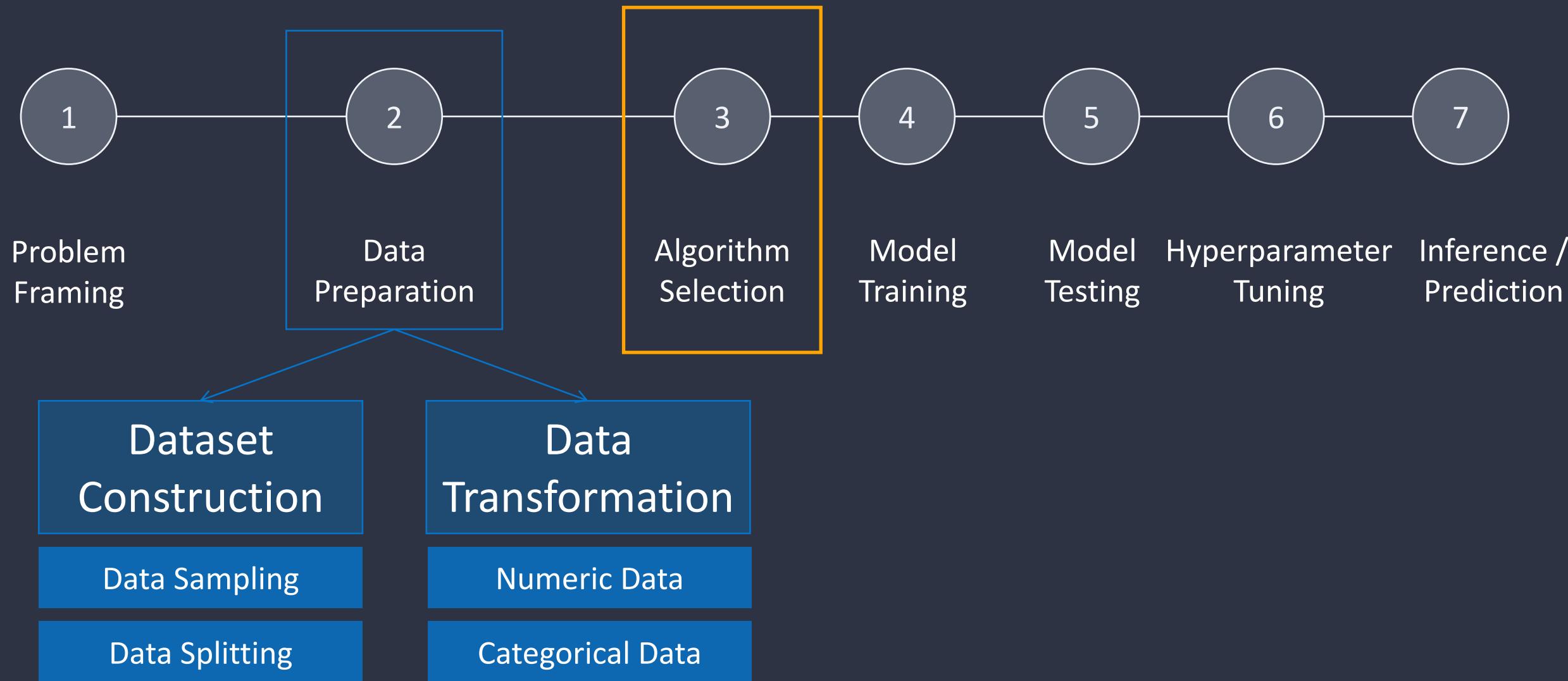


**COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING**

# Machine Learning Algorithms

Dr Yang Long

# Last Lecture



# Lecture Overview

1. Machine Learning Models and Algorithms
2. Overview of Machine Learning Algorithms
3. Supervised Learning
4. Regression versus Classification

# 1. Machine Learning Models and Algorithms



machine learning algorithm

to train

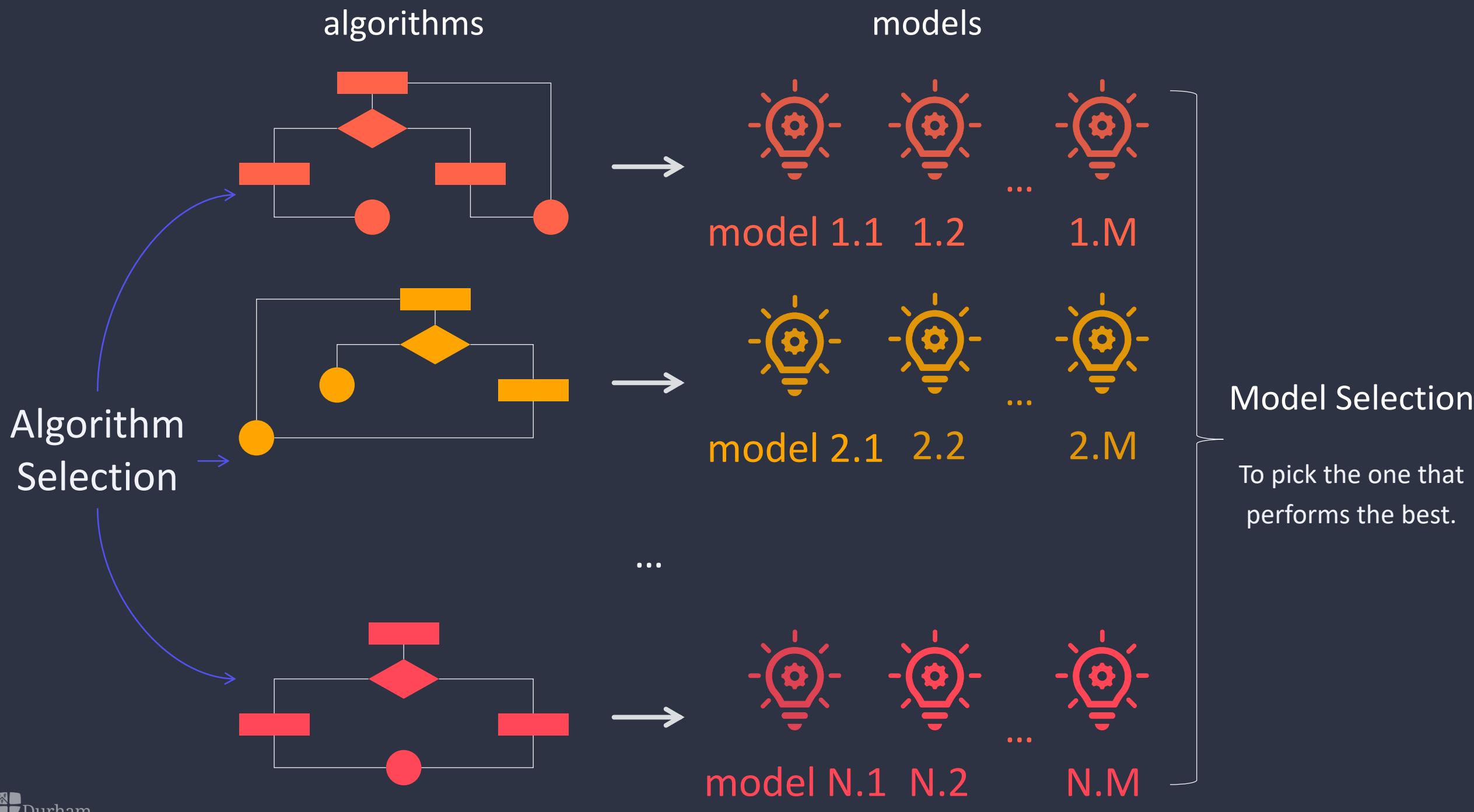
machine learning model



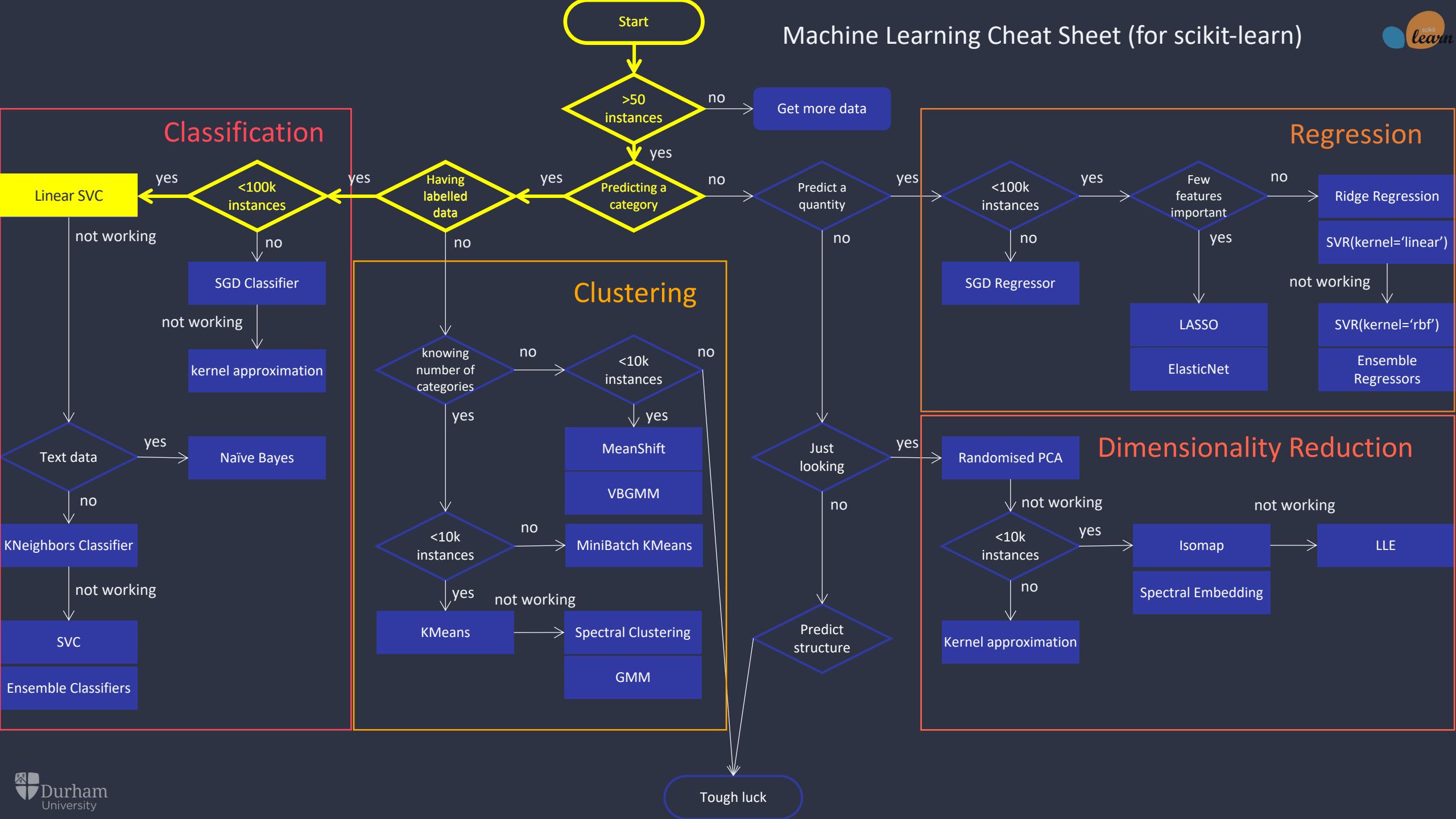
unseen data



inference / predictions



# Machine Learning Cheat Sheet (for scikit-learn)



Classification

Regression

Clustering

Dimensionality Reduction

# Classification / Regression / Clustering / Dimensionality Reduction

- Classification models are used to predict a (discrete) class label for examples/instances in the problem domain, e.g. diagnostics, fraud detection.
- Classification algorithms need to learn from a training set with many examples/instances of inputs (a set of features) and outputs (labels) to create a classification model.
- Supervised Learning.

# Classification / Regression / Clustering / Dimensionality Reduction

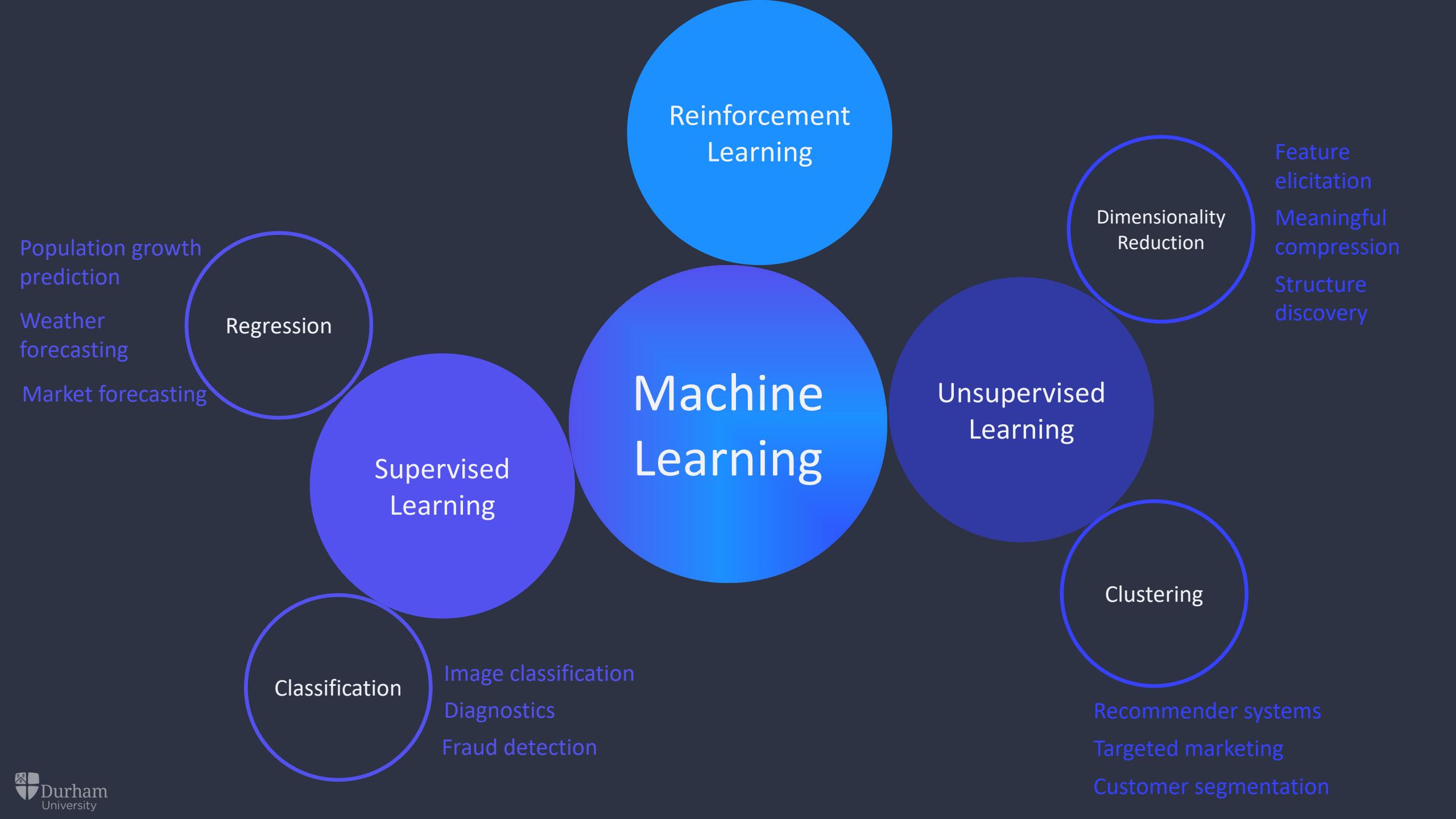
- Regression models are used to predict a specific value in a continuous distribution, e.g. weather forecasting, population growth prediction.
- Regression algorithms need to learn from a training set with many examples/instances of inputs and outputs to create a regression model.
- Supervised Learning.

# Classification / Regression / Clustering / Dimensionality Reduction

- Clustering models are used to discover interesting patterns in data, e.g., customer segmentation, recommender systems.
- Clustering algorithms need to learn from a training set with many examples/instances of inputs and outputs (unlabelled) to create a clustering model.
- Unsupervised learning.

# Classification / Regression / Clustering / Dimensionality Reduction

- Dimensionality Reduction models are used to decrease the number of input features, transforming data from high to low dimensional space, e.g. pre-processing data, big data visualisation, feature elicitation.
- Dimensionality reduction simplifies the dataset with fewer features, which is very useful, as more input features usually make a predictive modelling task more challenging and may cause problems e.g. overfitting.
- Unsupervised learning.



## 2. Overview of Machine Learning Algorithms

# Grouping Machine Learning Algorithms

Based on what kind of inference we want to see in data...

Classification

Clustering

Regression

Dimensionality Reduction

Useful for deciding which algorithms to try for a particular machine learning problem

# Grouping Machine Learning Algorithms

Based on what kind of inference we want to see in our data...

Classification

Regression

Clustering

Dimensionality  
Reduction

Useful for deciding which algorithms to try for a particular machine learning problem

In a particular problem domain,

- What do we want to do with our data set?
- What do we need to do in order to pre-process our data for training / fitting the most appropriate model?

# Grouping Machine Learning Algorithms

Based on the function similarity...

Bayesian Algorithms

Regression Algorithms

Association Rule Learning Algorithms

Regularisation Algorithms

Dimensionality Reduction Algorithms

Instance-based Algorithms

Ensemble Algorithms

Clustering Algorithms

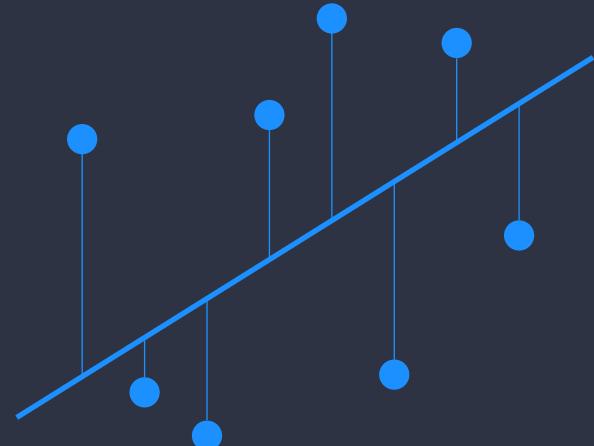
Artificial Neural Network Algorithms

Tree-based Algorithms

Deep Learning Algorithms

# Regression Algorithms

- To find an approximation function mapping input variable X to continuous output variable y.
- Output variable y can be real values i.e. integers or floating-point values.
- To predict quantities, height, weight, size, etc.
- Learning in iterations, using a measure of error in the predictions made by the model.



The most popular regression algorithms include:

- Linear Regression
- Logistic Regression
- Stepwise Regression
- Ordinary Least Squares Regression (OLSR)
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)

# Regularisation Algorithms

- Regularisation is a form of regression.
- Regularising / shrinking the coefficient estimates towards zero.
- To penalise models based on their complexity and flexibility.
- Favouriting simpler models to prevent the risk of overfitting.

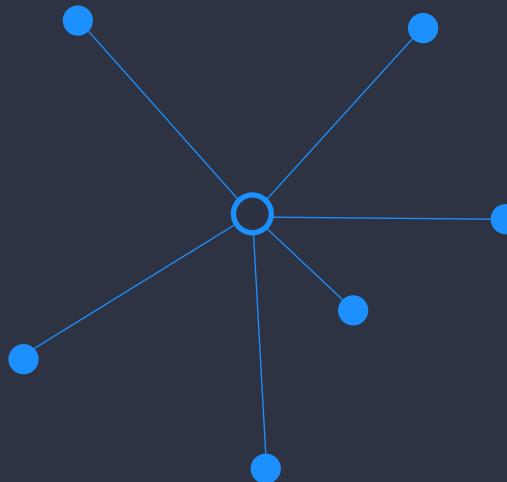


The most popular regularisation algorithms include:

- Ridge Regression
- Elastic Net
- Least-Angle Regression (LARS)
- Least Absolute Shrinkage & Selection Operator (LASSO)

# Instance-based Algorithms

- To compare new problem instances with instances seen in training, rather than performing explicit generalisation as what model-based algorithms do.
- Storing all instances in training process, and then predictions are made by comparing the new instance to the old ones using similarity measures (risk of overfitting).
- Can adapt to previously unseen data – simply store new instances or drop old instances.

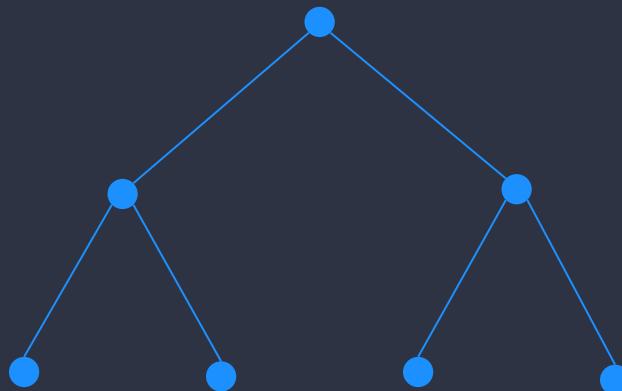


The most popular instance-based algorithms include:

- k Nearest Neighbours (kNN)
- Self-Organizing Map (SOM)
- Learning Vector Quantization (LVQ)
- Locally Weighted Learning (LWL)

# Tree-based Algorithms

- To construct tree-based models of decisions made based on actual values of attributes in the data.
- The decision flow goes along a tree structure until it reaches the leaf node, i.e. the final decision for a given instance.
- Straightforward to interpret even for non-technical people.
- Large decision trees are complex, time-consuming and less accurate in predicting outcomes.

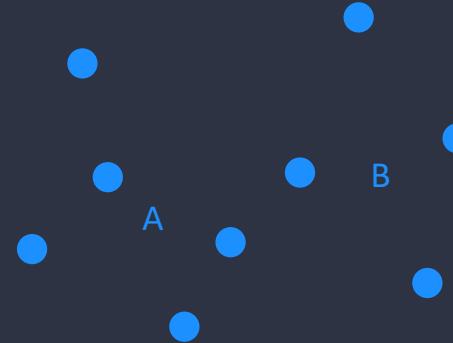


The most popular tree-based algorithms include:

- Classification and Regression Tree (CART)
- Conditional Decision Tree
- Iterative Dichotomiser 3 (ID3)
- Chi-squared Automatic Interaction Detection (CHAID)
- Decision Stump

# Clustering Algorithms

- To use inherent structures in the data to automatically group data for maximum commonality
- No explicit labels for clusters/groups of instances.
- To group instances based on their centroid, density, distribution, or hierarchy
- Unsupervised learning to find natural groups in the feature space of input data.



The most popular clustering algorithms include:

- K-Means
- K-Medians
- Mean-shift
- Hierarchical Clustering
- Fuzzy Clustering
- Expectation Maximisation (EM)

# Bayesian Algorithms

- To create models that explicitly apply Bayes' Theorem.



The most popular Bayesian algorithms include:

- Naïve Bayes
- Gaussian Naïve Bayes
- Multinomial Naïve Bayes
- Averaged One-Dependence Estimators (AODE)
- Bayesian Belief Network (BBN)
- Bayesian Network (BN)

# Association Rule Learning Algorithms

- A rule-based to discover interesting relations between variables in large databases.
- Seeking strong rules discovered in databases using some measures of interestingness.



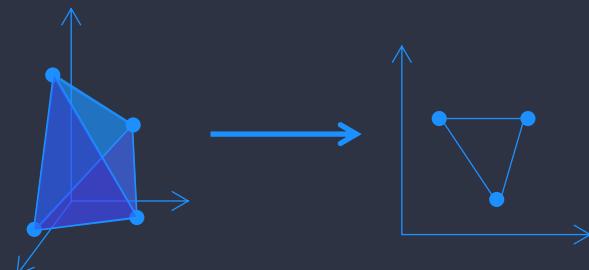
The most popular ARL algorithms include:

- Apriori algorithm
- Eclat algorithm
- FP-growth algorithm

# Dimensionality Reduction Algorithms

- To find the inherent structure in data in order to reduce the number of random variables.
- Feature selection, to find the subset of original set of variables/features to get a smaller subset for a particular problem.
- Feature extraction, to reduce data in order to have a space with lesser number of dimensions

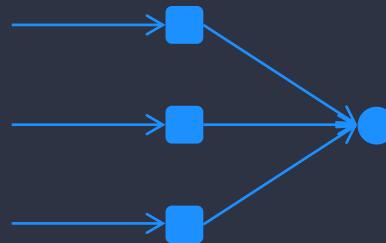
The most popular DR algorithms include:



- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)
- Quadratic Discriminant Analysis (QDA)
- Principal Component Regression (PCR)
- Partial Least Squares Regression (PLSR)
- Multidimensional Scaling (MDS)

# Ensemble Algorithms

- Combining several machine learning models into one optimal predictive model.
- To decrease variance (bagging) and bias (boosting), or to improve predictions (stacking).

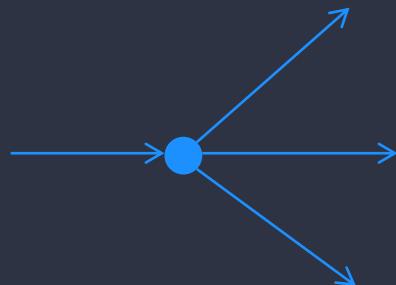


The most popular Ensemble algorithms include:

- AdaBoost
- Boosting
- Bootstrap aggregating (Bagging)
- Gradient boosted decision tree (GBDT)
- Gradient boosting machine (GBM)
- Random Forest
- Stacked Generalization (blending)

# Artificial Neural Network Algorithms

- Inspired by structure & function of biological neural networks that constitute animal brains.
- A neural network is based on a collection of connected units or nodes called artificial neurons, which model the neurons in a biological brain.
- Each neuron can transmit a signal to other neurons.

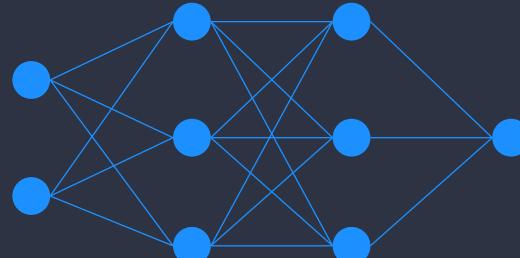


The most popular ANN algorithms include:

- Perceptron
- Multilayer Perceptron (MLP)
- Radial Basis Function Network (RBFN)
- Feedforward Algorithm
- Back Propagation
- Hopfield Network

# Deep Learning Algorithms

- Extension of Artificial Neural Networks.
- To build much larger and much more complex neural networks.
- Good at dealing with unstructured dataset such as texts, images, audios and videos.



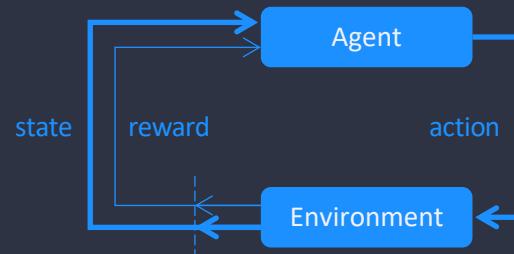
The most popular DL algorithms include:

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)
- Stacked Auto-Encoders

# Reinforcement Learning Algorithms

- Concerned with how AI agents take actions in environment to maximise cumulative rewards.
- Agents receive rewards/punishment depending upon how they take actions in certain states.
- An environment is typically stated in the form of a Markov decision process (MDP).
- Two types of RL: model-based (using transition/reward function) and model-free.

The most popular RL algorithms include:



- Model-based
  - Learn the Model: World Models, I2A, MBMF, MBVE
  - Given the Model: AlphaZero
- Model-free
  - Policy Optimisation: Policy Gradient, A2C/A3C, PPO
  - Stacked Auto-Encoders: DQN, C51, OR-DQN, HER

# Machine Learning algorithms



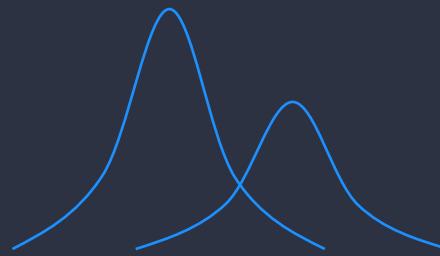
Regression



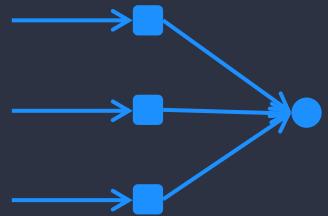
Regularisation



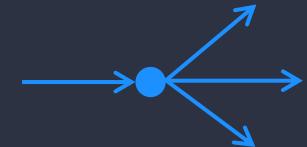
Clustering



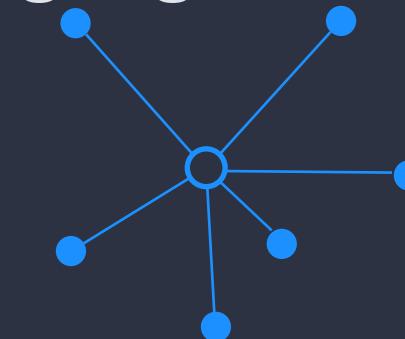
Bayesian



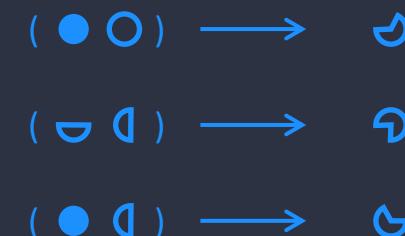
Ensemble



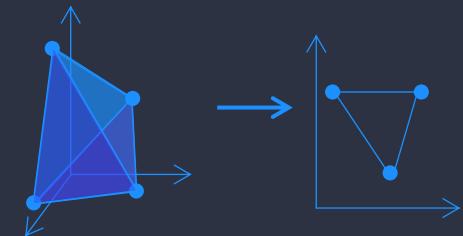
Neural Network



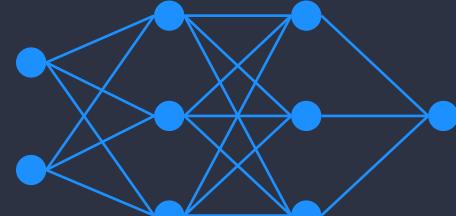
Instance-based



Associated Rule Learning



Dimensionality Reduction

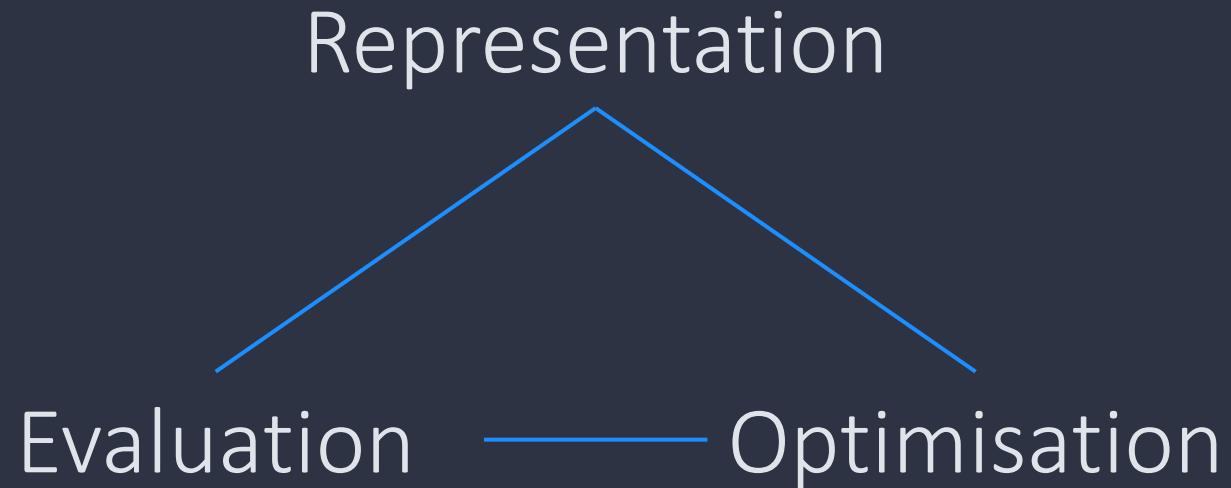


Deep Learning



Reinforcement Learning

# Decomposition of Machine Learning algorithms



Pedro Domingos, a CS professor at the University of Washington  
<https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>

# Decomposition of Machine Learning algorithms

## Representation

The hypothesis space of the learner, represented in some formal language that the computer can handle.

## Evaluation

An evaluation function (also called objective function or scoring function) to distinguish good models from bad ones.

## Optimisation

A method to search among the models in the language for the highest-scoring one.

Pedro Domingos, a CS professor at the University of Washington

<https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>

# Decomposition of Machine Learning algorithms

Representation	Evaluation	Optimisation
Instances	Accuracy / error rate	Combinatorial optimisation
k-Nearest Neighbors	Precision & recall	Greedy search
Support Vector Machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naïve Bayes	Posterior probability	Continuous optimisation
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

# Machine Learning Algorithms in a Nutshell

Tens of thousands of machine learning algorithms

Hundreds new each year

Each machine learning algorithm has three components

Representation

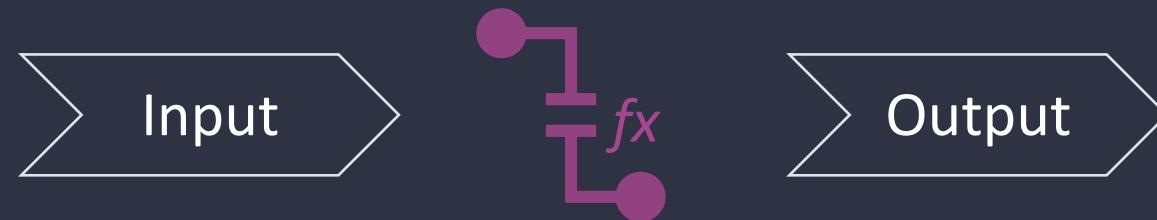
Evaluation

Optimisation

# 3. Supervised Learning

# Supervised Learning

- Learning a function, which is inferred from labelled training data containing a set of training examples, to map an input to an output.



- Called "supervised" learning, not because there must be someone standing beside the machine to supervise how the machine should work, but because there are labelled examples for learning algorithm to learn from or train on.

# Supervised Learning Task

m n-dimensional feature vectors (each vector has n elements)

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^n \quad i \in (1, 2, \dots, m)$$

A collection of corresponding labels y. (m is the number of examples)

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

# Supervised Learning Task

		feature vectors					label scalar	
		1 total credit	2 short term loan	3 credit utilisation	...	n missing payments	n+1 credit score	
1		$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	...	$x_n^{(1)}$	$y^{(1)}$	
2		$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$	$y^{(2)}$	
3		$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	...	$x_n^{(3)}$	$y^{(3)}$	
...		...	...	...	...	...	...	
m		$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	...	$x_n^{(m)}$	$y^{(m)}$	
		features		labels				

# Supervised Learning Task

- The goal is to build a model, which is just a real-valued function.

credit score =  $f$  ( total credit, short term loan, credit utilisation, missing payments)

# Supervised Learning Task

- The goal is to build a model, which is just a real-valued function.

credit score =  $f$  ( total credit, short term loan, credit utilisation, missing payments)

$$\hat{y} = h(X)$$

- Hypothesis function
  - is defined on a set of **feature vectors X**
  - to make certain conclusions about  $y$ , e.g. to make a **prediction of  $y$** , i.e.  $\hat{y}$
- In any supervised learning tasks, there will be a hypothesis  $h(X)$  created by the machine learning algorithm, and it can be represented as a **function  $h$  of  $X$** .

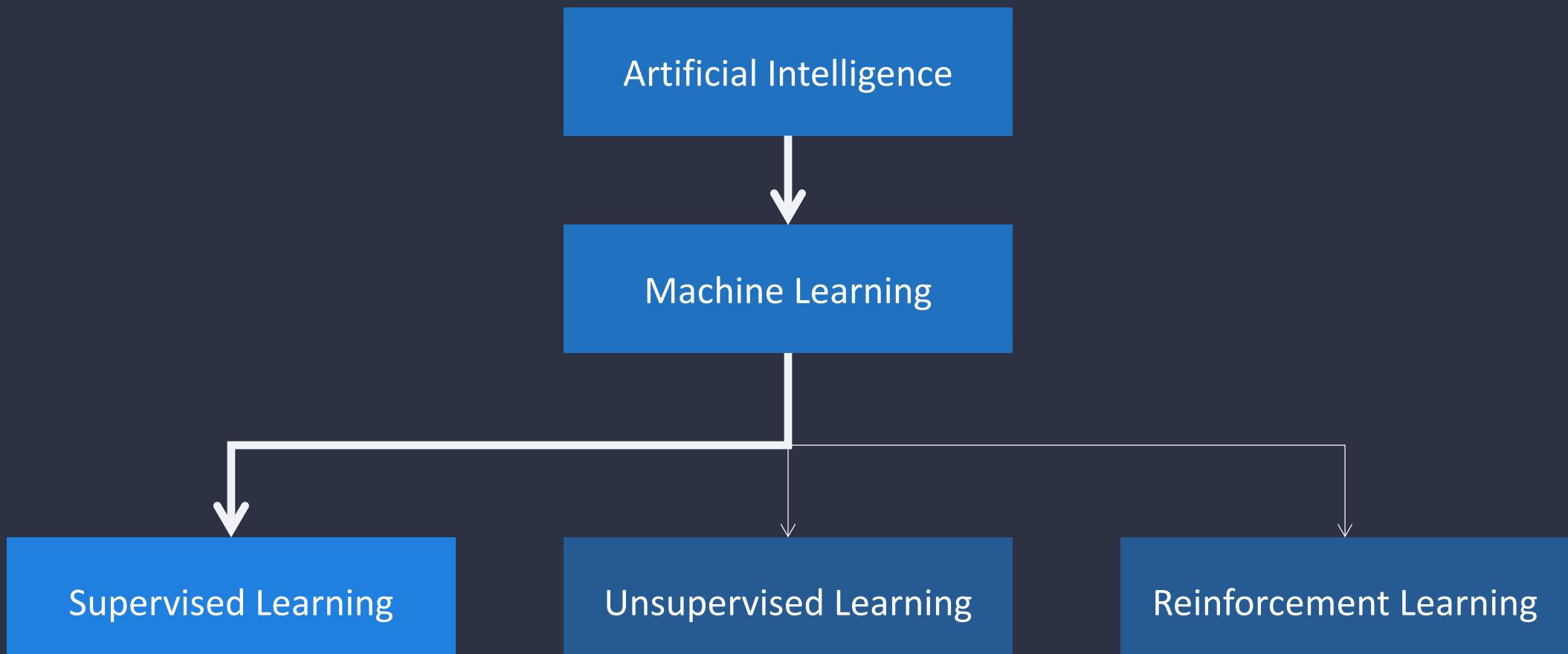
# Plotting function on graph

- E.g. use credit utilisation to predict credit score

# Plotting function on graph

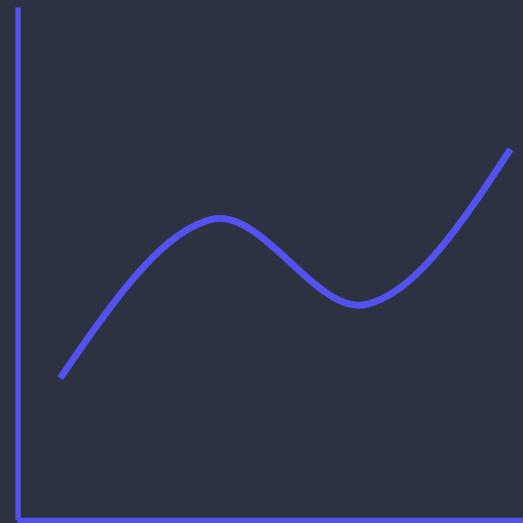
- E.g. use credit utilisation and missing payments to predict credit score

# 4. Regression versus Classification

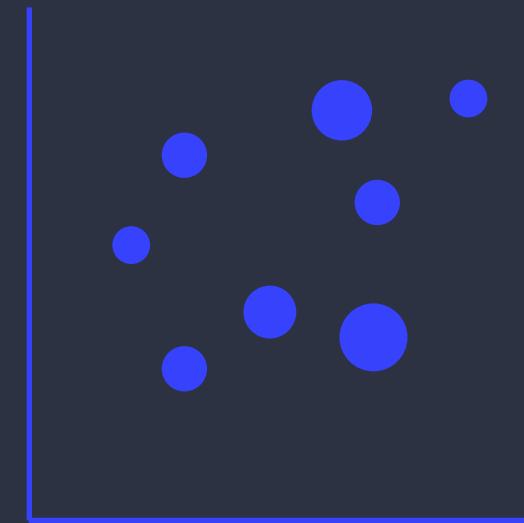


# Two tasks of supervised learning

- Regression: predicting a continuous (numerical) quantity output for an input.
- Classification: predicting a discrete (categorical) class label output for an input.



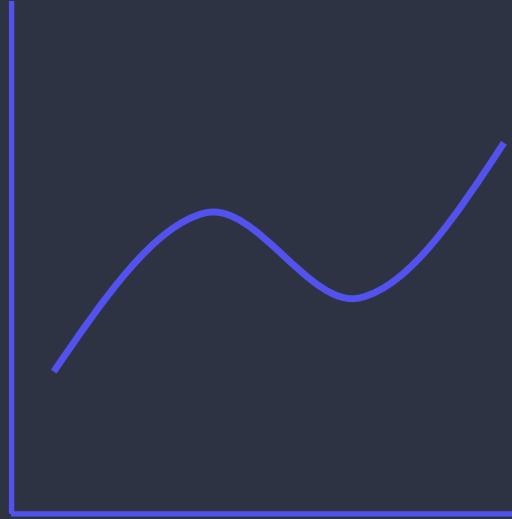
Regression



Classification

# Regression

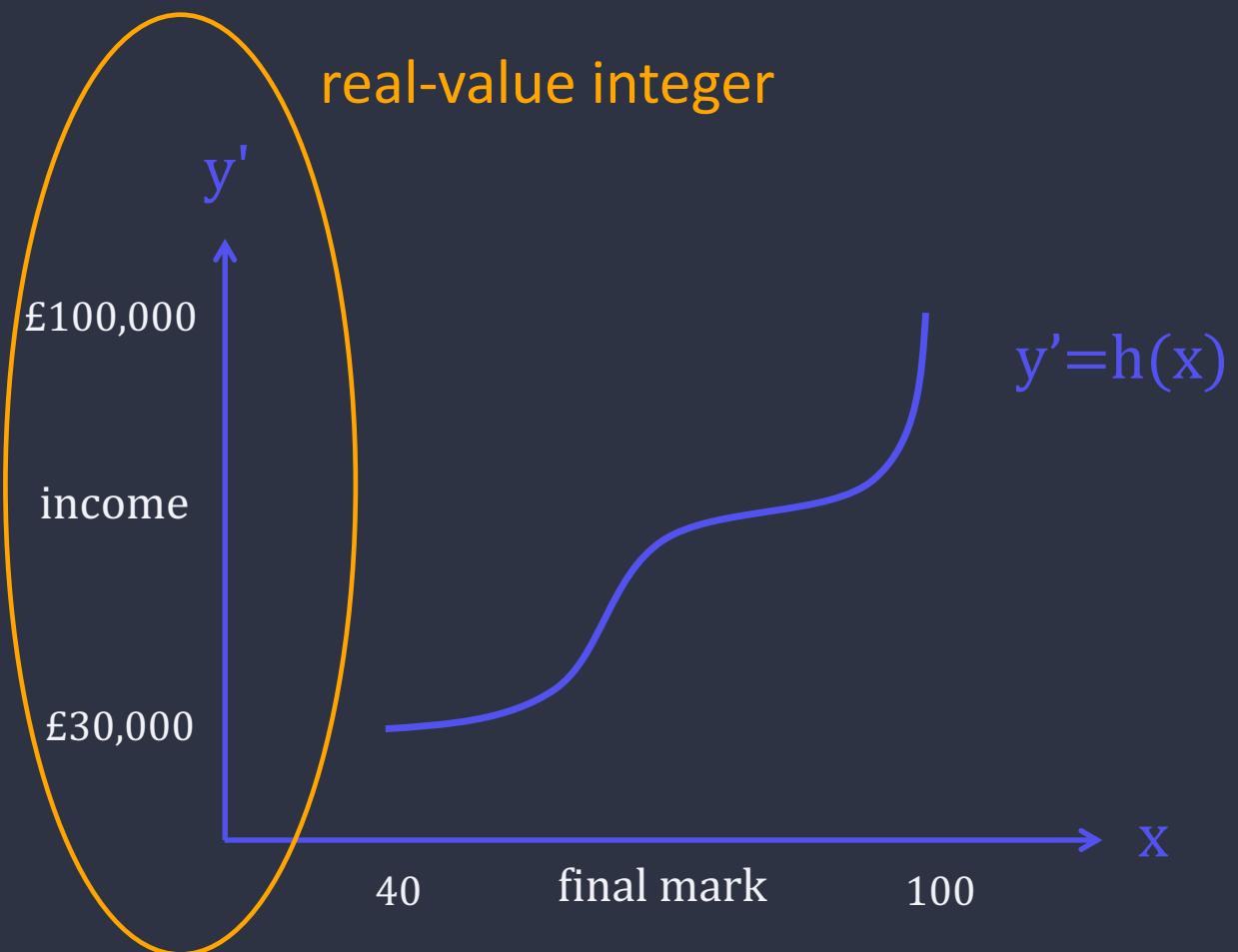
# Regression



- Trying to learn from labelled input-output pairs of instances.
- To find an approximation function that maps from input variables to numerical and continuous output variables.
- The output variables are real-valued data such as integers or floating-point values; often quantities e.g. sizes, weights and amounts.

EXAMPLE.

Final mark of ML module to predict income



# Regression

- Can have real-value or discrete input variables;
- Aims to make prediction of a quantity;
- Is a univariate regression task if the input variable is a single value;
- Is a multivariate regression task if the input variable is a vector containing a group of single values.

# Regression

- To estimate / evaluate how well a regression model performs:

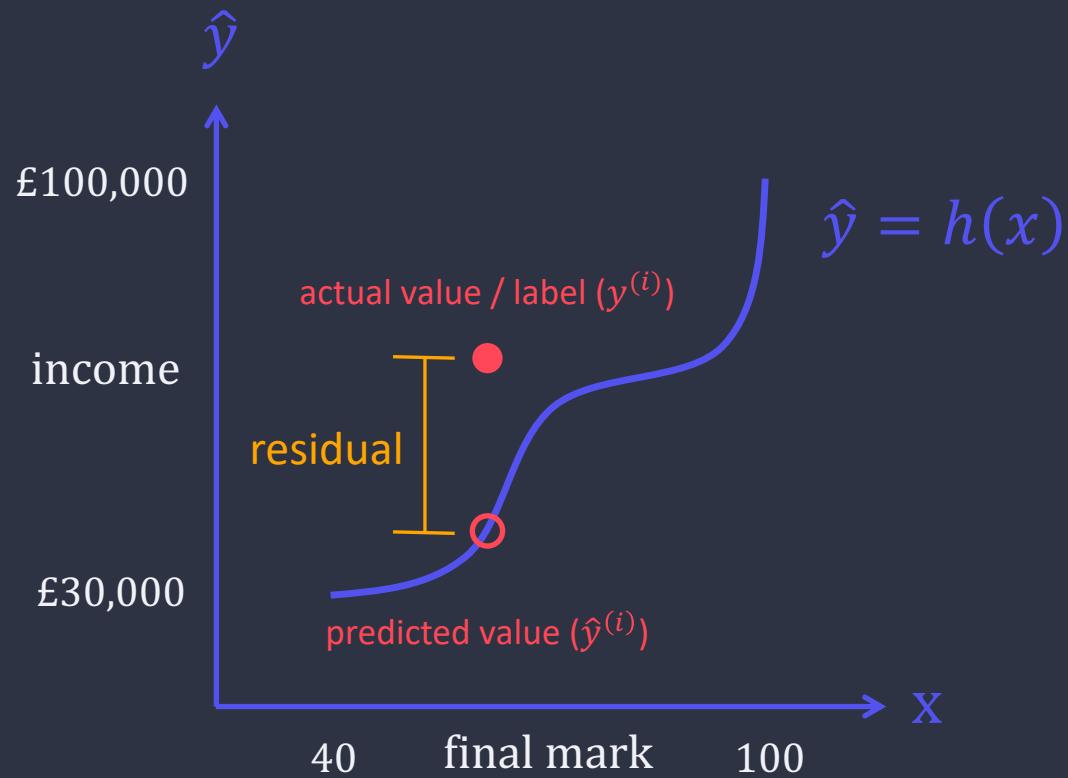
Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2}$$

(standard deviation of the Residuals, i.e. prediction errors)

# Regression

- Residuals : a measure of how far the regression line is from the data points.
- RMSE : a measure of how spread out these residuals are, i.e. how concentrated the data point is around the line of best fit.



EXAMPLE.

	predicted	actual
Student 1	£40k	£45k
Student 2	£80k	£70k

$$\begin{aligned} \text{RMSE} &= \text{sqrt}(\text{average}(\text{error}^2)) \\ &= \text{sqrt}((40 - 45)^2 + (80 - 70)^2 / 2) \\ &= \text{sqrt}(62.5) = 7.91k \end{aligned}$$

# Regression

Regression Model --

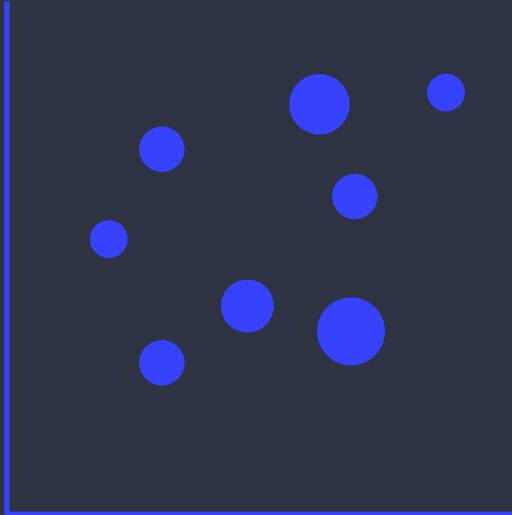
The model that makes such prediction.

Regression Algorithm --

The algorithm that learns from historical data to create a regression model.

