

SUMMER INTERNSHIP REPORT

ON

**MACHINE LEARNING**

UNDERTAKEN AT

**STANFORD UNIVERSITY THROUGH COURSERA**

*Submitted in partial fulfillment of requirements*

*For The Award Of Degree Of*

***Bachelor Of Technology In Computer Science And Engineering***

To

**CENTRAL UNIVERSITY OF HARYANA**

***Under the Guidance of Submitted By***

**Prof. Andrew Ng Janmaijai Rana**

**CEO/Founder Landing Ai BTech (Computer Science)**

**Co-founder Coursera Roll No:- 11485**

**Adjunct Professor Stanford University**

**Founding Lead of Google Brain**

SESSION 2017-2021

# CERTIFICATE

A picture containing screenshot

Description automatically generated

A picture containing bird

Description automatically generated

# DECLARATION BY STUDENT

Date: July 30, 2020

I do hereby declare

(a). That the internship report on **MACHINE LEARNING submitted by me to CENTRAL UNIVERSITY OF HARYANA in partial fulfillment for the award of the degree of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** is a presentation of my original work and every effort is made to indicate this clearly. The work was carried out by me under the guidance of Prof. Andrew Ng, Adjunct Professor at Stanford University.

(b). That the work conforms to the guidelines for presentation and style set out in the relevant documentation.

Name :- Janmaijai Rana

Roll No :- 11485

B.Tech ( CSE )

# ACKNOWLEDGEMENT

The internship opportunity I had with **Stanford University** through **Coursera** was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it during the times when the world is suffering from COVID-19 pandemic.

Bearing in mind previous I am using this opportunity to express my deepest gratitude and special thanks to the **Prof. Andrew Ng** of **Stanford University** who have taught the **Machine Learning** in such a way that is easily understandable and applicable in real life.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to **Dr. Rakesh Kumar** [ **Associate Professor ( CSE )** ], **Mr**. **Vishal Passricha**. **[** **Assistant Professor ( CSE )** for their careful and precious guidance which were extremely valuable for my study both theoretically and practically.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future,  
**Sincerely,**

**JANMAIJAI RANA**

**ROLL NO :- 11485**

**DATE :- 30 JULY 2020**

# ABOUT COURSERA

Coursera was founded by Daphne Koller and Andrew Ng with a vision of providing life-transforming learning experiences to anyone, anywhere. It is now a leading online learning platform for higher education, where 66 million learners from around the world come to learn skills of the future. More than 200 of the world’s top universities and industry educators partner with Coursera to offer courses, Specializations, certificates, and degree programs. 2,500 companies trust the company’s enterprise platform [Coursera for Business](https://www.coursera.org/business/) to transform their talent. [Coursera for Government](https://www.coursera.org/government) equips government employees and citizens with in-demand skills to build a competitive workforce. [Coursera for Campus](https://www.coursera.org/campus/) empowers any university to offer high-quality, job-relevant online education to students, alumni, faculty, and staff. Coursera is backed by leading investors that include Kleiner Perkins, New Enterprise Associates, Learn Capital, and SEEK Group.

Coursera courses last approximately four to twelve weeks, with one to two hours of video lectures a week. These courses provide quizzes, weekly exercises, peer-graded assignments, an optional Honors assignment and sometimes a final project or exam.[[17]](https://en.wikipedia.org/wiki/Coursera#cite_note-fastcompany1-17) Courses are also provided on-demand, in which case users can take their time in completing the course with all of the material available at once. As of May 2015 Coursera offered 104 on-demand courses it also provides guided projects which are short 2-3 hour projects that can be done and it is very useful for college going students .

As of 2017 Coursera offers full master's degrees. They first started with Master's in Innovation and Entrepreneurship (OMIE) from [HEC Paris](https://en.wikipedia.org/wiki/HEC_Paris) and Master's of Accounting (iMSA) from the [University of Illinois](https://en.wikipedia.org/wiki/University_of_Illinois), but have moved on to offer Master of Computer Science in Data Science and Master of Business Administration (iMBA), both from University of Illinois.[[18]](https://en.wikipedia.org/wiki/Coursera#cite_note-18) , Also as part of MBA; there are some courses are offered separately and will be included in the curriculum of MBA when being enrolled in like digital marketing courses.

TABLE OF CONTENTS

[WEEK :- 01 8](#_Toc47134918)

[[ PART 1 ] INTRODUCTION 8](#_Toc47134919)

[1.1.1 what is machine learning 8](#_Toc47134920)

[1.1.2 Supervised learning 9](#_Toc47134921)

[1.1.3 Unsupervised learning 10](#_Toc47134922)

[WEEK :- 01 11](#_Toc47134923)

[[ PART 2 ] LINEAR REGRESSION WITH ONE VARIABLE 11](#_Toc47134924)

[1.2.1 Linear Regression 11](#_Toc47134925)

[1.2.2 Linear regression Cost Function 12](#_Toc47134926)

[1.2.3 Gradient descent algorithm 13](#_Toc47134927)

[1.2.4 Linear regression with gradient descent 15](#_Toc47134928)

[WEEK :- 02 17](#_Toc47134929)

[LINEAR REGRESSION WITH MULTIPLE VARIABLE 17](#_Toc47134930)

[2.1 Linear regression with multiple features 17](#_Toc47134931)

[2.2 Gradient descent for multiple variables 18](#_Toc47134932)

[Week :- 03 [ PART 1 ] 20](#_Toc47134933)

[LOGISTIC REGRESSION 20](#_Toc47134934)

[3.1.1 Classification 20](#_Toc47134935)

[3.1.2 Hypothesis representation 21](#_Toc47134936)

[3.1.3 Simplified cost function and gradient descent 22](#_Toc47134937)

[3.1.4 How to minimize the logistic regression cost function 24](#_Toc47134938)

[WEEK :- 03 [ PART 2 ] 24](#_Toc47134939)

[REGULARIZATION 24](#_Toc47134940)

[3.2.1 Cost function optimization for regularization 24](#_Toc47134941)

[3.2.2 Regularized linear regression 26](#_Toc47134942)

[WEEK :- 04 27](#_Toc47134943)

[Neural Network Representation 27](#_Toc47134944)

[4.1 Model representation 27](#_Toc47134945)

[WEEK :- 05 29](#_Toc47134946)

[NEURAL NETWORKS LEARNING 29](#_Toc47134947)

[5.1 Cost function for neural networks 29](#_Toc47134948)

[5.2 Neural Network Learning Algorithm 30](#_Toc47134949)

[WEEK :- 06 32](#_Toc47134950)

[ADVICE FOR APPLYING MACHINE LEARNING 32](#_Toc47134951)

[6.1 Model selection and training validation test sets 32](#_Toc47134952)

[6.2 Diagnosis - Bias vs. Variance 34](#_Toc47134953)

[WEEK :- 07 35](#_Toc47134954)

[SUPPORT VECTOR MACHINE 35](#_Toc47134955)

[7.1 Support Vector Machine (SVM) - Optimization objective 35](#_Toc47134956)

[7.2 SVM  Cost Function 36](#_Toc47134957)

[WEEK :- 08 36](#_Toc47134958)

[UNSUPERVISED LEARNING 36](#_Toc47134959)

[8.1 Introduction to Unsupervised learning 36](#_Toc47134960)

[8.2 K-means algorithm 37](#_Toc47134961)

[WEEK :-09 39](#_Toc47134962)

[[ PART 1 ] ANOMALY DETECTION 39](#_Toc47134963)

[9.1.1 Anomaly detection algorithm Overview 39](#_Toc47134964)

[9.1.2 Anomaly Detection Algorithm 40](#_Toc47134965)

[WEEK :- 09 41](#_Toc47134966)

[[ PART 2 ] Recommender System And Collaborative Filtering Algorithm 41](#_Toc47134967)

[9.2.1 Collaborative filtering Algorithm 41](#_Toc47134968)

[9.2.2 Algorithm Structure 42](#_Toc47134969)

[WEEK :- 10 43](#_Toc47134970)

[LARGE SCALE MACHINE LEARNING 43](#_Toc47134971)

[10.1 Stochastic Gradient Descent 43](#_Toc47134972)

[10.2 Mini Batch Gradient Descent 45](#_Toc47134973)

[10.3 Mini-batch algorithm 45](#_Toc47134974)

[10.4 Mini-batch gradient descent vs. stochastic gradient descent 45](#_Toc47134975)

[WEEK :- 11 46](#_Toc47134976)

[PHOTO OCR 46](#_Toc47134977)

[11.1 OCR pipeline 46](#_Toc47134978)

[11.2 Sliding window image analysis 46](#_Toc47134979)

[11.3 Character Segmentation 48](#_Toc47134980)

[TOOLS AND TECHNOLOGY USE 49](#_Toc47134981)

[MATLAB 49](#_Toc47134982)

[FUTURE SCOPE OF MACHINE LEARNING 50](#_Toc47134983)

[Skills Required To Become A Machine Learning Engineer 50](#_Toc47134984)

[REFERENCES 51](#_Toc47134985)

