~Machine Learning Notes (Part 1)~
Prof. Andrew Ng.

Machine Learning is a process in which computer programs are able to perform task without explicitly programming.

Week 1

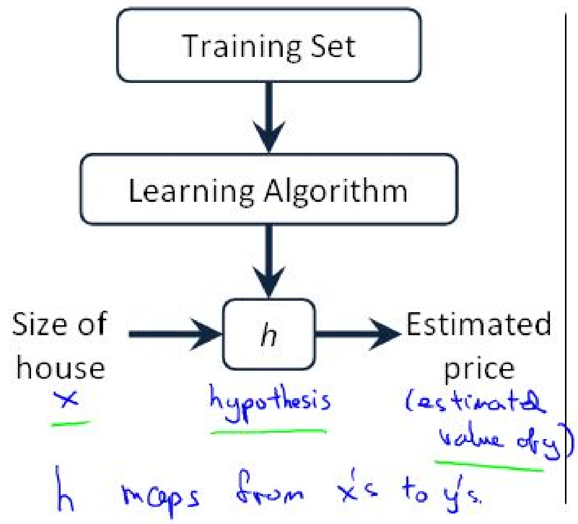
Part I - Type of ML:

- Supervised Learning: computer is given "right" answer or Experience, E to learn from in order to improve the Performance, P to complete Task T.
  - Regression Problem - deals with continuous output.
  - Classification Problem - deals with discrete output.
- Unsupervised Learning: Let computer learn by itself to complete task. e.g.
  - Clustering algo (market segmentation, social network analysis, news articles categorizing etc.)
  - Cock-tail party problem algo. (voice/sound sources differentiation)

Part II(1) - Training set (used in Supervised Learning):

Some notations:

- m: number of training sets.
- x: input value.
- y: output value.
- $(x^{(i)}, y^{(i)})$: the i^{th} training examples.



Hypothesis Function - takes in inputs x to estimate output.

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

Theta is parameter

The above example is a **Linear Regression function**.

Part II(2) - Cost Function:

- A function that gives parameters (weight) for the hypothesis function.
- Minimization of Cost Function has an objective of giving the **best** parameters in order to minimize the **average squared error** of

h(x) and y, such that the prediction given by the hypothesis is as close to the real value as possible.

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Part II(4~6) - Gradient Descent (GD) Algo:

Minimize any J($\theta_0, \theta_1 .... \theta_n$) (not only Linear Regression) by:

- choose an initial $\theta_0, \theta_1 ...... \theta_n$.
- update $\theta_0, \theta_1 .... \theta_n$ to reduce J($\theta_0, \theta_1 .... \theta_n$) by increment.

Definition - the updating of the parameters is subtracting a **derivative term times alpha** from it:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

- Alpha (Learning Rate) > 0.
    - Need to be chosen appropriately, converge too slowly if too small, diverge and miss the optimal minimum if too large. (like Newton mtd)

- Keep updating the Parameters simultaneously until J fxn converges to minimum.

- As J(θ_0, θ_1) approach local min, the overall update gets smaller and eventually becomes zero when min is reached, since the derivative term now = 0.

Part II(7) - Applying Gradient Descent to Linear Regression Cost Fxn:

Partial Derivative for θ_0 and θ_1 (This is the ones used in Linear regression):

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

- The special types of cost function that used for Linear Regression is always a **Convex function** (bowl shape), thus always converges to the global optimal minimum, no local min.
- **"Batch" GD** is the type of GD that involves the entire training set, where some other only involve one part of the training set data.

Linear Algebra:

- Matrix's dimension is present in row x col (m x n) format.
- Vector is matrix with 1 col.
- Addition of matrices MUST have same dimension.
- Scalar Multiplication is Commutative. i.e. a × B = B × a

- Matrix - Matrix Multiplication:
  - For A × B. Dimension have to be (m x n) (n x p) and will result in (m x p) matrix.
  - Not Commutative.
  - Is Associative. i.e A × (B × C) = (A × B) × C.

