Week-3

⇒ Logistic regression : Algo used for classification problems (Don't get confused by the name; it isn't used for regression problems & is named that way just for historical reasons)

⇒ Binary classification :

* As the name suggests, instead of our output vector y being a continuous range of values, it will only be 0 or 1.

$y \in \{0, 1\}$ ; 0 → negative class
1 → Positive class

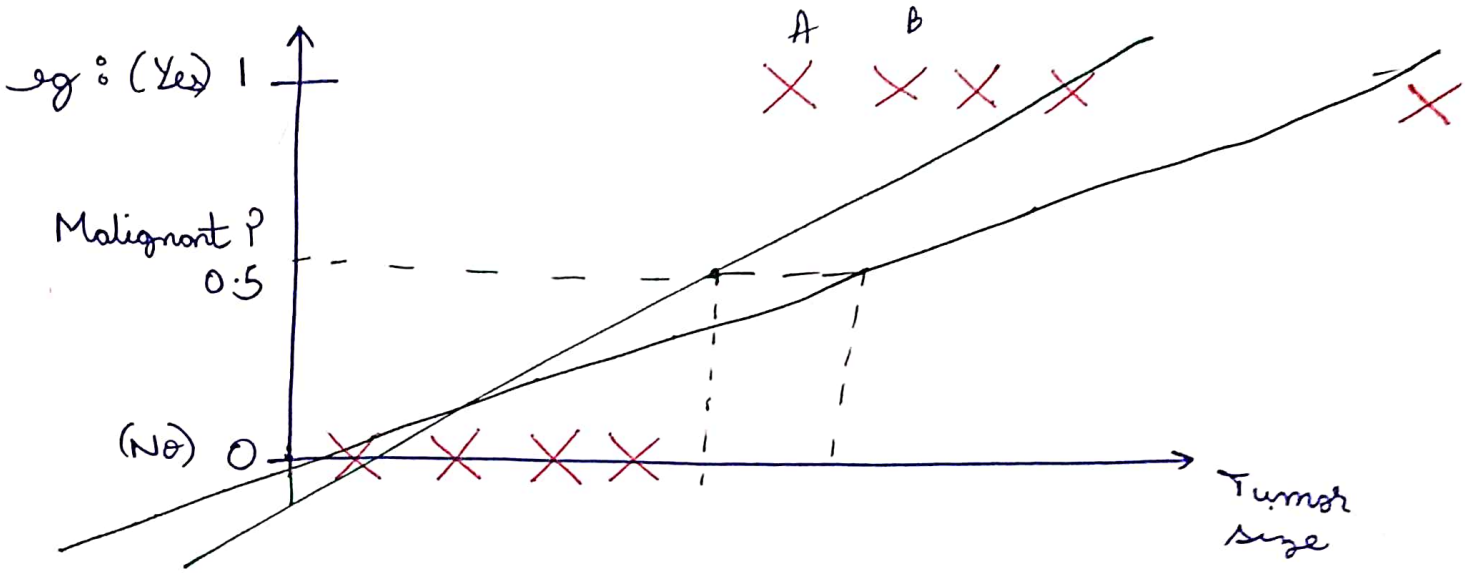
⇒ Developing a classification algo

* eg of classification problems : Email → Spam / not Spam?
Online transactions → fraudulent ?, Tumor → Malignant/Benign ?

⇒ Sketch :

** Logistic regression is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the +ve class "labelled 1"), & otherwise it predicts belong to the -ve class, labelled 0

⇒ Using linear regression




→ Let $h_\theta(x) = \theta^T x$

→ Put a threshold classifier output $h_\theta(x)$ at 0.5:

* $h_\theta(x) \geq 0.5 \rightarrow$ predict $y = 1$
* $h_\theta(x) < 0.5 \rightarrow$ predict $y = 0$

→ The green line worked well but once we introduced an outlier, (refer the black line) the algo started ~~the~~ mischaracterizing some elements {A & B}

→ Another issue with linear regression is:

* We know Y is 0 or 1.
* Still hypothesis can give values $> 1$ or $< 0$, which isn't that nice to observe as we're dealing with probabilities over here!

* The reason why regression didn't work is that classification algorithm doesn't actually requires a linear function. Lets add some non-linearity !

⇒ Logistic regression model

⇒ Hypothesis representation :

* We need our hypothesis $h_\theta(x)$ to satisfy

$$0 \leq h_\theta(x) \leq 1$$

This is accomplished by plugging $\theta^T x$ into the Logistic function

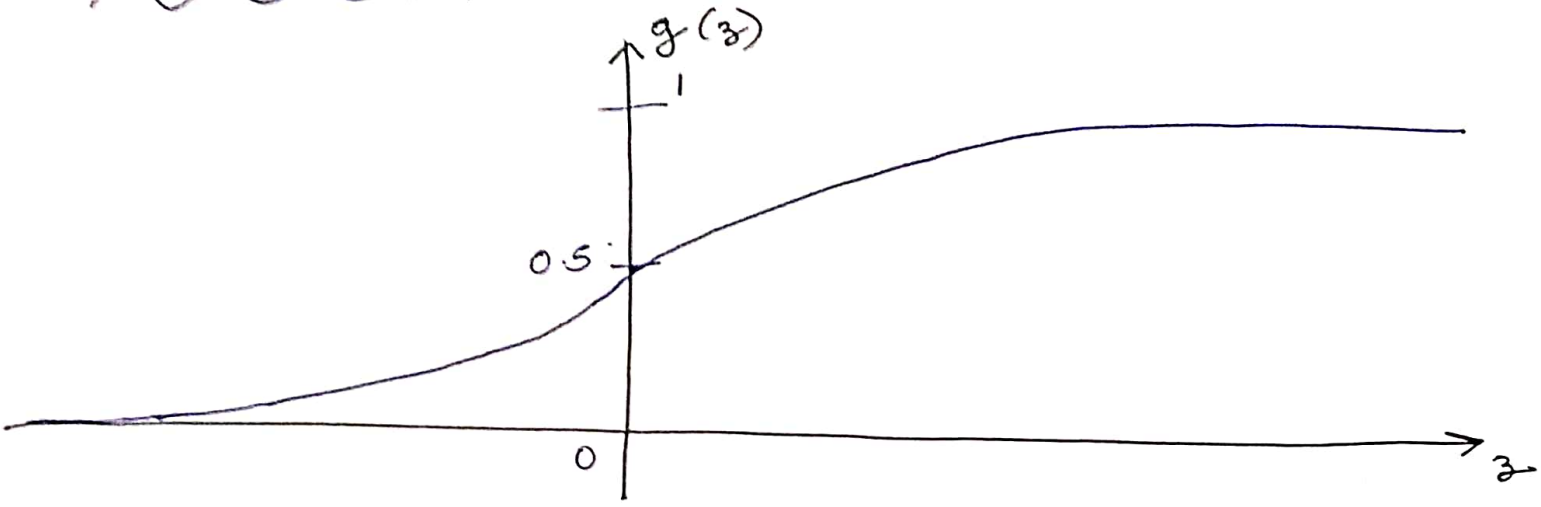
→ Our new form uses the "Sigmoid function", also called the "Logistic function".

