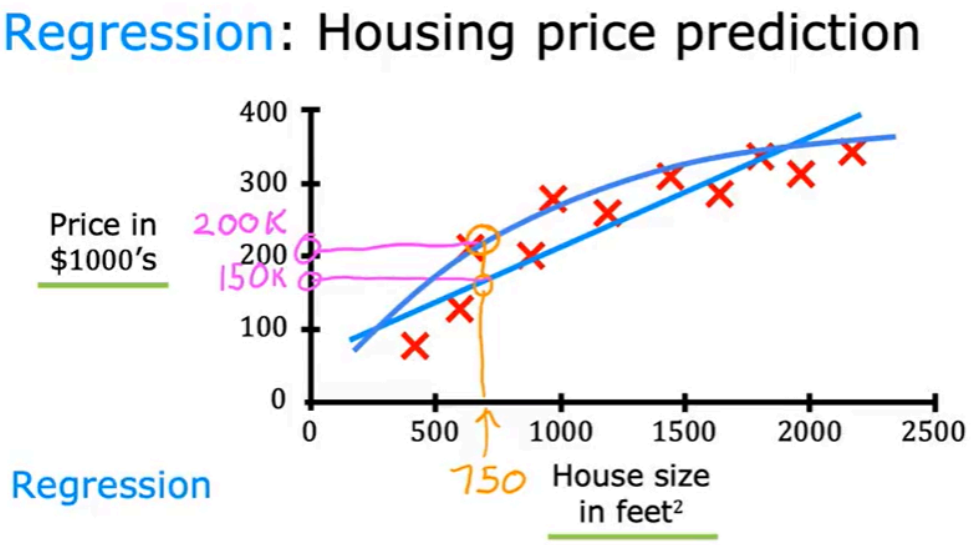
Supervised Machine learning : [Regression and Classification] :

There are various machine learning algorithm but the main 2 algorithm here are :

1) Supervised learning algorithm : A type of machine learning where the model learns from labeled data (input-output pairs) to **predict outcomes** for new data.

2) Unsupervised learning algorithm : A type of machine learning where the model learns patterns and structures from **unlabeled data** without any predefined output.

Example of supervised learning:



Supervised learning can be used for 2 types :

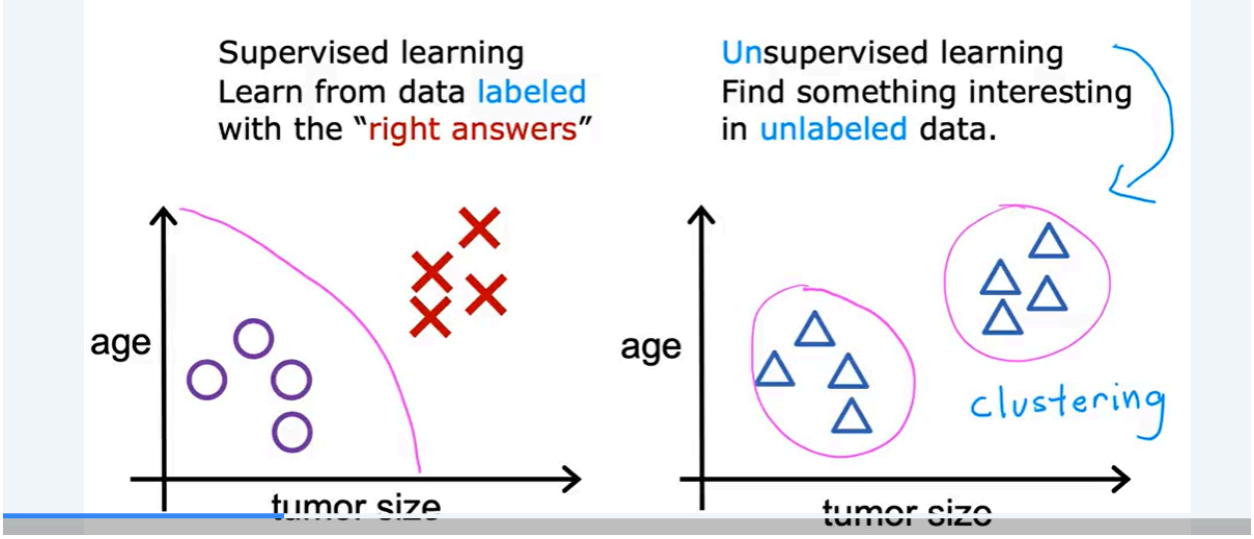
Supervised learning

Learns from being given "right answers"

Regression
Predict a number
infinitely many possible outputs

Classification
predict categories
small number of possible outputs

Supervised and Unsupervised difference :



Example of unsupervised learning algorithm:

Clustering: Grouping customers

Unsupervised learning

Data only comes with inputs x, but not output labels y.
Algorithm has to find structure in the data.

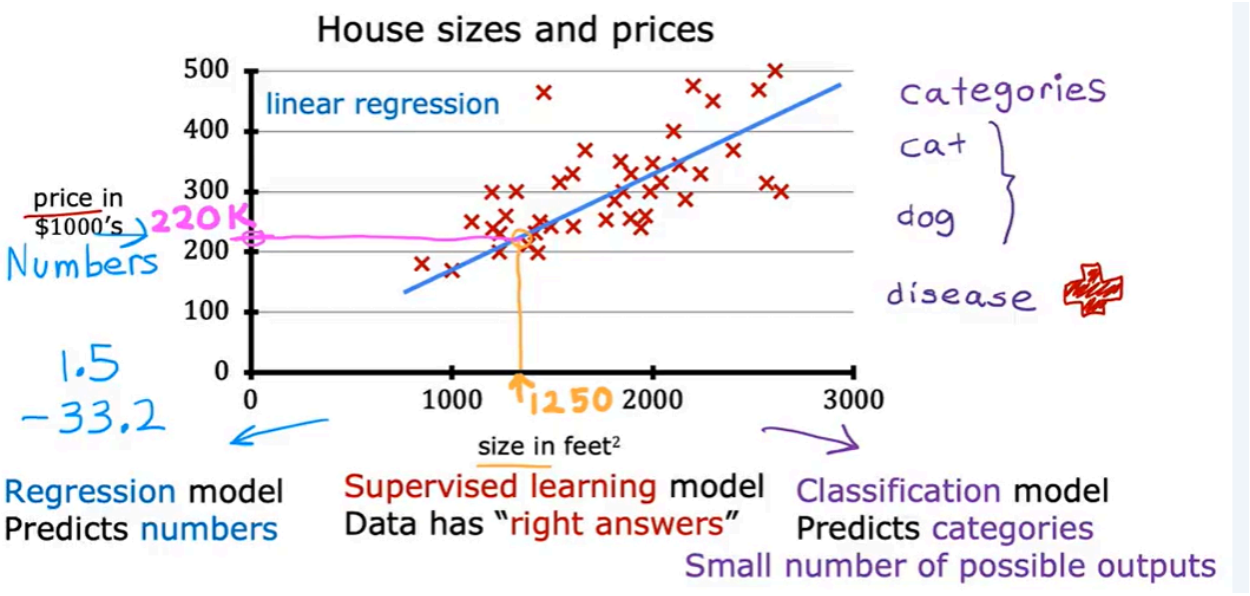
Clustering
Group similar data points together.

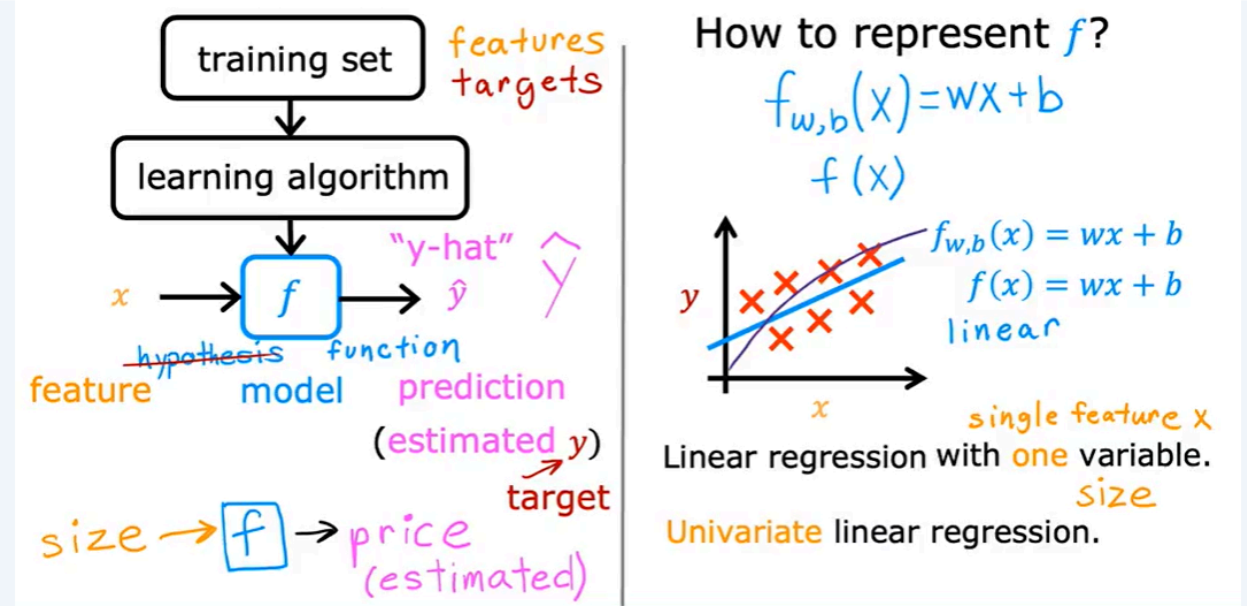
Dimensionality reduction
Compress data using fewer numbers.