Example

$$\begin{bmatrix}1 & 3 & 2\\4 & 0 & 1\end{bmatrix}\begin{bmatrix}\boxed{\begin{matrix}1\\0\\5\end{matrix}} & \boxed{\begin{matrix}3\\1\\2\end{matrix}}\end{bmatrix} = \begin{bmatrix}11 & 10\\9 & 14\end{bmatrix}$$

$$\underline{\begin{bmatrix}1 & 3 & 2\\4 & 0 & 1\end{bmatrix}} \times \underline{\begin{bmatrix}1\\0\\5\end{bmatrix}} = \begin{bmatrix}11\\9\end{bmatrix}$$

$$\underline{\begin{bmatrix}1 & 3 & 2\\4 & 0 & 1\end{bmatrix}} \times \underline{\begin{bmatrix}3\\1\\2\end{bmatrix}} = \begin{bmatrix}10\\14\end{bmatrix}$$

Application

House sizes:

$$\begin{cases} 2104 \\ 1416 \\ 1534 \\ 852 \end{cases}$$

Have 3 competing hypotheses:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

$$\underbrace{\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}}_{\text{Matrix}} \times \underbrace{\begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix}}_{\text{Matrix}} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

Identity Matrix

Denoted I (or $I_{n \times n}$).

Examples of identity matrices:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2 x 2 3 x 3 4 x 4

For any matrix A, A $\times$ I = I $\times$ A = A.

(note: The 2 I's here have diff dimension)

Matrix inverse:

If A is an m x m matrix, and if it has an inverse,

$$A(A^{-1}) = A^{-1}A = I.$$

- Only **square matrix** has inverse.
- Matrices that do not have an Inverse are called “Singular” or “Degenerated”

Matrix Transpose

Example: $A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$ $\qquad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$

==

(Week 2)

Part IV(1) - Multiple features(variables) Linear Regression:

notations:

- n = number of features.
- x^i_j = jth features in the ith training example.

Hypothesis function with multiple features

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

To simplified, define x_0 = 1 and add it to the function.

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

The data can be represent by 2 vectors, x and θ both have [n+1 $\times$ 1] dimension, where n is the number of features.

Taking the Transpose of θ yield a [1 $\times$ n+1] vector, thus the function can be written as the product with vector x:

$$h_\theta(x) = \theta^T X$$

Part IV(2) - Gradient Descent for Multiple features:

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Note that the Parameters can be simply represented by vector θ.

Gradient Descent Algo:

New algorithm $(n \geq 1)$:

Repeat $\{$

$\frac{\partial}{\partial \theta_j} J(\theta)$

$$\theta_j := \theta_j - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

Note: This is equivalent to the previous definition, where $x_0^{(i)}$ is defined to be 1.

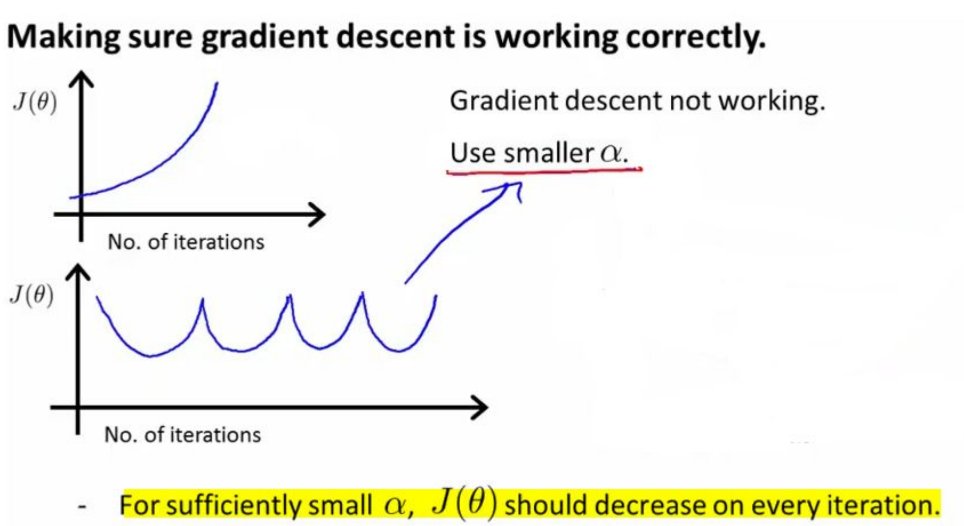
Part IV(3 & 4) - Scaling Features (x_j) for GD:

- Scaling the features that have values too large or too small into a similar scale in order to make GD algo more effectively converges to minimum of J function faster.
- Scale it to achieve a range of around (-3, 3).
- One way is to do Mean Normalization:

$$\frac{x_j - \mu_j}{S_j}$$

Where μ_j is mean and S_j is the range of the jth feature for all training examples.