## WEEK :- 01

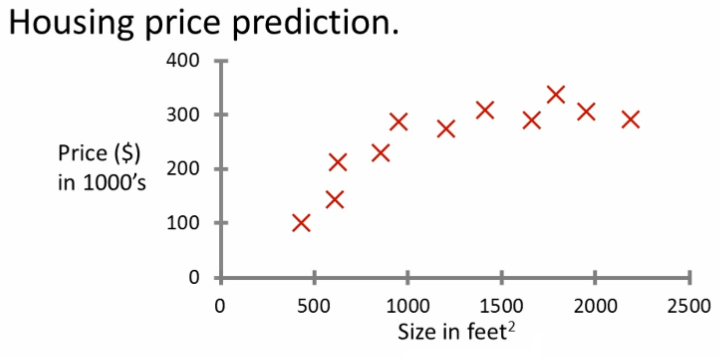
## [ PART 1 ] INTRODUCTION

### 1.1.1 what is machine learning

* Here we...
  + Define what it is
  + When to use it
* Not a well defined definition
  + Couple of examples of how people have tried to define it
* Arthur Samuel (1959)
  + ***Machine learning:* "Field of study that gives computers the ability to learn without being explicitly programmed"**
    - Samuels wrote a checkers playing program
      * Had the program play 10000 games against itself
      * Work out which board positions were good and bad depending on wins/losses
* Tom Michel (1999)
  + ***Well posed learning problem:****"***A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."**
    - The checkers example,
      * E = 10000s games
      * T is playing checkers
      * P if you win or not
* Several types of learning algorithms
  + **Supervised learning**
    - Teach the computer how to do something, then let it use it;s new found knowledge to do it
  + **Unsupervised learning**
    - Let the computer learn how to do something, and use this to determine structure and patterns in data
  + Reinforcement learning
  + Recommender systems
* This course
  + Look at practical advice for applying learning algorithms
  + Learning a set of tools and **how** to apply them

### 1.1.2 Supervised learning

* Probably the most common problem type in machine learning
* Starting with an example
  + How do we predict housing prices
    - Collect data regarding housing prices and how they relate to size in feet

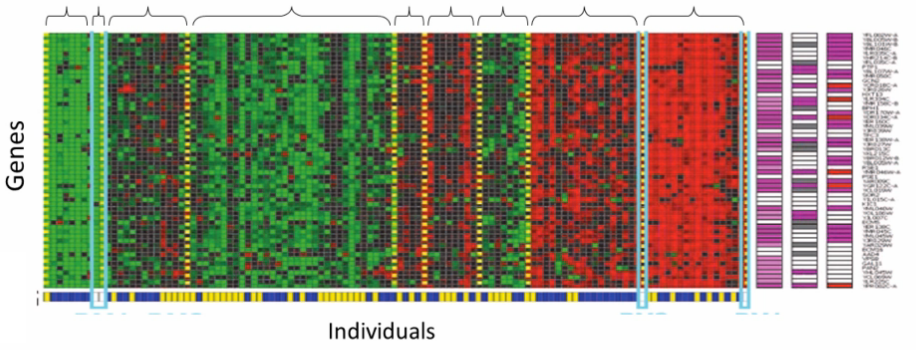


* **Example problem:** "Given this data, a friend has a house 750 square feet - how much can they be expected to get?"
* What approaches can we use to solve this?
  + Straight line through data
    - Maybe $150 000
  + Second order polynomial
    - Maybe $200 000
  + One thing we discuss later - how to chose straight or curved line?
  + Each of these approaches represent a way of doing supervised learning
* *What does this mean?*
  + We gave the algorithm a data set where a "right answer" was provided
  + So we know actual prices for houses
    - The idea is we can learn what makes the price a certain value from the **training data**
    - The algorithm should then produce more right answers based on new training data where we don't know the price already
      * i.e. predict the price
* We also call this a **regression problem**
  + Predict continuous valued output (price)
  + No real discrete delineation
* ***Summary***
  + Supervised learning lets you get the "right" data a
  + Regression problem
  + Classification problem

### 1.1.3 Unsupervised learning

* Second major problem type
* In unsupervised learning, we get unlabeled data
  + Just told - here is a data set, can you structure it
* One way of doing this would be to cluster data into to groups
  + This is a **clustering algorithm**

**Clustering algorithm**

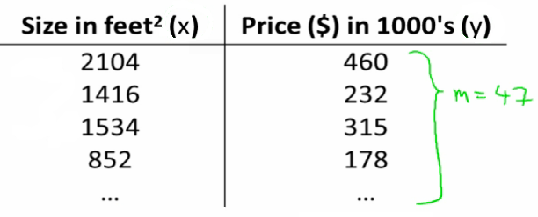
* Example of clustering algorithm
  + Google news
    - Groups news stories into cohesive groups
  + Used in any other problems as well
    - Genomics
    - Microarray data
      * Have a group of individuals
      * On each measure expression of a gene
      * Run algorithm to cluster individuals into types of people
    - Organize computer clusters
      * Identify potential weak spots or distribute workload effectively
    - Social network analysis
      * Customer data
    - Astronomical data analysis
      * Algorithms give amazing results
* Basically
  + Can you automatically generate structure
  + Because we don't give it the answer, it's unsupervised learning

## WEEK :- 01

## [ PART 2 ] LINEAR REGRESSION WITH ONE VARIABLE

### 1.2.1 Linear Regression

* Housing price data example used earlier
  + Supervised learning regression problem
* What do we start with?
  + Training set (this is your data set)
  + Notation (*used throughout the course*)
    - m = number of **training examples**
    - x's = input variables / features
    - y's = output variable "target" variables
      * (x,y) - single training example
      * (xi, yj) - specific example (ith training example)
        + i is an index to training set



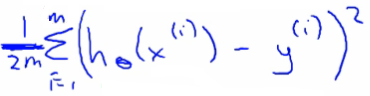
* With our training set defined - how do we used it?
  + Take training set
  + Pass into a learning algorithm
  + Algorithm outputs a function (denoted *h*) (h = **hypothesis**)
    - This function takes an input (e.g. size of new house)
    - Tries to output the estimated value of Y
* How do we represent hypothesis *h*?
  + Going to present h as;
    - hθ(x) = θ0 + θ1x
      * h(x) (shorthand)



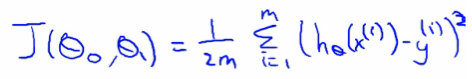
* What does this mean?
  + Means Y is a linear function of x!
  + θi are **parameters**
    - θ0 is zero condition
    - θ1 is gradient
* This kind of function is a linear regression with one variable
  + Also called **univariate linear regression**
* So in summary
  + A hypothesis takes in some variable
  + Uses parameters determined by a learning system
  + Outputs a prediction based on that input

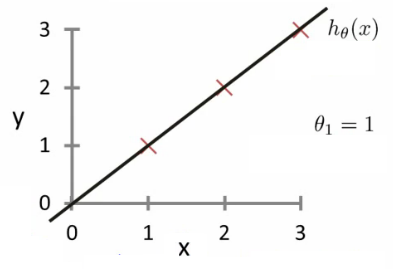
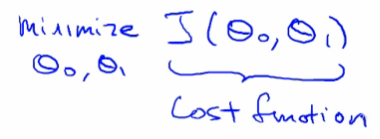
### 1.2.2 Linear regression Cost Function

* A cost function lets us figure out how to fit the best straight line to our data
* Choosing values for θi (parameters)
  + Different values give you different functions
  + If θ0 is 1.5 and θ1 is 0 then we get straight line parallel with X along 1.5 @ y
  + If θ1 is > 0 then we get a positive slope
* Based on our training set we want to generate parameters which make the straight line
  + Chosen these parameters so hθ(x) is close to y for our training examples
    - Basically, uses xs in training set with hθ(x) to give output which is as close to the actual y value as possible
    - Think of hθ(x) as a "y imitator" - it tries to convert the x into y, and considering we already have y we can evaluate how well hθ(x) does this
* To formalize this;
  + We want to want to solve a **minimization problem**
  + Minimize (hθ(x) - y)2
    - i.e. minimize the difference between h(x) and y for each/any/every example
  + Sum this over the training set



* Minimize squared different between predicted house price and actual house price
  + 1/2m
    - 1/m - means we determine the average
    - 1/2m the 2 makes the math a bit easier, and doesn't change the constants we determine at all (i.e. half the smallest value is still the smallest value!)
  + Minimizing θ0/θ1 means we get the values of θ0 and θ1 which find on average the minimal deviation of x from y when we use those parameters in our hypothesis function
* More cleanly, this is a cost function

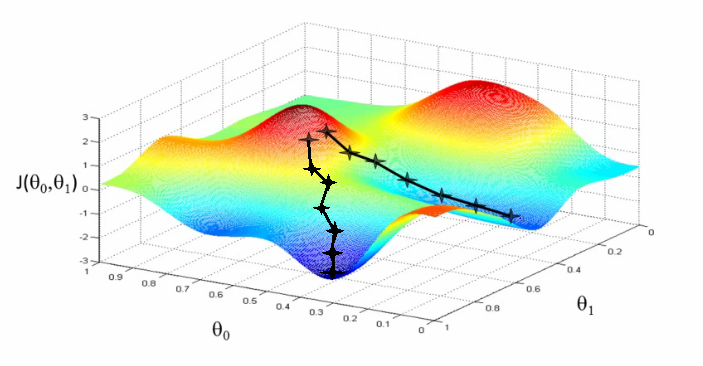


* And we want to minimize this cost function
  + Our cost function is (because of the summartion term) inherently looking at ALL the data in the training set at any time
* **So to recap**
  + **Hypothesis** - is like your prediction machine, throw in an *x* value, get a putative *y* value  
    
