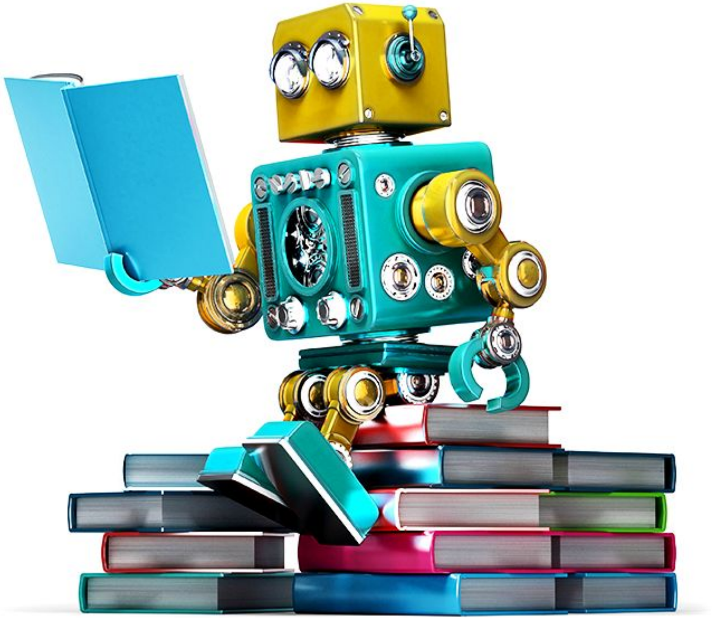
Machines are learning!

Mehrdad Mohammadian



Linear Regression
Logistic Regression
Support Vector Machines

🧐 **Problem**

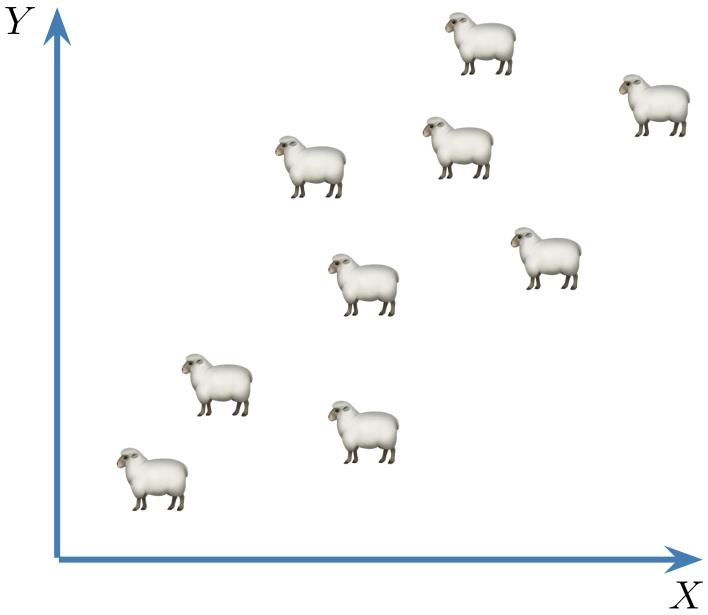
🤓 **Solution**

⚙️ **Implement**

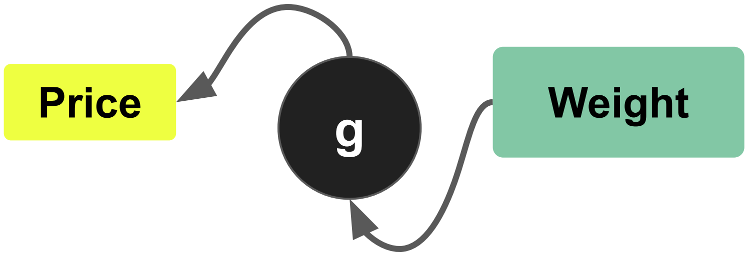
Linear Regression

Prediction!

	Weight (X)	Price (Y)
🐑	20	5.1
🐑	14	4
🐑	32	7.4
.	.	.
.	.	.
.	.	.
🐑	36	8.72



$$Y = g(X)$$



X : Feature (independent variable)

Y : Target (dependent variable)

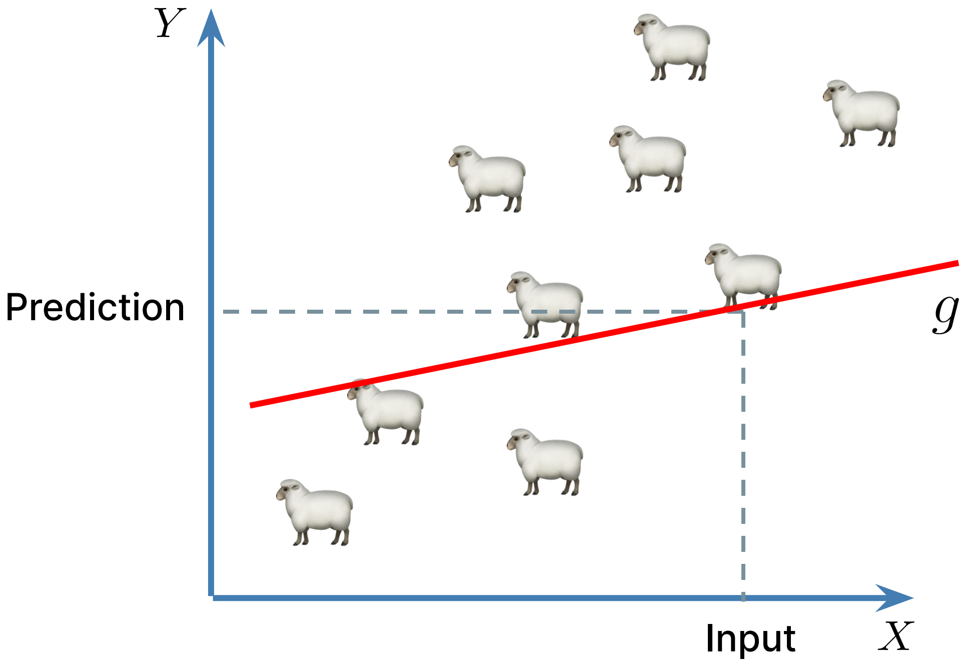
$$g(X) = \alpha X + \beta$$

α : Slope of the line

β : *Intercept*

Line (v1):