Choosing the Learning Rate Alpha:



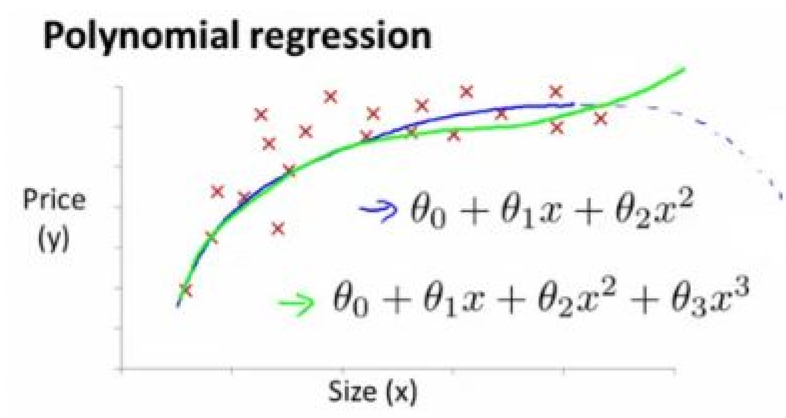
To choose α, try

$$\dots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \dots$$

Each number is 3 folds of the previous one.

Part IV(5) - Polynomial Regression:

features can be created and fit into polynomial functions



Part IV(6) - Normal Equation:

- Used to determine parameters (theta), that gives the minimum of Cost Function J, directly without having to use Gradient descent. The equation as follow:

$$\theta = (X^T X)^{-1} X^T y$$

Where X is the design matrix of [m * n+1] dimension which can be construct by:

1. Add an extra feature x_0 for each training example.
2. Make the transpose of each training example vector ($(x^{(i)})^T$) the rows of the design matrix.
3. Thus each column will contain value of one feature from all training examples.

m **training examples,** n **features.**

Gradient Descent	Normal Equation
• Need to choose α.	• No need to choose α.
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large.	• Need to compute $\boxed{(X^T X)^{-1}}$
	• Slow if n is very large.

Andrew Ng

*Normal Equation also doesn't need Features Scaling.

$(X^T X)$ may not be invertible (i.e. degenerated) when:

- There are redundant features. e.g. x_1 = size in ft², then another x_2 = size in m^2.

- Too many features. ($m \leq n$)

Side note on derivation:

* Normal Equation

$$\theta = (X^T X)^{-1} X^T y$$

Derivation:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$\therefore \; h_\theta(x) = \theta^T x \quad , \; \theta \in \mathbb{R}^{n+1}, \; x \in \mathbb{R}^{n+1}$$

cost function $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$

let $X = \begin{bmatrix} x_0^1 & x_1^1 & \dots & x_n^1 \\ x_0^m & x_1^m & \dots & x_n^m \end{bmatrix}$ $m \times (n+1)$ matrix

$$\therefore \; J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y) \quad , \; y \in \mathbb{R}^m$$

$$= \frac{1}{2m} \left[(X\theta)^T (X\theta) - 2(X\theta)^T y + y^T y \right]$$

Find min $J(\theta) \Rightarrow \frac{\partial J}{\partial \theta} = \frac{1}{2m} \left(2X^T X\theta - 2X^T y \right) = 0$

$$X^T X \theta = X^T y$$

$$\therefore \; \theta = (X^T X)^{-1} X^T y$$

==

(Week 3)