Anomaly detection
Find unusual data points.

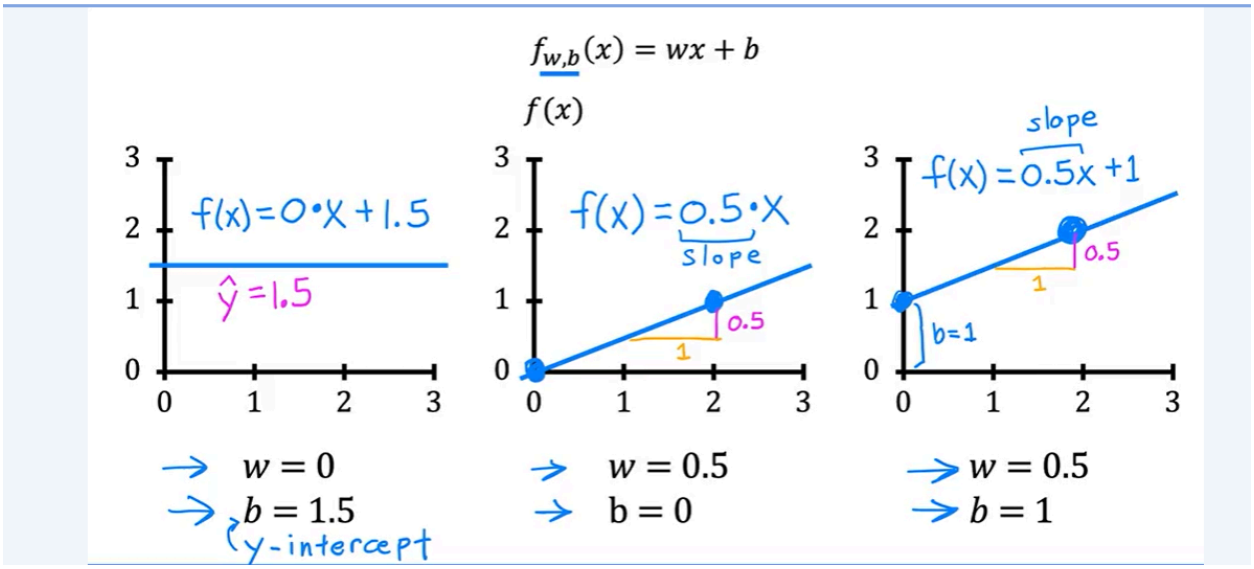
Supervised learning:

1) Regression:

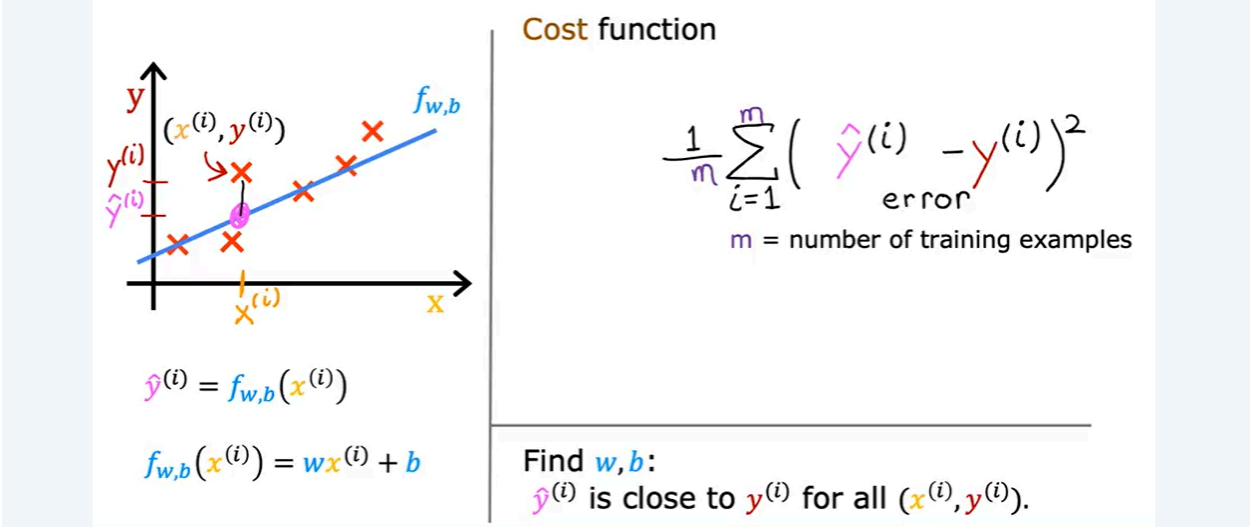




Fitting regression line on various condition :



Here the cost function is used to observe how well our data fits the model.



In machine learning people uses different cost function for different purposes but squared error cost function is most common and mostly used for regression task

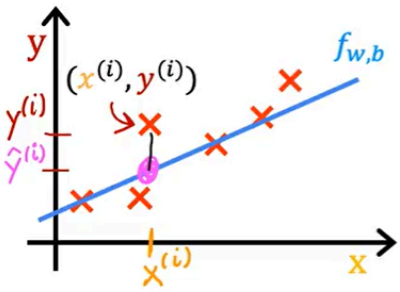
Cost function: Squared error cost function

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

$\hat{y}^{(i)}-y^{(i)}$: error

m = number of training examples

By convention the cost function that people uses is divided by 2*m , the extra division of 2 just meant to make some of our later calculation clear .



$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$ ←

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

Cost function: Squared error cost function

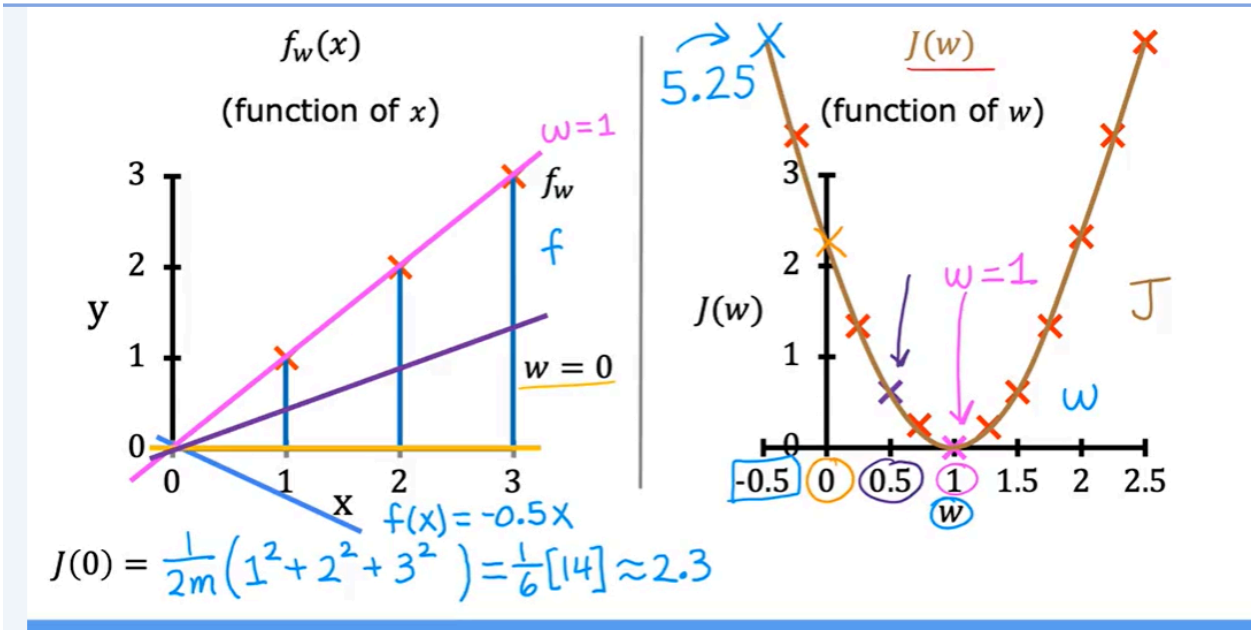
$$J(w,b)=\frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)}-y^{(i)}\right)^2$$

$\hat{y}^{(i)}-y^{(i)}$: error

m = number of training examples

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

Find w, b:
$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$.



Each value of parameter w corresponds to a different straight line fit. f(x) on the graph to the left. And for given training set that choice for w corresponds to a single point o right Because for each point of w the cost can be calculated as J(w) .

