Course 1

Machine Learning: "Field of study that gives computers the ability to learn without being explicitly programmed.

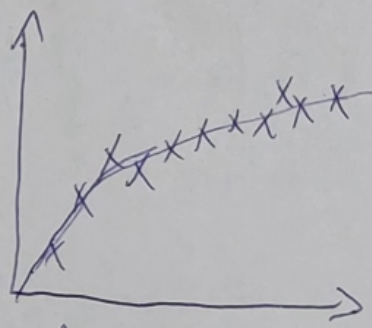
Algorithms:
- Supervised Learning
- Unsupervised Learning
- Recommender systems
- Reinforcement Learning

Supervised Learning:

$X \longrightarrow y$

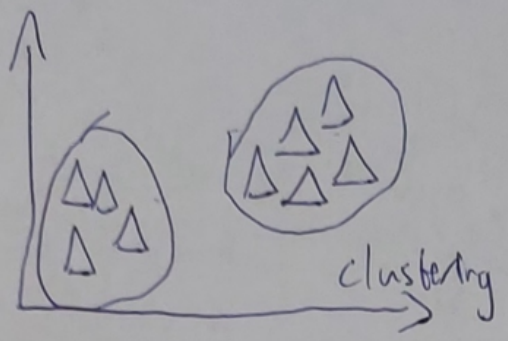
input label

Regression: learns to predict a number out of infinitely many possible numbers



Classification: predicts categories, don't have to be numbers.
Categories are finite/limited

Unsupervised Learning: Find something interesting in unlabeled data



Think Google News

Anomaly Detection: Find unusual data points /outliers

Dimensionality Reduction:

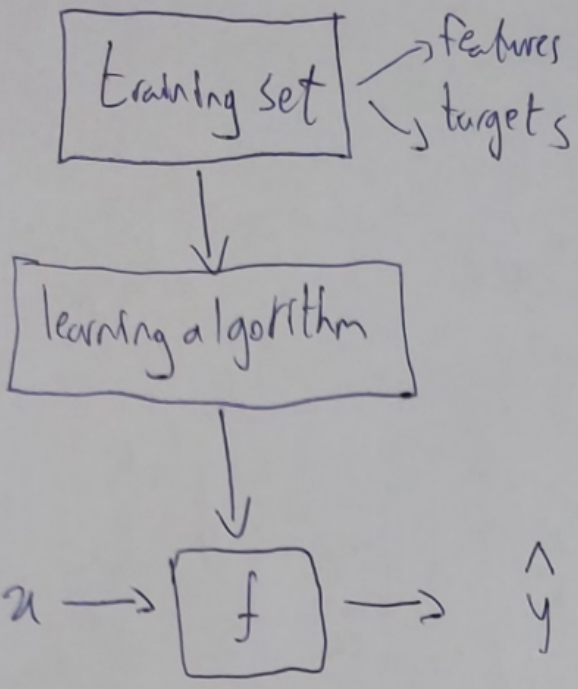
Compress data using fewer numbers

Notation:

x = "input" variable / feature

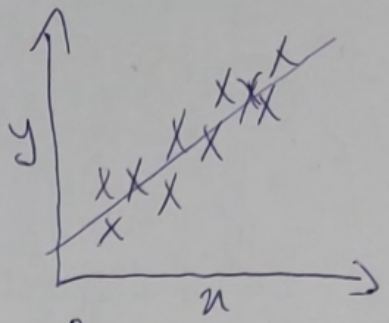
y = "output" variable / target

$(x^{(i)}, y^{(i)})$ = i^{th} training example



feature hypothesis/function model prediction (estimated y)

target



If $f_{w,b}(x) = wx + b$

and 1 variable

=> univariate linear regression

Cost function: used to find the best parameters for a model

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 \quad \text{// Squared error cost function}$$

error

m = number of training examples

Goal is to minimize J and find w and b corresponding to that

Gradient descent: algorithm to reach a minimum by taking the direction of steepest descent multiple times.

Repeat until convergence (simultaneously):

$$\begin{cases} w = w - \alpha \frac{\partial}{\partial w} J(w,b) \\ b = b - \alpha \frac{\partial}{\partial b} J(w,b) \end{cases}$$

α is the learning rate (determines how big our step will be)

α is a small positive number in $(0, 1)$

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

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

Gradient descent algorithm

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

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

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

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

"Batch" gradient descent:

"Batch": Each step of gradient descent uses all the training examples

Multiple features:

x_1 Size	x_2 # Bedrooms	x_3 # floors	x_4 Age	Price
2104	5	1	45	460
$\vec{x}^{(2)}$ 1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

x_j = jth feature

n = number of features

$\vec{x}^{(i)}$ = features of the i^{th} training example

$x_j^{(i)}$ = value of feature j in the i^{th} training example

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

$\vec{w} = [w_1\ w_2\ w_3 \cdots w_n]$ parameters of the model

$\vec{x} = [x_1\ x_2\ x_3 \cdots x_n]$

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

multiple linear regression

To implement this, we use a neat trick called vectorization.