  + **Cost** - is a way to, using your training data, determine values for your θ values which make the hypothesis as accurate as possible  
     
    - This cost function is also called the squared error cost function
      * This cost function is reasonable choice for most regression functions
      * Probably most commonly used function
  + In case J(θ0,θ1) is a bit abstract, going into what it does, why it works and how we use it in the coming sections

### 1.2.3 Gradient descent algorithm

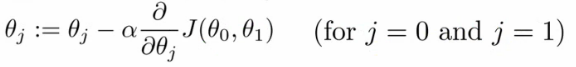
* Minimize cost function J
* Gradient descent
  + Used all over machine learning for minimization
* Start by looking at a general J() function
* Problem
  + We have J(θ0, θ1)
  + We want to get **min J(θ0, θ1)**
* Gradient descent applies to more general functions
  + J(θ0, θ1, θ2 .... θn)
  + min J(θ0, θ1, θ2 .... θn)

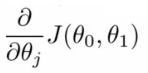
**How does it work?**

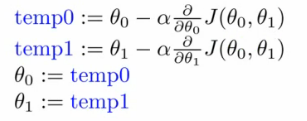
* Start with initial guesses
  + Start at 0,0 (or any other value)
  + Keeping changing θ0 and θ1 a little bit to try and reduce J(θ0,θ1)
* Each time you change the parameters, you select the gradient which reduces J(θ0,θ1) the most possible
* Repeat
* Do so until you converge to a local minimum
* Has an interesting property
  + Where you start can determine which minimum you end up  
    
  + Here we can see one initialization point led to one local minimum
  + The other led to a different one

**A more formal definition**

* Do the following until covergence



* What does this all mean?
  + Update θj by setting it to (θj - α) times the partial derivative of the cost function with respect to θj
* Notation
  + :=
    - Denotes assignment
    - NB a = b is a *truth assertion*
  + α (alpha)
    - Is a number called the **learning rate**
    - Controls how big a step you take
      * If α is big have an aggressive gradient descent
      * If α is small take tiny steps
* Derivative term  
   
  + Not going to talk about it now, derive it later
* There is a subtly about how this gradient descent algorithm is implemented
  + Do this for θ0 and θ1
  + For j = 0 and j = 1 means we **simultaneously**update both
  + How do we do this?
    - Compute the right hand side for both θ0and θ1
      * So we need a temp value
    - Then, update θ0and θ1 at the same time
    - We show this graphically below



* If you implement the non-simultaneous update it's not gradient descent, and will behave weirdly
  + But it might look sort of right - so it's important to remember this!

### 1.2.4 Linear regression with gradient descent

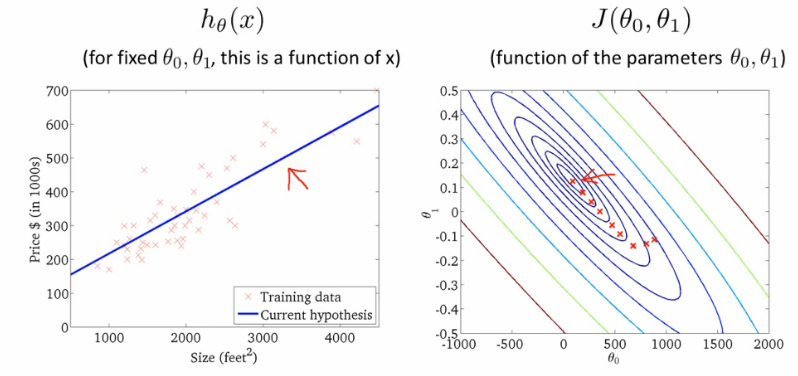
* Apply gradient descent to minimize the squared error cost function J(θ0, θ1)
* Now we have a partial derivative



* So here we're just expanding out the first expression
  + J(θ0, θ1) = 1/2m....
  + hθ(x) = θ0 + θ1\*x
* So we need to determine the derivative for each parameter - i.e.
  + When j = 0
  + When j = 1
* Figure out what this partial derivative is for the θ0 and θ1 case
  + When we derive this expression in terms of j = 0 and j = 1 we get the following



* To check this you need to know multivariate calculus
  + So we can plug these values back into the gradient descent algorithm
* How does it work
  + Risk of meeting different local optimum
  + The linear regression cost function is always a **convex function** - always has a single minimum
    - Bowl shaped
    - One global optima
      * So gradient descent will always converge to global optima
  + In action
    - Initialize values to
      * θ0 = 900
      * θ1 = -0.1



End up at a global minimum

* This is actually **Batch Gradient Descent**
  + Refers to the fact that over each step you look at all the training data
    - Each step compute over m training examples
  + Sometimes non-batch versions exist, which look at small data subsets
    - We'll look at other forms of gradient descent (to use when m is too large) later in the course

# WEEK :- 02

## LINEAR REGRESSION WITH MULTIPLE VARIABLE

### 2.1 Linear regression with multiple features

*New version of linear regression with multiple features*

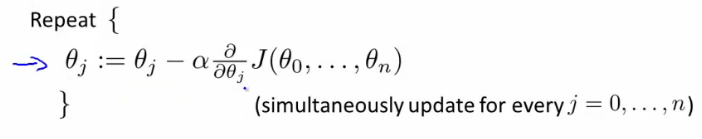
* Multiple variables = multiple features
* In original version we had
  + X = house size, use this to predict
  + y = house price
* If in a new scheme we have more variables (such as number of bedrooms, number floors, age of the home)
  + x1, x2, x3,x4are the four features
    - x1 - size (feet squared)
    - x2 - Number of bedrooms
    - x3 - Number of floors
    - x4 - Age of home (years)
  + y is the output variable (price)
* More notation
  + **n**
    - number of features (n = 4)
  + **m**
    - number of examples (i.e. number of rows in a table)
  + **xi**
    - vector of the input for an example (so a vector of the four parameters for the ithinput example)
    - i is an index into the training set
    - So
      * x is an n-dimensional feature vector
      * x3 is, for example, the 3rd house, and contains the four features associated with that house
  + **xji**
    - The value of feature j in the ith training example
    - So
      * x23is, for example, the number of bedrooms in the third house
* Now we have multiple features
  + What is the form of our hypothesis?
  + Previously our hypothesis took the form;
    - hθ(x) = θ0 + θ1x
      * Here we have two parameters (theta 1 and theta 2) determined by our cost function
      * One variable x
  + Now we have multiple features
    - hθ(x) = θ0 + θ1x1 + θ2x2 + θ3x3 + θ4x4

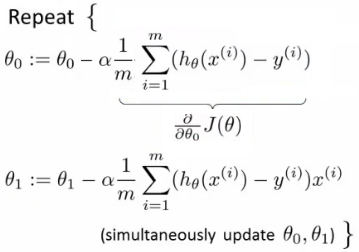
### 2.2 Gradient descent for multiple variables

* Fitting parameters for the hypothesis with gradient descent
  + Parameters are θ0 to θn
  + Instead of thinking about this as n separate values, think about the parameters as a single vector (θ)
    - Where θ is n+1 dimensional
* Our cost function is

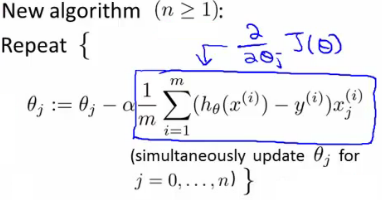


Similarly, instead of thinking of J as a function of the n+1 numbers, J() is just a function of the parameter vector

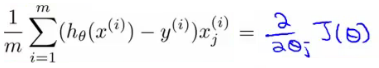
* + J(θ)
* **Gradient descent**
* Once again, this is
  + θj = θj - learning rate (α) times the partial derivative of J(θ) with respect to θJ(...)
  + We do this through a **simultaneous update** of every θj value
* Implementing this algorithm
  + When n = 1



* Above, we have slightly different update rules for θ0 and θ1
  + Actually they're the same, except the end has a previously undefined x0(i) as 1, so wasn't shown
* We now have an almost identical rule for multivariate gradient descent



* What's going on here?
  + We're doing this for each j (0 until n) as a simultaneous update (like when n = 1)
  + So, we re-set θj to
    - θj minus the learning rate (α) times the partial derivative of of the θ vector with respect to θj
    - In non-calculus words, this means that we do
      * Learning rate
      * Times 1/m (makes the maths easier)
      * Times the sum of
        + The hypothesis taking in the variable vector, minus the actual value, times the j-th value in that variable vector for EACH example
  + It's important to remember that



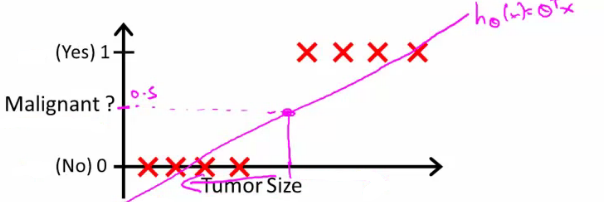
* These algorithm are highly similar

## Week :- 03 [ PART 1 ]

## LOGISTIC REGRESSION

### 3.1.1 Classification

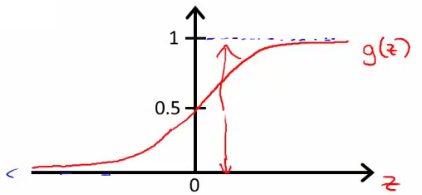
* Where y is a discrete value
  + Develop the logistic regression algorithm to determine what class a new input should fall into
* Classification problems
  + Email -> spam/not spam?
  + Online transactions -> fraudulent?
  + Tumor -> Malignant/benign
* Variable in these problems is Y
  + Y is either 0 or 1
    - 0 = negative class (absence of something)
    - 1 = positive class (presence of something)
* Start with **binary class problems**
  + Later look at multiclass classification problem, although this is just an extension of binary classification
* How do we develop a classification algorithm?
  + Tumour size vs malignancy (0 or 1)
  + We *could* use linear regression
    - Then threshold the classifier output (i.e. anything over some value is yes, else no)
    - In our example below linear regression with thresholding seems to work



