Machine learning

Definition: machine learning as the field of study that gives computers the ability to learn without being explicitly programmed.

3-4 types of machine learning

- Supervised machine learning
- Unsupervised machine learning
- Recommender systems
- reinforcement learning

Supervised learning : you will first train your model with examples of inputs x and the right answers, that is the labels y. After the model has learned from these input, output, or x and y pairs, they can then take a brand new input x, something it has never seen before, and try to produce the appropriate corresponding output y.

A particular type of supervised learning called regression. By regression, we're trying to predict a number from infinitely many possible numbers.
Ex: prediction of house cost based on the size of the house

And another kind of supervised learning is “classification”.

Classification supervised learning: outputs categories, instead of a single output as in the regression. That is a small number of outputs.

Summary v3: The two major types of supervised learning are regression and classification. In a regression application like predicting prices of houses, the learning algorithm has to predict numbers from infinitely many possible output numbers. Whereas in classification the learning algorithm has to make a prediction of a category, all of a small set of possible outputs.

Unsupervised learning(part 1): unsupervised learning, we call it unsupervised because we're not trying to supervise the algorithm. To give some quote the right answer for every input, instead, we asked our room to figure out all by ourselves what's interesting. Or what patterns or structures that might be in this data, with this particular data set.
clustering algorithm. Because it places the unlabeled data into different clusters and this turns out to be used in many applications.
to summarize a clustering algorithm. Which is a type of unsupervised learning algorithm, *takes data without labels and tries to automatically group them into clusters.*

Part 2:

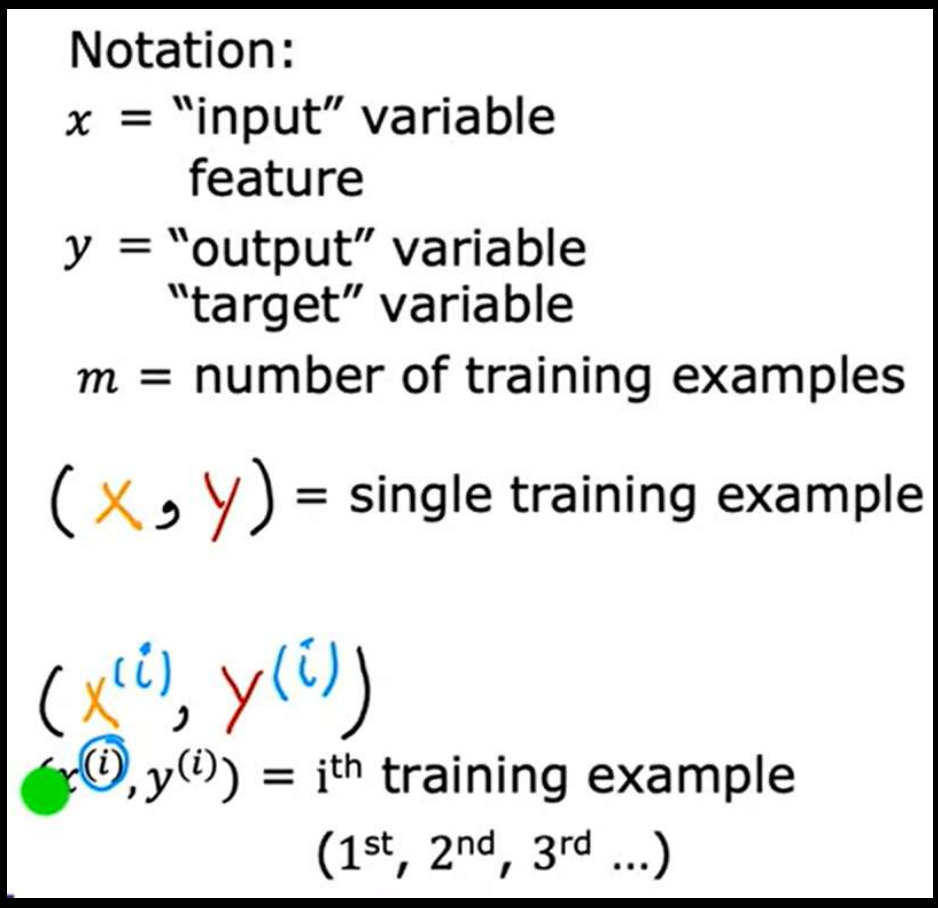
1. a clustering algorithm, which groups similar data points together.
2. anomaly detection, which is used to detect unusual events.
3. dimensionality reduction. This lets you take a big data-set and almost magically compress it to a much smaller data-set while losing as little information as possible.

Supervised machine learning

Linear regression:

With one variable:

The dataset that is used to train the model is called a **training set.**



Pass the training data(feature x) set to a machine learning algorithm and that algorithm outputs a prediction ŷ(y hat), our

aim is to get this ŷ as close as possible to the y(the target variable).

Cost function formula:

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

linear line equation: **y = mx + c**

$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

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

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

Cost function: Squared error cost function

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

error

m = number of training examples

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

intuition

2 isn’t compulsory in the above equation. And ŷ is the predicted value by the model and the y is the actual value.

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved(1/2m) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the 1/2 term.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (denoted by f(x)) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of the cost function will be 0.

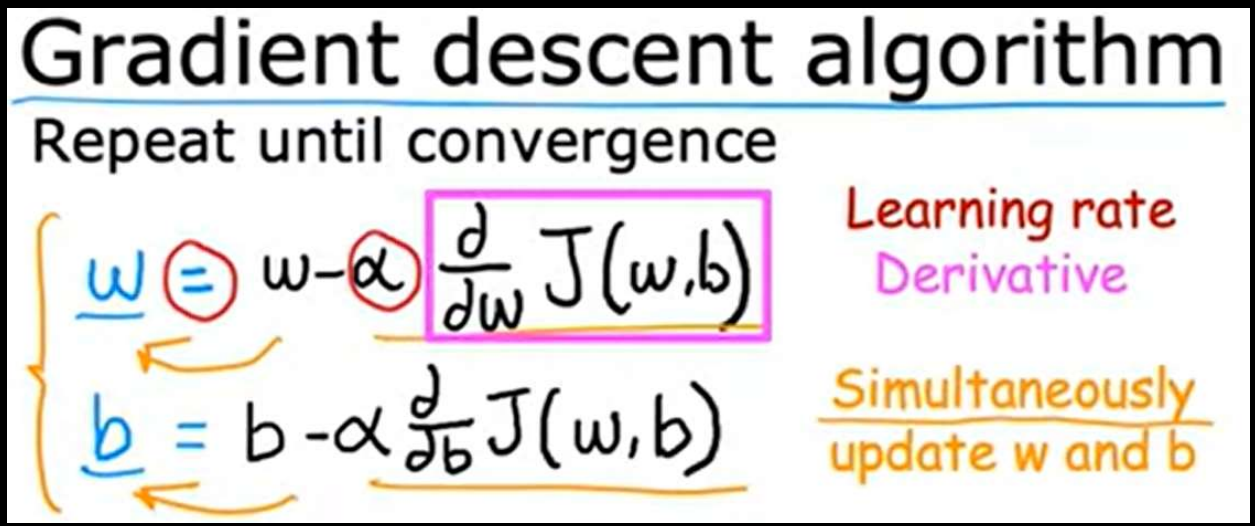
ML:Gradient Descent

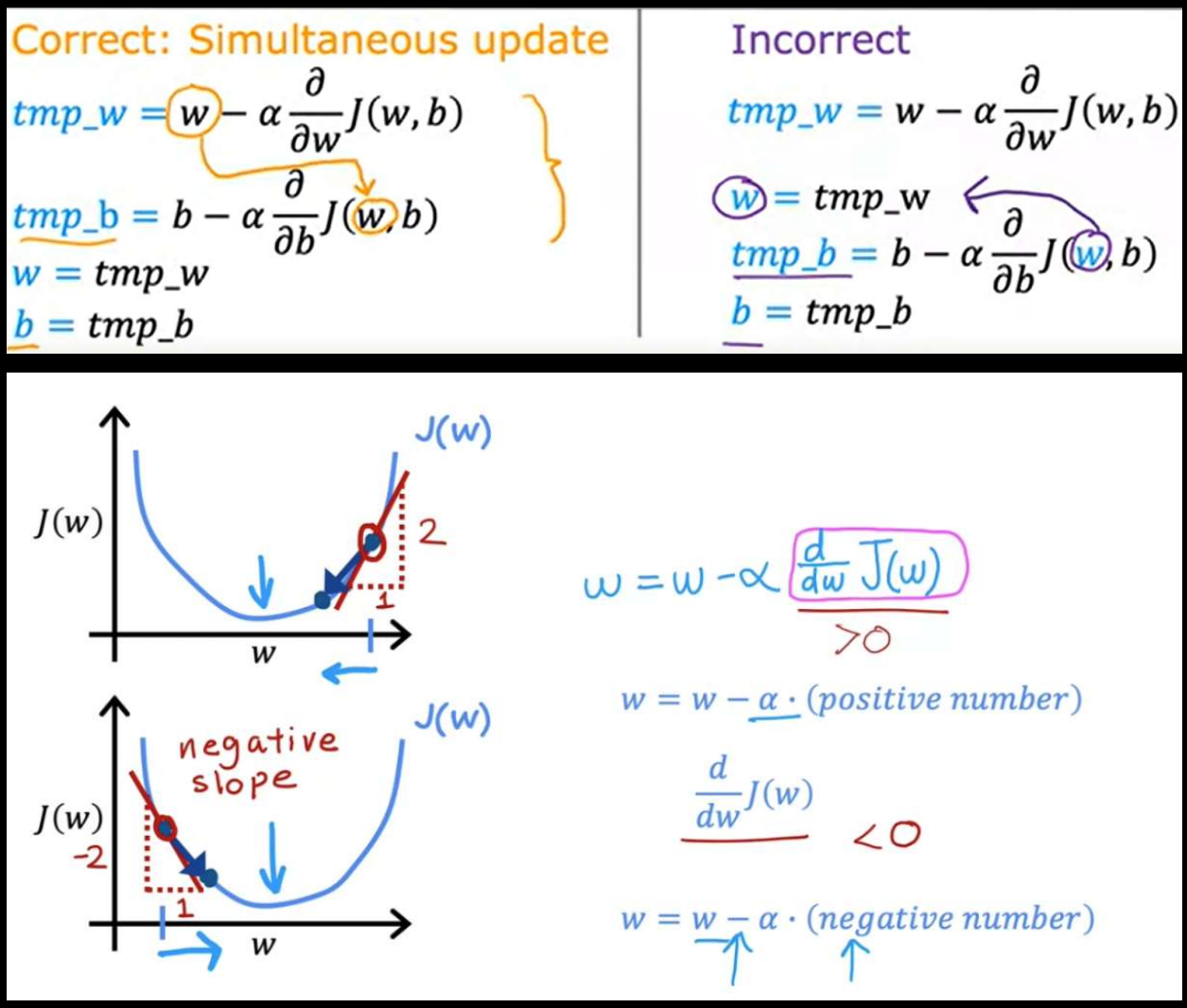
So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in hypothesis function. We do this using the gradient descent.