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

$$z = \theta^T x$$

$$g(z) = \left(\frac{1}{1 + e^{-z}}\right)$$

→ Sketch of $g(z)$




* $g(z)$ maps any real number to the $(0,1)$ interval, making it useful for transforming an arbitrary valued funcⁿ into a funcⁿ better suited for classification.

* We start with our old hypothesis (linear regression) except that we want to restrict the range to 0 & 1. This is accomplished by plugging $\theta^T x$ into the logistic function.

$$\therefore \quad h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

⇒ Interpretation of hypothesis output

* $h_\theta(x)$ = estimated probability that $y=1$ on input $x$

* eg: if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorsize} \end{bmatrix}$

& $h_\theta(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

* $h_\theta(x) = P(y=1 \mid x;\theta)$
$= 1 - P(y=0 \mid x;\theta)$

} "probability that $y=1$, given $x$, parameterized by $\theta$"

*

* $P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$

⇒ Decision boundary: Helps to differentiate probabilities into +ve & -ve class

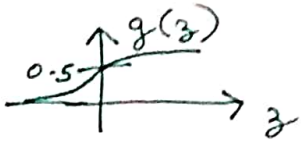
* In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis func$^n$ as follows:

$h_\theta(x) \geq 0.5 \rightarrow y = 1$

$h_\theta(x) < 0.5 \rightarrow y = 0$

* The way our logistic / sigmoid function g behaves that when the input is greater than or equal to 0, its output is greater than or equal to 0.5.

$$g(z) \geq 0.5 \quad \forall \quad z \geq 0$$




* Note:

$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$

$z = \infty, e^{-\infty} \to 0 \Rightarrow g(z) = 1$

$z = -\infty, e^{\infty} \to \infty \Rightarrow g(z) = 0$

*

$$h_\theta(x) = g(\theta^T x) \geq 0.5 \quad \text{when} \quad \theta^T x \geq 0$$

∴ **

$\theta^T x \geq 0 \Rightarrow y = 1$

$\theta^T x < 0 \Rightarrow y = 0$

→ The decision boundary is the line that separates the area where $y = 0$ & where $y = 1$. It is created by our hypothesis function.

→ Linear decision boundary

eg: Let $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$ & $x = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix}$

$$\Rightarrow h_\theta(x) = g(\theta^T x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$
$$= g(-3 + x_1 + x_2)$$

* ~~Suppose we have the~~

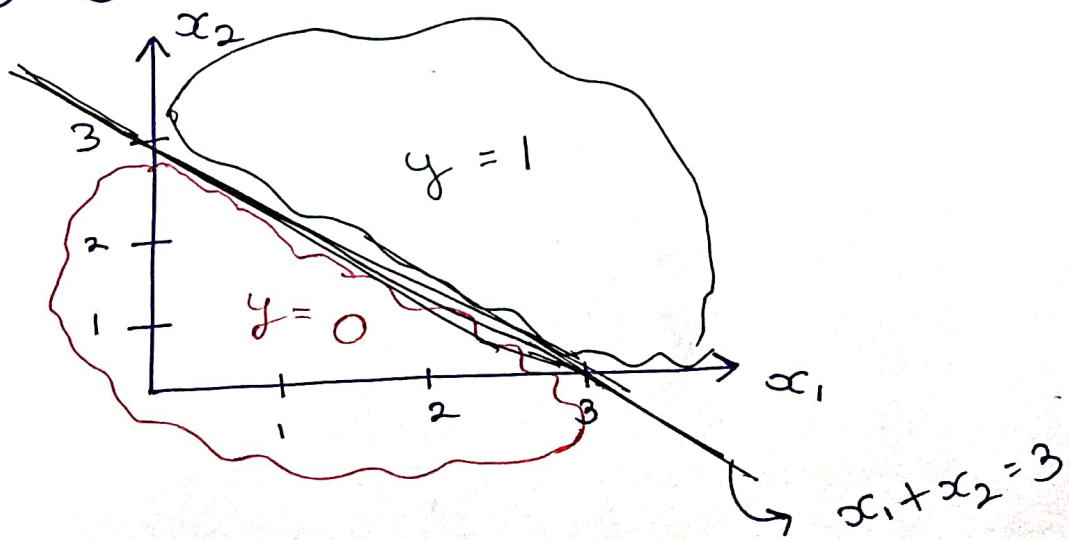
$\therefore$ $y = 1 \Rightarrow -3 + x_1 + x_2 \geq 0$

$\Rightarrow x_1 + x_2 \geq 3$

$\therefore$ $\boxed{y = 1 \Rightarrow x_1 + x_2 \geq 3}$

* $\boxed{x_1 + x_2 = 3 \rightarrow \text{Decision boundary}}$

* Sketch of decision boundary → $(x_1 + x_2) = 3$



## → Non linear decision boundary

* The input to the sigmoid function $g(z)$ {eg: $\theta^T x$} doesn't need to be linear, it could be a function that describes a circle {eg: $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$} or any shape to fit our data.