* We can see above this does a reasonable job of stratifying the data points into one of two classes
  + But what if we had a single Yes with a very small tumour
  + This would lead to classifying all the existing yeses as nos
* Another issues with linear regression
  + We know Y is 0 or 1
  + Hypothesis can give values large than 1 or less than 0
* So, logistic regression generates a value where is always either 0 or 1
  + Logistic regression is a **classification algorithm** - don't be confused

### 3.1.2 Hypothesis representation

* What function is used to represent our hypothesis in classification
* We want our classifier to output values between 0 and 1
  + When using linear regression we did hθ(x) = (θ*T* x)
  + For classification hypothesis representation we do hθ(x) = g((θ*T* x))
    - Where we define g(z)
      * z is a real number
    - g(z) = 1/(1 + e*-z*)
      * This is the **sigmoid function**, or the **logistic function**
    - If we combine these equations we can write out the hypothesis as  
      
* What does the sigmoid function look like
* Crosses 0.5 at the origin, then flattens out]
  + Asymptotes at 0 and 1



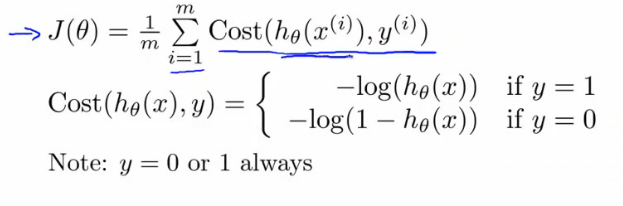
* Given this we need to fit θ to our data

**Interpreting hypothesis output**

* When our hypothesis (hθ(x)) outputs a number, we treat that value as the estimated probability that y=1 on input x
  + Example
    - If X is a feature vector with x0 = 1 (as always) and x1 = tumourSize
    - hθ(x) = 0.7
      * Tells a patient they have a 70% chance of a tumor being malignant
  + We can write this using the following notation
    - hθ(x) = P(y=1|x ; θ)
  + What does this mean?
    - Probability that y=1, given x, parameterized by θ
* Since this is a binary classification task we know y = 0 or 1
  + So the following must be true
    - P(y=1|x ; θ) + P(y=0|x ; θ) = 1
    - P(y=0|x ; θ) = 1 - P(y=1|x ; θ)

### 3.1.3 Simplified cost function and gradient descent

* Define a simpler way to write the cost function and apply gradient descent to the logistic regression
  + By the end should be able to implement a fully functional logistic regression function
* Logistic regression cost function is as follows

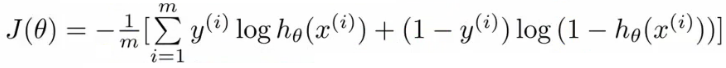


* This is the cost for a single example
  + For binary classification problems y is always 0 or 1
    - Because of this, we can have a simpler way to write the cost function
      * Rather than writing cost function on two lines/two cases
      * Can compress them into one equation - more efficient
  + Can write cost function is
    - **cost(hθ,(x),y) = -ylog( hθ(x) ) - (1-y)log( 1- hθ(x) )**
      * This equation is a more compact of the two cases above
  + We know that there are only two possible cases
    - y = 1
      * Then our equation simplifies to
        + -log(hθ(x)) - (0)log(1 - hθ(x))

-log(hθ(x))

Which is what we had before when y = 1

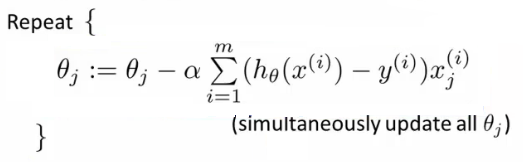
* + - y = 0
      * Then our equation simplifies to  
        + -(0)log(hθ(x)) - (1)log(1 - hθ(x))
        + = -log(1- hθ(x))
        + Which is what we had before when y = 0
    - Clever!
* So, in summary, our cost function for the θ parameters can be defined as



* Why do we chose this function when other cost functions exist?
  + This cost function can be derived from statistics using the principle of **maximum likelihood estimation**
    - Note this does mean there's an underlying Gaussian assumption relating to the distribution of features
  + Also has the nice property that it's convex
* To fit parameters θ:
  + Find parameters θ which minimize J(θ)
  + This means we have a set of parameters to use in our model for future predictions
* Then, if we're given some new example with set of features x, we can take the θ which we generated, and output our prediction using  
          
  + This result is
    - p(y=1 | x ; θ)
      * Probability y = 1, given x, parameterized by θ

### 3.1.4 How to minimize the logistic regression cost function

* Now we need to figure out how to minimize J(θ)
  + Use gradient descent as before
  + Repeatedly update each parameter using a learning rate



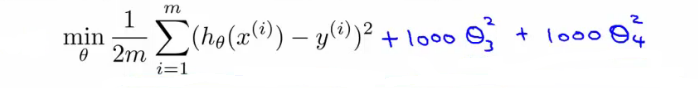
* If you had *n*features, you would have an n+1 column vector for θ
* This equation is the same as the linear regression rule
  + The only difference is that our definition for the hypothesis has changed
* Previously, we spoke about how to monitor gradient descent to check it's working
  + Can do the same thing here for logistic regression
* When implementing logistic regression with gradient descent, we have to update all the θ values (θ0 to θn) simultaneously
  + Could use a for loop
  + Better would be a vectorized implementation
* Feature scaling for gradient descent for logistic regression also applies here

## WEEK :-03 [ PART 2 ]

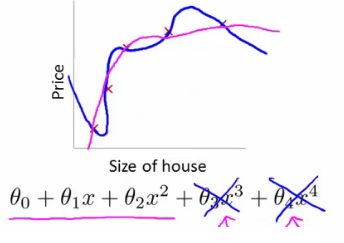
## REGULARIZATION

### 3.2.1 Cost function optimization for regularization

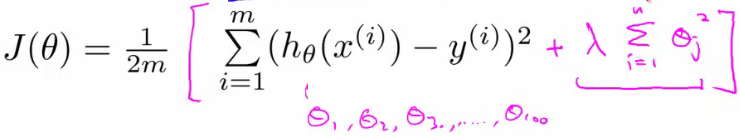
* Penalize and make some of the θ parameters really small
  + e.g. here θ3 and θ4



* The addition in blue is a modification of our cost function to help penalize θ3 and θ4
  + So here we end up with θ3 and θ4 being close to zero (because the constants are massive)
  + So we're basically left with a quadratic function



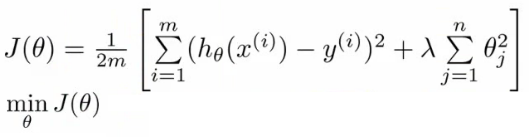
* In this example, we penalized two of the parameter values
  + More generally, regularization is as follows
* Regularization
  + Small values for parameters corresponds to a simpler hypothesis (you effectively get rid of some of the terms)
  + A simpler hypothesis is less prone to overfitting
* Another example
  + Have 100 features x1, x2, ..., x100
  + Unlike the polynomial example, we don't know what are the high order terms
    - How do we pick the ones to pick to shrink?
  + With regularization, take cost function and modify it to shrink all the parameters
    - Add a term at the end
      * This regularization term shrinks every parameter
      * By convention you don't penalize θ0 - minimization is from θ1 onwards



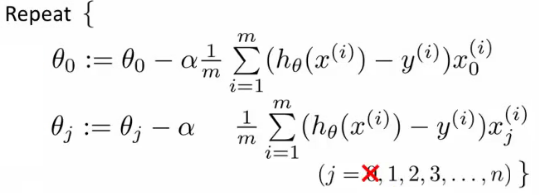
* In practice, if you include θ0 has little impact
* **λ**is the **regularization parameter**
  + Controls a trade off between our two goals
    - 1) Want to fit the training set well
    - 2) Want to keep parameters small
* With our example, using the **regularized objective** (i.e. the cost function with the regularization term) you get a much smoother curve which fits the data and gives a much better hypothesis
  + If **λ** is very large we end up penalizing ALL the parameters (θ1, θ2 etc.) so all the parameters end up being close to zero
    - If this happens, it's like we got rid of all the terms in the hypothesis
      * This results here is then underfitting
    - So this hypothesis is too biased because of the absence of any parameters (effectively)
* So, **λ**should be chosen carefully - not too big...
  + We look at some automatic ways to select **λ**later in the course

### 3.2.2 Regularized linear regression

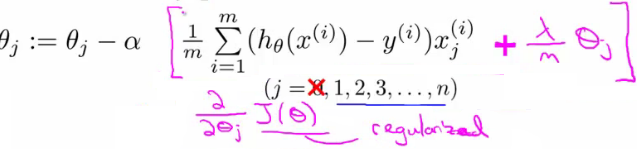
* Previously, we looked at two algorithms for linear regression
  + Gradient descent
  + Normal equation
* Our linear regression with regularization is shown below