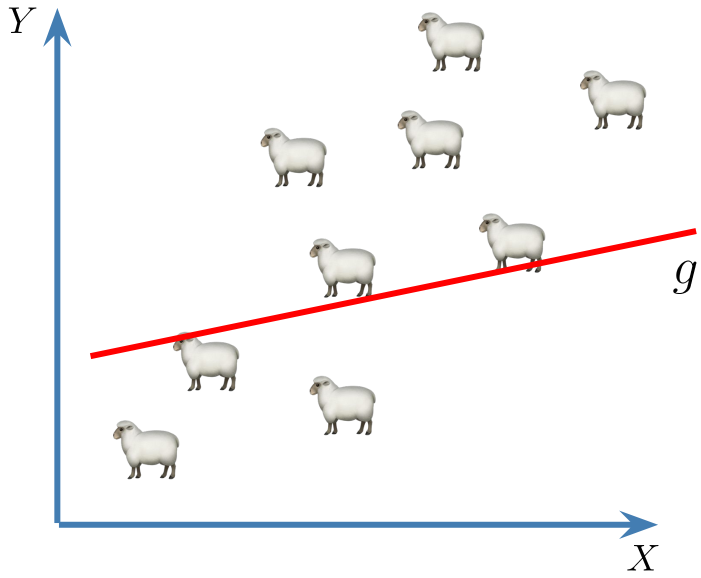
$$g(X) = 0.5X + 2$$

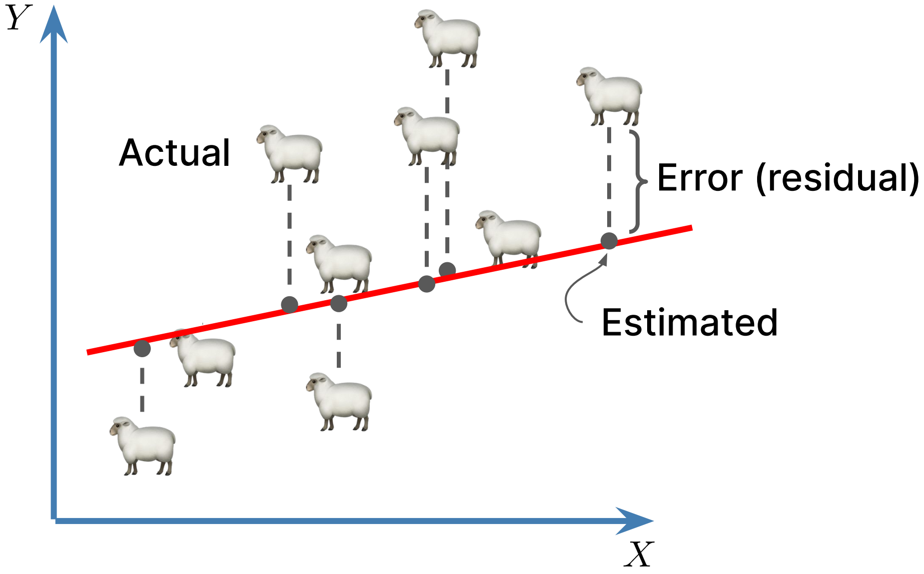


Is this good enough?!!

NOOOOOOOO! 😲

We need to minimize the error!
But, how?





Line equation: $g(X) = \theta_1 X + \theta_0$

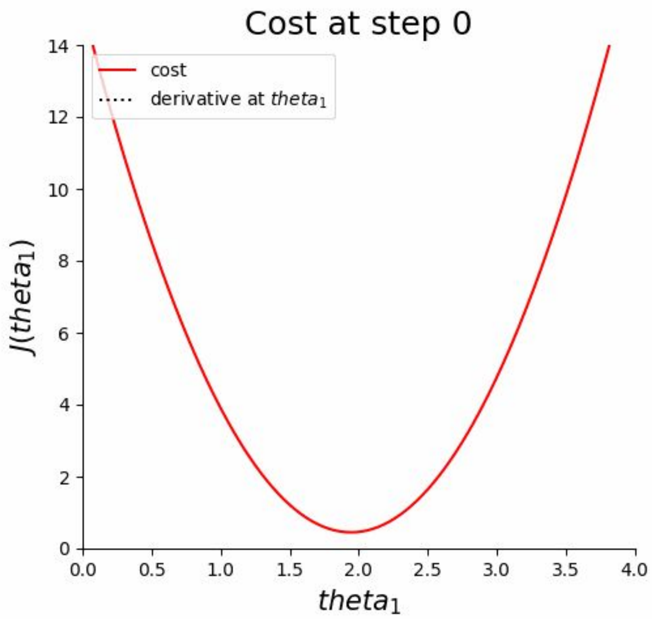
Parameters: θ_0, θ_1

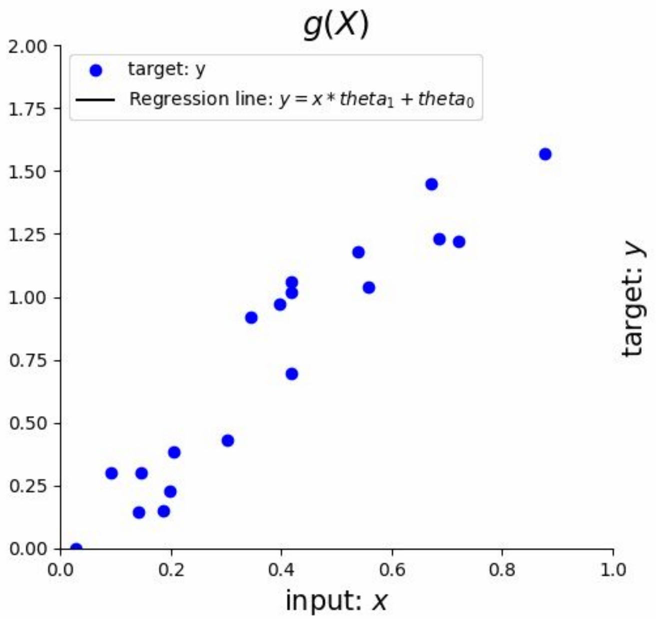
Cost function:

$$J(\theta_0, \theta_1) = \sum_{i=1}^{n} \frac{(g_\theta(x^{(i)}) - y^{(i)})^2}{n} = MSE$$

