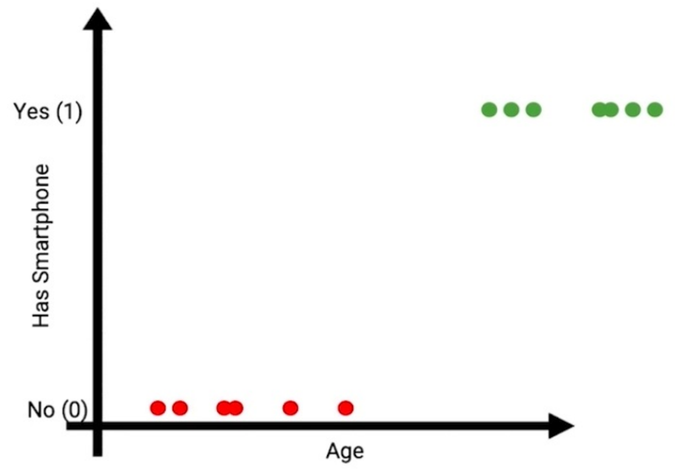
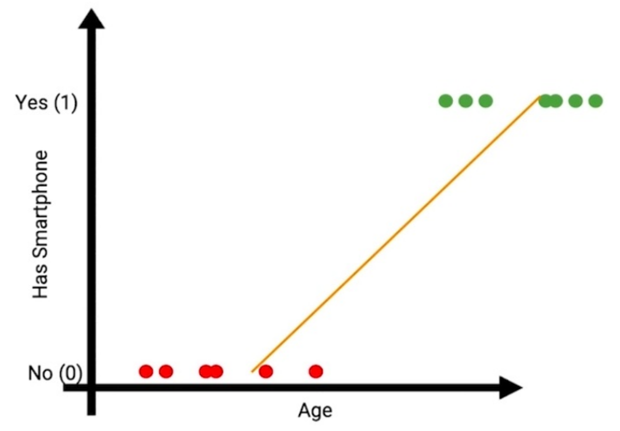
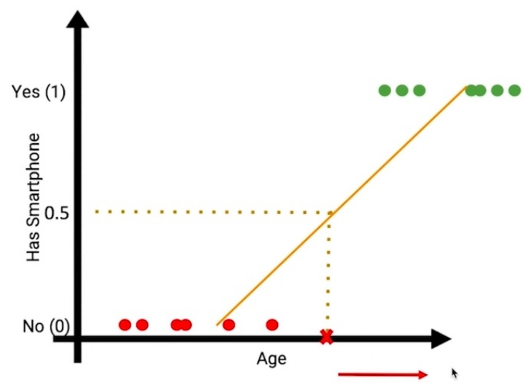
Logistic Regression



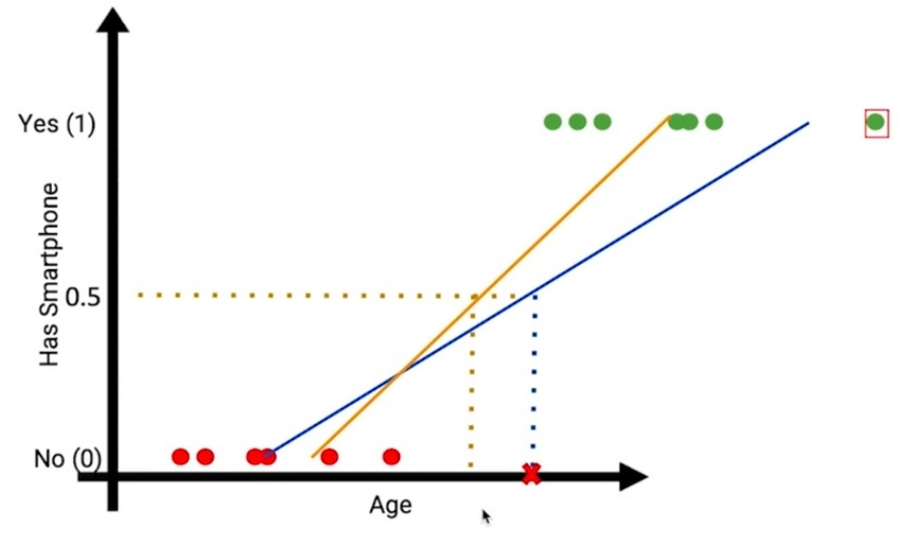
- If i want to predict, weather a person having smart phone or not with respect to linear line



