INTRODUCTION

Machine Learning gives computer the ability to learn without being explicitly programmed. A computer learns from examples, not by rules. A computer is said to learn when its performance P with respect to some task T improves with its experience E. Types of Machine Learning :

- **Supervised Learning** - In the given set of data there will be relationship between input and output.
    - Regression : Maps input variables to some continuous functions and predicts result within a continuous output. For example, predict price of a house using a data set of house sizes and their prices.
    - Classification : Maps input variables to discrete output values and predicts result within a discrete output. For example, predict if tumor is benign or malignant using the data set of patients with tumor (benign or malignant).
- **Unsupervised Learning** - Algorithm will automatically finds the structure in the data set and clusters the individuals into those types. It's also known as Clustering. For example, Google automatically clusters news stories that are all about the same topic and they get displayed together.

LINEAR REGRESSION

For a data set $[x^i, y^i]$ of m examples, where $i : 1\ to\ m$, hypothesis function (relation between x and y) is given by,

$$h(x) = \theta_0 + \theta_1 x$$

Cost function (error in the model) is $(h(x) - y)^2$ and for all m examples it is, $\Sigma_{i=1}^{m}(h(x) - y)^2$. So cost function is given by,

$$J(\theta) = \frac{1}{2m}\Sigma_{i=1}^{m}(h(x) - y)^2$$

Let $\theta_0 = 0$, so $h(x) = \theta_1 x$. For different values of θ_1 we get different straight lines. When we plot $J(\theta_1)$ vs θ_1 we observe that $J(\theta_1)$ is minimum at $\theta_1 = 1$. So for $h(x) = \theta_1 x, \theta_1 = 1$ is the best possible fit. When there's only one variable θ then $J(\theta)$ is like a bow shaped plot. If there are two variables then $J(\theta)$ will be a 3D bow shaped plot. We'll use contour plot for this.

GRADIENT DESCENT

It is an algorithm used to minimize cost function. Start with some values of θ_0 and θ_1 and keep changing them to reduce $J(\theta)$ until we hopefully end up at minimum. Gradient is actually the derivative of cost function. Plot $J(\theta)$ and take derivative (slope) at different points. When $J(\theta)$ is minimum, it's derivative is also minimum or zero.

> $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ for $j = 0$ and $j = 1$
> $\theta_0 = \theta_0 - \alpha[\frac{1}{m} \Sigma_{i=1}^{m} (h(x^i) - y^i)]$
> $\theta_1 = \theta_1 - \alpha[\frac{1}{m} \Sigma_{i=1}^{m} (h(x^i) - y^i)x^i]$

α is learning rate that is the size of steps we take from starting point to reach minimum $J(\theta)$. If α is too small then gradient descent will be slow as it takes smaller steps to converge to $J(\theta)_{min}$ and if α is too large then it can overshoot the minimum and fail to converge. We need to update θ_0 and θ_1 simultaneously until convergence.

Let $J(\theta)$ be a function of only one variable θ_1, so gradient descent becomes, $\theta_1 = \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$.
For point 'A' derivative is positive, so $\theta_1 = \theta_1 - \alpha$ * (positive value). θ_1 is decreased, so move towards left to get closer to $J(\theta)_{min}$. For point 'B' derivative is negative, so $\theta_1 = \theta_1 - \alpha$ * (negative value). θ_1 is increased, so move towards right to get closer to $J(\theta)_{min}$. If θ_1 is already at minimum then gradient descent leaves it unchanged, $\theta_1 = \theta_1 - \alpha(0) = \theta_1$.

As we approach local minimum derivative decreases, so even though learning rate is fixed, algorithm takes smaller steps. In gradient descent algorithm we end up computing sum over m training examples, so it's also called as Batch Gradient Descent Algorithm.

Gradient descent is said to be working correctly if $J(\theta)$ decreases after every iteration. If after some iterations $J(\theta)$ is not decreasing much and if the curve has flattened out then it means gradient descent has converged. If plot of $J(\theta)$ vs number of iterations is going down and getting flattened at some point then gradient descent is working fine. If the plot is going up then use smaller values of α and if the plot is going down very slowly then increase α. `Fig 3`

Automatic Convergence Test is an algorithm that tells if the gradient descent is converging. It declares convergence if $J(\theta)$ decreases by less than threshold in one iteration. But finding that threshold is difficult, so we don't use this method.

MATRIX/VECTOR REPRESENTATION

When there are multiple variables, $h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_n x_n$
Let $\theta = [\theta_0 \theta_1 \theta_2 ... \theta_n]$ and $X = [x_0 x_1 x_2 ... x_n]$ where $x_0 = 1$, so hypothesis function is given by,

> $h(x) = \theta^T X$
> $\theta_j = \theta_j - \alpha[\frac{1}{m}\Sigma_{i=1}^{m}(h(x^i) - y^i)x_j^i]$ for $j = 0, 1, 2...n$

FEATURE SCALING AND MEAN NORMALIZATION

When the input variables are having different ranges then contour plot (ellipse) will be steeper or skinny. So θ will descend slowly or oscillate inefficiently down to the optimum value. So we make the variables in the same range.
For example, let x_1: size of houses (1 to 1200 ftsqr) and x_2: number of bedrooms (1 to 5). When we divide features by their maximum values then we get all features in the same range (less than 1) i.e., $\frac{x_1}{1200}$: $\frac{x_2}{5}$. This is called as Feature scaling.
In addition to this we'll subtract average value of examples from each example in order to have approximately zero mean i.e., $x^i = x^i - \mu$. This is called as Mean normalization.

> $x^i = \frac{x^i - \mu}{x^i \ range}$ where μ : mean or average of x^i

For example, say data set consists of age of houses that ranges between 30 and 50 years with average age of 38 years. Using feature scaling and mean normalization, features become $x^i = \frac{age \ of \ house - 38}{50 - 30}$

Instead of blindly using the given features, based on the insight we can make new features to get a better model. Say $h(x) = \theta_0 + \theta_1(width) + \theta_2(height)$. We can create a new feature $area = (width) * (height)$, so $h(x) = \theta_0 + \theta_1(area)$ gives a better model.

For training sets that can't be separated by straight lines, we can use quadratic or cubic equations like $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ and so on. This is called Polynomial Regression.

NORMAL EQUATION

Taking derivative of cost function, setting it to zero and solving for variables is a long approach.
$J(\theta) = a\theta^2 + b\theta + c$
$\frac{d}{d\theta}J(\theta) = a2\theta + b = 0$ which solves for $\theta = \frac{-b}{2a}$.

Normal Equation is a method used to solve for θ analytically. Rather than needing to run the iterative algorithm we can just solve for optimal θ all at one go.

Say there are m training examples with n features.
$[(x^1, y^1), (x^2, y^2)...(x^m, y^m)]$ m examples
$x^1 = [x_0^1, x_1^1, x_2^1 \ ... \ x_n^1]$ n features in each example
$X = [x^1 \ x^2 ... \ x^m]$ is called as design matrix and $y = [y^1 \ y^2 ... \ y^n]$

> $\theta = (X^T X)^{-1} X^T y$ where $\theta \in R^{n+1}$ and $X \in R^{n+1}$

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iterations, so makes it slower	No need to iterate
Works well even if n is large	Need to compute $(X^T X)^{-1}$. It will be slow if n is large. It roughly costs $O(n^3)$

Causes for $X^T X$ to be non-invertible :

- Redundant features - Linearly dependent. For example, x_1 : size in ftsqr and x_2 : size in mtsqr. Remove one of the features.
- Too many features - $m \leq n$. Remove some of the features or use regularization.