❓⭐**How can the proper value of w be chosen so that the fitting of the line is well?**

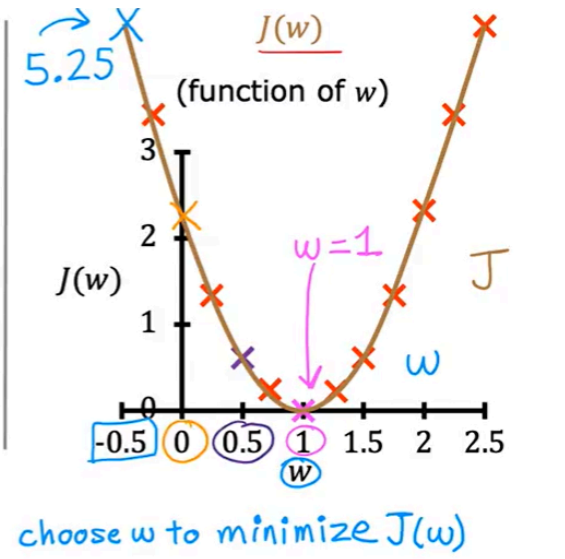
Ans: Choosing a value of w that minimizes the squared error as small as possible gives a good model.

goal of linear regression:

$$\underset{w}{\text{minimize}}\, J(w)$$

general case:

$$\underset{w,b}{\text{minimize}}\, J(w,b)$$



Model	$f_{w,b}(x) = wx + b$
Parameters	w, b
Cost Function	$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$
Objective	$\underset{w,b}{\text{minimize}}\, J(w,b)$

Gradient Descent:

Gradient Descent is a method to find the best solution (like the lowest error in a machine learning model) by taking small steps in the direction that reduces the error the most.

Why is it used in Machine Learning?

In ML, we often have a **loss function** (how wrong the model is) and we want to minimi -ze it. Gradient Descent helps us adjust the model's parameters (like weights) to reduce the loss.

How It Works (Steps):

- **Start with random parameters** (weights).
- **Compute the gradient** (slope) of the loss function.
- **Update the parameters** in the direction that reduces the loss.
- **Repeat** until the loss is minimized.

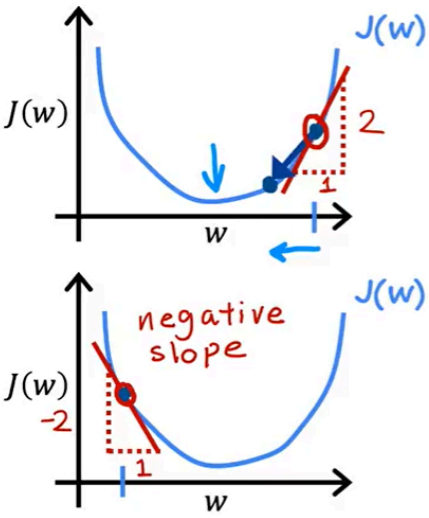
⚙️ Formula:

$$\theta := \theta - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta}$$

Where:

- θ = parameter (weight)
- α = learning rate (step size)
- $J(\theta)$ = loss/cost function
- $\frac{\partial J}{\partial \theta}$ = gradient (slope of the loss)

Here , alpha is the learning rate which determines how big step it will take to update model's parameter.



$w = w - \alpha \frac{d}{dw} J(w)$

> 0

$w = w - \alpha \cdot (positive\ number)$

$\frac{d}{dw} J(w)$ < 0

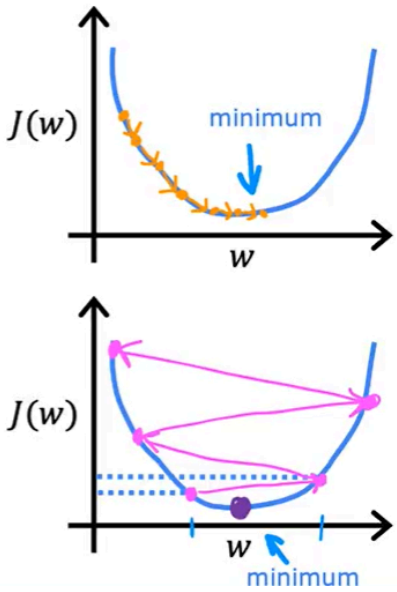
$w = w - \alpha \cdot (negative\ number)$

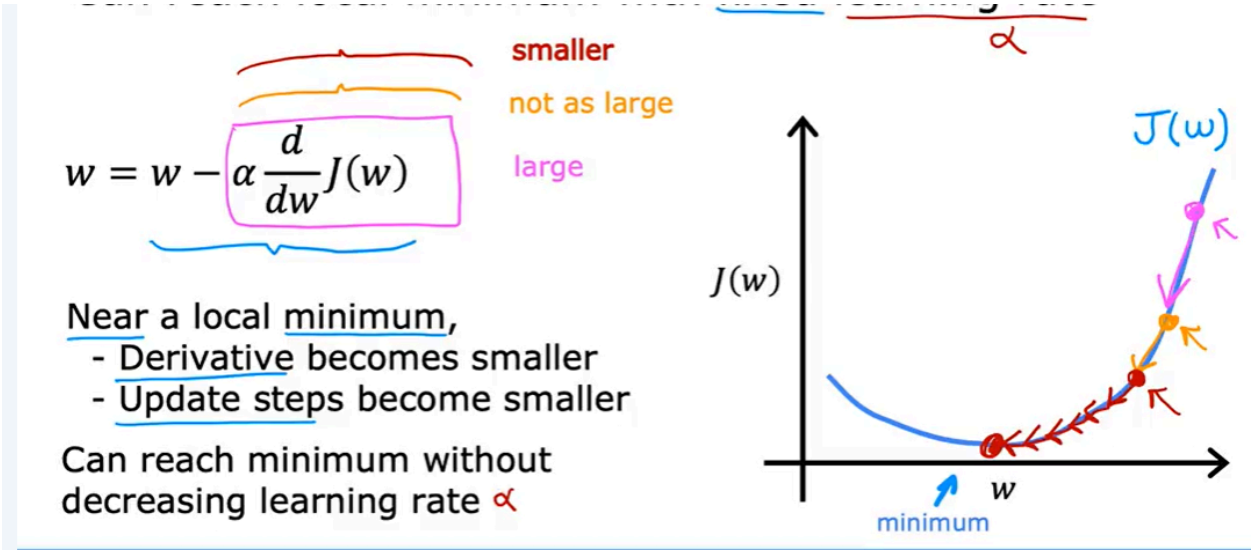
$$w = w - \alpha \frac{d}{dw} J(w)$$

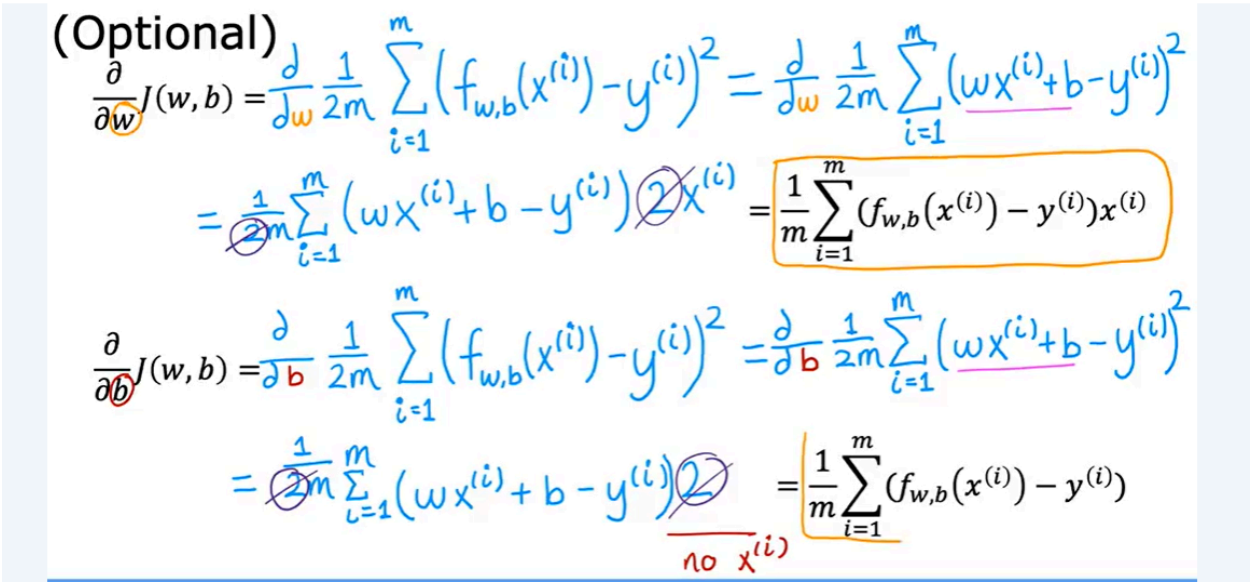
If α is too small...
Gradient descent may be slow.

If α is too large...

Gradient descent may:
- Overshoot, never reach minimum
- Fail to converge, diverge







Differentiate with respect to w and b and after differentiation the result of gradients are found .

📦 Types of Gradient Descent:

Type	Description
Batch Gradient Descent	Uses the entire dataset to compute the gradient.
Stochastic Gradient Descent (SGD)	Uses one training sample at a time.
Mini-Batch Gradient Descent	Uses a small batch of data for each update (most common).

Multiple Linear Regression:

📘 What is Multiple Linear Regression?

Multiple Linear Regression (MLR) is a statistical and machine learning technique used to **predict the value of a dependent variable** using **two or more independent variables**.

It extends **simple linear regression** (which uses one input) by allowing **multiple features** to influence the output.