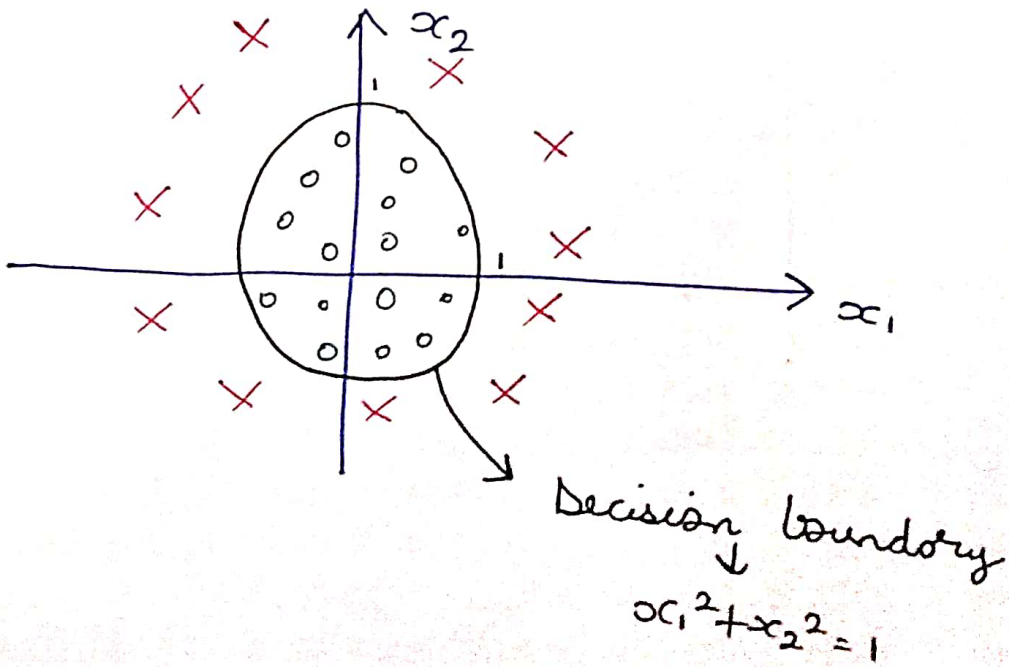
* eg: say we have: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_2^2)$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$\therefore y = 1 \Rightarrow -1 + x_1^2 + x_2^2 \geq 0$

$\Rightarrow x_1^2 + x_2^2 \geq 1$

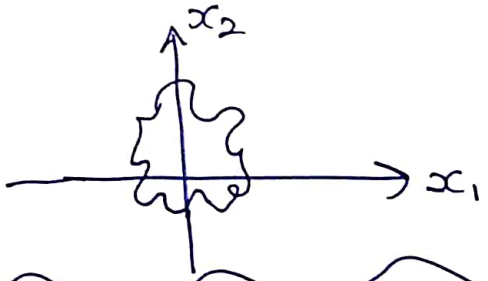
→ $x_1^2 + x_2^2 = 1$ is a circle of radius 1 around origin & is our decision boundary



* If we want to create some more complex decision boundary, we CAN formulate them using higher order polynomial terms.

eg: i) $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$

ii)




⇒ Notation

* m training samples
  - → Training set: $\{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$
  - → m samples: $x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}_{(n+1) \times 1}$ ; $x_0 = 1$

* $y \in \{0, 1\}$

* $h_\theta(x) = \dfrac{1}{1 + \exp(-\theta^T x)}$

→ Given the training set, how to chose $\theta$?

We need a cost function.

→ Revisiting linear regression

$$\text{Cost func}^n \equiv J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}\left(h_\theta(x^i), y^i\right)$$

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

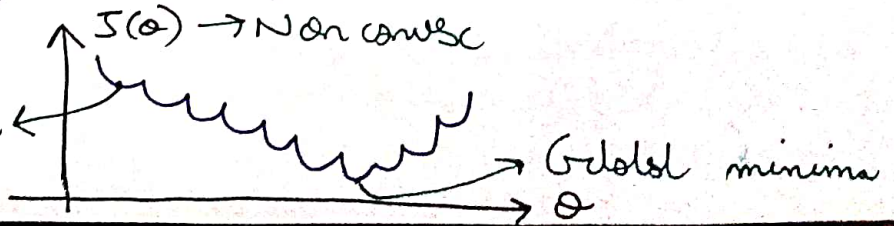
→ What if we use the above cost func$^n$ for logistic regression?

~~or~~ non convex

* As $h_\theta(x)$ is a non linear func$^n$ → sigmoid func$^n$. Using the above cost func$^n$ will lead to non linearities in $J(\theta)$ ~~as~~ and it might result in $J(\theta)$ being non convex.

* What's the horm if $J(\theta)$ becomes non convex?

If we apply gradient descent to such a cost func$^n$, the algo will keep getting stuck at local minimas instead of returning the global minima.

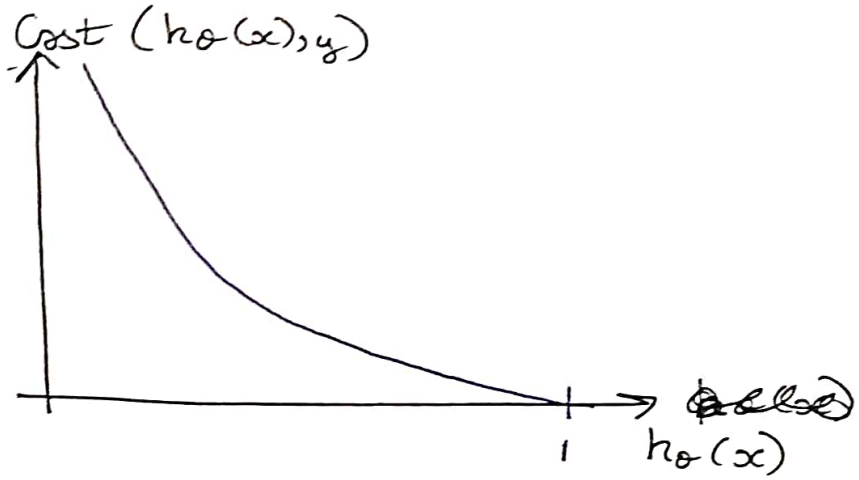


⇒ A convex logistic regression cost function

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

* This is the penalty that the algo pays!