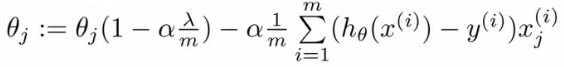
* Previously, gradient descent would repeatedly update the parameters θj, where j = 0,1,2...n simultaneously
  + Shown below



* We've got the θ0 update here shown explicitly
  + This is because for regularization we don't penalize θ0so treat it slightly differently
* How do we regularize these two rules?
  + Take the term and add λ/m \* θj
    - Sum for every θ (i.e. j = 0 to n)
  + This gives regularization for gradient descent
* We can show using calculus that the equation given below is the partial derivative of the regularized J(θ)



* The update for θj
  + θj gets updated to
    - θj- α \* [a big term which also depends on θj]
* So if you group the θjterms together



* The term   
      
  + Is going to be a number less than 1 usually
  + Usually learning rate is small and m is large
    - So this typically evaluates to (1 - a small number)
    - So the term is often around 0.99 to 0.95
* This in effect means θjgets multiplied by 0.99
  + Means the squared norm of θja little smaller
  + The second term is exactly the same as the original gradient descent

# Week :- 04

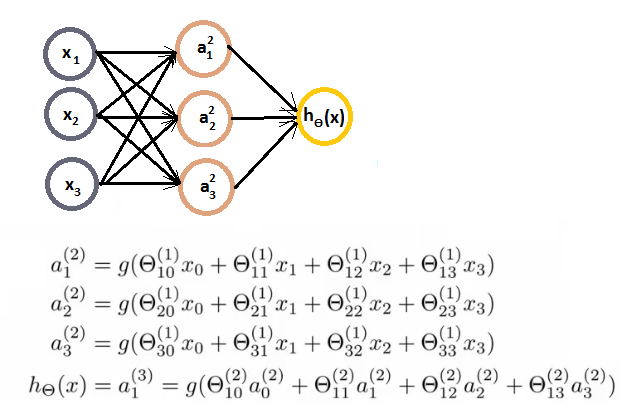
## Neural Network Representation

### 4.1 Model representation

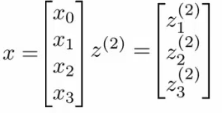
*Here we'll look at how to carry out the computation efficiently through a vectorized implementation. We'll also consider*

*why NNs are good and how we can use them to learn complex non-linear things*

* Below is our original problem from before
  + Sequence of steps to compute output of hypothesis are the equations below



* Define some additional terms
  + z12 = Ɵ101x0+ Ɵ111x1+ Ɵ121x2+ Ɵ131x3
  + Which means that
    - a12 = g(z12)
  + NB, superscript numbers are the layer associated
* Similarly, we define the others as
  + z22 and z32
  + These values are just a linear combination of the values
* If we look at the block we just redefined
  + We can vectorize the neural network computation
  + So lets define
    - x as the feature vector x
    - z2 as the vector of z values from the second layer



* z2 is a 3x1 vector
* We can vectorize the computation of the neural network as as follows in two steps
  + z2 = Ɵ(1)x
    - i.e. Ɵ(1) is the matrix defined above
    - x is the feature vector
  + a2 = g(z(2))
    - To be clear, z2 is a 3x1 vecor
    - a2 is also a 3x1 vector
    - g() applies the sigmoid (logistic) function element wise to each member of the z2 vector
* To make the notation with input layer make sense;
  + a1 = x
    - a1 is the activations in the input layer
    - Obviously the "activation" for the input layer is just the input!
  + So we define x as a1 for clarity
    - So
      * a1 is the vector of inputs
      * a2 is the vector of values calculated by the g(z2) function
* Having calculated then z2 vector, we need to calculate a02 for the final hypothesis calculation



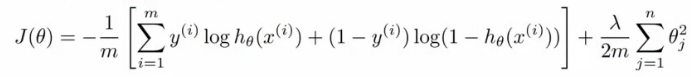
* To take care of the extra bias unit add a02 = 1
  + So add a02 to a2 making it a 4x1 vector
* So,
  + z3 = Ɵ2a2
    - This is the inner term of the above equation
  + hƟ(x) = a3 = g(z3)
* This process is also called **forward propagation**
  + Start off with activations of input unit
    - i.e. the x vector as input
  + Forward propagate and calculate the activation of each layer sequentially
  + This is a vectorized version of this implementation

# WEEK :- 05

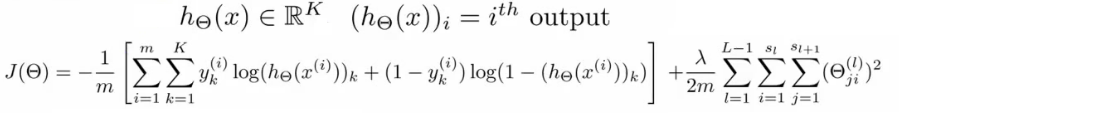
## NEURAL NETWORKS LEARNING

### 5.1 Cost function for neural networks

* The (regularized) logistic regression cost function is as follows;



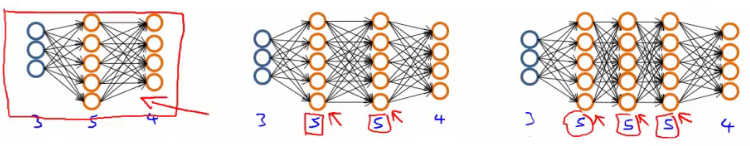
* For neural networks our cost function is a generalization of this equation above, so instead of one output we generate *k* outputs



* Our cost function now outputs a *k* dimensional vector
  + hƟ(x) is a k dimensional vector, so hƟ(x)*i* refers to the ith value in that vector
* Costfunction J(Ɵ) is
  + [-1/m] times a sum of a similar term to which we had for logic regression
  + But now this is also a sum from k = 1 through to K (K is number of output nodes)
    - Summation is a sum over the k output units - i.e. for each of the possible classes
    - So if we had 4 output units then the sum is k = 1 to 4 of the logistic regression over each of the four output units in turn
  + This looks really complicated, but it's not so difficult
    - We don't sum over the bias terms (hence starting at 1 for the summation)
      * Even if you do and end up regularizing the bias term this is not a big problem
    - Is just summation over the terms

### 5.2 Neural Network Learning Algorithm

* **1) - pick a network architecture**
  + Number of
    - **Input units** - number of dimensions x (dimensions of feature vector)
    - **Output units** - number of classes in classification problem
    - **Hidden units**
      * Default might be
        + 1 hidden layer
      * Should probably have
        + Same number of units in each layer
        + Or 1.5-2 x number of input features
      * Normally
        + More hidden units is better
        + But more is more computational expensive
  + We'll discuss architecture more later



* **2) - Training a neural network**
  + **2.1)** Randomly initialize the weights
    - Small values near 0
  + **2.2)**Implement forward propagation to get hƟ(x)i for any xi
  + **2.3)**Implement code to compute the cost function J(Ɵ)
  + **2.4)**Implement back propagation to compute the partial derivatives
  + General implementation below

**for i = 1:m {**

**Forward propagation on (xi, yi) --> get activation (a) terms**

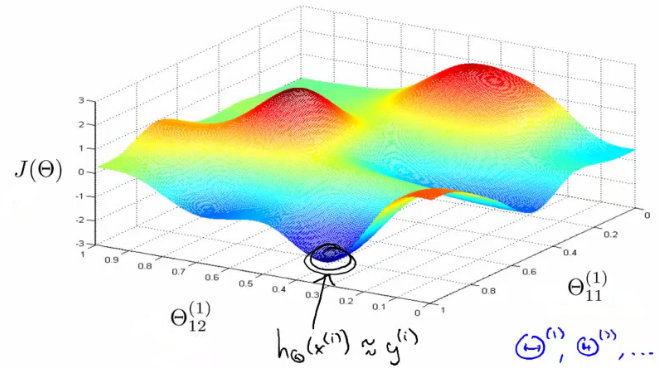
**Back propagation on (xi, yi) --> get delta (δ) terms**

**Compute Δ := Δl + δl+1(al)*T***

**}**

**With this done compute the partial derivative terms**

* + Notes on implementation
    - Usually done with a for loop over training examples (for forward and back propagation)
    - *Can* be done without a for loop, but this is a much more complicated way of doing things
    - Be careful
* **2.5)**Use gradient checking to compare the partial derivatives computed using the above algorithm and numerical estimation of gradient of J(Ɵ)
  + Disable the gradient checking code for when you actually run it
* **2.6)** Use gradient descent or an advanced optimization method with back propagation to try to minimize J(Ɵ) as a function of parameters Ɵ
  + Here J(Ɵ) is non-convex
    - Can be susceptible to local minimum
    - In practice this is not usually a huge problem
    - Can't guarantee programs with find global optimum should find good local optimum at least