- Lets fit a line
 # ideally the value should onle be zero and one

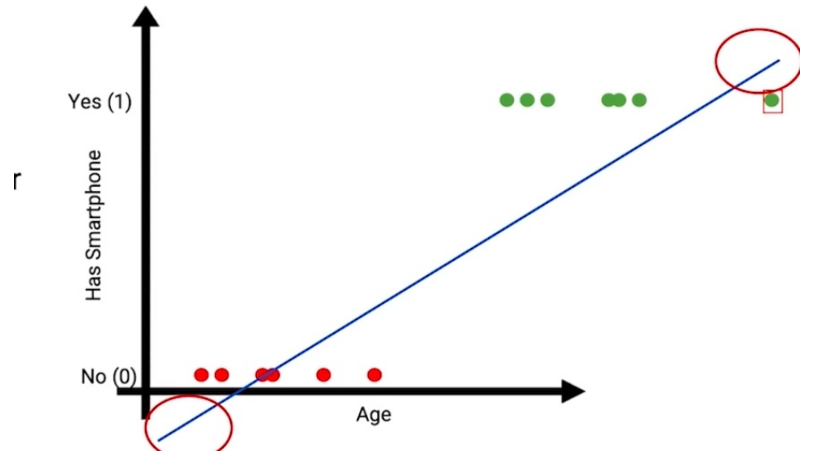


if we decide a threshold,
if the value of age is below the perticular pint the person does have a smart phone



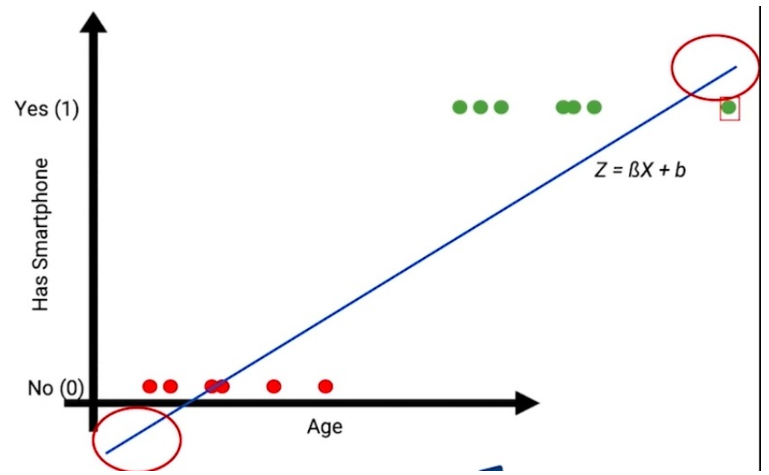
**the moment we add some data point,
the slope of the line changes and so that the thresold**

It does not give a right way to classify a problem



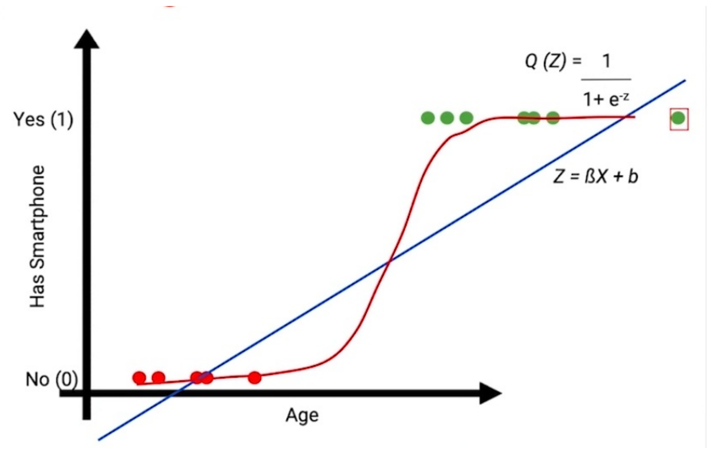
also, when we extend the line, it does not mean anything.....