Imagine that we graph our hypothesis function based on its elds and (actually we are graphing the cost function as a function of the parameter estimates).

We put the x axis “w” and “b” on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific (w and b) parameters. We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter α, which is called the learning rate.

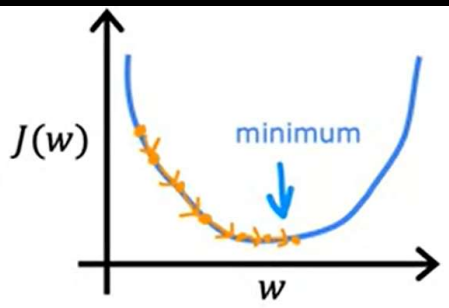




This image shows how the w equation helps to find the best version of itself.

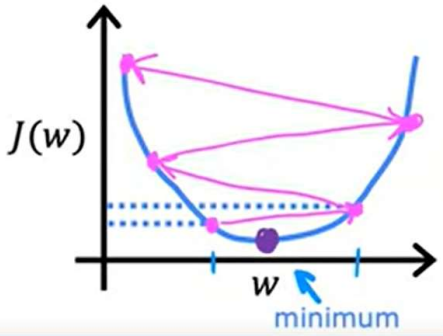
$$w = w - \alpha \frac{d}{dw} J(w)$$

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



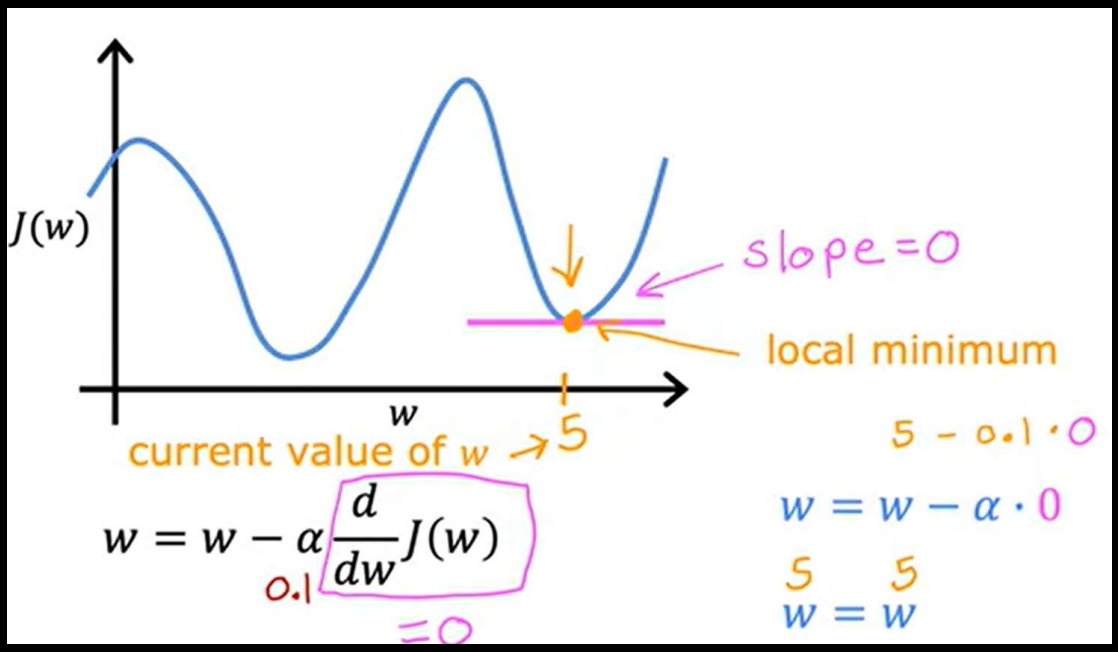
If α is too large...

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



When it's slow means it would take so many steps to come to the right value. And in the other case that's when it's too large then we may accidentally skip the real **wanted** value by overshooting.

Gradient descent remains at the local minima when it reaches one.



Near a local minimum,
- Derivative becomes smaller
- Update steps become smaller

Can reach minimum without decreasing learning rate α

(Optional)

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

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

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

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

no $x^{(i)}$

This above image shows the case where the partial derivative of j wrt to w and b is found out, the resulting term has only one

difference: that the derivative wrt to b doesn't get multiplied with x term in the end.

Gradient descent algorithm

repeat until convergence {

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

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

}

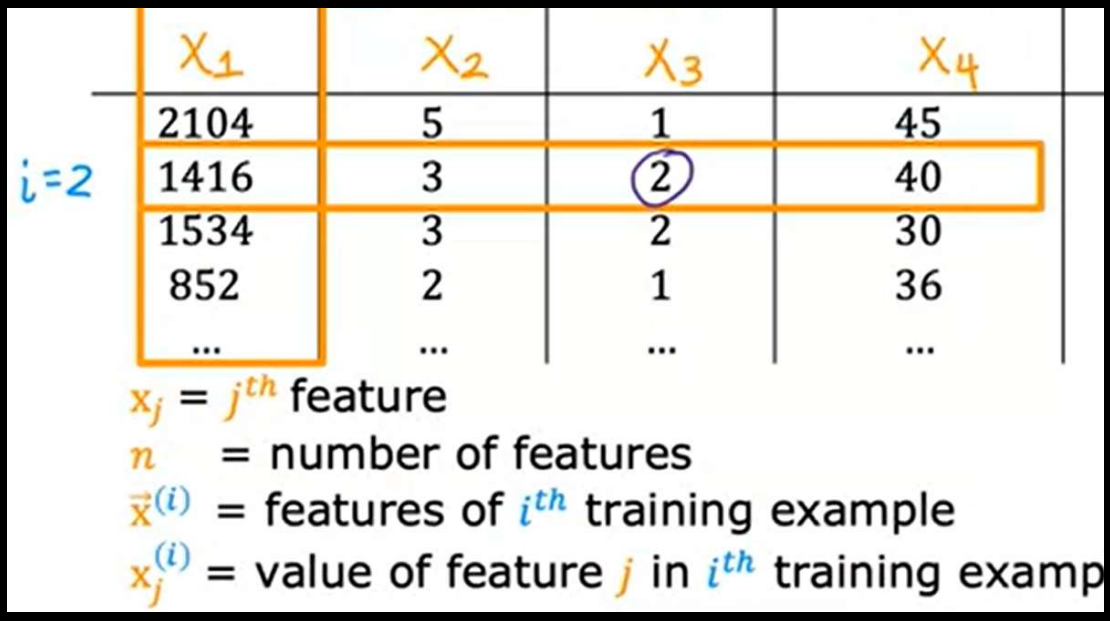
update w and b simultaneously.

When we are using a squared error cost function, then it is guaranteed that it will have only minima and it is called a convex function.

Batch gradient descent: each step of gradient descent uses all the training examples.

Multiple linear regression:

Some conventions: superscript >> its a row, contains features of (i)th training example.
Postscript >>> its a feature

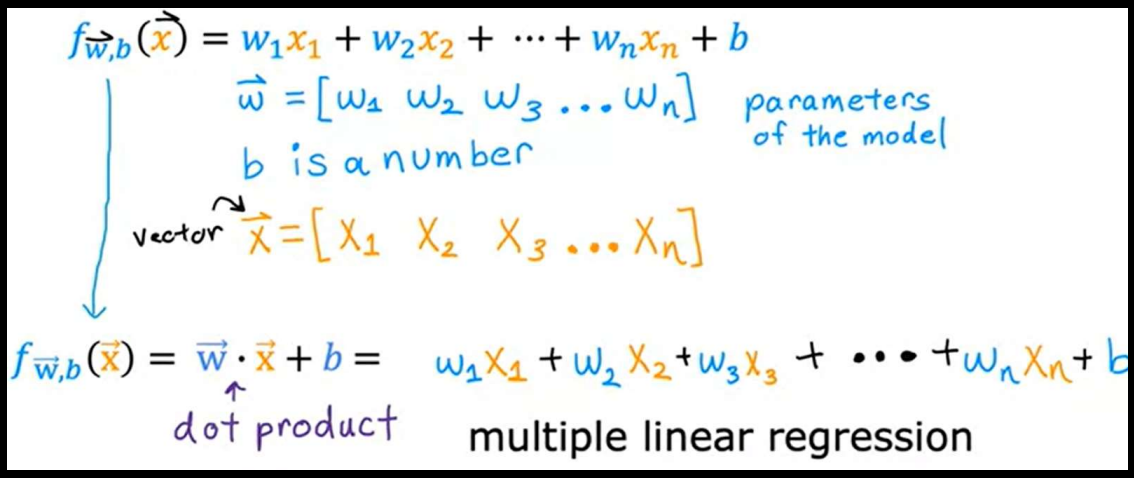


$x_j^{(i)}$ = value of feature j in the i^{th} training example

$x^{(i)}$ = the column vector of all the feature inputs of the i^{th} training example