Goal:

minimize $J(\theta_0, \theta_1)$





Set random values for θ_0 and θ_1

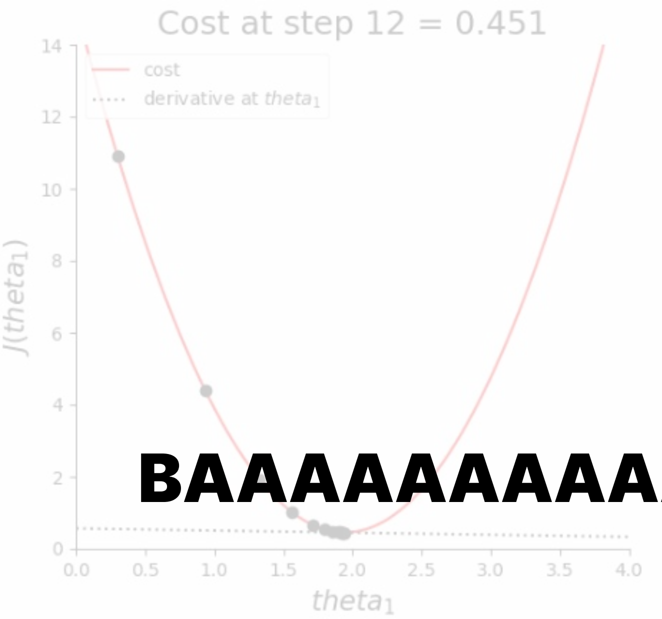
Changing θ_0 and θ_1 to minimize J

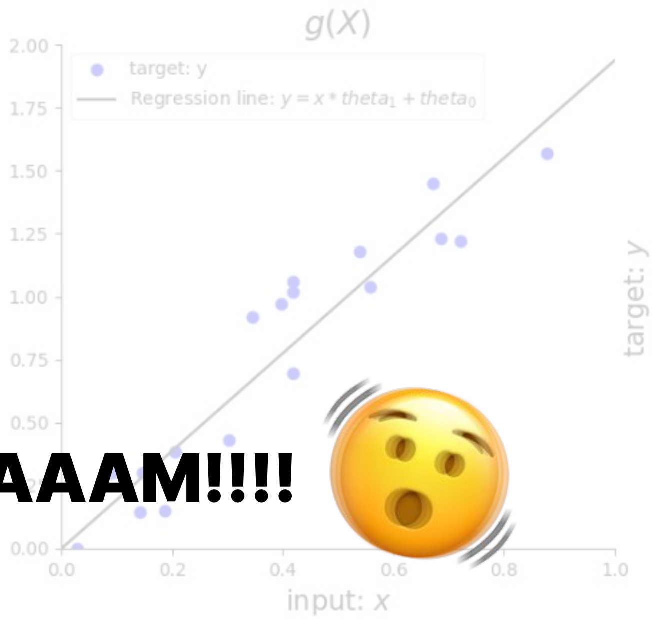
$$\text{repeat until convergence } \{$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

$$\}$$





BAAAAAAAAAAAM!!!!

Set random values for θ_0 and θ_1
Changing θ_0 and θ_1 to minimize J

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

}

Linear Regression: Single Variable

$$\underbrace{\boxed{\hat{y}}}_{\text{Predicted output}} = \underbrace{\beta_0 + \beta_1}_{\text{Coefficients}} \underbrace{\boxed{x}}_{\text{Input}} + \underbrace{\boxed{\epsilon}}_{\text{Error}}$$

Linear Regression: Multiple Variables

$$\boxed{\hat{y}} = \underbrace{\beta_0 + \beta_1}\boxed{x_1} + \ldots + \underbrace{\beta_p}\boxed{x_p} + \boxed{\epsilon}$$

Ridge (L2) Regression

Challenge:
Multicollinearity $\longrightarrow$ high correlation among predictor variables.
Multicollinearity can lead to unstable and unreliable coefficient estimates in linear regression.

Regularization:
It adds a penalty term to the cost function of linear regression.

$$\text{Cost Function} = \text{Least Squares Loss} + \underbrace{\alpha}_{\text{Regularization parameter}} \sum_{j=1}^{p} (\text{coefficient}_j^2)$$

Regularization parameter:
It shrinks the coefficients towards zero, reduces the effect of multicollinearity and stabilizes coefficient estimates.

Lasso (L1) Regression

Challenge: multicollinearity

Goal:
find the best-fitting linear model while preventing overfitting, a situation where the model captures noise and performs poorly on the test dataset.

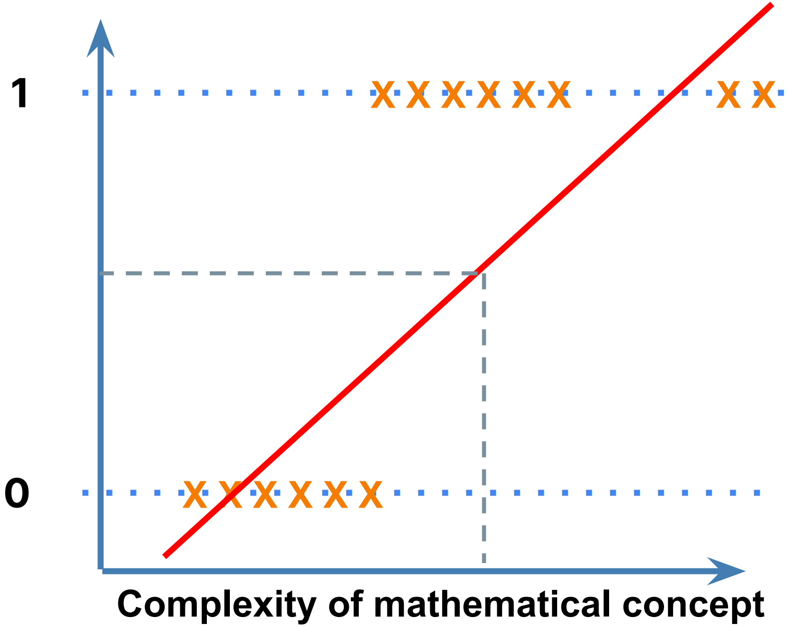
$$\text{Cost Function} = \text{Least Squares Loss} + \underbrace{\lambda}_{\text{Regularization parameter}} \times \sum_{j=1}^{\text{number of features}} |coefficient_j|$$

Traditional linear regression uses all predictors, risking overfitting in high-dimensional data. Higher Lasso penalty forces some coefficients to zero, automatically picking good features and dropping irrelevant ones from the model.