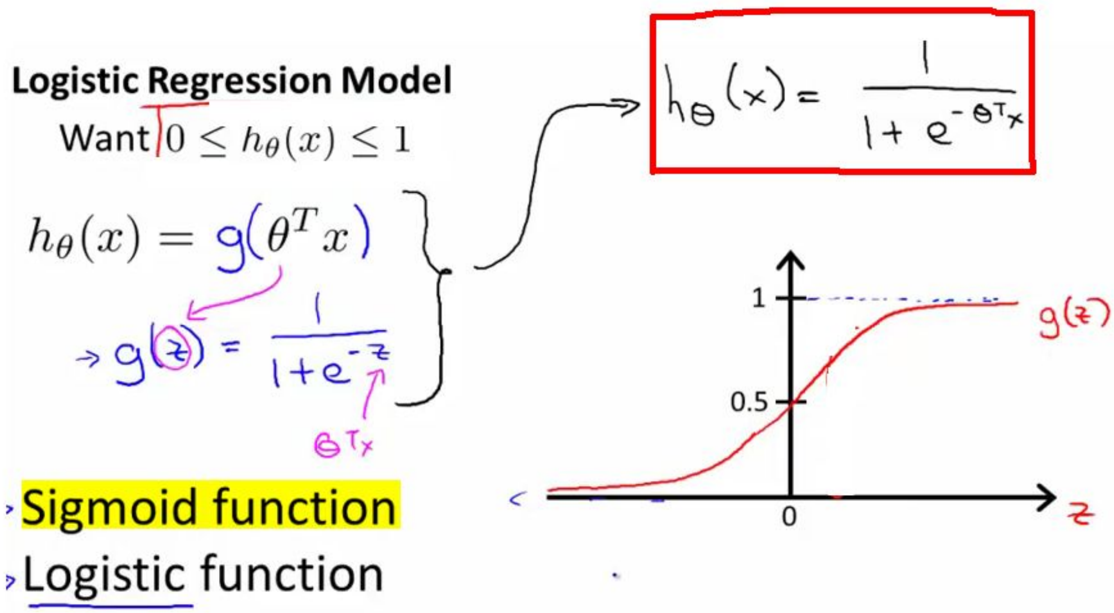
Part VI (1) - Classification:

- Linear Regression is a bad hypothesis for Classification problems (discrete outputs). Which for y = 0 and 1 (Binary classification), linear regression often gives outputs of $h_\theta(x) > 1$ or $h_\theta(x) < 0$.

- Therefore Logistic Regression is used to make prediction of Classification problems.
- Logistic Regression gives value of $0 \le h_\theta(x) \le 1$. (for binary inputs)

Part VI (2) - Logistic Regression hypothesis:

- Putting the previous form of hypothesis ($\theta^T x$) into the Sigmoid function or Logistic function, such that the outputs are always between 0 and 1.



Logistic regression

$$h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1+e^{-z}}$$

- The outputs of this Logistic Regression hypothesis can be interpret as "the estimated probability that y=1 on input x":

"probability that y = 1, given x, parameterized by θ"

$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$
$$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$$

Part VI (3) - Decision Boundary:

- The logistic regression hypothesis gives prediction of possible outcomes (e.g. tumor types).
- If only 2 outcomes is present (y= 1 or 0), we can say that:
  - if $h_\theta(x) < 0.5$, then it's predicting y = 0.
  - and if $h_\theta(x) \geq 0.5$, then it's predicting y = 1.
- The line at which the prediction switches is the Decision Boundary.

- In Sigmoid function g(z):
  - g(z) $\geq$ 0.5, when z $\geq$ 0.
  - g(z) < 0, when z < 0.
- Therefore, the Logistic Hypothesis $h_\theta(x) = g(\theta^T x)$ gives a value of:
  - greater than 0.5, when $\theta^T x \geq$ 0.
  - Less than 0.5, when $\theta^T x$ < 0.

- Thus the decision boundary is a property of the hypothesis, and not the dataset.

Part VI (4) - Logistic Regression Cost Function:

- Using Linear Regression Cost function on Logistic Regression problem (i.e. plugging in the Sigmoid function into the previous form of Cost fxn definition) will results in "non-convex" function with multiple local optima, where optimizing will be a problem.
- Therefore another cost function which produces convex function on Logistic regression function is needed.

General form of cost function can be written as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

Then Linear Regression Cost function in previously learnt will look like this:

$$\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

And the new Logistic Regression Cost function will be:

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

And Gradient Descent has the same form:

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

but note that the h(x) is different.