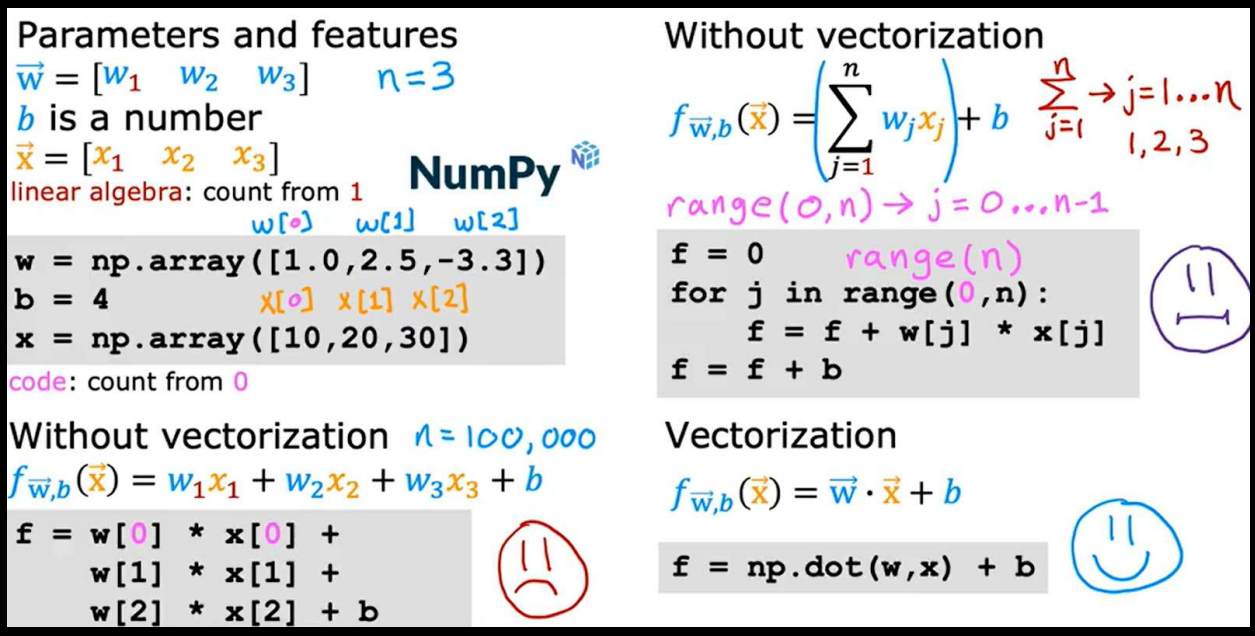
m = the number of training examples

$n = \left|x^{(i)}\right|$; (the number of features)

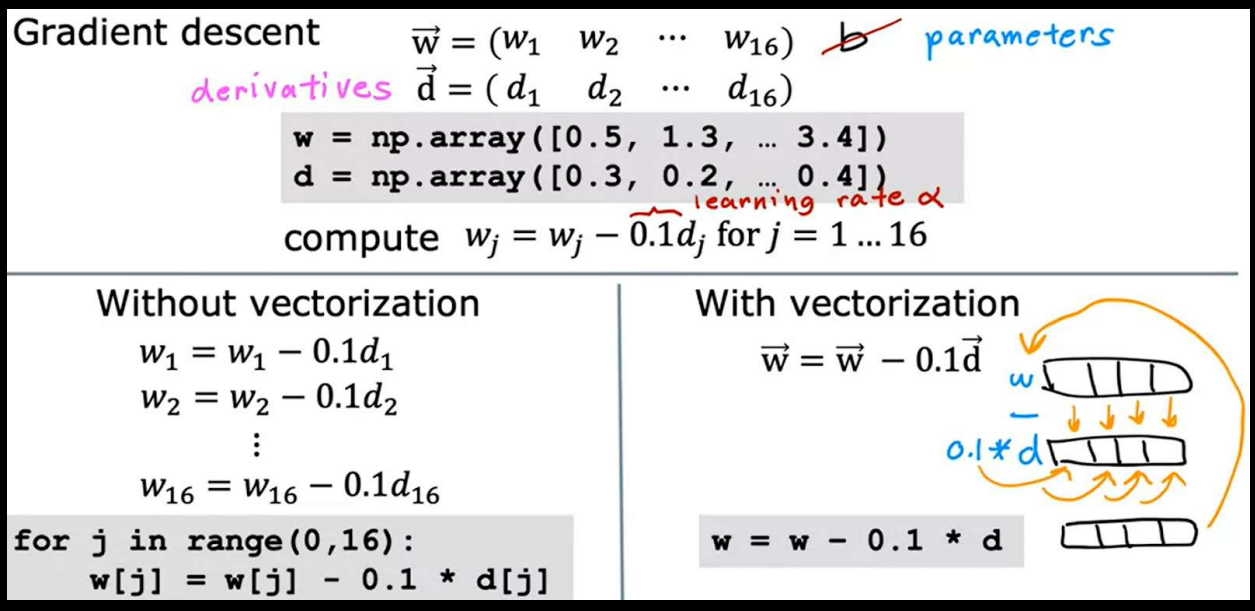


In order to develop intuition about this function, we can think about "**b**" as the basic price of a house,w1 as the price per square meter,w2 as the price per square meter, etc. x1 will be the number of square meters in the house,x2 the number of floors, etc.

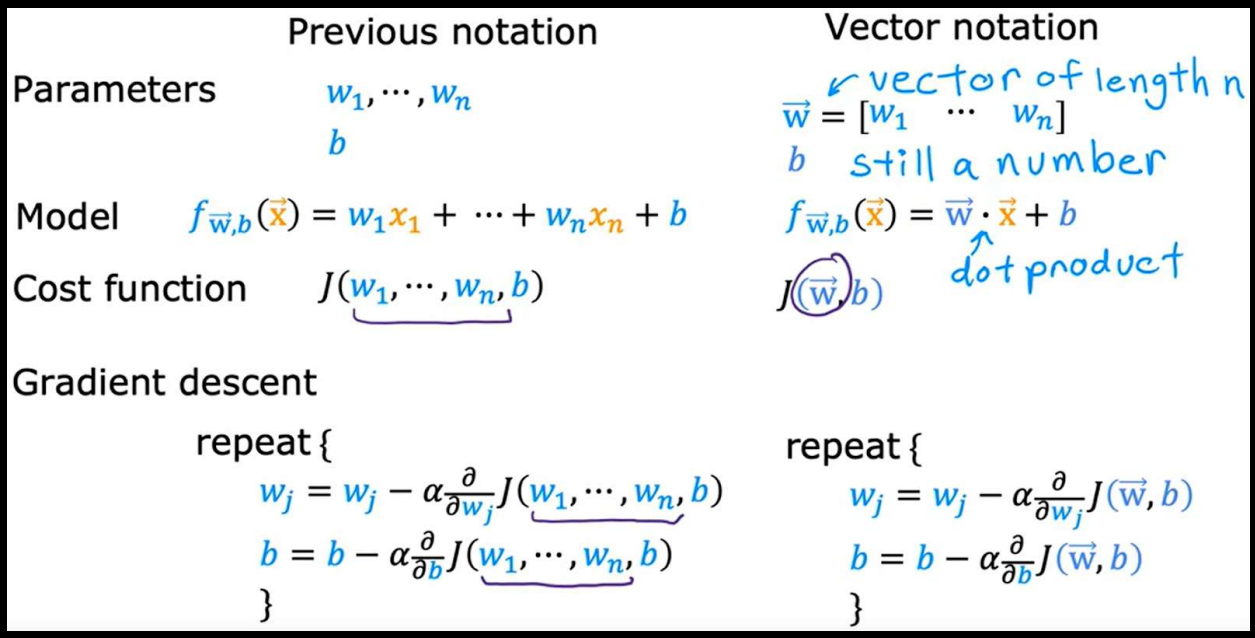
For example: if b is the base price of the house and let w_1 be a feature like price per square meter while x_1 be the number of square meters in the house.



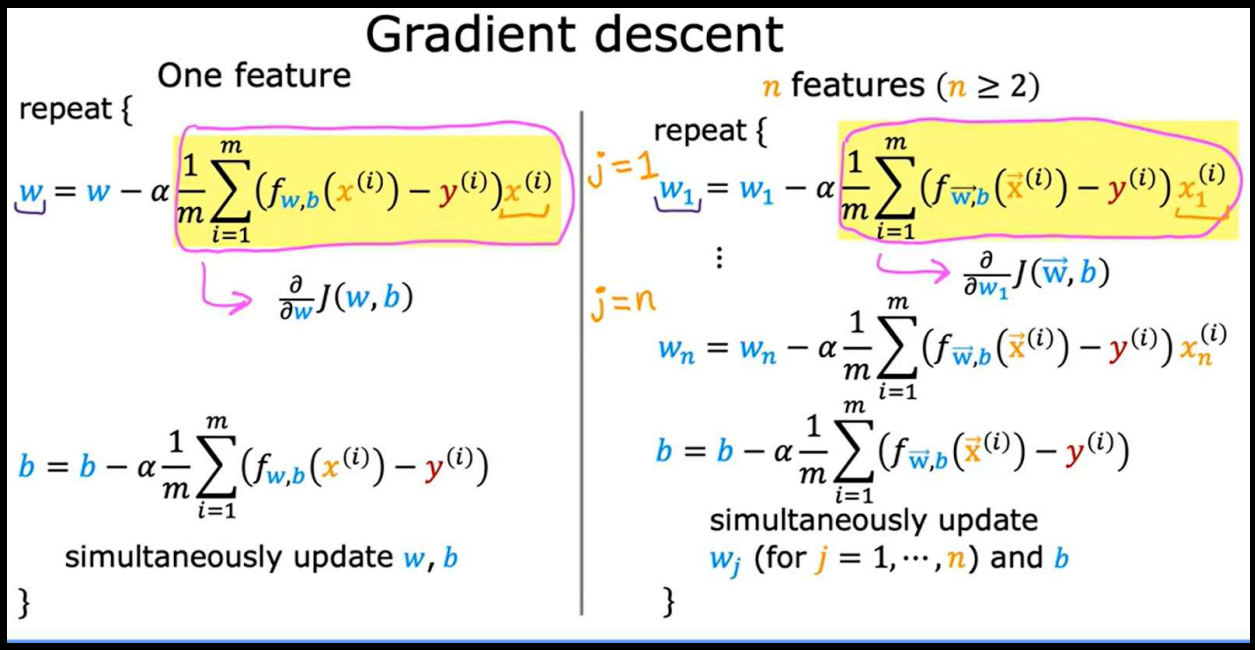
The NumPy dot function is able to use parallel hardware in your computer(i.e, make use of both CPU and GPU) .



