=> Week-1 :

Machine learning :

↳ Arthur Samuel : Field of study that gives computers the ability to learn without being explicitly programmed.

↳ Tom Mitchell : ~~A computer program is said~~ A computer / machine program is said to learn from experience E wrt some task T & some performance measure P if its performance on T, as measured by P, improves with experience E.

Ques : Suppose your e-mail program watches which mails you do or do not mark as spam, and based on that learns how to better filter spam. Identify the T, E & P.

→ T : Classifying mails as spam or not.

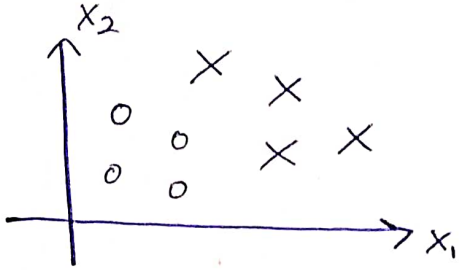
→ E : Watching you label mails as spam or not.

→ P : Number (or fraction) of mails correctly classified as spam / not spam.

⇒ Supervised learning : An ML algo that maps an input to an output based on sample input-output pairs.

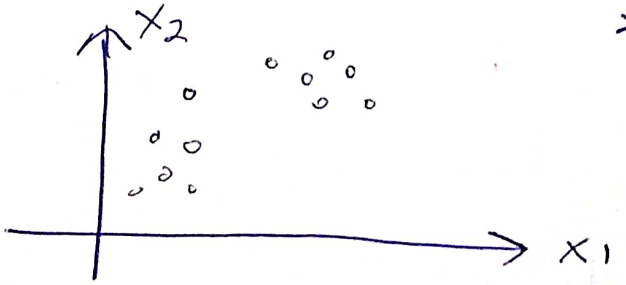
eg : SVM Support Vector Machines (SVM), Linear regression, Decision trees, K-nearest neighbour

* The data set used is completely labeled & categorized.



⇒ Unsupervised learning : An ML algo/technique where the model works on its own to discover information. eg : Clustering

* The data used is usually unlabelled.



* In unsupervised learning, there is no feedback based on the prediction results.

* Supervised learning problems are categorized into

Regression problems

→ We try to map input variables to some continuous function. eg : Given a picture of a person, predicting his age.

Classification problems

→ We try to map input variables into discrete categories. eg : Given a patient with a tumor, predicting whether tumor is malignant or benign

⇒ Simple linear regression (SLR)