if classification, below 0 and above 1 does not have any meaning



y = mx + c

here, i actully need a fucntion of y which transform the linear line to cover all the data points



$$Z = \beta X + b$$

$$\hat{Y} = Q(Z)$$

$$Q(Z) = \frac{1}{1+e^{-z}}$$

$$\boxed{\hat{Y} = \frac{1}{1+e^{-Z}}}$$

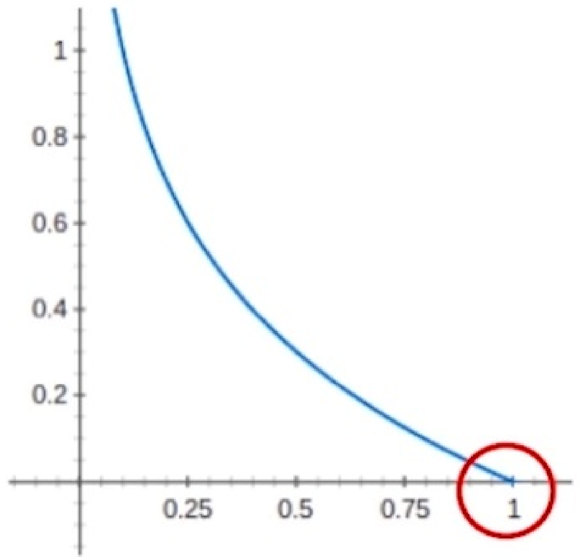
$$\hat{Y} = \frac{1}{1+e^{-(\beta X + b)}}$$

The prediction through line we get are contineous in nature, so we can use them as the probabilities to make prediction.....

- if the data point is extream right than probabilities are close to one and visa-versa

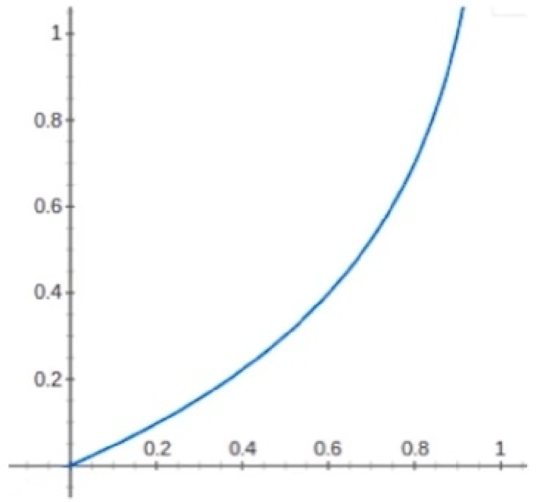
$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -(y_i * \log(\hat{Y}_i) + (1 - y_i) * \log(1 - \hat{Y}_i))$$

When y=1



$$J = \frac{1}{N} \sum_{i=1}^{N} -(\mathbf{y_i * \log(\hat{Y}_i)} + (1 - y_i) * \log(1 - \hat{Y}_i))$$

When y=0



$$J = \frac{1}{N} \sum_{i=1}^{N} -(y_i * \log(\hat{Y}_i) + \mathbf{(1 - y_i) * \log(1 - \hat{Y}_i)})$$