→ $y = 1$ : $\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x))$



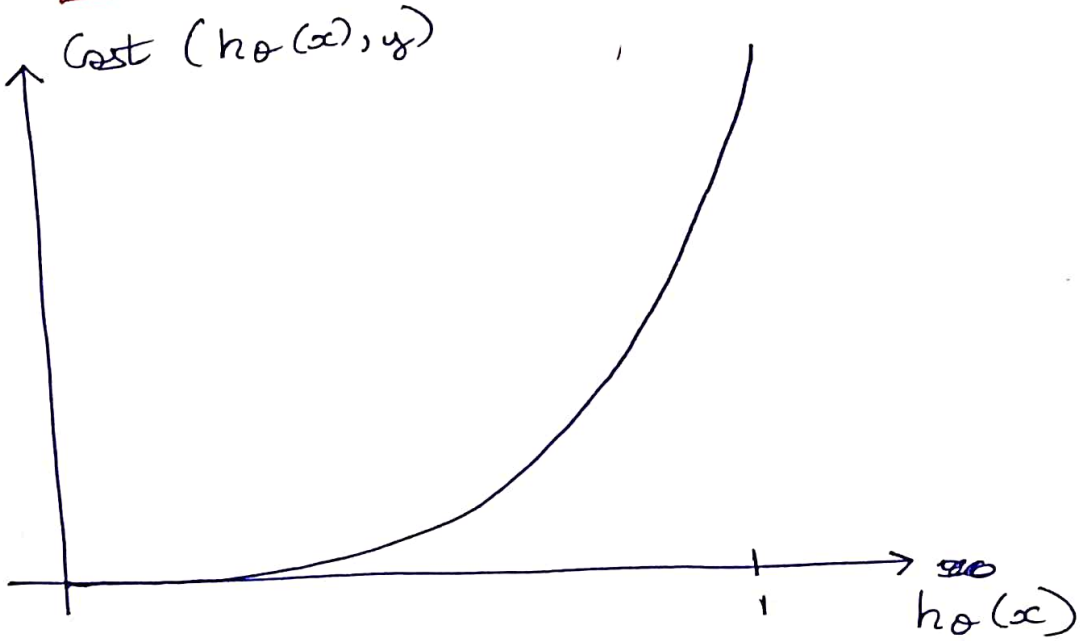

* X axis → What we/our algo predicts
* Y axis → Cost associated with that prediction

→ Interesting properties

* If $y = 1$ & $h_\theta(x) = 1$ : Means our prediction is exactly correct that corresponds to 0 cost (or penalty).
* As $h_\theta(x) \to 0$ : Cost goes to infinity. This captures the intuition that our algo is predicting $y = 0$ but we know $y = 1$, this will produce

the learning algorithm with a massive cost!

→ $y = 0 \Rightarrow$ $h_\theta(x) = -\log(1 - h_\theta(x))$




* Just get inverse of the other func$^n$.

* Cost $(h_\theta(x), y) \to \infty$ if $y = 0$ & $h_\theta(x) \to 1$

* If our correct answer 'y' is 0, then the cost func$^n$ will be 0 if our hypothesis func$^n$ also outputs 0.

If our hypothesis approches 1, then the cost func$^n$ will approach $\infty$.

* ~~Thus~~ ∞

* Thus, writing the cost function in this way guarantees that $J(\theta)$ is convex for logistic regression.

* ~~All~~ Insights summary

$\text{Cost}(h_\theta(x), y) = 0$ if $h_\theta(x) = y$

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

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

## ⇒ Simplified cost func$^n$ & gradient descent

* For binary classification $y$ is always 0 or 1.

* We can compress our cost function's 2 conditional cases into one case:

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$$

↓

Also called binary cross entropy loss function

→ When $y=1$ : Then the second term formulates to 0 & will not affect the result

→ When $y=0$ : 1st term → $-y \log(h_\theta(x))$ will be 0 & won't affect the result

**We can fully write our cost function as :

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

↳ Derived using maximum likelihood estimation

** A vectorised implementation is :

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \cdot \left( -y^T \log(h) - (1-y)^T \log(1-h) \right)$$

* Above : $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times 1}$ & $X \equiv$ design matrix $= \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^m)^T \end{bmatrix}_{m \times (n+1)}$

where $x_j = \begin{bmatrix} x_0^j \\ x_1^j \\ \vdots \\ x_n^j \end{bmatrix}_{(n+1)\times 1}$

## ⇒ Gradient descent

→ Remember that the general form of gradient descent is

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad ; \forall j \in [0, n]$$

}

→ We can work out the derivative using calculus to get:

Repeat until convergence {

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

$$; \forall j \in [0, n]$$

}

* Clearly the eqn is the same as the one we got for linear regression !!

The only difference is that our definition for the hypothesis has changed.

* We can monitor if the gradient descent is working properly with the same methods, as discussed for linear regression.

* Feature scaling for gradient descent also applies over here.

* Vectorized version :

$$\theta := \theta - \frac{\alpha}{m} X^T \left( g(X\theta) - \vec{y} \right)$$

⇒ Using advanced cost minimization algorithms

* Aside from gradient descent, there are more advanced & sophisticated algos that can provide different approaches to optimize the cost function for us.

→ Optimization algos :

* Conjugate gradient
* BFGS
* L-BFGS

* ~~These algos~~

→ Advantages :

* No need to manually pick α
* Often faster than the gradient descent

→ Disadvantages :

* More complex

* Prof. Ng suggests not to write these algos yourself but use the libraries instead, as they're already tested & optimized.