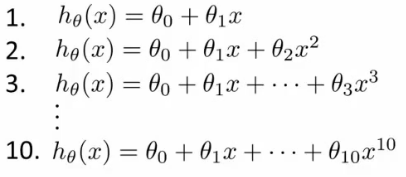
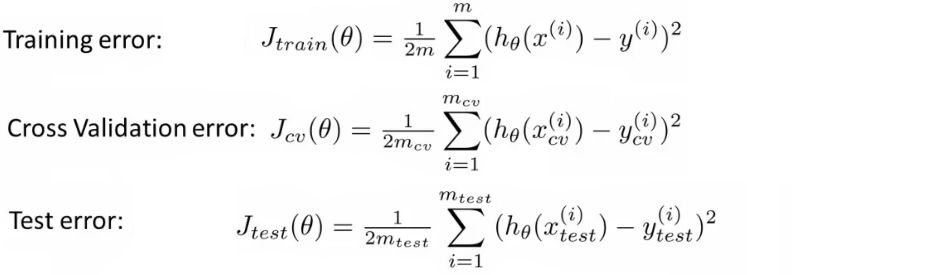


* e.g. above pretending data only has two features to easily display what's going on
  + Our minimum here represents a hypothesis output which is pretty close to y
  + If you took one of the peaks hypothesis is far from y
* Gradient descent will start from some random point and move downhill
  + Back propagation calculates gradient down that hill

# WEEK :- 06

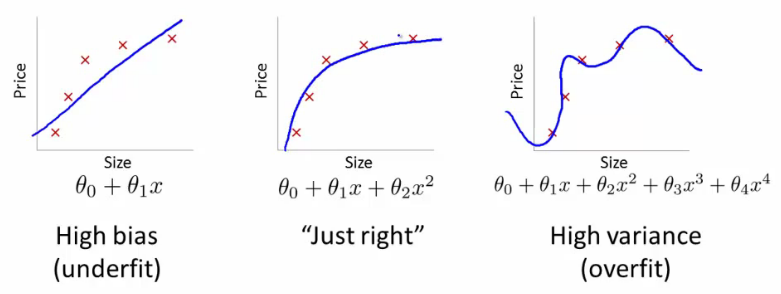
## ADVICE FOR APPLYING MACHINE LEARNING

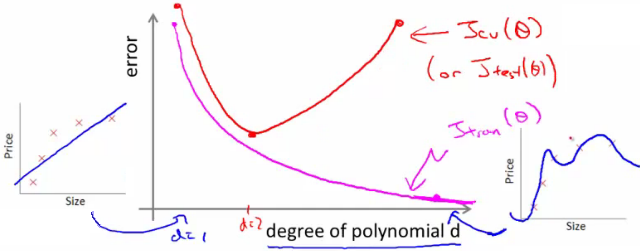
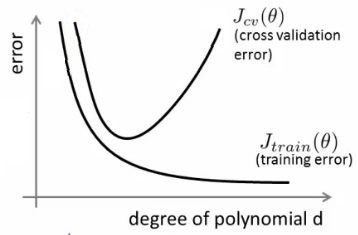
### 6.1 Model selection and training validation test sets

* How to chose regularization parameter or degree of polynomial (**model selection problems**)
* We've already seen the problem of overfitting
  + More generally, this is why training set error is a poor predictor of hypothesis accuracy for new data (generalization)
* Model selection problem
  + Try to chose the degree for a polynomial to fit data  
    
  + d = what degree of polynomial do you want to pick
    - An additional parameter to try and determine your training set
      * d =1 (linear)
      * d=2 (quadratic)
      * ...
      * d=10
    - Chose a model, fit that model and get an estimate of how well you hypothesis will generalize
  + You could
    - Take model 1, minimize with training data which generates a parameter vector θ1 (where d =1)
    - Take mode 2, do the same, get a *different* θ2 (where d = 2)
    - And so on
    - Take these parameters and look at the test set error for each using the previous formula
      * Jtest(θ1)
      * Jtest(θ2)
      * ...
      * Jtest(θ10)
  + You could then
    - See which model has the lowest test set error
  + Say, for example, d=5 is the lowest
    - Now take the d=5 model and say, how well does it generalize?
      * You could use Jtest(θ5)
      * BUT, this is going to be an optimistic estimate of generalization error, because our parameter is fit to that test set (i.e. specifically chose it because the test set error is small)
      * So not a good way to evaluate if it will generalize
  + To address this problem, we do something a bit different for model selection
* Improved model selection  
  + Given a training set instead split into three pieces
    - 1 - **Training set** (60%) - m values
    - 2 - **Cross validation** (CV) set (20%)mcv
    - 3 - **Test set** (20%) mtest
  + As before, we can calculate
    - Training error
    - Cross validation error
    - Test error  
      
  + So
    - Minimize cost function for each of the models as before
    - Test these hypothesis on the cross validation set to generate the cross validation error
    - Pick the hypothesis with the lowest cross validation error
      * e.g. pick θ5
    - Finally
      * Estimate generalization error of model using the test set
* Final note
  + In machine learning as practiced today - many people will select the model using the test set and then check the model is OK for generalization using the test error (which we've said is bad because it gives a bias analysis)
    - With a MASSIVE test set this is maybe OK
  + But considered much better practice to have separate training and validation sets

### 6.2 Diagnosis - Bias vs. Variance

* If you get bad results usually because of one of
  + **High bias** - under fitting problem
  + **High variance** - over fitting problem
* Important to work out which is the problem
  + Knowing which will help let you improve the algorithm
* Bias/variance shown graphically below



* The degree of a model will increase as you move towards overfitting
* Lets define training and cross validation error as before
* Now plot
  + x = degree of polynomial d
  + y = error for both training and cross validation (two lines)
    - CV error and test set error will be very similar   
      
    - This plot helps us understand the error
  + We want to minimize both errors
    - Which is why that d=2 model is the sweet spot
* How do we apply this for diagnostics
  + If cv error is high we're either at the high or the low end of d  
    
  + if d is too small --> this probably corresponds to a high bias problem
  + if d is too large --> this probably corresponds to a high variance problem
* **For the high bias case, we find both cross validation and training error are high**
  + Doesn't fit training data well
  + Doesn't generalize either
* **For high variance, we find the cross validation error is high but training error is low**
  + So we suffer from overfitting (training is low, cross validation is high)
  + i.e. training set fits well
  + But generalizes poorly

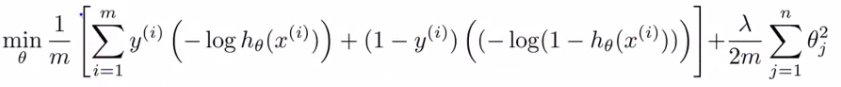
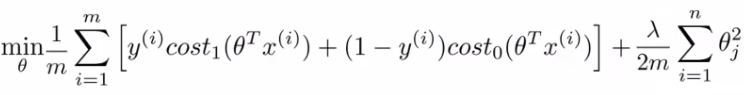
# WEEK :07

## SUPPORT VECTOR MACHINE

### 7.1 Support Vector Machine (SVM) - Optimization objective

* So far, we've seen a range of different algorithms
  + With supervised learning algorithms - performance is pretty similar
    - What matters more often is;
      * The amount of training data
      * Skill of applying algorithms
* One final supervised learning algorithm that is widely used - **support vector machine (SVM)**
  + Compared to both logistic regression and neural networks, a SVM sometimes gives a cleaner way of learning non-linear functions
  + Later in the course we'll do a survey of different supervised learning algorithms

### 7.2 SVM  Cost Function

* As a comparison/reminder we have logistic regression below  
  
  + If this looks unfamiliar its because we previously had the - sign outside the expression
* For the SVM we take our two logistic regression y=1 and y=0 terms described previously and replace with
  + cost1(θ*T* x)
  + cost0(θ*T* x)
* So we get  
  

# WEEK :-08

## UNSUPERVISED LEARNING

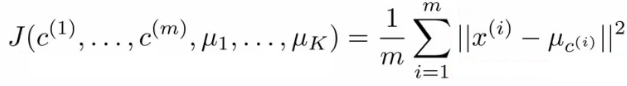
### 8.1 Introduction to Unsupervised learning

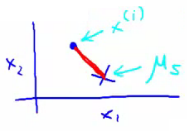
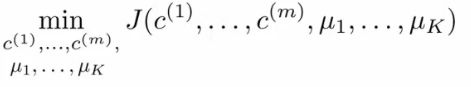
* Talk about **clustering**
  + **Learning from unlabeled data**
* Unsupervised learning
  + Useful to contras with supervised learning
* Compare and contrast
  + Supervised learning
    - Given a set of labels, fit a hypothesis to it
  + Unsupervised learning
    - Try and determining structure in the data
    - Clustering algorithm groups data together based on data features
* What is clustering good for
  + **Market segmentation** - group customers into different market segments
  + **Social network analysis** - Facebook "smartlists"
  + **Organizing computer clusters** and data centers for network layout and location
  + **Astronomical data analysis** - Understanding galaxy formation

### 8.2 K-means algorithm

 While K-means is running we keep track of two sets of variables

* ci is the index of clusters {1,2, ..., K} to which xi is currently assigned
  + i.e. there are *m* ci values, as each example has a ci value, and that value is one the the clusters (i.e. can only be one of K different values)
* μk, is the cluster associated with centroid *k*
  + Locations of cluster centroid k
  + So there are K
  + So these the centroids which exist in the training data space
* μci, is the cluster centroid of the cluster to which example xi has been assigned to
  + This is more for convenience than anything else
    - You could look up that example i is indexed to cluster j (using the c vector), where j is between 1 and K
    - Then look up the value associated with cluster j in the μ vector (i.e. what are the features associated with μj)
    - But instead, for easy description, we have this variable which gets exactly the same value
  + Lets say xias been assigned to cluster 5
    - Means that
      * ci = 5
      * μci, = μ5