Gradient Descent in Logistic Regression

$$J = \frac{\sum_{i=1}^{n} -(Y_i \log(\hat{Y}_i) + (1-Y_i)\log(1-\hat{Y}_i))}{n}$$

$$ß = ß - \alpha G_ß$$

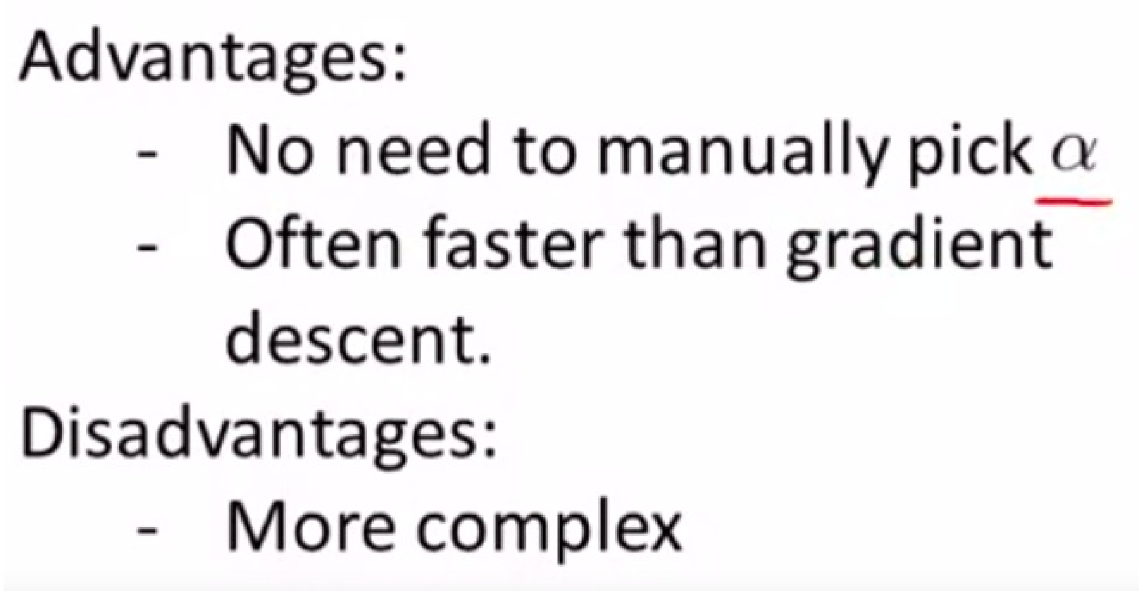
$$b = b - \alpha G_b$$

$$G_ß = \frac{\partial(J)}{\partial ß} = \frac{-2\sum_{i=1}^{n}(\hat{Y}_i - Y_i)X_i}{n} = \frac{-2}{n}\sum_{i=1}^{n}\left(\frac{1}{1+e^{-(ßXi+b)}} - Y_i\right)X_i$$

$$G_b = \frac{\partial(J)}{\partial b} = \frac{-2\sum_{i=1}^{n}(\hat{Y}_i - Y_i)}{n} = \frac{-2}{n}\sum_{i=1}^{n}\left(\frac{1}{1+e^{-(ßXi+b)}} - Y_i\right)$$

Optimization Algorithms:

# Gradient Descent
# Conjugate gradient
# BFGS
# L - BFGS



Advantages:

- No need to manually pick $\underline{\alpha}$
- Often faster than gradient descent.

Disadvantages:

- More complex

Multiclass Classification

One Vs. All method : Building a model with one class versus all (Not that class)

class- A Vs. Not class - A
class- B Vs. Not class - B
class- C Vs. Not class - C

ID	AGE	Class A
1	2	1
2	33	0
3	7	1
4	25	0
5	4	1
6	48	0

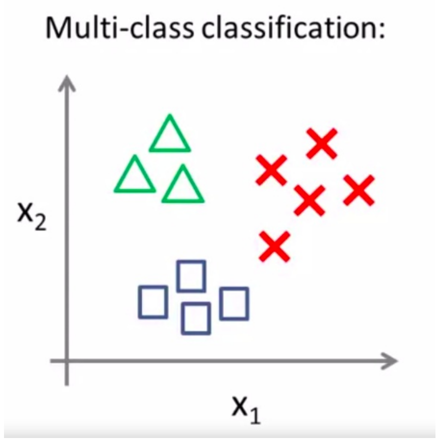
ID	AGE	Class B
1	2	0
2	33	0
3	7	0
4	25	0
5	4	0
6	48	1