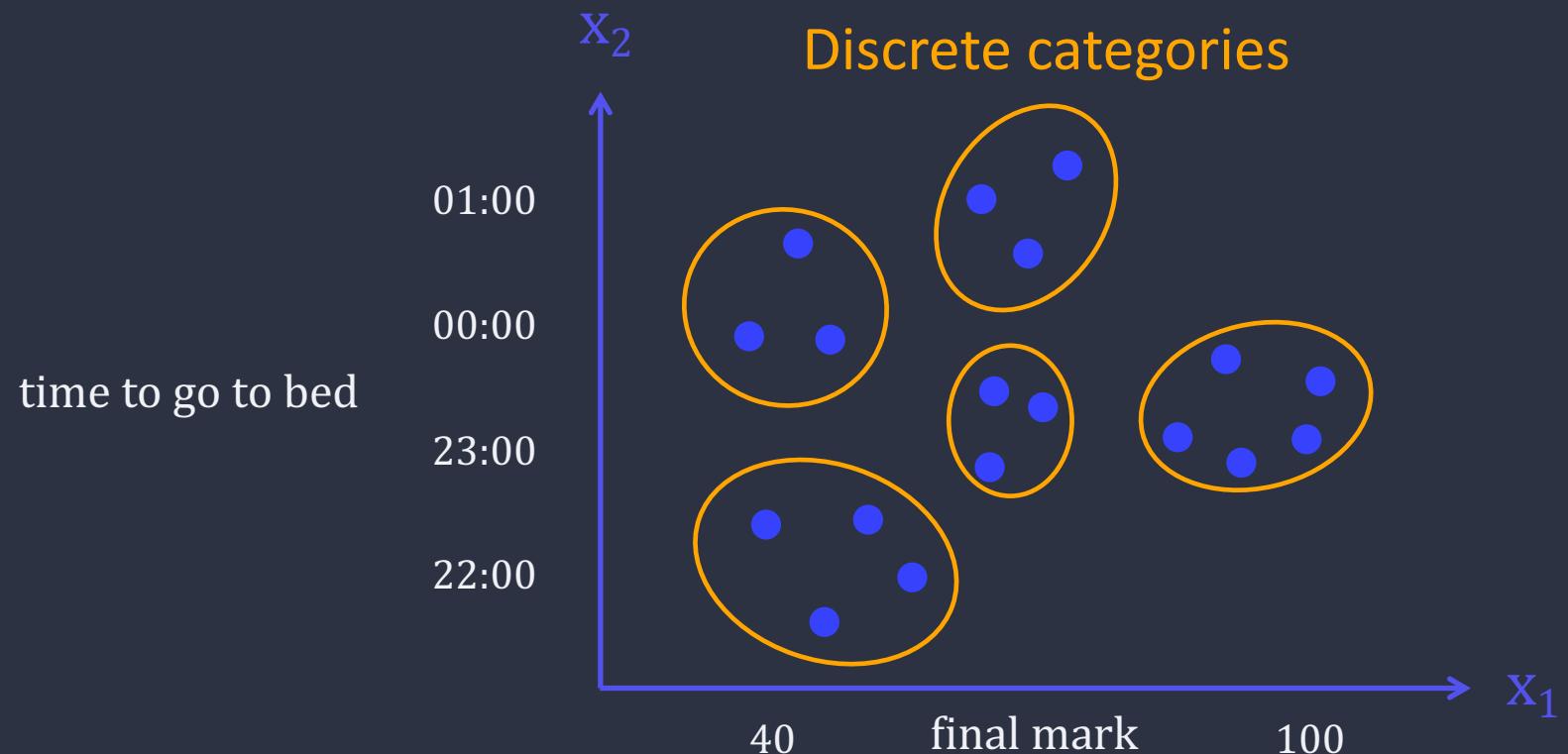
# Classification

# Classification



- Trying to learn from labelled input-output pairs of examples.
- To find an approximation function that maps from input variables to categorical and discrete output variables.
- The output variables are called categories / labels.

EXAMPLE. Final mark of ML module and time to go to bed, to predict degree award



# Classification

- Can have real-value or discrete input variables;
- Aims to make prediction of a class or a category;
- Is a binary classification task if the output variable could be 2 classes;
- Is multi-class classification task is the output variable could be  $>2$  classes;

# Classification

- To estimate / evaluate how well a classification model performs:

**accuracy = correct predictions / total predictions**

## EXAMPLE.

If our model made predictions on 50 students' UG degree award, of which 45 were correct, then the classification accuracy would be: **accuracy = 45 / 50 = 90%**

Other measurements: sensitivity, specificity and precision

# Classification

**Classification Model --**

The model that makes such prediction.

**Classification Algorithm --**

The algorithm that learns from historical data to create a classification model.

# Regression versus Classification

# Difference: prediction result / output variable

## Regression

continuous quantity, e.g., 99, 19.85

## Classification

discrete labels, e.g., {malignant, benign},  
{orange, clementine, lemon}

# Overlaps

## Regression

may predict discrete values, but as integer quantity, e.g., a model could make predictions in a Likert scale, e.g. -2, -1, 0, 1, 2.

## Classification

may predict continues values, but as probability for a class label, e.g. the output could be 98.6% and it means the model is 98.6% sure the cancer is malignant.

Method to estimate how they perform has no overlap

## Regression

can be RMSE, but classification not

## Classification

can be accuracy, but regression not

# Summary

# Summary

ML Models and Algorithms – Model Selection

12 groups of ML Algorithms

Decomposition of ML Algorithms

Representation, Evaluation, Optimisation

Supervised Learning (formal definition)

Regression vs Classification

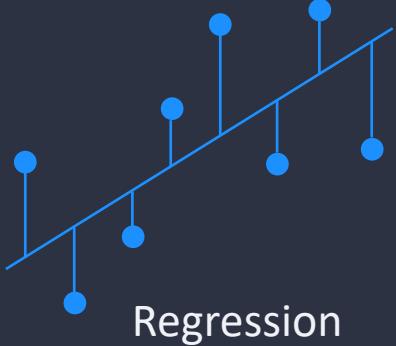
**COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING**

# Linear Regression

Dr Yang Long

# Last Lecture

## Machine Learning Algorithms



Regression



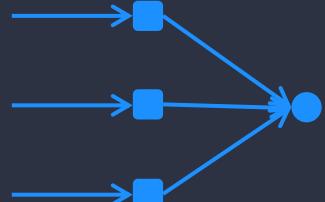
Regularisation



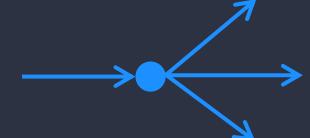
Clustering



Bayesian



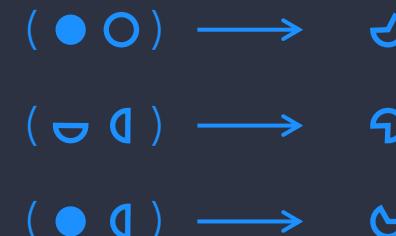
Ensemble



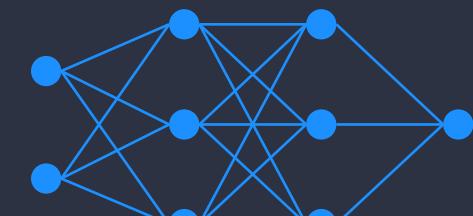
Neural Network



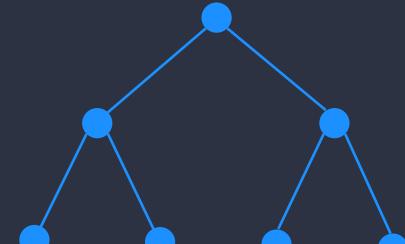
Instance-based



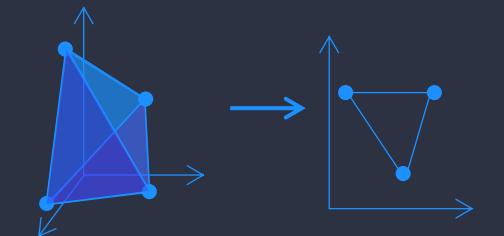
Associated Rule Learning



Deep Learning



Tree-based



Dimensionality Reduction



Reinforcement Learning

# Lecture Overview



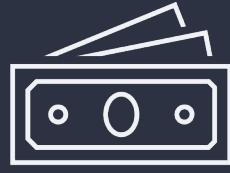
## Linear Regression

1. Intuition
2. Cost Function
3. Gradient Descent
4. Code in Practice (`scikit-learn`)

# 1. Intuition

EXAMPLE. Annual income to predict happiness

given



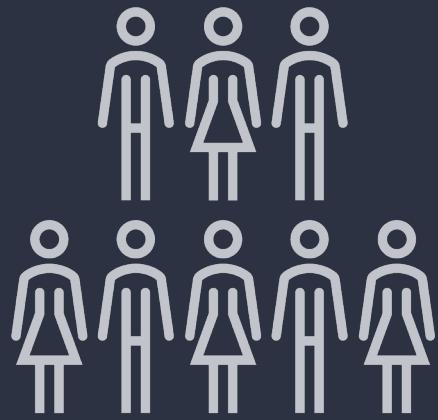
annual income

to predict



happiness

## EXAMPLE. Annual income to predict happiness



500 people in Durham

survey

annual income

£15k -- £75k

independent variable

happiness

0 -- 10

dependent variable

A supervised learning task

-- we know “right answer”, correct label i.e. given annual income, we know exact happiness.

A regression task

-- we want to predict a real-valued output i.e. happiness in a continuous scale from 0 to 10.

## EXAMPLE. Annual income to predict happiness



annual income

£15k -- £75k

independent variable

happiness

0 -- 10

dependent variable

500 people in Durham

survey

350 (training set)

150 (test set)

The task: to **train** a linear regression (supervised learning) model on the training set and **evaluate** the trained model on the test set, so that the model can **predict** happiness of someone in Durham with certain amount of annual income.

# Some notions...

$m$  the number of training examples/instances

$x$  input/independent variable/feature

$y$  output/dependent variable/target, prediction

$(x, y)$  a single training example/instance

$(x^{(i)}, y^{(i)})$  the  $i^{th}$  training example/instance ( $i$  is the index)

## EXAMPLE. Annual income to predict happiness

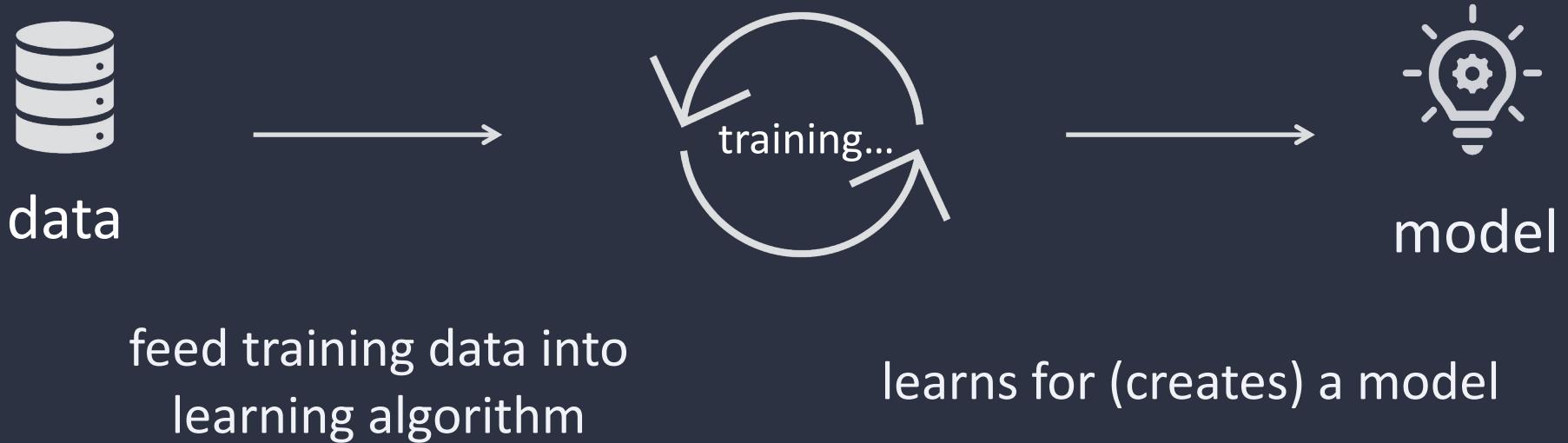
$$m = 350$$

$$(x^{(2)}, y^{(2)}) = (49.79, 3.433)$$

$$(x^{(i)}, y^{(i)}) = (21.18, 1.45)$$

	$x$	$y$
1	38.63	2.314
2	49.79	3.433
...	...	...
i	21.18	1.45
...	...	...
350	71.19	5.95

## EXAMPLE. Annual income to predict happiness



EXAMPLE. Annual income to predict happiness

income (input  $x$  / independent variable)



model (function)

Hypothesis function  $h$ , mapping from  $x$  to  $y$



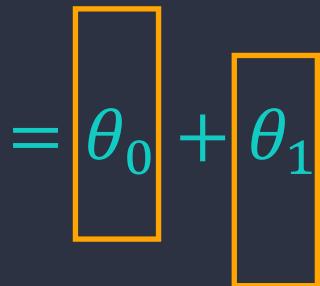
happiness (output  $y$  / dependent variable)

EXAMPLE. Annual income to predict happiness

Hypothesis function  $h$ , mapping from  $x$  to  $y$

$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad \text{or} \quad h(x) = \boxed{\theta_0} + \boxed{\theta_1} x$$

y-intercept, bias



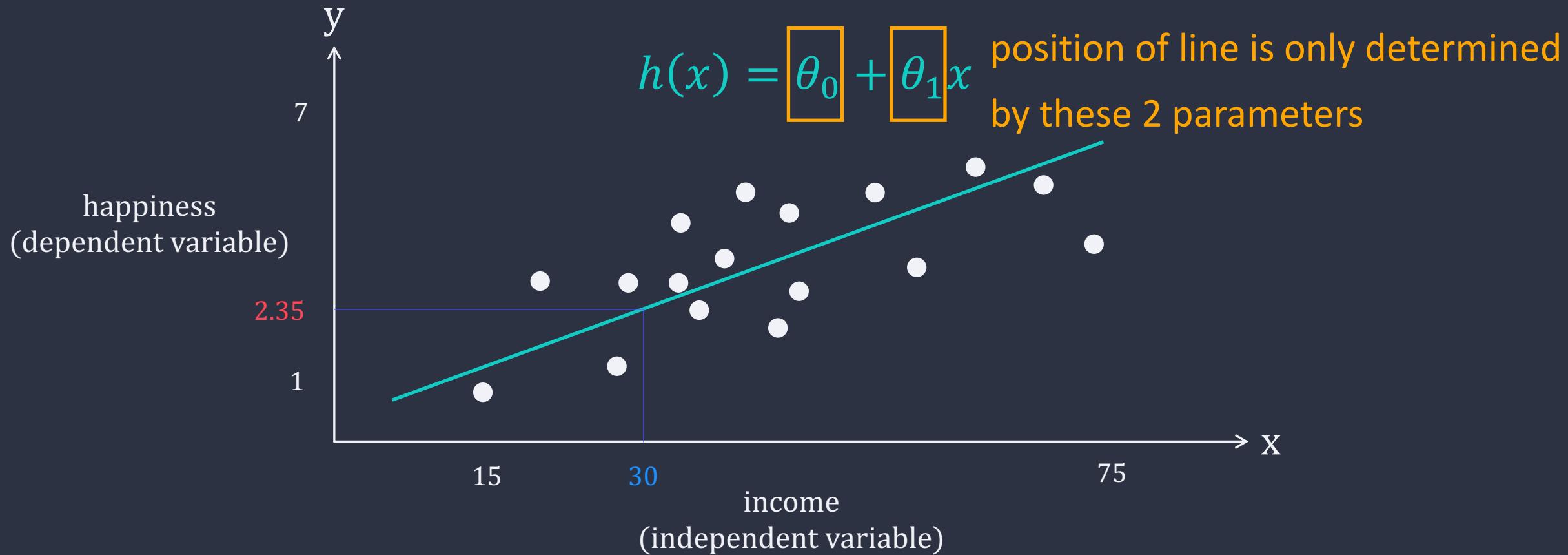
slope, weight (regression coefficient)

$\hat{y} = h(x)$  : predicted value (estimation), given a specific  $x$

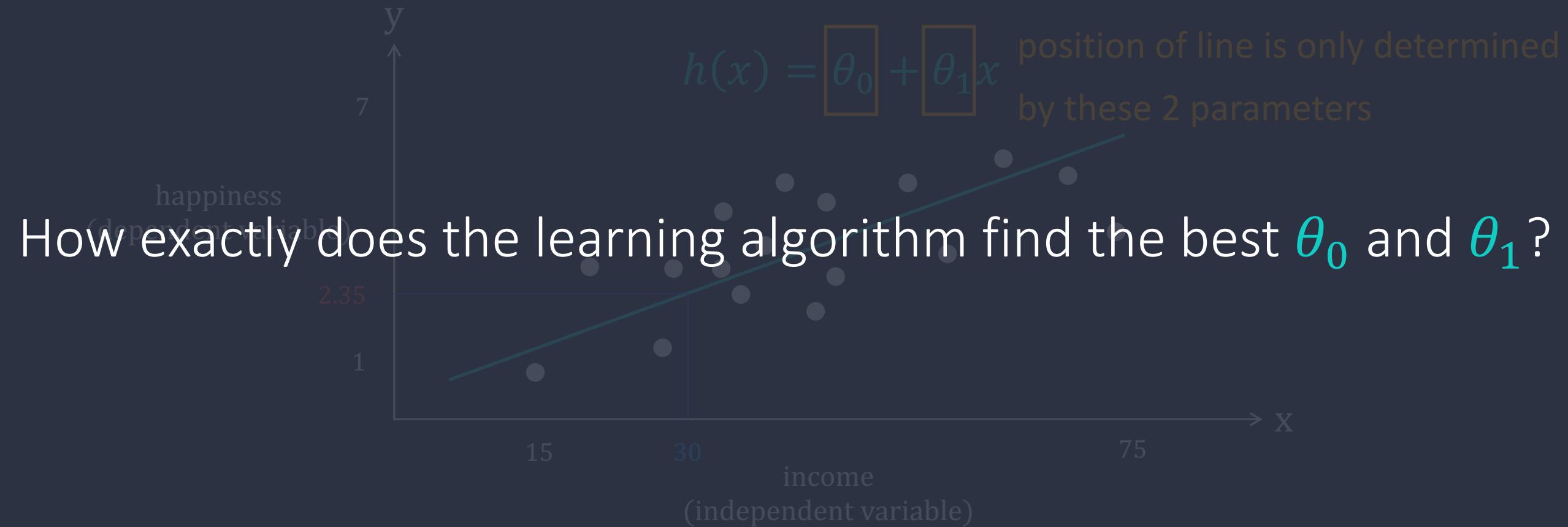
## EXAMPLE. Annual income to predict happiness



## EXAMPLE. Annual income to predict happiness



## EXAMPLE. Annual income to predict happiness

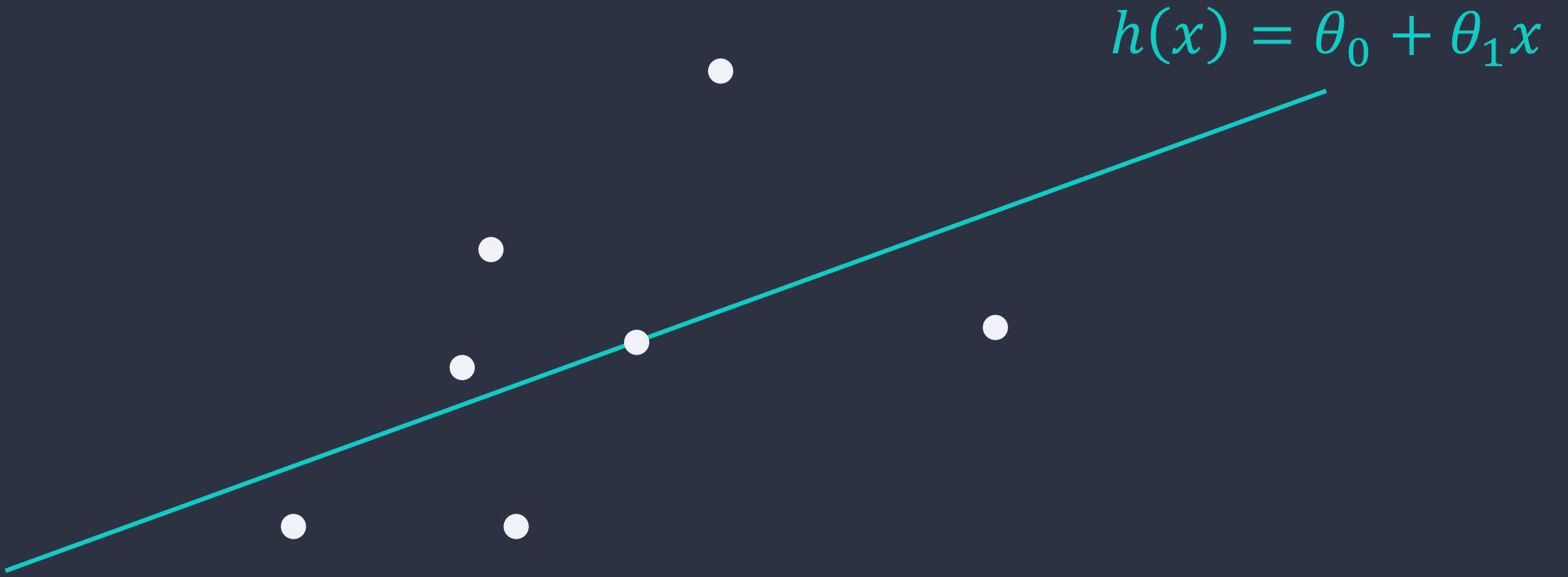


## 2. Cost Function

# Cost

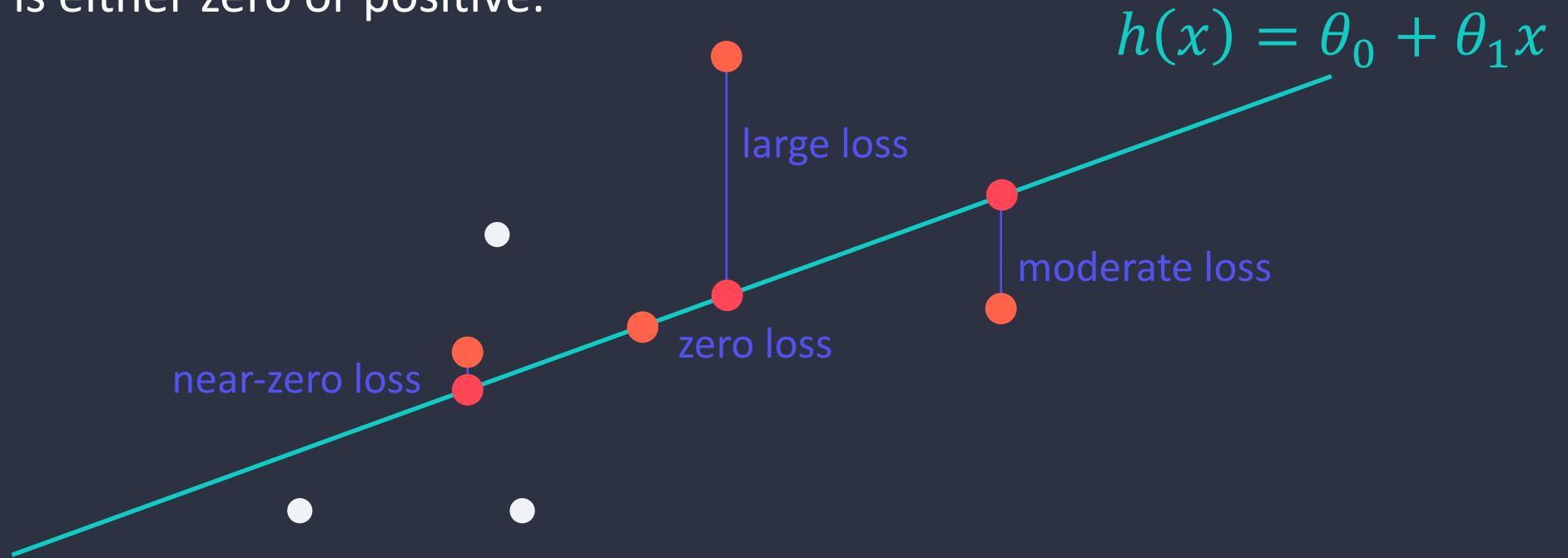
# Cost

- Cost measures how **bad** a **model** performs on the whole training set.



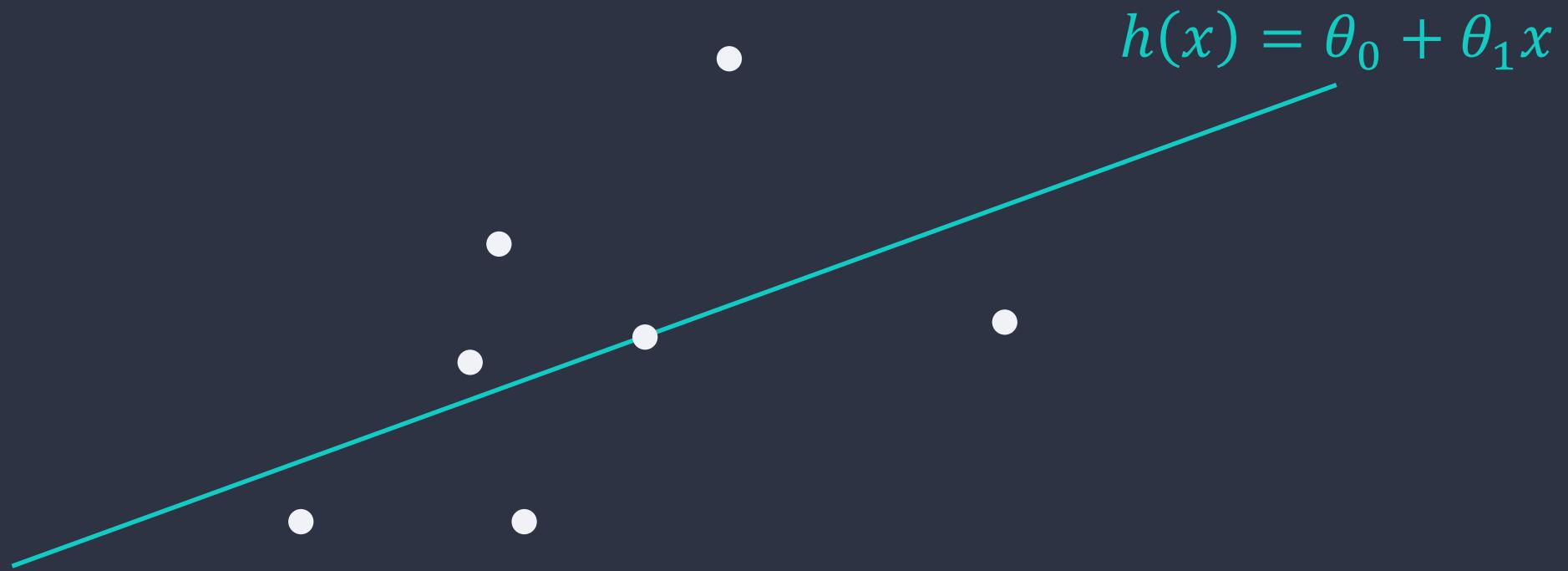
# Cost

- Cost measures how bad a **model** performs on the **whole** training set.
- Loss (error) measures how bad a **model** performs on one **single** training example.
  - The difference between the **prediction** and **actual value** for **a given x**.
  - Value is either zero or positive.



# Cost

- Cost measures how bad a **model** performs on the **whole** training set.
- Loss (error) measures how bad a **model** performs on one **single** training example.
  - The difference between the **prediction** and **actual value** for **a given x**.
  - Value is either zero or positive.
- We want **straight line (model)** so that not only one or a few but all having small/zero **loss**.

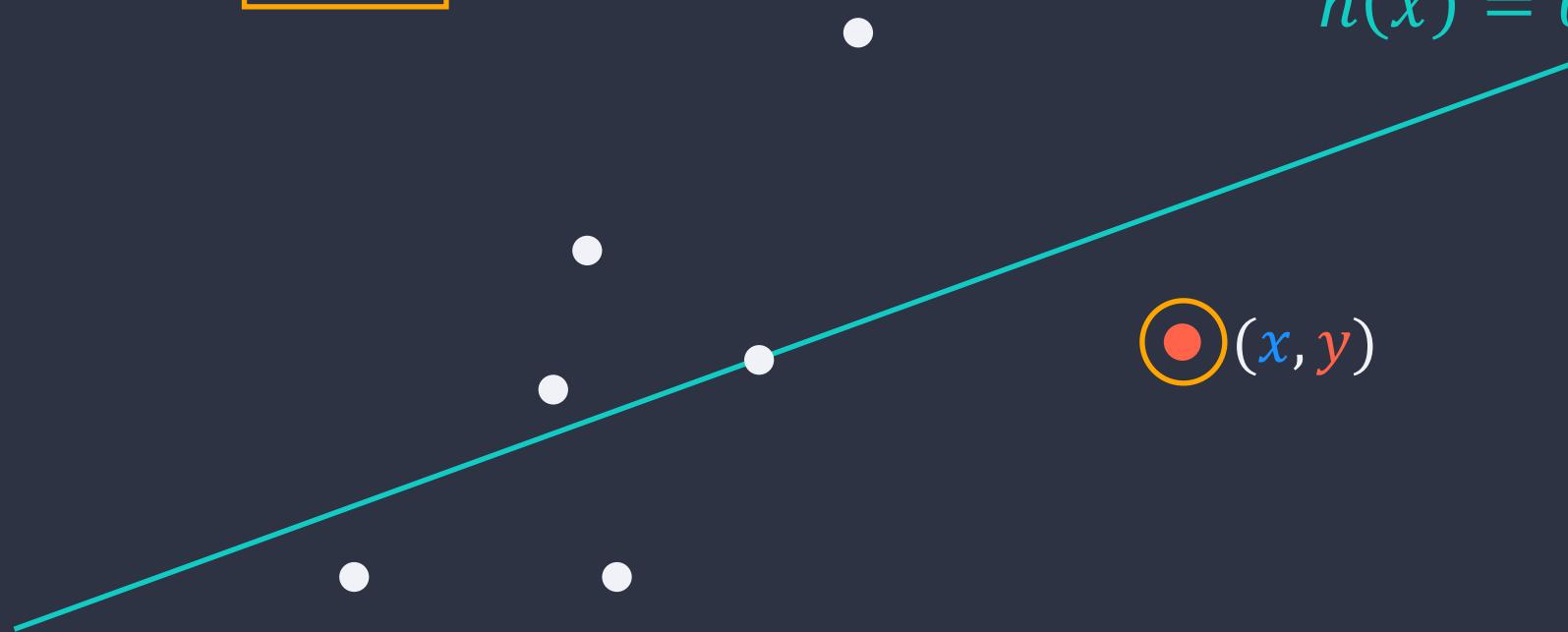


# Cost Function

- We want **straight line** (model) so that not only one or a few but all having small/zero **loss**.
- Squared Error Cost

$$\sum_{i=1}^m (predicted^{(i)} - \boxed{actual^{(i)}})^2$$

$$h(x) = \theta_0 + \theta_1 x$$

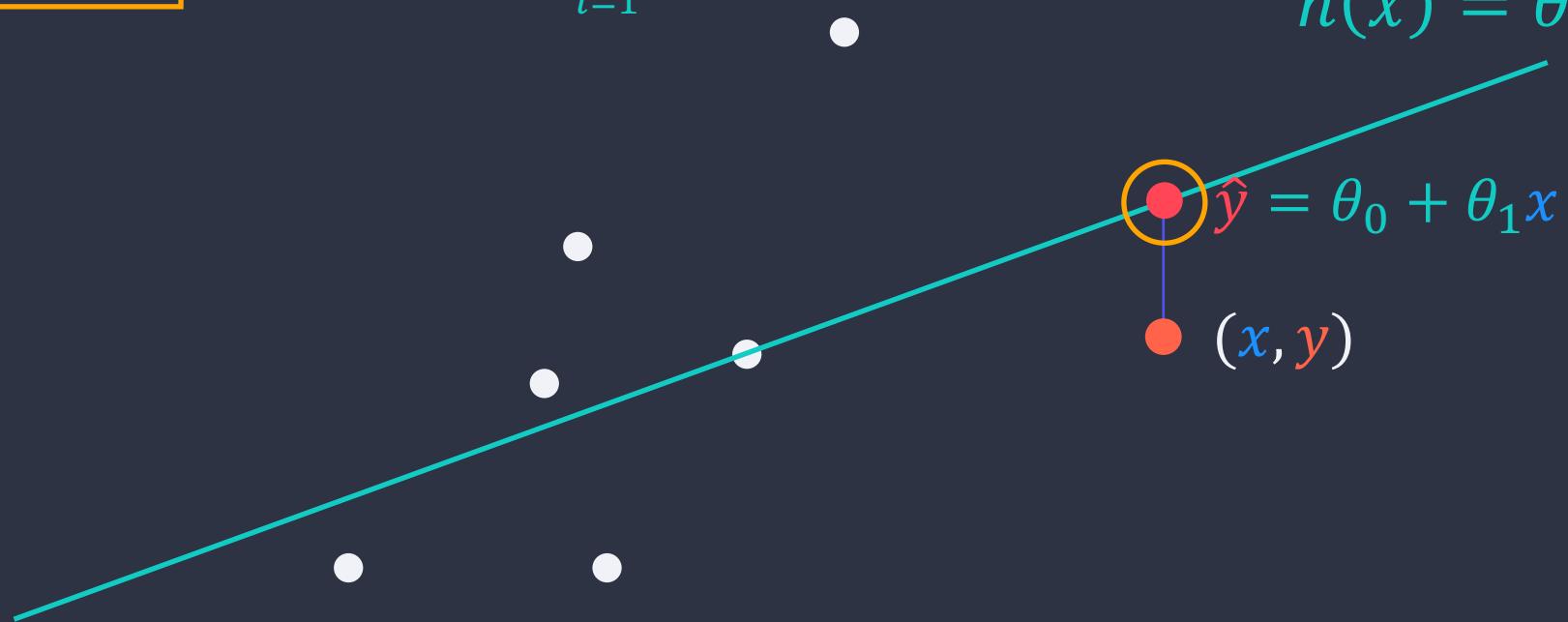


# Cost Function

- We want **straight line** (model) so that not only one or a few but all having small/zero **loss**.
- Squared Error Cost

$$\sum_{i=1}^m (\text{predicted}^{(i)} - \text{actual}^{(i)})^2 = \sum_{i=1}^m (h(x)^{(i)} - y^{(i)})^2$$

$$h(x) = \theta_0 + \theta_1 x$$



# Cost Function

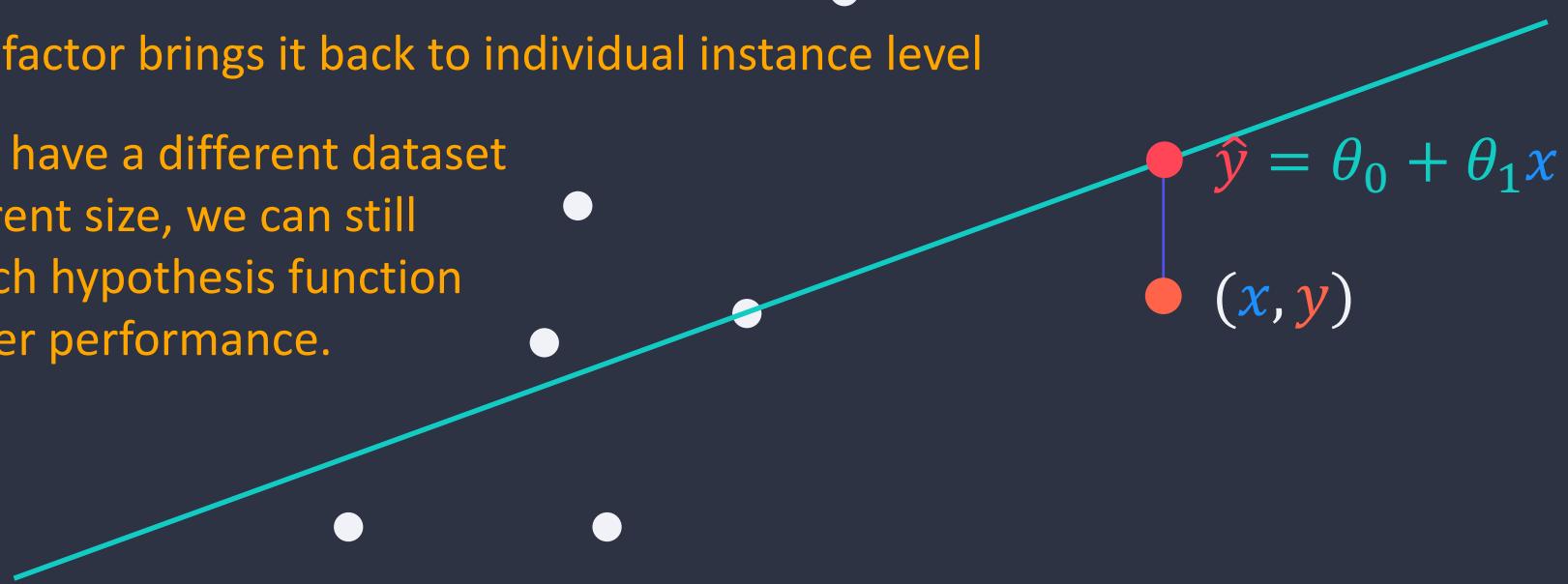
- We want straight line (model) so that not only one or a few but all having small/zero loss.
- Squared Error Cost

$$\frac{1}{m} \sum_{i=1}^m (\text{predicted}^{(i)} - \text{actual}^{(i)})^2 = \frac{1}{m} \sum_{i=1}^m (h(x)^{(i)} - y^{(i)})^2$$

$$h(x) = \theta_0 + \theta_1 x$$

Normalising factor brings it back to individual instance level

Even if we have a different dataset with different size, we can still judge which hypothesis function has a better performance.

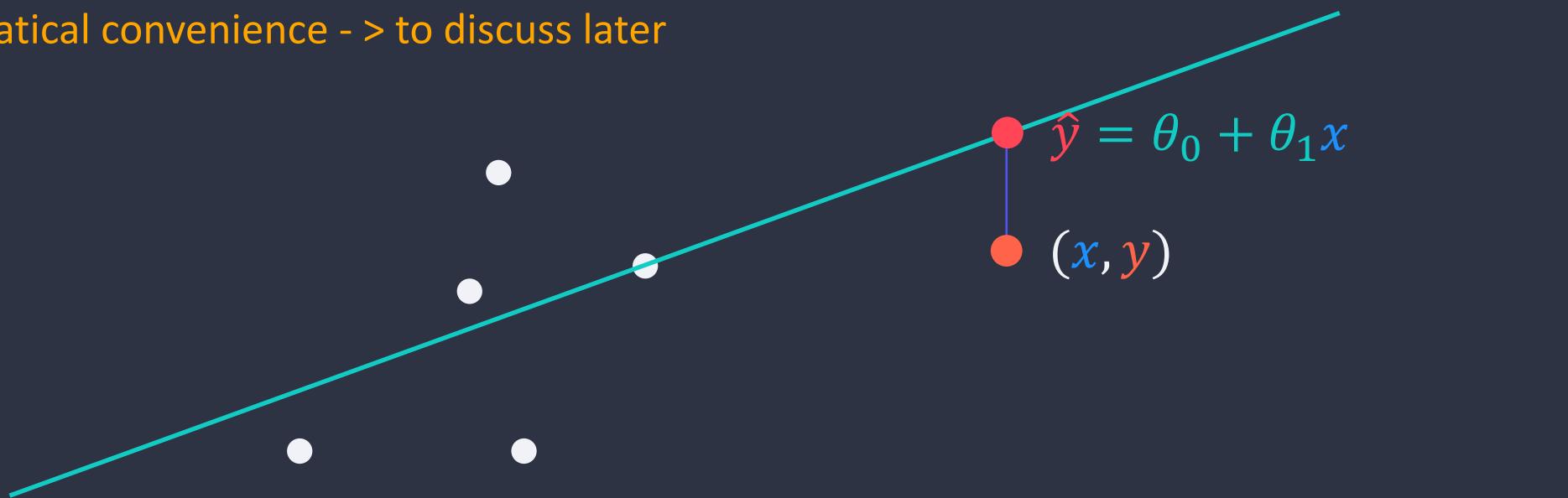


# Cost Function

- We want **straight line** (model) so that not only one or a few but all having small/zero **loss**.
- Squared Error Cost

$$\frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (\text{predicted}^{(i)} - \text{actual}^{(i)})^2 = \frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (h(x)^{(i)} - y^{(i)})^2$$
$$h(x) = \theta_0 + \theta_1 x$$

For mathematical convenience - > to discuss later

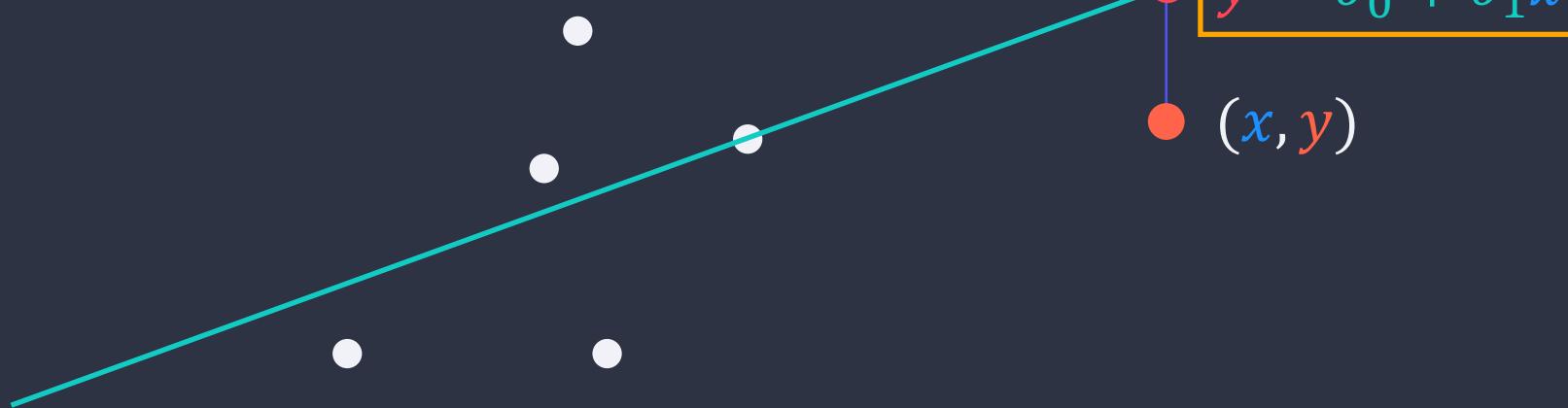


# Cost Function

- We want **straight line** (model) so that not only one or a few but all having small/zero **loss**.
- Squared Error Cost

$$\begin{aligned} \frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (\text{predicted}^{(i)} - \text{actual}^{(i)})^2 &= \frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (\boxed{h(x)^{(i)}} - y^{(i)})^2 \\ &= \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2 \end{aligned}$$

$h(x) = \theta_0 + \theta_1 x$



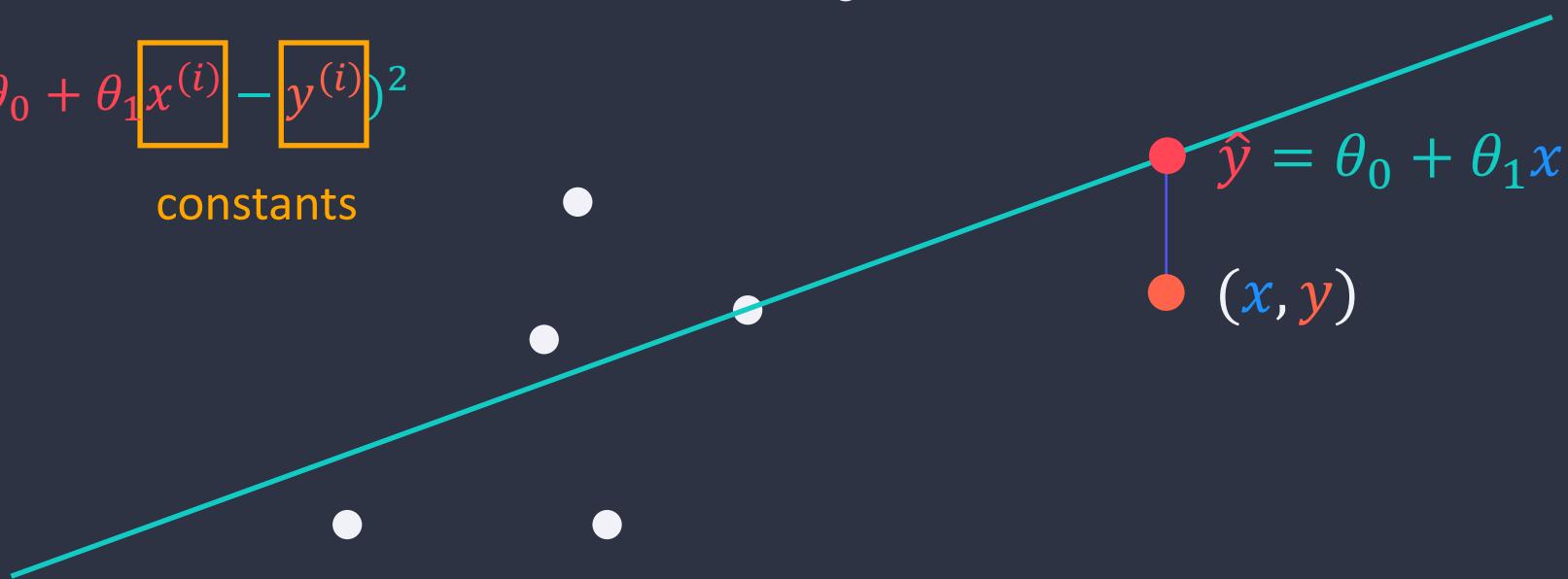
# Cost Function

- We want **straight line (model)** so that not only one or a few but all having small/zero **loss**.
- Squared Error Cost

$$\frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (\text{predicted}^{(i)} - \text{actual}^{(i)})^2 = \frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^m (h(x)^{(i)} - y^{(i)})^2$$
$$h(x) = \theta_0 + \theta_1 x$$

$$= \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 \boxed{x^{(i)}} - \boxed{y^{(i)}})^2$$

constants

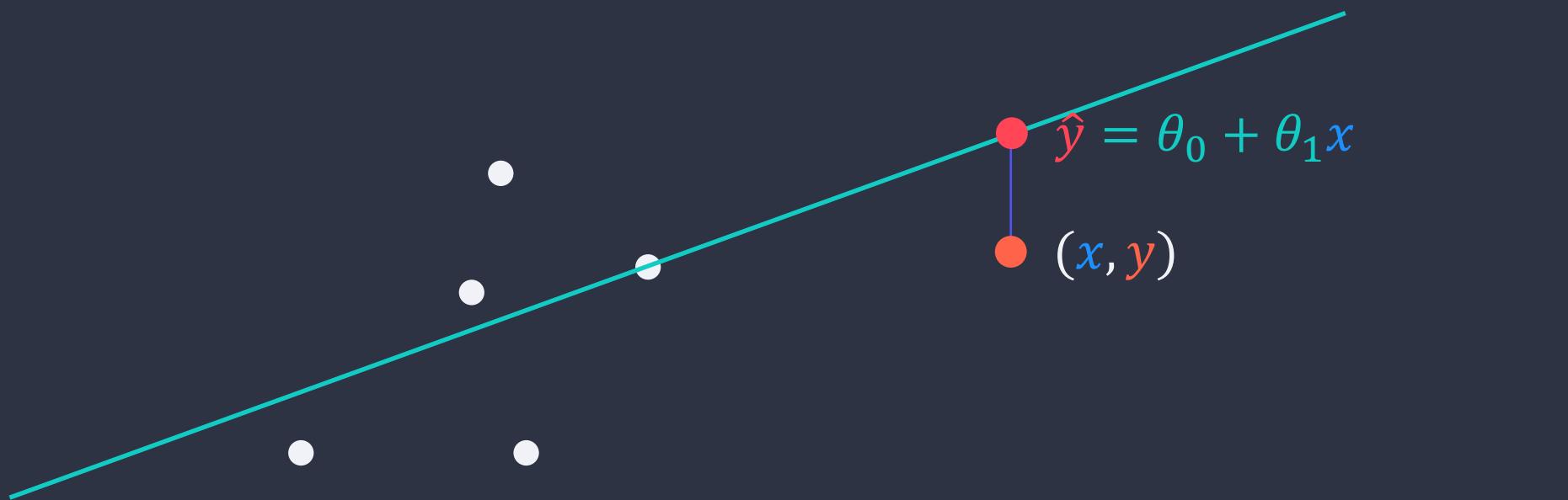


# Cost Function

- We want **straight line** (**model**) so that not only one or a few but all having small/zero **loss**.
- Mean Squared Error Cost Function (a function of  $\theta_0$  and  $\theta_1$ )

$$MSE(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

$$h(x) = \theta_0 + \theta_1 x$$



# Cost Function

- We want straight line (model) so that not only one or a few but all having small/zero loss.
- Mean Squared Error Cost Function (a function of  $\theta_0$  and  $\theta_1$ )
- Goal: to solve a minimisation problem

$$\underset{\theta_0, \theta_1}{argmin} = \frac{1}{2m} \sum_{i=1}^m (\boxed{\theta_0} + \boxed{\theta_1}x^{(i)} - y^{(i)})^2$$

$\theta_0$  and  $\theta_1$  to be used in the hypothesis function  $h(x)$

$$h(x) = \boxed{\theta_0} + \boxed{\theta_1}x$$

# Cost Function

- We want straight line (model) so that not only one or a few but all having small/zero loss.
- Mean Squared Error Cost Function (a function of  $\theta_0$  and  $\theta_1$ )
- Goal: to solve a minimisation problem

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

To find  $\theta_0$  and  $\theta_1$ , so that  $J(\theta_0, \theta_1)$  is minimised (close to zero).

## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

- Represents the model
- Values of  $\theta_0$  and  $\theta_1$  are fixed
- $x$  is the independent variable
- A function of independent variable  $x$

## Cost Function

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

- Represents the cost
- Values of  $x^{(i)}$  and  $y^{(i)}$  are fixed
- $\theta_0$  and  $\theta_1$  are the independent variables
- A function of independent variables  $\theta_1, \theta_2$

## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

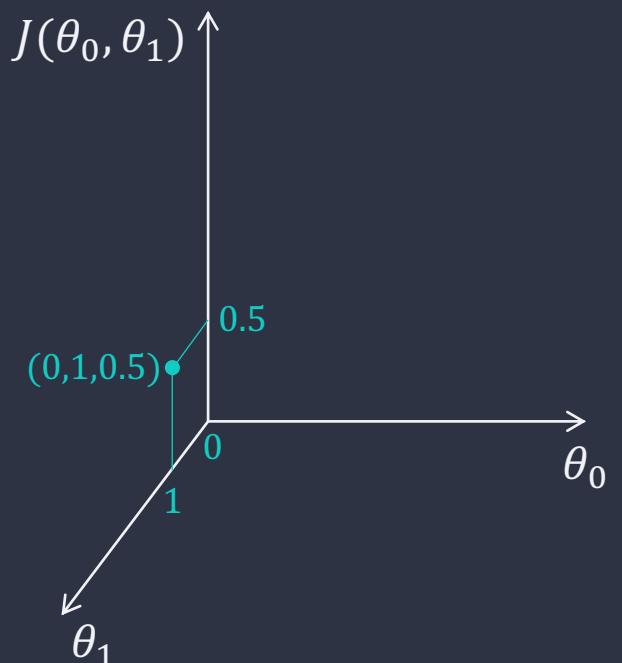


if  $(\theta_0 = 0, \theta_1 = 1)$

$$J(0,1) = \frac{1}{2 \times 3} (1^2 + 1^2 + 1^2) = 0.5$$

## Cost Function

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



## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

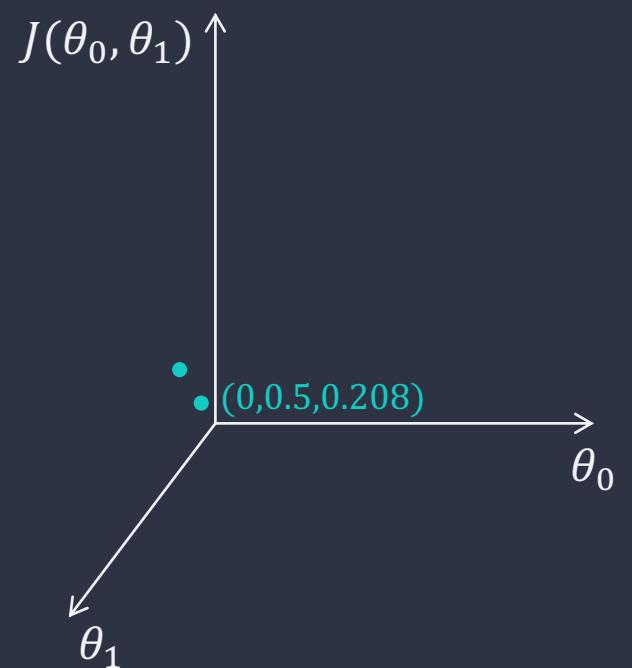


if  $(\theta_0 = 0, \theta_1 = 0.5)$

$$J(0, 0.5) = \frac{1}{2 \times 3} (0^2 + 0.5^2 + 1^2) = 0.208$$

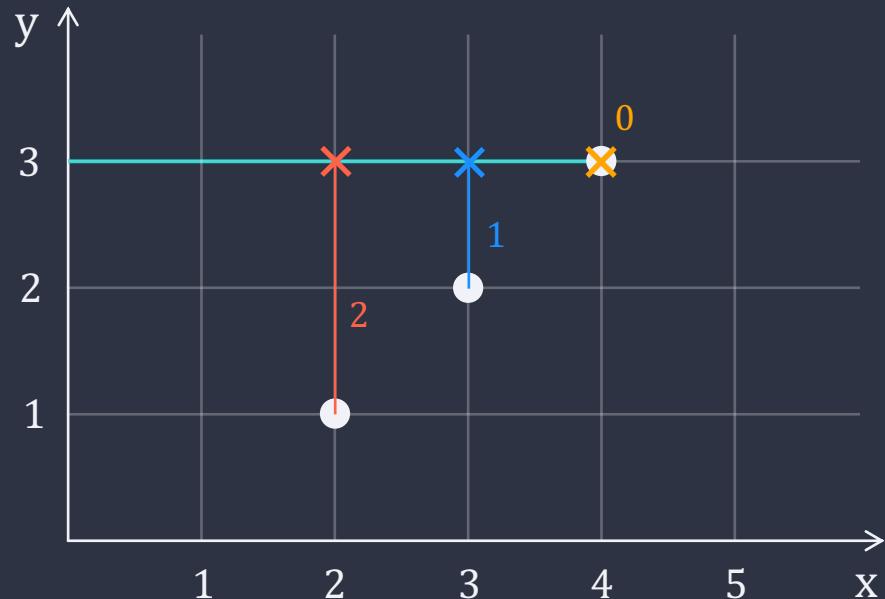
## Cost Function

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



## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

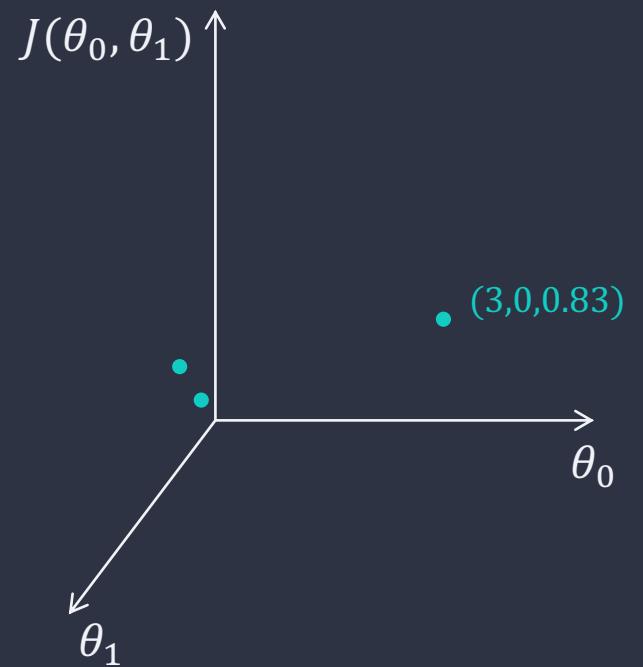


if  $(\theta_0 = 3, \theta_1 = 0)$

$$J(3,0) = \frac{1}{2 \times 3} (2^2 + 1^2 + 0^2) = 0.83$$

## Cost Function

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



## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

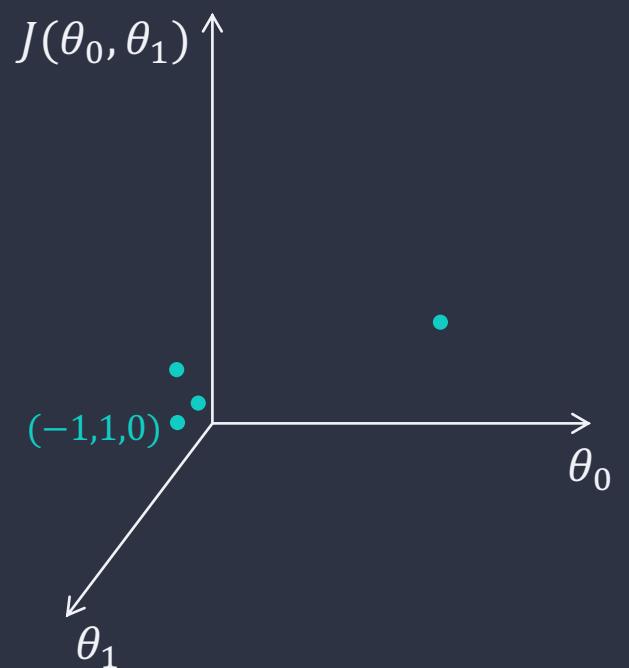


if  $(\theta_0 = -1, \theta_1 = 1)$

$$J(-1, 1) = \frac{1}{2 \times 3} (0^2 + 0^2 + 0^2) = 0$$

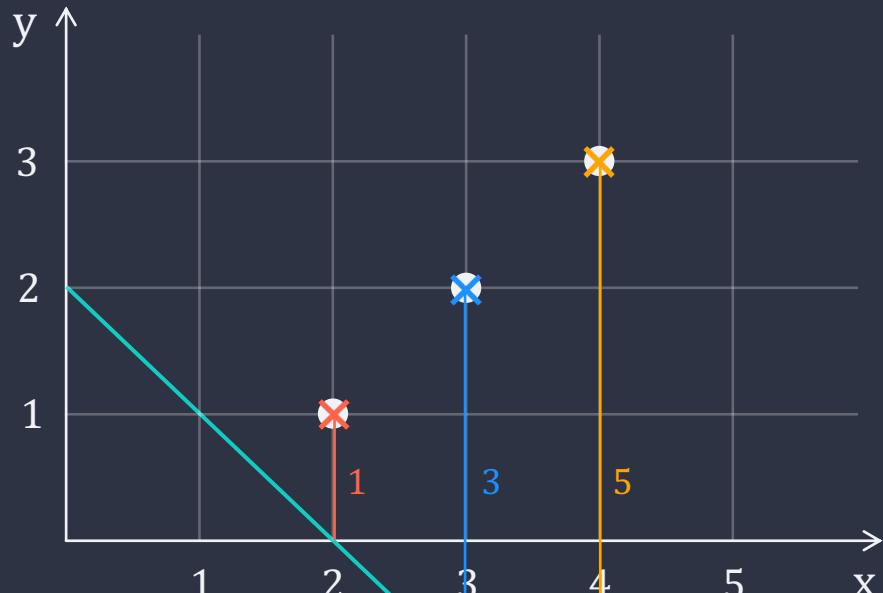
## Cost Function

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



## Hypothesis Function

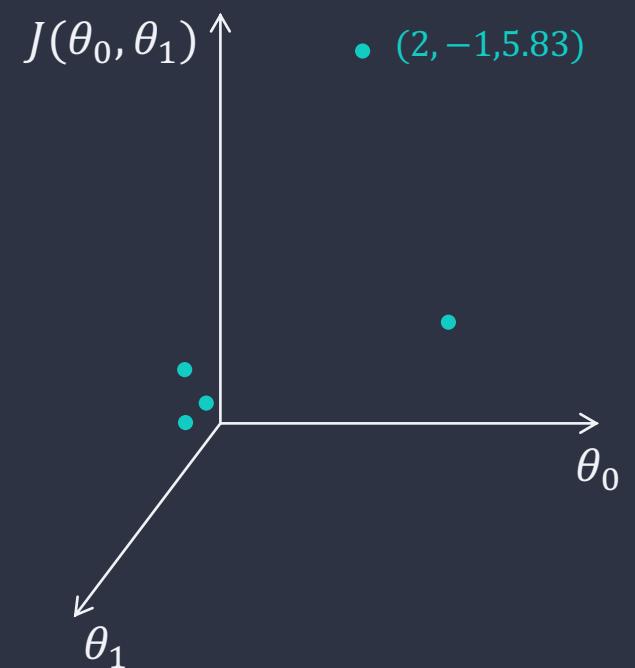
$$h(x) = \theta_0 + \theta_1 x$$



$$J(2, -1) = \frac{1}{2 \times 3} (1^2 + 3^2 + 5^2) = 5.83$$

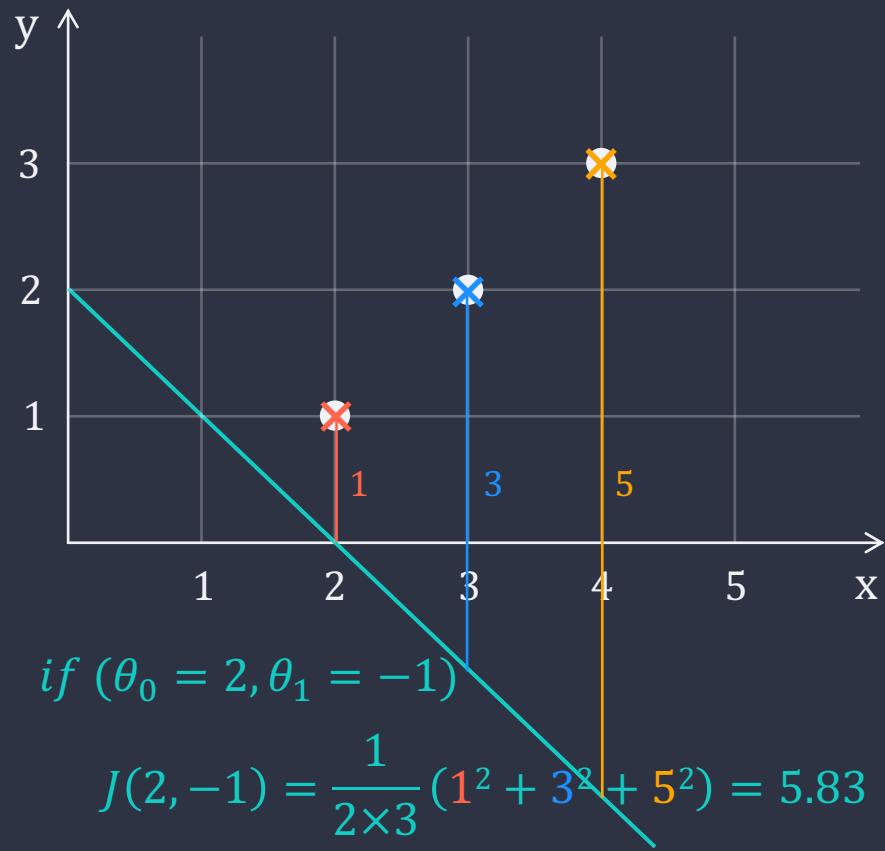
## Cost Function

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



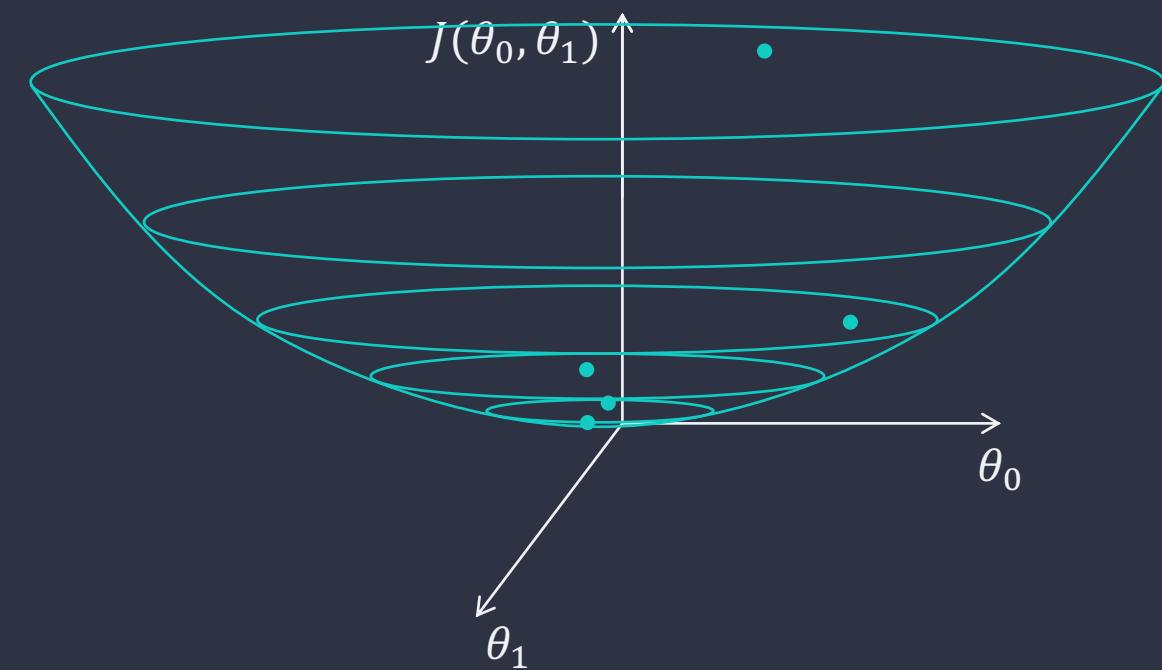
## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$



## Cost Function

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



## Hypothesis Function

$$h(x) = \theta_0 + \theta_1 x$$

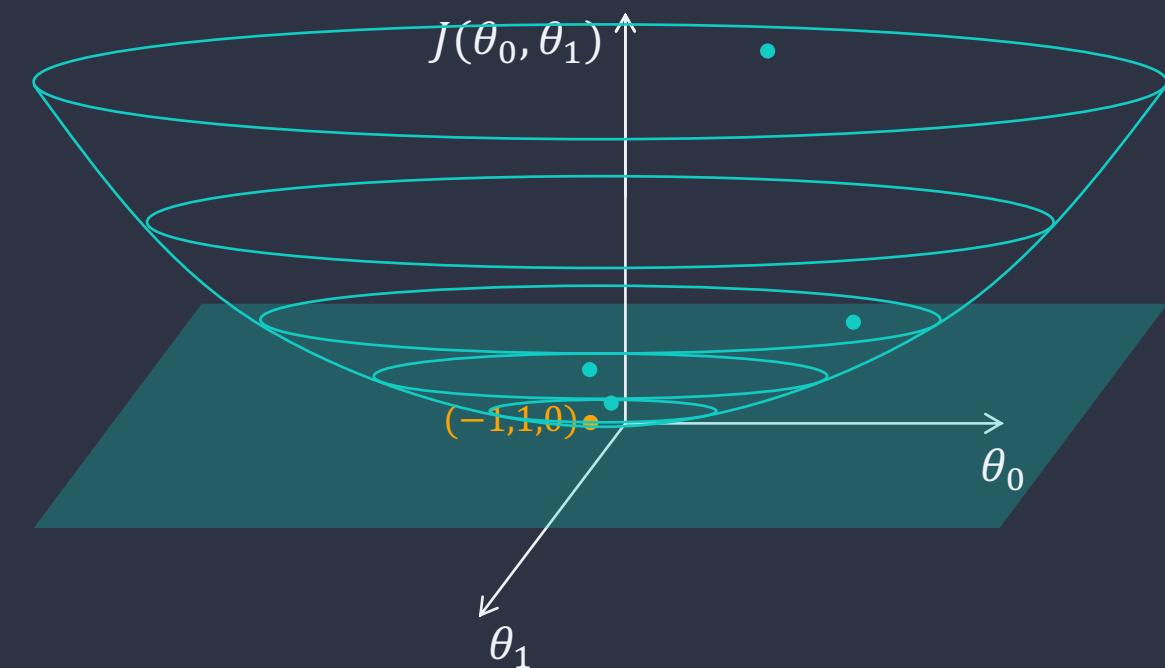


$$\theta_0 = -1, \theta_1 = 1$$

$$J(0,1) = 0$$

## Cost Function

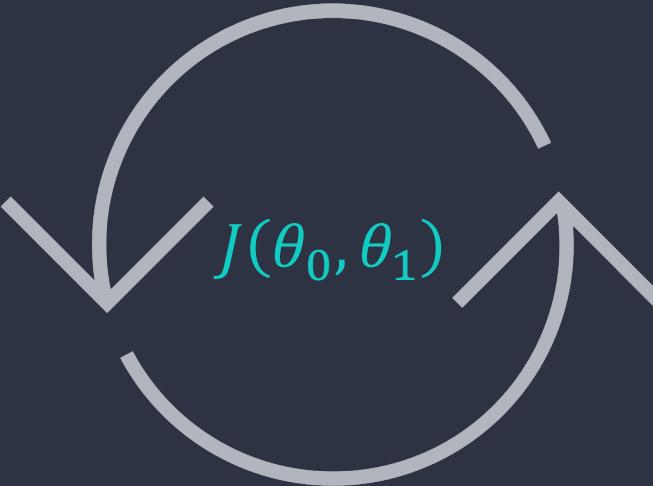
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$



# 3. Gradient Decent

# Gradient Decent - General Idea

Try different pairs of  $\theta_0$  and  $\theta_1$



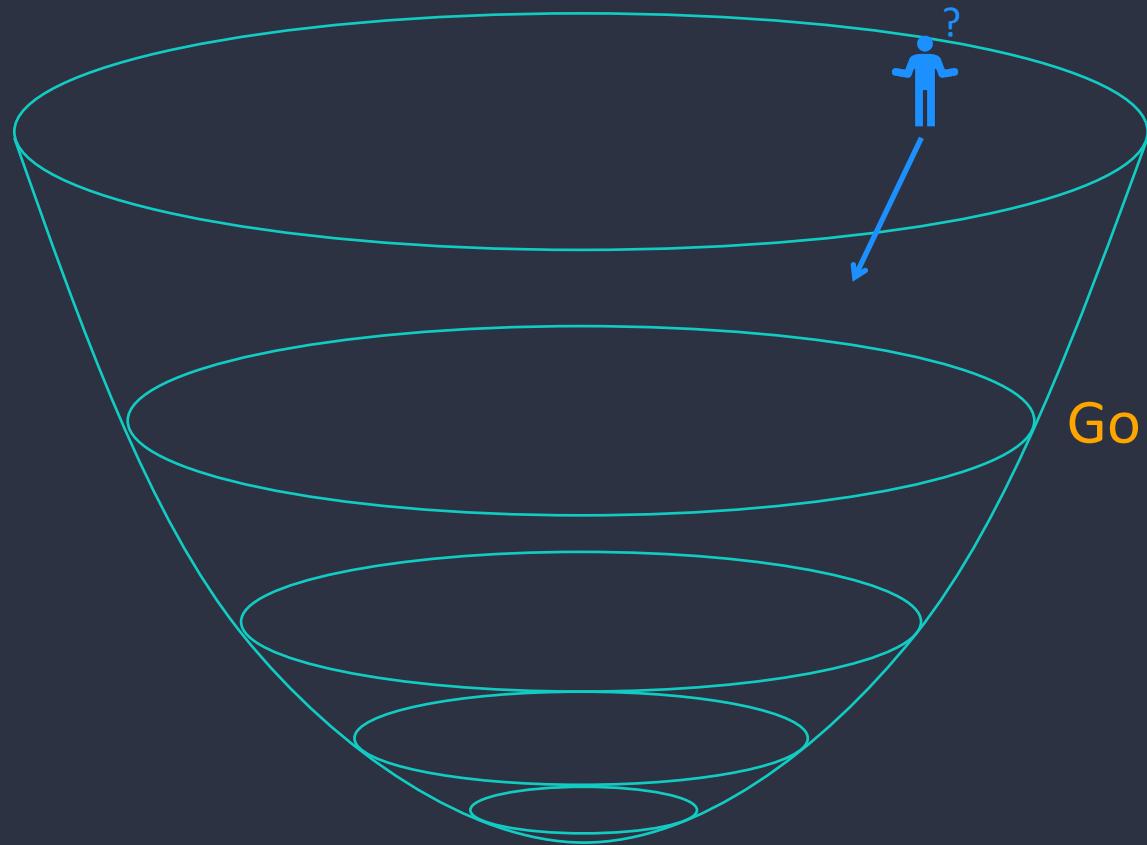
to reduce the cost.

in iterations

- ✓ Gradient Descent helps us on how to change  $\theta_0$  and  $\theta_1$  values and when to stop.

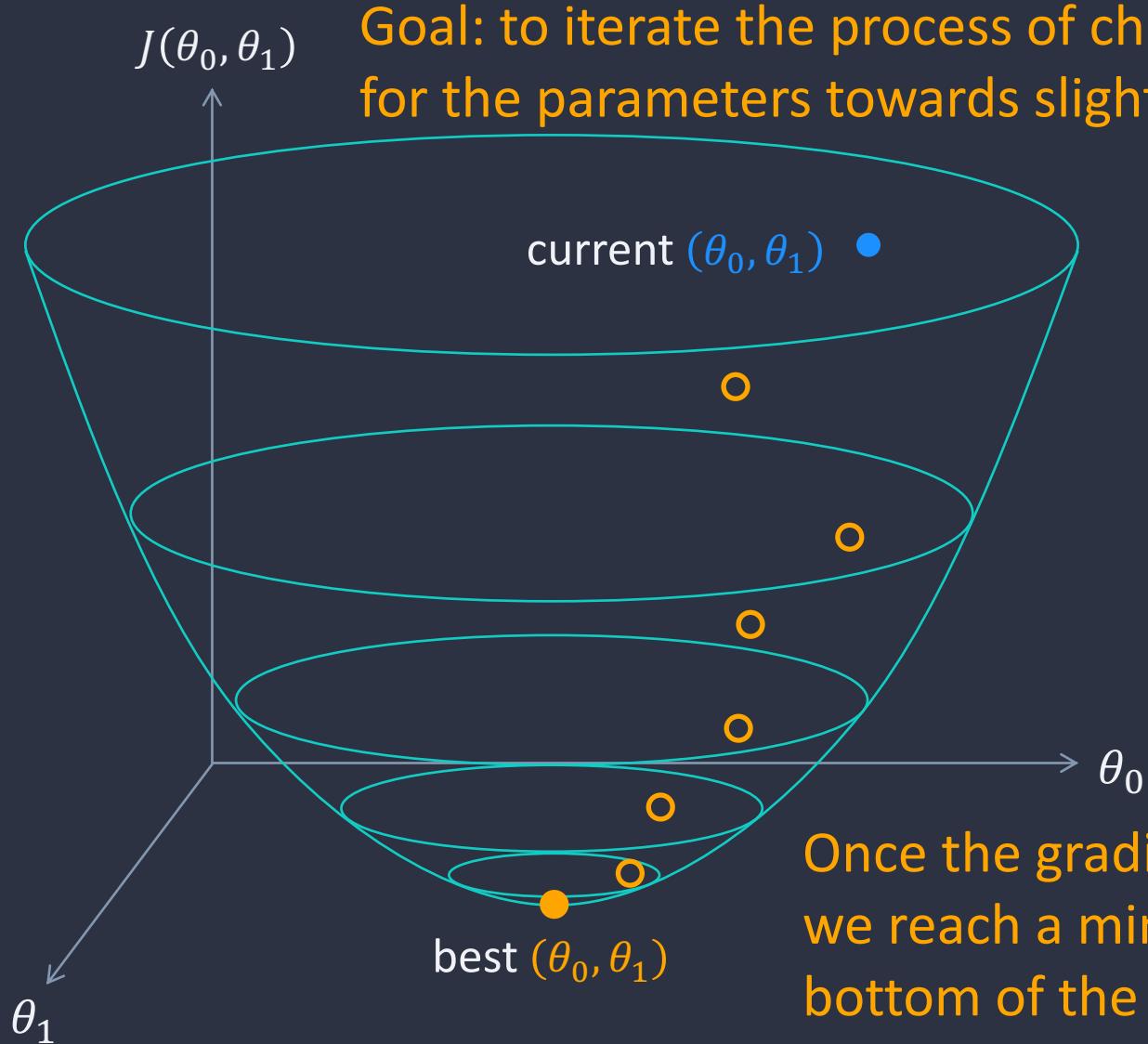
# Gradient Decent - Intuition

How to reach the bottom as quickly as possible?



Go downhill in the direction  
of the steepest slope.

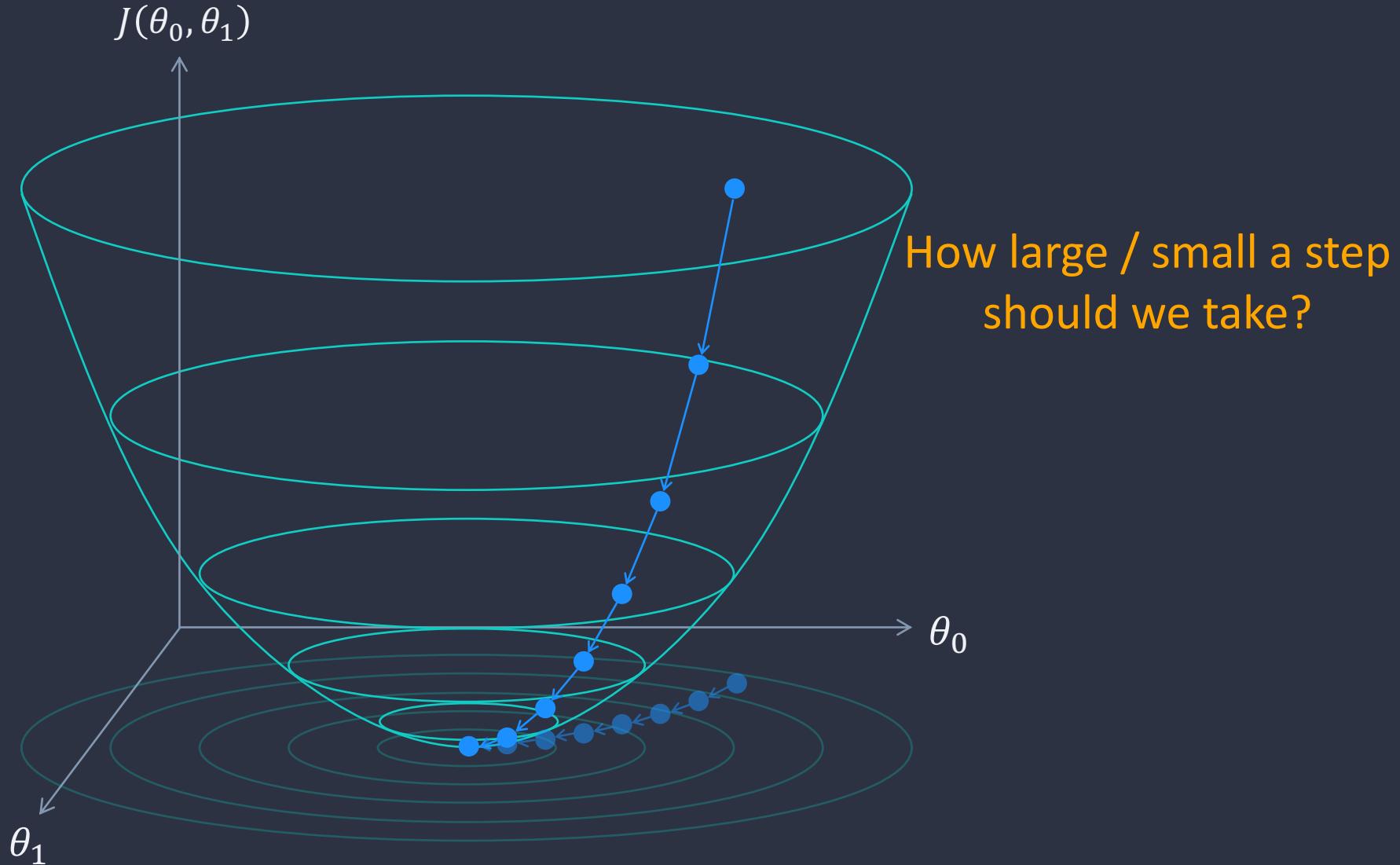
# Gradient Decent - Intuition



Goal: to iterate the process of choosing new values for the parameters towards slightly lower cost.

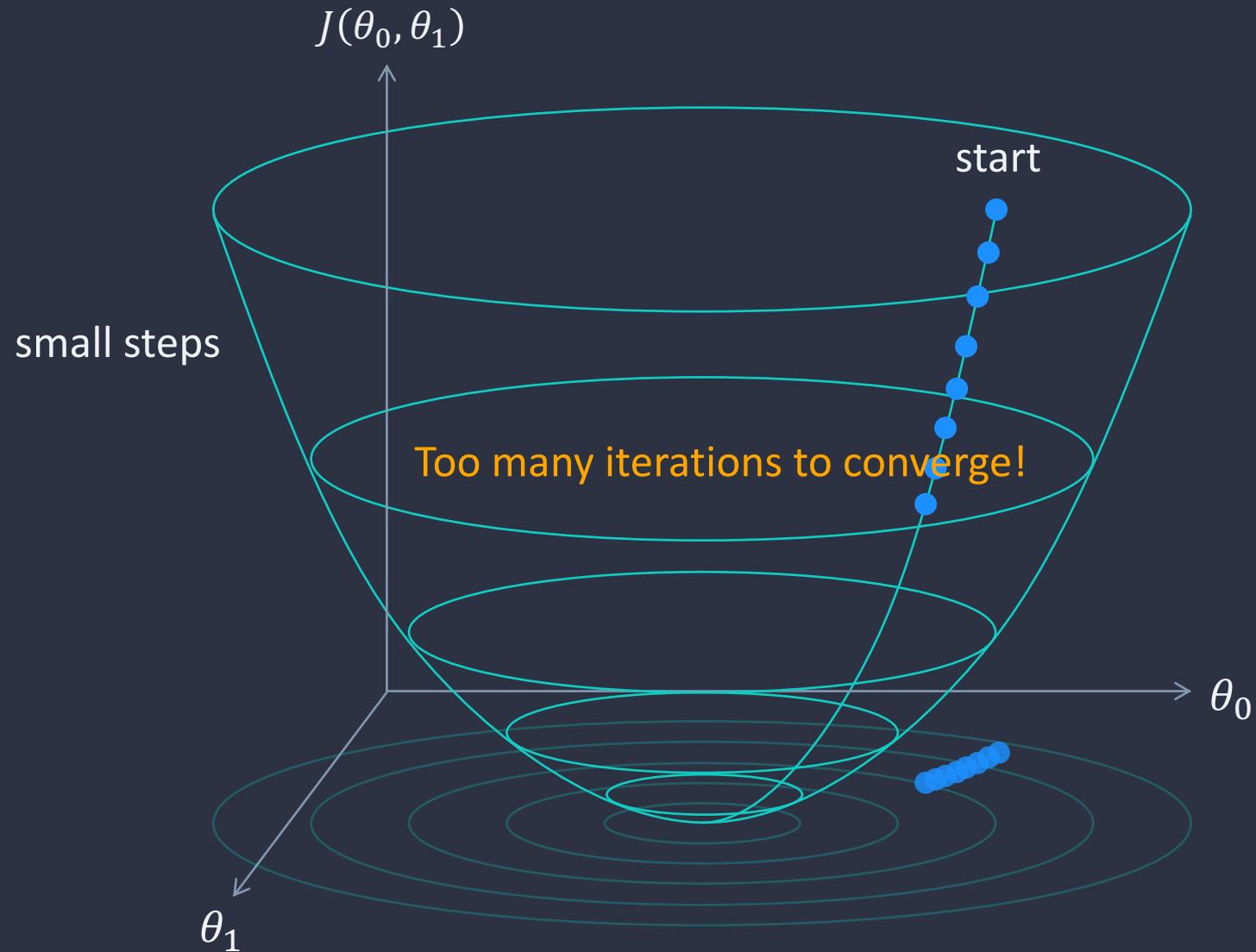
Once the gradient is zero,  
we reach a minimum, the  
bottom of the giant bowl.

# Gradient Decent - Intuition



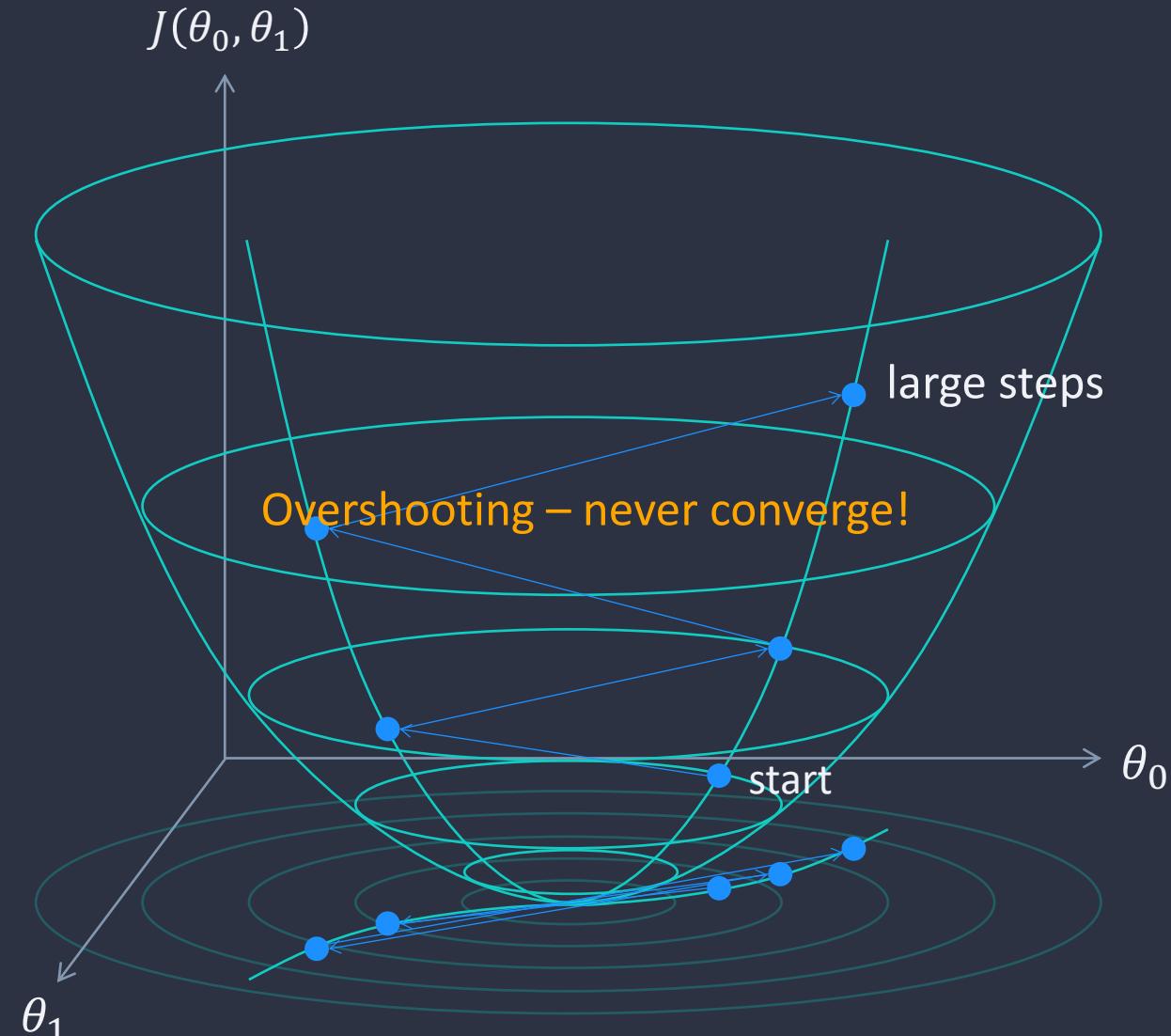
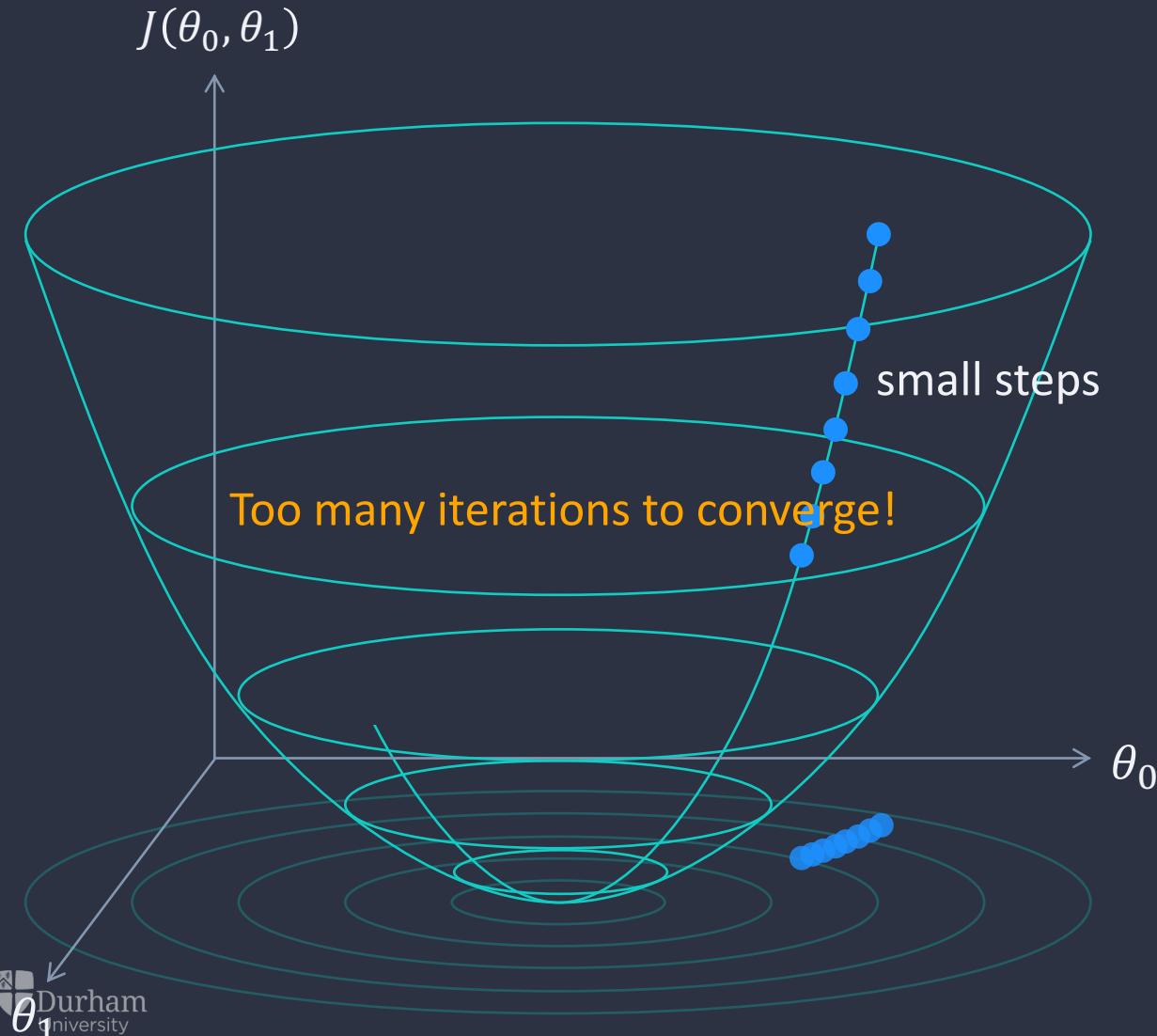
# Gradient Decent - Intuition

Hyperparameter: learning rate (size of step)

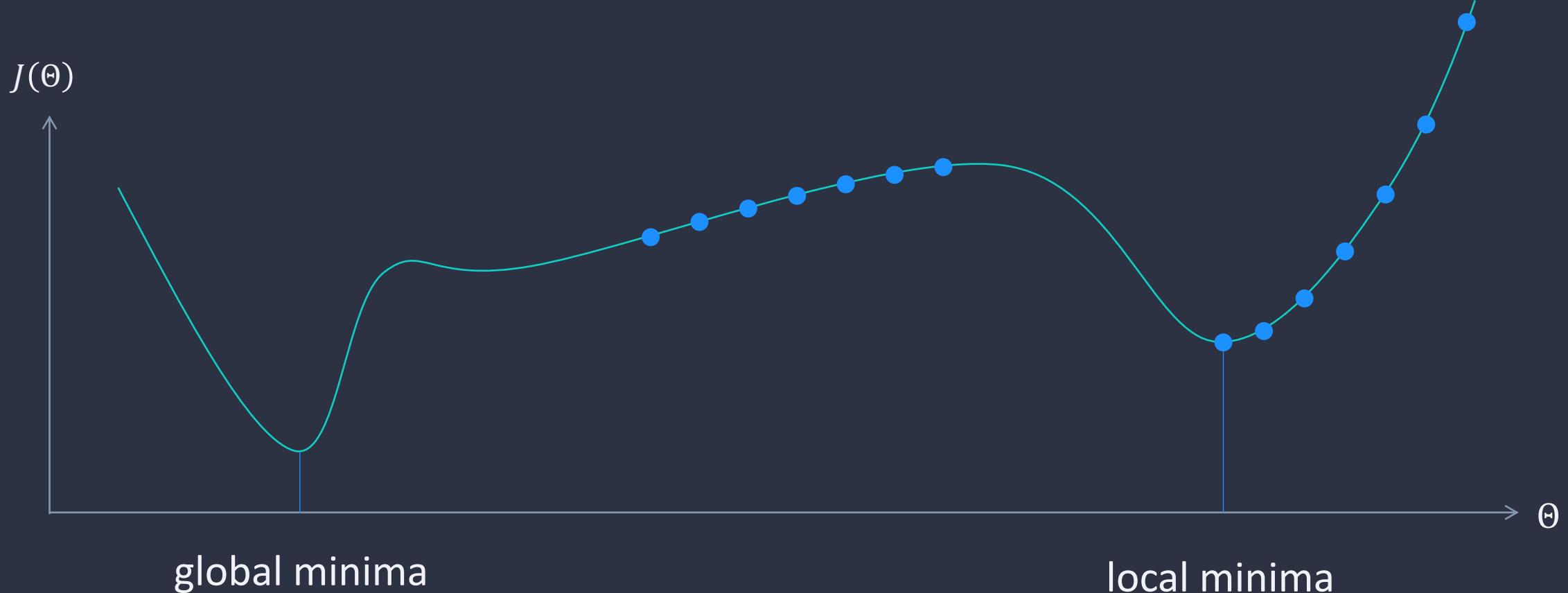


# Gradient Decent - Intuition

Hyperparameter: learning rate (size of step)

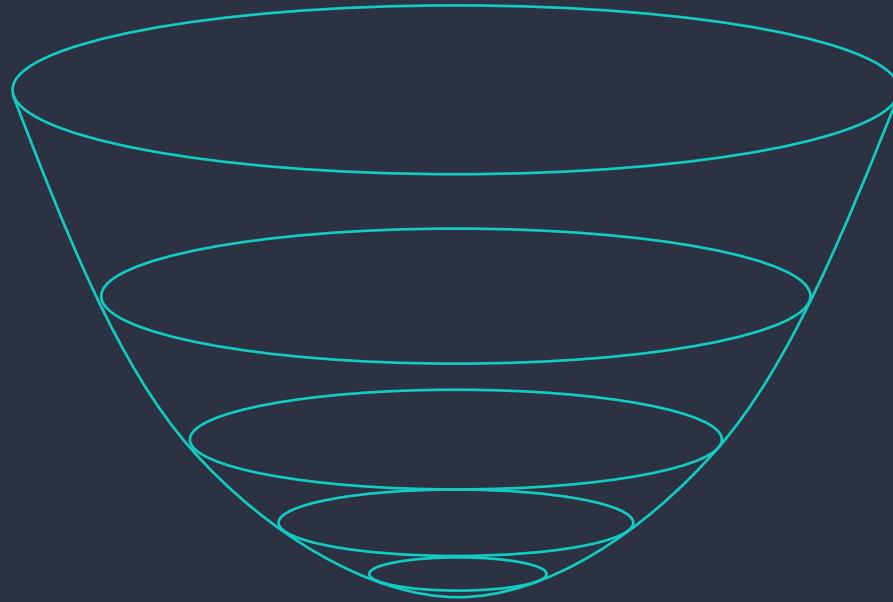


# Gradient Decent - Pitfall



- Learning algorithm may end up with the local minima not the global minima.
- Learning algorithm may stops before reaching the minima.

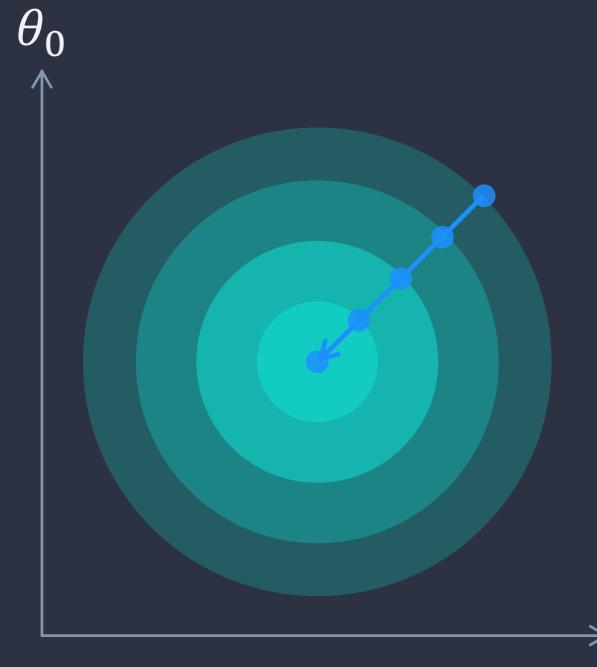
# Gradient Decent - Pitfall



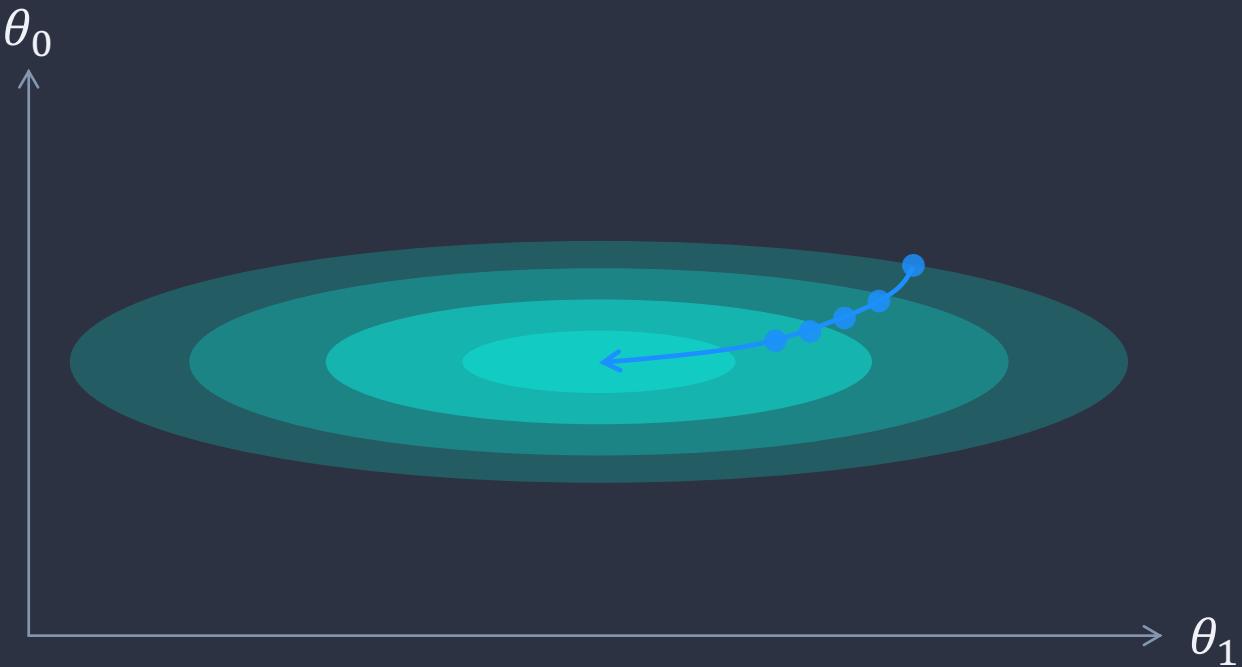
- MSE cost function for a linear regression model is always convex, meaning they are always shaped like a giant bowl with only one global minima.
- As long as we start somewhere on the giant bowl and we take steps in reasonable sizes and we follow the gradient, it is guaranteed to eventually approach to the global minima.

# Gradient Decent - Pitfall

- If features in training set have very different scales, and this means the model parameters or the cost function variables have very different scales, the cost function will be in an elongated bowl shape, and so the learning algorithm will take much longer time to converge.