Part VI (5) - Advanced Optimization of Cost function:

- Other more advanced optimizing cost function other than GD include:
  - Conjugate Gradient
  - BFGS
  - L-BFGS.
- Advantages:
  - No need to manually pick alpha (learning rate).
  - Faster than GD.
- Disadvantage:
  - complex. Therefore use built in libraries for application:

Example: $\min_{\theta} J(\theta)$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \qquad \theta_1 = 5,\ \theta_2 = 5.$$

$$\boxed{J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2}$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient]
          = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
         (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

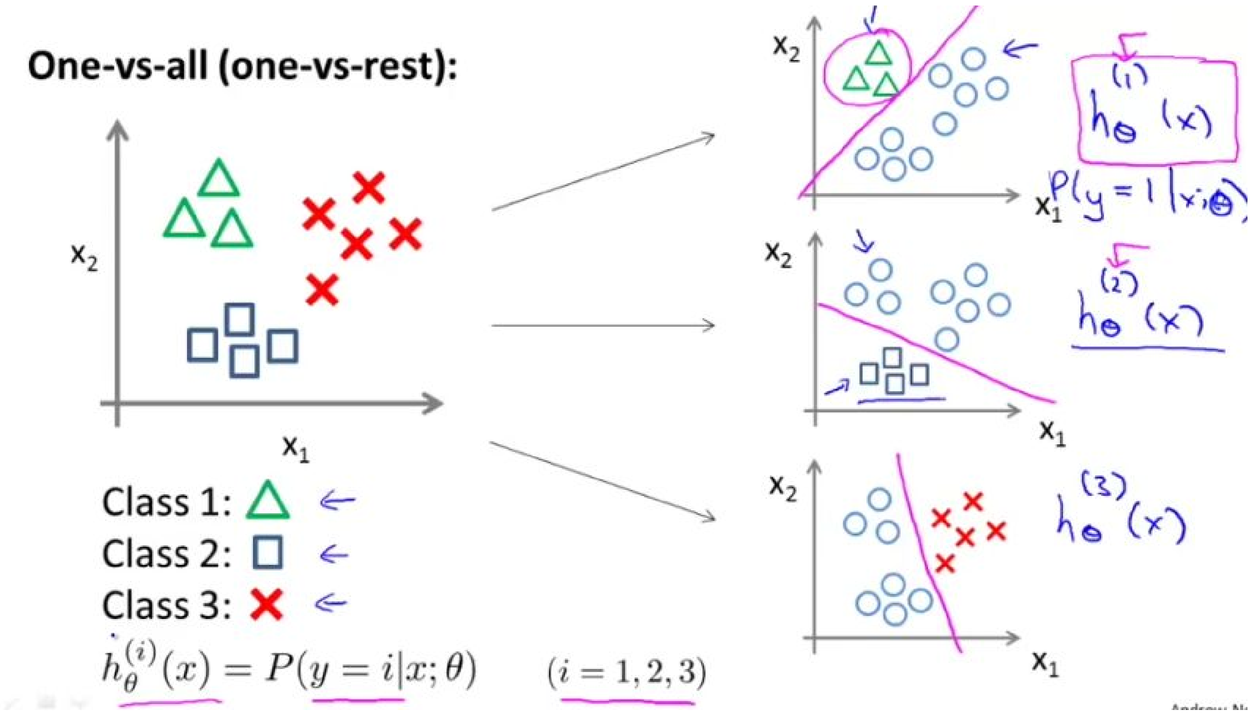
```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
     = fminunc(@costFunction, initialTheta, options);
```

$$\texttt{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \begin{matrix} \text{theta(1)} \\ \text{theta(2)} \\ \\ \text{theta(n+1)} \end{matrix}$$

```
function [jVal, gradient] = costFunction(theta)

    jVal = [ code to compute J(θ) ];

    gradient(1) = [ code to compute ∂/∂θ0 J(θ) ];

    gradient(2) = [ code to compute ∂/∂θ1 J(θ) ];
      ⋮
    gradient(n+1) = [ code to compute ∂/∂θn J(θ) ];
```

Part VI (6) - Multiclass Classification - One-vs-all:

- creating fake training examples to have only 1 class as positive and the others identically negative.