ID	AGE	Class C
1	2	0
2	33	1
3	7	0
4	25	1
5	4	0
6	48	0

Get the Probability outcomes

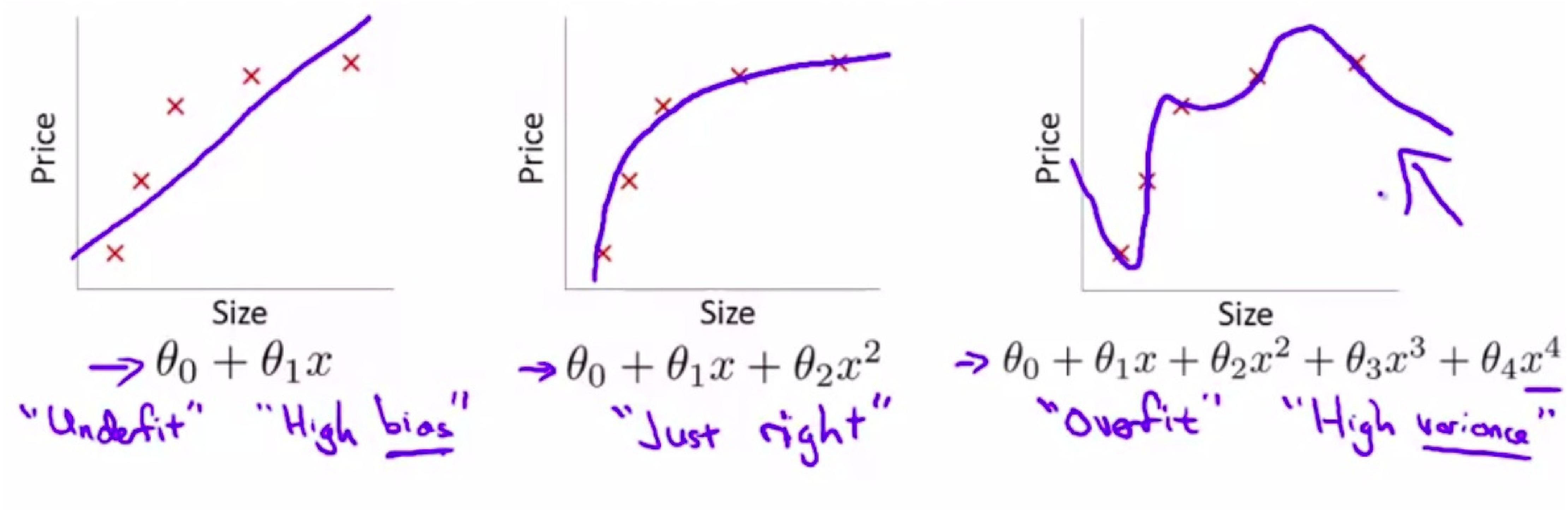
ID	AGE	Number of phones	P(A)	P(B)	P(C)
1	2	A	0.92	0.12	0.09
2	33	C	0.33	0.17	0.94
3	7	A	0.83	0.11	0.10
4	25	C	0.14	0.40	0.88
5	4	A	0.87	0.23	0.21
6	48	B	0.21	0.79	0.39

Which ever class has higher probability, we will make that prediction



# Bias: Error in training $\longrightarrow$ leads to UNDERFITTING: model has not fitted training data

# Variance: Error in $\longrightarrow$ leads to OVERFITTING: model has overly trained on training data, not generalized model



Underfittin
g

$\downarrow$

High bias,
High

Overfitting

$\downarrow$

Low bias,
High

Bias: Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.

- Model with the high bias pays little attention to the training data and oversimplifies the model.
- It always leads to high error in training and testing data.

Variance: Variance is the variability of model prediction for a given data point or value which tells us the spread of the data.