Instead of manually typing the expression or using a for loop, just use `f = np.dot(w, x) + b`

Without vectorization:

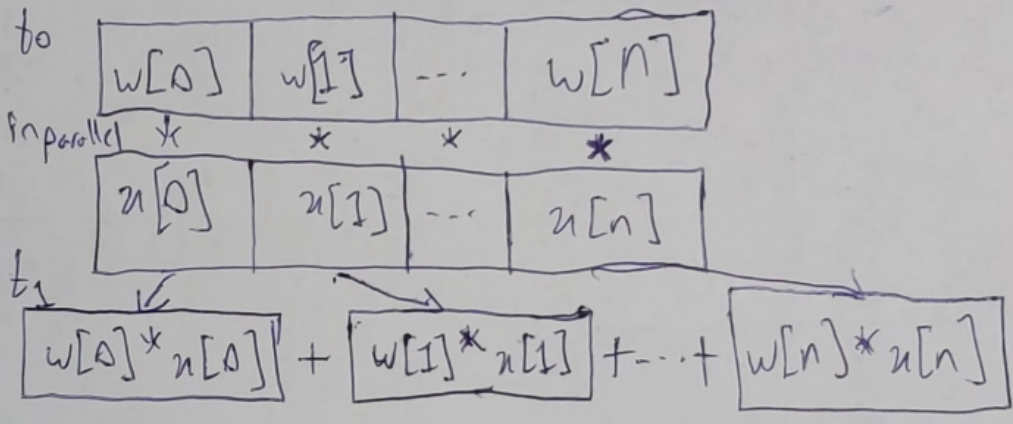
t_0: `f += w[0] * x[0]`

t_1: `f += w[1] * x[1]`

⋮

t_n: `f += w[n] * x[n]`

Vectorization

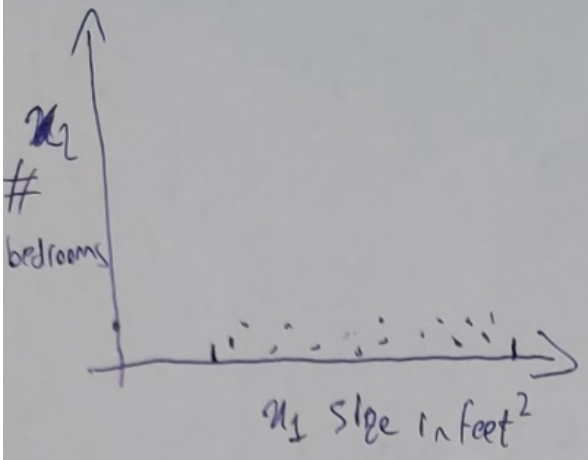


Utilizes computer hardware

Efficient + scalable

Normal equation: Alternative to gradient descent for linear regression only. It is not generalized and can become less efficient for larger inputs.

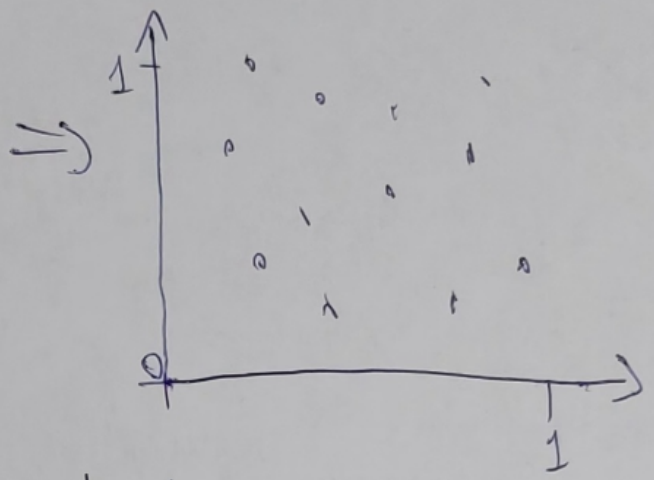
Feature Scaling: Dividing by the max so that scaling ≤ 1



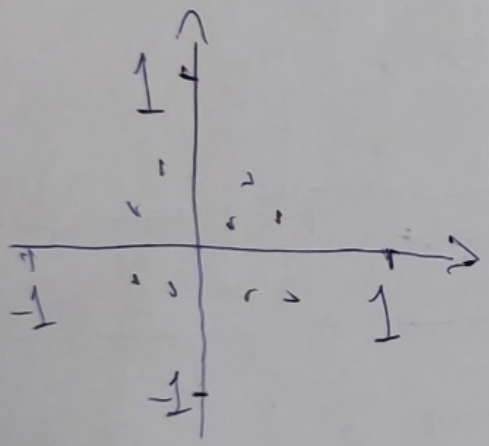
$300 \leq x_1 \leq 2000$ $\quad 0 \leq x_2 \leq 5$

$x_{1,\text{scaled}} = \frac{x_1}{2000}$ $\quad x_{2,\text{scaled}} = \frac{x_2}{5}$

$0.15 \leq x_{1,\text{scaled}} \leq 1$ $\quad 0 \leq x_{2,\text{scaled}} \leq 1$



Mean normalization: rescaling and centering around 0.



then for x: $x_{\text{rescaled}} = \frac{x - \mu}{\max - \min}$

Z-score normalization: $x_{\text{scaled}} = \frac{x - \mu}{\sigma}$

- Aim for about $-1 \leq x_j \leq 1$ for each feature x_j

Why should we use it?