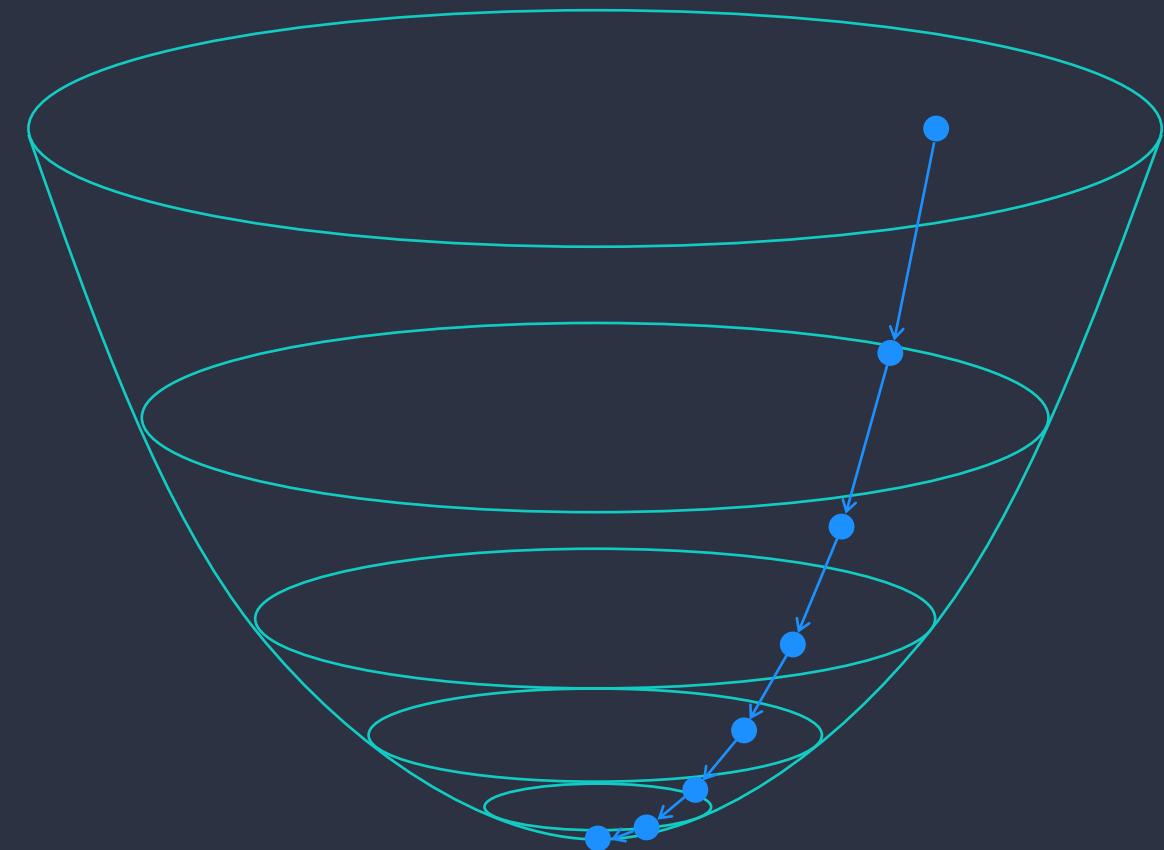


Gradient Descent with Feature Scaling



Gradient Descent without Feature Scaling

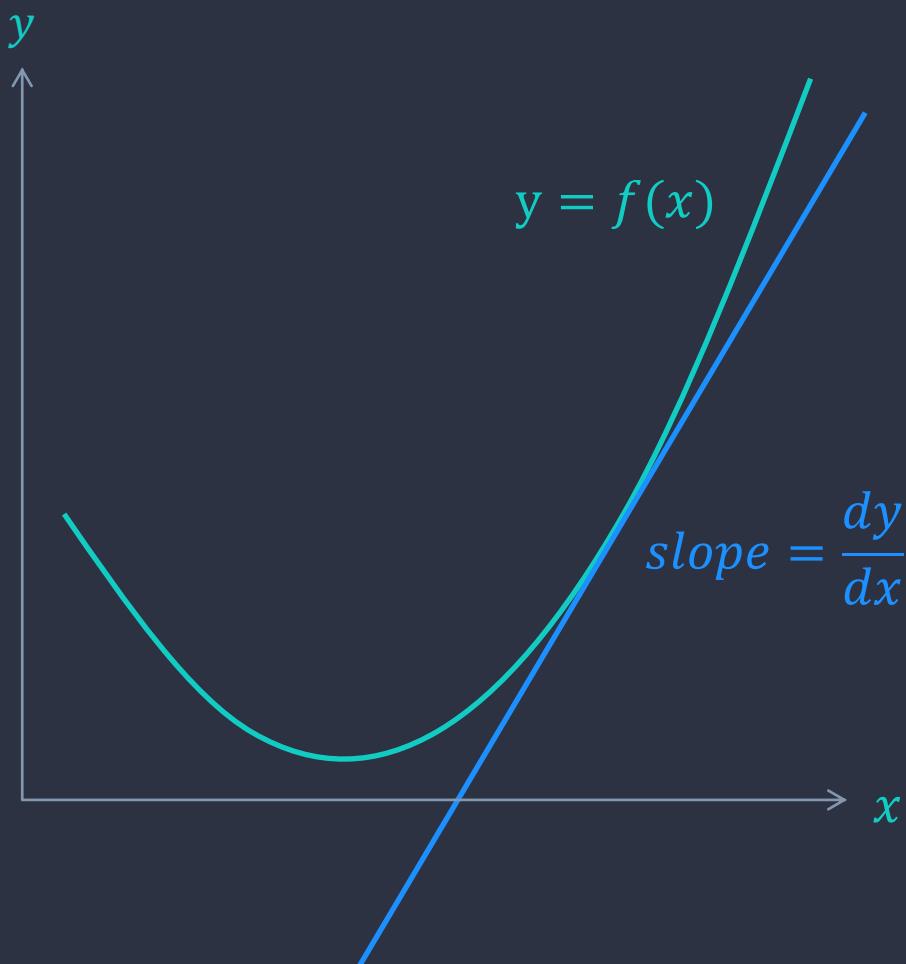
# Gradient Decent - Learning Algorithm



- In order to find the model parameter vector that minimises the cost function  $J(\theta_0, \theta_1)$ , we iteratively try different  $\theta_0$ s and  $\theta_1$ s in our learning algorithm, and gradient descent can help decide which  $\theta_0$ s and  $\theta_1$ s to try and when to stop.
- In order for a gradient descent learning algorithm to converge, i.e. reach the minima of the cost function, as quickly as possible, we need to change the model parameters in the direction of the steepest slope.

# Gradient Decent - Learning Algorithm

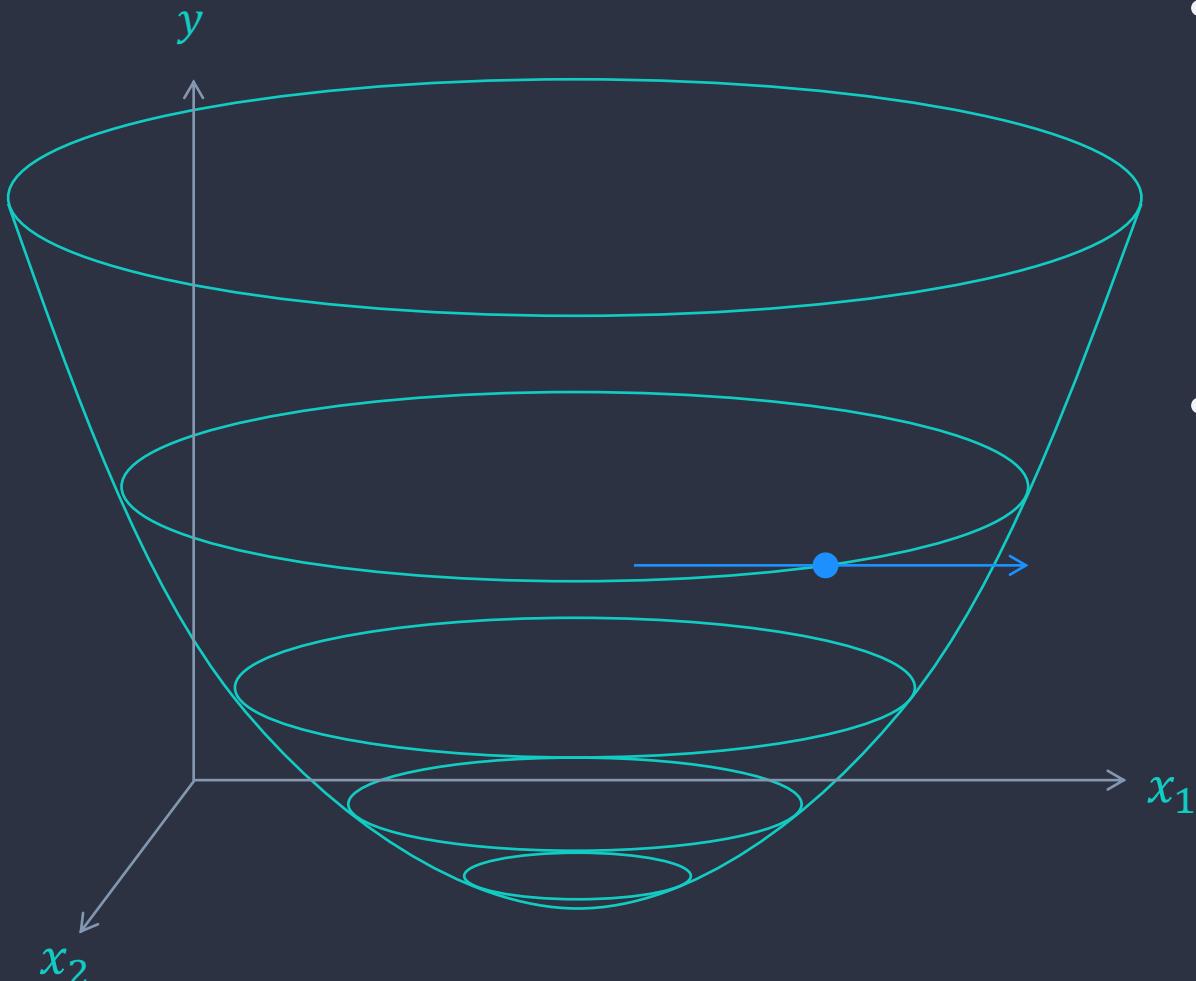
## Partial Derivative and Gradient



- Derivative at a point is the slope of the curve at that point, and it measures the steepness of the curve at that point.
- Derivative of  $y$  with respect to  $x$  :  $\frac{dy}{dx}$

# Gradient Decent - Learning Algorithm

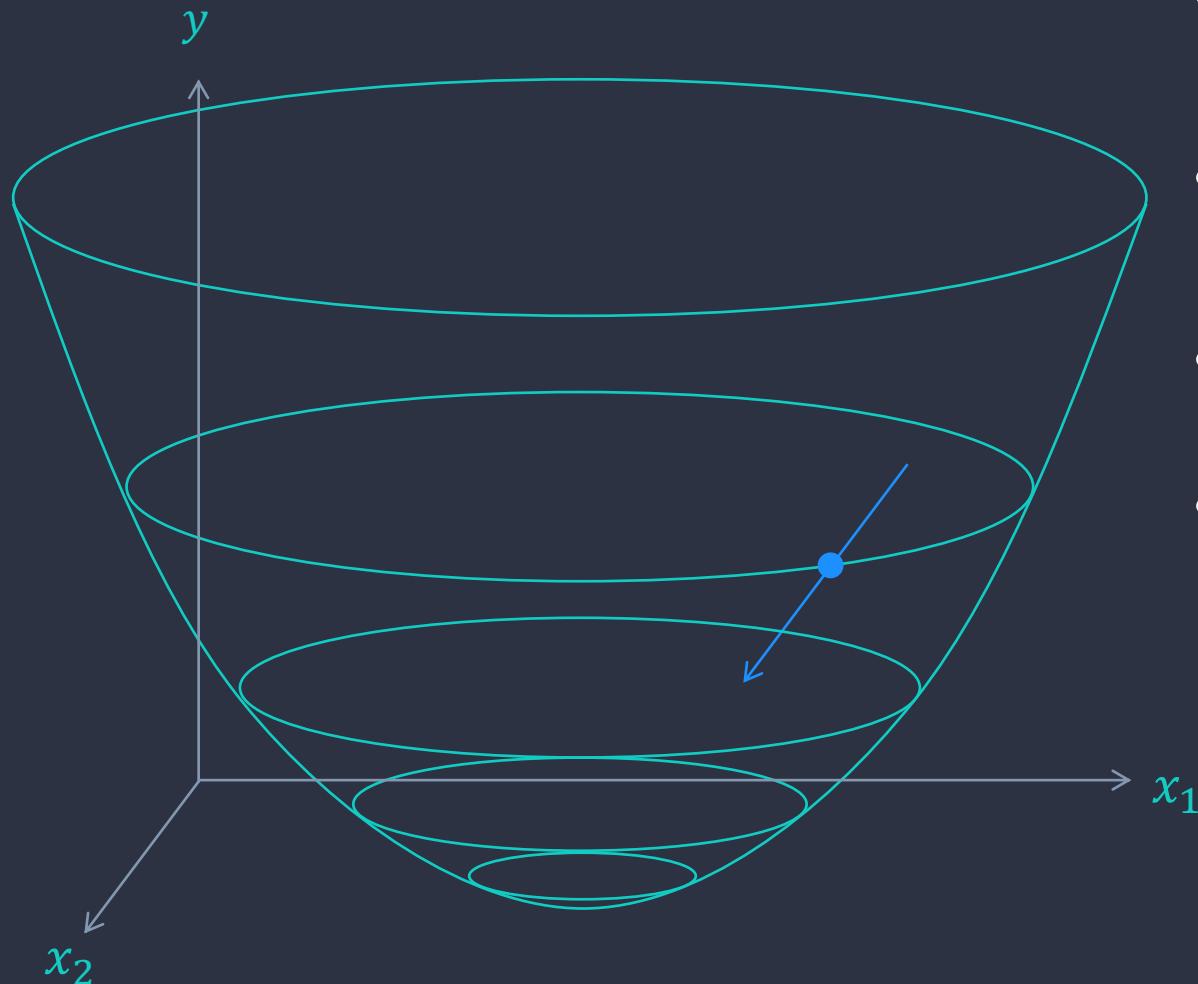
## Partial Derivative and Gradient



- For a surface, we measure the steepness at a point in specific directions independently, e.g. in the direction of  $x_1$  and in the direction of  $x_2$ .
- Partial Derivative of  $y$  with respect to  $x_1$  :  $\frac{\partial y}{\partial x_1}$

# Gradient Decent - Learning Algorithm

## Partial Derivative and Gradient



- Partial Derivative of  $y$  with respect to  $x_1$  :  $\frac{\partial y}{\partial x_1}$
- Partial Derivative of  $y$  with respect to  $x_2$  :  $\frac{\partial y}{\partial x_2}$
- Gradient of  $y$ :  $\text{grad } y = \frac{\partial}{\partial x_1} \vec{i} + \frac{\partial}{\partial x_2} \vec{j}$
- Gradient can be considered as the slop in higher dimensions. It points in the direction of maximum change of the function.

$$\nabla y = \left\langle \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2} \right\rangle$$

# Gradient Decent - Learning Algorithm

Univariate Linear Regression model  $h_{\theta}(x) = \theta^T x = [\theta_0, \theta_1] \begin{bmatrix} 1 \\ x \end{bmatrix} = \theta_0 + \theta_1 \cdot x$   
(one independent variable and one dependent variable)

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

actual label  
predicted label

# Gradient Decent - Learning Algorithm

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

# Gradient Decent - Learning Algorithm

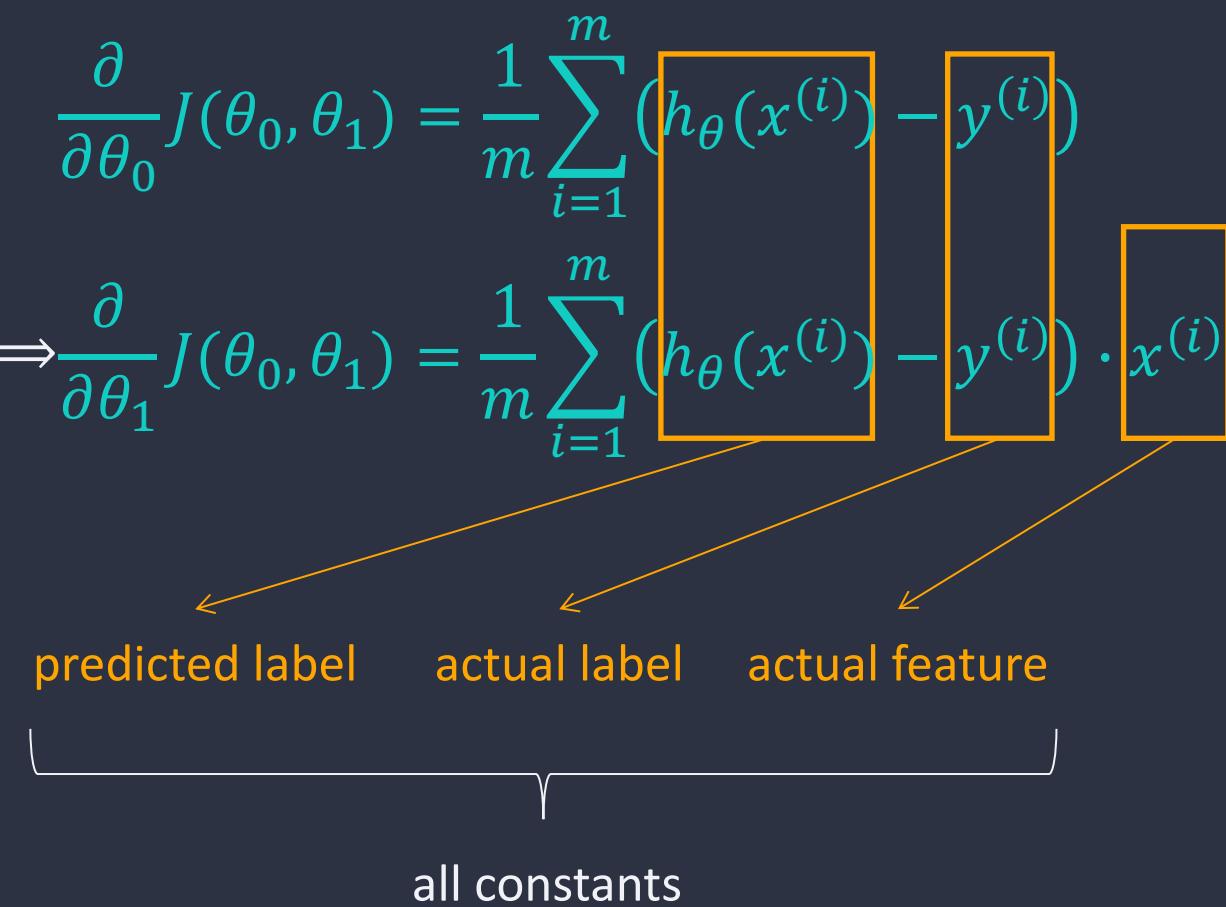
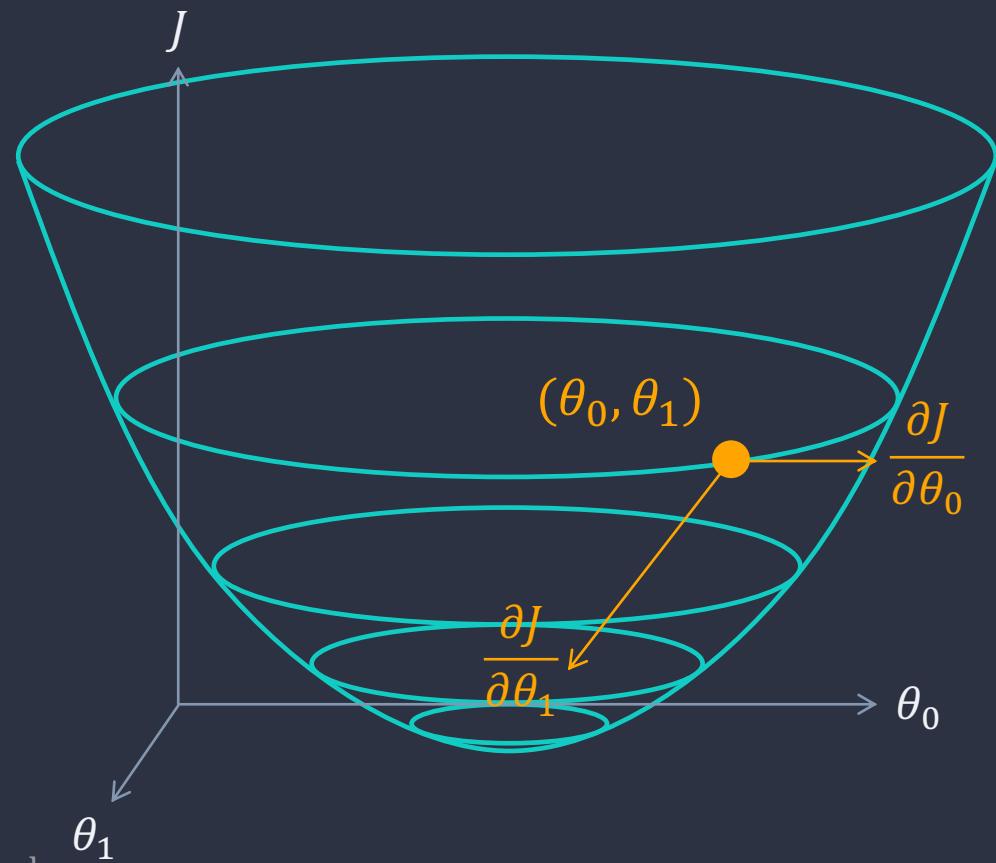
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\boxed{\theta_0 + \theta_1 \cdot x^{(i)}} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (\boxed{h_\theta(x^{(i)})} - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\boxed{\theta_0 + \theta_1 \cdot x^{(i)}} - y^{(i)}) \Rightarrow \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\boxed{h_\theta(x^{(i)})} - y^{(i)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\boxed{\theta_0 + \theta_1 \cdot x^{(i)}} - y^{(i)}) \cdot x^{(i)} \Rightarrow \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\boxed{h_\theta(x^{(i)})} - y^{(i)}) \cdot x^{(i)}$$

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

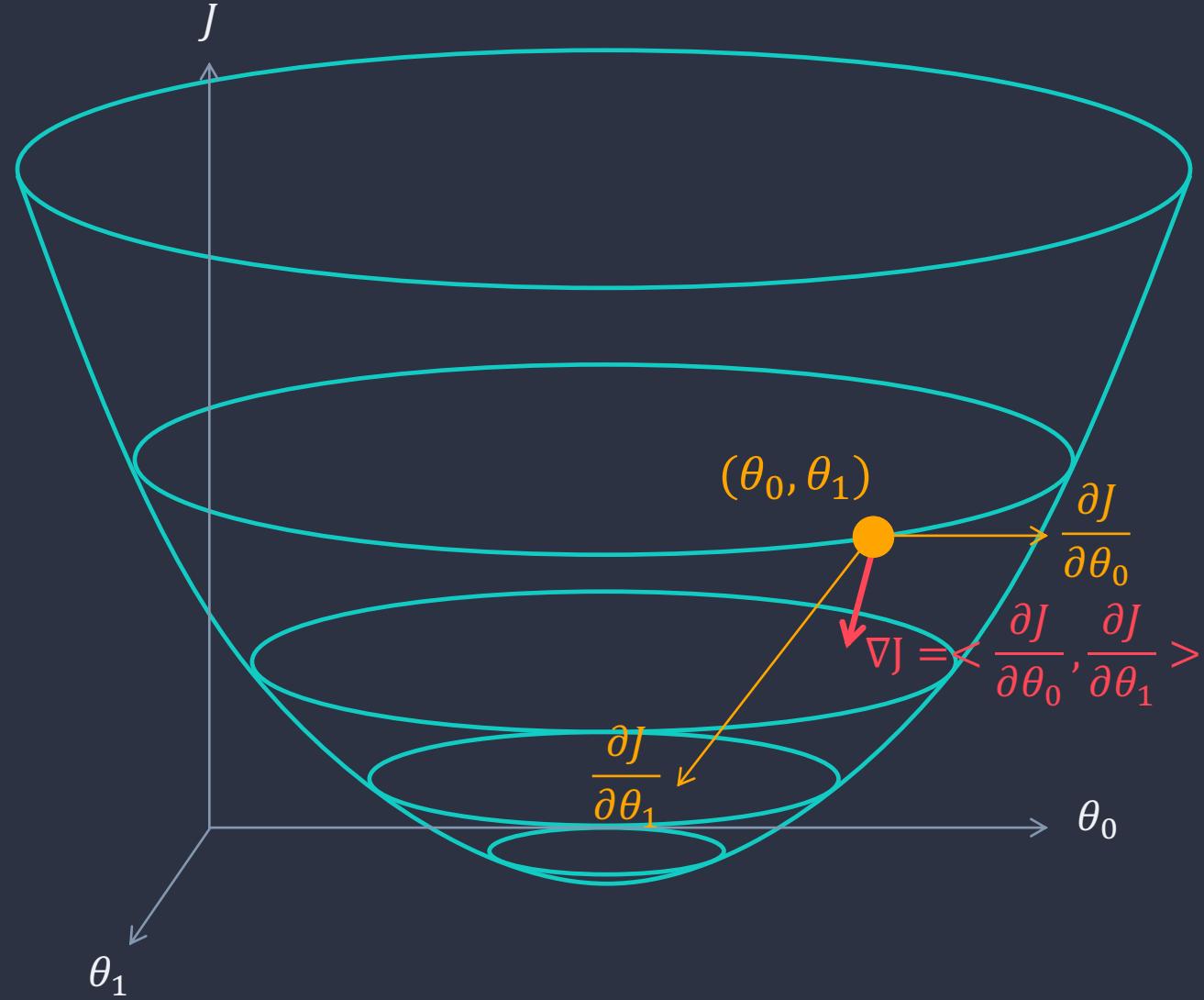
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x^{(i)} - y^{(i)}) \cdot x^{(i)} \Rightarrow \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$



Gradient:  $\nabla J = \left\langle \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1), \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \right\rangle$

Note:  $\frac{\partial J}{\partial \theta_0}$  is the same as  $\frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1)$

# Gradient Decent - Learning Algorithm

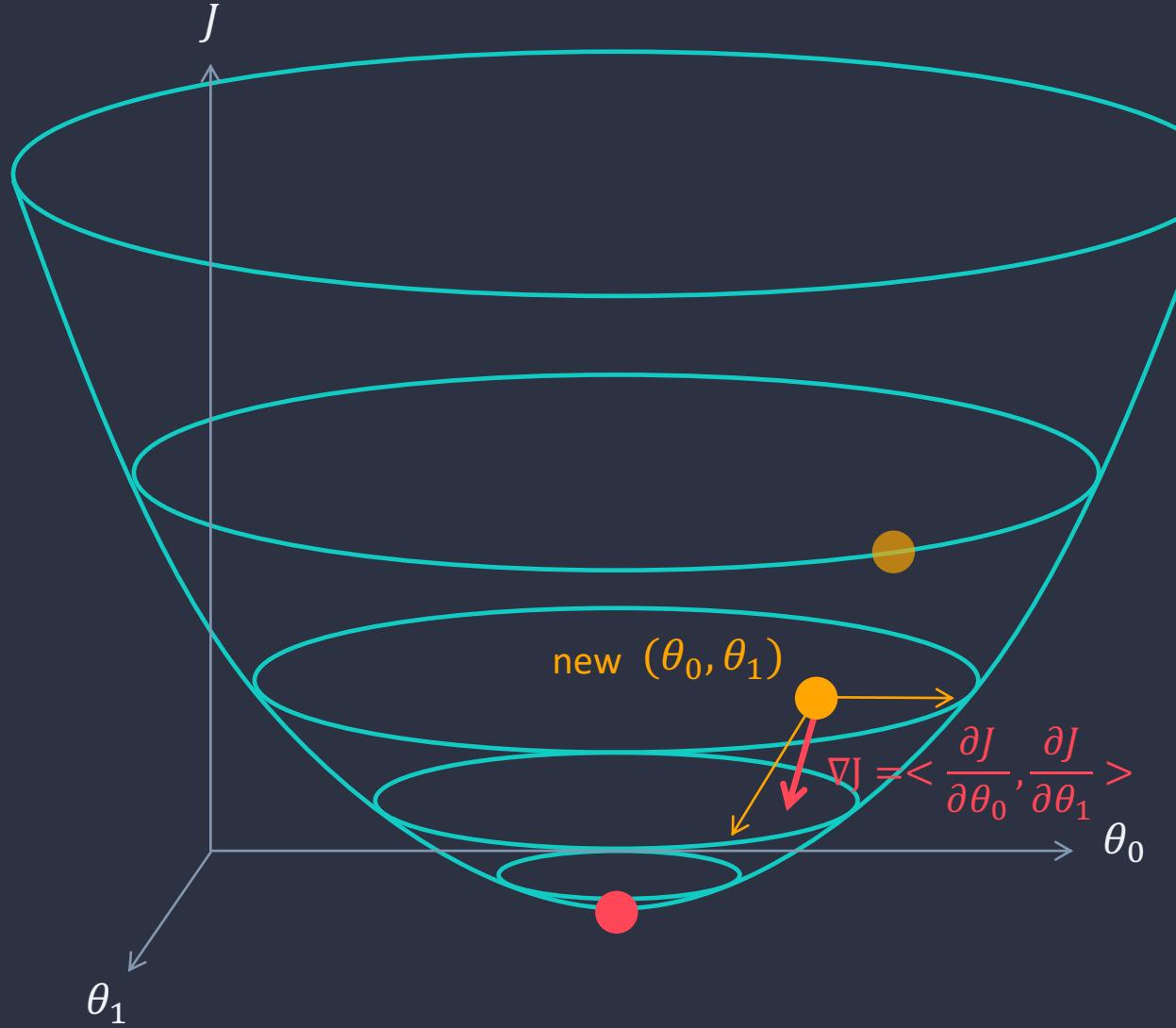


$$\theta_0 := \theta_0 - \boxed{\alpha} \cdot \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1)$$

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

Learning Rate  
(hyperparameter of the learning algorithm)

# Gradient Decent - Learning Algorithm

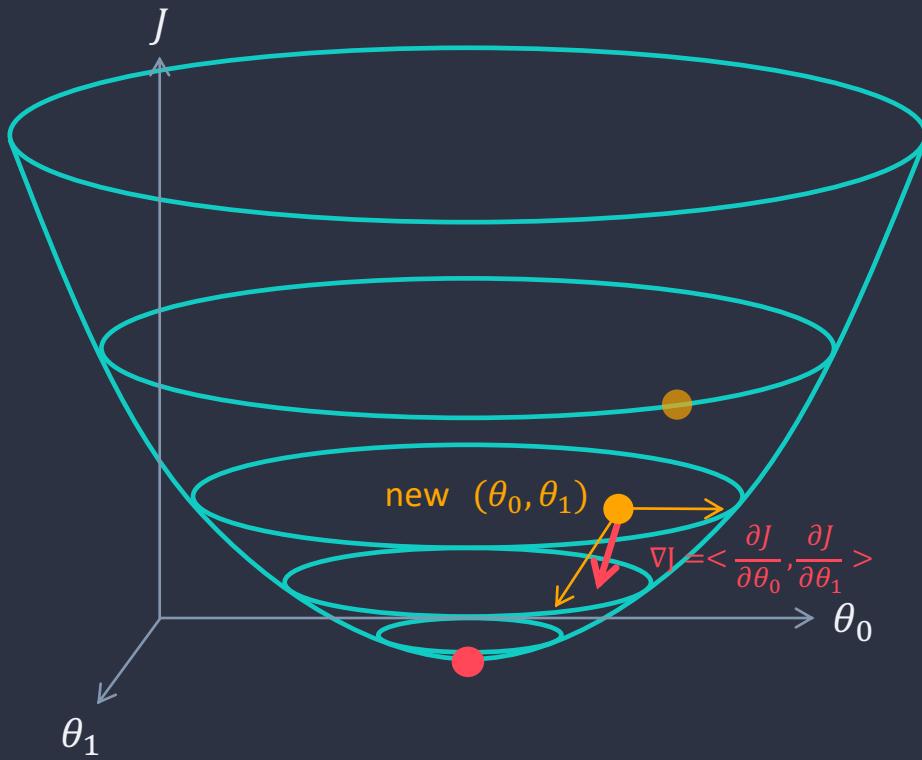


Repeat until convergence {

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

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

}



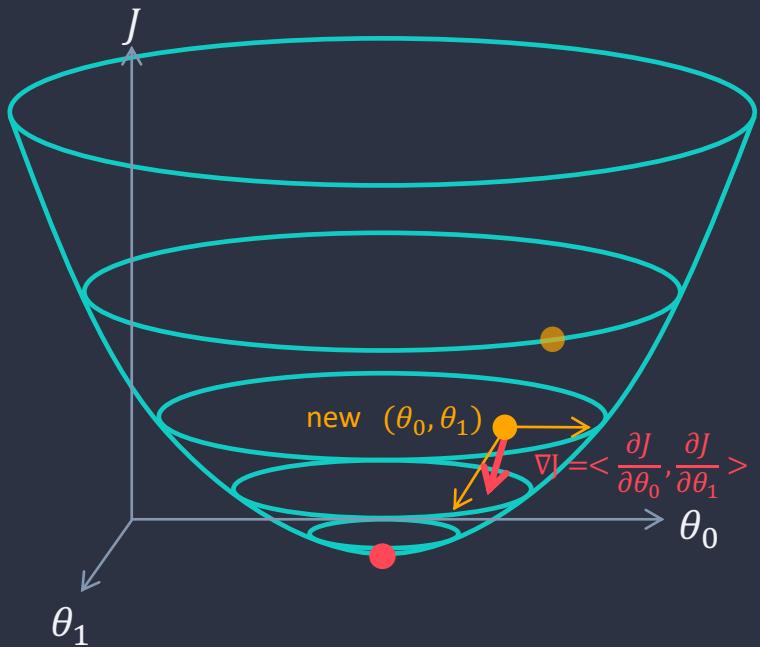
Repeat until convergence {

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

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

}

- Learning rate  $\alpha$  controls how big a step our gradient descent learning algorithm is moving towards a minima.
  - Large  $\alpha$ : learning algorithm is very aggressive (huge steps to a minima).
  - Small  $\alpha$ : learning algorithm only takes tiny steps towards a minia.



Repeat until convergence {

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

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

}

- To update all these model parameters ( $\theta$ s) simultaneously.

correct: simultaneously update

$$temp_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1)$$

$$temp_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1)$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

incorrect: not simultaneously update

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

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

# 4. Code in Practice (scikit-learn)

# Code in Practice (scikit-learn)

[https://scikit-learn.org/stable/auto\\_examples/linear\\_model/plot\\_ols.html](https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

```
# Split the data into training/testing sets  
diabetes_X_train = diabetes_X[:-20]  
diabetes_X_test = diabetes_X[-20:]
```

```
# Split the targets into training/testing sets  
diabetes_y_train = diabetes_y[:-20]  
diabetes_y_test = diabetes_y[-20:]
```

```
# Create linear regression object  
regr = linear_model.LinearRegression()
```

```
# Train the model using the training sets  
regr.fit(diabetes_X_train, diabetes_y_train)
```

```
# Make predictions using the testing set  
diabetes_y_pred = regr.predict(diabetes_X_test)
```

```
# The coefficients  
print('Coefficients: \n', regr.coef_)  
# The mean squared error
```



```
regressor.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regressor.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regressor.coef_)

# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))

# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))
```

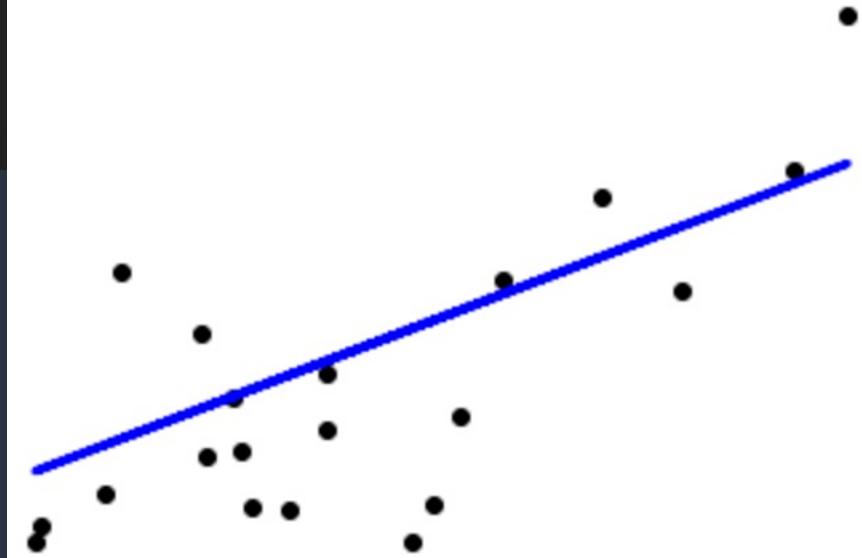
```
Coefficients:
[938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

# Code in Practice (scikit-learn)

```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



# Summary

# Summary

Hypothesis function  $h_{\theta}(x) = \theta_0 + \theta_1 x$

Cost function  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$

Gradient descent      Repeat until convergence {  
     $\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$   
     $\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$   
}

Next Lecture

Multivariate Linear Regression

**COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING**

# Multivariate Linear Regression

Dr Yang Long

# Last Lecture



## Univariate Linear Regression

One independent variable and one dependent variable

Hypothesis function  $h_{\theta}(x) = \theta_0 + \theta_1 \cdot x$

Cost function

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

Gradient descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$$

}

# Lecture Overview

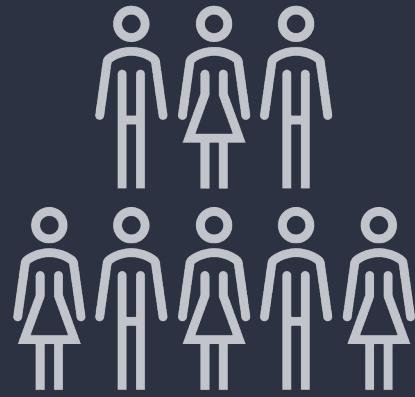


## Multivariate Linear Regression

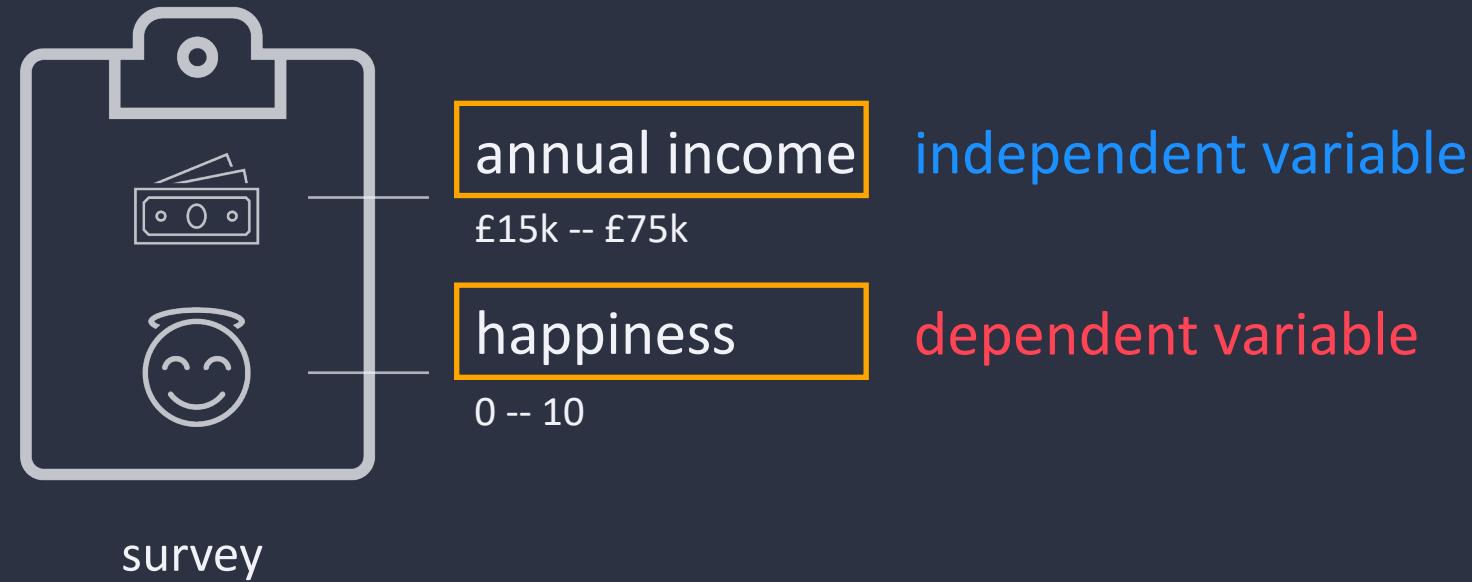
1. Multiple Variables
2. Cost Function and Gradient Descent
3. Vectorisation
4. Normal Equation

# 1. Multiple Variables

## EXAMPLE. Annual income to predict happiness



500 people in Durham



survey

A supervised learning task

-- we know “right answer”, correct label i.e. given annual income, we know exact happiness.

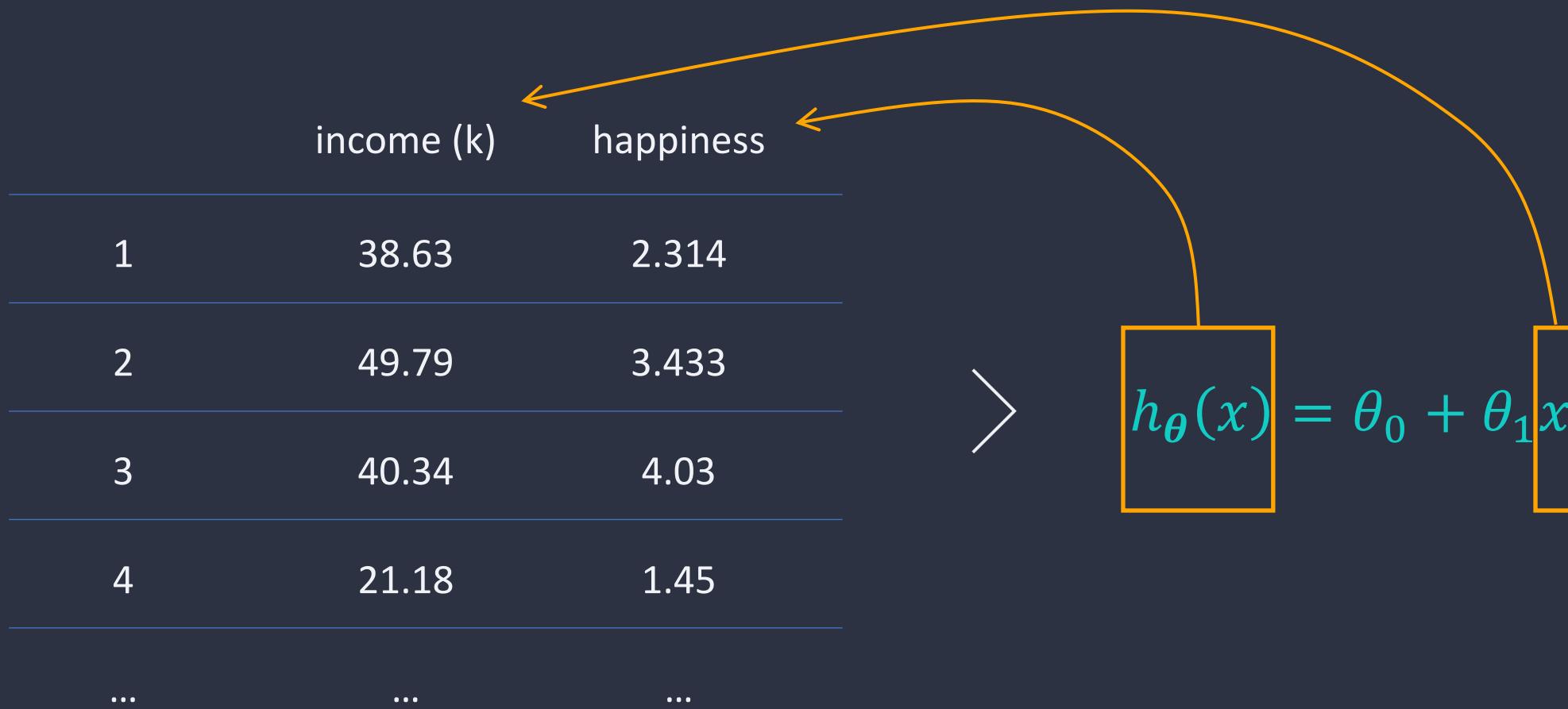
A regression task

-- we want to predict a real-valued output i.e. happiness in a continuous scale from 0 to 10.

A univariate regression task

-- there is a single independent variable, annual income.

## EXAMPLE. Annual income to predict happiness



EXAMPLE. Annual income & age, number of children, cups of tea/week to predict happiness

	1 income (k)	2 age	3 # of children	4 tea (cups/week)	happiness
1	38.63	46	1	7	2.314
2	49.79	37	0	15	3.433
3	40.34	52	3	20	4.03
4	21.18	25	0	3	1.45
...	...	...	...	...	...

$m$  number of training examples  $m = 350$

$n$  number of independent variables / features  $n = 4$

## EXAMPLE. Annual income & age, number of children, cups of tea/week to predict happiness

	income (k)	age	# of children	tea (cups/week)	happiness
1	38.63	46	1	7	2.314
2	49.79	37	0	15	3.433
3	40.34	52	3	20	4.03
4	21.18	25	0	3	1.45
...	...	...	...	...	...

$m$  number of training examples  $m = 350$

$n$  number of independent variables / features  $n = 4$

$y^{(i)}$  the label of the  $i^{th}$  training example  $y^{(3)} = 4.03$

EXAMPLE. Annual income & age, number of children, cups of tea/week to predict happiness

	income (k)	age	# of children	tea (cups/week)	happiness
1	38.63	46	1	7	2.314
2	49.79	37	0	15	3.433
3	40.34	52	3	20	4.03
4	21.18	25	0	3	1.45
...	...	...	...	...	...

$m$  number of training examples  $m = 350$

$n$  number of independent variables / features  $n = 4$

$y^{(i)}$  the label of the  $i^{th}$  training example  $y^{(3)} = 4.03$

$x_j^{(i)}$  the  $j^{th}$  feature of the  $i^{th}$  training example  $x_4^{(2)} = 15$

EXAMPLE. Annual income & age, number of children, cups of tea/week to predict happiness

	income (k)	age	# of children	tea (cups/week)	happiness
1	38.63	46	1	7	2.314
2	49.79	37	0	15	3.433
3	40.34	52	3	20	4.03
4	21.18	25	0	3	1.45
...	...	...	...	...	...

$m$  number of training examples  $m = 350$

$n$  number of independent variables / features  $n = 4$

$y^{(i)}$  the label of the  $i^{th}$  training example  $y^{(3)} = 4.03$

$x_j^{(i)}$  the  $j^{th}$  feature of the  $i^{th}$  training example  $x_4^{(2)} = 15$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

feature vector

$$\mathbf{x}^{(1)} = \begin{bmatrix} 38.63 \\ 46 \\ 1 \\ 7 \end{bmatrix}$$

$\in \mathbb{R}^4$

EXAMPLE. Annual income & age, number of children, cups of tea/week to predict happiness

	income (k)	age	# of children	tea (cups/week)	happiness
1	38.63	46	1	7	2.314
2	49.79	37	0	15	3.433
3	40.34	52	3	20	4.03
4	21.18	25	0	3	1.45
...	...	...	...	...	...

Hypothesis Function  $h_{\theta}(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 + \theta_4 \cdot x_4$

$$h_{\theta}(x) = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4] \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$x_0 = 1$   
constant

# Hypothesis Function

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 + \theta_4 \cdot x_4 + \cdots + \theta_n \cdot x_n$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = [\theta_0, \theta_1, \theta_2, \dots, \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$     Multivariate Linear Regression (with multivariable i.e. multiple variables.)

## 2. Cost Function and Gradient Descent

# Supervised Learning

- To build a model represented as a hypothesis function  $h_{\theta}(x)$ .



# Supervised Learning

input  $x$  (dependent variable)



model (hypothesis function, mapping  $x \rightarrow y$ )

cost function



output  $y$  (independent variable)

# Univariate Linear Regression

Hypothesis Function

$$\hat{y} = h_{\theta}(x) = \theta_0 + \theta_1 x$$



# Multivariate Linear Regression

Independent Variable

$$x$$



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

Model Parameters

$$\theta_0, \theta_1$$



$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad (x_0 = 1, \text{constant})$$

Cost Function

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



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

vectors

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_{\theta}(x) = \theta^T x$$

# Univariate Linear Regression $(n = 1)$

## Cost Function

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

# Multivariate Linear Regression

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

Partial derivative of  $J$  with respect to parameters

$$\frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x)^{(i)} - y^{(i)}) \boxed{x_0^{(i)}} \Big|_{x_0^{(i)} = 1}$$

$$\frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x)^{(i)} - y^{(i)}) \boxed{x_1^{(i)}}$$

$$\frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)}) \boxed{x_0^{(i)}} \Big|_{x_0^{(i)} = 1}$$

$$\frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)}) \boxed{x_1^{(i)}}$$

...

$$\frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)}) x_n^{(i)}$$

# Univariate Linear Regression

## Gradient Descent

Repeat until convergence {

$$\frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1)$$

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

learning rate

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_\theta(x)^{(i)} - y^{(i)})x^{(i)}$$

learning rate

} simultaneously update  $\theta_0, \theta_1$

$$\frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1)$$

# Multivariate Linear Regression

Repeat until convergence {

$$\frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1, \dots, \theta_n)$$

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

learning rate

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

learning rate

...

$$\frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1, \dots, \theta_n)$$

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

learning rate

} simultaneously update  $\theta_0, \dots, \theta_n$

$$\frac{\partial J}{\partial \theta_n}(\theta_0, \theta_1, \dots, \theta_n)$$

# 3. Vectorisation

# Vectorisation of Hypothesis Function

# Vectorisation of Hypothesis Function

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



$$h_{\theta}(x) = \theta^T x$$

# Vectorisation of Hypothesis Function

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



$$h_{\theta}(x) = \theta_0 \cdot \boxed{1} + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n$$

$$= [\theta_0, \theta_1, \theta_2, \dots, \theta_n] \cdot \begin{matrix} \\ \end{matrix}$$

$\mathbb{R}^{1 \times (n+1)}$

$$\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$\mathbb{R}^{(n+1) \times 1}$

# Vectorisation of Hypothesis Function

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



$$h_{\theta}(x) = \theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

parameter vector

variable vector

# Vectorisation of Hypothesis Function

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



$$h_{\theta}(x) = \theta_0 \cdot x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n \quad (x_0 = 1, \text{constant})$$

$$= [\theta_0, \theta_1, \theta_2, \dots, \theta_n] \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$= \theta^T x$  for one single example

# Vectorisation of Hypothesis Function for the whole Training Set

## EXAMPLE. Happiness predictor

Features $\mathbf{x}$ s				Label $y$
income (k)	age	# of children	tea (cups/week)	happiness
38.63	46	1	7	2.314
49.79	37	0	15	3.433
40.34	52	3	20	4.03
21.18	25	0	3	1.45
...	...	...	...	...

## EXAMPLE. Happiness predictor

$x_1$	$x_2$	$x_3$	$x_4$	$y$
income (k)	age	# of children	tea (cups/week)	happiness
38.63	46	1	7	2.314
49.79	37	0	15	3.433
40.34	52	3	20	4.03
21.18	25	0	3	1.45
...	...	...	...	...

## EXAMPLE. Happiness predictor

$x_1$	$x_2$	$x_3$	$x_4$	...	$x_n$	$y$
income (k)	age	# of children	tea (cups/week)	...	the n <sup>th</sup> feature	happiness
38.63	46	1	7	...	*	2.314
49.79	37	0	15	...	*	3.433
40.34	52	3	20	...	*	4.03
21.18	25	0	3	...	*	1.45
...	...	...	...	...	...	...

## EXAMPLE. Happiness predictor

	income (k)	age	# of children	tea (cups/week)	...	the n <sup>th</sup> feature	happiness
1 <sup>st</sup> example	38.63	46	1	7	...	*	2.314
2 <sup>nd</sup> example	49.79	37	0	15	...	*	3.433
3 <sup>rd</sup> example	40.34	52	3	20	...	*	4.03
4 <sup>th</sup> example	21.18	25	0	3	...	*	1.45
...	...	...	...	...	...	...	...