This image talks about the speed provided by numpy array to deal with the problems in a faster way than compared to the brute force method.
Numpy mainly uses parallel computing to solve the problem in a much faster way.



Here to represent multiple values of the “w”, we start using the vector notation for the same. The model representation is the dot product of the parameter vector $w_{(1 \text{ to } n)}$ and $x_{(1 \text{ to } n)}$.



On the left is the case when the w is found out for only one feature x, while on the right is the case w is found for cases 1 to n and for all these steps the b is simultaneously getting updated.

There is an alternative to the gradient descent i.e, **Normal equation** that is optimized only for the linear regression. This doesn't go through the usual iterative stuff and all. Even then gradient descent is recommended. There are advanced libraries to use normal equation to find the w and b.

To make gradient descent run faster:

Features and parameter values:

For a feature whose values are usually large mostly smaller values for the w will be chosen

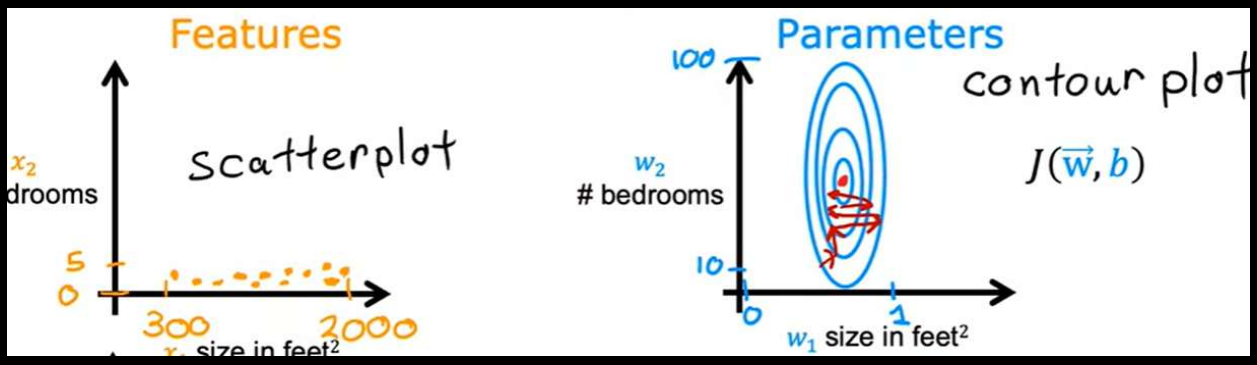
and

features whose values are usually small, larger values of w can be chosen. Since smaller valued feature won't give significantly large output compared to the former case.

	size of feature x_j	size of parameter w_j
size in feet^2	⟷	⟷
#bedrooms	⟷	⟷

Here the feature size would have a large number of possible values while the number of bedrooms would be very small and parameter would be chosen appropriately.

In this case

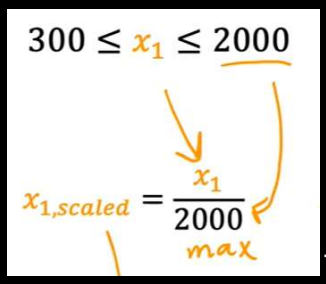


In this case, to find the gradient descent it would have to take several steps as the contour plot is not very easy to comprehend/symmetric.

So we use a method in which we rescale the features and that would scatter the data more evenly and thus resulting in a very symmetrical contour plot and its much easier to go to the local minima in this scenario.

Here both the features are scaled from 0 to 1(**that is comparable range values)** instead of their original ranges, this alters the contour plot for data and results in easier finding of the local minima(or the cost).

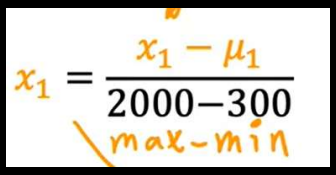
Feature scaling methods



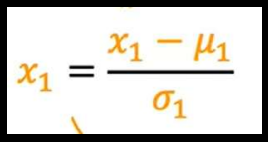
This is the feature scaling method in which the actual value is divided by the max of the range resulting in the values in between 0 and 1.

Mean normalization:

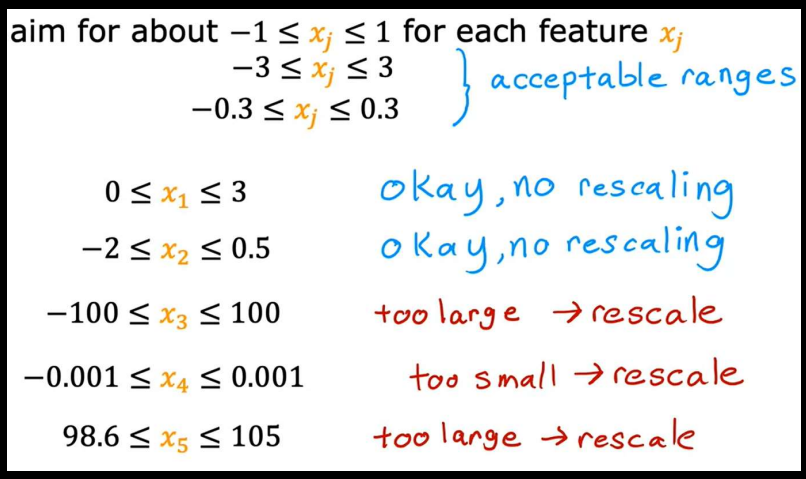
Mu1 is the mean of the given data

Scaled values lie in between -1 and +1. 

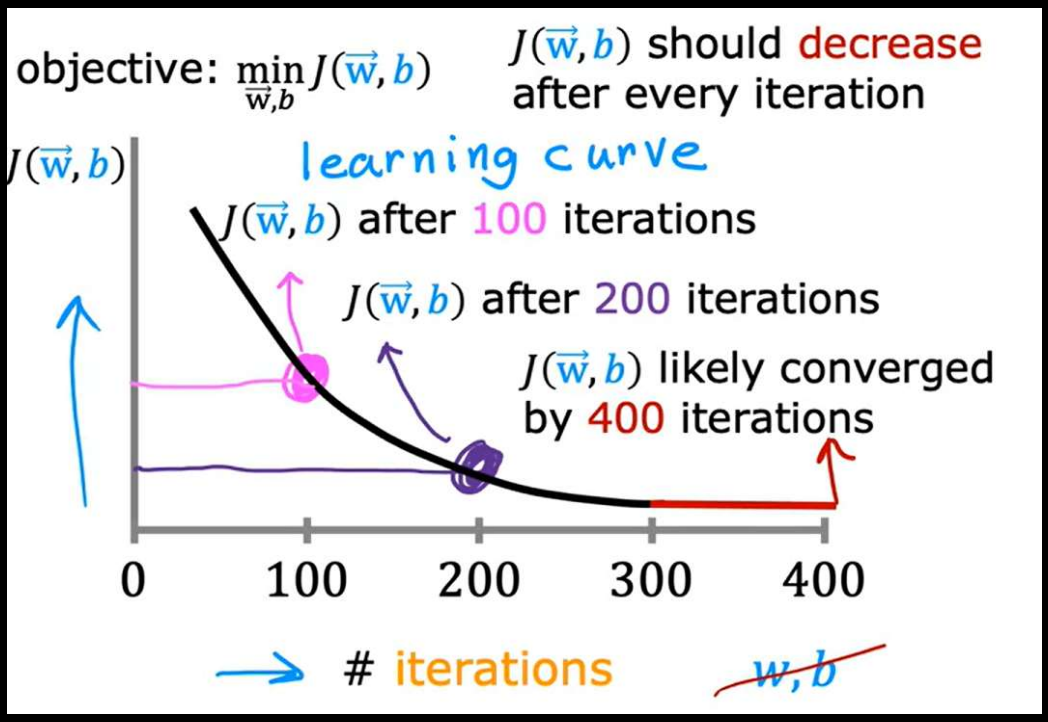
Z score normalization:



here standard deviation σ_1 is used and the mean mu1 is used as in the above case to find the scaled value for the x_1.



To measure whether the gradient descent is working correctly or not we plot a graph of cost(j) v/s number of iterations of gradient descent.



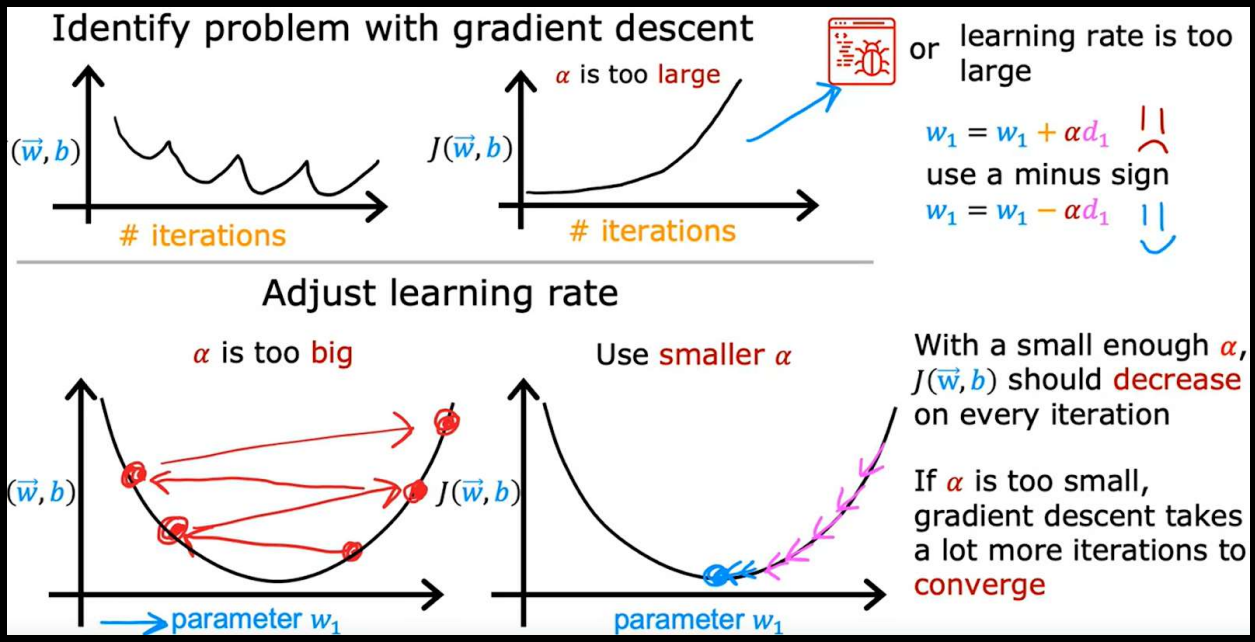
Here on every iteration the cost is decreasing and as it reaches iterations around 300 the cost almost becomes stagnant meaning that's the minimum cost for the given example.