🧮 Mathematical Formula:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Where:

- y: predicted (dependent) variable
- $x_1, x_2, \ldots, x_n$: independent variables (features)
- β_0: intercept
- $\beta_1, \beta_2, \ldots, \beta_n$: coefficients (weights)
- ϵ: error term

✅ In Simple Terms:

> **Multiple Linear Regression finds the best-fitting line or plane that predicts an outcome (like house price) based on multiple features (like size, number of rooms, and location).**

📌 Real-Life Example:

Predicting house price using:

- x_1 = square footage
- x_2 = number of bedrooms
- x_3 = distance to city center

Then MLR finds a relationship like:

$$\text{Price} = \beta_0 + \beta_1(\text{size}) + \beta_2(\text{bedrooms}) + \beta_3(\text{distance})$$

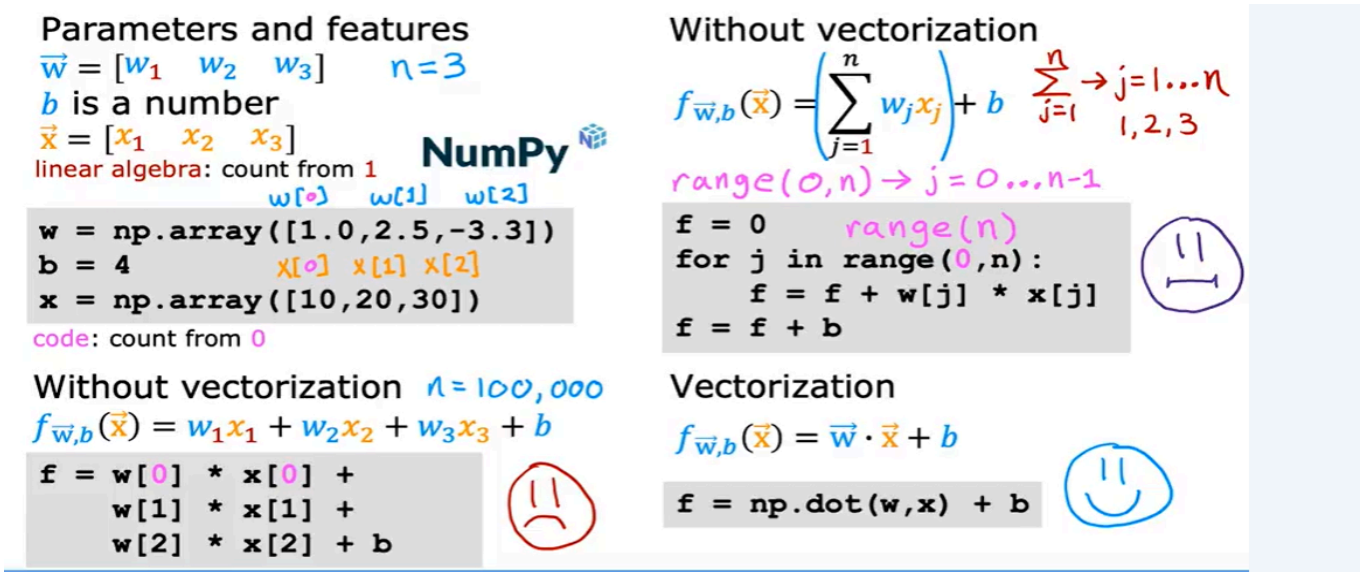
Vectorization:

Vectorization is the process of converting operations (especially mathematical computations) into **vector or matrix form** so that they can be executed efficiently using **optimized low-level implementations** (like NumPy or BLAS).

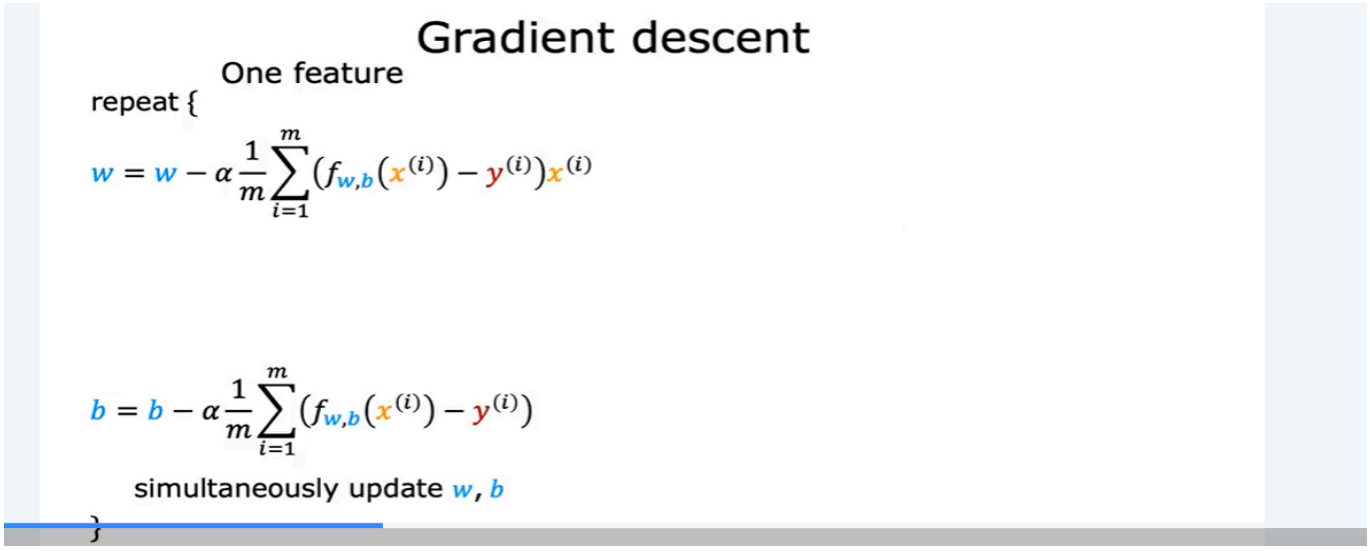
In simpler words , Vectorization is about **removing loops** and using **arrays/matrices** to perform faster, cleaner, and more efficient computations.

⚡ Why Use Vectorization?

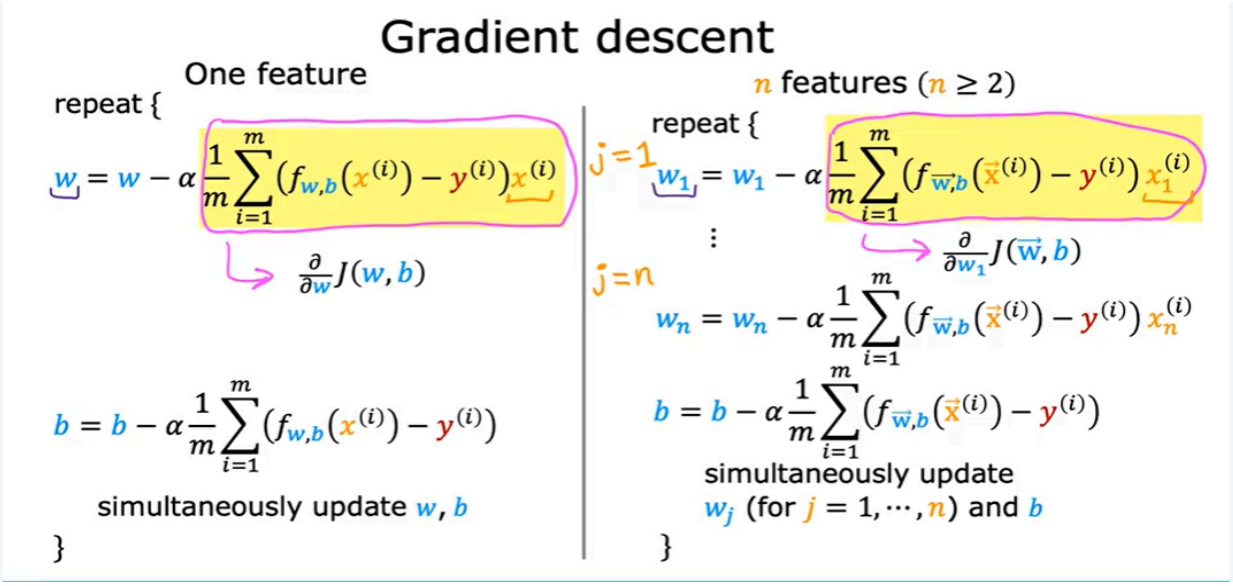
- ✅ **Faster execution** (especially on large datasets)
- ✅ **Cleaner code**
- ✅ **Better use of CPU/GPU parallelism**



Gradient descent for one feature is simple. The equation is given below:



When gradient descent for multiple variables is needed then with respect to each feature variable the differentiation is done and the weight of each feature gets updated.



Feature Scaling:

Feature Scaling is a preprocessing technique used in machine learning to **normalize the range of independent variables (features)**. It ensures that **no single feature dominates** others simply because of its scale.

Many algorithms (like Gradient Descent, KNN, SVM, PCA) are sensitive to the **magnitude of features**. For example:

- Age (in years): `25, 30, 35`
- `Income (in dollars): 40,000, 60,000, 80,000`

Without scaling, **Income** dominates due to larger values.

With feature scaling , anyone often get gradient descent to run mush more faster.

✅ 1. Min-Max Scaling (Normalization)

Purpose: Scales features to a fixed range, usually **[0, 1]**.

Formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Where:

- x = original value
- $x_{\min}$ = minimum value in the feature
- $x_{\max}$ = maximum value in the feature
- x' = scaled value

✅ 2. Standardization (Z-score Normalization)

Purpose: Rescales data so that it has **mean = 0** and **standard deviation = 1**.

Formula:

$$x' = \frac{x - \mu}{\sigma}$$

Where:

- μ = mean of the feature
- σ = standard deviation
- x' = standardized value

✅ 3. MaxAbs Scaling

Purpose: Scales values to the range **[-1, 1]** by dividing by the **maximum absolute value**.

Formula:

$$x' = \frac{x}{|x_{\max}|}$$

Where:

- $x_{\max}$ = maximum absolute value in the feature
- x' = scaled value

✅ 4. Robust Scaling

Purpose: Scales data using the **median** and **interquartile range (IQR)** — robust to outliers.

Formula:

$$x' = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

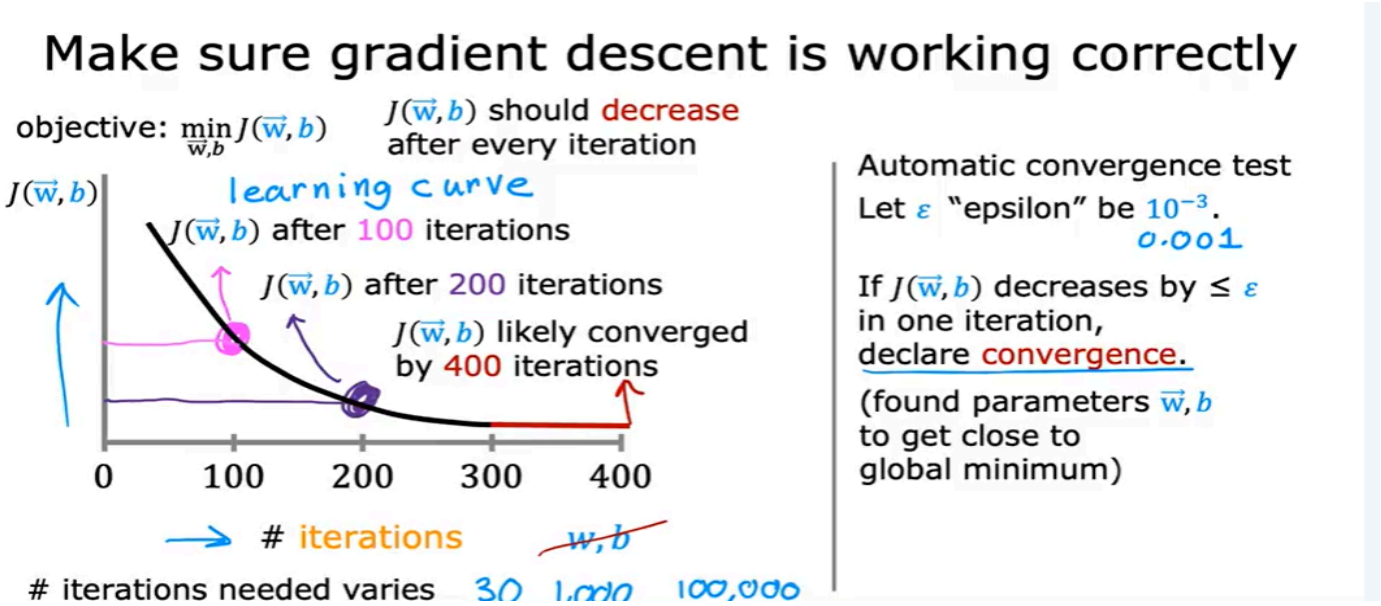
Where:

- $\text{IQR} = Q3 - Q1$
- x' = scaled value

✅ Summary Table

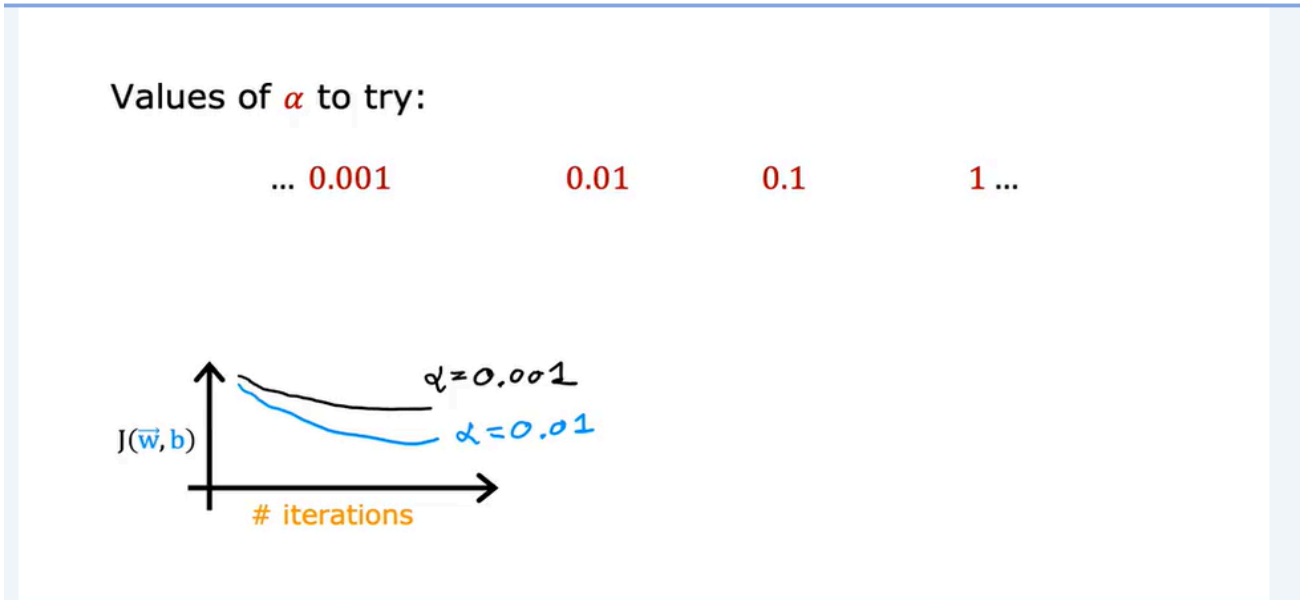
Technique	Centered at 0	Handles Outliers	Range	Common Use Case
Min-Max Scaling	❌	❌	[0, 1]	Algorithms like KNN, neural networks
Standardization	✅	❌	Mean = 0, SD = 1	Most ML algorithms
MaxAbs Scaling	✅	❌	[-1, 1]	Sparse data, when negatives matter
Robust Scaling	✅	✅	Variable	Data with outliers

Checking Gradient Descent For Convergence:



If the cost function gradually decreases with respect to iterations then this is a proof that gradient descent is working and after a fixed iteration when a flat line is derived that means that it converges.

Choosing Learning Rate:



If learning rate is very big then the global minima of cost function can not be found and if it is very low then the model will learn very slowly and it will take much more time which is inefficient

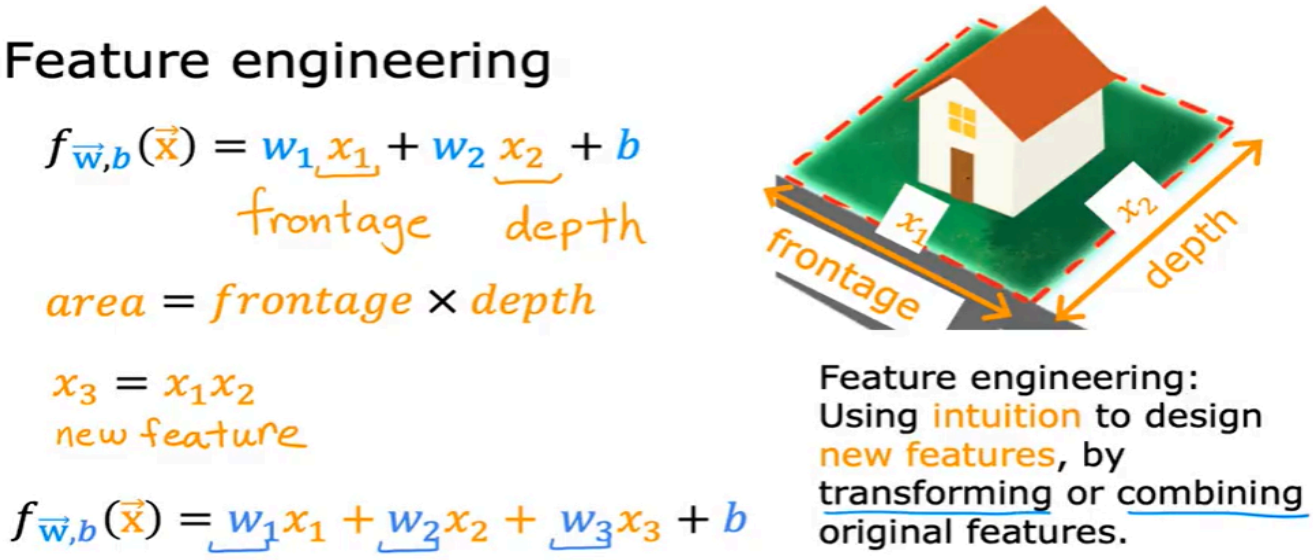
With a small enough alpha , J(w,b) should decrease on every iteration.