Let $\hat{price} = w_1 x_1 + w_2 x_2 + b$

x_1: ranges: 300 – 2000 (large) — size

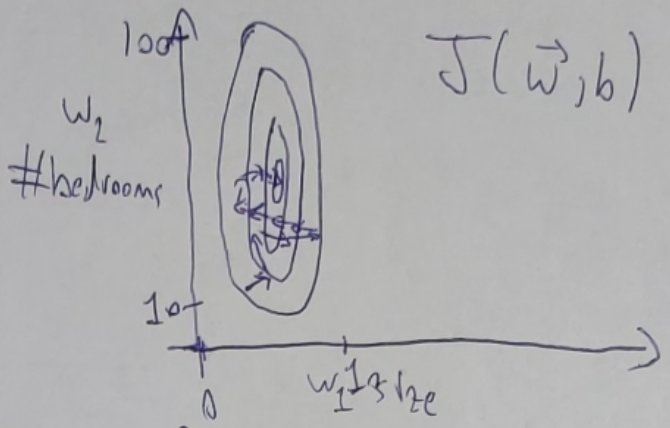
x_2: ranges: 0–5 (small) — #bedrooms

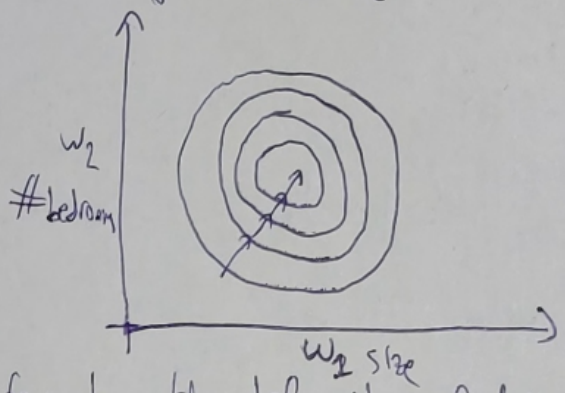
In a contour plot:

The contours form ellipses and are shorter on one side and longer on the other.

This is because a very small change to w_1 can have a very large impact on J. (w_1 is multiplied by a very large number (size of house)).

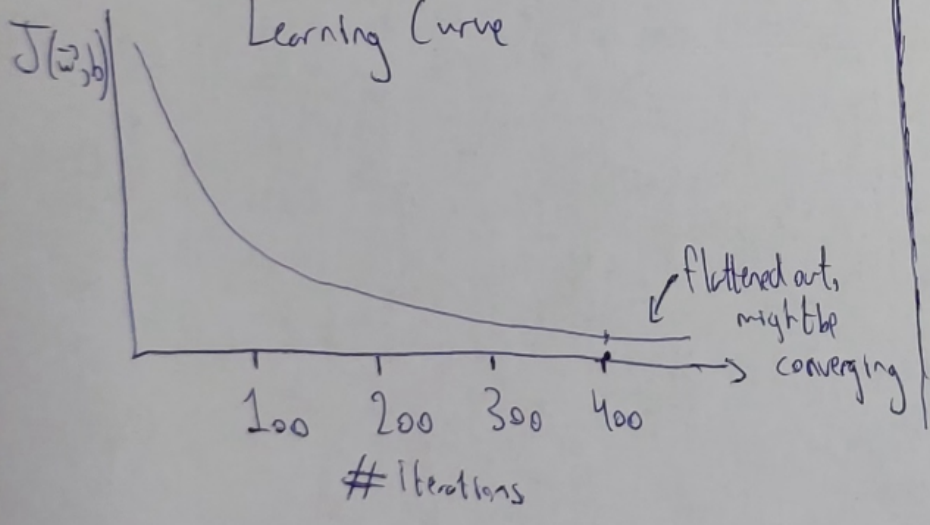
It is the opposite for w_2.





Because the contours are so tall and skinny, gradient descent may bounce back and forth for a long time before it can find the way to the global min

Checking Gradient Descent for convergence:

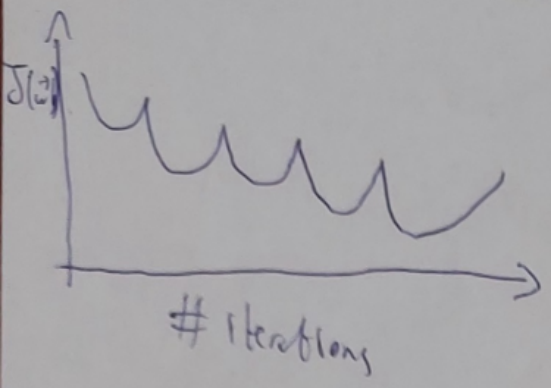


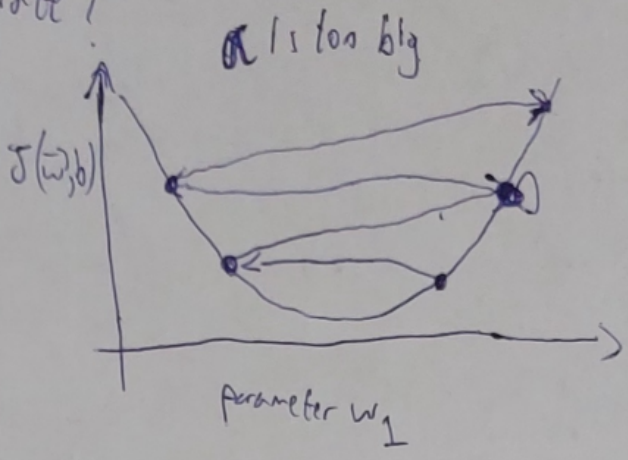
Automatic convergence test

Let $\varepsilon = 10^{-3}$

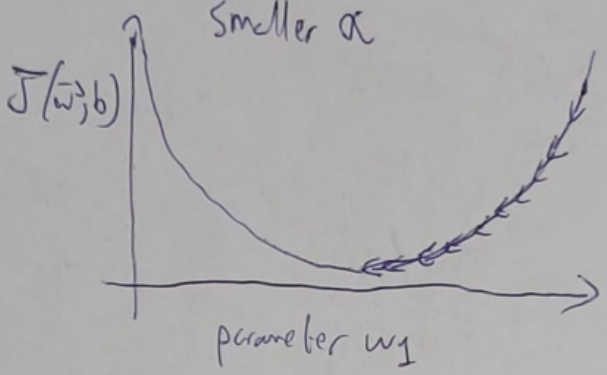
If $J(\vec{w}, b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence

How to choose the learning rate?





overshooting the minimum



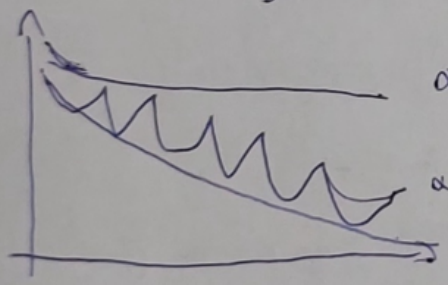
If the learning curve increases, then either α is too big or there is a bug in the code.

Examples $w_1 = w_1 + a d_1$

Values of α to try:

0.001 0.003 0.01 0.03 0.1 0.3 1

≈3x ≈3x 3x ≈3x 3x ≈3x



α too small (1)

α too big (2)

α just right (3) (max possible α < big)

Polynomial Regression:

Just use Linear Regression with $x^2, x^3 \cdots x^n$.

Classification

class = category

binary classification when y can have 2 values for y.