Constant Coefficient

$$* \quad y = b_0 + b_1 x$$

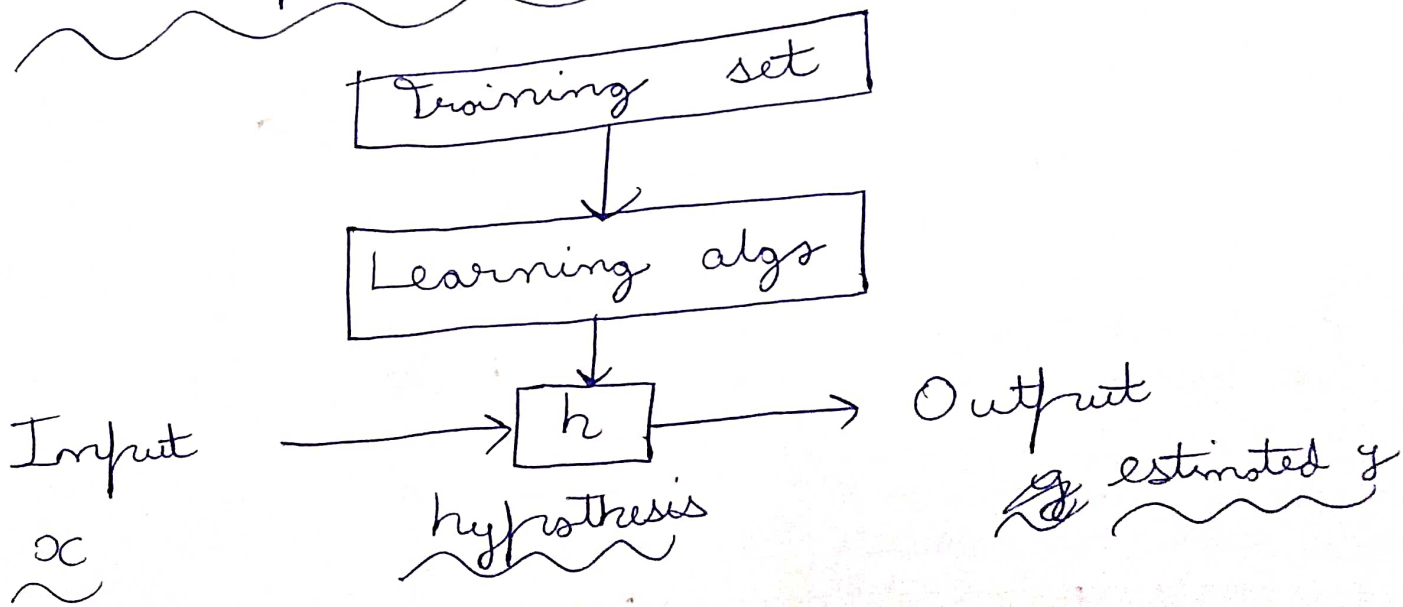
y ↙ Dependent variable

x ↑ Independent variable

* Intuitively, linear regression represents a given model as a "trend line" that "best fits" the data.

→ How supervised

→ How supervised learning works



* h maps from x's to y's.

* Formally, our goal in supervised learning is, given a training set, to learn a func^n $h: X \to Y$ so that $h(x)$ is a "good predictor" for the corresponding value of y. For historical reasons, this func^n h is called hypothesis

Ques: How do we represent h?

Ans: There are many ways. But for linear regression with one variable (or SLR), h can be represented as:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

* θ_i : Parameters

Ques: How to choose θ_0, θ_1?

Ans: Choose θ_0, θ_1 s.t. $h_\theta(x)$ is close to y ∀ training samples (x, y).

→ For SLR, choose θ_0, θ_1 s.t.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)^2$$

J: Squared error cost function

m: Total no. of training samples

is minimum.

Here $h_\theta(x_i) = \theta_0 + \theta_1 x_i$

⇒ Cost function : We can measure the accuracy of our hypothesis function by using a cost function.

→ Squared error function or mean squared error

* This type of error function works well for most regression problems.

* This takes an average diff. (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's & the actual output y's.

* To break it apart it is $\frac{1}{2}$ times the diff. b/w the predicted value & the actual value.

* The mean is halved as a convience for the computation of the gradient descent.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)^2$$

* ~~$x_2(t) = x(t) \; s_5(t)$~~

=> Cost function - Intuition 0 - 1

* In visual terms, our training data set is scattered on the xy plane. We're trying to make a straight line (defined by $h_\theta(x)$) which passes through these scattered data points.

* Our objective is to get the best possible line.

The best possible line will be such that the cost (or "error" function) $J(\theta_0, \theta_1)$ will be 0. Thus, as a goal, we should try to minimize the cost function

→ ~~Sketch of~~

→ Sketch of cost function ~~for~~ $J(\theta_1)$ i.e. $\theta_0 = 0$

⇒ Cost function - Intuition -2

→ Contour plot : Graphical technique for representing a 3-D surface by plotting constant z slices (called contours) or contour lines on a 2-D format.

A contour line of a 2 variable function has a constant value at all points of the same line.

* These can be used for visualizing 2 variable cost function.

⇒ Gradient descent

→ So we have our hypothesis function & we have a way of measuring how well it fits into the data (the cost func^n!). Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in!

→ It's an optimization algo that finds the parameter, while minimising the cost function.