 Using this notation we can write the optimization objective;  


* i.e. squared distances between training example xiand the cluster centroid to which xihas been assigned to
  + This is just what we've been doing, as the visual description below shows;  
    
  + The red line here shows the distances between the example xiand the cluster to which that example has been assigned
    - Means that when the example is very close to the cluster, this value is small
    - When the cluster is very far away from the example, the value is large
* This is sometimes called the **distortion** (or **distortion cost function**)
* So we are finding the values which minimizes this function;  
  

 If we consider the k-means algorithm

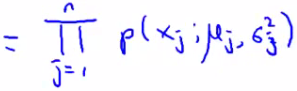
* The **cluster assigned step** is minimizing J(...) with respect to c1, c2 ... ci
  + i.e. find the centroid closest to each example
  + Doesn't change the centroids themselves
* The **move centroid step**
  + We can show this step is choosing the values of μ which minimizes J(...) with respect to μ
* So, we're partitioning the algorithm into two parts
  + First part minimizes the c variables
  + Second part minimizes the J variables

 We can use this knowledge to help debug our K-means algorithm

## WEEK :-09

## [ PART 1 ] ANOMALY DETECTION

### 9.1.1 Anomaly detection algorithm Overview

* Unlabeled training set of m examples
  + Data = {x1, x2, ..., xm}
    - Each example is an n-dimensional vector (i.e. a feature vector)
    - We have n features!
  + Model P(x) from the data set
    - What are high probability features and low probability features
    - x is a vector
    - So model p(x) as
      * = p(x1; μ1 , σ12) \* p(x2; μ2 , σ22) \* ... p(xn; μn , σn2)
    - Multiply the probability of each features by each feature
      * We model each of the features by assuming each feature is distributed according to a Gaussian distribution
      * p(xi; μi , σi2)
        + The probability of feature xi given μi and σi2, using a Gaussian distribution
  + As a side comment
    - Turns out this equation makes an **independence assumption** for the features, although algorithm works if features are independent or not
      * Don't worry too much about this, although if you're features are tightly linked you should be able to do some dimensionality reduction anyway!
  + We can write this chain of multiplication more compactly as follows;  
    
    - Capital PI (Π) is the product of a set of values
  + The problem of estimation this distribution is sometimes call the problem of **density estimation**

### 9.1.2 Anomaly Detection Algorithm

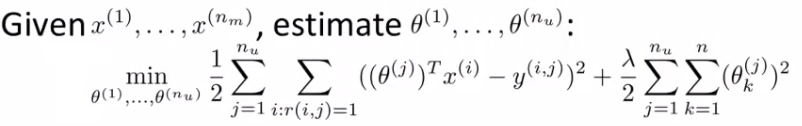
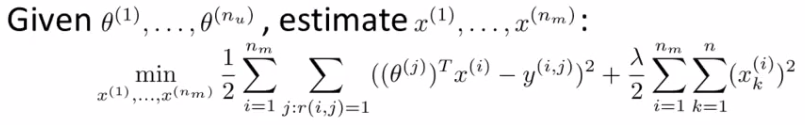


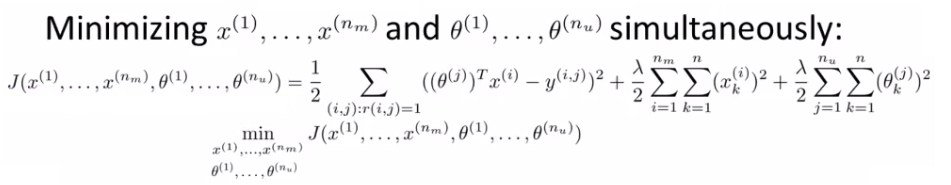
* **1 - Chose features**
  + Try to come up with features which might help identify something anomalous - may be unusually large or small values
  + More generally, chose features which describe the general properties
  + This is nothing unique to anomaly detection - it's just the idea of building a sensible feature vector
* **2 - Fit parameters**
  + Determine parameters for each of your examples μi and σi2
    - Fit is a bit misleading, really should just be "Calculate parameters for 1 to n"
  + So you're calculating standard deviation and mean for each feature
  + You should of course used some vectorized implementation rather than a loop probably
* **3 - compute p(x)**
  + You compute the formula shown (i.e. the formula for the Gaussian probability)
  + If the number is very small, very low chance of it being "normal"

## WEEK :- 09

## [ PART 2 ] Recommender System And Collaborative Filtering Algorithm

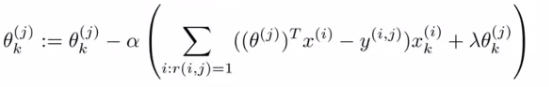
### 9.2.1 Collaborative filtering Algorithm

* Here we combine the ideas from before to build a collaborative filtering algorithm
* Our starting point is as follows
  + If we're given the film's features we can use that to work out the users' preference  
    
  + If we're given the users' preferences we can use them to work out the film's features  
    
* One thing you could do is
  + Randomly initialize parameter
  + Go back and forward
* But there's a more efficient algorithm which can solve θ and x simultaneously
  + Define a new optimization objective which is a function of x and θ



* Understanding this optimization objective
  + The squared error term is the same as the squared error term in the two individual objectives above  
    
    - So it's summing over every movie rated by every users
    - Note the ":" means, "for which"
      * Sum over all pairs (i,j) for which r(i,j) is equal to 1
  + The regularization terms
    - Are simply added to the end from the original two optimization functions
* This newly defined function has the property that
  + If you held x constant and only solved θ then you solve the, "Given x, solve θ" objective above
  + Similarly, if you held θ constant you could solve x
* In order to come up with just one optimization function we treat this function as a function of both film features x and user parameters θ
  + Only difference between this in the back-and-forward approach is that we minimize with respect to both x and θ simultaneously
* When we're learning the features this way
  + Previously had a convention that we have an x0 = 1 term
  + When we're using this kind of approach we have no x0,
    - So now our vectors (both x and θ) are n-dimensional (not n+1)
  + We do this because we are now learning all the features so if the system needs a feature always = 1 then the algorithm can learn one

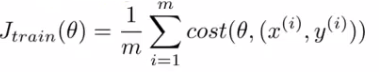
### 9.2.2 Algorithm Structure

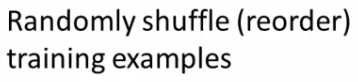
* **1)** Initialize θ1, ..., θnuand x1, ..., xnm to small random values
  + A bit like neural networks - initialize all parameters to small random numbers
* **2)** Minimize cost function (J(x1, ..., xnm, θ1, ...,θnu) using gradient descent
  + We find that the update rules look like this  
      
    
  + Where the top term is the partial derivative of the cost function with respect to xki while the bottom is the partial derivative of the cost function with respect to θki
  + So here we regularize EVERY parameters (no longer x0parameter) so no special case update rule
* **3)** Having minimized the values, given a user (user j) with parameters θ and movie (movie i) with learned features x, we predict a start rating of (θj)*T*xi
  + This is the collaborative filtering algorithm, which should give pretty good predictions for how users like new movies

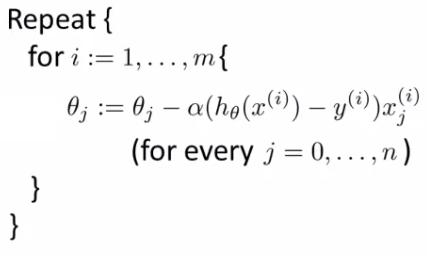
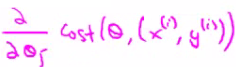
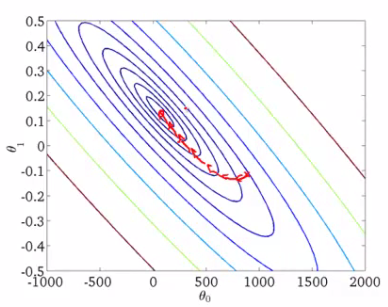
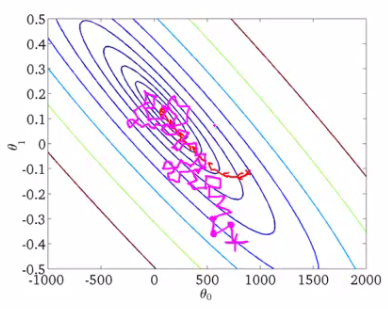
# WEEK :- 10

## LARGE SCALE MACHINE LEARNING

### 10.1 Stochastic Gradient Descent

* Define our cost function slightly differently, as  
  
  + So the function represents the cost of θ with respect to a specific example (xi, yi)
    - And we calculate this value as one half times the squared error on that example
  + Measures how well the hypothesis works on a single example
* The overall cost function can now be re-written in the following form;  
  
  + This is equivalent to the batch gradient descent cost function
* With this slightly modified (but equivalent) view of linear regression we can write out how stochastic gradient descent works