Another way to decide when your model is done training is with an **automatic convergence test.**

Here is the Greek alphabet epsilon. Let's let epsilon be a variable representing a small number, such as 0.001 or 10^-3. If the cost J decreases by less than this number epsilon on one iteration, then you're likely on this flattened part of the curve that you see on the left and you can declare convergence.

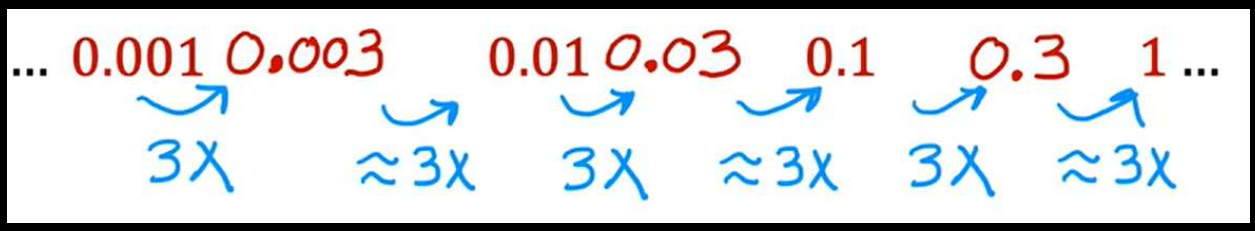
Learning rate(alpha)

If it's too large we may miss the actual value and the cost might keep increasing while it's too small that would mean that it would take too many iterations to find the local minima for the cost j.



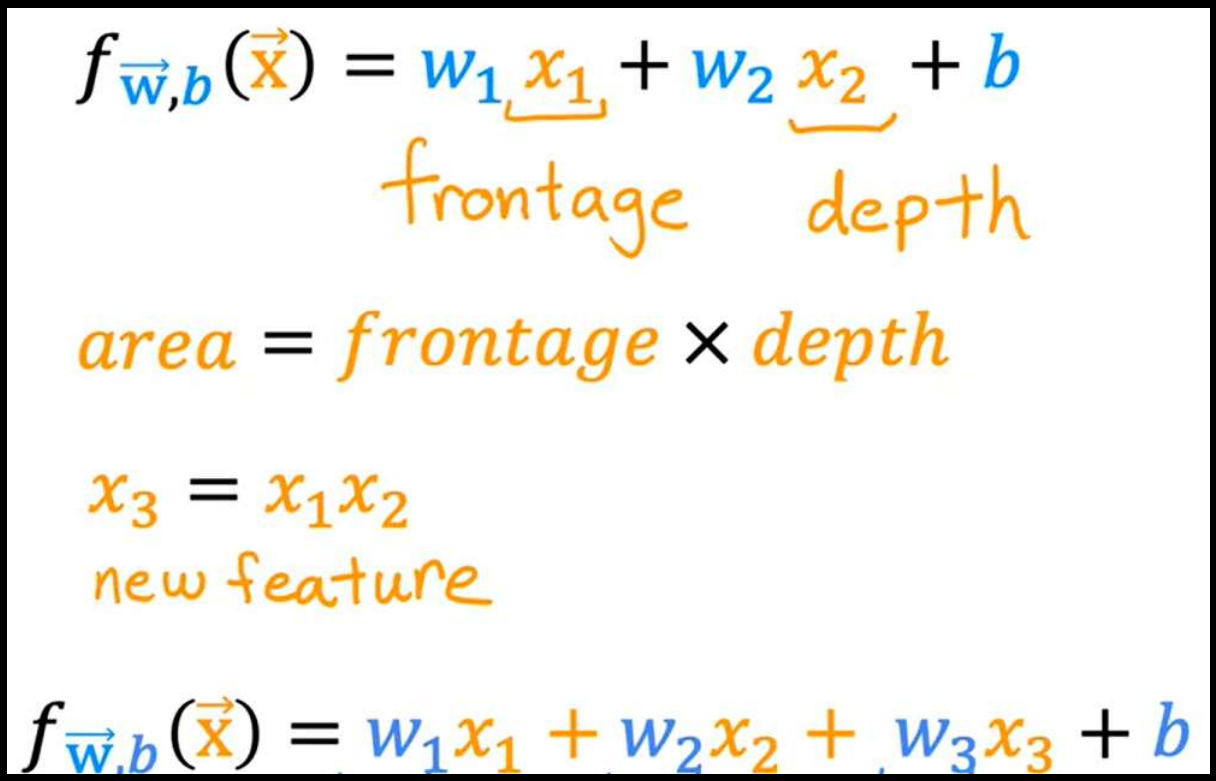
if your learning rate is too small, then gradient descents can take a lot of iterations to converge. So when I am running gradient descent, I will usually try a range of values for the learning rate Alpha. I may start by trying a learning rate of 0.001 and I may also try learning rate as 10 times as large say 0.01 and 0.1 and so on. For each choice of Alpha, you might run gradient descent just for a handful of iterations and plot the cost function J as a function of the number of iterations and after trying a few different values, you might then pick the value of Alpha that seems to decrease the learning rate rapidly, but also consistently. In fact, what I actually do is try a range of values like this. After trying 0.001, I'll then increase the learning rate

threefold to 0.003. After that, I'll try 0.01, which is again about three times as large as 0.003. So these are roughly trying out gradient descents with each value of Alpha being roughly three times bigger than the previous value. What I'll do is try a range of values until I found the value of that's too small and then also make sure I've found a value that's too large. I'll slowly try to pick the largest possible learning rate, or just something slightly smaller than the largest reasonable value that I found.



Feature engineering

Using intuition to design new features by combining or transforming existing features.



Polynomial regression:
Use x^2 and x^3 or x^0.5 for finding more suitably fitting lines. Based on these, one can select better features for the data processing.

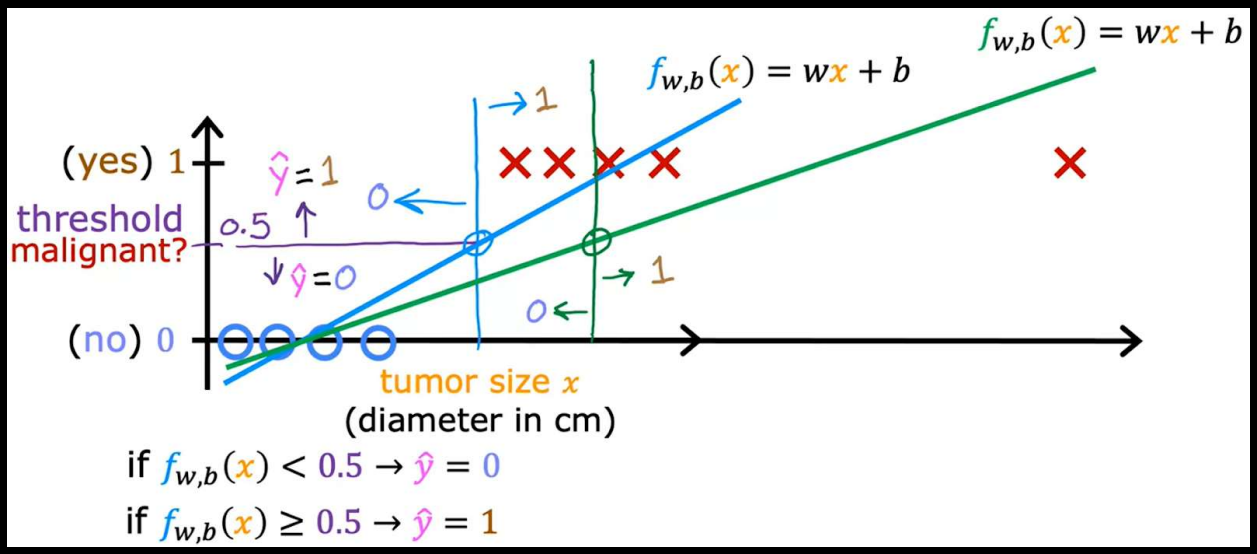
Classification

Intro:

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.

If the output has only two categories as the solutions then its known as binary classification, conventionally 0(or its equivalent like false) could be termed as negative class while 1 could be termed as positive class.

Linear regression doesn't work quite well with the classification problems.



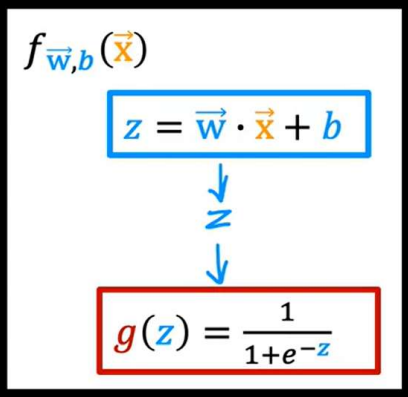
Here after adding a new data point for yes in the above plot the best fit line shifts resulting in classifying two malignant cancers as non-malignant cancers.

Ex. for classification problems: is an email spam or not?Is cancer malignant or not?, is a transaction fraudulent or not?.. For these the answers could be just two categories like "yes" or "no".

Logistic regression:

logistic regression fits in a curve defined by the sigmoid function/logistic function instead of the straight line as in the linear regression.

$$g(z) = \frac{1}{1+e^{-z}}$$ This equation outputs the values between 0 and 1 only.



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

"logistic regression"

This is the logistic regression model.