Logistic Regression

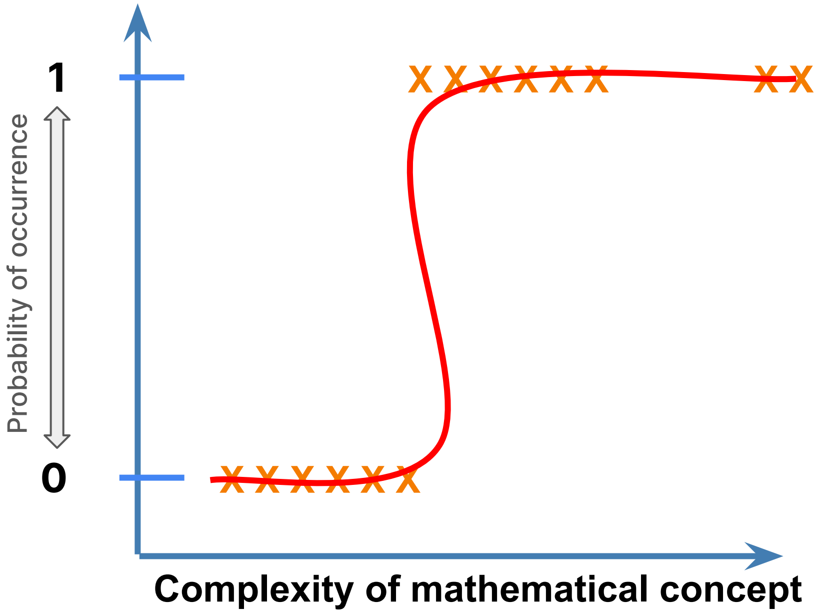
Classification!

Complexity (X)	Intuition (Y)
2	0
11	0
40	1
.	.
.	.
.	.
36	1



Classification!

Complexity (X)	Intuition (Y)
2	0
11	0
40	1
.	.
.	.
.	.
36	1



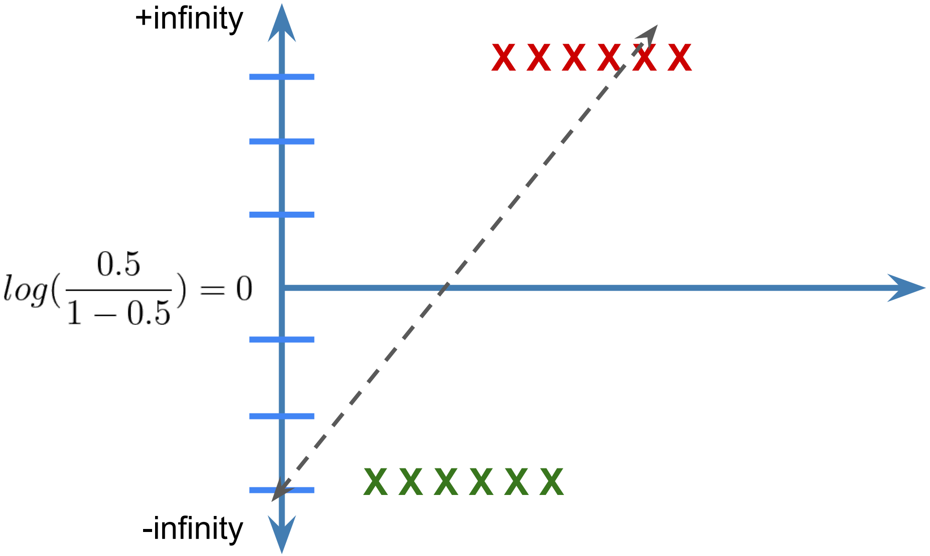
It is relatively easy to fit lines to things, and relatively hard to fit squiggles. So, we use the log() space to fit a line to the data and then translate that back to probabilities!!!

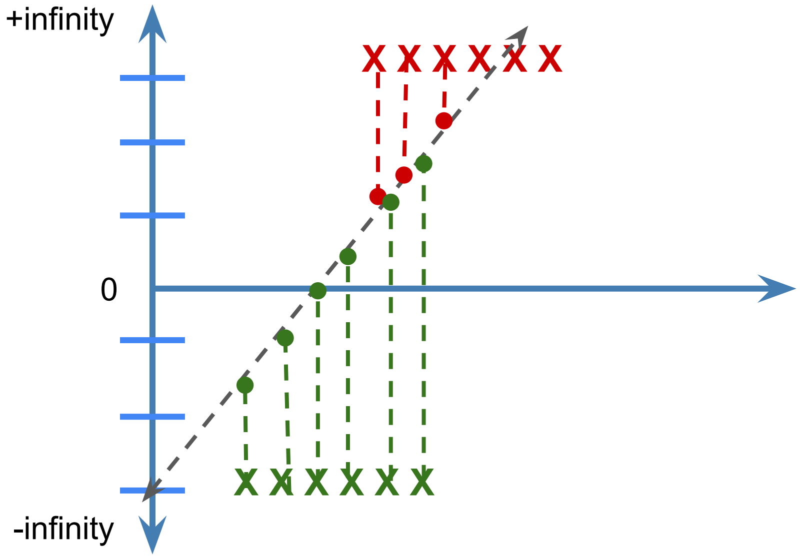
In y- axis of linear regression values can be any number
In y-axis of logistic regression values are bounded to be between 0 and 1

In order to get a linear relationship between predictors and target,
we transfer the y-axis from “probability of intuition” to “log(odds of intuition)”.

