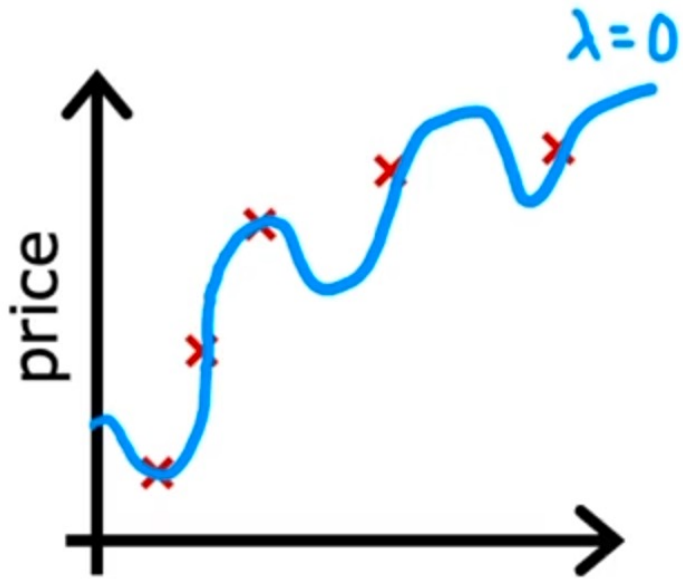
Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b} \left[\overbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}} \right]$$

fit data

Keep w_j small

λ balances both goals



$$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

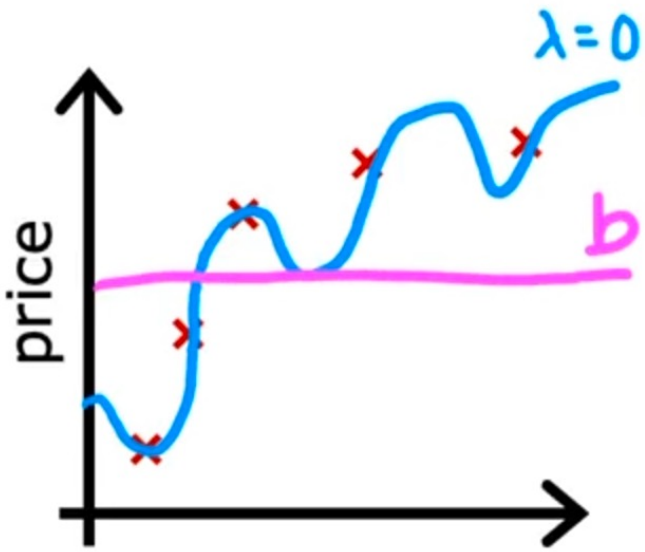
Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b} \left[\overbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}} \right]$$

fit data

Keep w_j small

λ balances both goals



choose $\lambda = 10^{10}$

$$f_{\vec{w},b}(\vec{x}) = \underset{\approx 0}{\cancel{w_1 x}} + \underset{\approx 0}{\cancel{w_2 x^2}} + \underset{\approx 0}{\cancel{w_3 x^3}} + \underset{\approx 0}{\cancel{w_4 x^4}} + b$$

$$f(x) = b$$

If lambda is 0 this model will over fit If lambda is enormous like 10 to the power of 10. This model will under fit.

Regularization to Reduce Overfitting

Regularized Linear Regression

Regularized linear regression

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$$

} simultaneous update

Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$j = 1, \ldots, n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

} simultaneous update

Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m} w_j$$

$j = 1, \ldots, n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

} simultaneous update

don't have to regularize b

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

} simultaneous update

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

} simultaneous update $j = 1 \ldots n$

$$w_j = \underbrace{1 w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \left(1 - \alpha \frac{\lambda}{m} \right)} \underbrace{- \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{\text{usual update}}$$