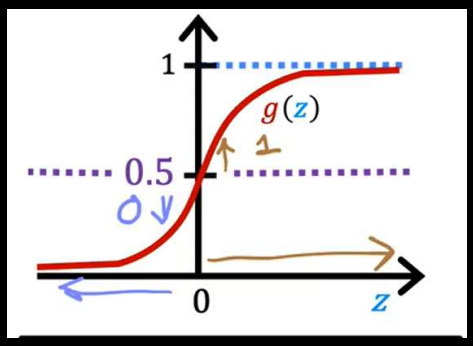
Here $f_{w,b}(x)$ denotes the "Probability" that the class is 1.

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

I.e, probability of y = 1 for x =1 with w and b as the parameters.

p(y=0) + p(y=1) = 1;

Decision Boundary:



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

$= P(Y=1|\vec{x};\vec{w},b)$ to decide 0 or 1

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

Yes: $\hat{Y}=1$. No: $\hat{Y}=0$. | $\hat{y}$ = prediction

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

$g(z) \geq 0.5$

$z \geq 0$

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

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

decision boundary

for ex. taking & plotting a graphs that includes 2 features x_1 & x_2:

i.e; $f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1+w_2x_2+b)$

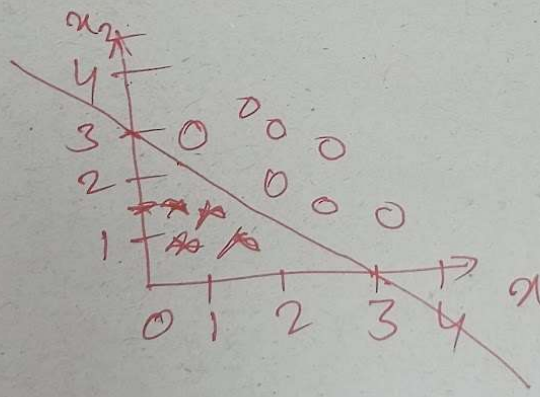
for ex. take $w_1=w_2=1$ & $b=-3$

decision boundary $z = \vec{w}\cdot\vec{x}+b=0$ | equal to zero

$z = x_1+x_2+b=0$

$x_1+x_2-3=0$

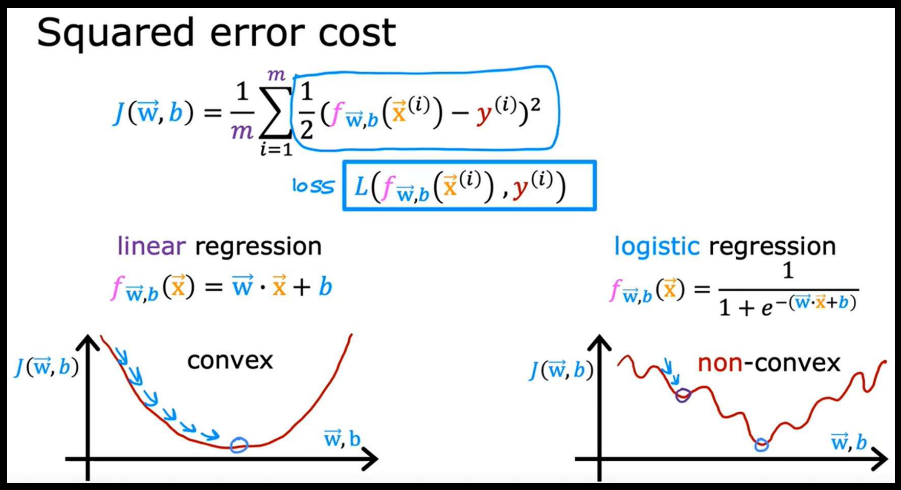
$x_1+x_2=3$



all the data points that come left are into one class & other into another class

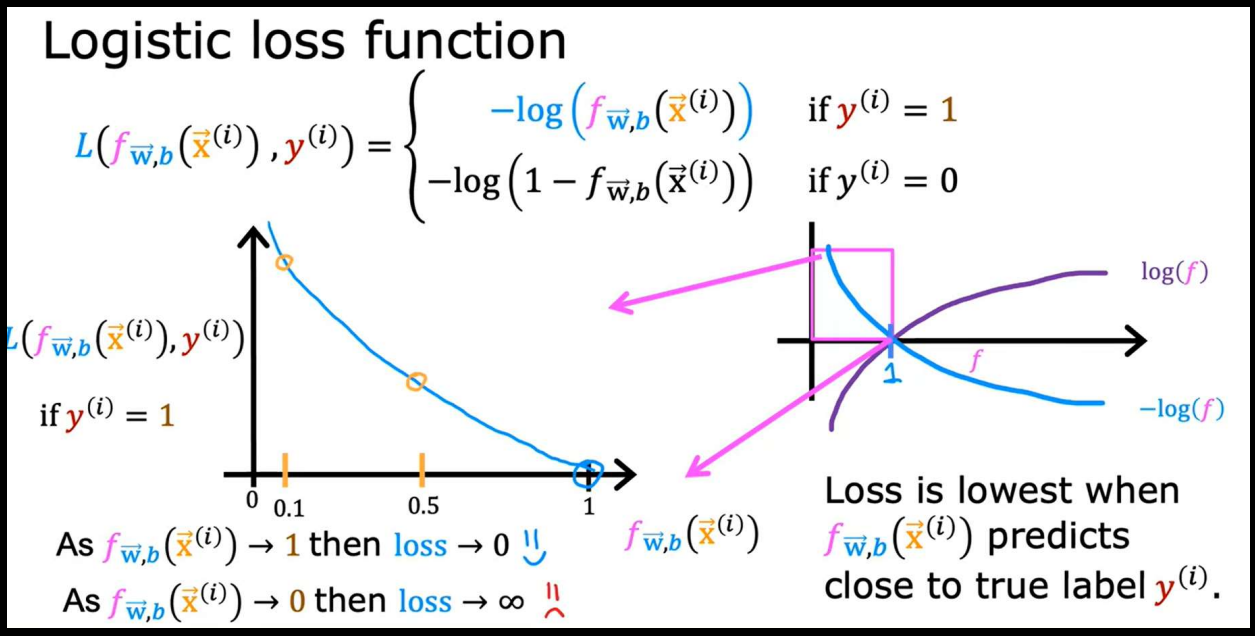
there are non linear decision boundaries are also possible, where z is complex. for ex

$z = w_1x_1^2 + w_2x_2^2 + b$. this is a circle



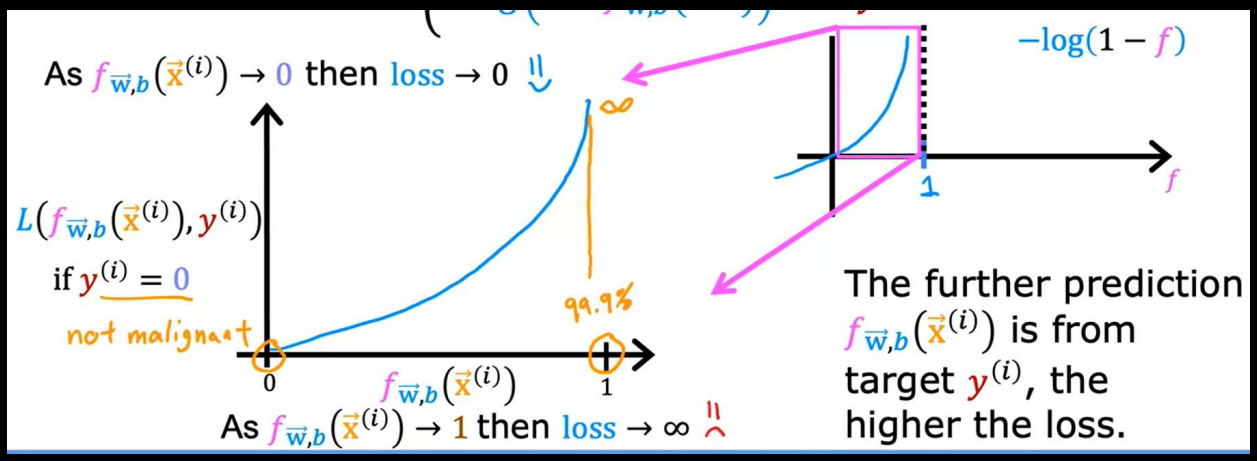
Here the cost function for the linear regression is given on the top left and resulting plotting of the same graph. But for logistic regression it doesn't give a convex curve as an output which would result in having so many local minimas.

Thus we change the definition of the cost function by introducing the new term known as **L** which is a function of prediction and the true value.



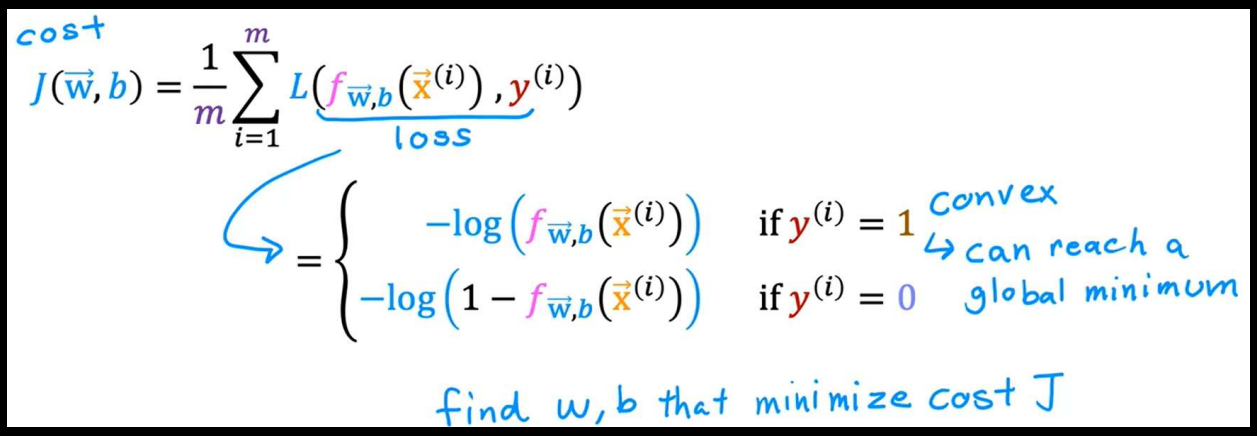
Here **L** has two outputs based on the true label(or the true value y), i.e, the top line deals with the case where the true value of y is 1. And the other line deals with a situation where the true value is 0.