$$\log(\text{odds of intuation}) = \frac{p}{1-p}$$

p: prob. of the concept to be intuitive



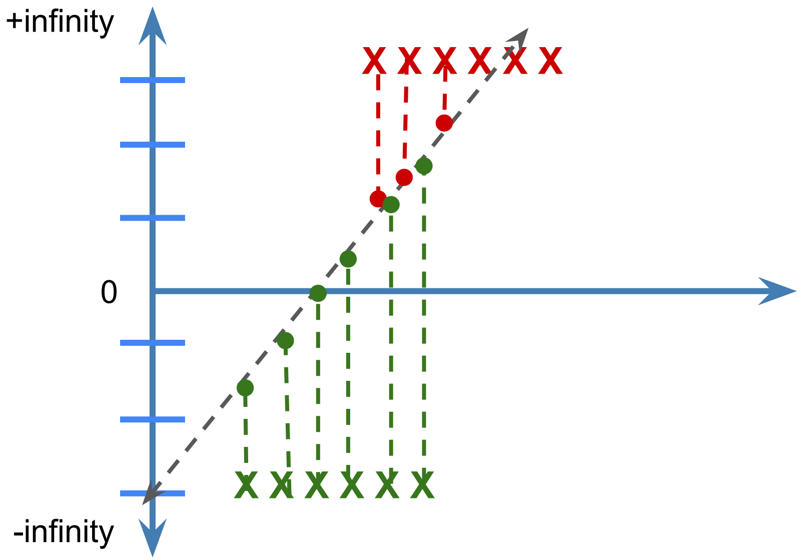


Problem!!!!! ☠️

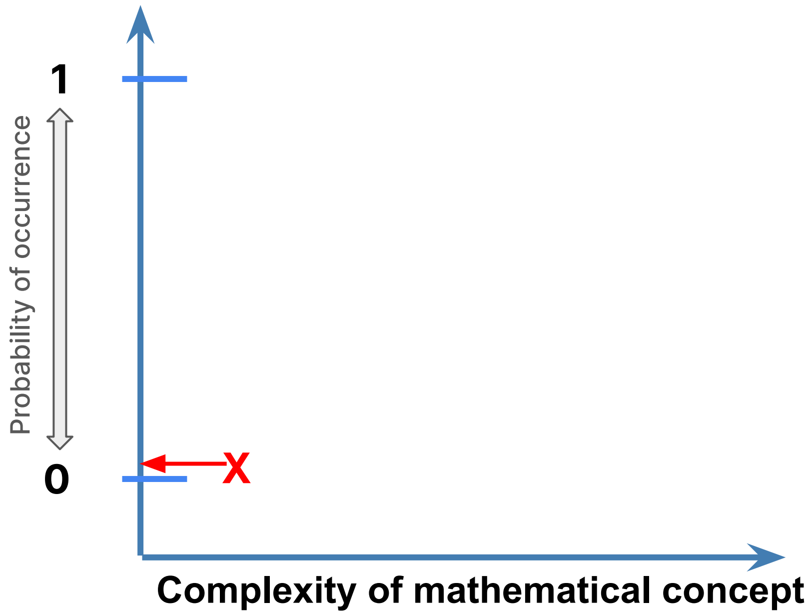
The y-axis transformation pushes the data points to **positive** and **negative** infinity … This means that residuals are also equal to positive and negative infinity.

Therefore, we can't use least-squares to find the best fitting line!

To solve that problem,
We fit the line by using Maximum Likelihood!



1. First we project data points to our candidate line. This gives a log(odds) value to each sample.

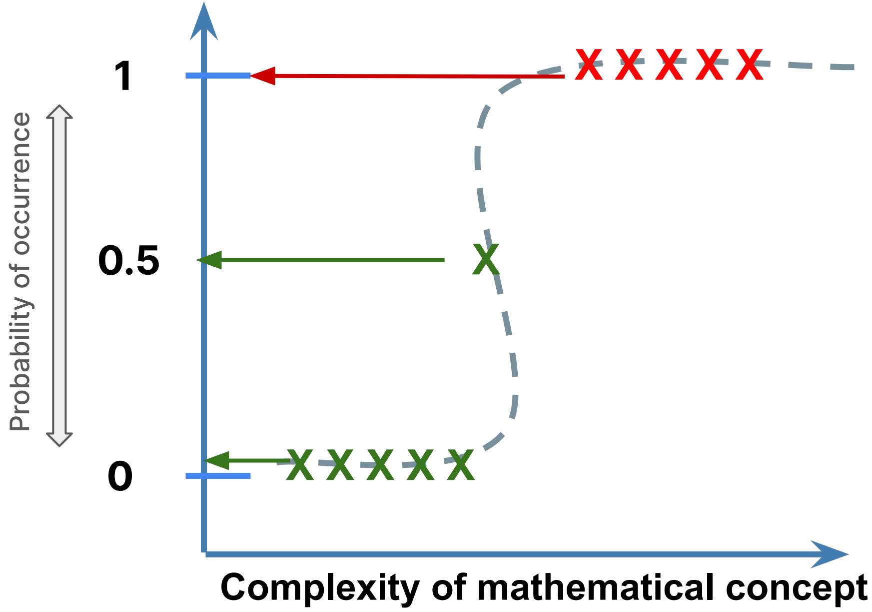


1. First we project data points to our candidate line. This gives a log(odds) value to each sample.

2. Then, we transform log(odds) to probabilities using below formula:

$$p = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

$$p = \frac{e^{-2.1}}{1 + e^{-2.1}} = 0.1$$

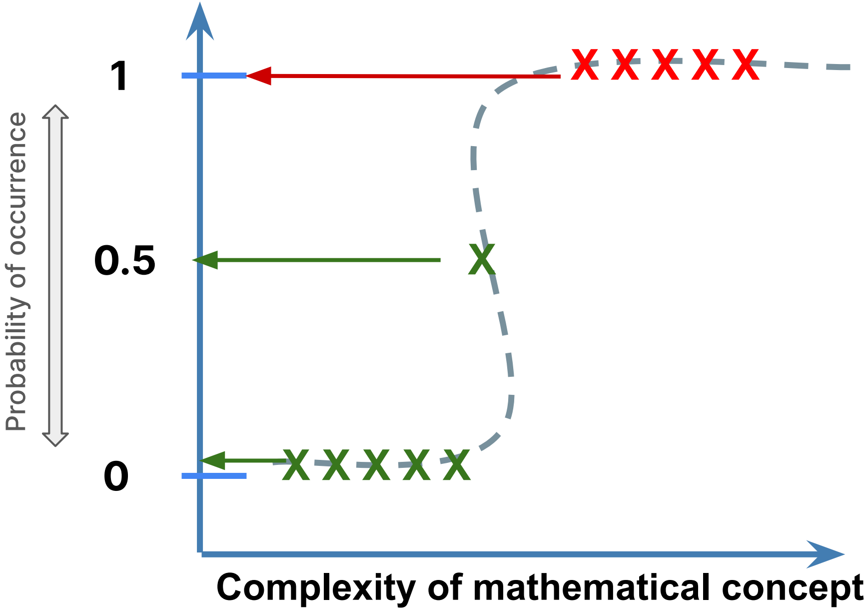


1. First we project data points to our candidate line. This gives a log(odds) value to each sample.

2. Then, we transform log(odds) to probabilities using below formula:

$$p = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

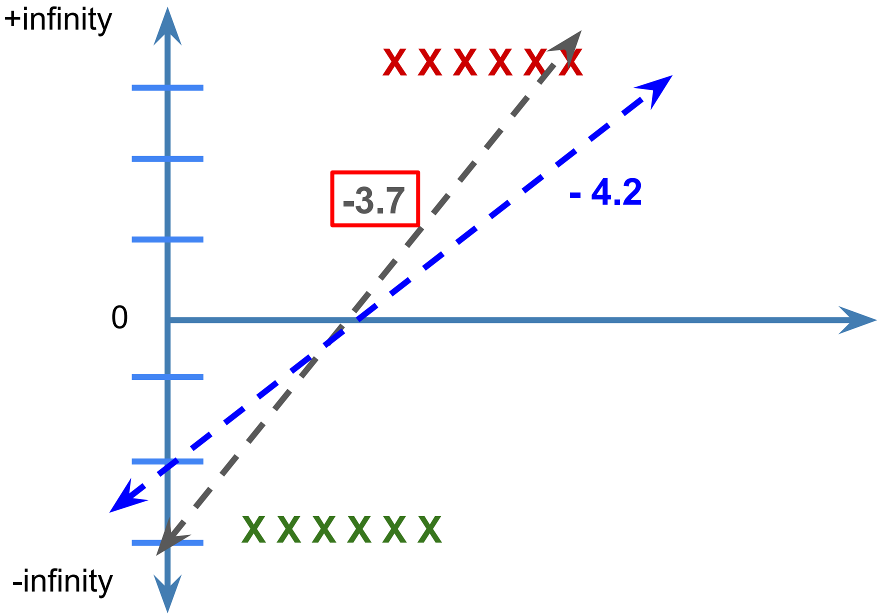
3. Calculate likelihood of transformed data points to find the shape of our squiggle.