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

} simultaneous update $j = 1 \ldots n$

$$w_j = \underbrace{1 w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \underbrace{\left(1 - \alpha \frac{\lambda}{m}\right)}_{\text{shrink } w_j}} \underbrace{- \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{\text{usual update}}$$

$$\alpha \frac{\lambda}{m}$$

$$0.01 \frac{1}{50} = 0.0002$$

$$w_j (\underbrace{1 - 0.0002}_{0.9998})$$

How we get the derivative term (optional)

$$\frac{\partial}{\partial w_j} J(\vec{w}, b) = \frac{\partial}{\partial w_j}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(\underbrace{f(\vec{x}^{(i)})}_{\vec{w}\cdot\vec{x}^{(i)}+b} - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

$$= \frac{1}{\cancel{2}m}\sum_{i=1}^{m}\left[\left(\vec{w}\cdot\vec{x}^{(i)} + b - y^{(i)}\right)\cancel{2}x_j^{(i)}\right] + \frac{\lambda}{\cancel{2}m}\cancel{2}w_j \quad \text{No } \sum_{j=1}^{n}$$

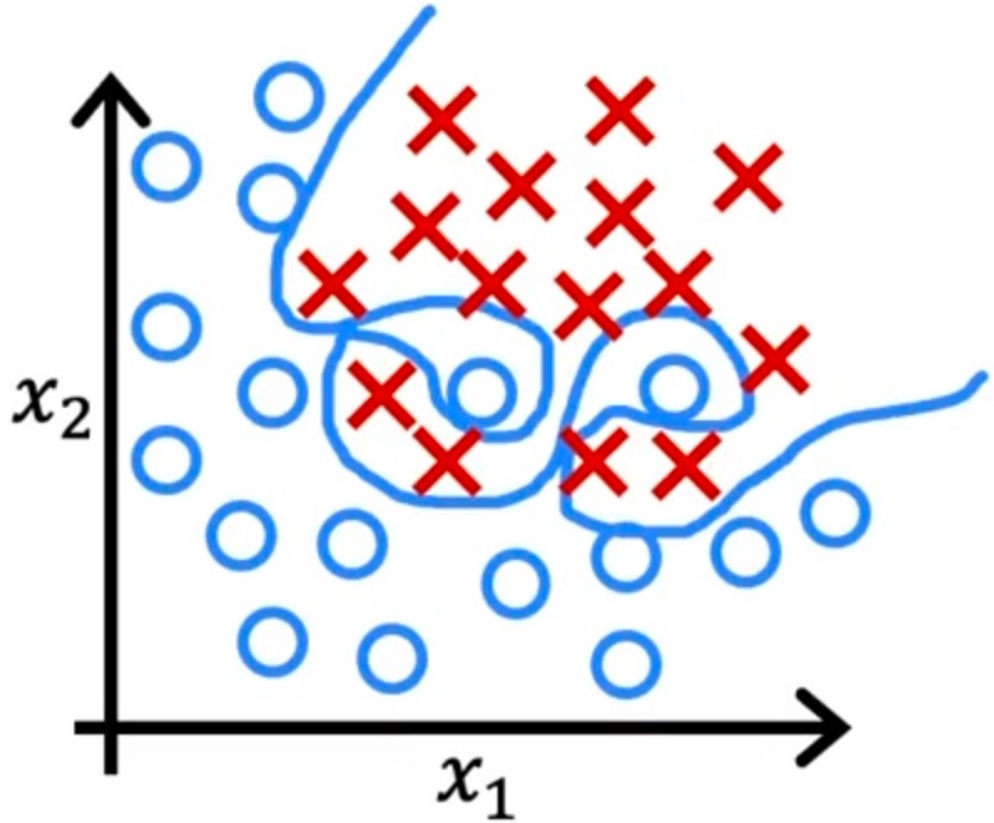
$$= \frac{1}{m}\sum_{i=1}^{m}\left[\left(\underbrace{\vec{w}\cdot\vec{x}^{(i)} + b}_{f(\vec{x})} - y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m}w_j$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m}w_j$$