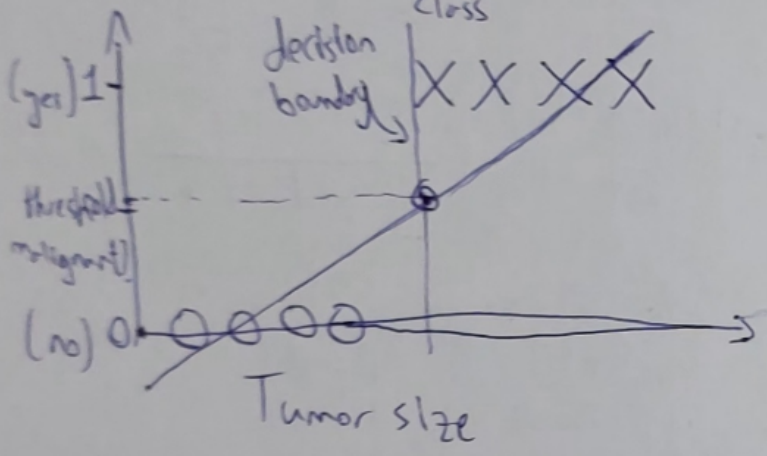
no yes

false true

0 1

negative class

positive class

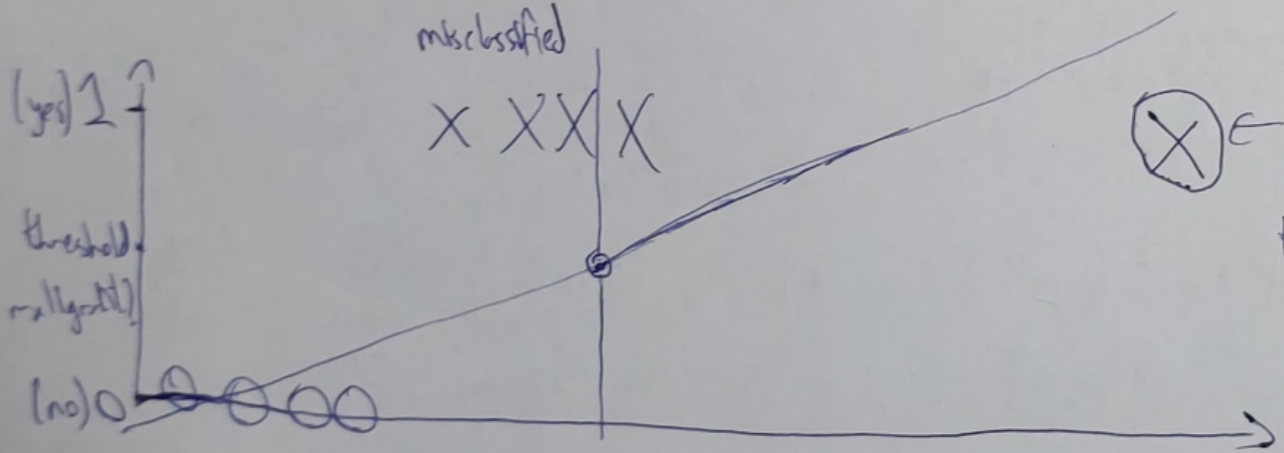


$f_{w,b}(x) = wx+b$ by linear regression.

linear regression outputs numbers greater than 0, less than 1, and even greater than 1. Let's define a threshold (0.5).

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

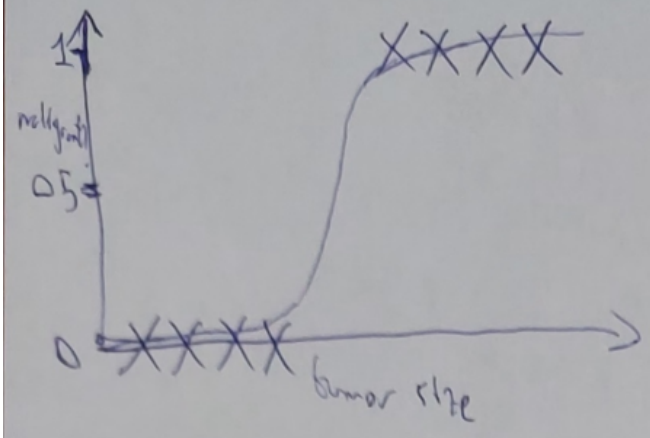
if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$

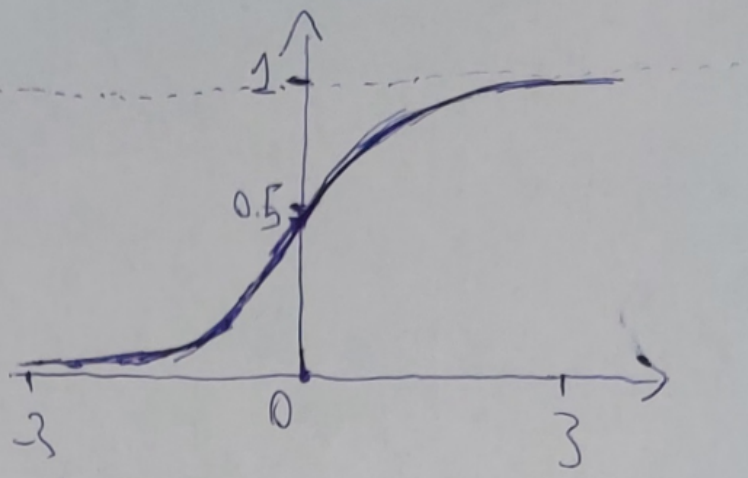


← Another example would change the line of best fit

Therefore linear regression is not reliable

Logistic Regression:





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

$0 < g(z) < 1$

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

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

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

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

"logistic regression"

Interpretation of the output: $f_{\vec{w},b}(\vec{x}) = 0.7$

$\Rightarrow$ 70% chance that y is 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 $\vec{w}, b$