* We first need to provide a function that evaluates the foll. 2 functions for a given input value $\theta$ :

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

* We can write a single function that returns both of these :

```
function [jVal, gradient] = costFunction(theta)
    jVal = [.... Code to compute J(θ) ....]
    gradient = [.. Code to compute derivatives of J(θ)
                   wrt θ_j ∀ j ∈ [0, n] ..]
end
```

eg : Suppl. $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, $J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$$\therefore \frac{\partial J(\theta)}{\partial \theta_1} = 2(\theta_1 - 5) \quad \& \quad \frac{\partial J(\theta)}{\partial \theta_2} = 2(\theta_2 - 5)$$

For these : $jVal = \left[(\theta_1 - 5)^2 + (\theta_2 - 5)^2\right]$ for given $\theta$

$$\& \text{ gradient} = \begin{bmatrix} 2(\theta_1 - 5) \\ 2(\theta_2 - 5) \end{bmatrix} \text{ for given } \theta$$

* With the cost function implemented, we can call the advanced algs using

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

→ Here options : A datastructure giving options for the algorithm

→ fminunc : function used to minimise the cost function (find minimum of unconstrained multivariable function)

* We give to the function "fminunc" our cost function, our initial vector of theta values, & the "options" object that we created beforehand.

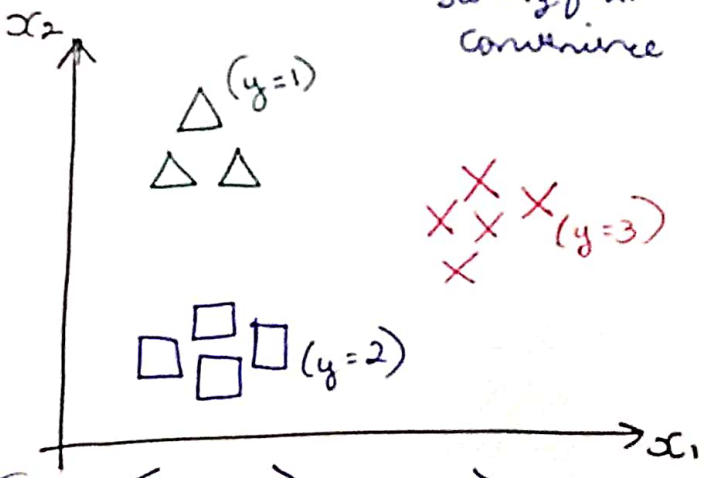
## ⇒ One vs All classification

* We keep choosing one class & then lump all the others into a single second class.

We do this repeatedly, applying binary logistic regression to each case & then use until we've calculated hypothesis' corresponding to each class possible.

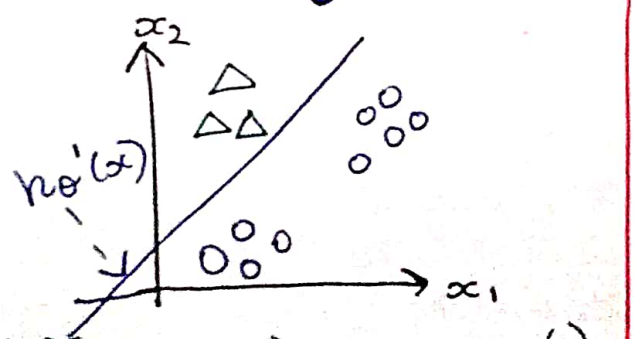
* ~~We do this repeatedly~~ We then use the hypothesis that returned the highest value as our prediction.

eg: For the given training set: * We'll be indexing categories starting from 1 here for convenience



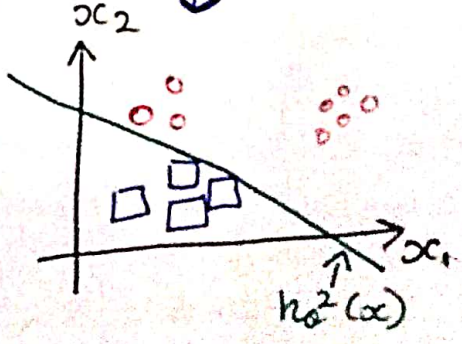

Split it into 3 separate binary classification problems

Split the ~~training set~~ into 3 separate ~~binary classification~~ problems



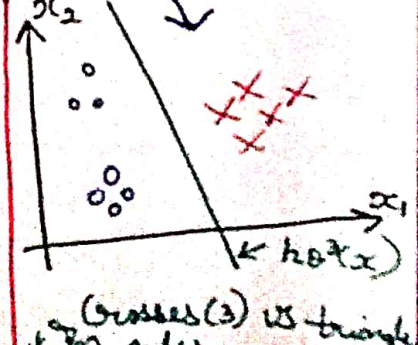

* Triangles (1) vs crosses & squares (0)

$h_\theta^1(x) = P(y=1 \mid x_i; \theta)$




* Squares (2) vs triangle & cross (0)

$h_\theta^2(x) = P(y=2 \mid x; \theta)$




* Crosses (3) vs triangle & squares (0)

$h_\theta^3(x) = P(y=3 \mid x; \theta)$

## ⇒ Multiclass classification

* Data can have more than 2 discrete outputs.
* Instead of $y = \{0, 1\}$, we expand our definition so that $y = \{0, 1, \ldots, k\}$; here $(k+1)$ are the no. of categories available. We indexed from 0 but you can also start indexing from 1.

$$y \in \{0, 1, \ldots, k\}$$

$$h_\theta^{(0)}(x) = P(y = 0 \mid x; \theta)$$

$$h_\theta^{(1)}(x) = P(y = 1 \mid x; \theta)$$

$$\ldots$$

$$h_\theta^{(k)}(x) = P(y = k \mid x; \theta)$$

$$\text{prediction} = \max\left(h_\theta^{(i)}(x)\right)$$

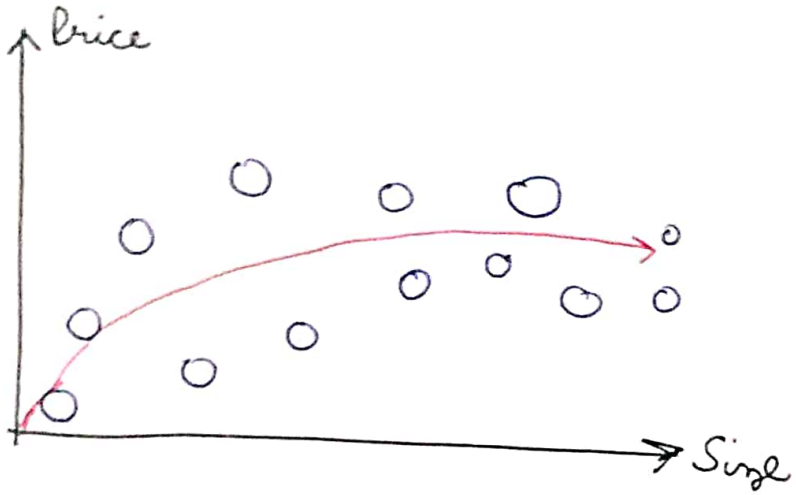
eg: Weather : Sunny, Cloudy, Rain, Snow

$$y = 0 \;,\; 1 \;,\; 2 \;,\; 3$$

* We are basically classing one class as true & lumping all the others into a single second class
We do this repeatedly