Regularization to Reduce Overfitting

Regularized Logistic Regression

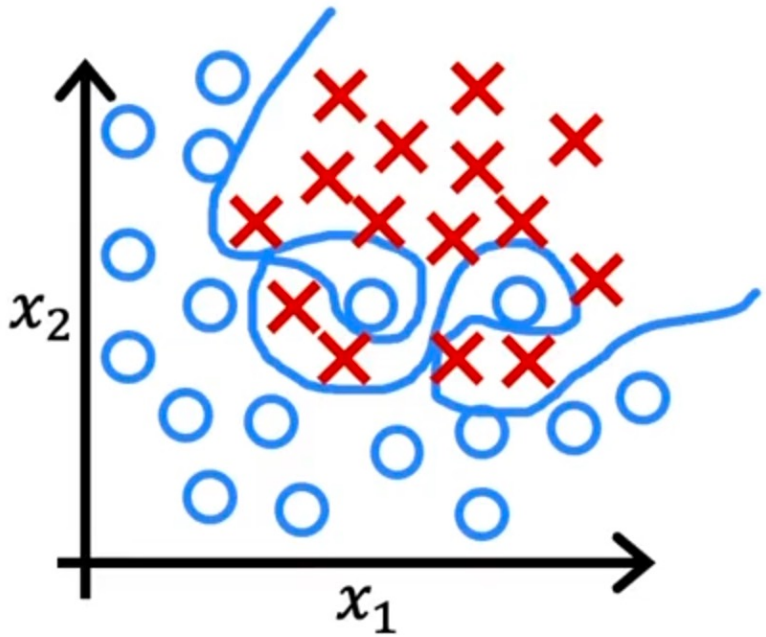
Regularized logistic regression



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + \cdots + b$$

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-z}}$$

Regularized logistic regression



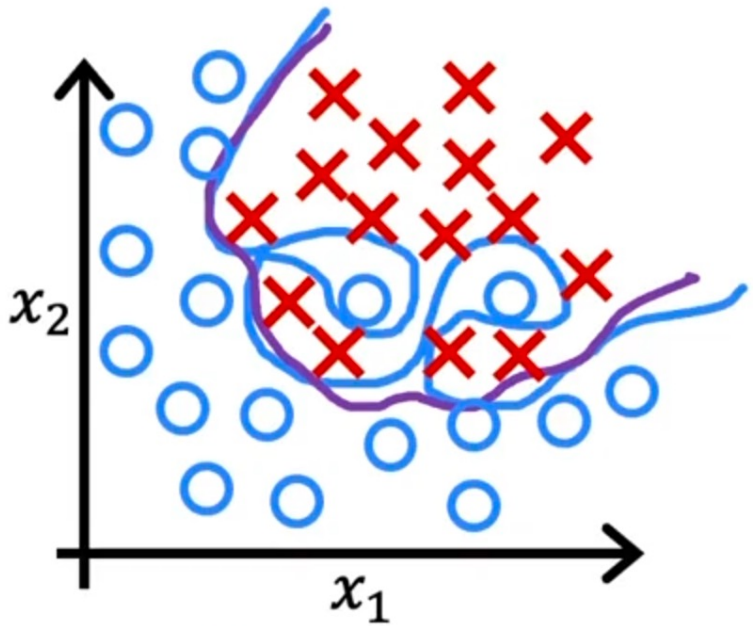
$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + \cdots + b$$

$$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \frac{1}{1+e^{-z}}$$

Cost function

$$J(\vec{\mathrm{w}},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Regularized logistic regression



$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + \cdots + b$$

$$f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}) = \frac{1}{1+e^{-z}}$$

Cost function

$$J(\vec{\mathrm{w}},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$$\min_{\vec{\mathrm{w}},b} J(\vec{\mathrm{w}},b) \rightarrow w_j \downarrow$$

Regularized logistic regression

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$\min_{\vec{\mathrm{w}},b}$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$j = 1 \ldots n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

}

Regularized logistic regression

$$J(\vec{\mathrm{w}}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$\min_{\vec{w},b}$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left[\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right) x_j^{(i)}\right] + \frac{\lambda}{m} w_j$$

$j = 1 \ldots n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

}

Looks same as for linear regression!