One-vs-all

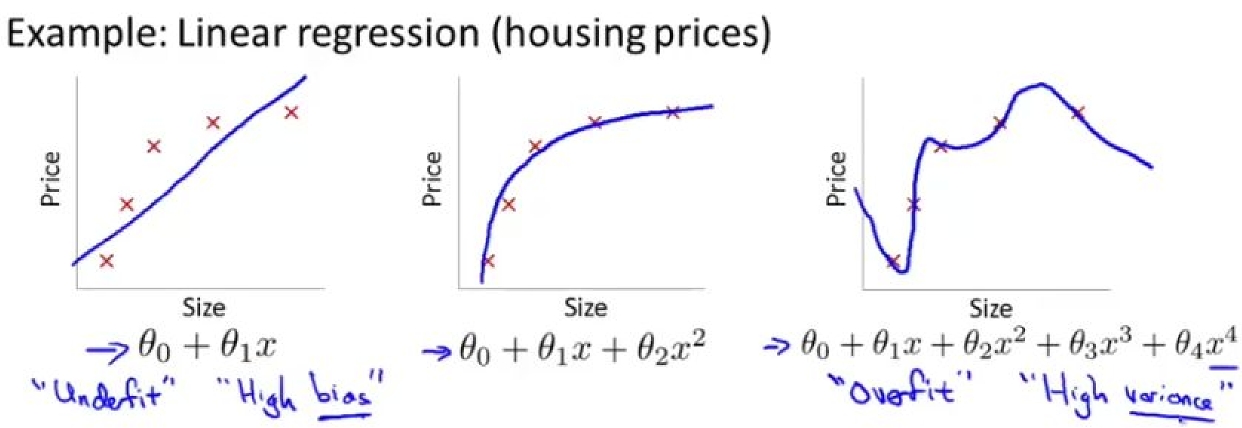
Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

Part VII (1) - Overfitting:

- Occurs when the learning algo tries too hard to fit all the training examples (i.e. J(θ) $\approx$ 0) but end up with a hypothesis with order that is too high, resulting in useless prediction when feed with new input.



(high variance is describing the high number of space of possible hypothesis when the order is high)

- 2 ways to address overfitting:
  - cut down number of features.
    - manually.
    - or Model selection algorithm.
  - Regularization:
    - Keeping all the features but reducing the magnitude of θ_j.

Part VII (2) - Regularization:

- a way to address "overfitting" hypothesis by penalizing the parameters (theta).

Regularization.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

- λ (regularization parameter) controls both:
  - fitting the training set well.
  - Keeping hypothesis simple (rid of higher order terms) by penalizing the parameter (θ). i.e. making parameters small (starting from θ_1 not θ_0).
- λ is a constant, usually quite large to penalize the parameter, such that if lambda is large, in order to make J(θ) small, parameters have to shrink.
- if λ is set to be too large, most terms in hypothesis apart from θ_0 will ≈ 0, resulting in "underfitting".

Part VII (3) - Regularized Linear regression:

- The GD for regularized linear regression cost function:

$$\theta_j := \theta_j\underline{(1 - \alpha\frac{\lambda}{m})} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

$$1 - \alpha\frac{\lambda}{m} < 1$$

however updating of θ_0 is treated separately:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

- The normal function when using on regularized linear regression cost function.

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Note: the matrix has a dimension of (n+1) x (n+1).

Part VII (4) - Regularized Logistic Regression:

- The cost function for regularized Logistic regression:

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \left(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log 1 - h_\theta(x^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

- The GD for regularized Logistic regression has the same form as the regularized Linear regression, but note that h(θ) is different.

fminunc (@costFunction

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ theta(1), theta(2), theta(n+1)

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
```

jVal = [code to compute $J(\theta)$];

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \left(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log 1 - h_\theta(x^{(i)}) \right] + \boxed{\frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2}$$

gradient(1) = [code to compute $\boxed{\frac{\partial}{\partial \theta_0} J(\theta)}$];

$$\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

gradient(2) = [code to compute $\boxed{\frac{\partial}{\partial \theta_1} J(\theta)}$];

$$\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1$$

gradient(3) = [code to compute $\frac{\partial}{\partial \theta_2} J(\theta)$];

$$\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} + \frac{\lambda}{m} \theta_2$$

⋮

gradient(n+1) = [code to compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

==