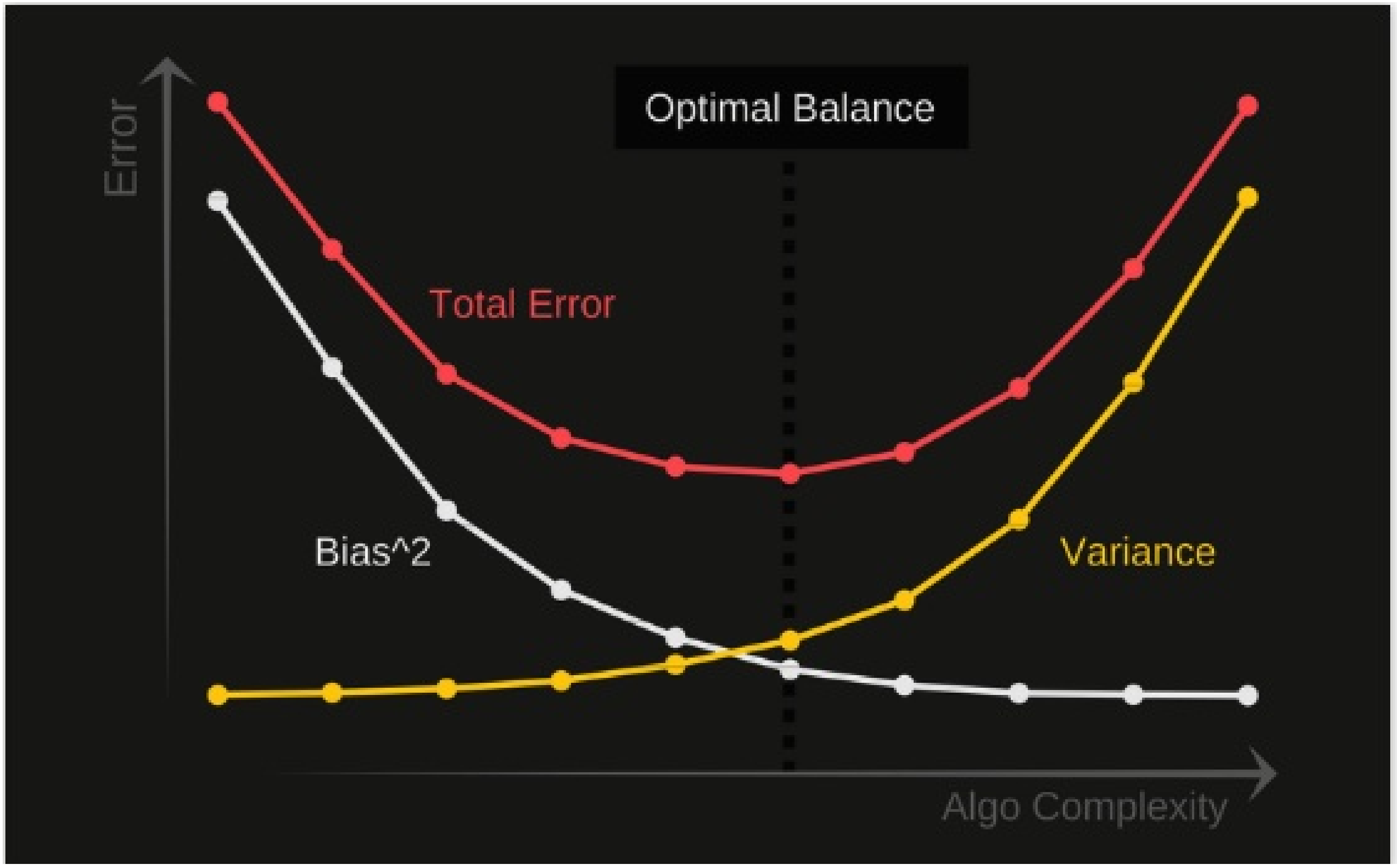
- Model with high variance pays a lot of attention to training data and does not generalize on the data which it has not seen before.
- As a result, the model performs very well on training data, but has high error in testing data.