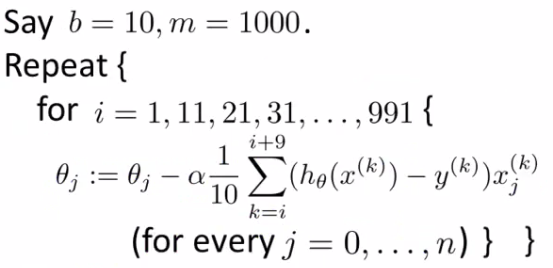
**1) - Randomly shuffle**  


* **2) - Algorithm body**  
  
* *So what's going on here?*
  + The term  
    
  + Is the same as that found in the summation for batch gradient descent
  + It's possible to show that this term is equal to the partial derivative with respect to the parameter θj of the cost(θ, (xi, yi))  
    
* What stochastic gradient descent algorithm is doing is scanning through each example
  + The inner for loop does something like this...
    - Looking at example 1, take a step with respect to the cost of just the 1st training example
      * Having done this, we go on to the second training example
    - Now take a second step in parameter space to try and fit the second training example better
      * Now move onto the third training example
    - And so on...
    - Until it gets to the end of the data
  + We may now repeat this who procedure and take multiple passes over the data
* The **randomly shuffling** at the start means we ensure the data is in a random order so we don't bias the movement
  + Randomization should speed up convergence a little bit
* Although stochastic gradient descent is a lot like batch gradient descent, rather than waiting to sum up the gradient terms over all *m*examples, we take just one example and make progress in improving the parameters already
  + Means we update the parameters on EVERY step through data, instead of at the end of each loop through all the data
* What does the algorithm do to the parameters?
  + As we saw, batch gradient descent does something like this to get to a global minimum  
    
  + With stochastic gradient descent every iteration is much faster, but every iteration is flitting a single example  
    
    - What you find is that you "generally" move in the direction of the global minimum, but not always
    - Never actually converges like batch gradient descent does, but ends up wandering around some region close to the global minimum
      * In practice, this isn't a problem - as long as you're close to the minimum that's probably OK

### 10.2 Mini Batch Gradient Descent

* **Mini-batch gradient descent** is an additional approach which can work even faster than stochastic gradient descent
* To summarize our approaches so far
  + Batch gradient descent: Use all *m* examples in each iteration
  + Stochastic gradient descent: Use 1 example in each iteration
  + Mini-batch gradient descent: Use *b* examples in each iteration
    - *b* = mini-batch size
* So just like batch, except we use tiny batches
  + Typical range for *b* = 2-100 (10 maybe)
* For example
  + *b* = 10
  + Get 10 examples from training set
  + Perform gradient descent update using the ten examples

### 10.3 Mini-batch algorithm

****

* We for-loop through *b-*size batches of *m*
* Compared to batch gradient descent this allows us to get through data in a much more efficient way
  + After just *b* examples we begin to improve our parameters
  + Don't have to update parameters after *every* example, and don't have to wait until you cycled through all the data

### 10.4 Mini-batch gradient descent vs. stochastic gradient descent

* Why should we use mini-batch?
  + Allows you to have a vectorized implementation
  + Means implementation is much more efficient
  + Can partially parallelize your computation (i.e. do 10 at once)
* A disadvantage of mini-batch gradient descent is the optimization of the parameter *b*
  + But this is often worth it!
  + To be honest, stochastic gradient descent and batch gradient descent are just specific forms of batch-gradient descent
  + For mini-batch gradient descent, b is somewhere in between 1 and m and you can try to optimize for it!

# WEEK :- 11

## PHOTO OCR

### 11.1 OCR pipeline

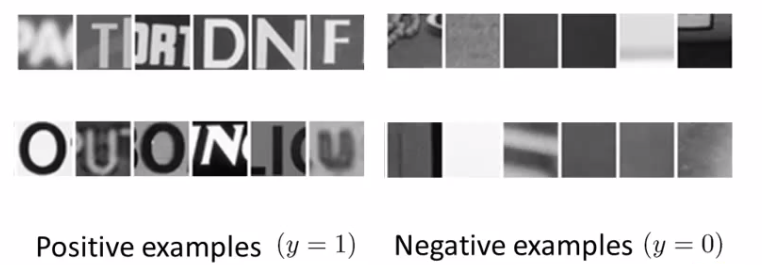
* 1) Look through image and find text
* 2) Do character segmentation
* 3) Do character classification
* 4) *Optional* some may do spell check after this too
  + We're not focussing on such systems though

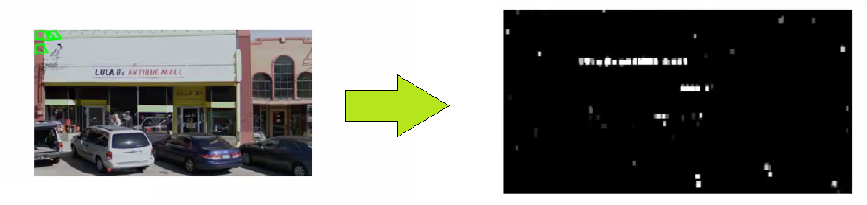


* **Pipelines** are common in machine learning
  + Separate modules which may each be a machine learning component or data processing component
* If you're designing a machine learning system, pipeline design is one of the most important questions
  + Performance of pipeline and each module often has a big impact on the overall performance a problem
  + You would often have different engineers working on each module
    - Offers a natural way to divide up the workload

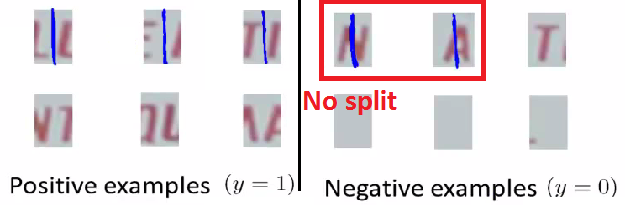
### 11.2 Sliding window image analysis

* How do the individual models work?
* Here focus on a sliding windows classifier
* As mentioned, stage 1 is **text detection**
  + Unusual problem in computer vision - different rectangles (which surround text) may have different aspect ratios (aspect ratio being height : width)
    - Text may be short (few words) or long (many words)
    - Tall or short font
    - Text might be straight on
    - Slanted  
      



* Having trained the classifier we apply it to an image
  + So, run a sliding window classifier at a fixed rectangle size
  + If you do that end up with something like this
  + White region show where text detection system thinks text is
    - Different shades of gray correspond to probability associated with how sure the classifier is the section contains text
      * Black - no text
      * White - text
    - For text detection, we want to draw rectangles around all the regions where there is text in the image
  + Take classifier output and apply an **expansion algorithm**
    - Takes each of white regions and expands it
    - How do we implement this
      * Say, for every pixel, is it within some distance of a white pixel?
      * If yes then colour it white  
         
    - Look at connected white regions in the image above 
    - Draw rectangles around those which make sense as text (i.e. tall thin boxes don't make sense)
  + This example misses a piece of text on the door because the aspect ratio is wrong
    - Very hard to read

### 11.3 Character Segmentation

* Use supervised learning algorithm
* Look in a defined image patch and decide, is there a split between two characters?
  + So, for example, our first training data item below looks like there is such a split
  + Similarly, the negative examples are either empty or hold a full character
* We train a classifier to try and classify between positive and negative examples
  + Run that classifier on the regions detected as containing text in the previous section
* Use a 1-dimensional sliding window to move along text regions
  + Does each window snapshot look like the split between two characters?
    - If yes insert a split
    - If not move on
  + So we have something that looks like this  
    

**Character classification**

* Standard OCR, where you apply standard supervised learning which takes an input and identify which character we decide it is
  + Multi-class characterization problem

# TOOLS AND TECHNOLOGY USE

## MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

* Math and computation
* Algorithm development
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

# FUTURE SCOPE OF MACHINE LEARNING

The scope of Machine Learning is not limited to the investment sector. Rather, it is expanding across all fields such as banking and finance, information technology, media & entertainment, gaming, and the automotive industry. As the Machine Learning scope is very high, there are some of the areas where researchers are working toward revolutionizing the world for the future

* + 1. Automotive Industry
    2. Robotics
    3. Quantum Computing
    4. Computer Vision

# Skills Required To Become A Machine Learning Engineer

There are certain skills that you need to master for becoming a successful Machine Learning Engineer and they are:

* **Programming:** Programming is one of the important aspects for any Machine Learning enthusiast. For Machine Learning, we generally use R and Python languages. We can learn both. However, the scope of Machine Learning with Python is high.
* **Understanding of** **data structures:** The data structure is the core of any software. Thus, it is recommended to have a good grasp of the concepts of data structure.
* **Mathematics:** We cannot perform computation without mathematics. Therefore, we should have knowledge of applying mathematical concepts into Machine Learning models. These concepts include calculus, linear algebra, statistics, and probability.
* **Software engineering:** Machine Learning models are built to integrate with the software. Thus, an ML Engineer should have a thorough knowledge of software engineering.
* **Data mining and visualization:** As we built Machine Learning models on top of various data, it becomes essential to understand the data. For this, a Machine Learning enthusiast must have experience in data visualization and mining.
* **Machine Learning algorithms:** Along with all these, most importantly, we should have experience in implementing various ML algorithms.

# REFERENCES

* + 1. https://www.coursera.org/learn/machine-learning
    2. <https://machinelearningmastery.com>
    3. https://www.ml.cmu.edu/
    4. [www.datasciencecentral.com](http://www.datasciencecentral.com)
    5. <http://cs229.stanford.edu>
    6. <https://towardsdatascience.com>
    7. [www.holehouse.org](http://www.holehouse.org)
    8. <https://medium.com>