⇒ The problem of overfitting {Refer vid "Bias & Variance" by Statsquest}

→ Imagine we collected prices of diff. houses & their sizes. We plot the data:

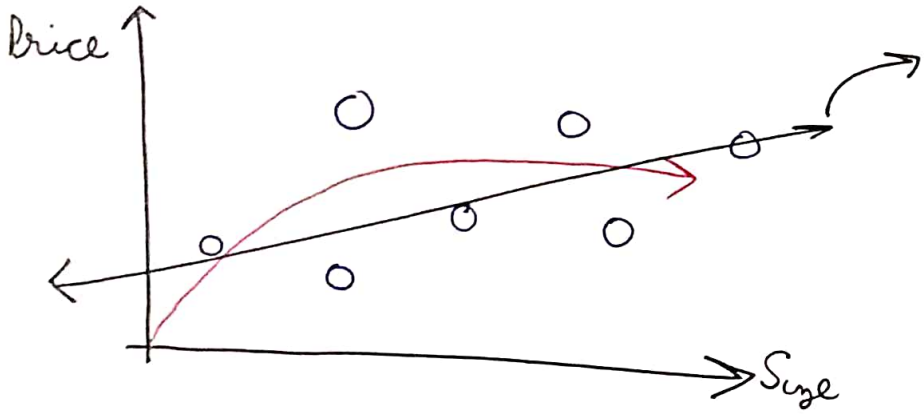



* Ideally we'd know the exact mathematical formula that describes the relationship b/w size & house price.
* But in this case, we don't know the formula, so we're going to use 2 machine learning methods to approximate this relationship.
* However, I'll love the "true" relationship curve in the figure for reference. {Refer the red line above}

⇒ Workflow

→ Firstly, we'll split the data into 2 sets, one for training the ML algs (blue dots) & one for testing (green dots).

→ Using Linear regression on training set




Linear regression fits a straight line to the training set. $h_\theta(x) = \theta_0 + \theta_1 x$

* Note the straight line doesn't have the "flexibility" to accurately replicate the arc in the "true" relationship.

Thus the "straight line" will never capture the true relationship b/w size & price, no matter how well we fit it to the "training set".

→ Bias: The inability for a machine learning method to capture the true relationship is called bias.

* Mathematically, bias is the difference b/w the average prediction of our model & the correct value which we are trying to predict.

eg: Let Price (x) denote the "actual" price of a 'x' sized house

& $h_\theta(x)$ be our hypothesis function for our model / "predicted" price of a 'x' sized house

Then

$$\text{Bias} = E[\text{Price}(x)] - h_\theta(x)$$

↳ Expectation operator

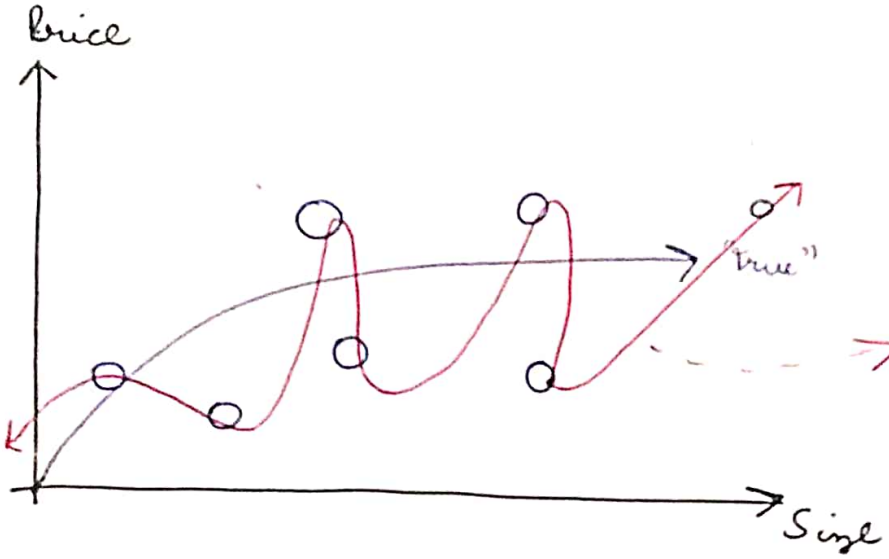
; $E[\text{price}(x)]$ = mean of Price (x)

* Models with high bias pays very little attention to the training data & oversimplifies the model.

* Because the straight line can't be curved like the "true" relationship, it has a relatively large amount of bias.

* Bias is the accuracy of our prediction. Bias occurs when an algo has limited flexibility to learn the true signal from the dataset.

→ Using polynomial regression ("Squiggly line") on training set




Polynomial regression with high order polynomials in hypothesis

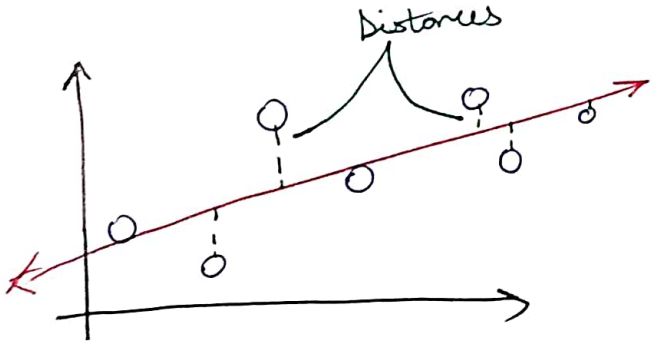
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

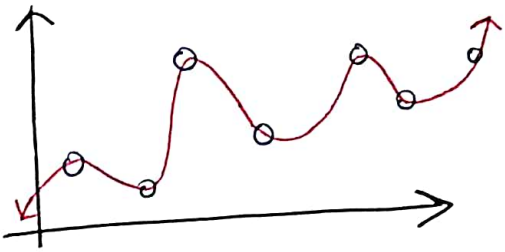
* Because the "squiggly line" can handle the arc in the true relationship b/w ~~Size~~ size ~~weight~~ & price ~~Height~~, it has very little bias.