## EXAMPLE. Happiness predictor

	income (k)	age	# of children	tea (cups/week)	...	the n <sup>th</sup> feature	happiness
$(\mathbf{x}^{(1)})^T$	[ 38.63 ]	46	1	7	...	*	2.314 <span style="border: 1px solid orange; padding: 2px;">y<sup>(1)</sup></span>
$(\mathbf{x}^{(2)})^T$	[ 49.79 ]	37	0	15	...	*	3.433 <span style="border: 1px solid orange; padding: 2px;">y<sup>(2)</sup></span>
$(\mathbf{x}^{(3)})^T$	[ 40.34 ]	52	3	20	...	*	4.03 <span style="border: 1px solid orange; padding: 2px;">y<sup>(3)</sup></span>
$(\mathbf{x}^{(4)})^T$	[ 21.18 ]	25	0	3	...	*	1.45 <span style="border: 1px solid orange; padding: 2px;">y<sup>(4)</sup></span>
...	...	...	...	...	...	...	...
$(\mathbf{x}^{(m)})^T$	[ * ]	*	*	*	...	*	1.45 <span style="border: 1px solid orange; padding: 2px;">y<sup>(m)</sup></span>

## EXAMPLE. Happiness predictor

For each example, insert an extra feature and set its value to constant 1.

	income (k)	age	# of children	tea (cups/week)	...	the n <sup>th</sup> feature	happiness	
$(\mathbf{x}^{(1)})^T$	1 38.63	46	1	7	...	*	2.314	$y^{(1)}$
$(\mathbf{x}^{(2)})^T$	1 49.79	37	0	15	...	*	3.433	$y^{(2)}$
$(\mathbf{x}^{(3)})^T$	1 40.34	52	3	20	...	*	4.03	$y^{(3)}$
$(\mathbf{x}^{(4)})^T$	1 21.18	25	0	3	...	*	1.45	$y^{(4)}$
...	...	...	...	...	...	...	...	...
$(\mathbf{x}^{(m)})^T$	1 *	*	*	*	...	*	1.45	$y^{(m)}$

## EXAMPLE. Happiness predictor

$$(\mathbf{x}^{(2)})^T = [ \begin{array}{ccccccc} x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \dots & x_n^{(2)} \\ 1 & 49.79 & 37 & 0 & 15 & \dots & * \end{array} ]$$

## EXAMPLE. Happiness predictor

$$\boldsymbol{x}^{(2)} = \begin{bmatrix} 1 \\ 49.79 \\ 37 \\ 0 \\ 15 \\ \dots \\ * \end{bmatrix} \quad x_0^{(2)} \\ x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \\ \dots \\ x_n^{(2)}$$
$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ x_4^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix}$$

For the  $i$ -th example

## EXAMPLE. Happiness predictor

$$(\mathbf{x}^{(2)})^T = [ \begin{array}{ccccccc} x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \dots & x_n^{(2)} \\ 1 & 49.79 & 37 & 0 & 15 & \dots & * \end{array} ]$$

## EXAMPLE. Happiness predictor

$X$

	income (k)	age	# of children	tea (cups/week)	...	the n <sup>th</sup> feature	happiness
$(\mathbf{x}^{(1)})^T$	1 38.63	46	1	7	...	*	2.314 $y^{(1)}$
$(\mathbf{x}^{(2)})^T$	1 49.79	37	0	15	...	*	3.433 $y^{(2)}$
$(\mathbf{x}^{(3)})^T$	1 40.34	52	3	20	...	*	4.03 $y^{(3)}$
$(\mathbf{x}^{(4)})^T$	1 21.1	25	0	3	...	*	1.45 $y^{(4)}$
...	...	...	...	...	...	...	...
$(\mathbf{x}^{(m)})^T$	1 *	*	*	*	...	*	1.45 $y^{(m)}$

# Vectorisation of Hypothesis Function

$$X = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T \\ (\boldsymbol{x}^{(2)})^T \\ (\boldsymbol{x}^{(3)})^T \\ (\boldsymbol{x}^{(4)})^T \\ \dots \\ (\boldsymbol{x}^{(m)})^T \end{bmatrix}$$

# Vectorisation of Hypothesis Function

$$X = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ (\mathbf{x}^{(3)})^T \\ (\mathbf{x}^{(4)})^T \\ \vdots \\ (\mathbf{x}^{(m)})^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_j^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_j^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_j^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & x_2^{(i)} & \cdots & x_j^{(i)} & \cdots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

# Vectorisation of Hypothesis Function

$$X = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ (\mathbf{x}^{(3)})^T \\ (\mathbf{x}^{(4)})^T \\ \vdots \\ (\mathbf{x}^{(m)})^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_j^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_j^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_j^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & x_2^{(i)} & \cdots & x_j^{(i)} & \cdots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix}$$

# Vectorisation of Hypothesis Function

$$X\theta = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_j^{(2)} & \dots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \dots & x_j^{(3)} & \dots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & x_2^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix}$$

$\mathbb{R}^{m \times (n+1)}$

$\mathbb{R}^{(n+1) \times 1}$

$\mathbb{R}^{m \times 1}$

# Vectorisation of Hypothesis Function

$$X\theta = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_j^{(2)} & \dots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \dots & x_j^{(3)} & \dots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & x_2^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 + \theta_1 \cdot x_1^{(1)} + \theta_2 \cdot x_2^{(1)} + \dots + \theta_n \cdot x_n^{(1)} \\ \theta_0 + \theta_1 \cdot x_1^{(2)} + \theta_2 \cdot x_2^{(2)} + \dots + \theta_n \cdot x_n^{(2)} \\ \theta_0 + \theta_1 \cdot x_1^{(3)} + \theta_2 \cdot x_2^{(3)} + \dots + \theta_n \cdot x_n^{(3)} \\ \vdots \\ \theta_0 + \theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)} + \dots + \theta_n \cdot x_n^{(i)} \\ \vdots \\ \theta_0 + \theta_1 \cdot x_1^{(m)} + \theta_2 \cdot x_2^{(m)} + \dots + \theta_n \cdot x_n^{(m)} \end{bmatrix}$$

$\mathbb{R}^{m \times (n+1)}$        $\mathbb{R}^{(n+1) \times 1}$        $\mathbb{R}^{m \times 1}$

# Vectorisation of Hypothesis Function

$$X\theta = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_j^{(2)} & \dots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \dots & x_j^{(3)} & \dots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & x_2^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix} = \boxed{\begin{bmatrix} h_{\theta}(x^{(1)}) \\ h_{\theta}(x^{(2)}) \\ h_{\theta}(x^{(3)}) \\ \dots \\ h_{\theta}(x^{(i)}) \\ \dots \\ h_{\theta}(x^{(m)}) \end{bmatrix}}$$

Vectorised Hypothesis Function  
→  $h_{\theta}(X) = X \theta$

vector / matrix

A vector containing the results of the hypothesis function, i.e.,  
the predicted value, for all the examples from the training set.

# Vectorisation of Hypothesis Function

$$h_{\theta}(x) = \theta^T x$$

vs

$$h_{\theta}(X) = X \theta$$

takes single example

outputs a predicted label

takes all examples

outputs a vector of predicted labels

# Vectorisation of Cost Function

# Vectorisation of Cost Function

$$\mathbf{X}\boldsymbol{\theta} - \mathbf{y} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)} \\ \vdots \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)} \end{bmatrix}$$

# Vectorisation of Cost Function

$$X\theta - y = \begin{bmatrix} h_{\theta}(x^{(1)}) - y^{(1)} \\ h_{\theta}(x^{(2)}) - y^{(2)} \\ h_{\theta}(x^{(3)}) - y^{(3)} \\ \vdots \\ h_{\theta}(x^{(m)}) - y^{(m)} \end{bmatrix}$$



# Vectorisation of Cost Function

$$(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \cdot (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$= [h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)}, h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)}, h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)}, \dots, h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)}] \cdot \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)} \\ \vdots \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)} \end{bmatrix}$$

# Vectorisation of Cost Function

$$(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \cdot (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$= [h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)}, h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)}, h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)}, \dots, h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)}] \cdot \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)} \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)} \\ \vdots \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)} \end{bmatrix}$$

$$= (h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)})^2 + (h_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}) - y^{(2)})^2 + (h_{\boldsymbol{\theta}}(\mathbf{x}^{(3)}) - y^{(3)})^2 + \dots + (h_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) - y^{(m)})^2$$

$$= \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

# Vectorisation of Cost Function



$$(X\theta - y)^T \cdot (X\theta - y) = \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

# Vectorisation of Cost Function

$$\sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 = (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

# Vectorisation of Cost Function

$$\boxed{\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2} = \frac{1}{2m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2 = \frac{1}{2m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

# Vectorisation of Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

# Vectorisation of Gradient Descent

# Vectorisation of Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \boxed{\frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta})}$$

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta})}$$

...

$$\theta_n := \theta_n - \alpha \boxed{\frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta})}$$

key is how to vectorise these partial derivative components

} simultaneously update  $\theta_0, \dots, \theta_n$

# Vectorisation of Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta})$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta})$$

...

$$\theta_n := \theta_n - \alpha \frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta})$$

}

$$\begin{aligned}\boxed{\frac{\partial J}{\partial \theta_k}(\boldsymbol{\theta})} &= \frac{1}{m} \sum_{i=1}^m \frac{(h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)})x_k^{(i)}}{\text{scalar}} \\ &= \frac{1}{m} \sum_{i=1}^m \underline{x_k^{(i)}} \frac{(h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)})}{\text{scalar}} \\ &= \frac{1}{m} \mathbf{x}_k^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})\end{aligned}$$

use the product of  
two matrix to replace  
the sigma notation

*switch the position*

# Vectorisation of Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \left[ \frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta}) \right]$$
$$\theta_1 := \theta_1 - \alpha \left[ \frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta}) \right]$$

...

$$\theta_n := \theta_n - \alpha \left[ \frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta}) \right]$$

*Replace all these partial derivatives with the vectorized version.*

$$\frac{\partial J}{\partial \theta_k}(\boldsymbol{\theta}) = \boxed{\frac{1}{m} \mathbf{x}_k^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})}$$

}

# Vectorisation of Gradient Descent

Repeat until convergence {

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \mathbf{x}_0^T (\mathbf{X}\theta - \mathbf{y}) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \mathbf{x}_1^T (\mathbf{X}\theta - \mathbf{y}) \\ &\dots \\ \theta_n &:= \theta_n - \alpha \frac{1}{m} \mathbf{x}_n^T (\mathbf{X}\theta - \mathbf{y})\end{aligned}$$

}

# Vectorisation of Gradient Descent

Repeat until convergence {

$$\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - y)$$

}

- Vectorisation of Hypothesis Function
- Vectorisation of Cost Function
- Vectorisation of Gradient Descent

- Vectorisation of Hypothesis Function

$$h_{\theta}(x) = \theta^T x \quad h_{\theta}(x) = X \theta$$

- Vectorisation of Cost Function

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

- Vectorisation of Gradient Descent

Repeat until convergence {

$$\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - y)$$

}

Comparing with un-vectorised

- ✓ Easier to implement
- ✓ Computationally more efficient

# 4. Normal Equation

# Normal Equation

## Cost Function

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

To find  $\theta_0, \theta_1, \dots, \theta_n$  that minimise  $J$

# Normal Equation

## Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta})$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta})$$

...

$$\theta_n := \theta_n - \alpha \frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta})$$

In each iteration, we simultaneously update all the parameter thetas, so that we can get a smaller cost comparing to the last iteration. Thus after a series of iterations, once new parameter thetas can no longer decrease the cost, or decrease the cost very little, we stop the iterations, and this pair of thetas will be our optimal parameters for our regression model.

} simultaneously update  $\theta_0, \dots, \theta_n$

# Normal Equation

## Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \left[ \begin{array}{c} \frac{\partial J}{\partial \theta_0}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta}) \end{array} \right]$$

}

learnpingrate derivative

# Normal Equation for Univariate Linear Regression

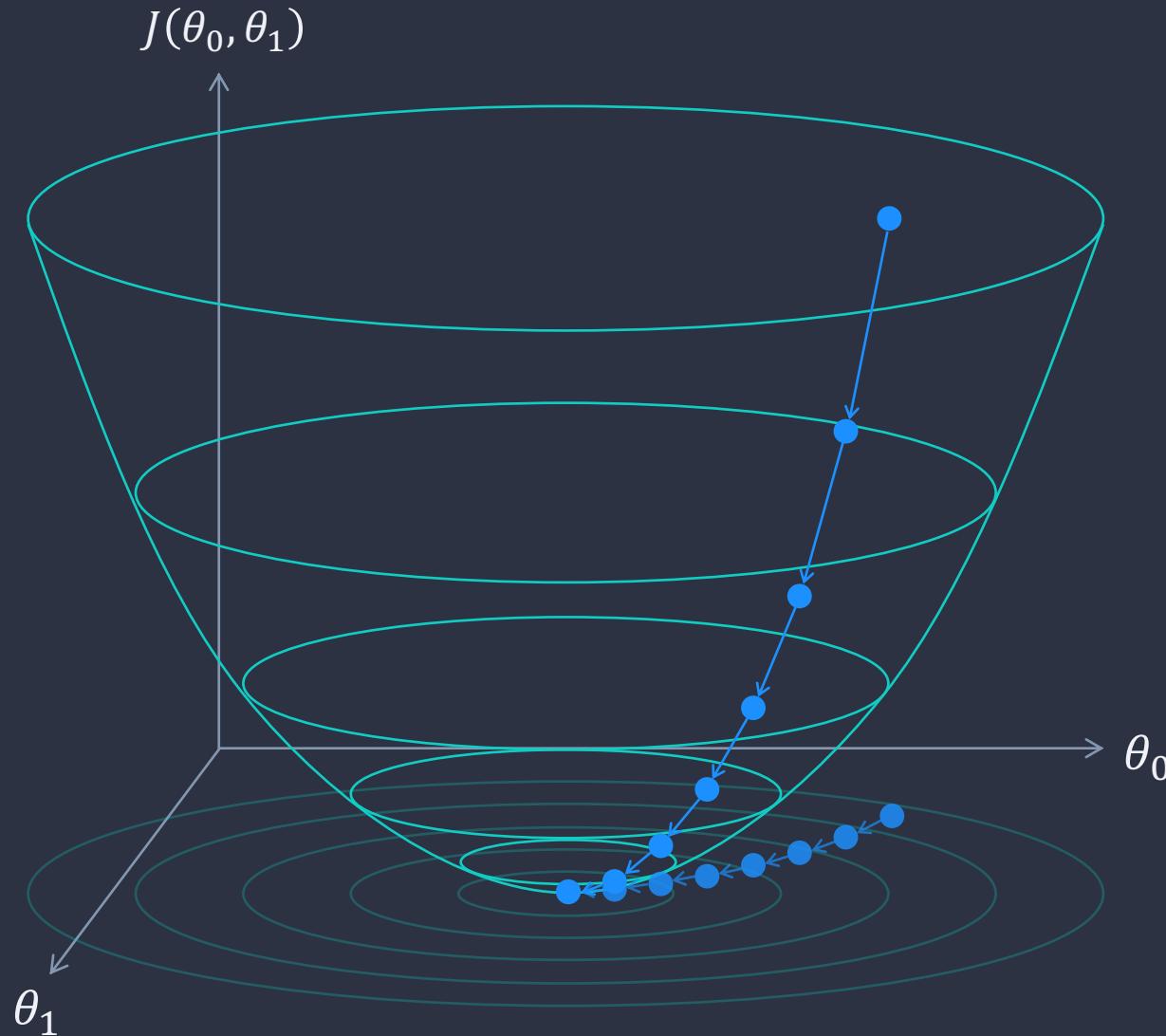
## Cost Function

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

# Normal Equation for Univariate Linear Regression

Cost Function

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



# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) = 0$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0$$

variable data in training set

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\begin{cases} \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) = 0 \\ \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0 \end{cases} \Rightarrow \begin{cases} \theta_0 = ? \\ \theta_1 = ? \end{cases}$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\begin{cases} \cancel{\frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)})} = 0 \\ \cancel{\frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)}} = 0 \end{cases} \Rightarrow \begin{cases} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) = 0 \\ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0 \end{cases}$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\begin{cases} \left[ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \right] = 0 \\ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0 \end{cases} \Rightarrow \begin{cases} \theta_0 m + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)} - \sum_{i=1}^m y^{(i)} = 0 \\ \theta_0 \sum_{i=1}^m \mathbf{x}^{(i)} + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)2} - \sum_{i=1}^m \mathbf{x}^{(i)} y^{(i)} = 0 \end{cases}$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\left[ \begin{array}{l} \left[ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \right] = 0 \\ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0 \end{array} \right] \Rightarrow \left[ \begin{array}{l} \theta_0 m + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)} - \sum_{i=1}^m y^{(i)} = 0 \\ \theta_0 \sum_{i=1}^m \mathbf{x}^{(i)} + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)2} - \sum_{i=1}^m \mathbf{x}^{(i)} y^{(i)} = 0 \end{array} \right]$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\left[ \begin{array}{l} \left[ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \right] = 0 \\ \sum_{i=1}^m (\theta_0 + \theta_1 \cdot \mathbf{x}^{(i)} - y^{(i)}) \cdot \mathbf{x}^{(i)} = 0 \end{array} \right] \Rightarrow \left[ \begin{array}{l} \theta_0 m + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)} - \left[ \sum_{i=1}^m y^{(i)} \right] = 0 \\ \theta_0 \sum_{i=1}^m \mathbf{x}^{(i)} + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)2} - \sum_{i=1}^m \mathbf{x}^{(i)} y^{(i)} = 0 \end{array} \right]$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\left\{ \begin{array}{l} \theta_0 m + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)} - \sum_{i=1}^m y^{(i)} = 0 \\ \theta_0 \sum_{i=1}^m \mathbf{x}^{(i)} + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)2} - \sum_{i=1}^m \mathbf{x}^{(i)} y^{(i)} = 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \theta_0 m + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)} - \sum_{i=1}^m y^{(i)} = 0 \\ \theta_0 \sum_{i=1}^m \mathbf{x}^{(i)} + \theta_1 \sum_{i=1}^m \mathbf{x}^{(i)2} - \sum_{i=1}^m \mathbf{x}^{(i)} y^{(i)} = 0 \end{array} \right.$$

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\begin{cases} \theta_0 m + \theta_1 \sum x^{(i)} - \boxed{\sum y^{(i)}} = 0 \\ \theta_0 \sum x^{(i)} + \theta_1 \sum x^{(i)}{}^2 - \boxed{\sum x^{(i)} y^{(i)}} = 0 \end{cases} \Rightarrow \begin{cases} \theta_0 m + \theta_1 \sum x^{(i)} = \boxed{\sum y^{(i)}} \\ \theta_0 \sum x^{(i)} + \theta_1 \sum x^{(i)}{}^2 = \boxed{\sum x^{(i)} y^{(i)}} \end{cases}$$

# Normal Equation for Univariate Linear Regression

# Cost Function

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

# Normal Equation for Univariate Linear Regression

## Cost Function

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

$$\begin{bmatrix} m & \Sigma x^{(i)} \\ \Sigma x^{(i)} & \Sigma x^{(i)2} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \Sigma y^{(i)} \\ \Sigma x^{(i)}y^{(i)} \end{bmatrix}$$

must be invertible  
(limitation of normal equation)

Cramer's Rule  $\Rightarrow$

$$\theta_0 = \frac{\begin{vmatrix} \Sigma y^{(i)} & \Sigma x^{(i)} \\ \Sigma x^{(i)}y^{(i)} & \Sigma x^{(i)2} \end{vmatrix}}{\begin{vmatrix} m & \Sigma x^{(i)} \\ \Sigma x^{(i)} & \Sigma x^{(i)2} \end{vmatrix}} = \frac{\Sigma y^{(i)}\Sigma x^{(i)2} - \Sigma x^{(i)}\Sigma x^{(i)}y^{(i)}}{m\Sigma x^{(i)2} - (\Sigma x^{(i)})^2}$$
$$\theta_1 = \frac{\begin{vmatrix} m & \Sigma y^{(i)} \\ \Sigma x^{(i)} & \Sigma x^{(i)}y^{(i)} \end{vmatrix}}{\begin{vmatrix} m & \Sigma x^{(i)} \\ \Sigma x^{(i)} & \Sigma x^{(i)2} \end{vmatrix}} = \frac{m\Sigma x^{(i)}y^{(i)} - \Sigma y^{(i)}\Sigma x^{(i)}}{m\Sigma x^{(i)2} - (\Sigma x^{(i)})^2}$$

Different from gradient descent – we don't try thetas in iterations, but we calculate their values directly.

# Normal Equation for Multivariate Linear Regression

## Cost Function

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

As the cost function is quadratic, we can always find  $\theta_0, \theta_1, \dots, \theta_n$  that make partial derivatives to be zero.

# Normal Equation for Multivariate Linear Regression

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



A graph of a convex function  $J(\theta)$  plotted against  $\theta$ . The curve is U-shaped, representing the cost function. The minimum of the function is at the origin (0,0), which corresponds to the values of the parameters being optimized.

$$\frac{\partial J}{\partial \theta_0}(\theta) = 0$$
$$\frac{\partial J}{\partial \theta_1}(\theta) = 0$$

...

$$\frac{\partial J}{\partial \theta_n}(\theta) = 0$$

Task:

1. Take partial derivatives of  $J$  with respect to every parameter and then set all of them to be zero.
2. Solve for the values of parameters, to get their values that minimize the cost function  $J$ .

How do we do this using  
normal equation approach?

# Normal Equation for Multivariate Linear Regression

Vectorised partial derivative

$$\frac{\partial J}{\partial \theta} = X^T (X\theta - y) = 0$$

# Normal Equation for Multivariate Linear Regression

$$\frac{\partial J}{\partial \theta} = X^T (X\theta - y) = 0$$

$$X^T X\theta - X^T y = 0$$

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

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

# Normal Equation for Multivariate Linear Regression

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

feature values      labels

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

# Normal Equation      vs      Gradient Descent

# Normal Equation

vs

# Gradient Descent

- Non-iterative
  - No learning rate
  - Doesn't need feature scaling
  - Slow if training set is very large  
 $(\mathbf{X}^T \mathbf{X})^{-1}$   $O(n^3)$
  - Doesn't work for many cases
- Multiple iterations
  - Experimental learning rate
  - Needs feature scaling
  - Works well with very large training set
  - Works with other types of tasks too

# Summary

# Summary

## Hypothesis function

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = [\theta_0, \theta_1, \theta_2, \dots, \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

## Cost function

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

# Summary

## Gradient descent

Repeat until convergence {

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

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)}) x_1^{(i)}$$

...

$$\theta_n := \theta_n - \alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x})^{(i)} - y^{(i)}) x_n^{(i)}$$

}

# Summary

## Vectorisation of Hypothesis Function

$$h_{\theta}(x) = \theta^T X \quad h_{\theta}(x) = X \theta$$

## Vectorisation of Cost Function

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

## Vectorisation of Gradient Descent

Repeat until convergence {

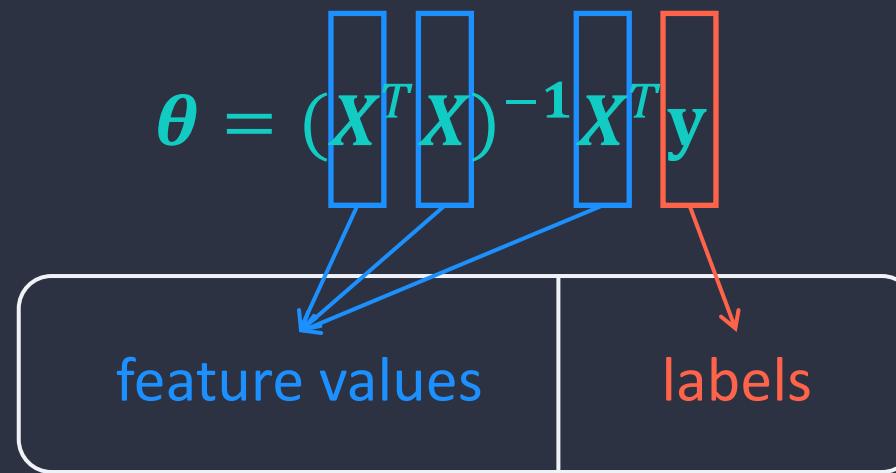
$$\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - y)$$

}

# Summary

## Normal equation

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



Next Lecture

Polynomial Linear Regression

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Polynomial Linear Regression

Dr Yang Long

# Last Lecture



## Multivariate Linear Regression

1. Multiple Variables
2. Cost Function and Gradient Descent
3. Vectorisation
4. Normal Equation

# Lecture Overview

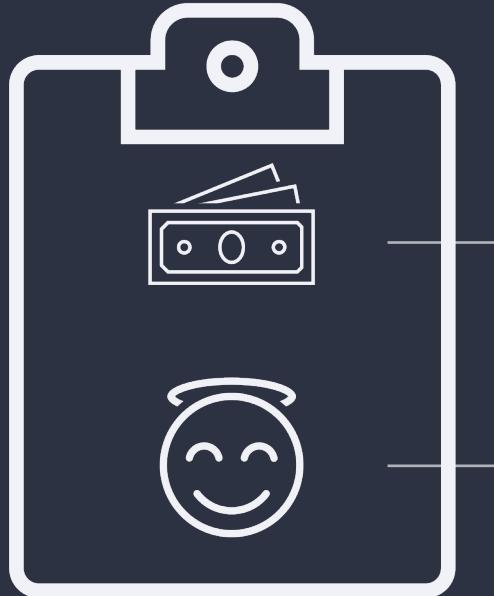
1. Polynomial Linear Regression
2. Underfitting versus Overfitting
3. Bias versus Variance

# 1. Polynomial Linear Regression

EXAMPLE. Annual income to predict happiness



500 people in Durham



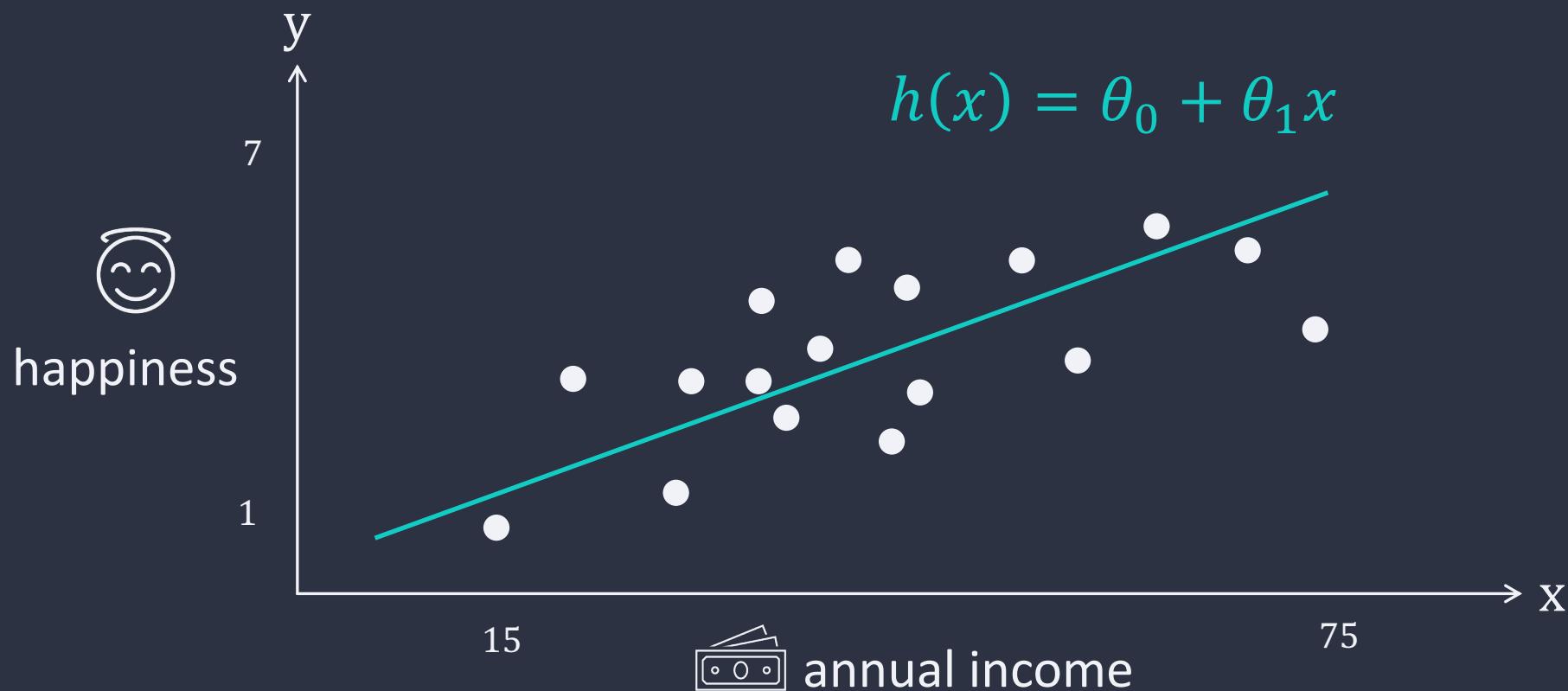
survey

annual income  
£15k -- £75k

happiness  
0 -- 10

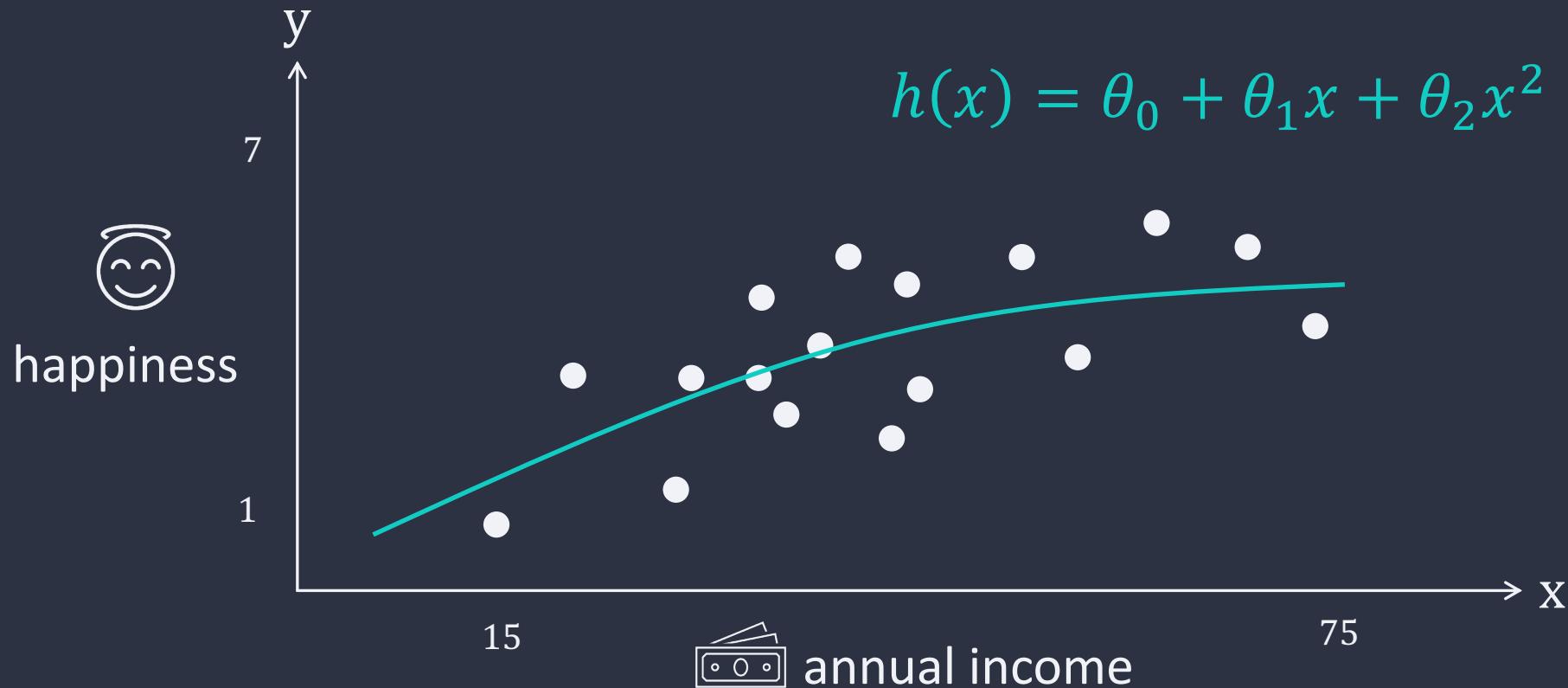
## EXAMPLE. Annual income to predict happiness

Simple linear regression: a linear regression model with a single explanatory variable ( $x$ ).



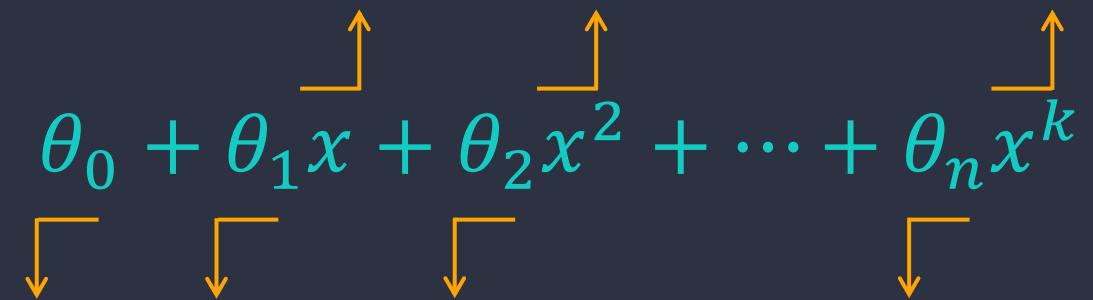
EXAMPLE. Annual income to predict happiness

Polynomial linear regression: a linear regression model with higher order.



# Polynomial

data  $x$ , taken to increasingly high power

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^k$$


coefficients, scaling data

# Polynomial

$$\theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_k x^k$$
$$\theta_0 x^0 \quad 2 + 4x + \frac{3}{7}x^2 + 5.6x^3$$

# Polynomial

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

$$2 + \boxed{4}x + \frac{3}{7}x^2 + 5.6x^3$$

# Polynomial

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

$$2 + 4x + \boxed{\frac{3}{7}}x^2 + 5.6x^3$$

# Polynomial

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

$$2 + 4x + \frac{3}{7}x^2 + \boxed{5.6x^3}$$

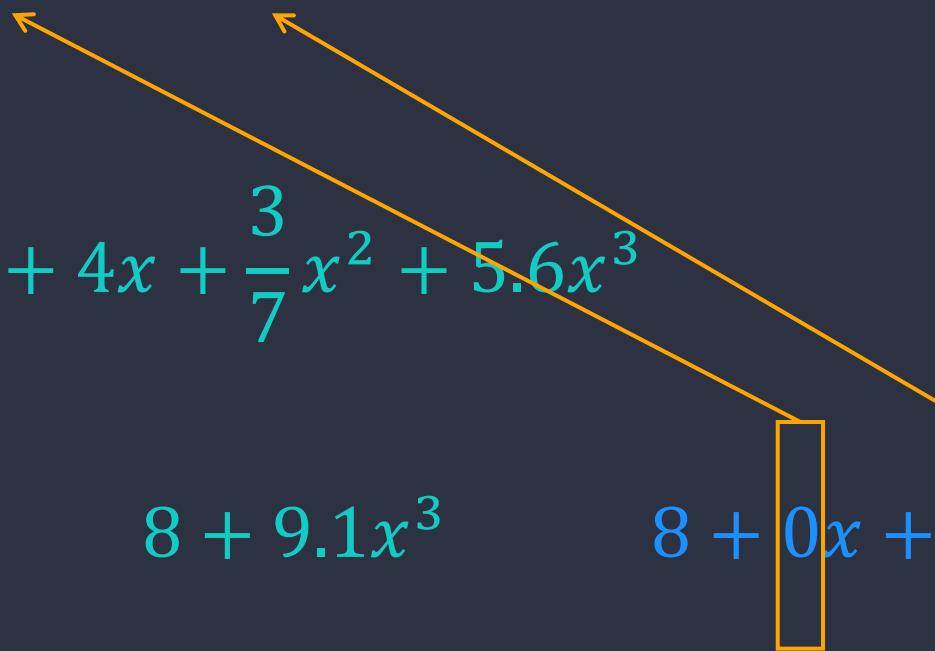
# Polynomial

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

$$2 + 4x + \frac{3}{7}x^2 + 5.6x^3$$

$$8 + 9.1x^3$$

$$8 + \boxed{0}x + \boxed{0}x^2 + 9.1x^3$$



# Polynomial

Order: the highest coefficient

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^{\boxed{k}}$$

$$2 + 4x + \frac{3}{7}x^2 + 5.6x^3 \quad \longleftrightarrow \text{third order polynomial}$$

$$8 + 9.1x^3$$

# Polynomial Regression

$$h(x) = \theta_0 + \theta_1 x \quad \text{first order polynomial}$$

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \quad \text{second order polynomial}$$

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k \quad \text{k-th order polynomial}$$

# Fit a Polynomial Regression with Nonlinearities

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

nonlinear terms

# Fit a Polynomial Regression with Nonlinearities

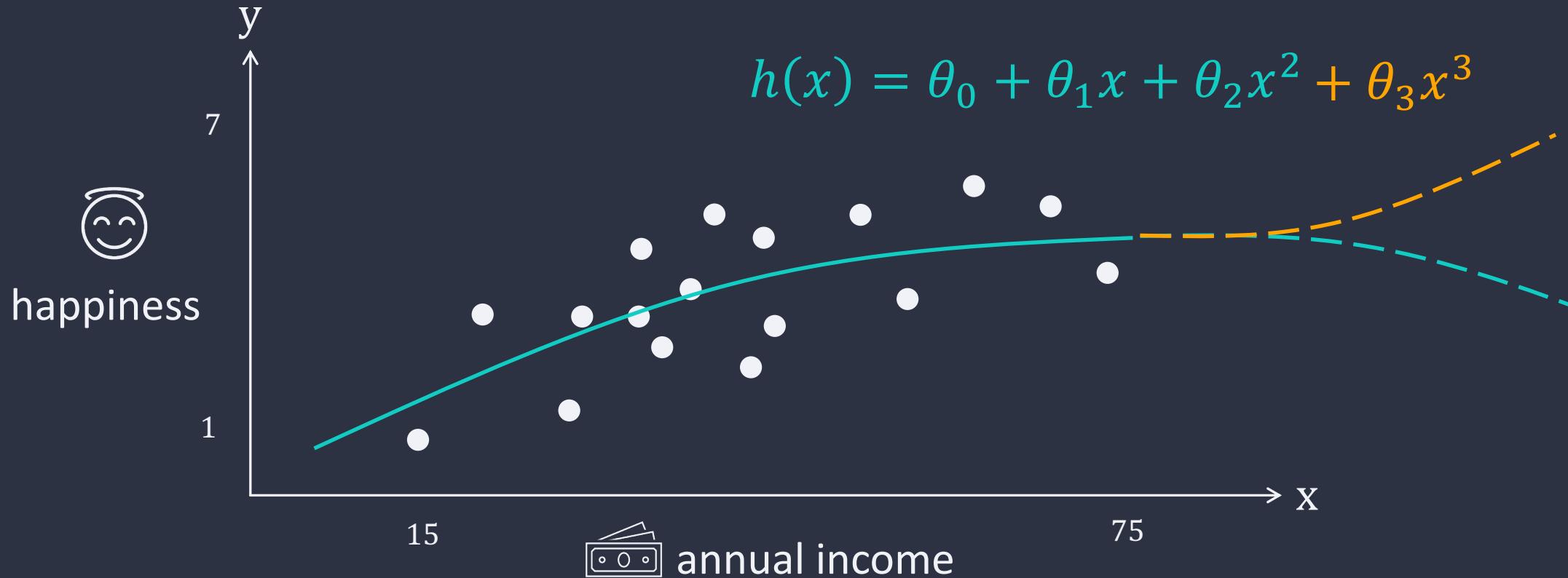
$$h(x) = \boxed{\theta_0} + \boxed{\theta_1}x + \boxed{\theta_2}x^2 + \dots + \boxed{\theta_k}x^k$$

The coefficients are all linear.



This is a standard linear model!

EXAMPLE. Annual income to predict happiness



EXAMPLE. Annual income to predict happiness

Transforming polynomial linear regression to multivariate linear regression.

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



$$h(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Gradient descent / normal equation to get  $\theta_0, \theta_1, \theta_2, \theta_3$

EXAMPLE. Annual income to predict happiness

Transforming polynomial linear regression to multivariate linear regression.

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$x_1 = x \quad 10 \dots 1000$$



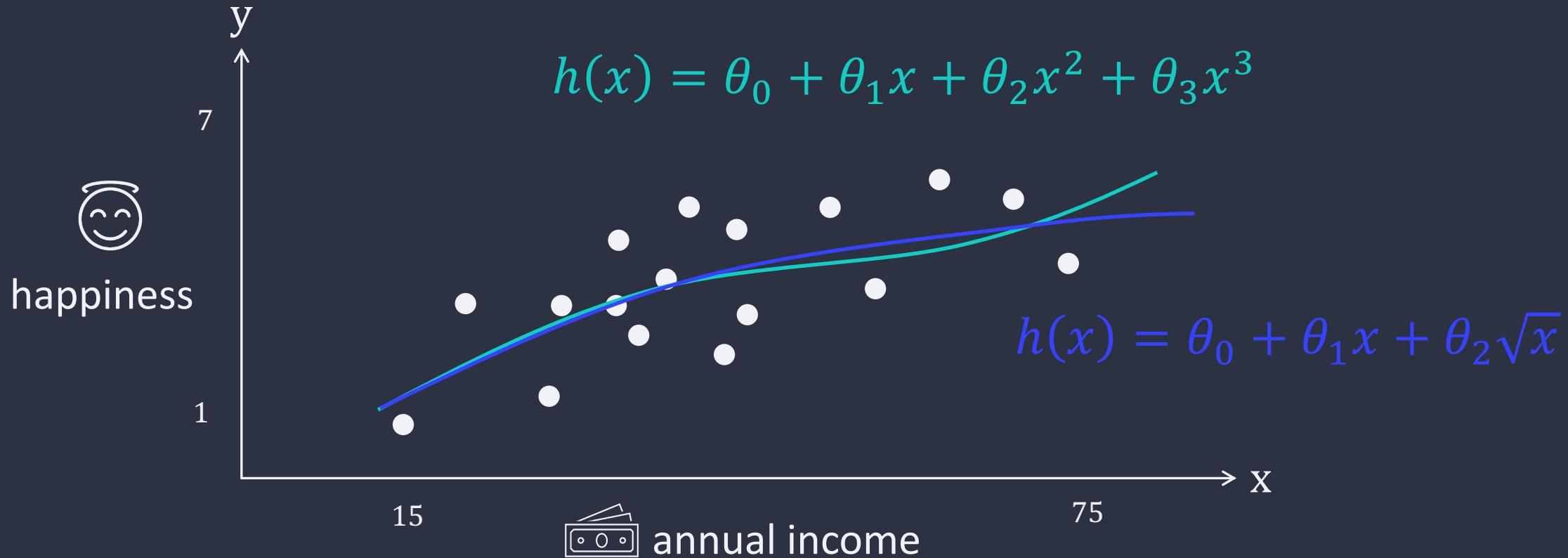
$$x_2 = x^2 \quad 100 \dots 1,000,000$$

$$h(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$x_3 = x^3 \quad 1000 \dots 1,000,000,000$$

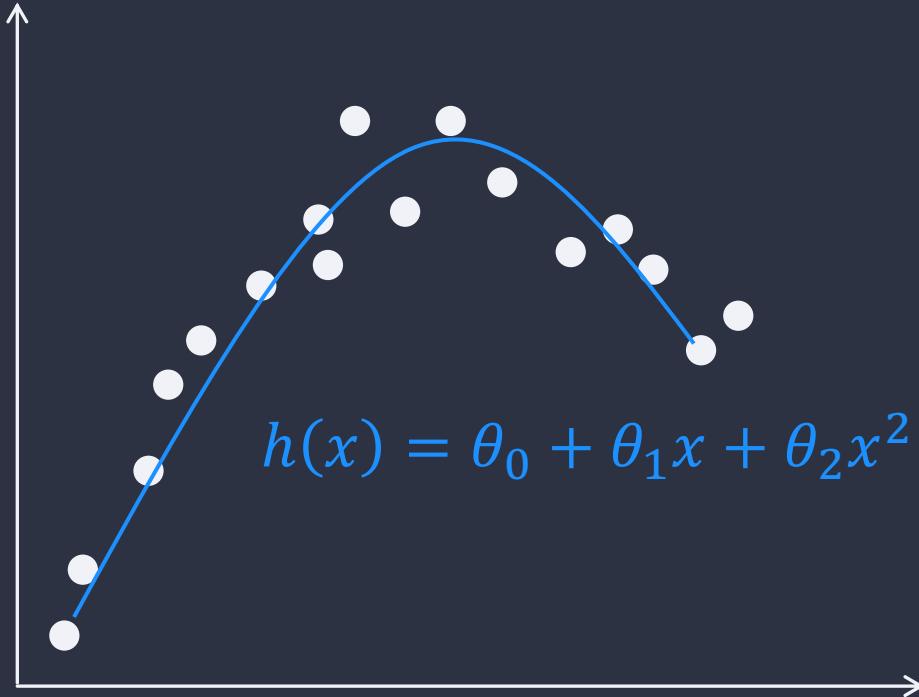
Feature scaling is important!

EXAMPLE. Annual income to predict happiness

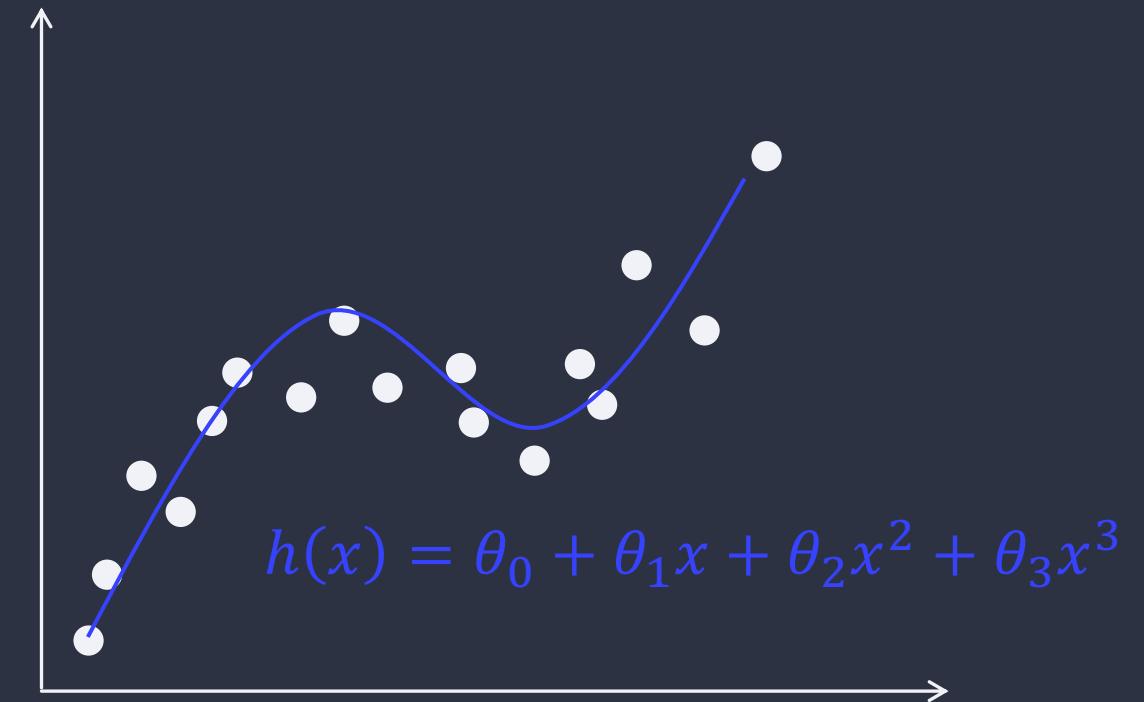


# When to use Polynomial

Second-order polynomial



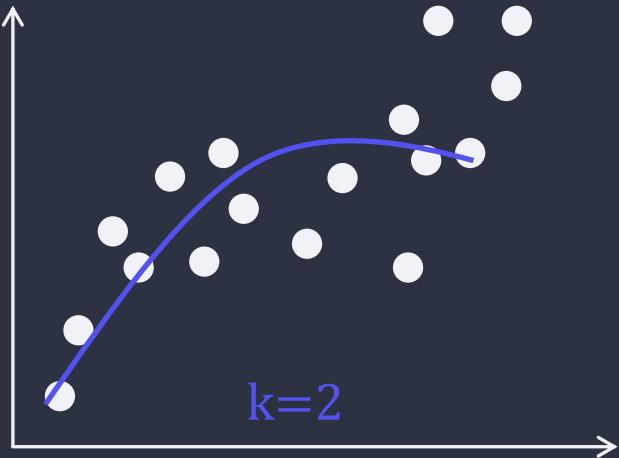
Third-order polynomial



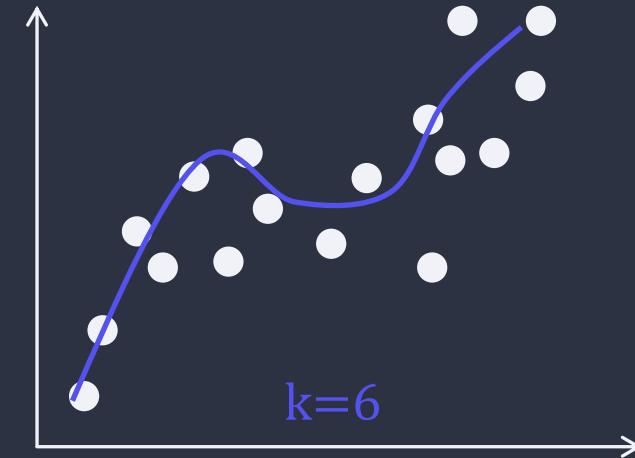
How do we decide the order of a polynomial?

$$h(x) = \theta_0 + \theta_1x + \theta_2x^2 + \cdots + \theta_kx^k$$

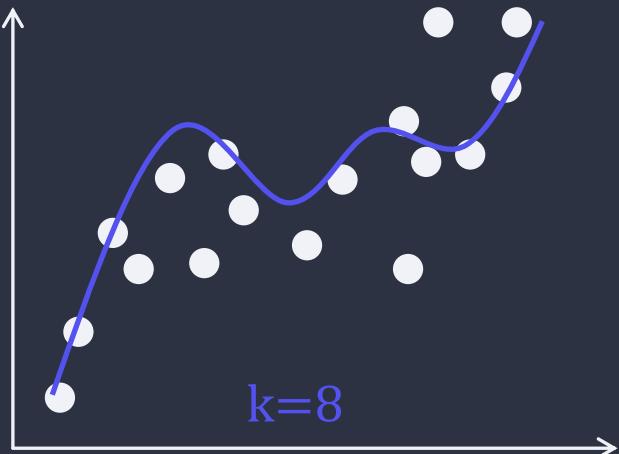
# Order of Polynomial



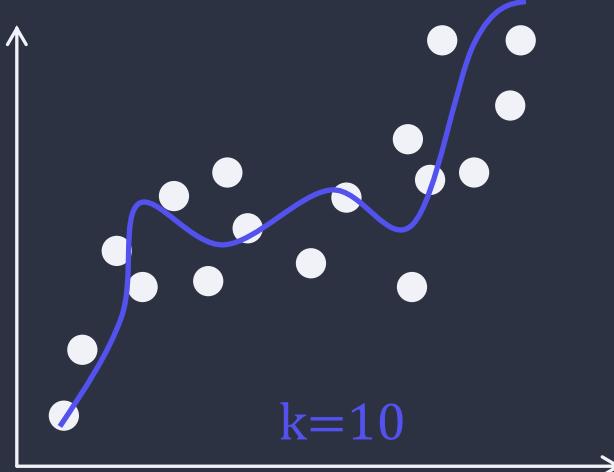
$k=4$



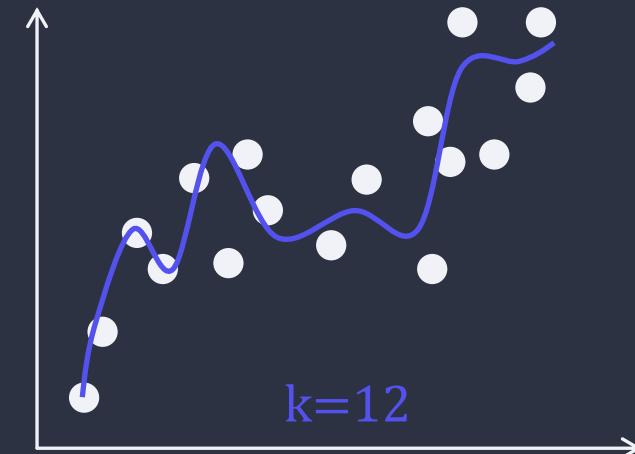
$k=6$



$k=8$



$k=10$



$k=12$

# Deciding on Order of Polynomial

## Bayesian Information Criterion

$$BIC_k = m \cdot \ln(SS_{\epsilon}) + k \cdot \ln(m)$$

$k$  : the order of the polynomial regression model

$m$  : the number of examples in the training set

$\ln$  : natural log

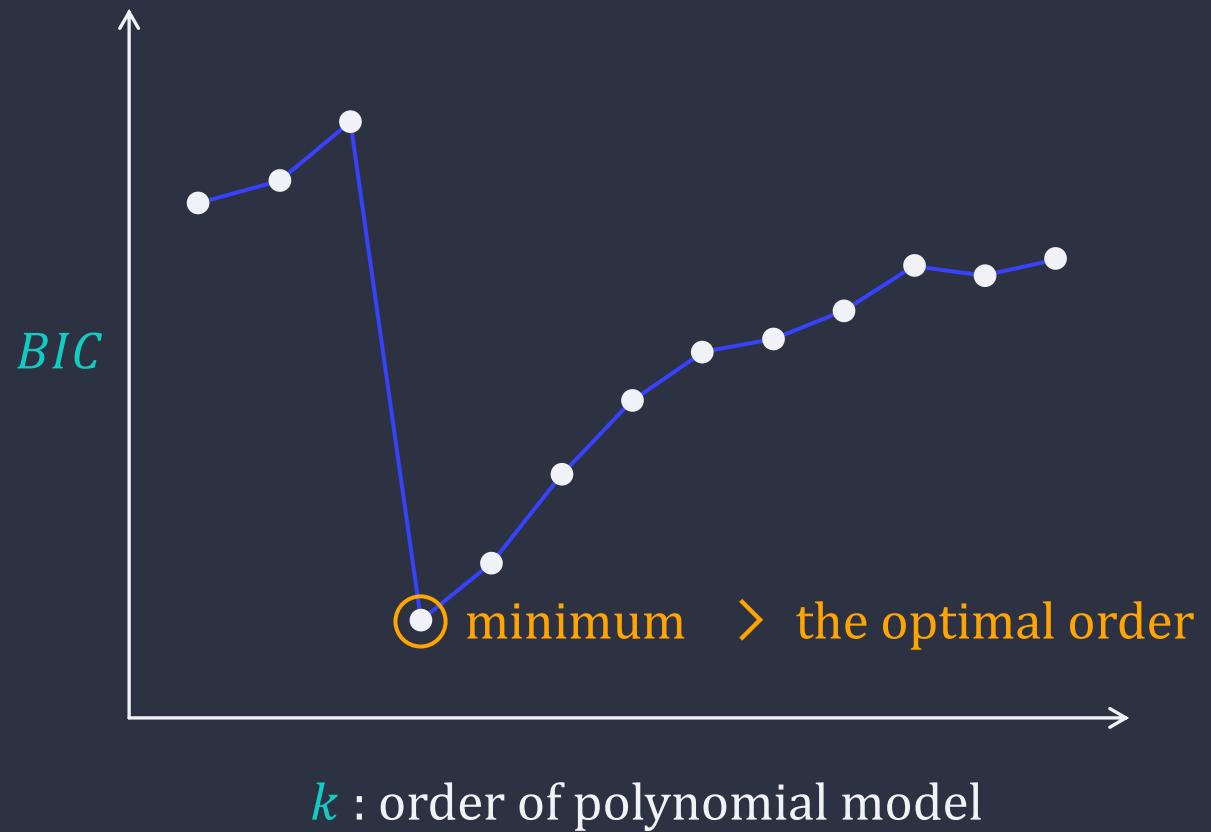
$SS_{\epsilon}$  : sum of the squares of the errors

- Run a series of different model orders to fit training data, to get different BIC values for those models.