Likelihood of a sample
==
predicted probability

Likelihood of all data points →
(0.91 * 0.9 * 0.92) *
((1 - 0.9) * (1 - 0.3) * (1 - 0.001)) = - 3.7

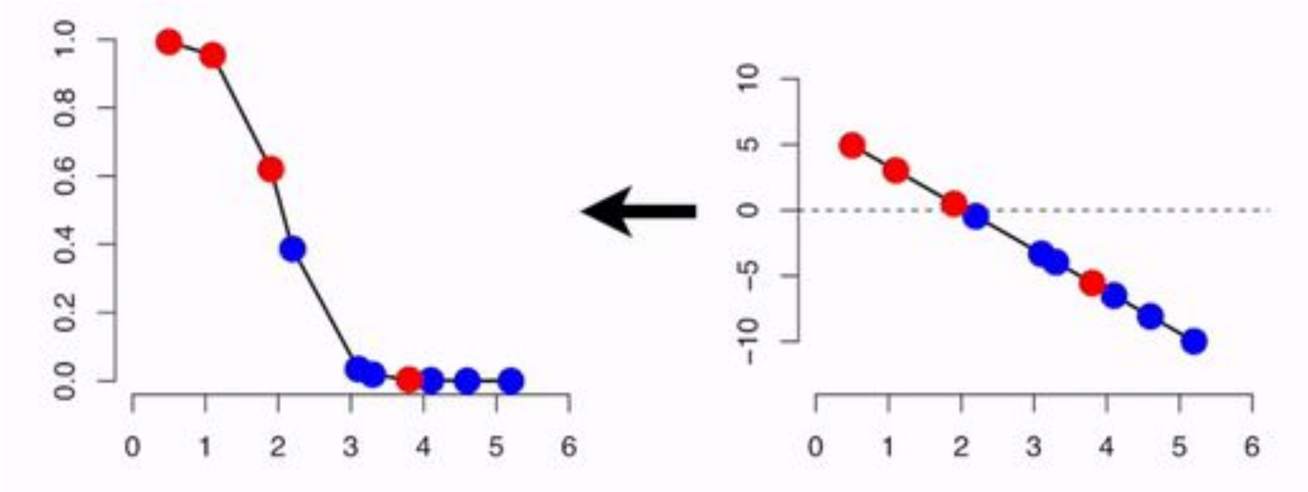
This also means that the likelihood of the log(odds) line is -3.7



5. Now, we rotate log(odss) line and again calculate the likelihood.

6. Finally, we choose a line that have the **biggest overall likelihood (maximum)**!

DOUBLE BAAAAAAAAAAAAAAM!!!!! 😲

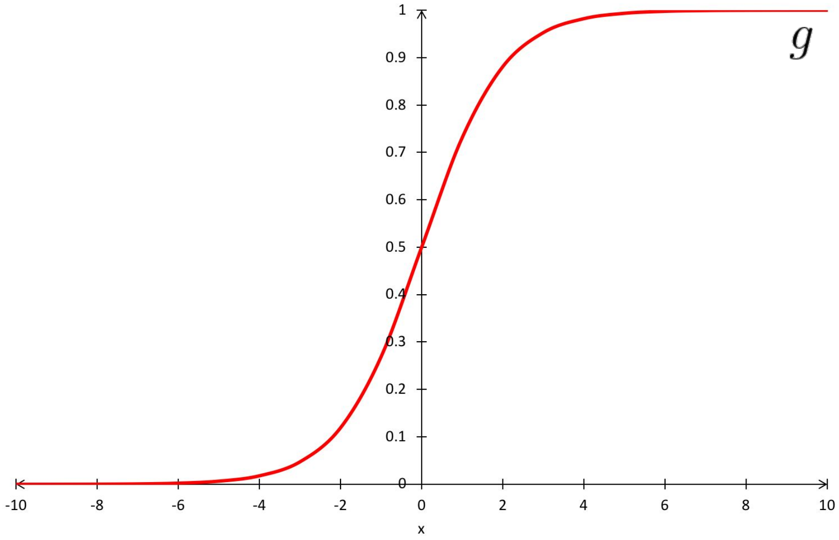


$$Y = \theta_0 + \theta_1 \cdot X_1 \cdots \theta_n \cdot X_n$$

$$g(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\theta_0+\theta_1 \cdot X_1 \cdots \theta_n \cdot X_n)}}$$