In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.

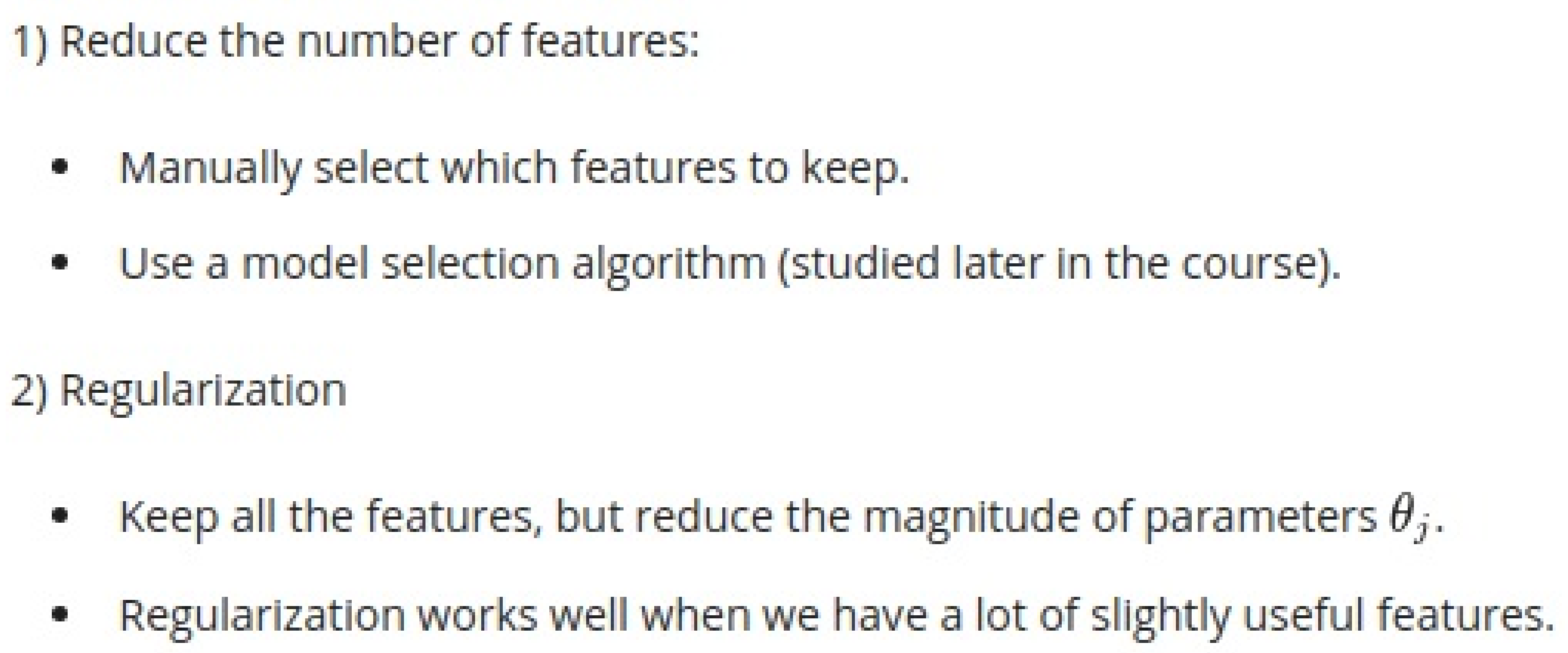
In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.

Bias-Variance Tradofff

Total Error = Bias^2 + Variance + Irreducible Error



Addressing OVERFITTING



1) Reduce the number of features:

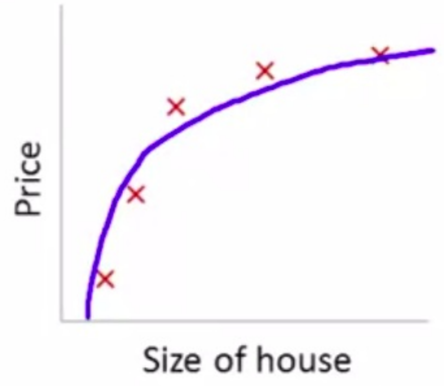
- Manually select which features to keep.
- Use a model selection algorithm (studied later in the course).