Since logistic regression mainly deals with values in between 0 and 1 only. The loss is lowest whenever f(x) predicts values closest to the true label y.

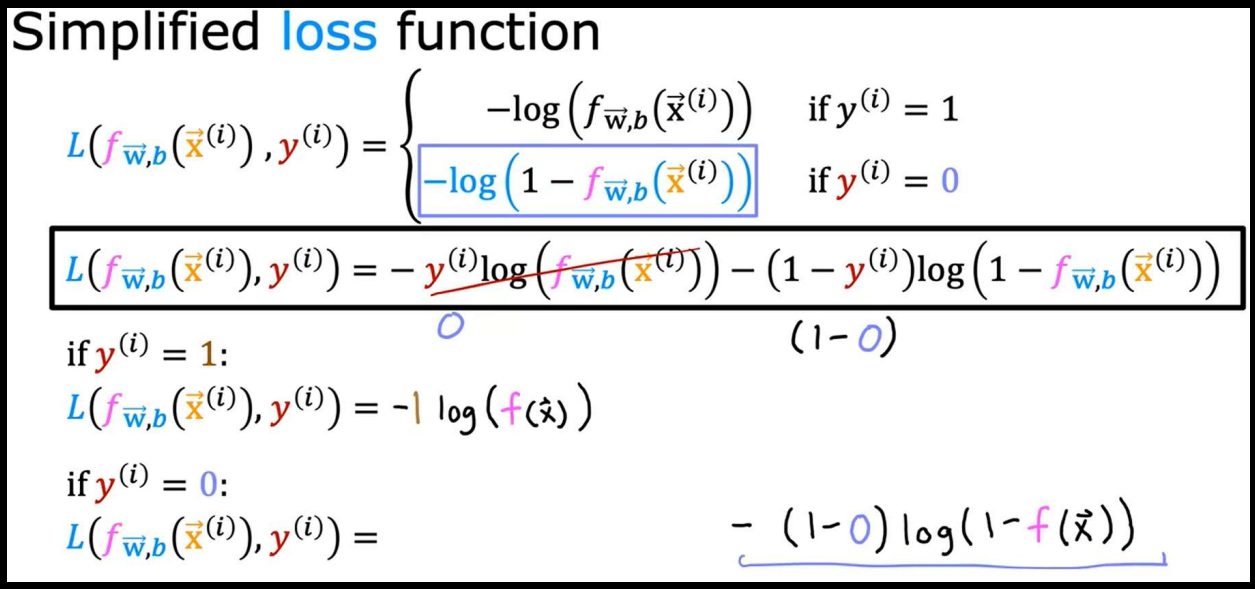


This is the case when the true label is actually 0. Our model is predicting the values by f(x).

Further the prediction from the target y, the higher is the loss.



This is the cost of logistic regression.



In this, the equation is more comprehensive as it takes care of both cases of the true labels for y i.e, 1 or 0.

IBM ML0101EN

Machine Learning with Python: A Practical Introduction

Python for machine learning:

Numpy, matplotlib, sciPy, pandas, scikit learn(contains machine learning algorithms).

Regression

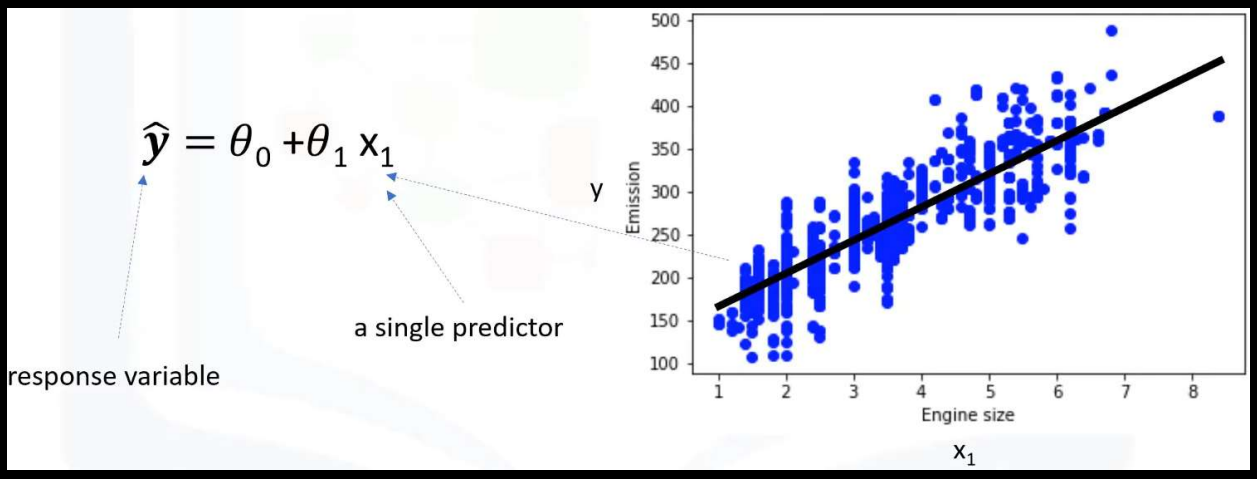
2 types

1.Simple(2 variables x and y, y is dependent on x only)(also linearity and nonlinearity exists)

2.Multiple regression(y is dependent on more than one independent variables, linearity and nonlinearity exists)

simple regression:

linear regression model representation:



This equation is of the basic equation of the straight line i.e, **y = mx +c**

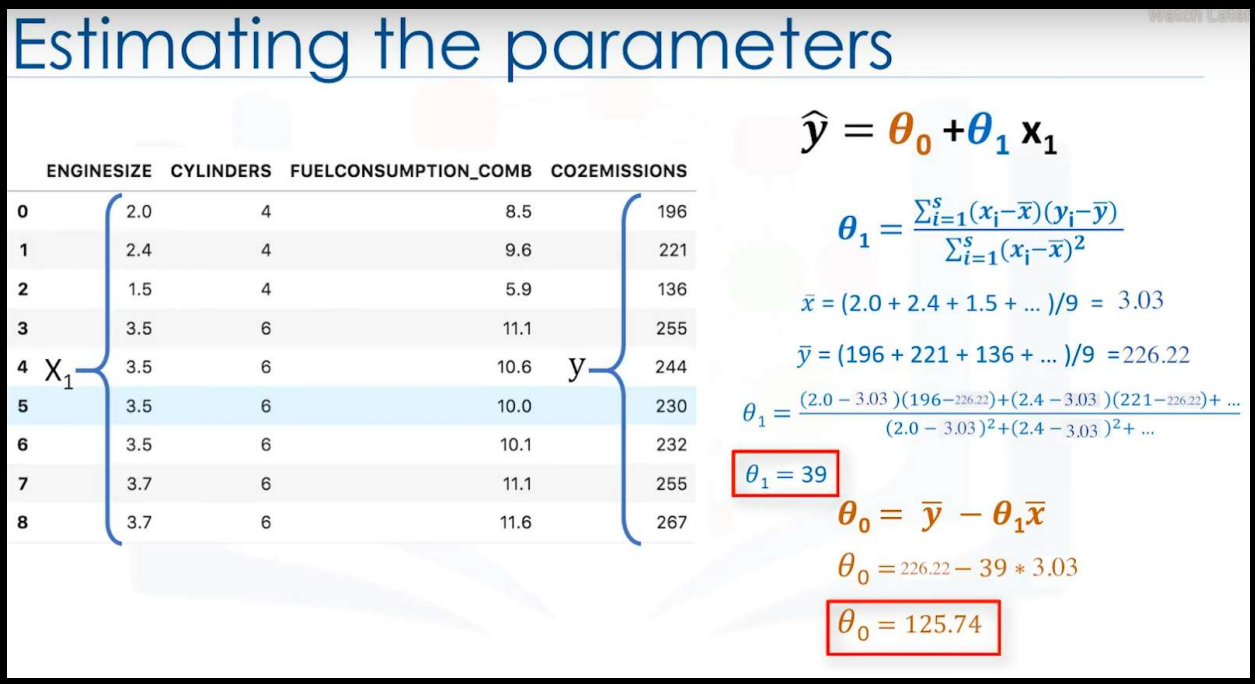
Here m is theta1 and c is theta 2 and they too have the same meanings i.e, m is slope while, c is the intercept.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Here this is used to measure how best-fitting our predicted line is to the given data. More mse means the model is pretty bad.

Ŷ is the predicted value while the y(i) is the actual value.

To minimize the error we have to find the best slope and the intercept for the model. which can be done in several ways.



The image is self explanatory.

Model evaluation:

1. **The goal of regression is to build a model to accurately predict an unknown case.**

2 methods available

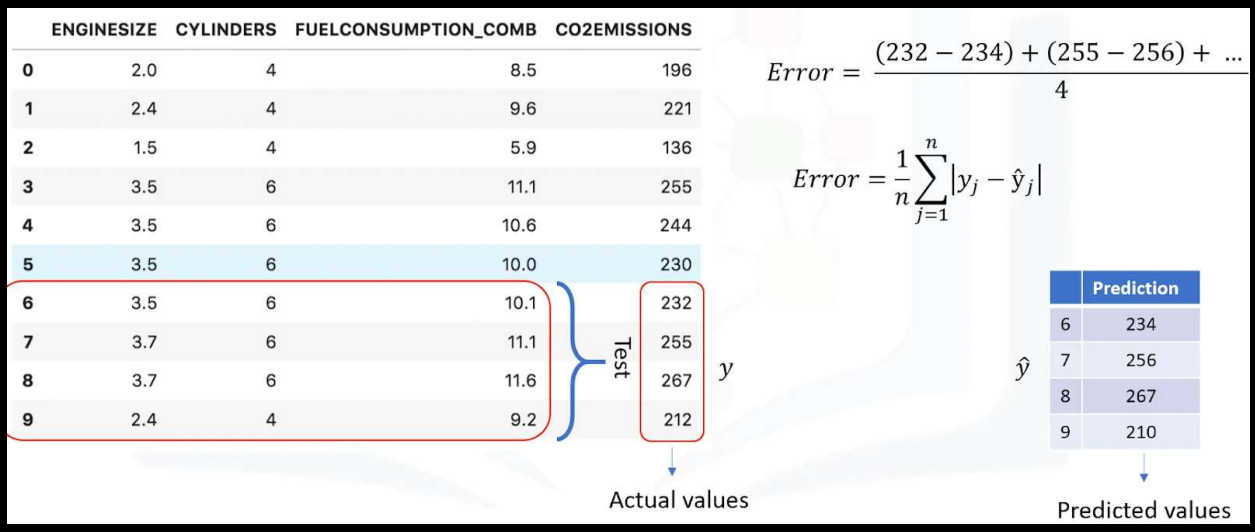
1 . train and test on same data set

2. train/test split.

train and test on same data set

is to select a portion of our dataset for testing.
For instance, assume that we have 10 records in our dataset.
We use the entire dataset for training,
and we build a model using this training set.
Now, we select a small portion of the dataset,
such as row number six to nine,

but without the labels.
This set is called a test set,
which has the labels,
but the labels are not used for prediction and is used only as ground truth.
The labels are called actual values of the test set.
Now we pass the feature set of the testing portion
to our built model and predict the target values.
Finally, we compare the predicted values by
our model with the actual values in the test set.
This indicates how accurate our model actually is.