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

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

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

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

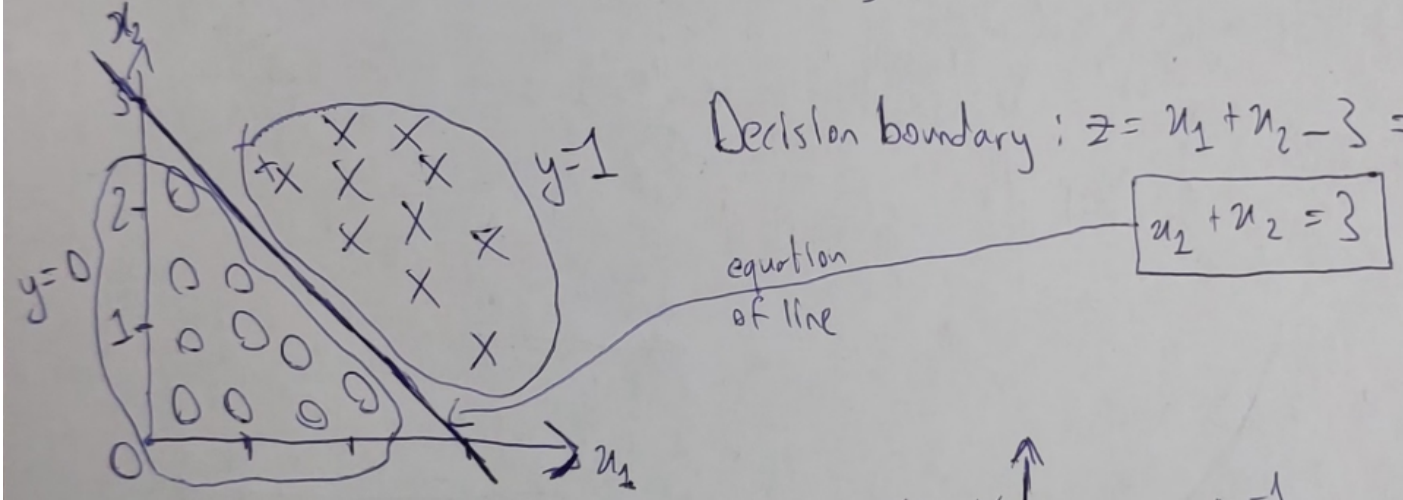
$g(z) \geq 0.5$

$z \geq 0$

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

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

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + b)$$
$$1 \quad 1 \quad -3$$



Decision boundary: $z = x_1 + x_2 - 3 = 0$

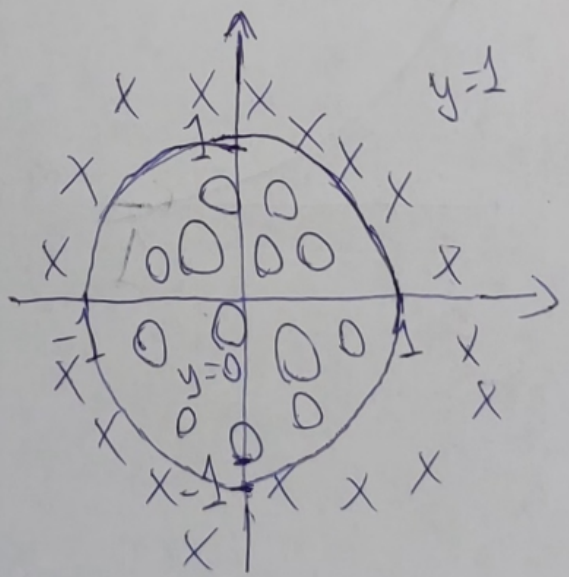
equation of line

$x_1 + x_2 = 3$

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1^2 + w_2x_2^2 + b)$$
$$1 \quad 1 \quad -1$$

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

decision boundary: $z=0 \Rightarrow$ $\boxed{x_1^2 + x_2^2 = 1}$

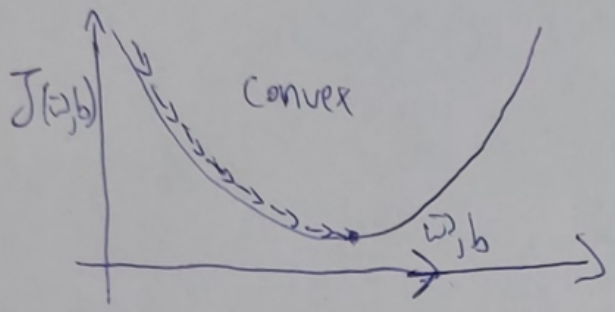


Squared Error cost for linear regression:

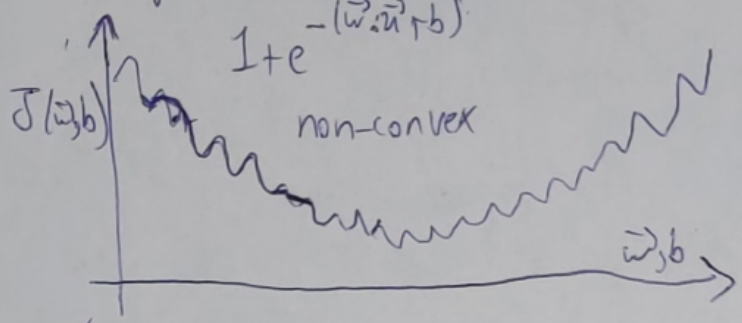
$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \boxed{\frac{1}{2}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}$$

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

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



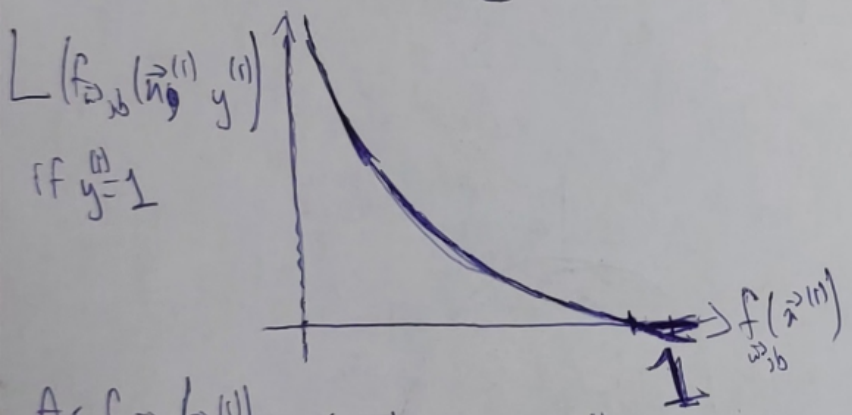
$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