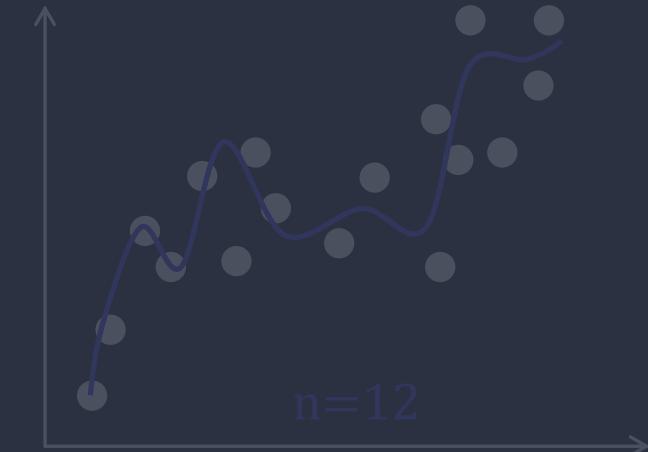
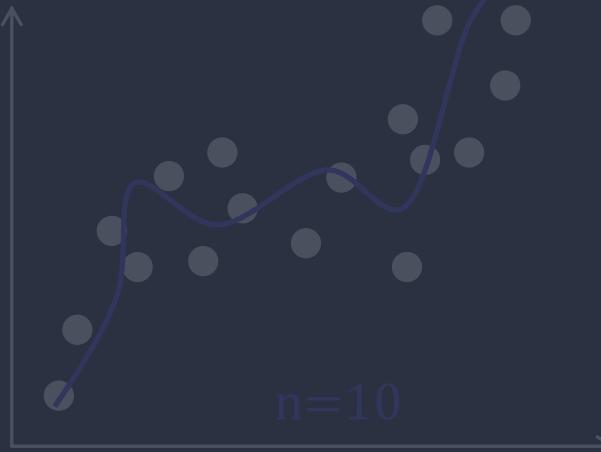
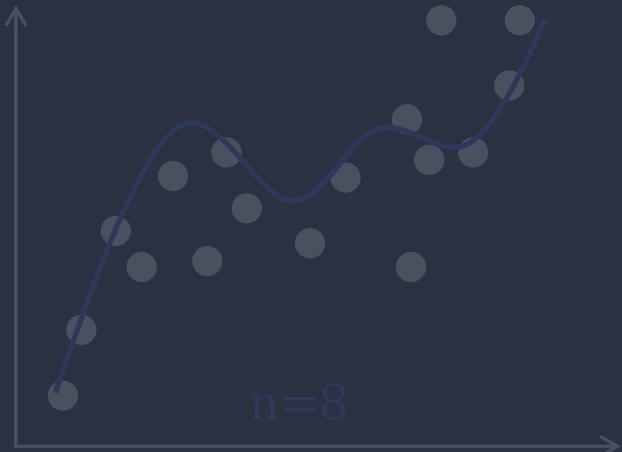
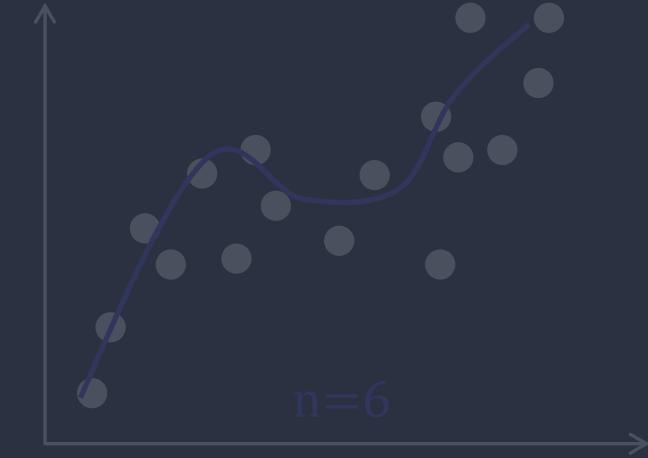
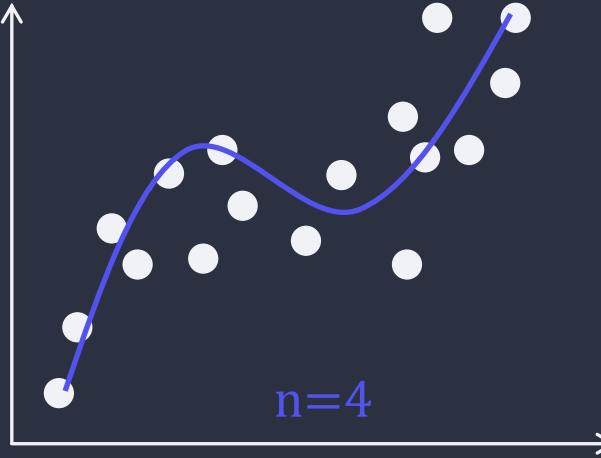
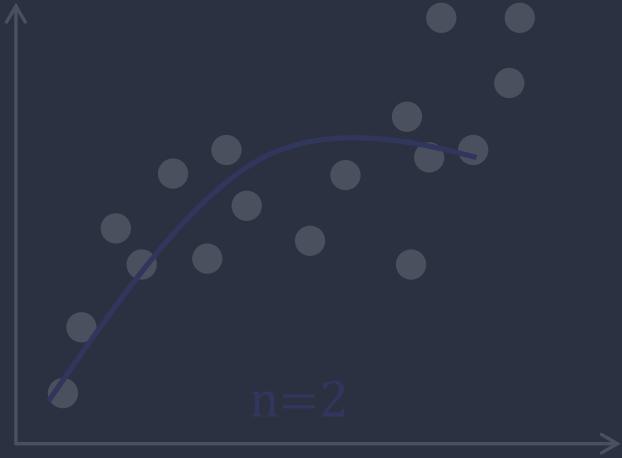
# Deciding on Order of Polynomial

Bayesian Information Criterion

$$BIC_k = m \cdot \ln(SS_{\epsilon}) + k \cdot \ln(m)$$



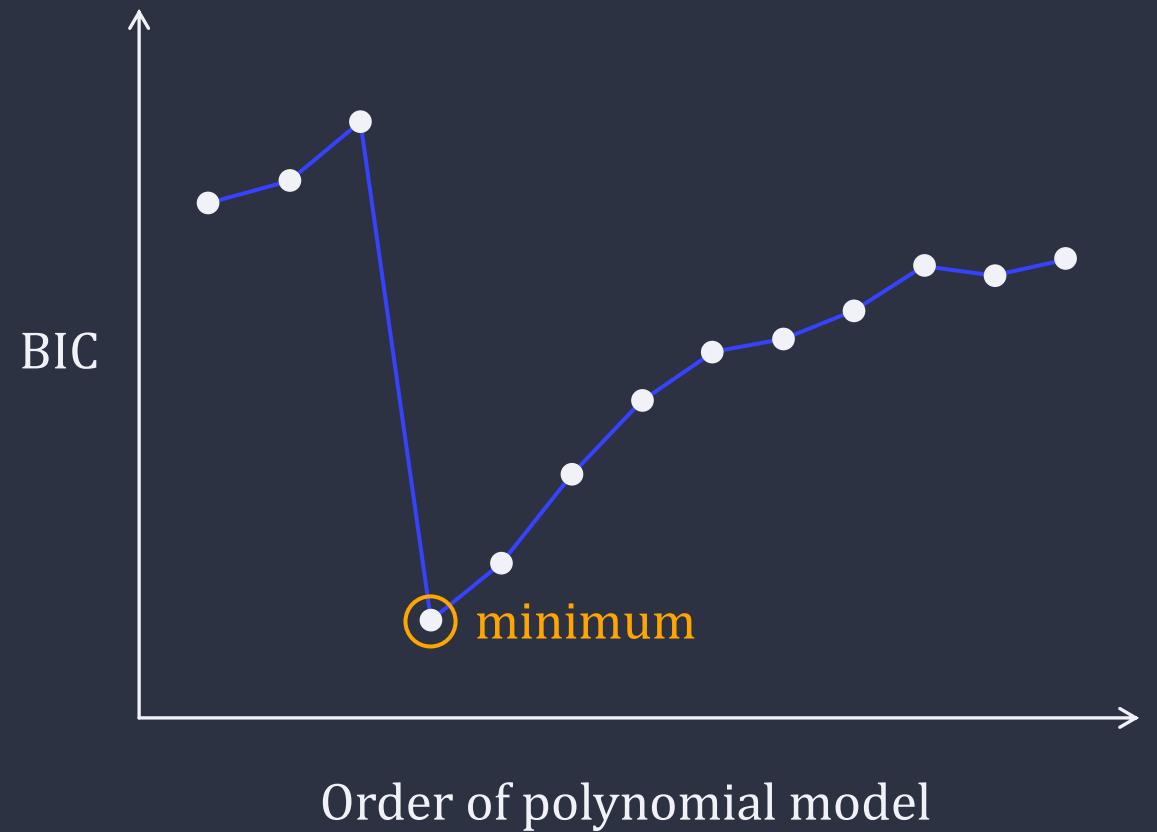
# Deciding on Order of Polynomial



# Deciding on Order of Polynomial

Bayesian Information Criterion

$$BIC_k = m \cdot \ln(SS_{\epsilon}) + k \cdot \ln(m)$$



# Coding with sci-kit learn

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2$$

[0,1]

$X$  : training data (2 features) [2,3] →  $X_{trans}$  : transformed training data (5 features)  
[4,5]

```
class sklearn.preprocessing.PolynomialFeatures(  
    degree=2, *, interaction_only=False, include_bias=True, order='C')
```

```
from sklearn.preprocessing import PolynomialFeatures  
  
poly = PolynomialFeatures(2)  
X_trans = poly.fit_transform(X)
```

$[a, b] \rightarrow [1, a, b, a^2, ab, b^2]$	[0,1] → [1,0,1,0,0,1] [2,3] → [1,2,3,4,6,9] [4,5] → [1,4,5,16,20,25]
--	--

# Coding with sci-kit learn

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2$$

[0,1]

$X$  : training data (2 features) [2,3] →  $X_{trans}$  : transformed training data (5 features)

[4,5]

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
print(X)

poly = PolynomialFeatures(2)
X_trans = poly.fit_transform(X)

print(X_trans)

poly = PolynomialFeatures(interaction_only=True)
X_trans2 = poly.fit_transform(X)

print(X_trans2)
```

```
[[0 1]
 [2 3]
 [4 5]]
[[ 1.  0.  1.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]]
[[ 1.  0.  1.  0.]
 [ 1.  2.  3.  6.]
 [ 1.  4.  5. 20.]]
```

# Coding with sci-kit learn

```
# fit Polynomial Regression to dataset
from sklearn.preprocessing import PolynomialFeatures

trans = PolynomialFeatures(degree=2) # try different degrees e.g. 2, 3, 4, 5, ...
X_poly = trans.fit_transform(X)

trans.fit(X_poly, y)
lin_regr_poly = LinearRegression()
lin_regr_poly.fit(X_poly, y)