2) Regularization

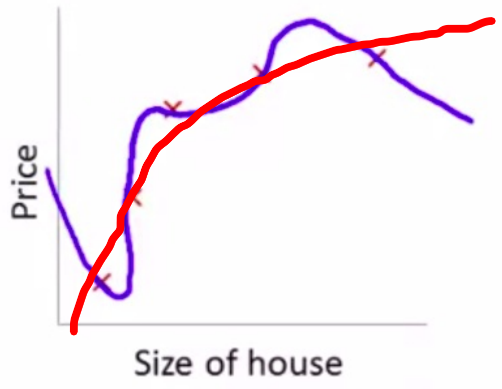
- Keep all the features, but reduce the magnitude of parameters θ_j.
- Regularization works well when we have a lot of slightly useful features.

Regularization

Intuition



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \qquad \theta_4 \approx 0$

To overcome the OVERFITTING we can convert this polynomial equation to quadratic equestion(red line)

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis $\leftarrow$
- Less prone to overfitting

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

$\theta_1, \theta_2, \theta_3, \ldots, \theta_{100}$

We have to add regularization term to the cost function

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

$$\min_\theta J(\theta)$$

Regularization Perameter;

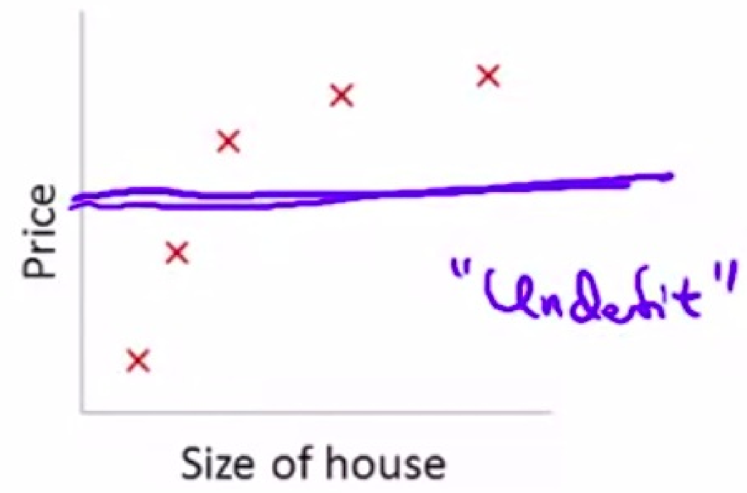
Lambda choose the tradoff between two different goals

i) We would like to fit training data well

ii) We would like to keep perameters small

The λ, or lambda, is the **regularization parameter**. It determines how much the costs of our theta parameters are inflated.

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



$$h_\theta(x) \longrightarrow \theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

$\underline{\theta_1}, \underline{\theta_2}, \underline{\theta_3}, \underline{\theta_4}$

$\theta_1 \approx 0, \theta_2 \approx 0$

$\theta_3 \approx 0, \theta_4 \approx 0$

$$\boxed{h_\theta(x) = \underset{\uparrow}{\theta_0}}$$

If lambda is chosen to be too large, it may smooth out the function too much and cause underfitting.

Regularization in Linear Regression

Repeat {

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

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

$$(j = \cancel{0}, 1, 2, 3, \dots, n) \}$$

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

$$(j = \cancel{0}, \underbrace{1, 2, 3, \dots, n}_{\text{regularized}})$$

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

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

The first term in the above equation, 1 - \alpha\frac{\lambda}{m}$1-\alpha\frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of \theta_jθ_j by some amount on every update. Notice that the second term is now exactly the same as it was before.