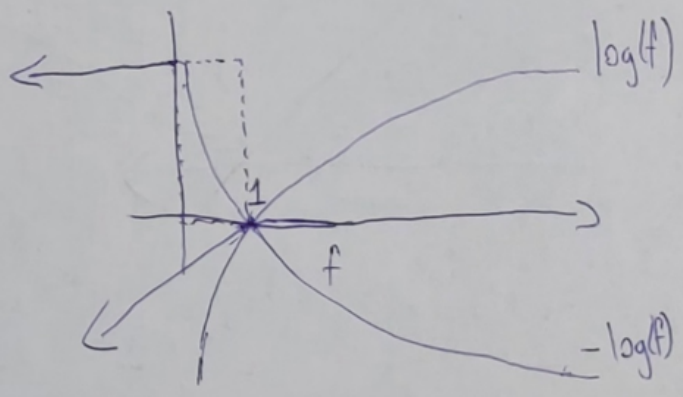


"Wiggly" with many local minima so gradient descent won't work. So squared error cost is not a good choice for logistic regression.

We need a new loss function.

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

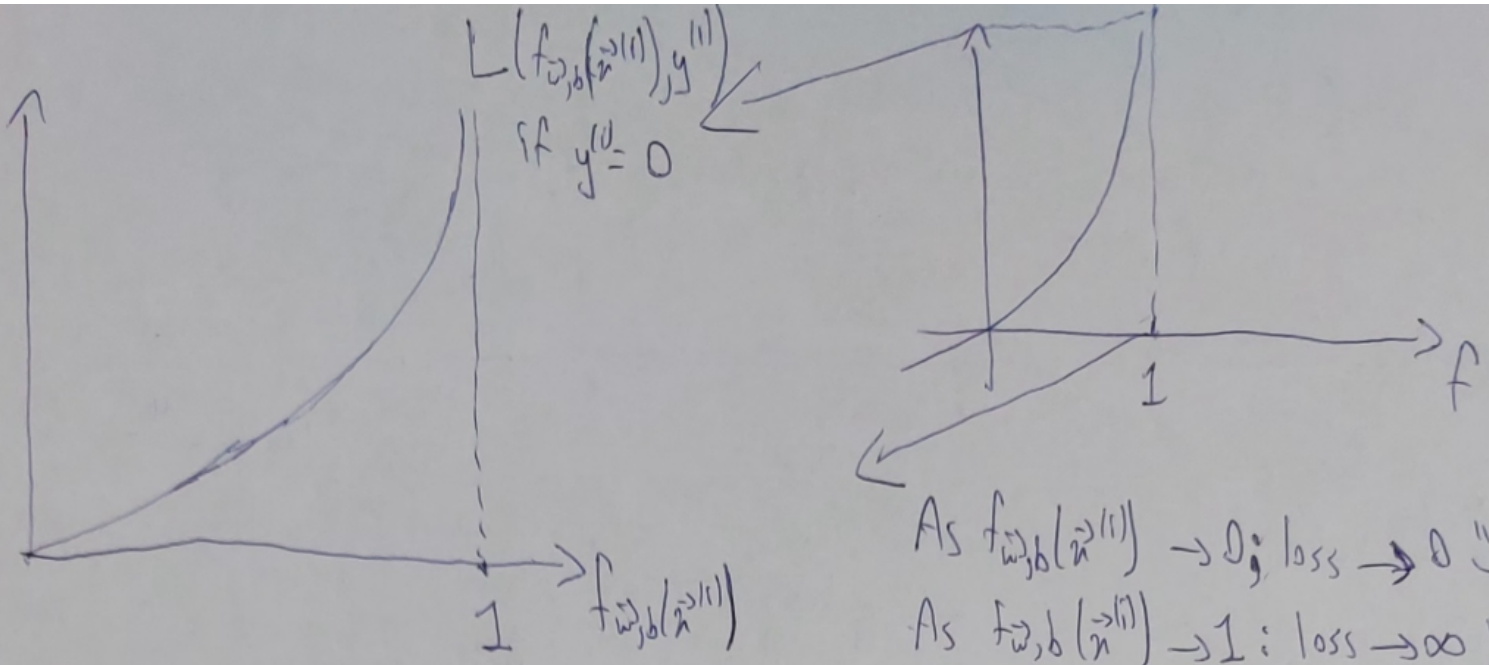




As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$: loss $\to 0$ ☺

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$; loss $\to \infty$ ☹

f = output of logistic regression $\in (0,1)$



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$; loss $\to 0$:)

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

(penalize the model with a very high loss)

Simplified Cost Function for Logistic Regression:

Because y can be either 0 or 1, and cannot take on values in between, we can rewrite the cost function as:

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

Therefore:

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

$$\frac{\partial}{\partial w_j} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$ ↪ j^{th} feature of training example i

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

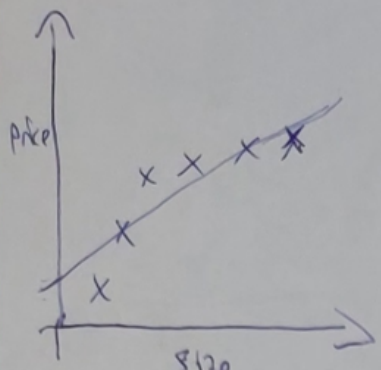
repeat {

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

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

} simultaneous updates

Looks the same as linear regression!