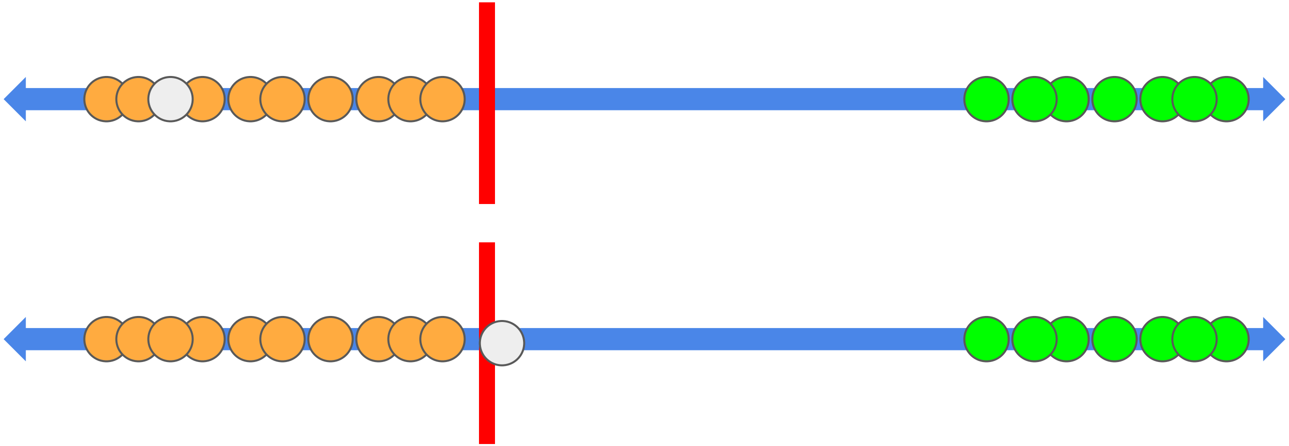
$$g(z) = P(y=1|x;\theta)$$

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



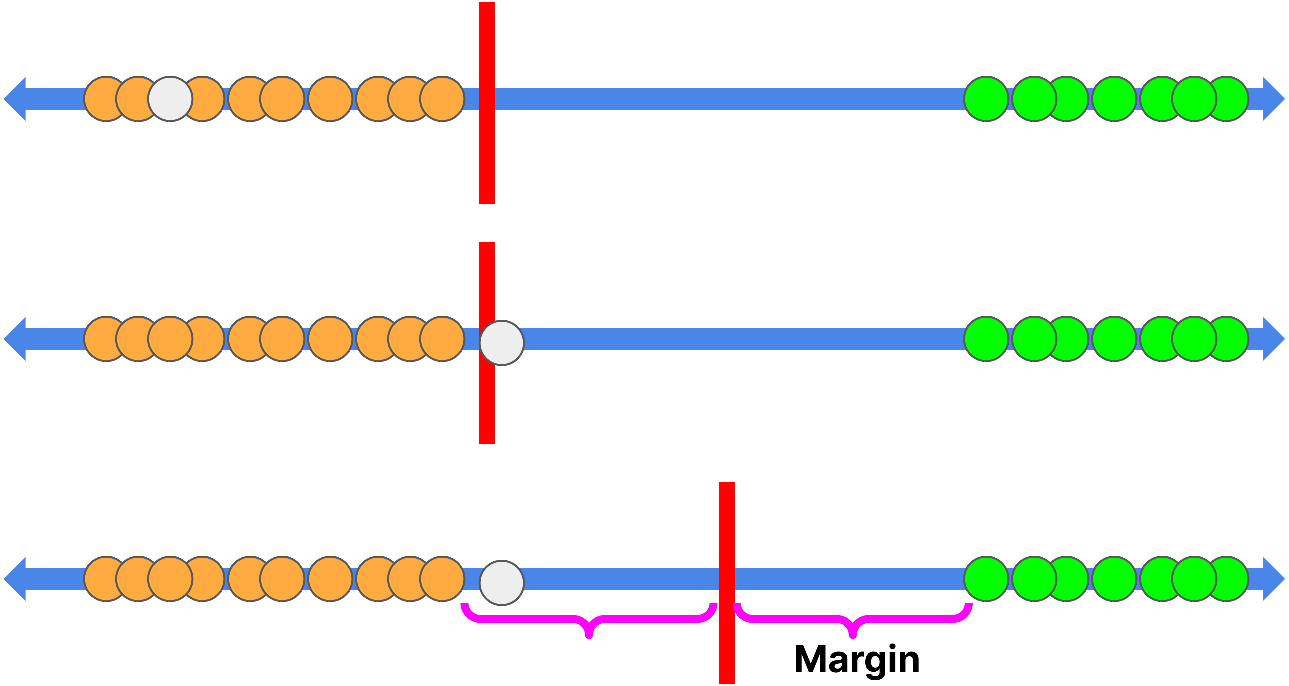
Support Vector Machines

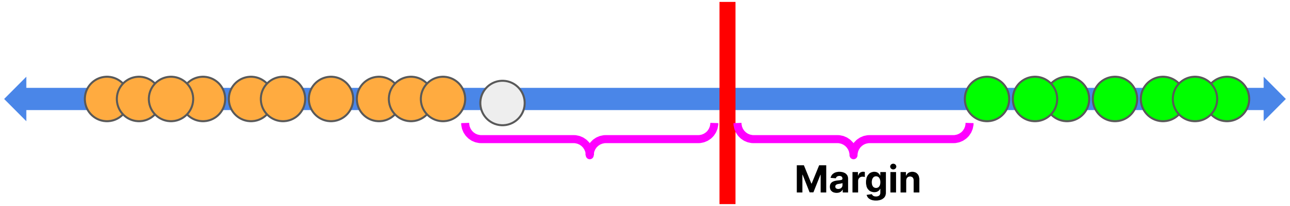
White ball is classified!



Is it logical to classify the white ball as green?!

White ball is classified!





When we use a threshold that give the biggest margin
We are using a maximal margin classifier!!!

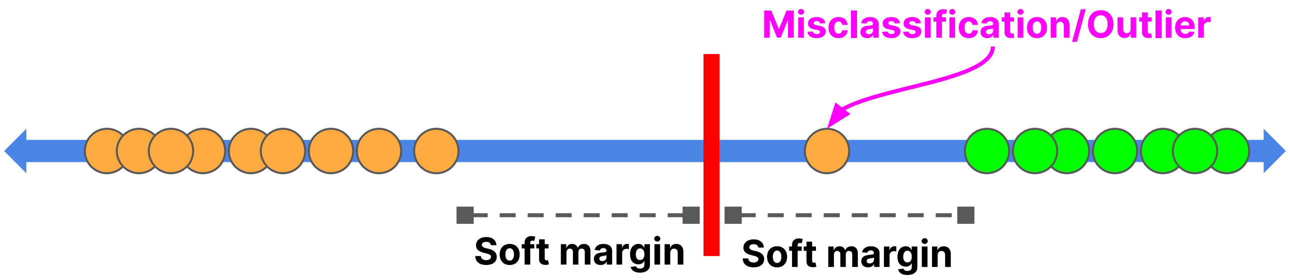
In this case, the middle point is the largest margin.

Problem!!!!
Maximal margin classifier are sensitive to outliers!

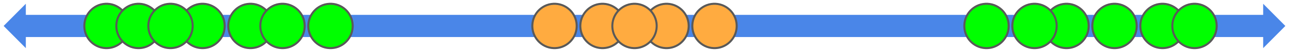
To solve this:
We should allow misclassifications.
A margin that allows misclassification are called "Soft Margin”.

In order to find the best soft margin, we need to do “Cross Validation”!

When we use a soft margin to determine the best threshold to classify observations, we are using a "Soft Margin Classifier" a.k.a "Support Vector Classifier"!



What if our data would be like this?