→ Workflow :

* 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 & it will give us a direction to move towards.
* We make steps down the cost function in the dir^n with the steepest descent.
* The size of each step is determined by the parameter α, which is called the learning rate.

→ Depending on where one starts on the graph, one could end up at different points.

→ Algorithm :

Repeat until convergence :

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} [J(\theta_0, \theta_1)]$$

where

$j = 0, 1$ represent the feature index number

$:=$ represents the assignment operator

* The algo will work for any number of indices : $\theta_1, \theta_2, \cdots, \theta_n$.

Just change $\frac{\partial}{\partial \theta_j}[J(\theta_0, \theta_1)]$ by $\frac{\partial}{\partial \theta_j}[J(\theta_0, \theta_1, \cdots, \theta_n)]$

* At each iteration j, one should simultaneously update the parameters $\theta_1, \theta_2, \cdots, \theta_n$.

Note: Updating a specific parameter prior to calculating another one on the j^{th} iteration would yield to a wrong implementation.

* For 2 variable J, the correct implementation will be:

$$\text{temp}0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0}[J(\theta_0, \theta_1)]$$

$$\text{temp}1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1}[J(\theta_0, \theta_1)]$$

$$\theta_0 := \text{temp}0$$

$$\theta_1 := \text{temp}1$$

* This algorithm can be used to minimize ANY cost function (irrespective of whether its concerned with regression or not)

⇒ Gradient descent intuition

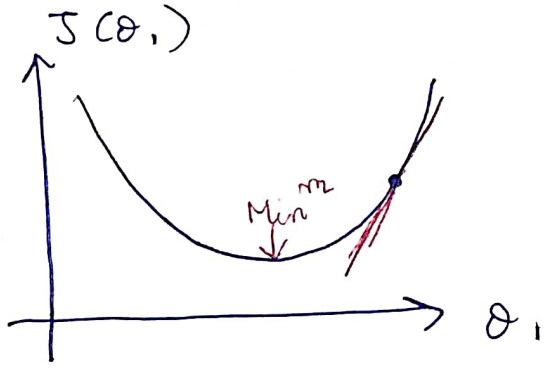
* Suppose we use only 1 parameter θ_1 is plotted its cost function to implement gradient descent.

Our formula for a single parameter was :

Repeat until convergence :

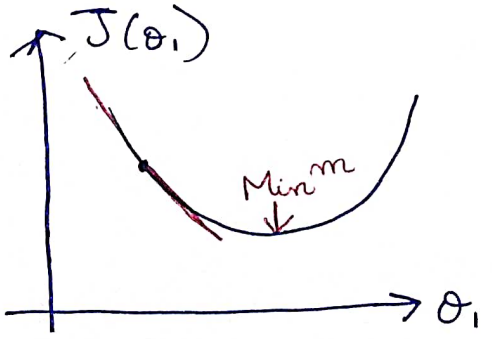
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} [J(\theta_1)]$$

* Regardless of the sign of $\frac{d}{d\theta_1} J(\theta_1)$, θ_1 eventually converges to its minimum value.



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

$$\Rightarrow \theta_1 := \theta_1 - \alpha \text{ (+ve number)}$$



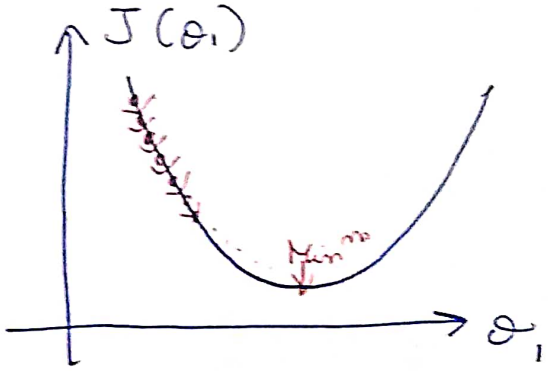
$$\theta_1 := \theta_1 - \alpha \text{ (-ve number)}$$

* We should ajust our parameter α to ensure that the gradient descent algo converges in a reasonable time.