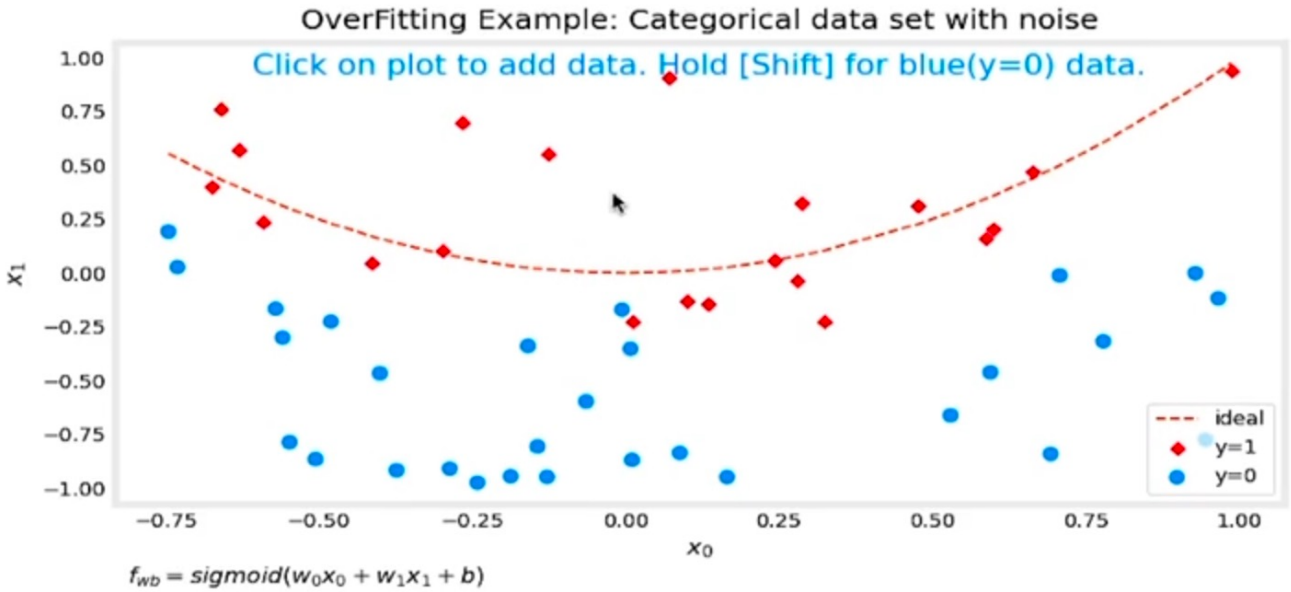
logistic regression

don't have to regularize

jupyter Regularization Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown



Degree

- [x] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6

fit data

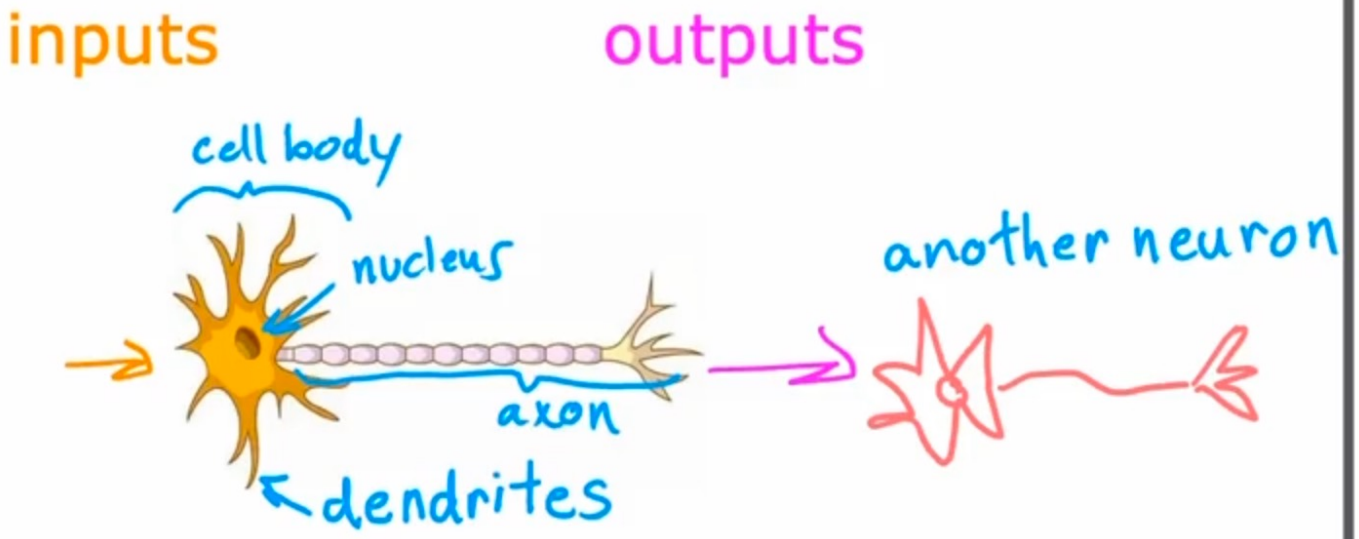
- [ ] Regression
- [x] Categorical