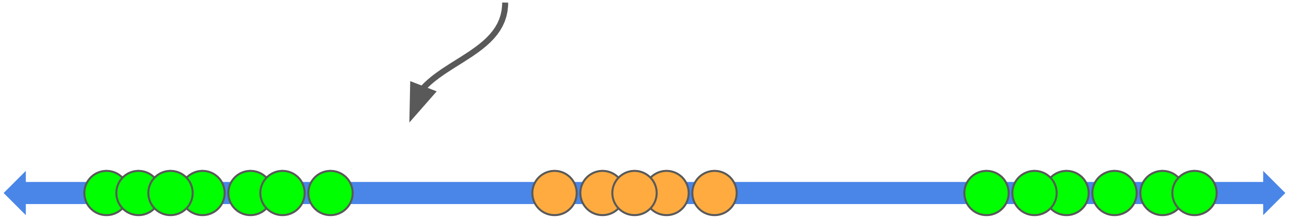


No matter where you set the margins …
Support vector classifier cannot handle it!

This is where we need “Support Vector Machines”!!!!

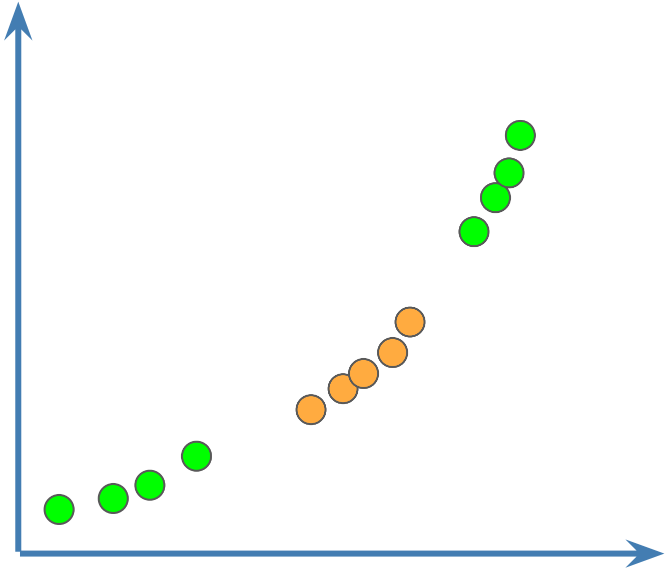
The main idea behind the SVM:

1. Start with the data in a lower dimension
   In this case, a 1-dimensional space



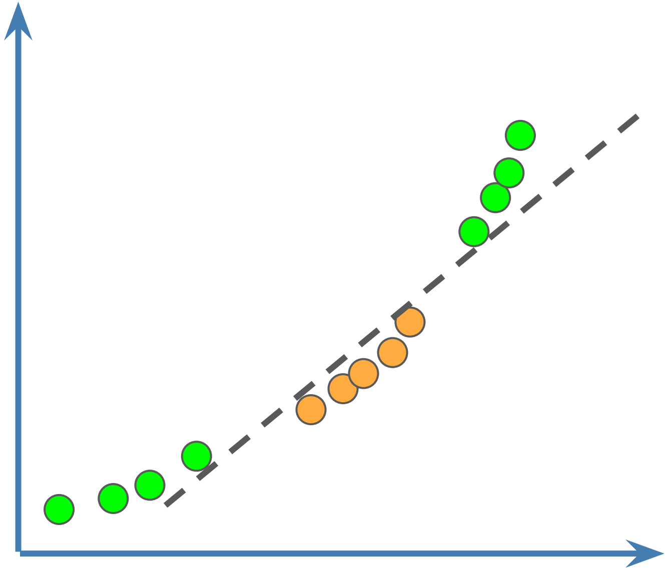
The main idea behind the SVM:

2. Move the data to a higher dimension.
   In this case, a 2-dimensional space



The main idea behind the SVM:

3. **Find the support vector classifier that can separate the higher dimension data into two groups.**

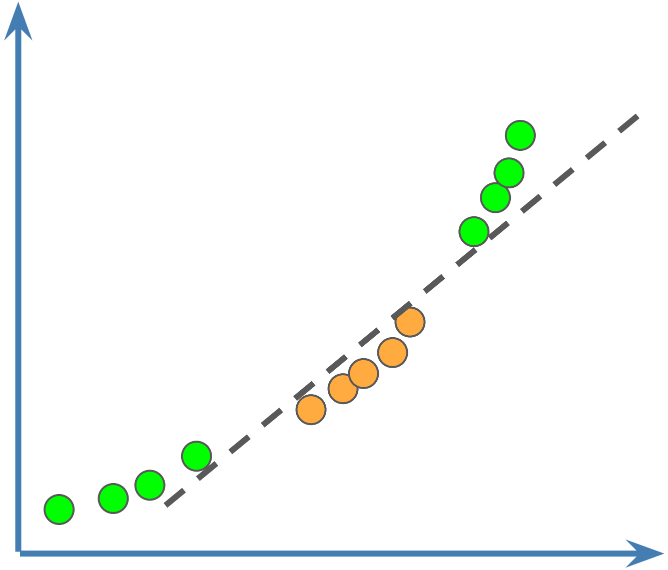


BAAAAAAAM!

How we decide how to transform the data to a higher dimension?

In below case, we used power 2 of the x-axis values to create the y-axis.

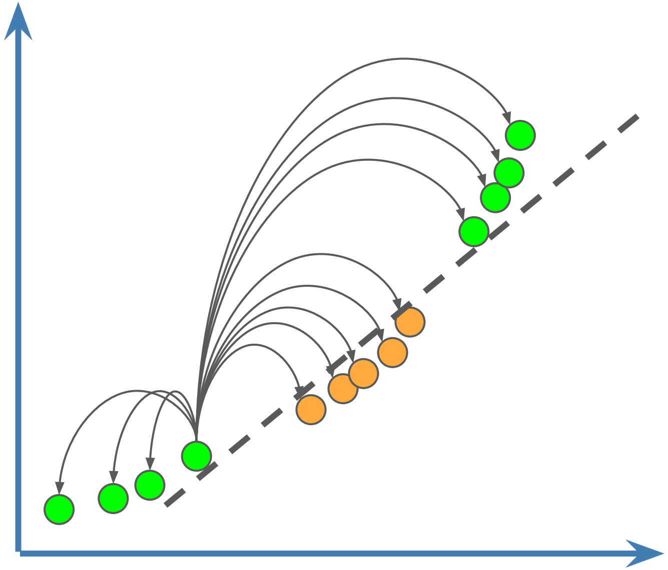
But, what about square root? Or power 3?



By using Kernels!!!!

For example: **"Polynomial Kernel Function"**

It increases the dimension by setting **d** parameter (degree of the polynomial)
In below case, d is equal to 2. To be exact, it computes 2-dimensional relationships between each sample pair.



Thank you for your time!