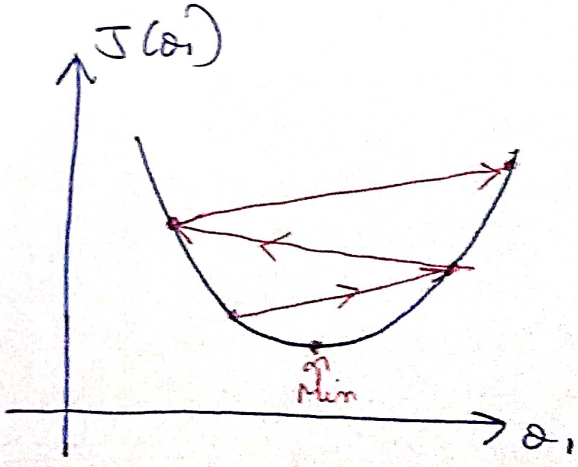
Failure to converge or too much time to obtain the min^m value imply that our step size is wrong.

→ Illustration : $\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} [J(\theta_1)]$

* If α is too small, gradient descent can be slow.



* If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



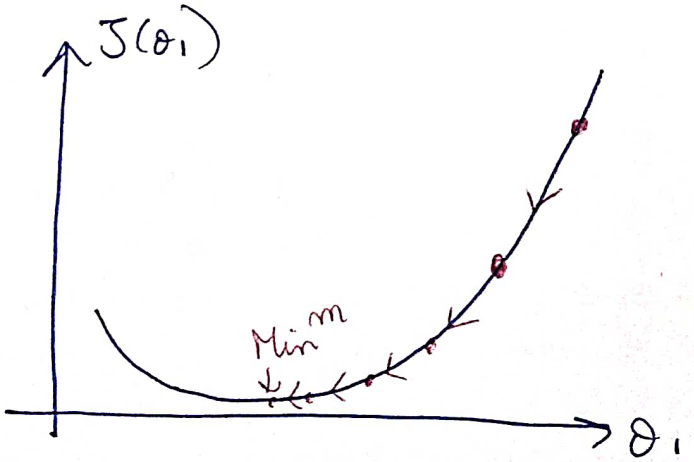
⇒ How does gradient descent converge with a fixed step size α

* The intuition behind the convergence is that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ approaches 0 as we approach the bottom of our convex function.

* At the ~~bottom~~ minimum, the derivative will always be 0 & thus we get:

$$\theta_1 := \theta_1 - \alpha * 0$$

* As we approach a local minimum, gradient descent will automatically take smaller ~~steps~~. So, no need to decrease α over time.



⇒ Gradient descent for linear regression

* When specifically applied to the case of linear regression, a new form of the gradient descent eqⁿ can be derived.

We can substitute our actual cost funcⁿ & our actual hypothesis funcⁿ & modify the eqⁿ to :

repeat until convergence : {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

→ m = size of the training set

$$\rightarrow J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} = \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} [h_\theta(x_i) - y_i]^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} 2 (h_\theta(x_i) - y_i) \frac{\partial}{\partial \theta_j} [\theta_0 + \theta_1 x_i - y_i]$$

$$= \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \left[x_i \frac{\partial(\theta_1)}{\partial \theta_j} \right]$$

$$= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \left[\frac{\partial \theta_0}{\partial \theta_j} + x_i \frac{\partial \theta_1}{\partial \theta_j} - 0 \right]$$

$$\Rightarrow \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) [1 + 0 + 0]$$

$$\& \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) [0 + x_i \frac{\partial \theta_1}{\partial \theta_1} - 0]$$

* As this method of calculating gradient descent looks at ~~the~~ every ~~entire~~ example in the entire training set on every step, is is called batch gradient descent.

* As J in linear regression is a convex quadratic function, gradient descent will always converge to the global minimum (assuming min α not too large)