Feature Engineering :

Feature Engineering is the process of **creating, transforming, or selecting the most relevant features** (input variables) from raw data to improve the performance of a machine learning model.

Why is Feature Engineering Important?

- Models are only as good as the **input features**.
- Better features → Better predictions (even with simple models).
- Helps the model **learn patterns** more easily.



Feature Engineering = Smart transformation of raw data into meaningful inputs that boost model performance.

Classification:

Classification is a type of **supervised learning** where the goal is to predict **discrete labels** (categories) for input data.

🔁 Classification vs Regression:

Task	Output Type	Example
Classification	Category	Predict if email is spam
Regression	Continuous	Predict house price in $

🧠 Common Algorithms for Classification:

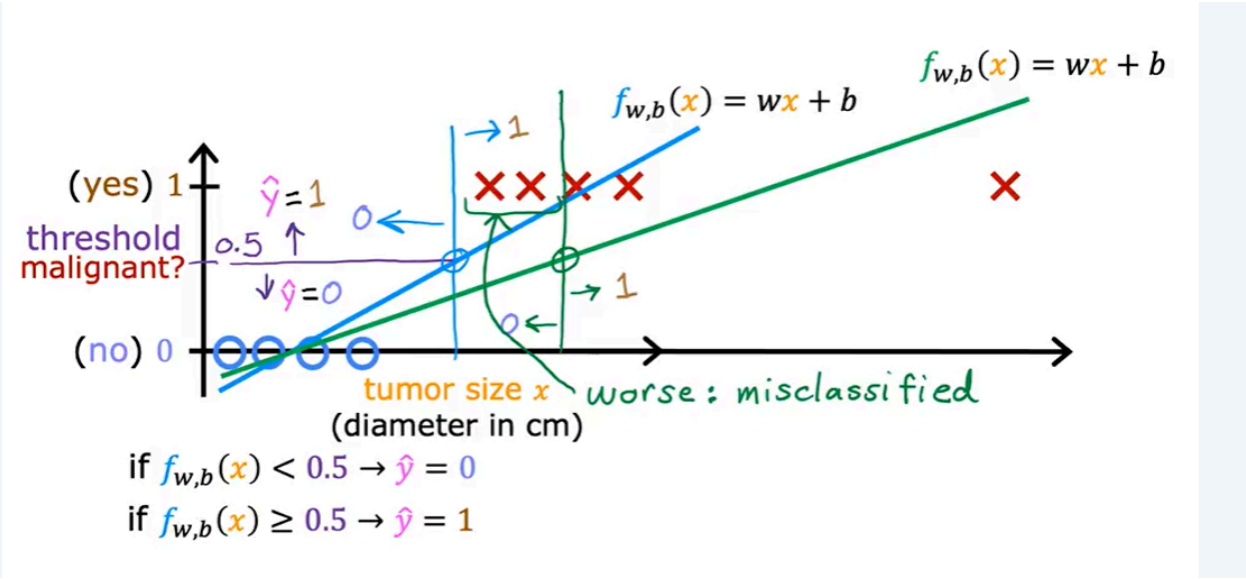
- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Naive Bayes
- K-Nearest Neighbors (KNN)
- Neural Networks

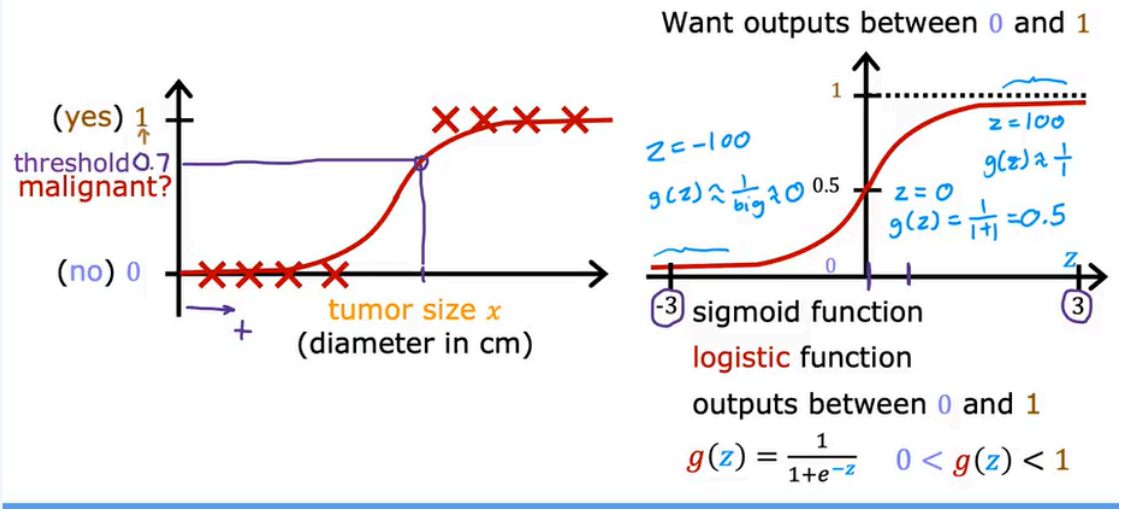
Evaluation metrics:

- **Accuracy** = (Correct predictions) / (Total predictions)
- **Precision, Recall, F1-score**
- **Confusion Matrix**
- **ROC AUC Score**

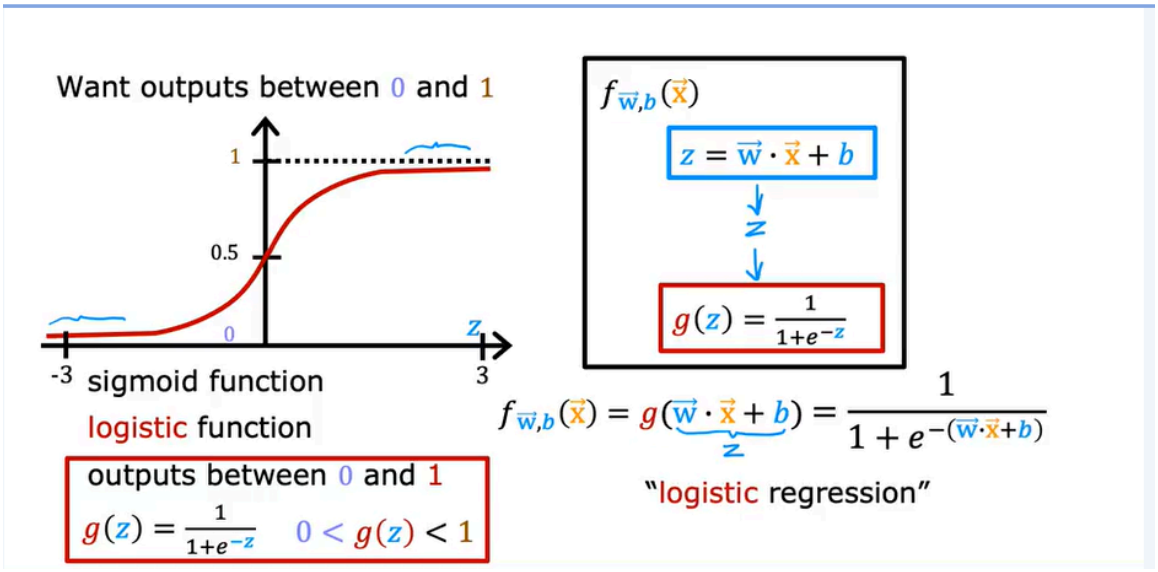
Logistic Regression:

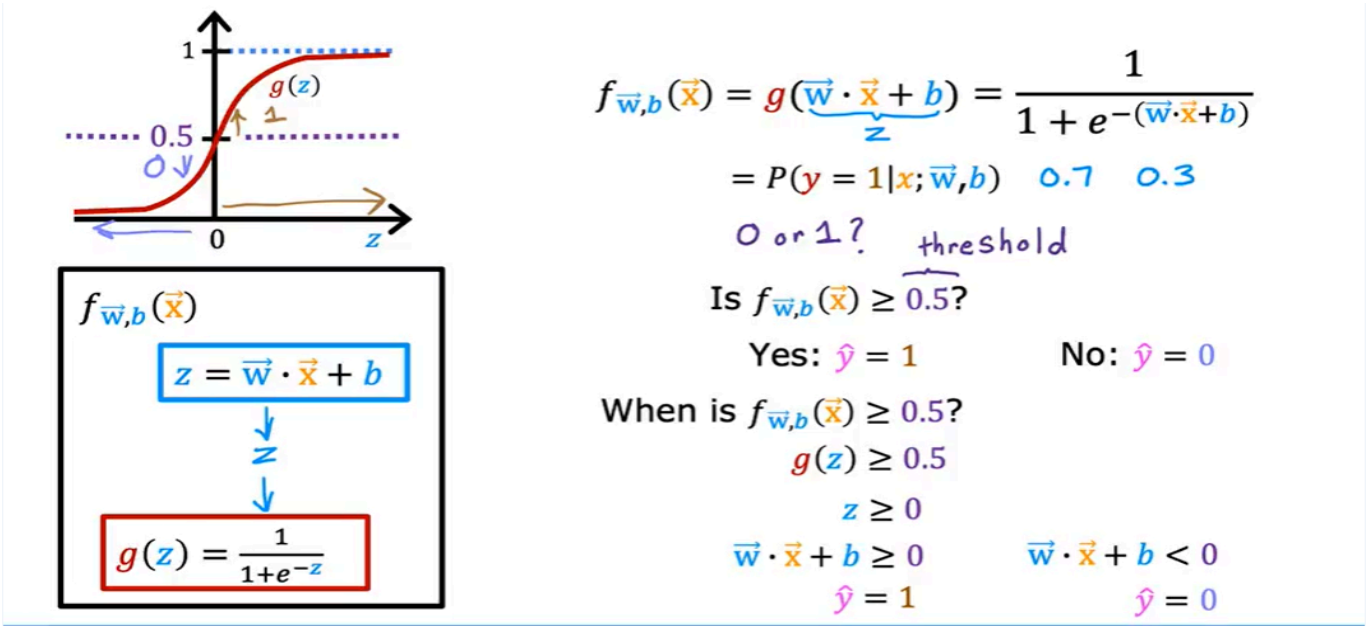
Logistic Regression is a supervised machine learning algorithm used for **binary** (and sometimes **multi-class**) classification tasks. Unlike linear regression which predicts a continuous output, **logistic regression predicts the probability of a class label (0 or 1)**.