lambda(λ)

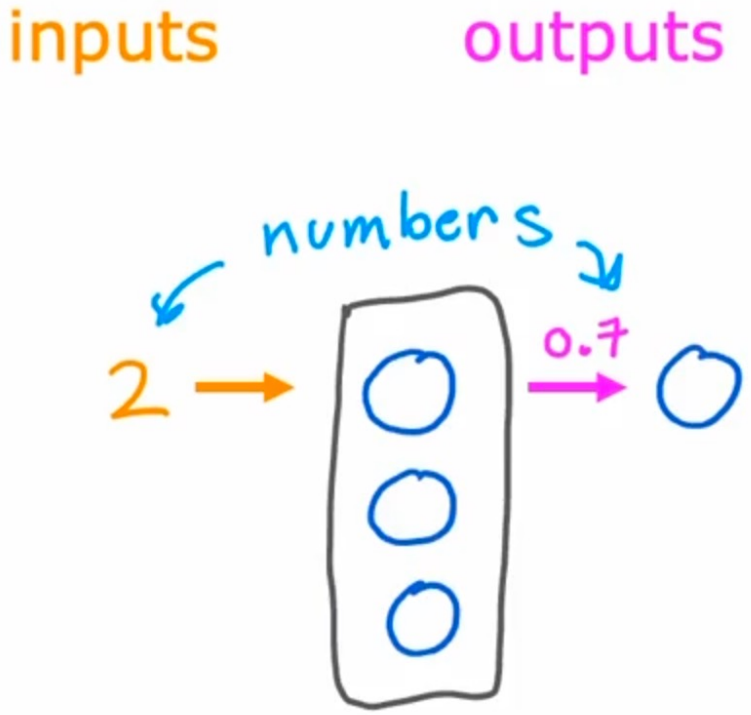
- [x] 0.0
- [ ] 0.2
- [ ] 0.4
- [ ] 0.6
- [ ] 0.8
- [ ] 1

What is Deep Learning?

Biological neuron



Simplified mathematical model of a neuron



Beyond Supervised Learning

- Unsupervised Learning
  - Clustering
  - Anomaly detection
- Recommender Systems
- Reinforcement Learning