⇒ Comparing the performances of the 2 algos

* We'll do so by measuring the distances from the fit lines (or curves) to the data, square them & add them up.

→ Testing on training set:



(Notice how the squiggly line fits the data so well that the dist. b/w lines & data are all 0!!

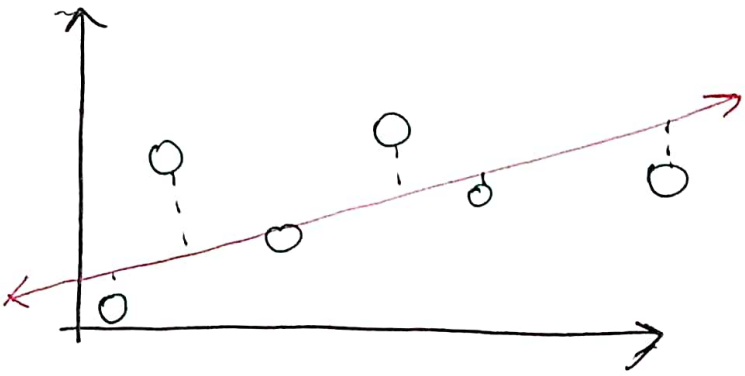
* ~~In the contest to see whether the~~

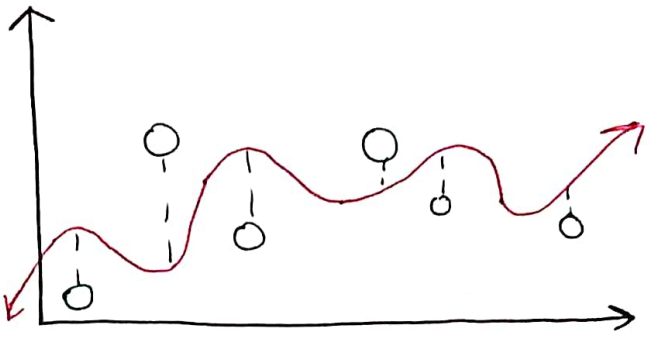
* Clearly, the "squiggly line" fits the "training set" better/better when compared to the "straight line".

* The thing is, we still have the "test set"!!

* We can conclude that the straight line has high bias & squiggly line has low bias.

→ Performance on test set





* Clearly the "straight line" performed "better" on the "test set".

⇒ Variance :

* Even though squiggly line performed better while "fitting" the training set, it performed terribly while "fitting" the test set. This is called overfitting.

In Machine Learning lingo, the difference in "fits" between different datasets is called variance.

* "The variance is how much the predictions for a given data point vary b/w diff. realizations of the models."

* In contrast, the straight line has relatively low variance, because the sums of squares are very similar for diff. datasets.

This is also called underfitting.

⇒ Underfitting:

* Underfitting, or high bias is when when the form of our hypothesis function h maps poorly to the trend of the data.

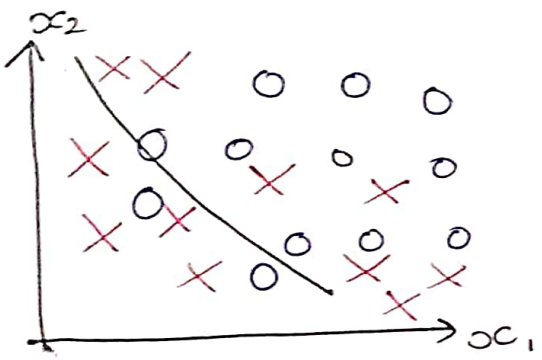
* Usually caused by a function that is too simple or uses too few features.

⇒ ~~Underf~~ Overfitting:

* Overfitting, or high variance is caused by a hypothesis function that fits the available data but does not generalise well to predict new data.

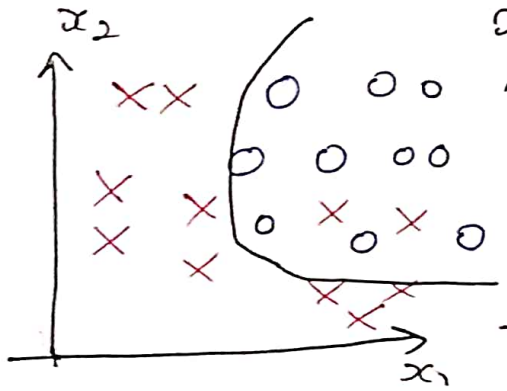
* Usually caused by a complicated function that creates a lot of un-necessary curves & angles unrelated to the data.

eg: Overfitting with logistic regression



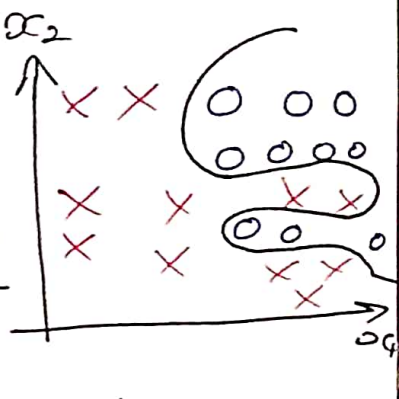

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

Underfitting (High bias)




$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

Nice fit




$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \dots)$

Overfitting (high variance)

⇒ Addressing overfitting

* If you have too many features & little data — overfitting can be a problem.

→ How to deal with this?