At the heart of logistic regression is the **sigmoid (logistic) function**. This function squashes any real-valued number into the range (0, 1), making it suitable to interpret as a **probability**.





When wx+b >=0 the model predicts 1 and when wx+b<0 the model predicts 0.

Cost Function For Logistic Regression:

The cost function tells us **how well our logistic regression model is performing**. We want to **minimize** this cost during training using optimization (like gradient descent).

Cost function for logistic regression:

Cost

cost

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

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

convex
↳ can reach a global minimum

find w, b that minimize cost J

Gradient Descent for Logistic Regression:

Gradient descent for logistic regression

repeat {

looks like linear regression!

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

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

} simultaneous updates

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

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

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

Gradient descent

cost

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

repeat {

j = 1 ... n

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

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

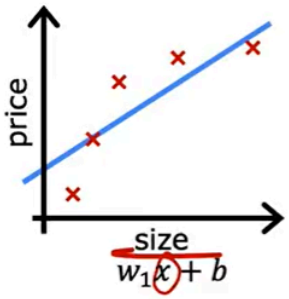
$$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w}, b)$$

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

} simultaneous updates

Problem of Overfitting:

Regression example

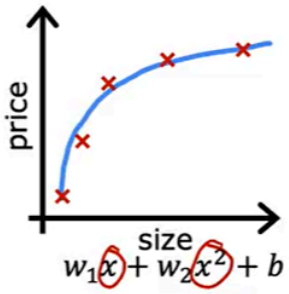


$w_1x + b$

underfit

- Does not fit the training set well

high bias

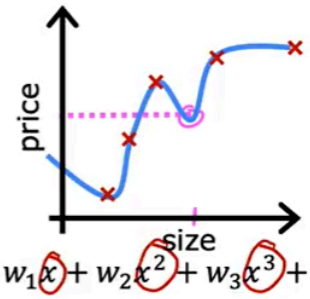


$w_1x + w_2x^2 + b$

just right

- Fits training set pretty well

generalization



$w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

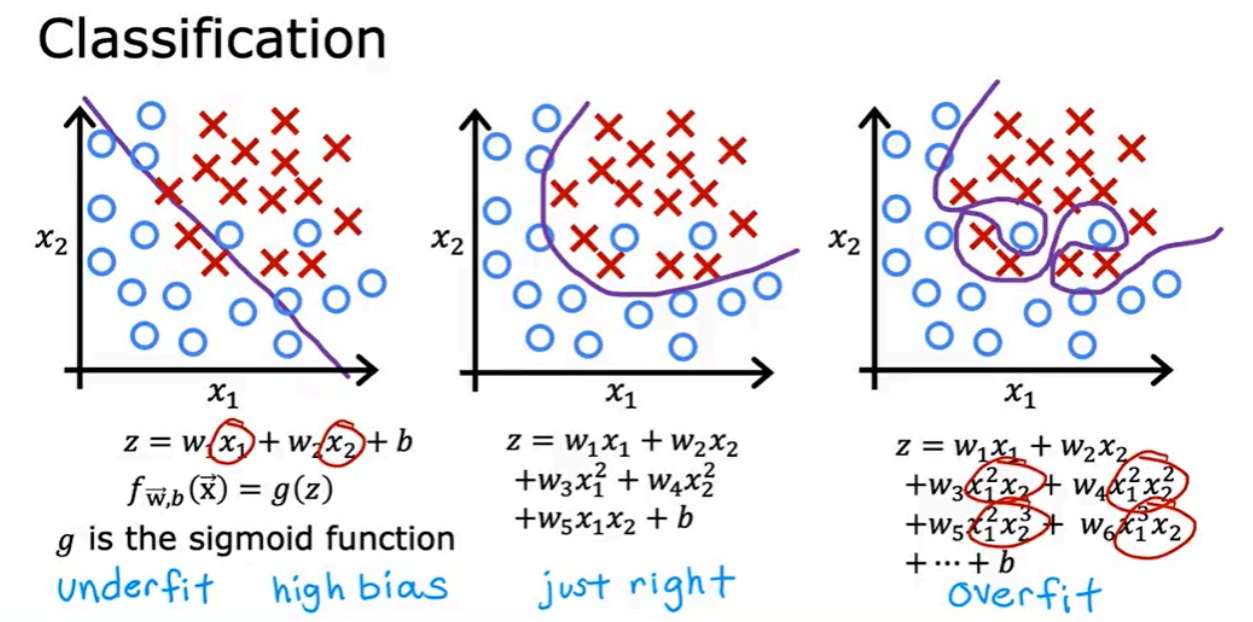
overfit

- Fits the training set extremely well

high variance

Overfitting : **Overfitting** is a common problem in machine learning and statistical modeling where a model **learns the details and noise** of the training data to such an extent that it **negatively impacts the performance** of the model on new data (i.e., it generalizes poorly).

Underfitting: **Underfitting** is the opposite of overfitting. It occurs when a model is too **simple** or **too weak** to capture the underlying patterns in the data. An underfitted model will perform poorly on both the training data and new (test) data because it fails to learn the relevant structure in the data.

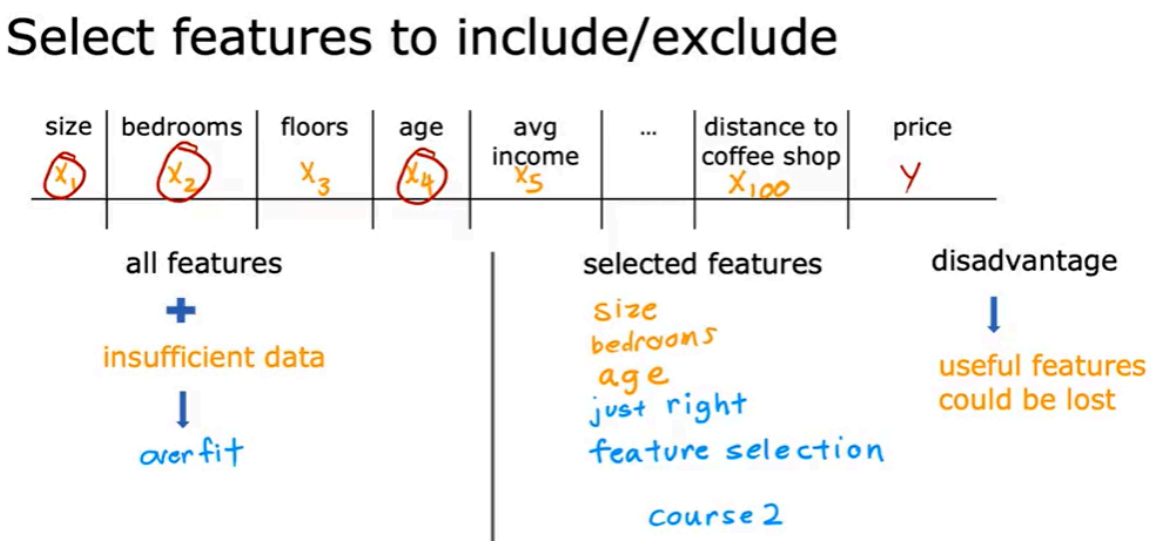


Addressing Overfitting:

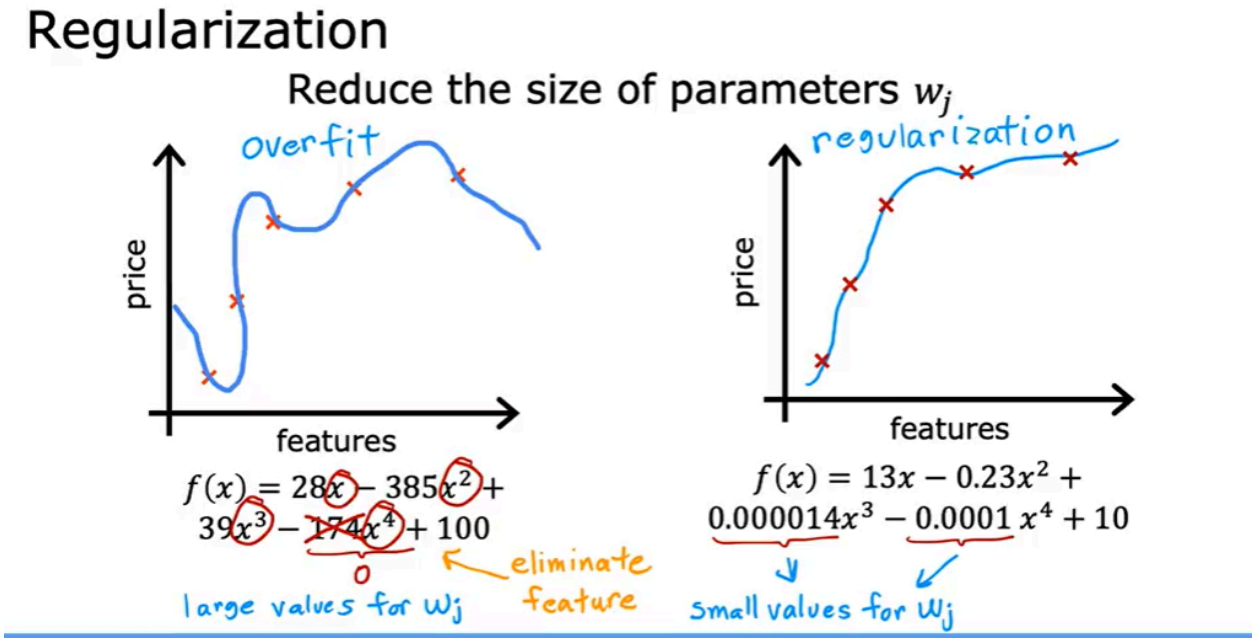
Solution to overfitting:

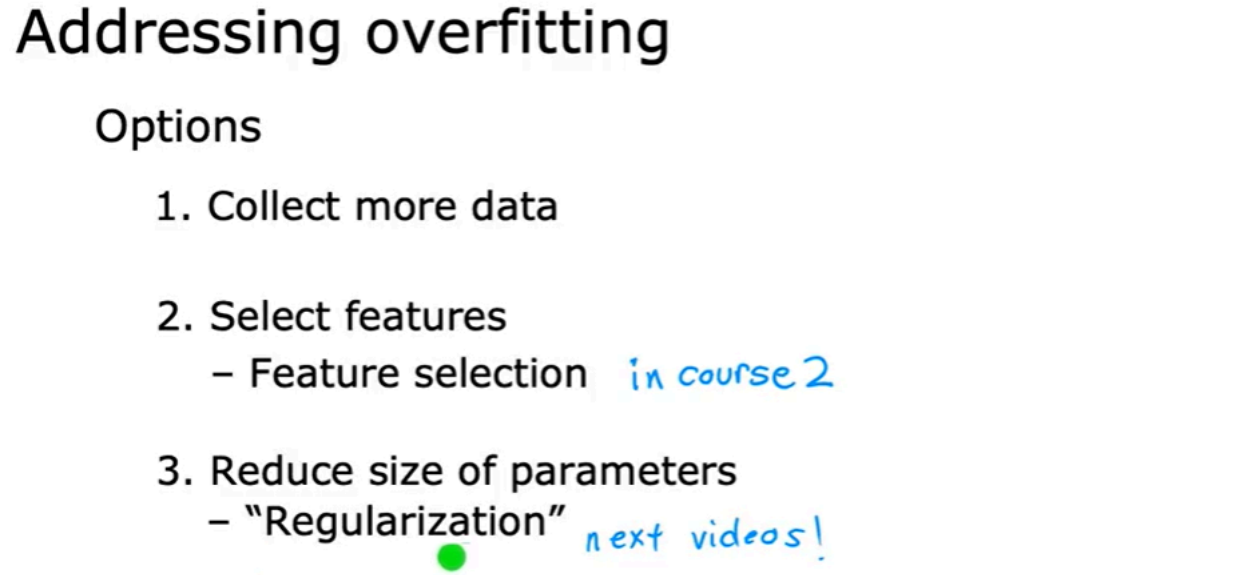
1. Collect more training examples:

2. Select features to include and exclude:



3. Regularization:





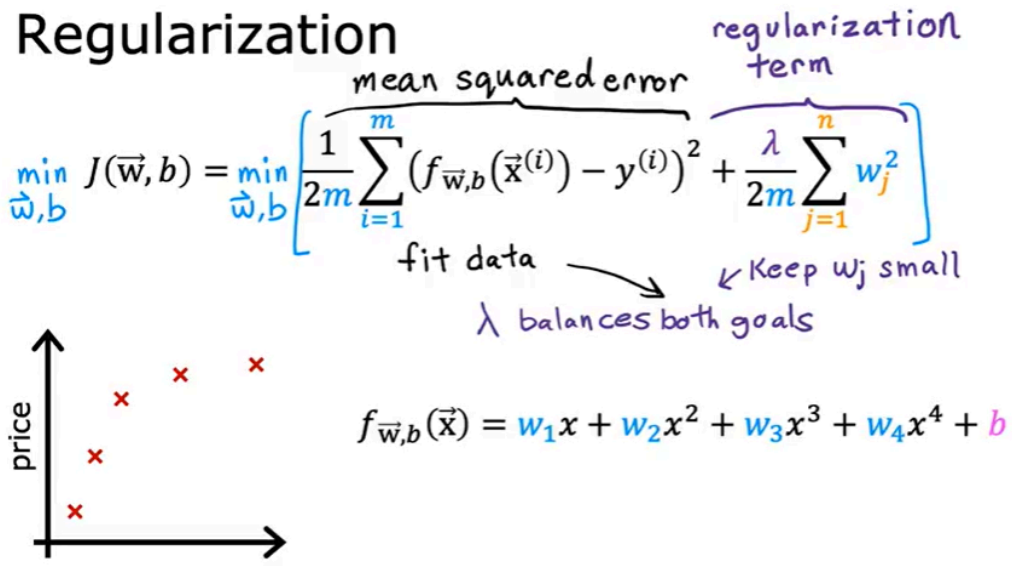
Regularization: **Regularization** is a technique used to **reduce overfitting** by adding a penalty to the loss function, discouraging overly complex models.

L1 Regularization (Lasso): Adds the absolute sum of weights as a penalty and can perform feature selection.

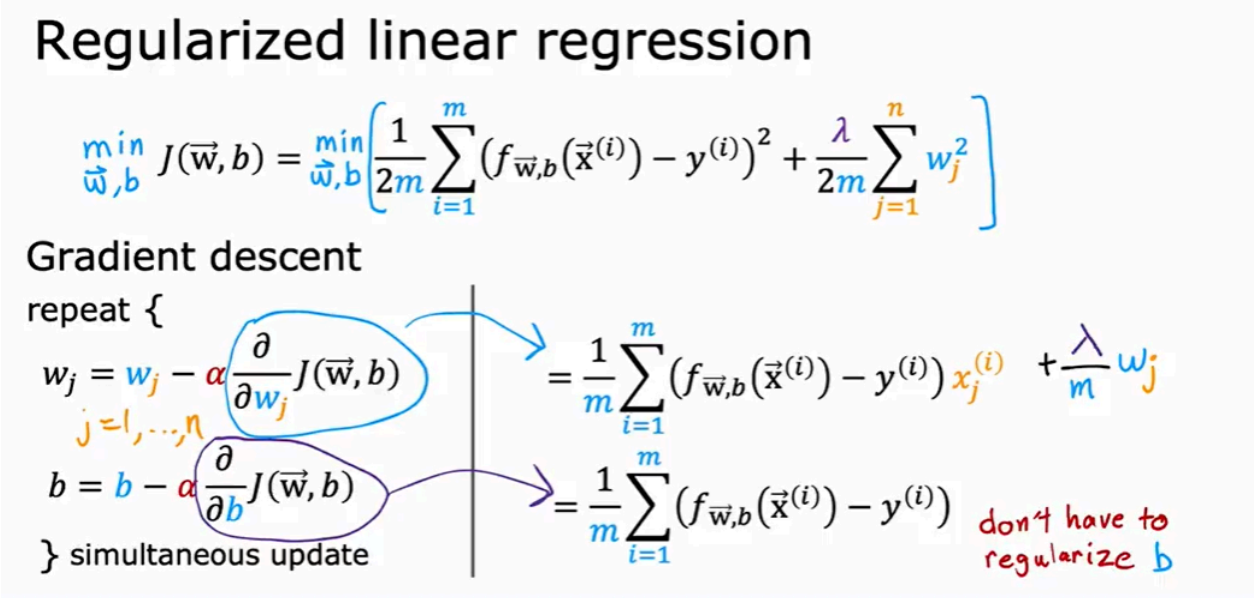
L2 Regularization (Ridge): Adds the squared sum of weights as a penalty and helps prevent large weights but doesn't eliminate them.

Elastic Net: A combination of L1 and L2 regularization.

Cost Function with Regularization:



Regularized Linear Regression:



Implementing gradient descent with regularization:

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 $j = 1 \ldots n$

Regularized Logistic Regression:

Regularized logistic regression

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

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

Gradient descent

Looks same as for linear regression!

repeat {

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

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

logistic regression

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

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

}