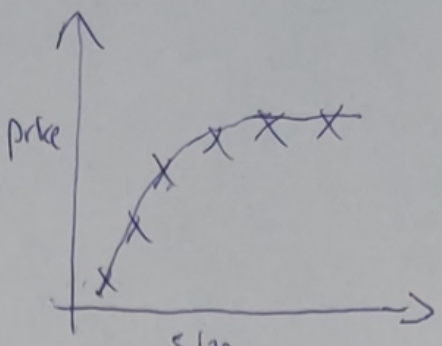
$w_1 x + b$

underfit:

Does not fit the training set well

high bias: it has a preconception that the pattern is linear despite the data showing otherwise

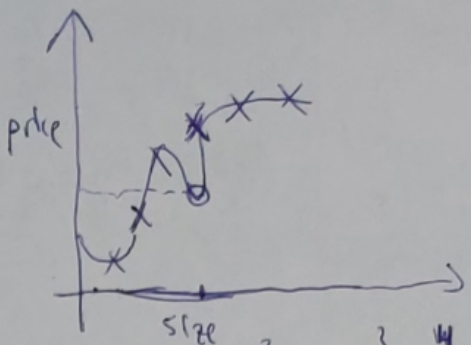
"High bias"



$w_1 x + w_2 x^2 + b$

Fits training set pretty well (not perfectly: cost $\neq 0$)

generalizations making the model do well even on examples that are not on the training set. (make predictions on examples it has never seen before.



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

Overfit

Fits the training set extremely well (cost = 0)

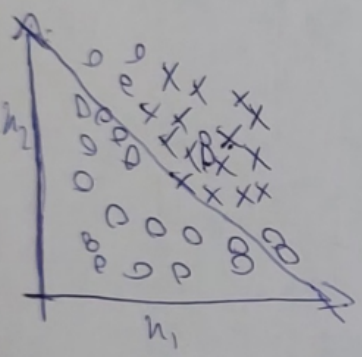
But it's wiggly and might predict data incorrectly such as a house of a larger size being cheaper than one with a smaller size.

It looks like the model has fit the training data too well and can't generalize to new examples

"High variance"

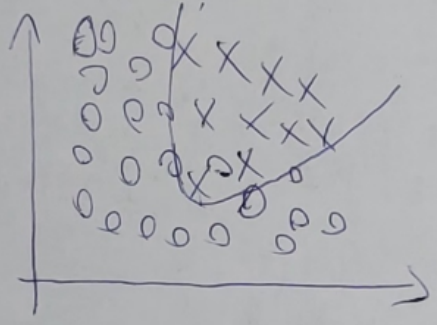
Two models with slightly different training sets might end up with totally different / highly variable predictions

Applies to classification as well



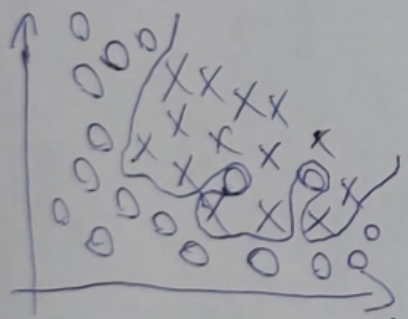
$z = w_1 x_1 + w_2 x_2 + b$

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



Overfitting

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

Addressing Overfitting:

Options:

1. Collect more data
2. Select Features
   - Feature selection
3. Reduce size of parameters
   - Regularization

Regularization: penalize for high/large parameters in the cost function:

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

← can exclude

Regularization term: keep w_j small

λ: regularization parameter

Fit data → ← λ balances both goals

If $\lambda = 0$: overfits

If $\lambda = 10^{10}$ (large number): underfits because model focuses too much on making $w_j \approx 0$ so $f(x) \approx b$

So we'll have a horizontal line.

Gradient descent:

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b) \quad j=1\text{--}n$$

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

} simultaneous update

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

$$\frac{\partial}{\partial b} J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) \quad \text{(because we don't regularize } b\text{)}$$

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

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

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

$0.01 \cdot \frac{1}{50} = 0.0002$ then $w_j \left(1 - \alpha \frac{\lambda}{m} \right) = 0.9998 \, w_j$

This is what regularization does. This is how it works: by multiplying w_j by a number slightly < 1, effectively shrinking it every iteration.

End of Course One