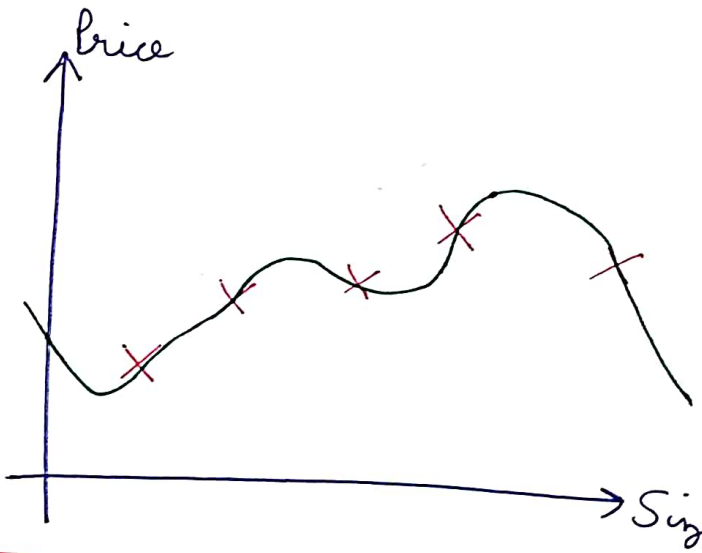
1) * Manually select which features to keep.
   * Model selection algos (Coming up soon!)
   * Ideally select those features which minimize data loss

2) Regularization: * Keep all features, but reduce magnitude of parameters $\theta$.
   * Works well when we have a lot of features, each of which contributes a bit to $y$.

## ⇒ Cost function optimization for regularization

* If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their "cost".

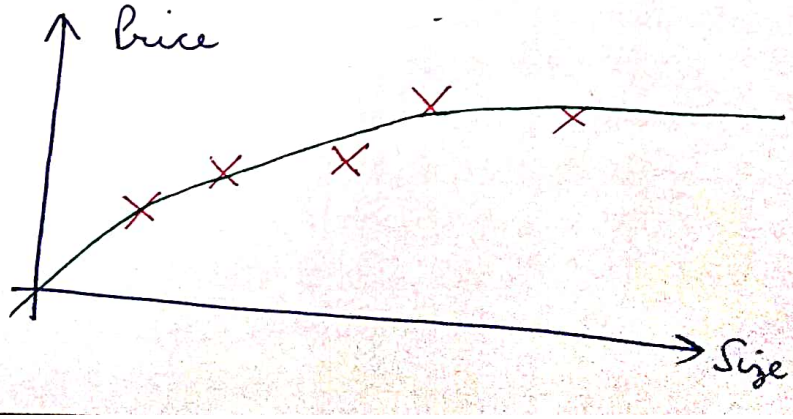
* eg: We wanted to make the following function more quadratic ☺




Let $x \rightarrow$ Size of house,

then $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

more quadratic like:




$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

* We'd want to eliminate the influence of $\theta_3 x^3$ & $\theta_4 x^4$.

Without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our cost function:

$$\text{Suppl. } J(\theta) = \frac{1}{2m}\left[\left\{\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2\right\} + (1000\ \theta_3^2 + 1000\ \theta_4^2)\right]$$

→ We've added 2 extra terms at the end to inflate the cost of $\theta_3$ & $\theta_4$.

→ Now, in order for the cost function to get close to zero, we'll have to reduce the values of $\theta_3$ & $\theta_4$ to near zero.

This will in turn greatly reduce the values of $\theta_3 x^3$ & $\theta_4 x^4$ in our hypothesis function.

⇒ Regularization : Technique used for tuning the function by adding an additional penalty term in error function

* The idea is that small values for parameters $(\theta_0, \theta_1, \ldots, \theta_n)$ & results in :

→ "Simpler" hypothesis

→ Less prone to overfitting

* With regularization, take cost function & modify it to shrink all the parameters :

→ Add a term at the end

~~→ The regularization~~

- The regularization term shrinks every all the parameters.
- By convention, you don't penalize $\theta_0$ ; minimization is from $\theta_1$ onwards.

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

→ $\lambda$ is the regularization parameter

* Controls tradeoff b/w our 2 goals:
  - 1) Want to fit the training set well
  - 2) Want to keep the parameters small

* If $\lambda$ is very large, we end up penalizing ALL the parameters ($\theta_1, \theta_2, \ldots$, 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

eg: $h_\theta(x) = \theta_0$ (!!)

This results in underfitting!!

* $\lambda$ should be chosen carefully.

* Will study automatic way to select $\lambda$ later!

## ⇒ Regularized linear regression

→ Gradient descent : We will modify our gradient descent function to separate out $\theta_0$ from the rest of the parameters because we do not want to penalize $\theta_0$.

Repeat {

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

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m}$$

$$\theta_j := \theta_j - \alpha \left[ \left( \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i \right) + \frac{\lambda \theta_j}{m} \right]$$

$$\forall j \in \{1, 2, \ldots, n\}$$

* The term $\frac{\lambda \theta_j}{m}$ performs our regularization.

* With some manipulation, our update rule can also be represented as:

$$\theta_j := \theta_j \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{\text{Always less than 1}} - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

** → Intuitively, you can see $\left(1 - \frac{\alpha\lambda}{m}\right)$ reduces the value of $\theta_j$ by some amount on every update.

---

⇒ Normal eq^n:

* To add in regularization, just add another term inside parenthesis:

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$$

where & $L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$

is a square matrix of size $(n+1)$

⇒ Non invertibility issue :

* Consider an example, where number of examples$^{(m)}$ is less than or equal to the no. of features ($n$). (Cuz it'll turn into a non square matrix then)

Then the matrix $(X^TX)$ will be non-invertible (or singular, or degenerate) !

* Using pinv() can work, but won't give the "exact or concrete" result.

* However, it can be proven that if we add the term $\lambda L$ then $(X^TX + \lambda L)$ becomes invertible.

So, regularization also handles the non invertibility case !

## ⇒ Regularized logistic regression

→ Cost function :

* Just add a $\frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$ to our old cost function.

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

* The 2 in $(2m)$ is just there for convenience while calculating $\frac{\partial J(\theta)}{\partial \theta}$

→ We've explicitly excluded the bias term $\theta_0$.

Thus, when computing the gradient descent eqn, we should continuously update the 2 full. eqn's

→ Gradient descent :

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)\, x_j^i + \frac{\lambda}{m}\theta_j\right]$$

$$j \in (1, 2, 3, \ldots n)$$