# predict all data points
lin_regr_pred_poly = lin_regr_poly.predict(trans.fit_transform(X))
```

[https://github.com/lshi/ml/blob/main/Polynomial\\_Regression.ipynb](https://github.com/lshi/ml/blob/main/Polynomial_Regression.ipynb)

## 2. Underfitting versus Overfitting

## EXAMPLE.

Annual income to predict happiness

$$h(x) = \theta_0 + \theta_1 x$$

**Underfitting**

Cannot catch much of details from data

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

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

**Overfitting**

Catches too much of noise

## Underfitting

Occurs when a learning algorithm can't capture underlying trend of the data. The model doesn't fit data well enough. This often happens if our model is excessively simple, e.g. polynomial order is too small.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)})^2 \gg 0$$

## Overfitting

Occurs when a learning algorithm captures too much of noise from the data. The model fits the data too well. This often happens if the model is excessively complicated, e.g. polynomial order is too high.

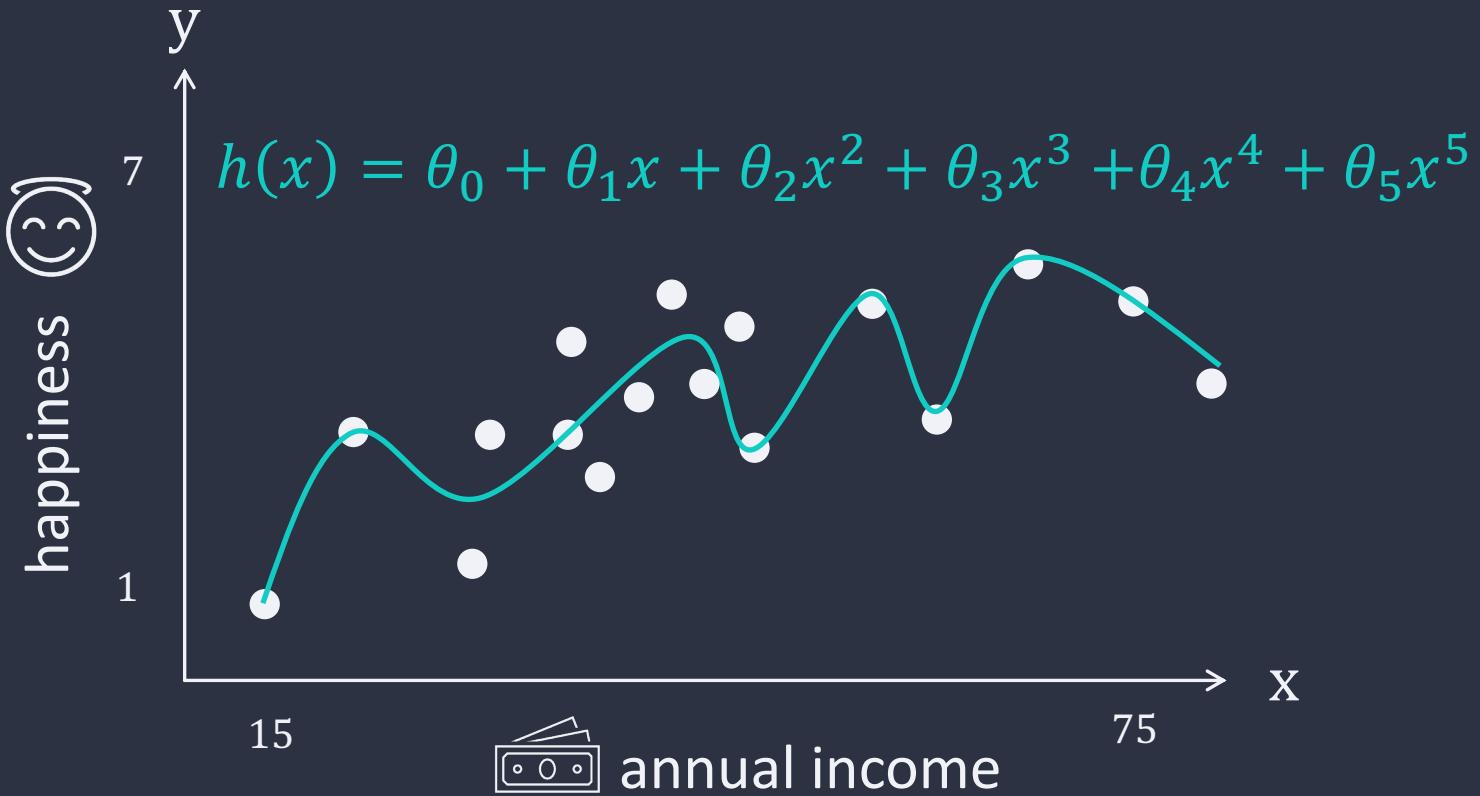
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)})^2 \approx 0$$

Both lead to poor predictions on new data.

# Tackling Overfitting

# Tackling Overfitting

- plotting and observing



$x_1$ : annual income

$x_2$ : age

$x_3$ : number of children

$x_4$ : cups of tea per week

...

$x_{50}$

With too many features, we are unable to plot them in graph

Too many features but not enough training data -> overfitting would become a big problem!

# Tackling Overfitting

## Include more data

- Collect more data
- Data augmentation

## Cross-Validation

- K-fold cross validation
- Leave-one-out cross validation

## Feature selection / reduction

- Manually
- Feature selection algorithms

## Regularisation

- Keep all the features
- Regularise parameters

# Tackling Underfitting

## Include more features

- Increase complexity
- Relevant & decisive features

## Increase model complexity

- Higher order polynomial
- Linear to non-linear

## Reduce regularisation

- Reduce penalty
- Reduce regularisation values

## Increase training time

- Keep all the features
- Regularise parameters

# 3. Bias versus Variance

EXAMPLE.

Annual income to predict happiness

$$h(x) = \theta_0 + \theta_1 x$$

Underfitting / High Bias

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

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

Overfitting / High Variance

Low Bias

Low Variance



High Variance



High Bias

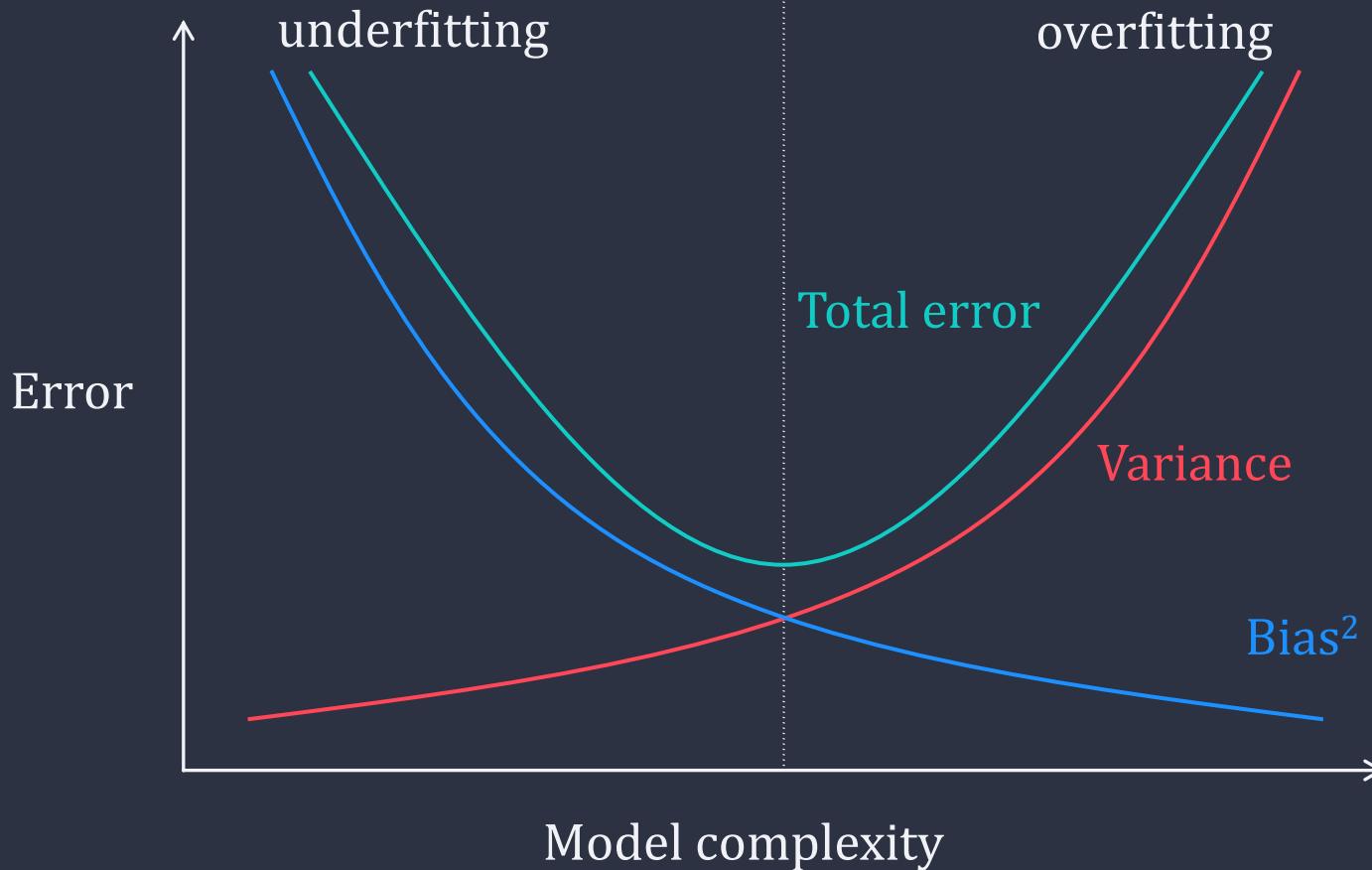


# Bias-Variance Trade-off

Optimal model complexity

$$J_{train}(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X^{(i)}) - y^{(i)})^2$$

$$J_{test}(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X^{(i)}) - y^{(i)})^2 \quad \text{or } J_{cv}(\boldsymbol{\theta})$$



High bias / underfitting

$J_{train}(\boldsymbol{\theta})$  high     $J_{train}(\boldsymbol{\theta}) \approx J_{test}(\boldsymbol{\theta})$

High variance / overfitting

$J_{train}(\boldsymbol{\theta})$  low     $J_{train}(\boldsymbol{\theta}) \ll J_{test}(\boldsymbol{\theta})$

# Summary

# Summary

## Polynomial regression

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$



$$h(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Underfitting versus Overfitting

Bias versus Variance

Next Lecture

Cost Functions for Regression

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Cost Functions for Regression

Dr Yang Long

# Previously

- Univariate Regression
- Multivariate Regression
- Polynomial Regression



Cost Function

$$\text{MSE } J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

# Lecture Overview

1. Mean Squared Error (MSE) Cost Function
2. Mean Absolute Error (MAE) Cost Function
3. Huber-M Cost Function
4. Log-Cosh Cost Function
5. Quantile Cost Function

# Supervised Learning

- To build a model represented as a hypothesis function  $h_{\theta}(x)$ .



# Supervised Learning

input X (independent variable)



model (hypothesis function, mapping  $X \rightarrow y$ )

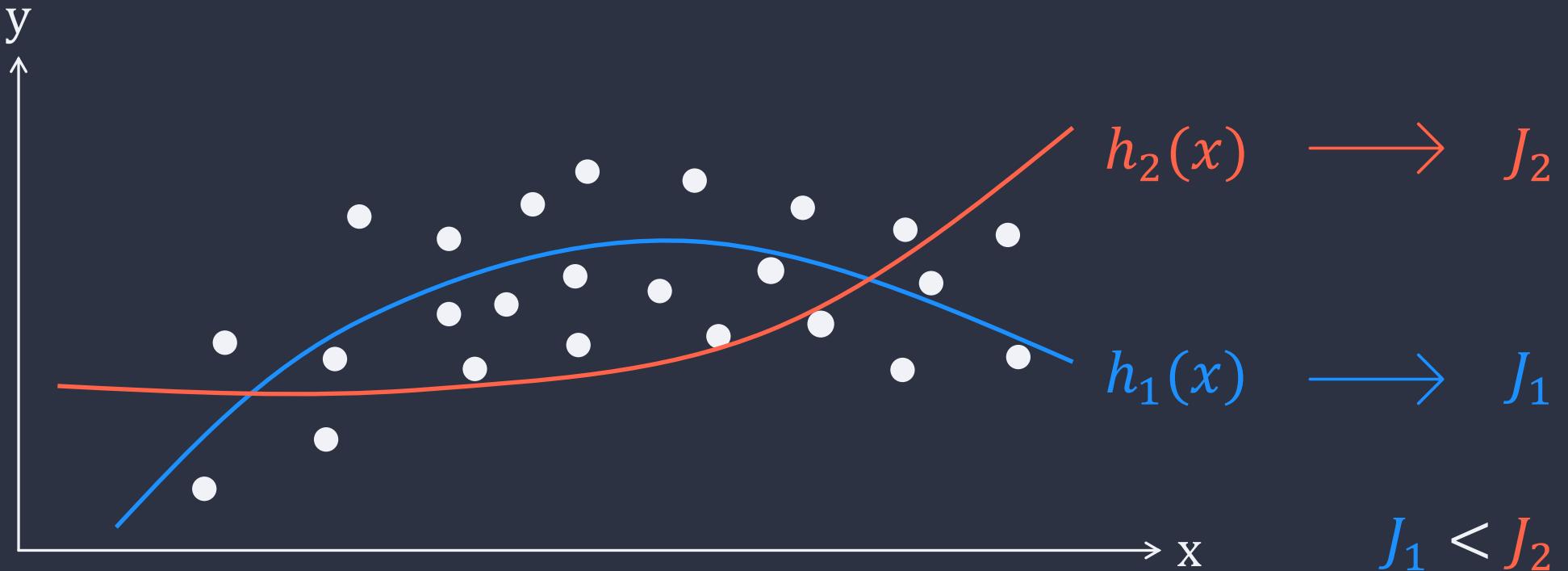


Cost Functions

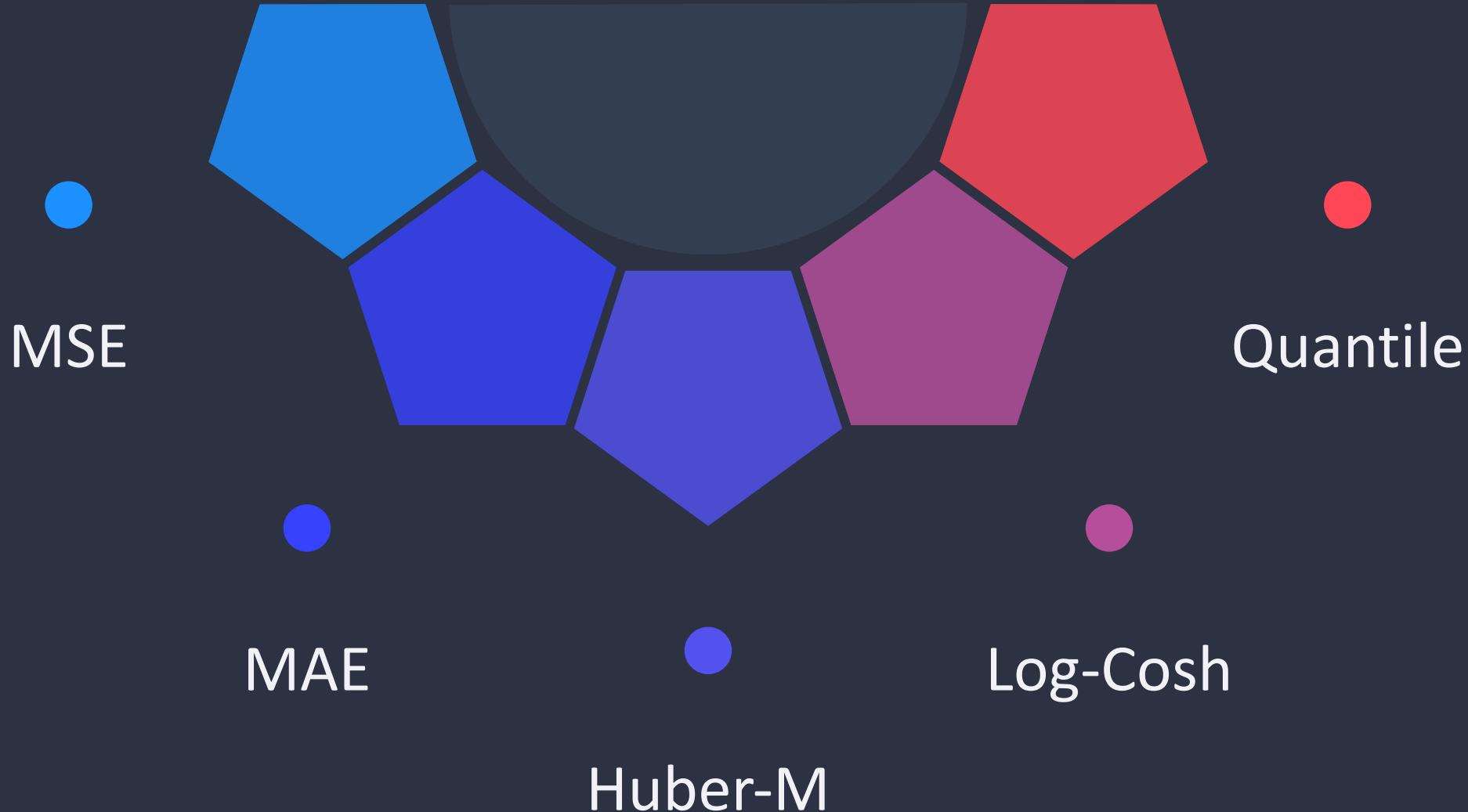


output y (dependent variable)

# Cost Functions for Regression



- The smaller values of the cost function, the better the model fits the data.
  - The goal of ML is, in general, to create a hypothesis function such that  $J$  is minimised.
- Cost function to compare predicted values and actual values, using specific measure of "goodness of fit".



# 1. Mean Squared Error (MSE) Cost Function

# Mean Squared Error (MSE) Cost Function

Also called Mean Squared Deviation (MSD) Cost Function

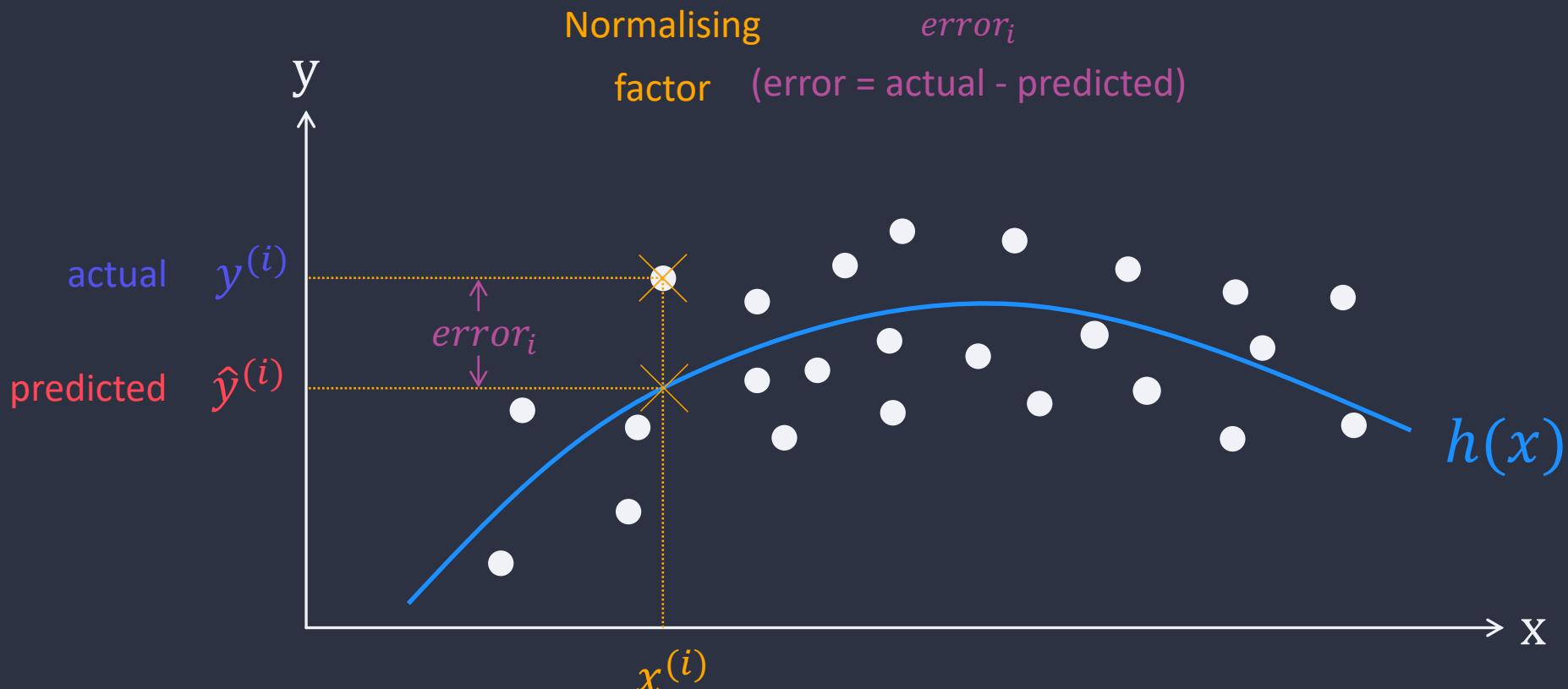
The most commonly used cost function for regression tasks.

$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

actual      predicted

# Mean Squared Error (MSE) Cost Function

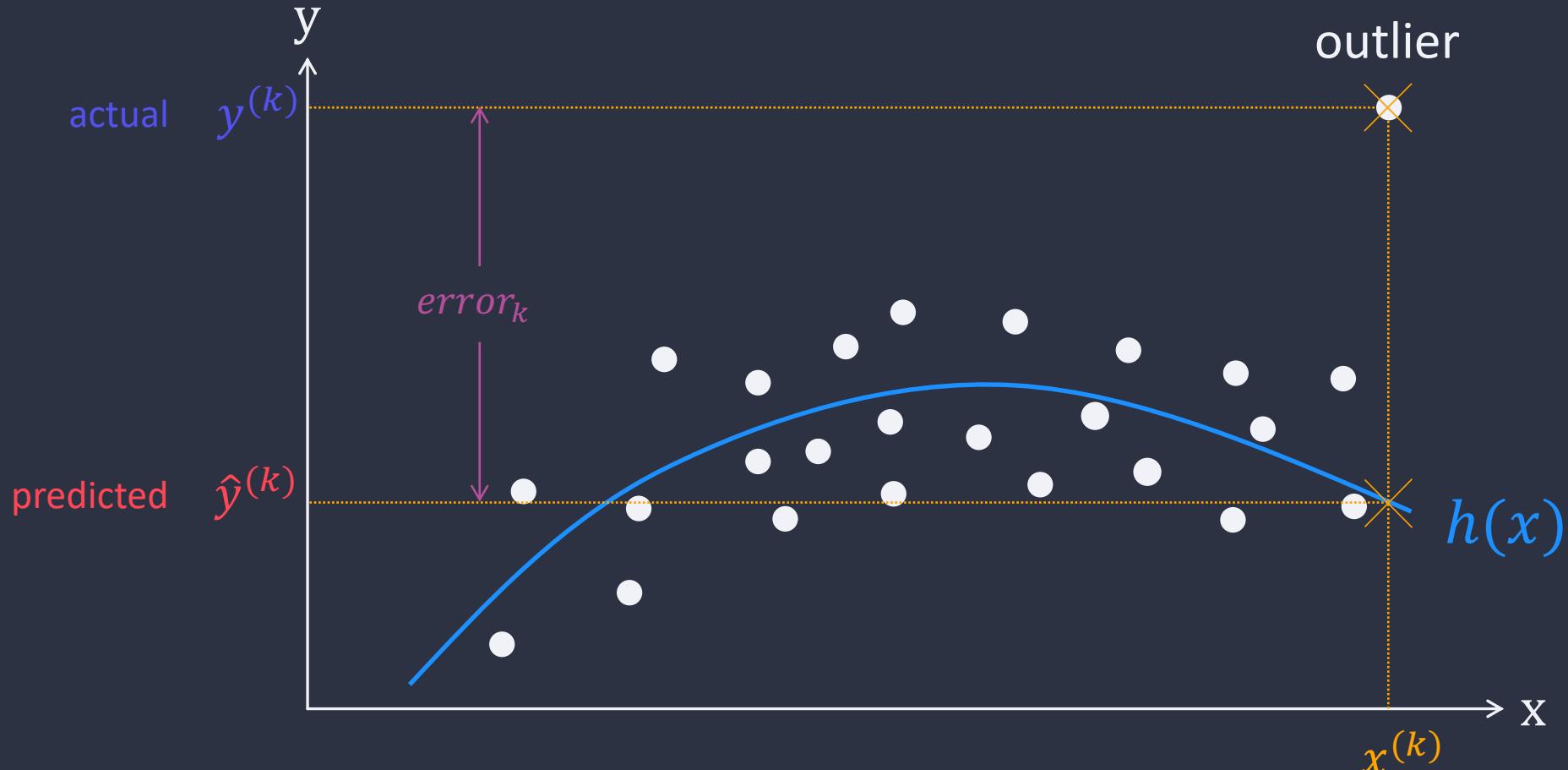
$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$



# Mean Squared Error (MSE) Cost Function

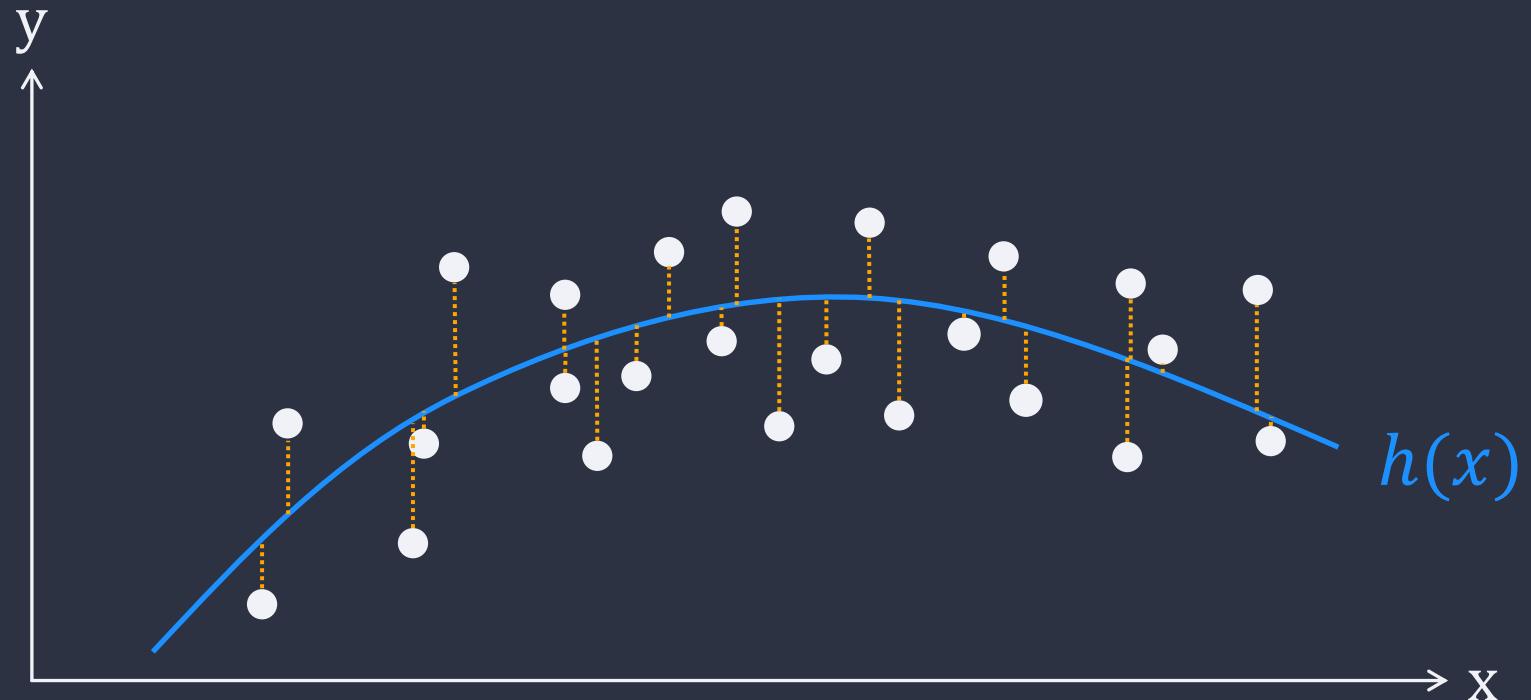
$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

Outlier dominates cost function



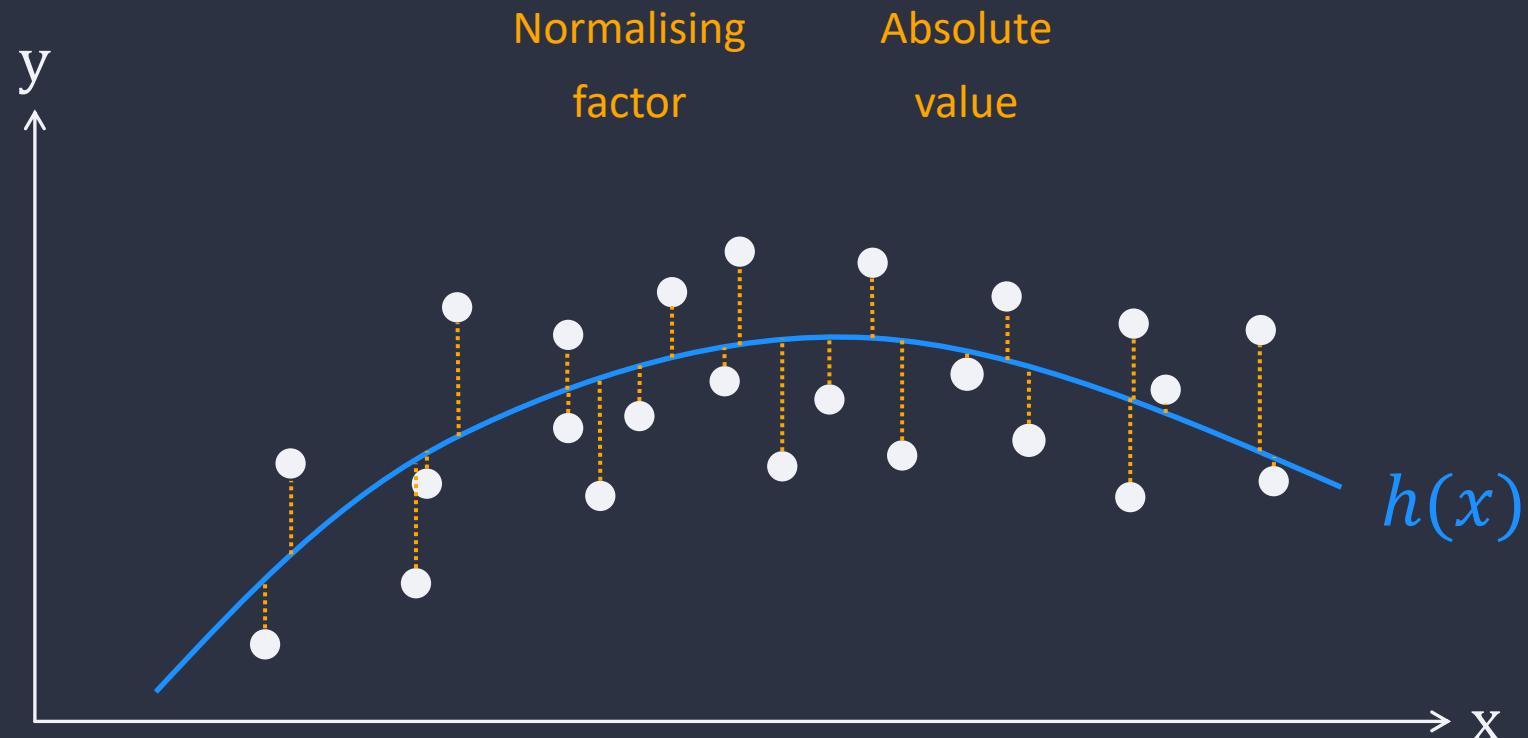
## 2. Mean Absolute Error (MAE) Cost Function

# Mean Absolute Error (MAE) Cost Function



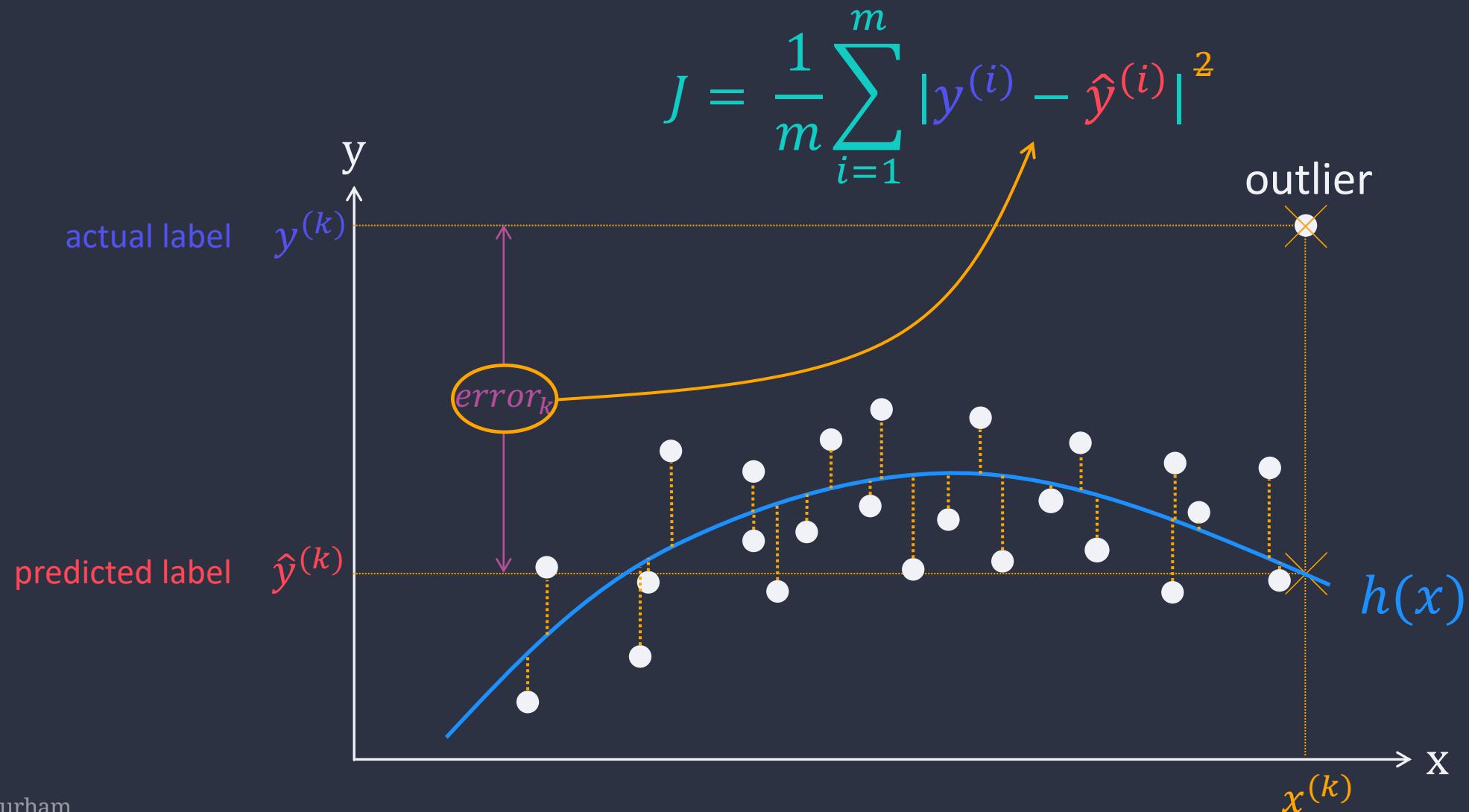
# Mean Absolute Error (MAE) Cost Function

$$J = \left[ \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}| \right]$$



# Mean Absolute Error (MAE) Cost Function

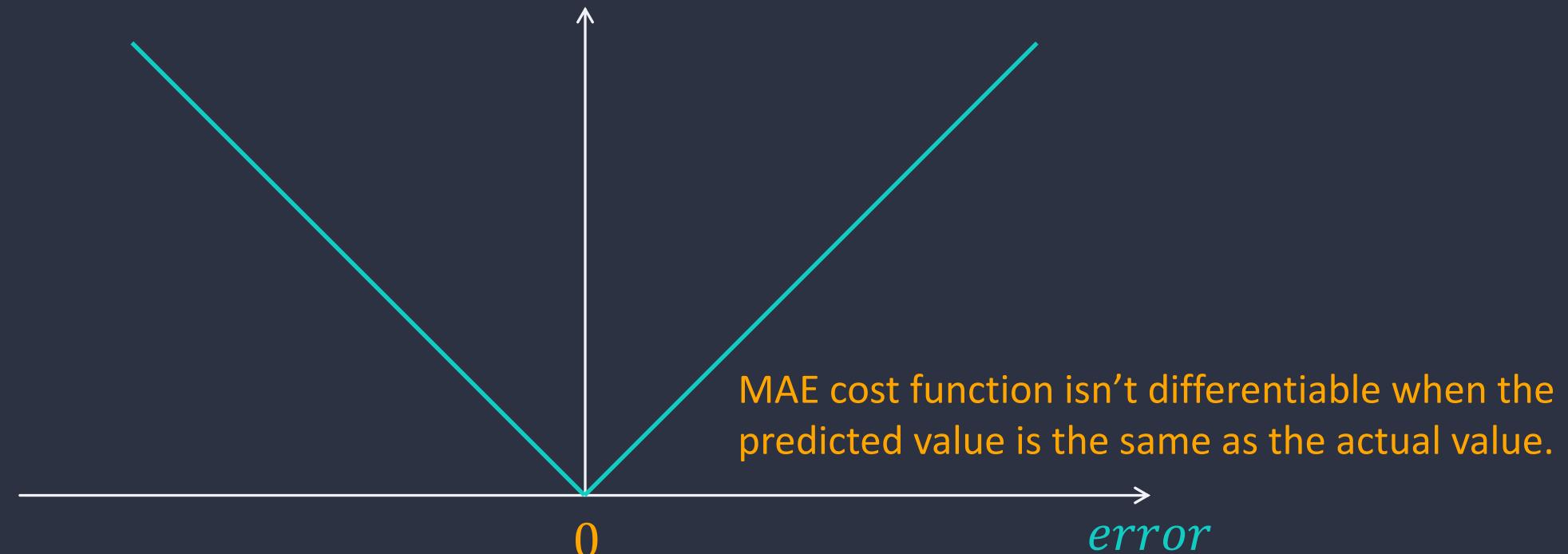
- More robust with respect to outliers.



# Mean Absolute Error (MAE) Cost Function

- More robust with respect to outliers.
- Poses computational challenges.

$$J = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}| \quad \frac{d}{dr} = \begin{cases} -1, & \text{error} < 0 \\ +1, & \text{error} > 0 \end{cases} \quad (\text{error} = y - \hat{y})$$



# MSE Cost Function vs MAE Cost Function

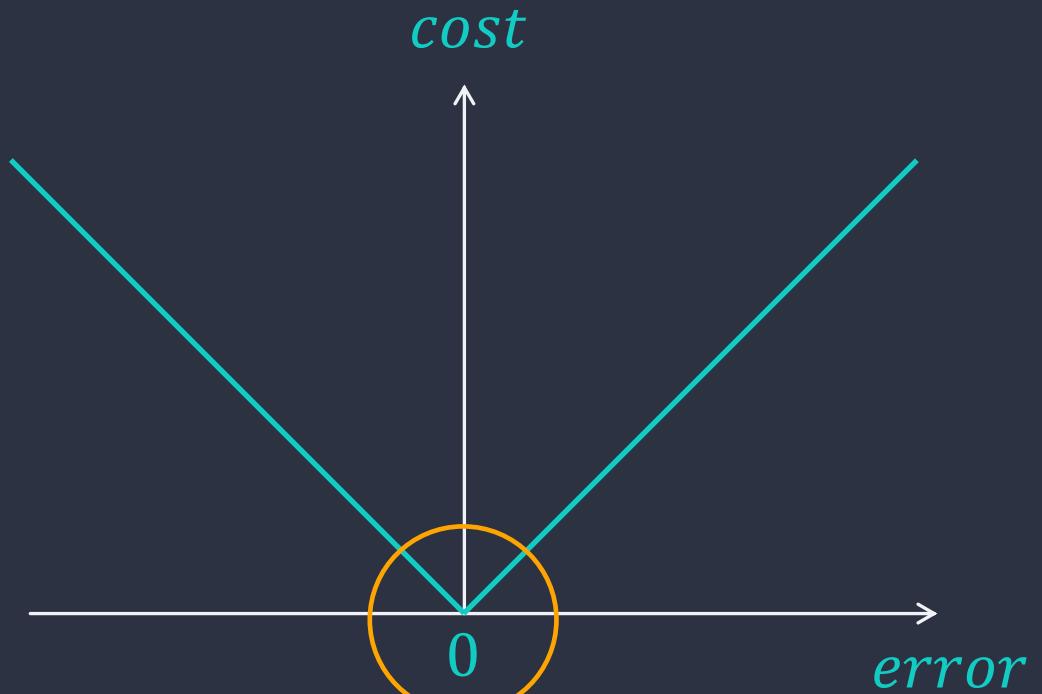
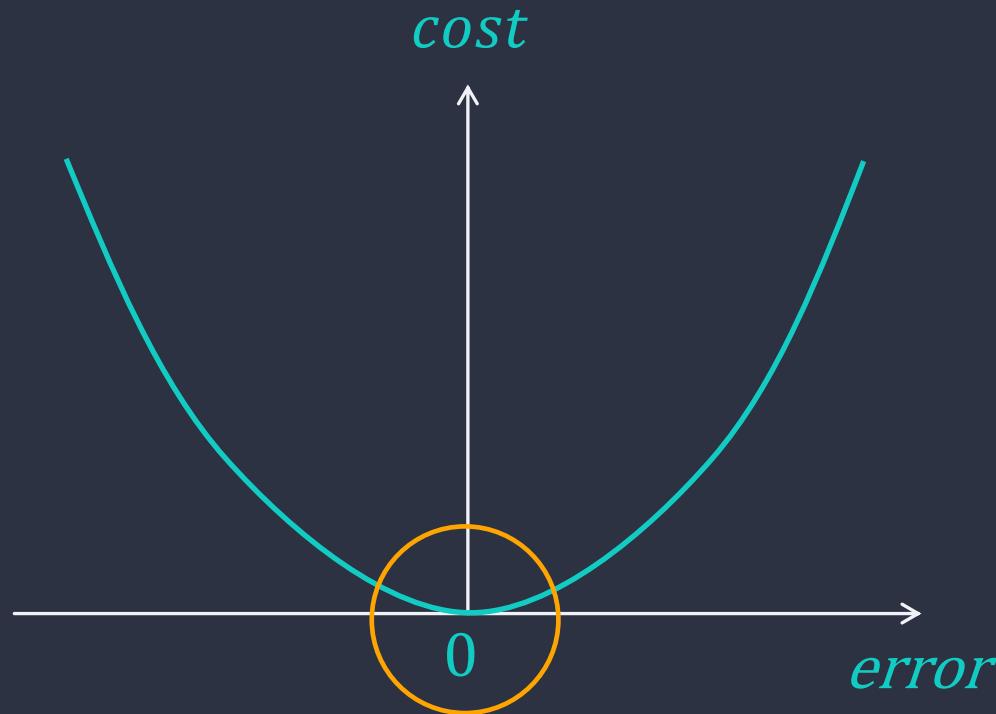
# MSE Cost Function

vs

# MAE Cost Function

$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

$$J = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}|$$



Reach minima when actual value ( $y$ ) is exactly equal to predicted value ( $\hat{y}$ ), i.e.  $\text{error} = y - \hat{y} = 0$ .

# MSE Cost Function vs MAE Cost Function

with slight variance

$$J_{MSE} = \frac{1}{2 \times 5} \cdot (0 + 0.25 + 1 + 2.25 + 4) = 0.75$$

$$J_{MAE} = \frac{1}{5} \cdot (0 + 0.5 + 1 + 1.5 + 2) = 1$$

$$J_{MSE} = \frac{1}{2 \times 6} \cdot (0 + 0.25 + 1 + 2.25 + 4 + 400) = 33.96$$

$$J_{MAE} = \frac{1}{6} \cdot (0 + 0.5 + 1 + 1.5 + 2 + 20) = 4.17$$

with outlier

<i>index</i>	<i>error</i>	<i>error</i> <sup>2</sup>	<i>error</i>
1	0	0	0
2	0.5	0.25	0.5
3	-1	1	1
4	1.5	2.25	1.5
5	-2	4	2
6	outlier	400	20

# MSE Cost Function

vs

# MAE Cost Function

with slight variance

$$J_{MSE} = \frac{1}{2 \times 5} \cdot (0 + 0.25 + 1 + 2.25 + 4) = 0.75$$

$$J_{MAE} = \frac{1}{5} \cdot (0 + 0.5 + 1 + 1.5 + 2) = 1$$

with outlier

$$J_{MSE} = \frac{1}{2 \times 6} \cdot (0 + 0.25 + 1 + 2.25 + 4 + 400) = 33.96$$