This evaluation approach would most likely have a high training accuracy and a low out-of-sample accuracy since the model knows all of the testing data points from the training.

What exactly is **training accuracy**?
Training accuracy is the percentage of correct predictions that the model makes when using the test dataset. However, a high training accuracy isn't necessarily a good thing.
For instance, having a high training accuracy may result in an over-fit of the data. This means that the model is overly trained to the dataset, which may capture noise and produce a non-generalized model.

Out-of-sample accuracy is the percentage of correct predictions that the model makes use of data that the model has not been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy due to the likelihood of being over-fit.

It's important that our models have high out-of-sample accuracy because the purpose of our model is, of course, to make correct predictions on unknown data.

To improve **Out-of-sample accuracy** we use another method for evaluation known as train/test split model.

Train/test split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive.

After which, you train with the training set and test with the testing set.

This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that has been used to train the data.

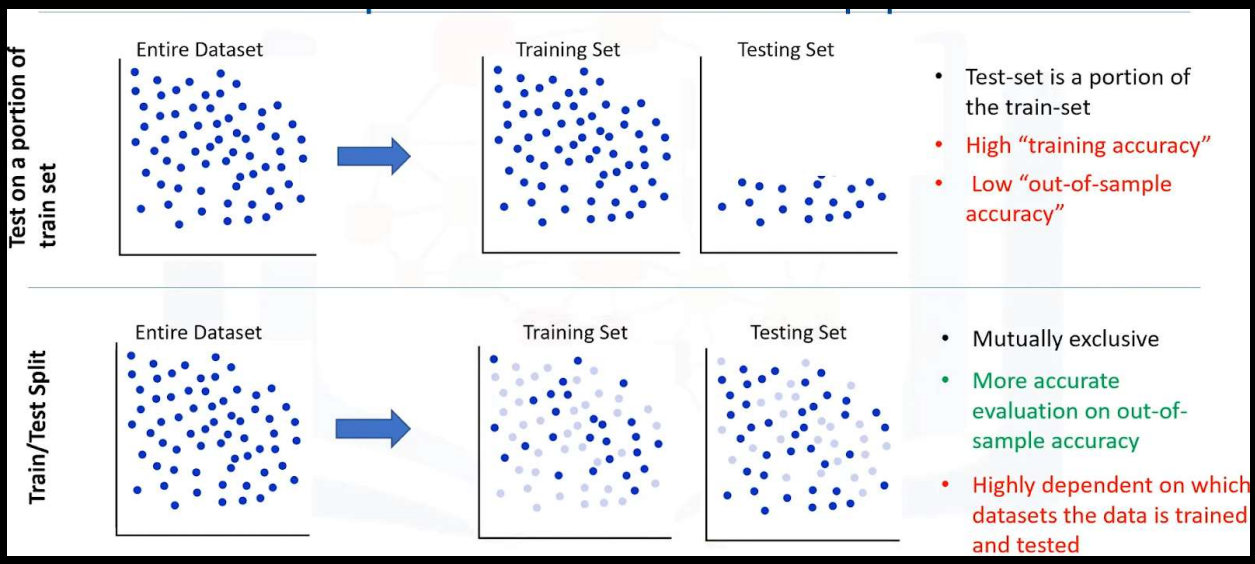
It is more realistic for real-world problems. This means that we know the outcome of each data point in the dataset, making it great to test with. Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points.

So, in essence, it's truly out-of-sample testing.

However, please ensure that you train your model with the testing set afterwards, as you don't want to lose potentially valuable data.

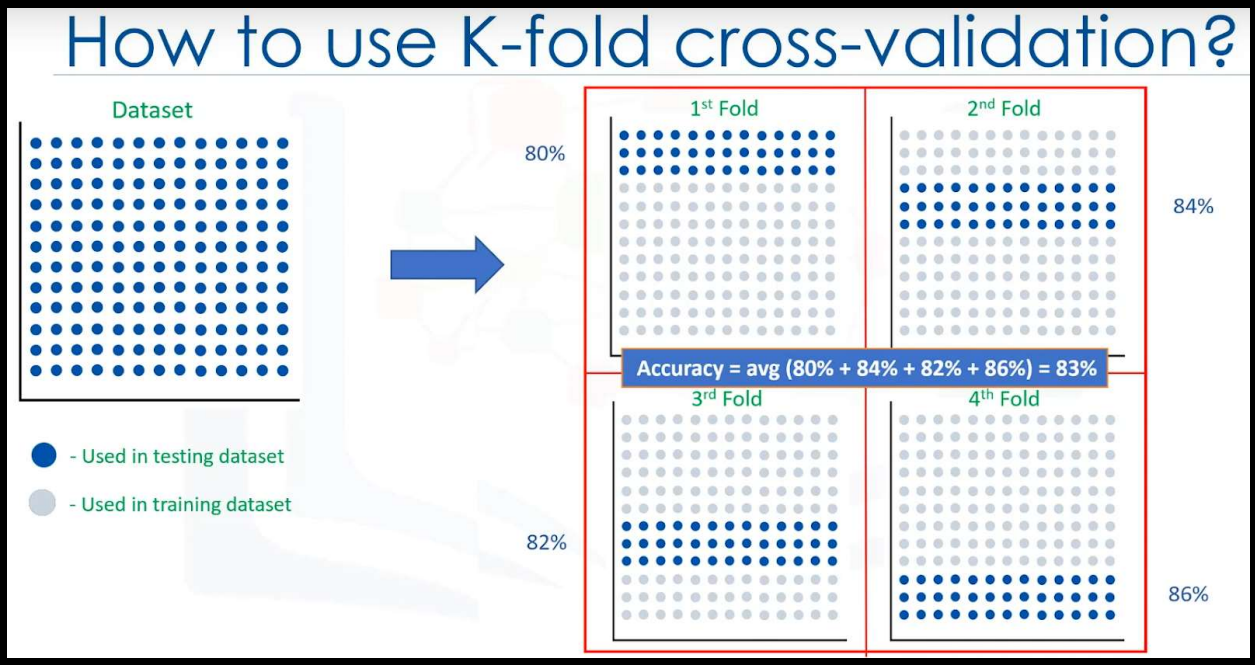
The issue with train/test split is that it's highly dependent on the datasets on which the data was trained and tested.

The variation of this causes train/test split to have a better out-of-sample prediction than training and testing on the same dataset, but it still has some problems due to this dependency.



K-fold cross-validation

in its simplest form performs multiple train/test splits, using the same dataset where each split is different.
Then, the result is average to produce a more consistent out-of-sample accuracy.

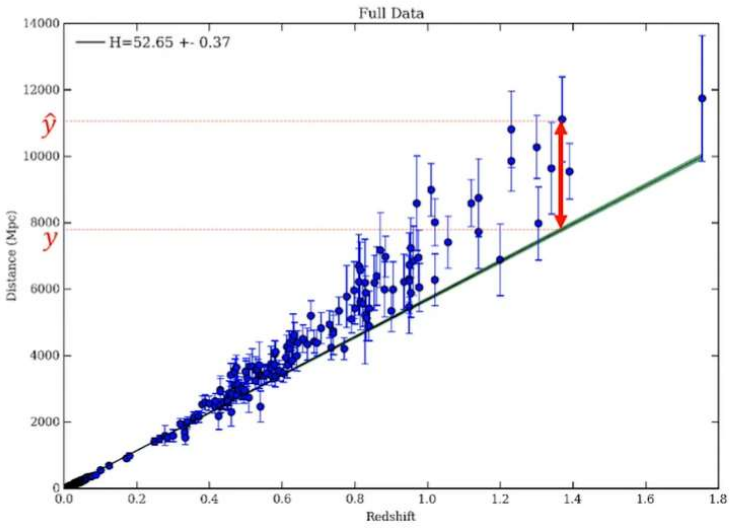


If we have K equals four folds, then we split up this dataset as shown here.
In the first fold for example, we use the first 25 percent of the dataset for testing and the rest for training.
The model is built using the training set and is evaluated using the test set.
Then, in the next round or in the second fold, the second 25 percent of the dataset is used for testing and the rest for training the model. Again, the accuracy of the model is calculated.
We continue for all folds. Finally, the result of all four evaluations are averaged.
That is, the accuracy of each fold is then averaged, keeping in mind that each fold is distinct, where no training data in one fold is used in another.

Accuracy metrics for model evaluation

Evaluation metrics are used to explain the performance of a model.

What is an error of the model?



$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

$$RAE = \frac{\sum_{j=1}^{n}|y_j - \hat{y}_j|}{\sum_{j=1}^{n}|y_j - \bar{y}|}$$

$$RSE = \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$$

$$R^2 = 1 - RSE$$

The error of the model is the difference between the data points and the trend line generated by the algorithm.

Mean absolute error

Mean squared error:the focus is geared more towards large errors.
This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.

Root mean squared error:is interpretable in the same units as the response vector or y units,
making it easy to relate its information.

Relative absolute error:also known as residual sum of square,
where y bar is a mean value of y, takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Relative squared error:it is used for calculating R squared.
R squared is not an error per se but is a popular metric for the accuracy of your model.
It represents how close the data values are to the fitted regression line.
The higher the R-squared,the better the model fits your data.