$$J_{MAE} = \frac{1}{6} \cdot (0 + 0.5 + 1 + 1.5 + 2 + 20) = 4.17$$

$$J_{RMSE} = J_{\sqrt{MSE}} = 5.83$$

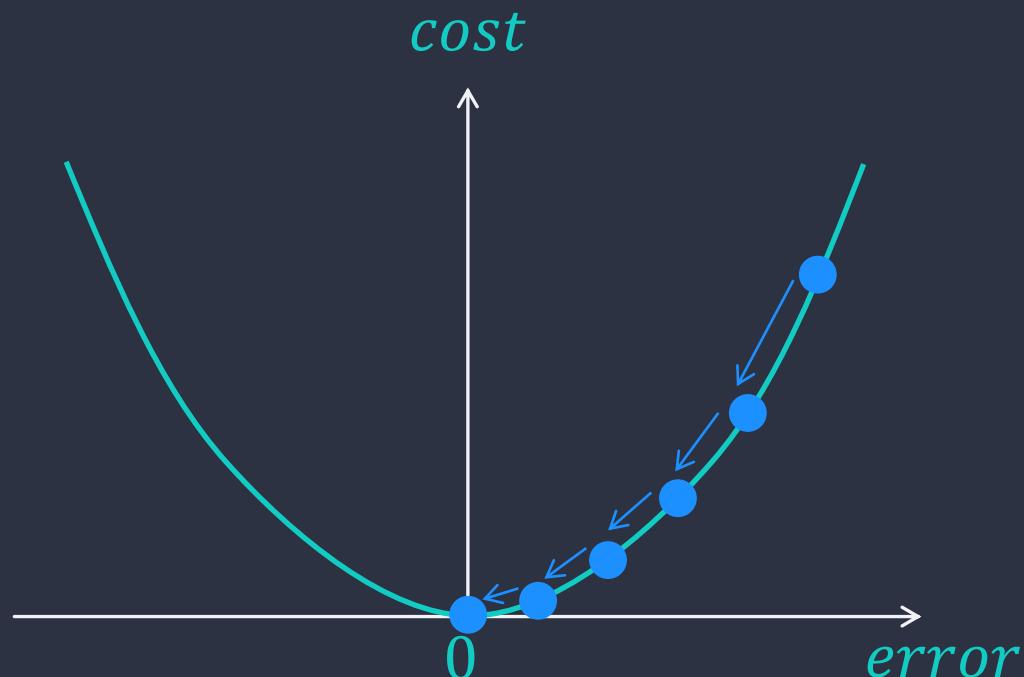
<i>index</i>	<i>error</i>	<i>error</i> <sup>2</sup>	$ error $
1	0	0	0
2	0.5	0.25	0.5
3	-1	1	1
4	1.5	2.25	1.5
5	-2	4	2
6	outlier	400	20

# MSE Cost Function

vs

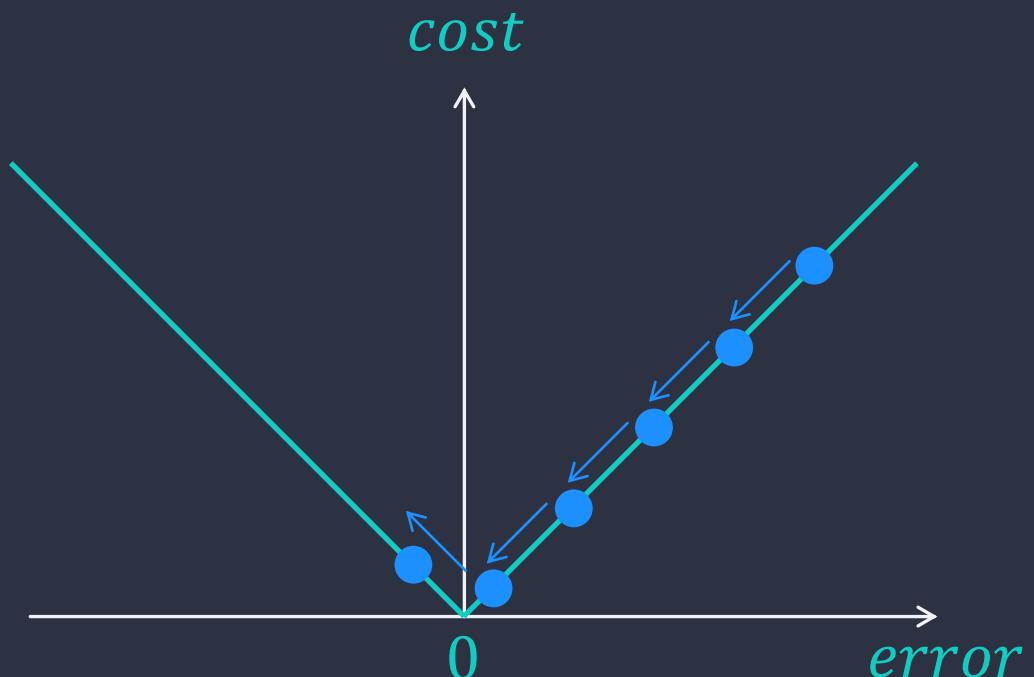
# MAE Cost Function

A big issue in MAE!



gradient becomes smaller  
with fixed learning rate

Will converge even learning rate  
remains the same.



gradient remains the same  
with fixed learning rate

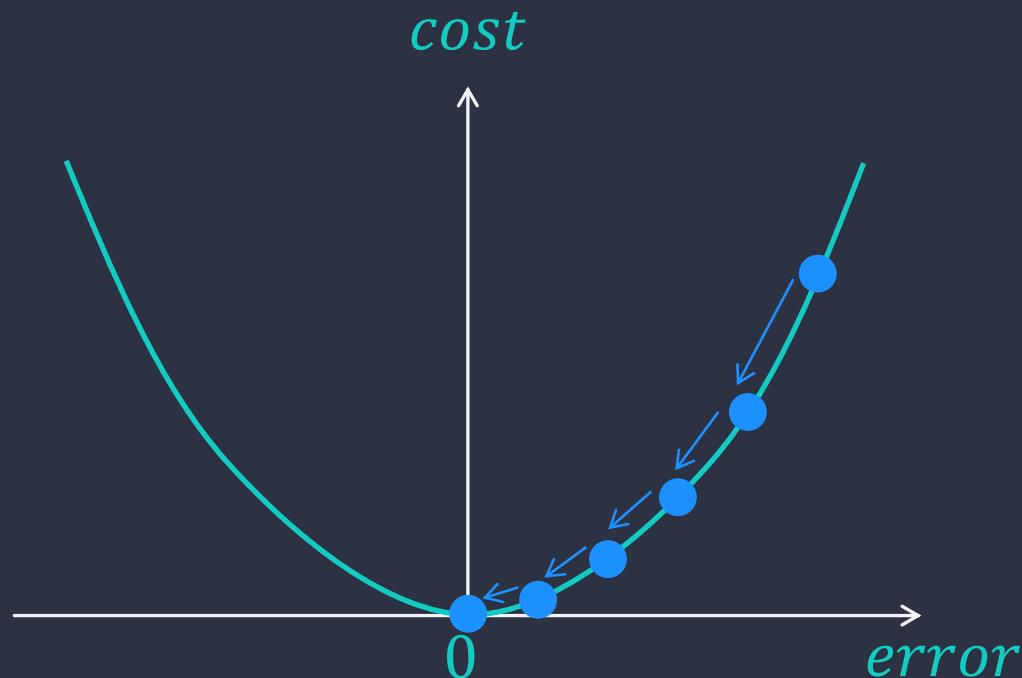
Could make it dynamic – decrease as  
approaching 0.

# MSE Cost Function

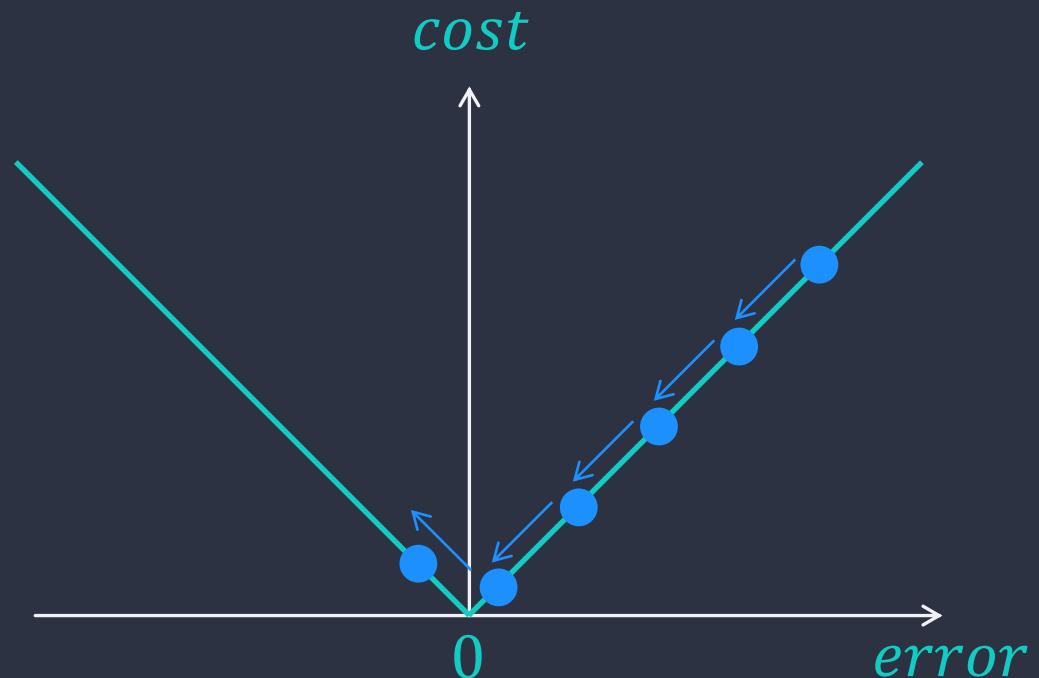
vs

# MAE Cost Function

When to use which?



When outliers represent anomalies



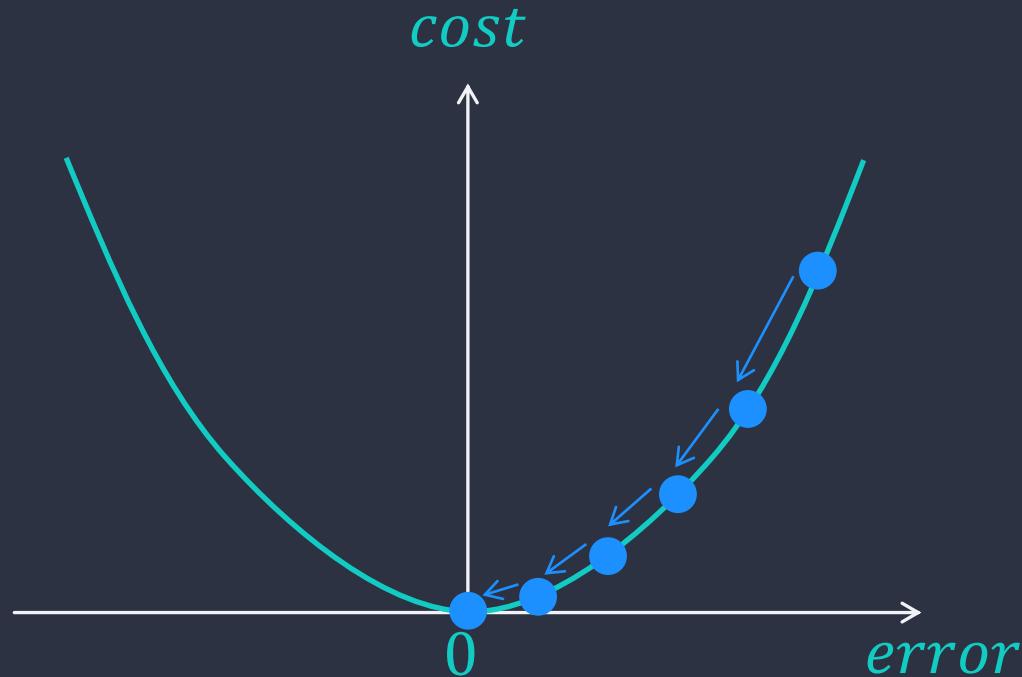
When outliers represent corrupted data

# MSE Cost Function

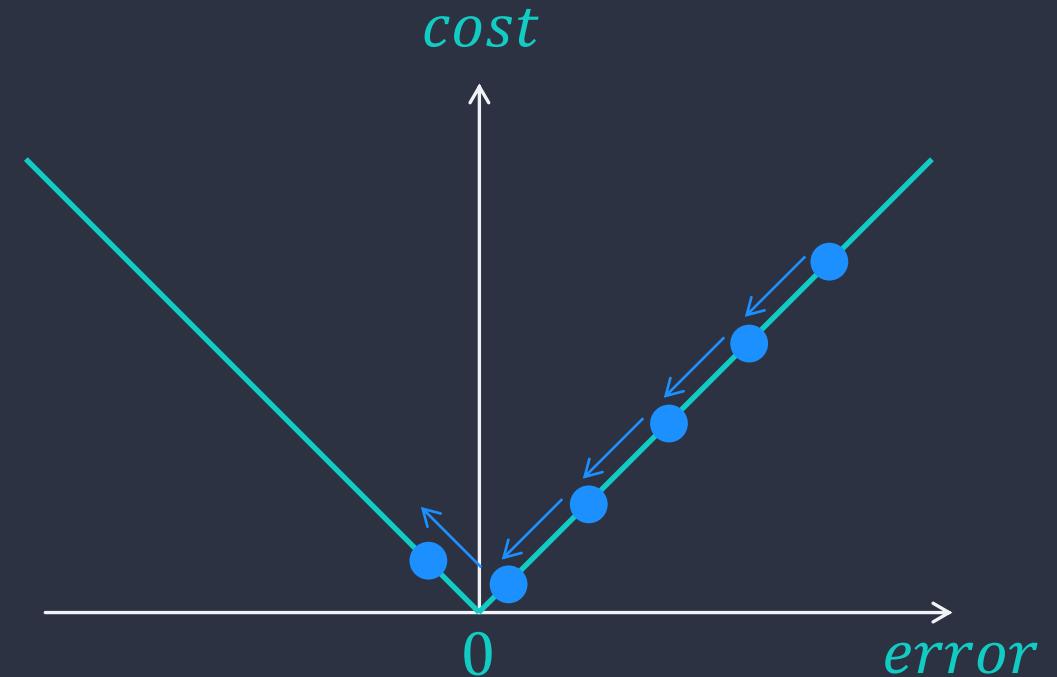
vs

# MAE Cost Function

Issue for both, when learning from skewed/imbalanced data.



Got skewed towards outliers,  
achieving low accuracy.



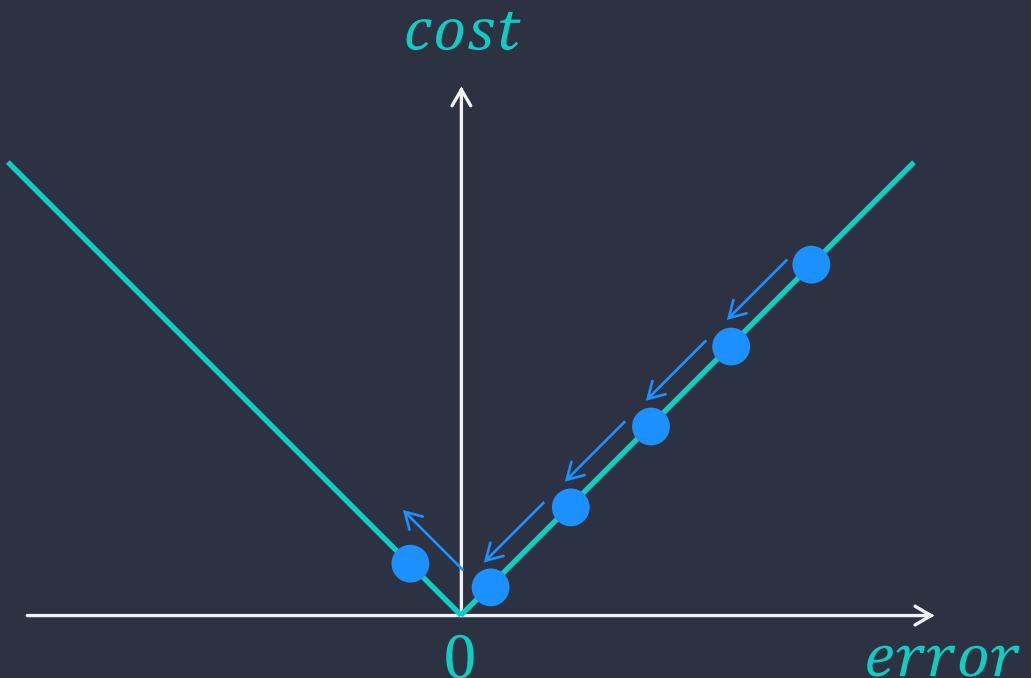
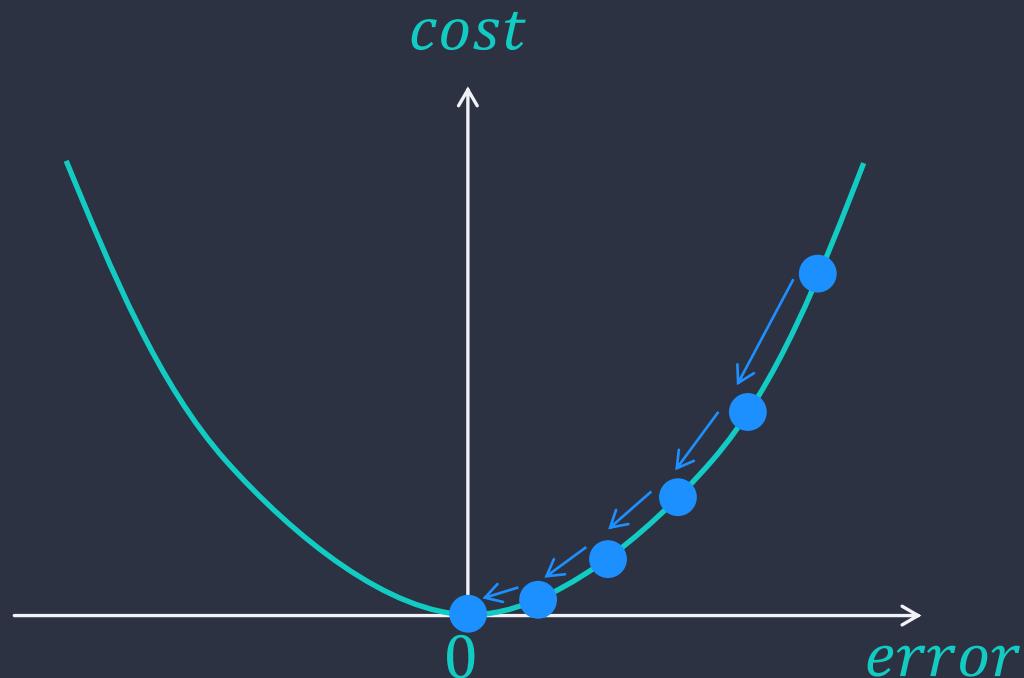
Ignoring outliers and achieving  
unrealistic high accuracy.

# MSE Cost Function

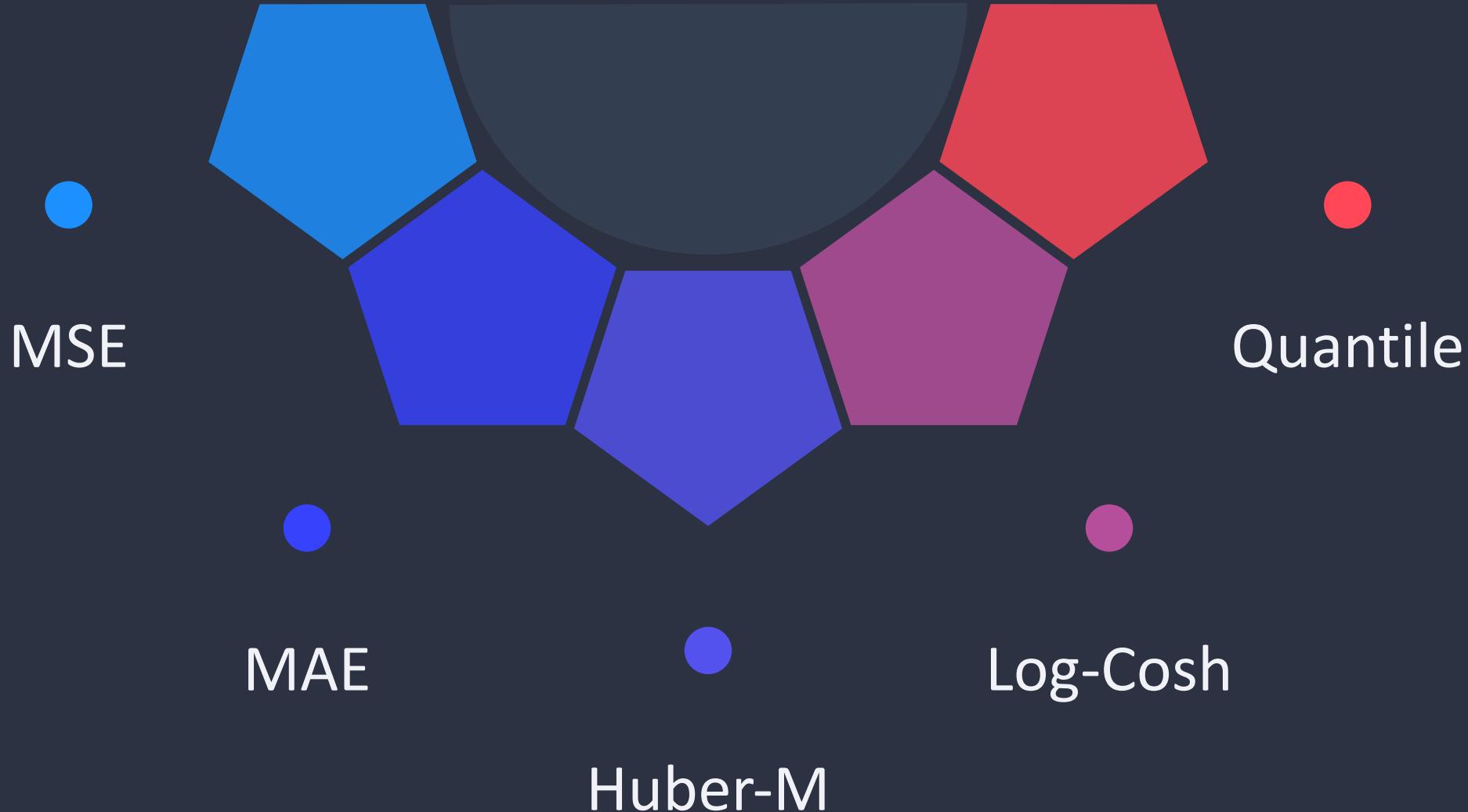
vs

# MAE Cost Function

Issue for both, when learning from skewed/imbalanced data.



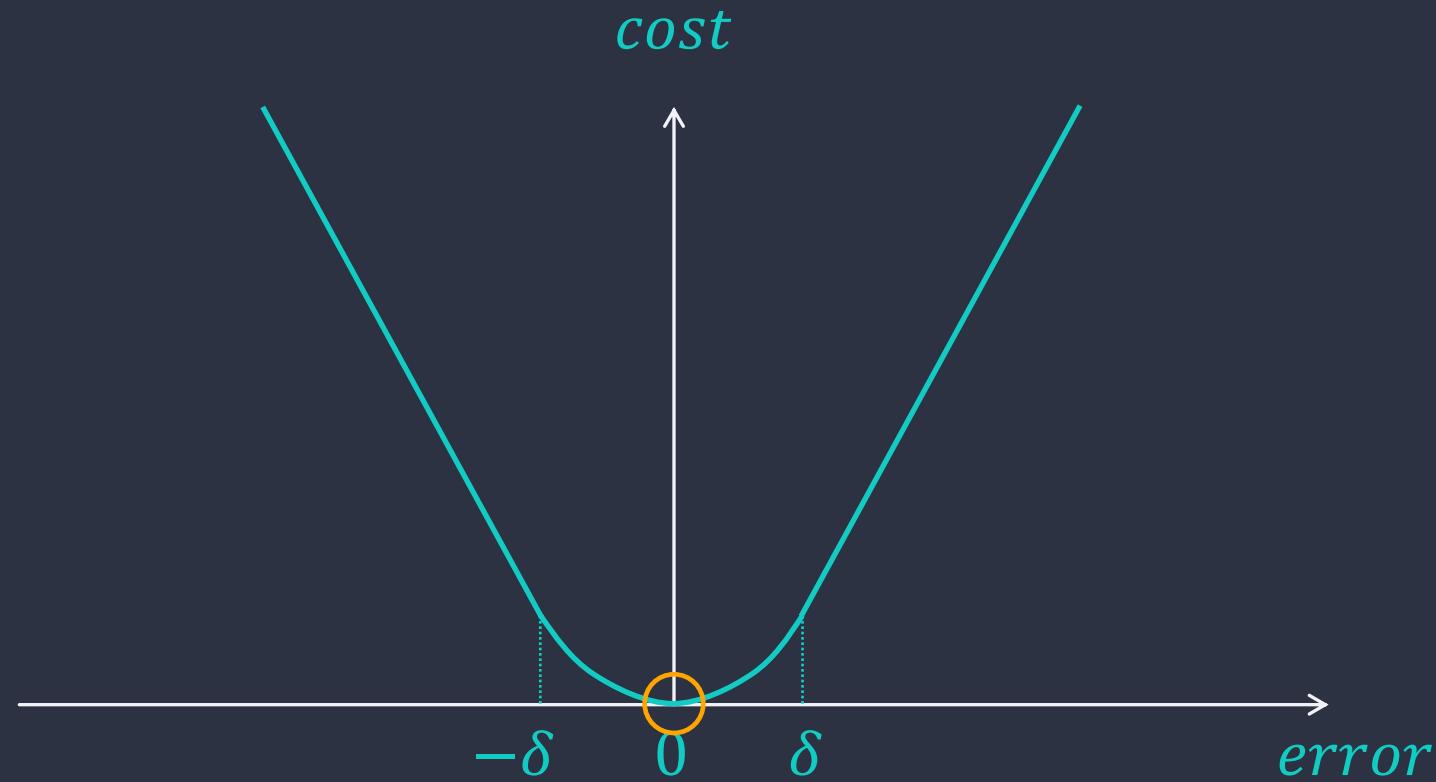
Solutions: data transformation,  
or, Huber-M (in the next section)



### 3. Huber-M Cost Function

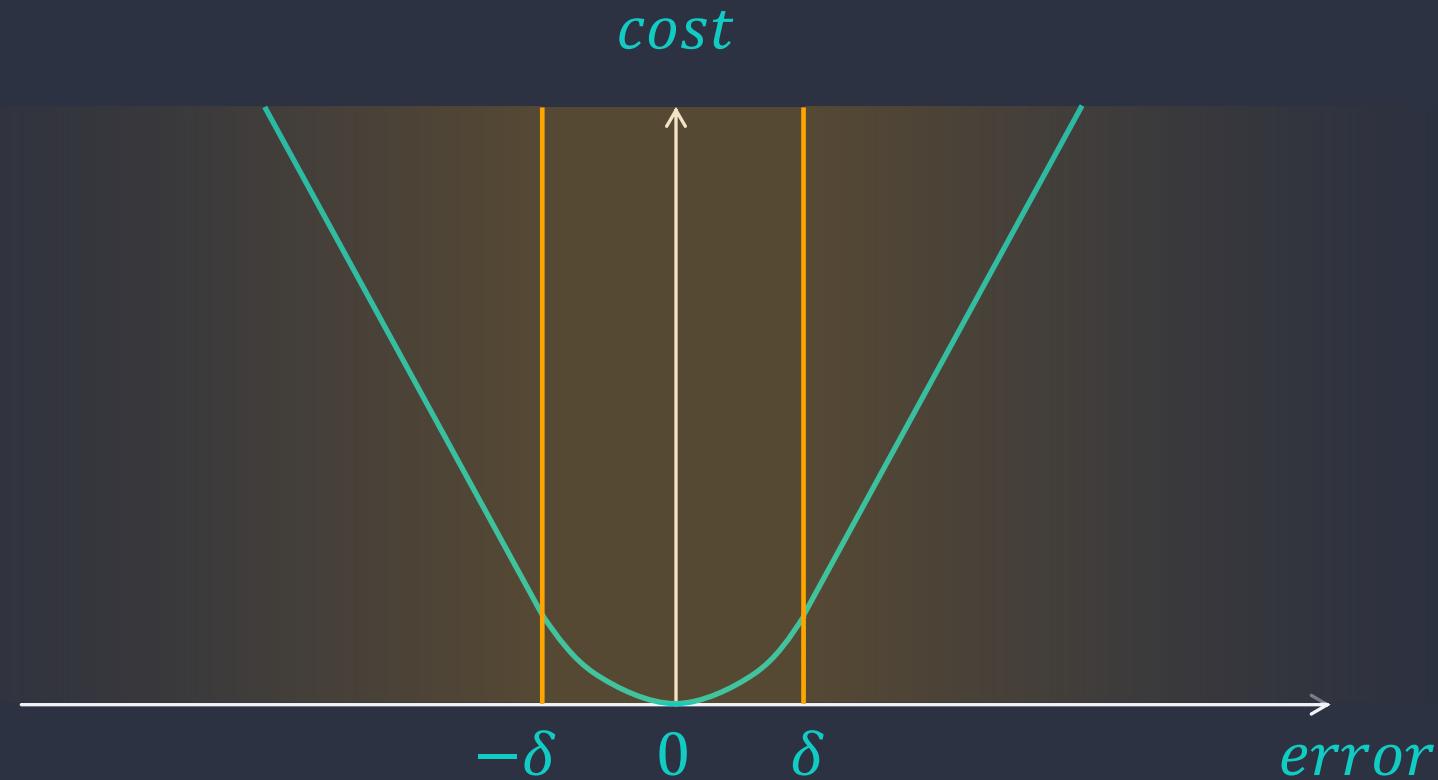
# Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & otherwise \end{cases}$$



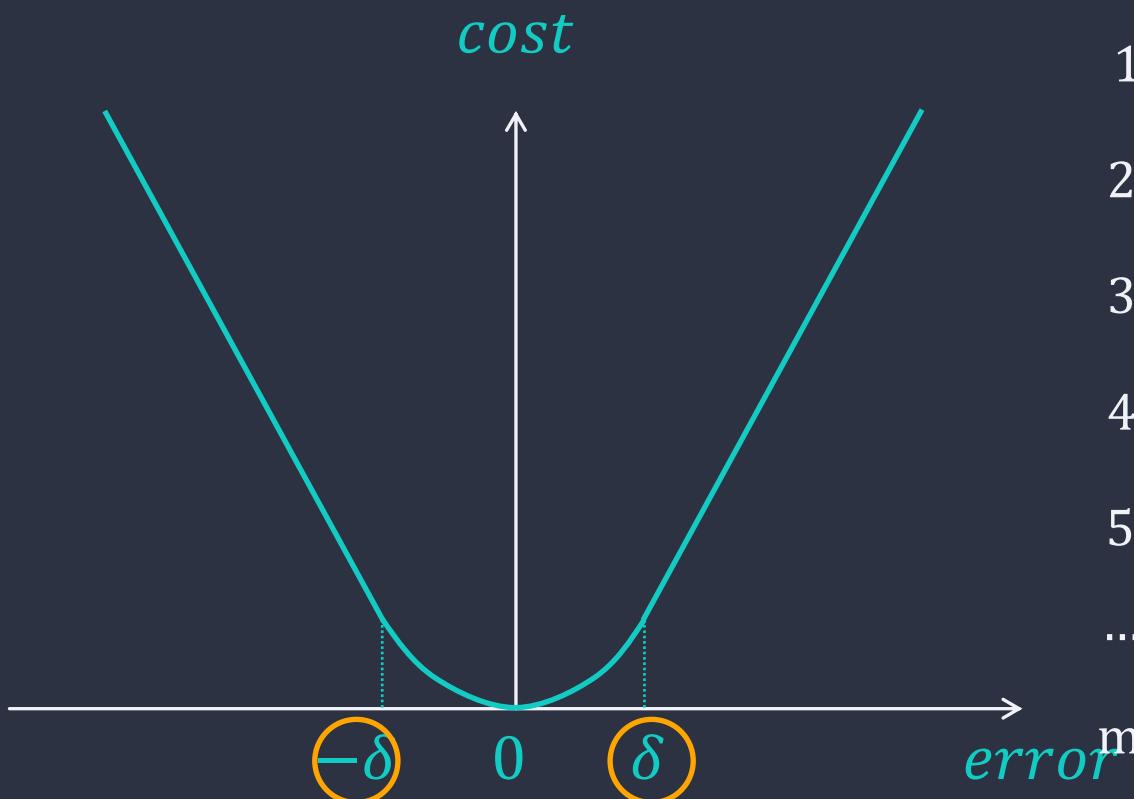
# Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & otherwise \end{cases}$$



# Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$



1 2 3 4 5 ... n

1    □    □    □    □    □    ...    □

2    □    □    □    □    □    ...    □

3    □    □    □    □    □    ...    □

4    □    □    □    □    □    ...    □

5    □    □    □    □    □    ...    □

...    ...    ...    ...    ...    ...    ...

...    ...    ...    ...    ...    ...    ...

label (predicted)

$y$	$\hat{y}$	error
-----	-----------	-------

:: max |error|

:: :

:: 10%  $\delta$

:: :

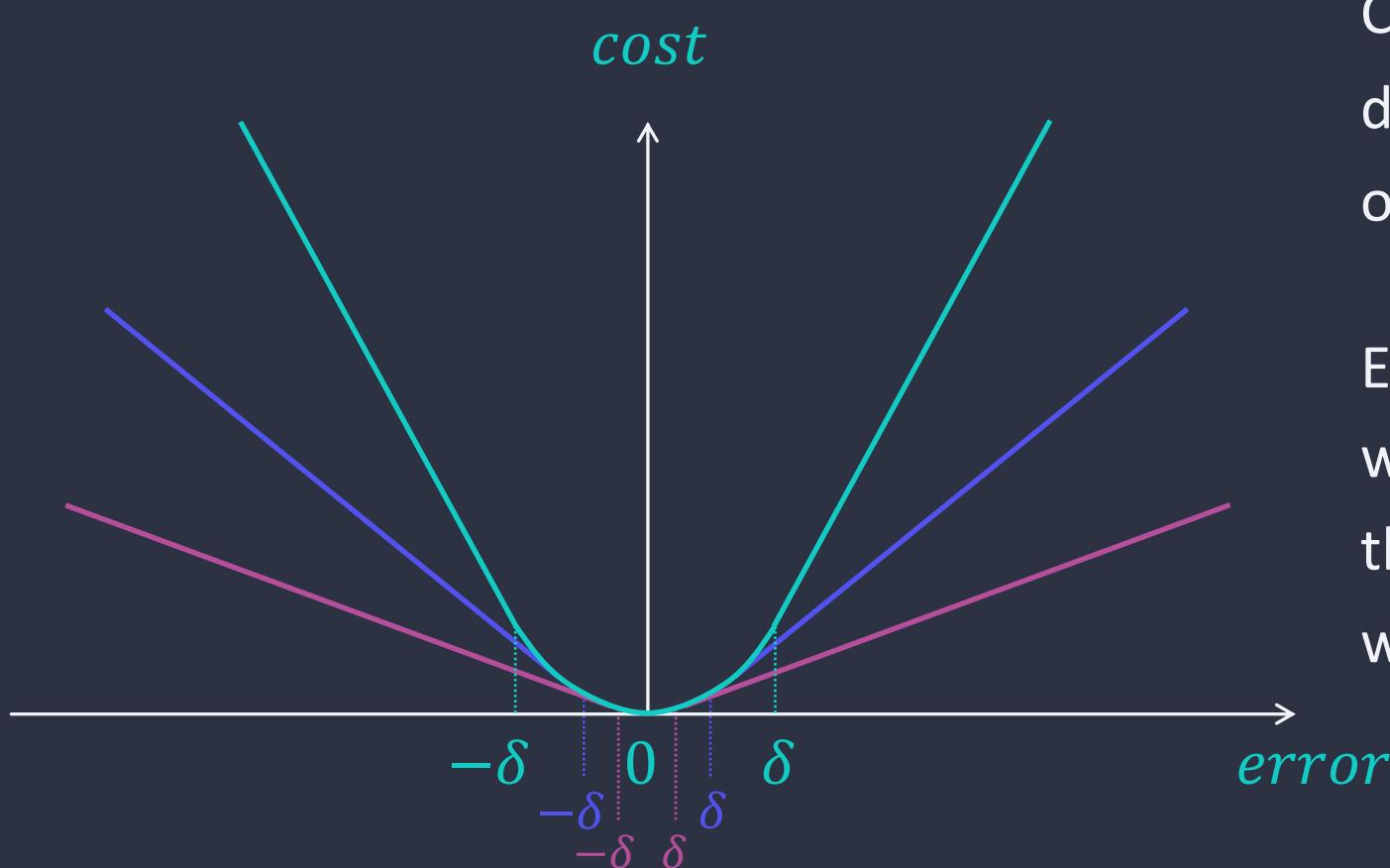
:: :

:: :

:: min |error|

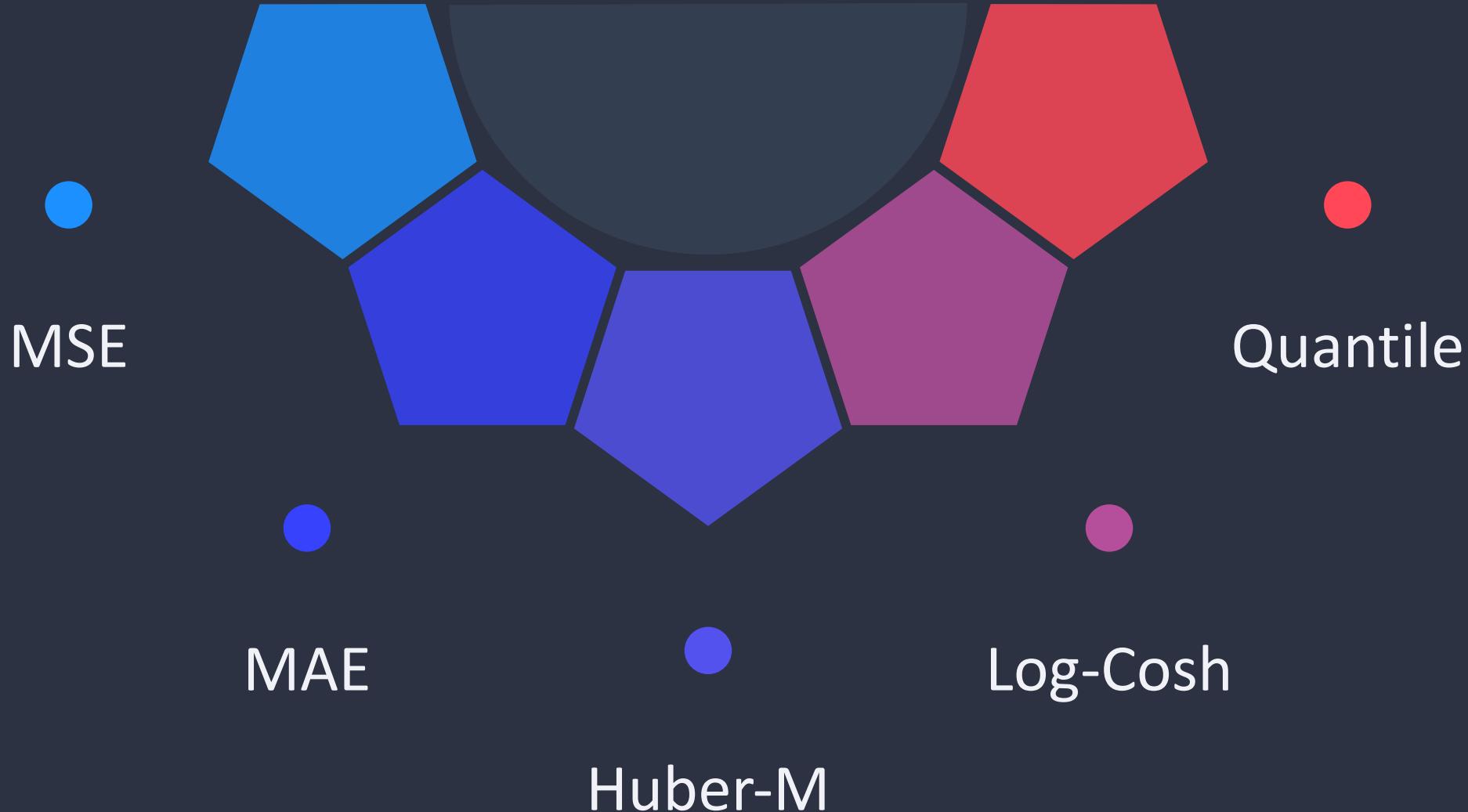
# Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & otherwise \end{cases}$$



Choice of  $\delta$  is very important, as it determines what are considered as outliers and how they are dealt with.

Errors smaller than  $\delta$  are minimised with MSE, whereas errors larger than  $\delta$  are minimised with MAE with less sensitive to large outliers.



## 4. Log-Cosh Cost Function

# Log-Cosh Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \log(\cosh(y^{(i)} - \hat{y}^{(i)}))$$

$$f(x) = \log(\cosh(x)) = \text{logcosh}(x) = \ln \frac{e^x + e^{-x}}{2}$$

If  $x$  is very large     $x > 0$      $= \ln \frac{e^x + e^{-x}}{2} = \ln \frac{e^x}{2} = \ln(e^x) - \ln 2 = x - \ln 2$

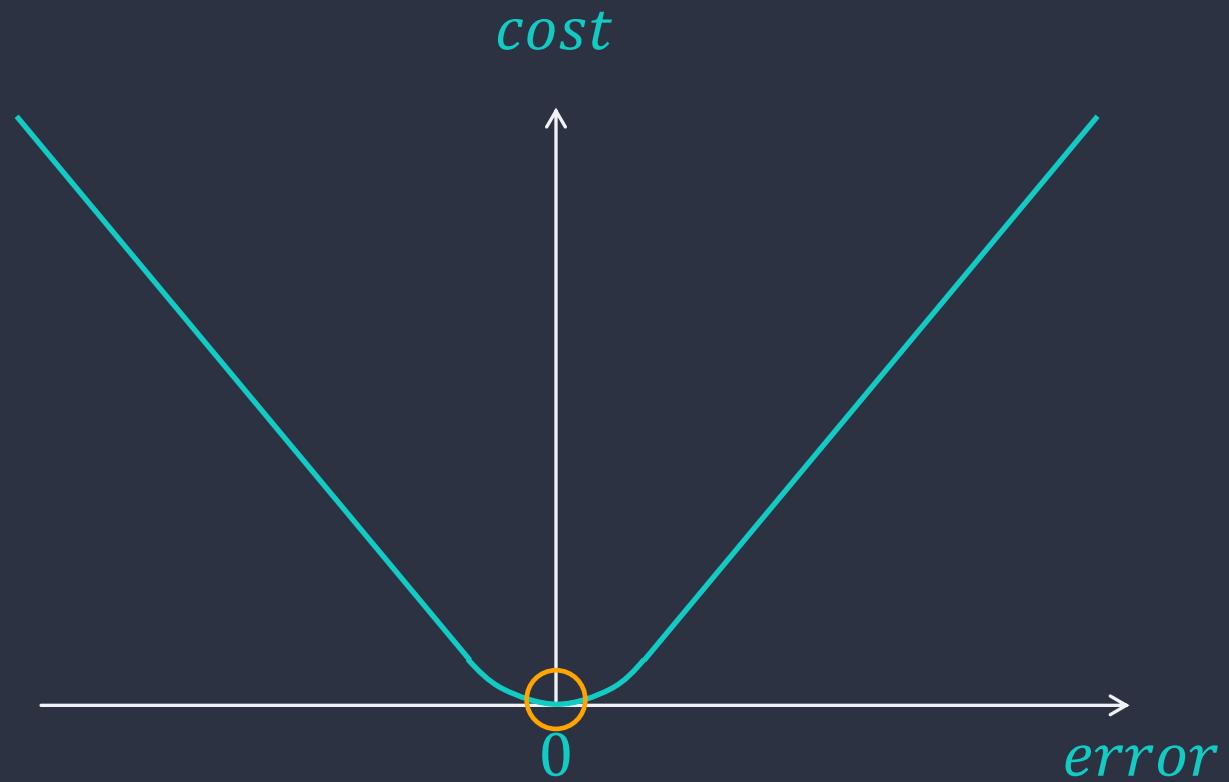
$x < 0$      $= \ln \frac{e^x + e^{-x}}{2} = \ln \frac{e^{-x}}{2} = \ln(e^{-x}) - \ln 2 = -x - \ln 2$

$$\log(\cosh(x)) = |x| - \ln 2$$

If  $x$  is very small     $\log(\cosh(x)) = \frac{x^2}{2} - \frac{x^4}{12} + \frac{x^6}{45} - \dots = \frac{x^2}{2}$

# Log-Cosh Cost Function

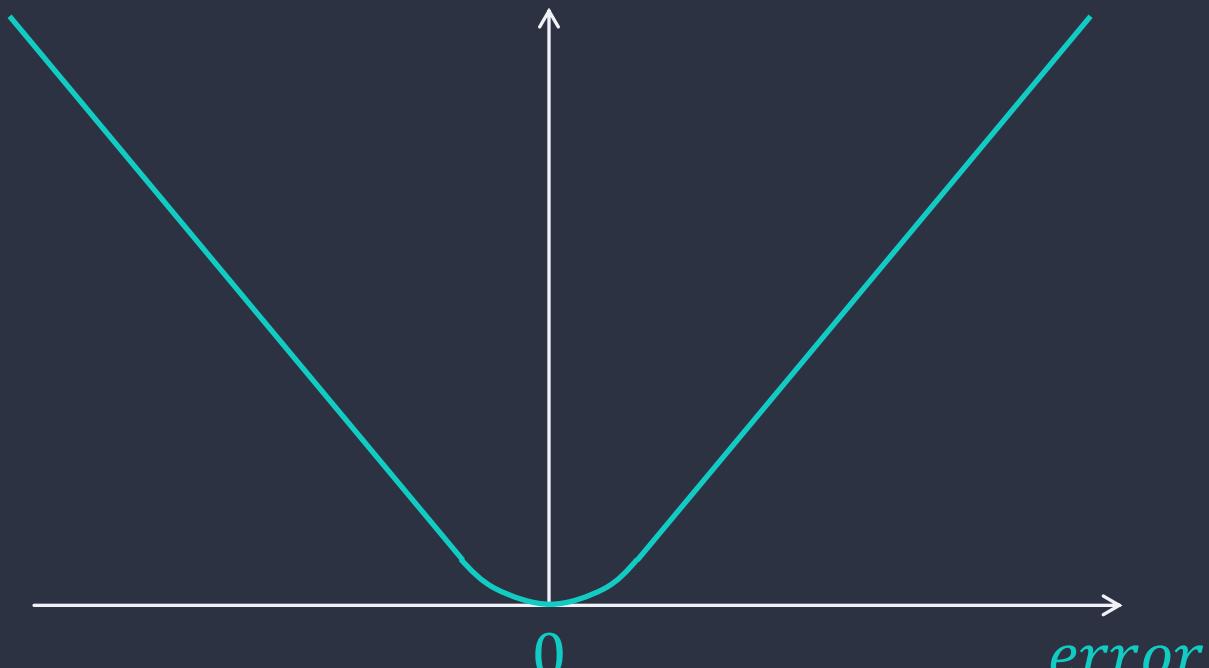
$$J = \frac{1}{m} \sum_{i=1}^m \log(\cosh(y^{(i)} - \hat{y}^{(i)}))$$



## Log-Cosh Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \log(\cosh(y^{(i)} - \hat{y}^{(i)}))$$

*cost*

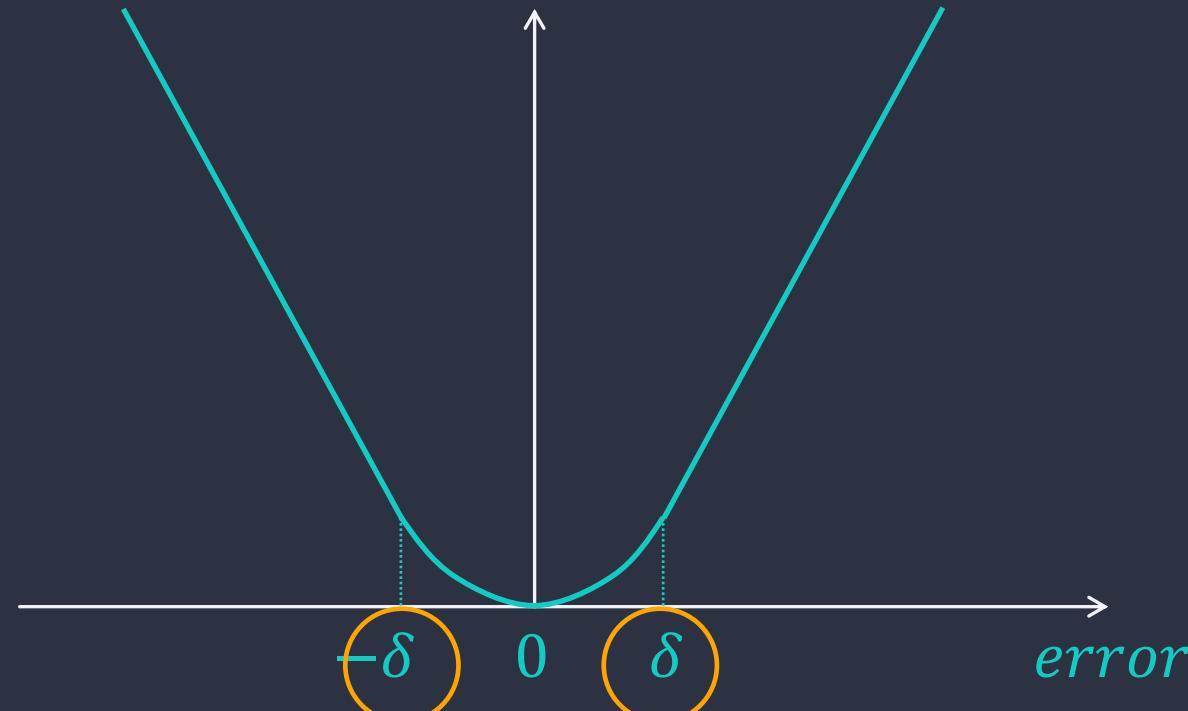


No hyperparameter

## Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & otherwise \end{cases}$$

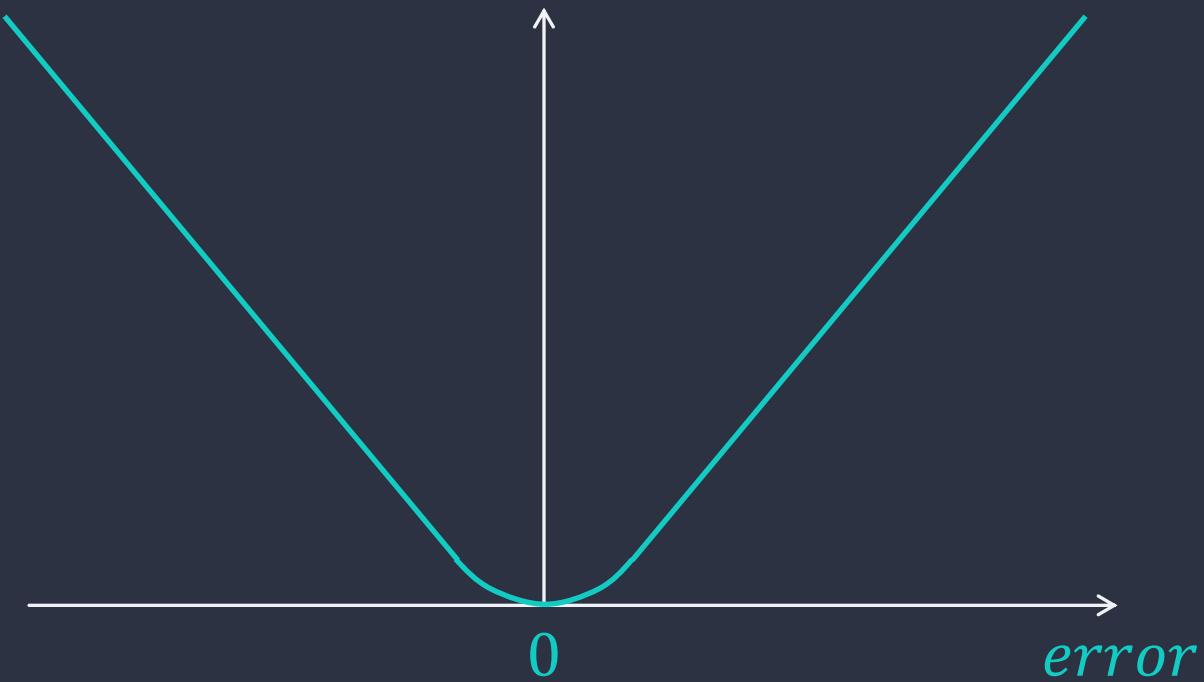
*cost*



## Log-Cosh Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \log(\cosh(y^{(i)} - \hat{y}^{(i)}))$$

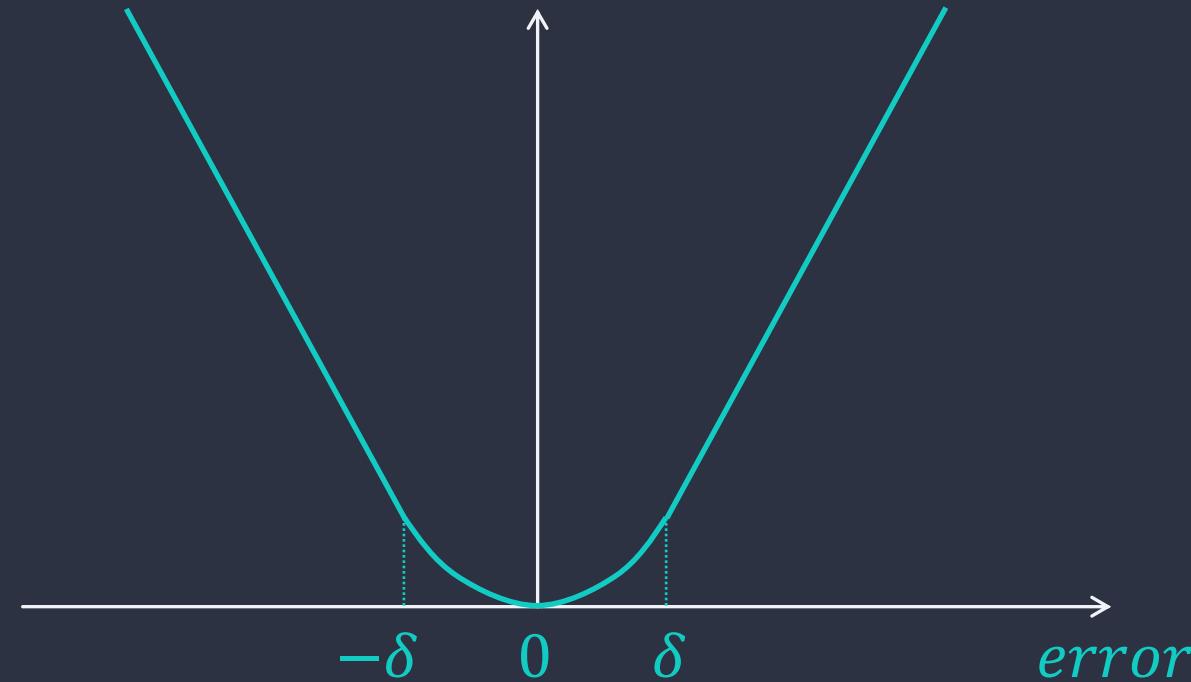
*cost*



## Huber-M Cost Function

$$J = \frac{1}{m} \sum_{i=1}^m \begin{cases} \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2, & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta), & otherwise \end{cases}$$

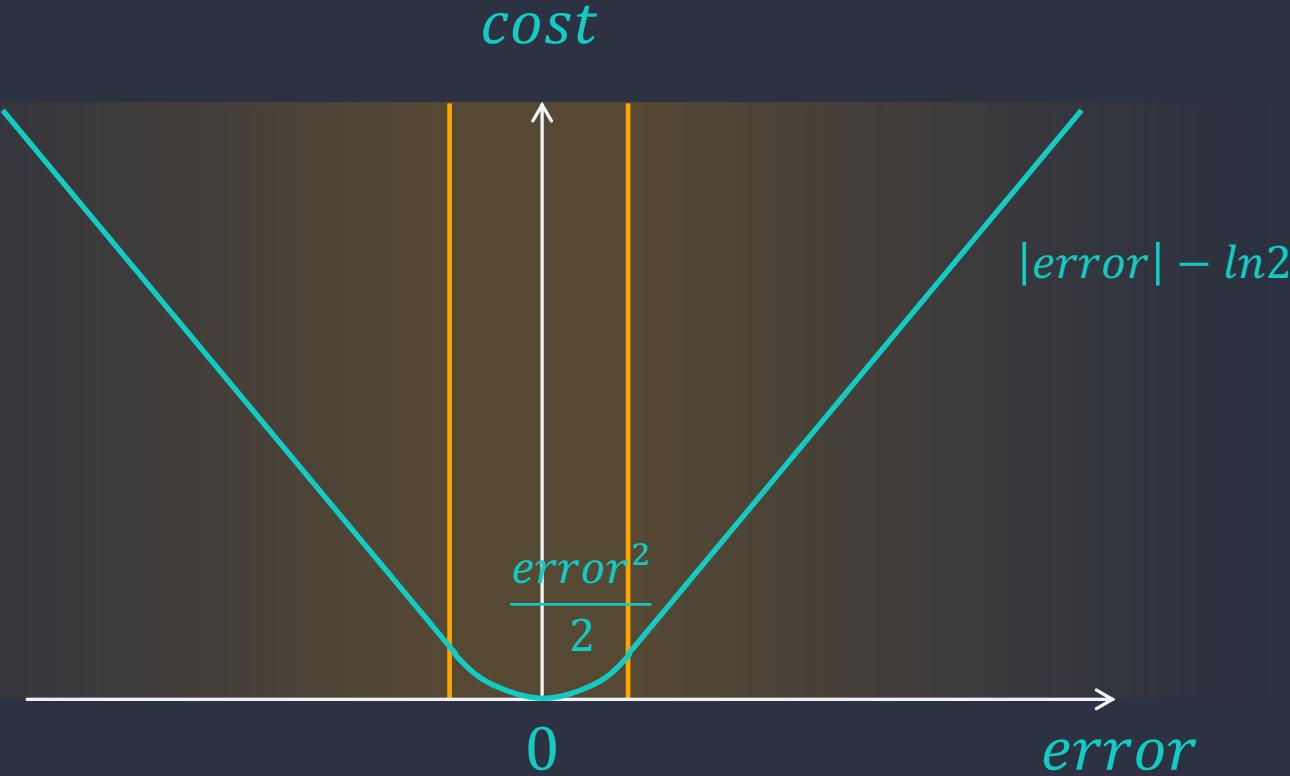
*cost*

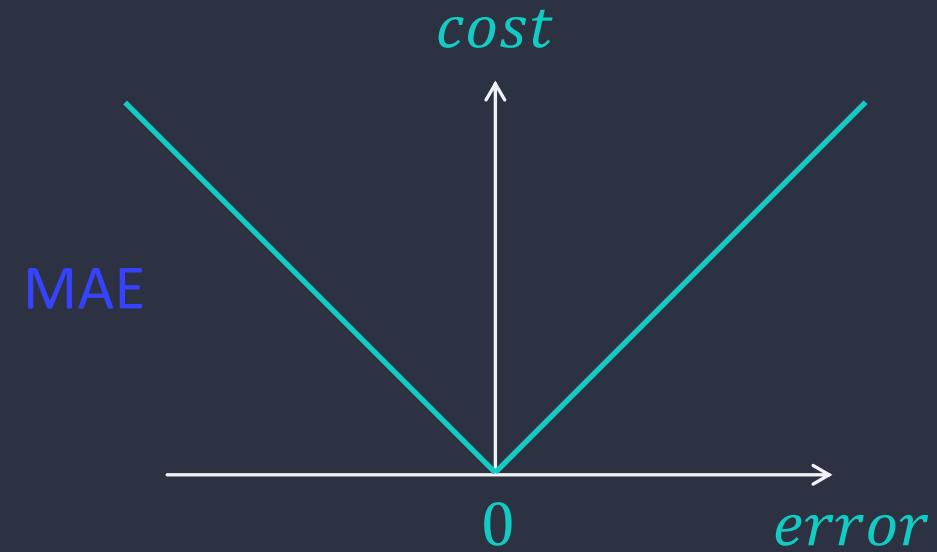
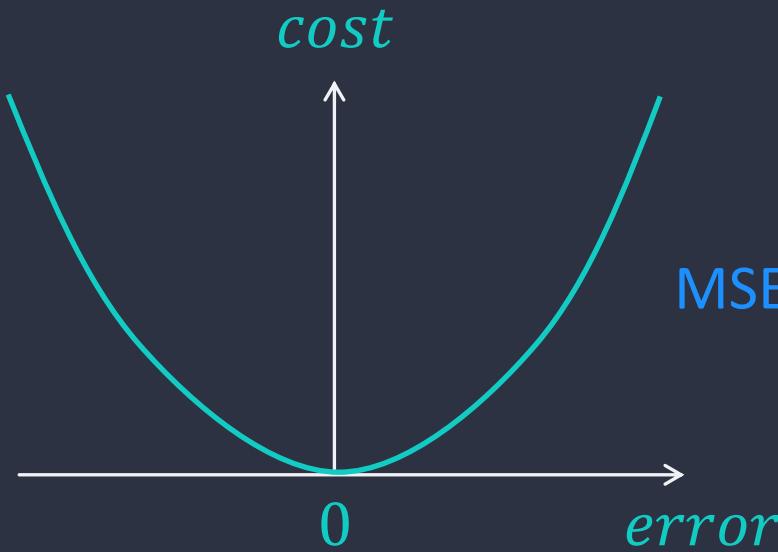


# Log-Cosh Cost Function

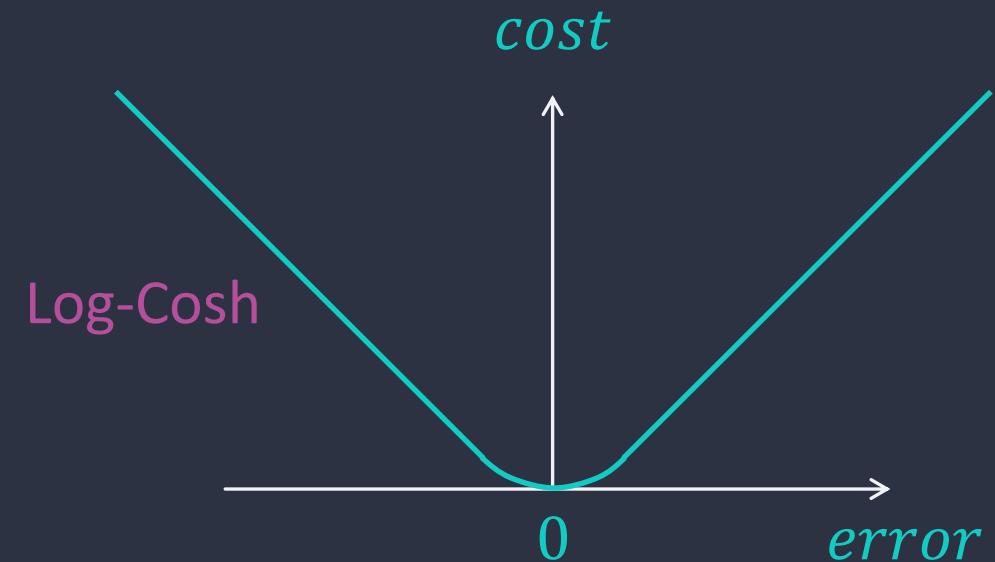
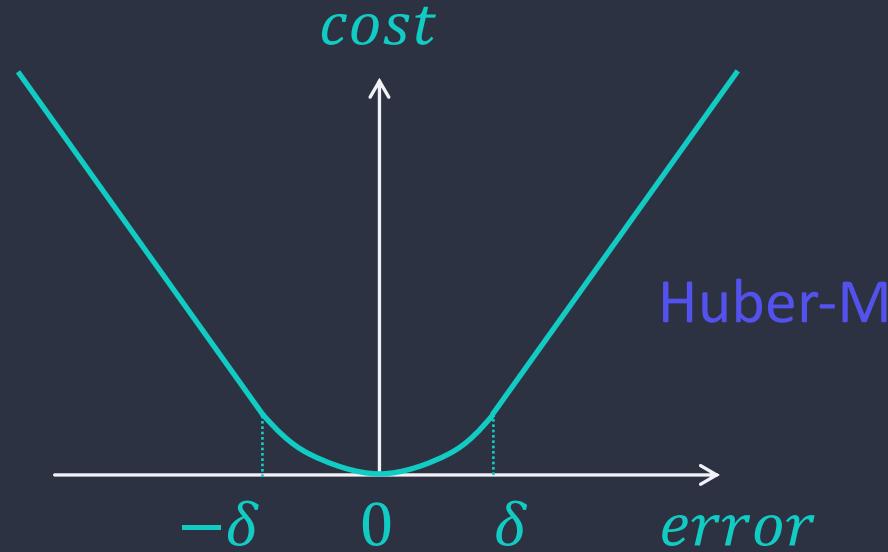
$$J = \frac{1}{m} \sum_{i=1}^m \log(\cosh(y^{(i)} - \hat{y}^{(i)}))$$

Twice differentiable everywhere.





## Point Prediction



# Alternative to Point Prediction?

# Prediction Interval

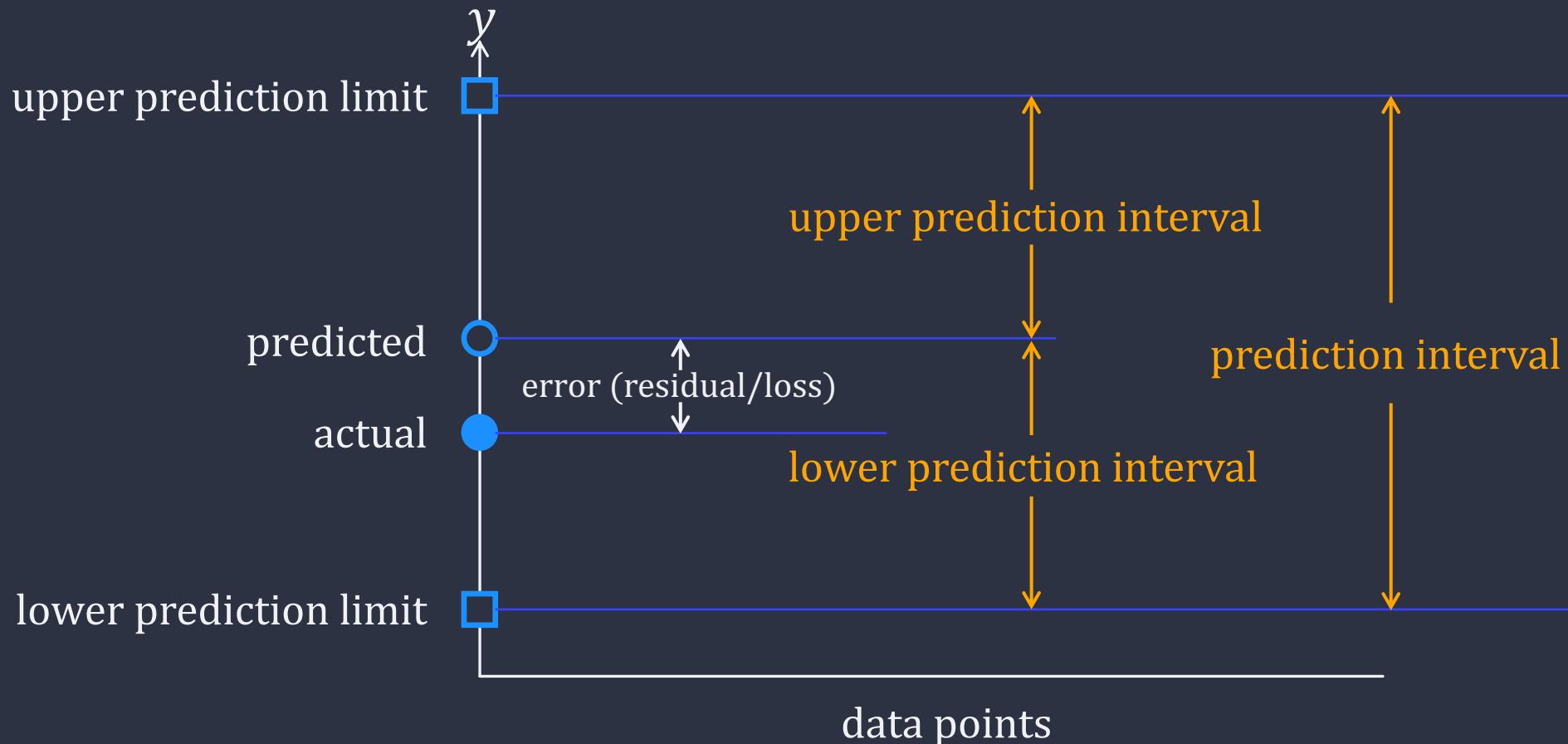
## Alternative to Point Prediction

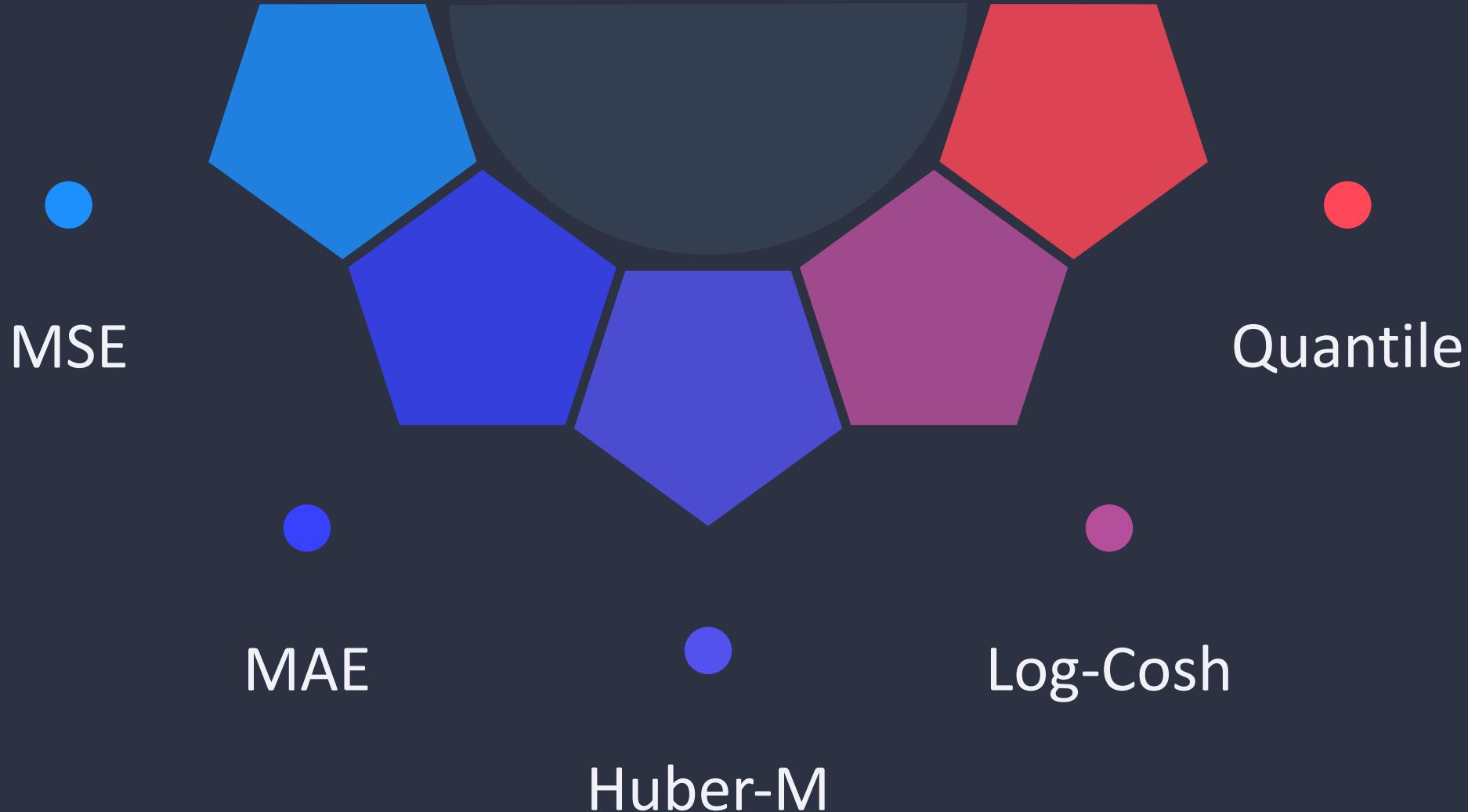
- Instead of prediction of a single value, interval prediction predicts an interval in which a future data point will fall, with a certain probability.
- Different from confidence interval
  - Confidence Interval: quantifies uncertainty on a predicted population variable e.g. mean, standard deviation.
  - Prediction Interval: quantifies the uncertainty on a single data point predicted from the population.

# Prediction Interval

- A quantification of the uncertainty on a prediction

- Can be used as measure of the likeliness of the correctness of the prediction
- Determines an interval of possible values -> provide probabilistic upper & lower prediction limits





## 5. Quantile Cost Function

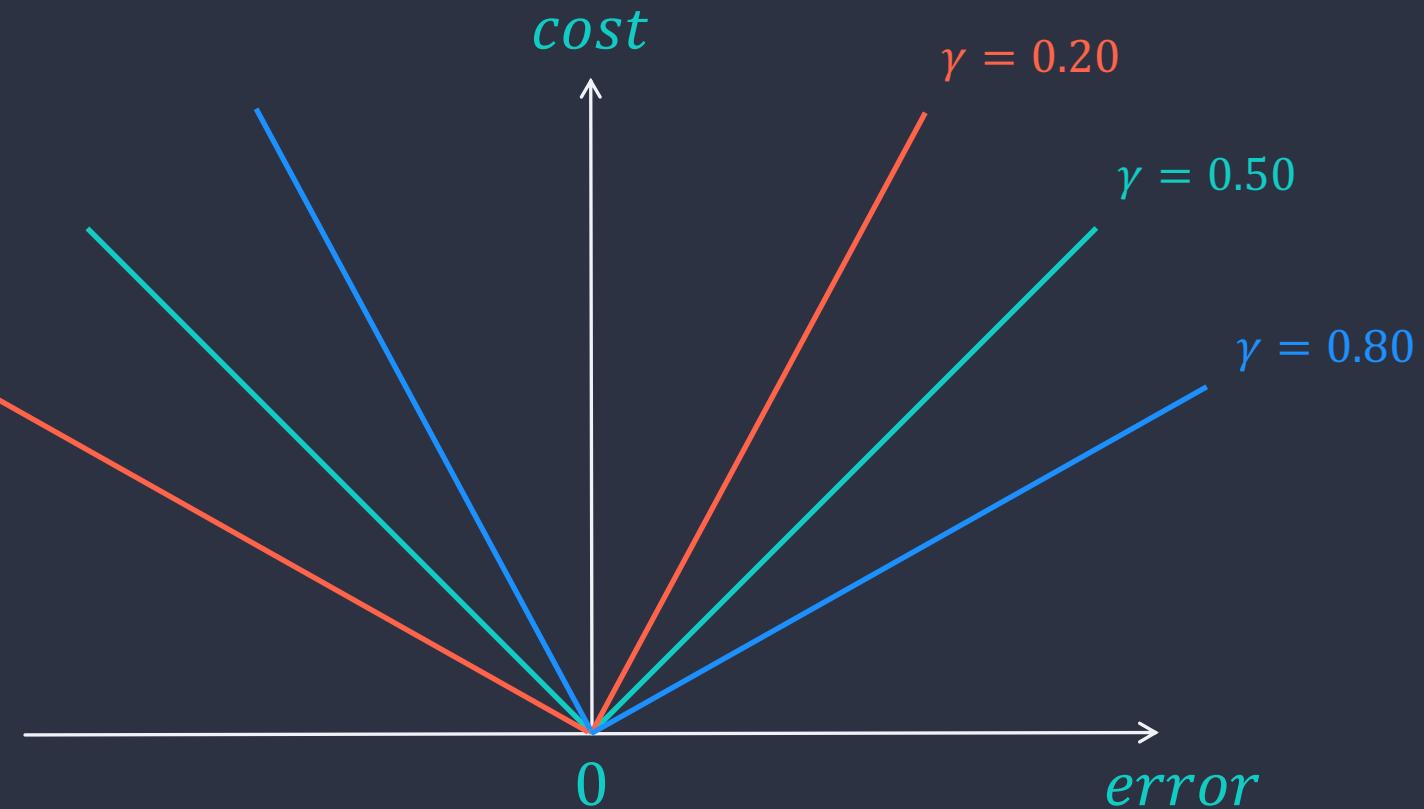
# Quantile Cost Function

$$J = \frac{1}{m} \left( \sum_{y^{(i)} < \hat{y}^{(i)}} (\gamma - 1) |y^{(i)} - \hat{y}^{(i)}| + \sum_{y^{(i)} > \hat{y}^{(i)}} \gamma |y^{(i)} - \hat{y}^{(i)}| \right) \quad \gamma \in (0,1)$$

- Like MAE cost function, but with an additional hyperparameter  $\gamma$ , which specifies the  $\gamma$ -th quantile of the dependent variable.
- Differs depending upon the evaluated quantile:
  - more negative errors are penalised more when we specify a higher quantile, and
  - more positive errors are penalised more for lower quantiles.

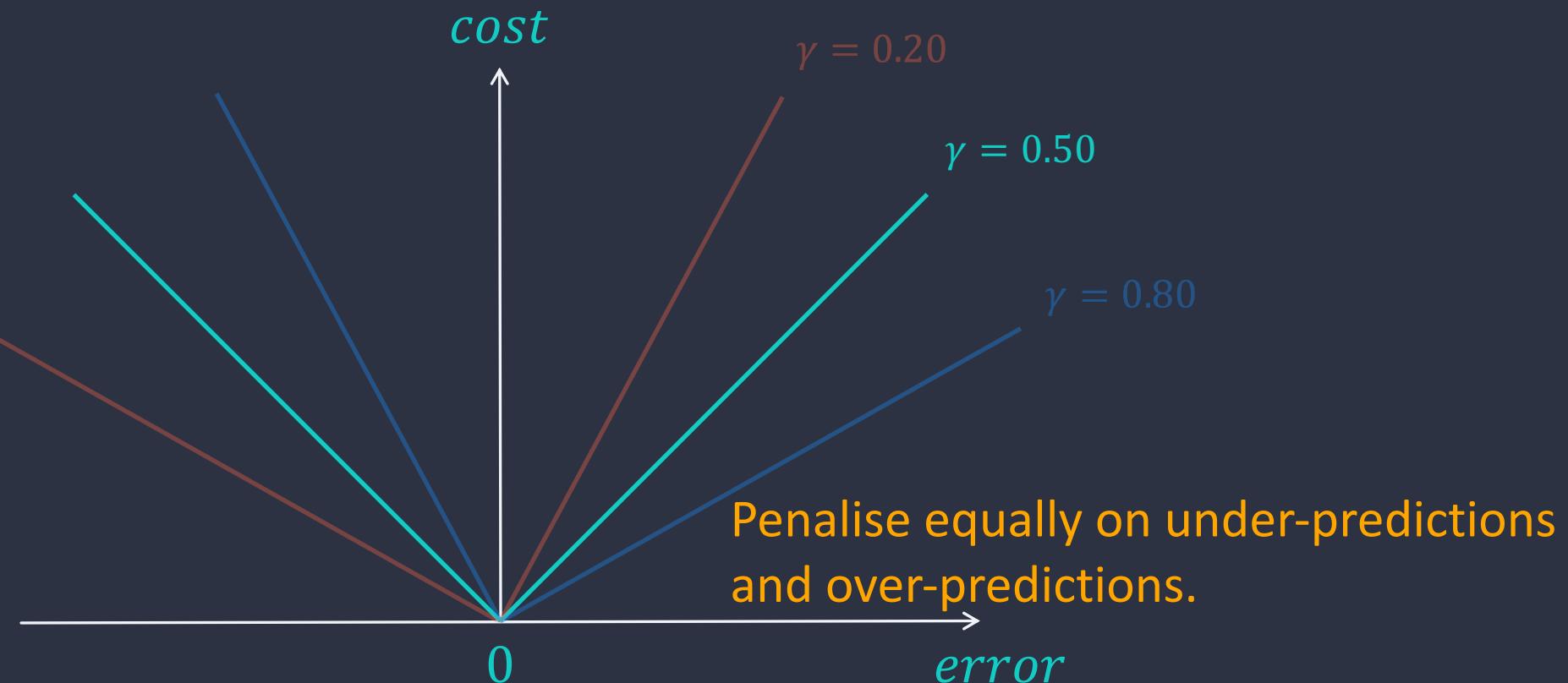
# Quantile Cost Function

$$J = \frac{1}{m} \left( \sum_{y^{(i)} < \hat{y}^{(i)}} (\gamma - 1) |y^{(i)} - \hat{y}^{(i)}| + \sum_{y^{(i)} > \hat{y}^{(i)}} \gamma |y^{(i)} - \hat{y}^{(i)}| \right) \quad \gamma \in (0,1)$$



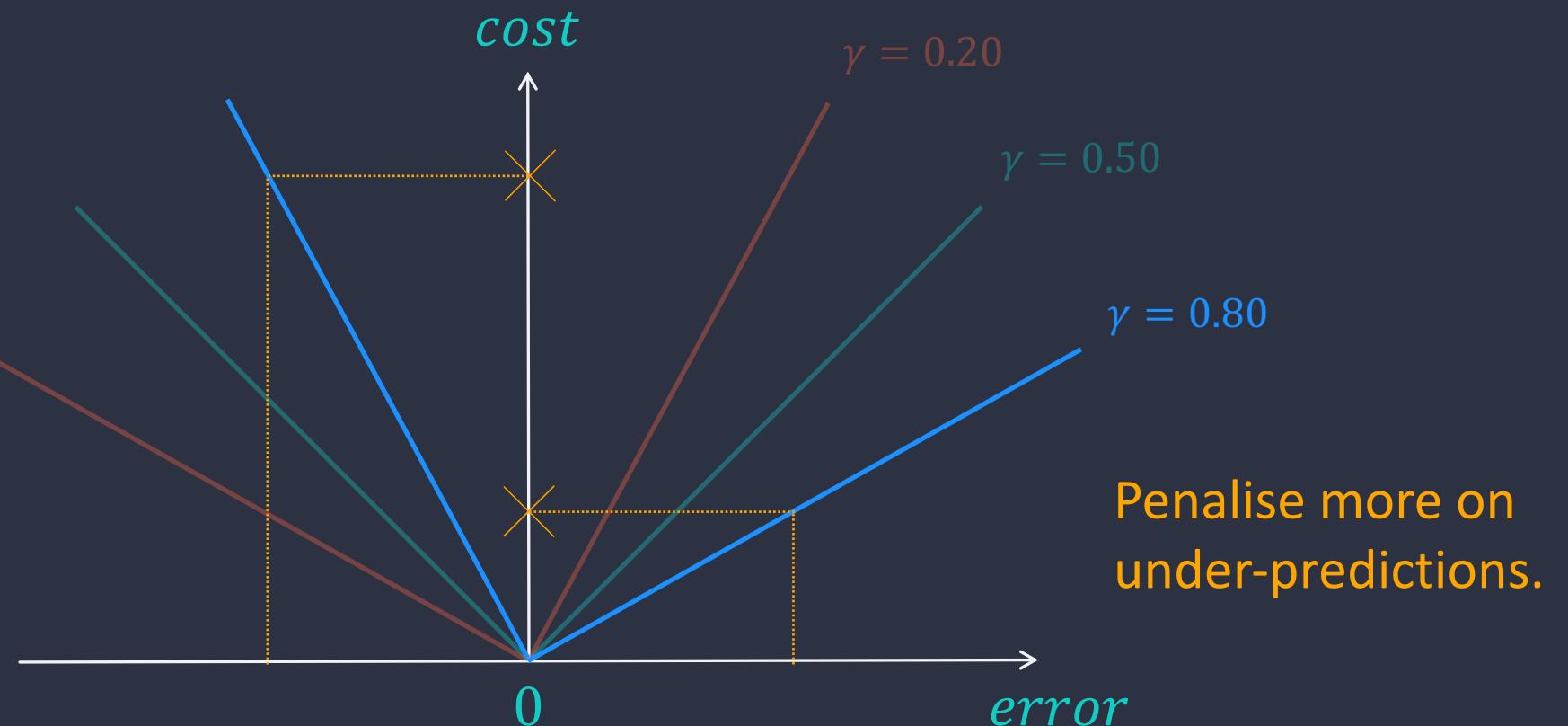
# Quantile Cost Function

$$J = \frac{1}{m} \left( \sum_{y^{(i)} < \hat{y}^{(i)}} (\gamma - 1) |y^{(i)} - \hat{y}^{(i)}| + \sum_{y^{(i)} > \hat{y}^{(i)}} \gamma |y^{(i)} - \hat{y}^{(i)}| \right) \quad \gamma \in (0,1)$$



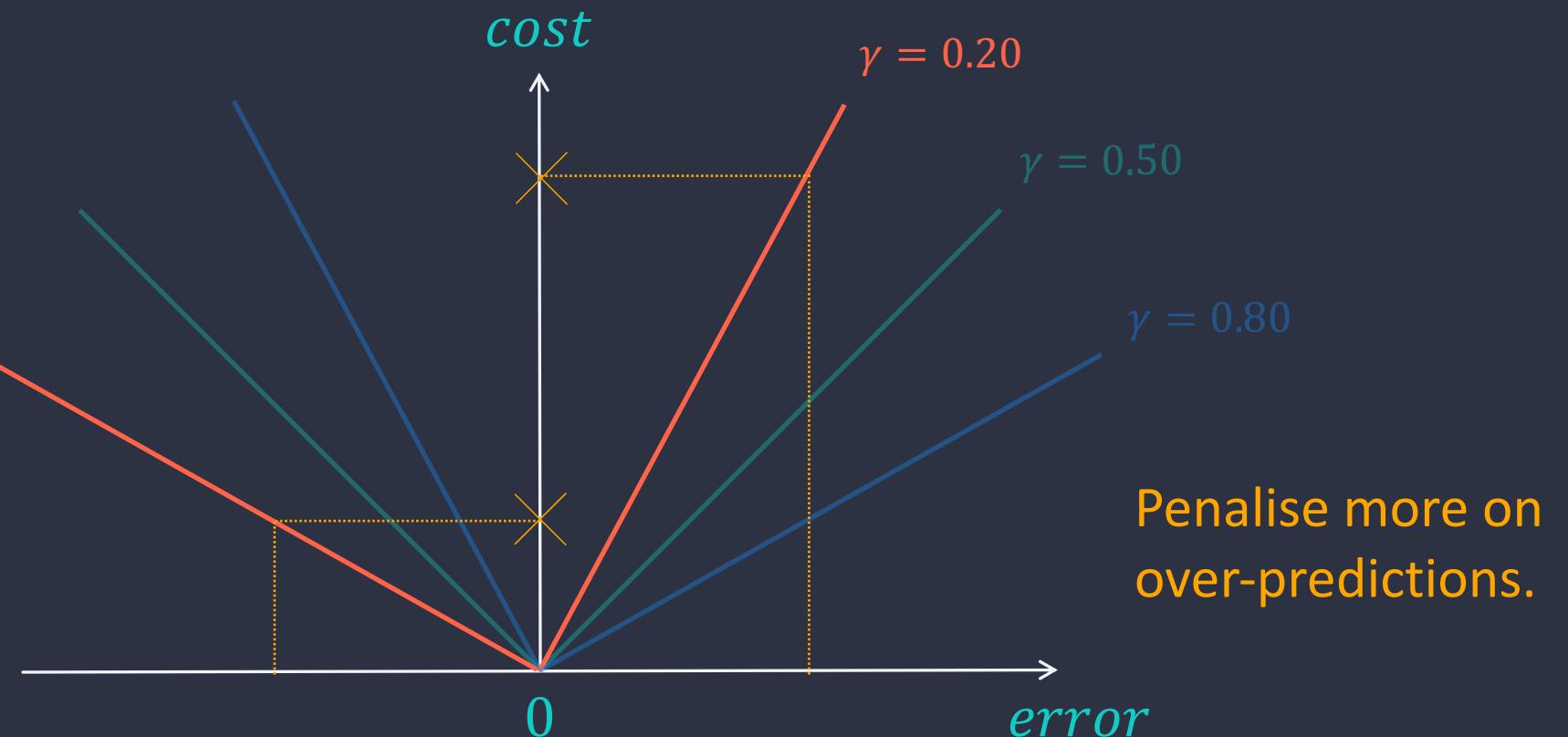
# Quantile Cost Function

$$J = \frac{1}{m} \left( \sum_{y^{(i)} < \hat{y}^{(i)}} (\gamma - 1) |y^{(i)} - \hat{y}^{(i)}| + \sum_{y^{(i)} > \hat{y}^{(i)}} \gamma |y^{(i)} - \hat{y}^{(i)}| \right) \quad \gamma \in (0,1)$$



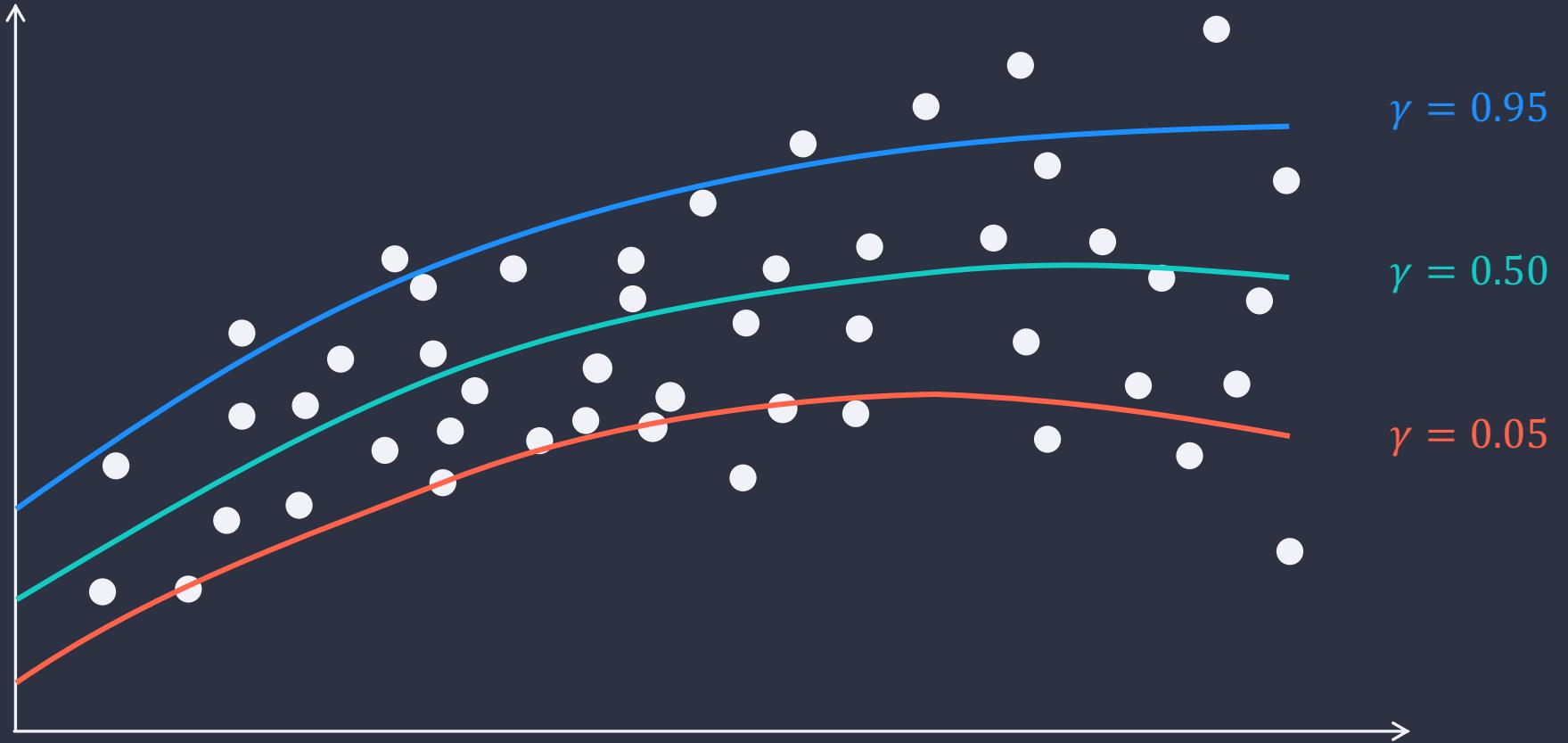
# Quantile Cost Function

$$J = \frac{1}{m} \left( \sum_{y^{(i)} < \hat{y}^{(i)}} (\gamma - 1) |y^{(i)} - \hat{y}^{(i)}| + \sum_{y^{(i)} > \hat{y}^{(i)}} \gamma |y^{(i)} - \hat{y}^{(i)}| \right) \quad \gamma \in (0,1)$$



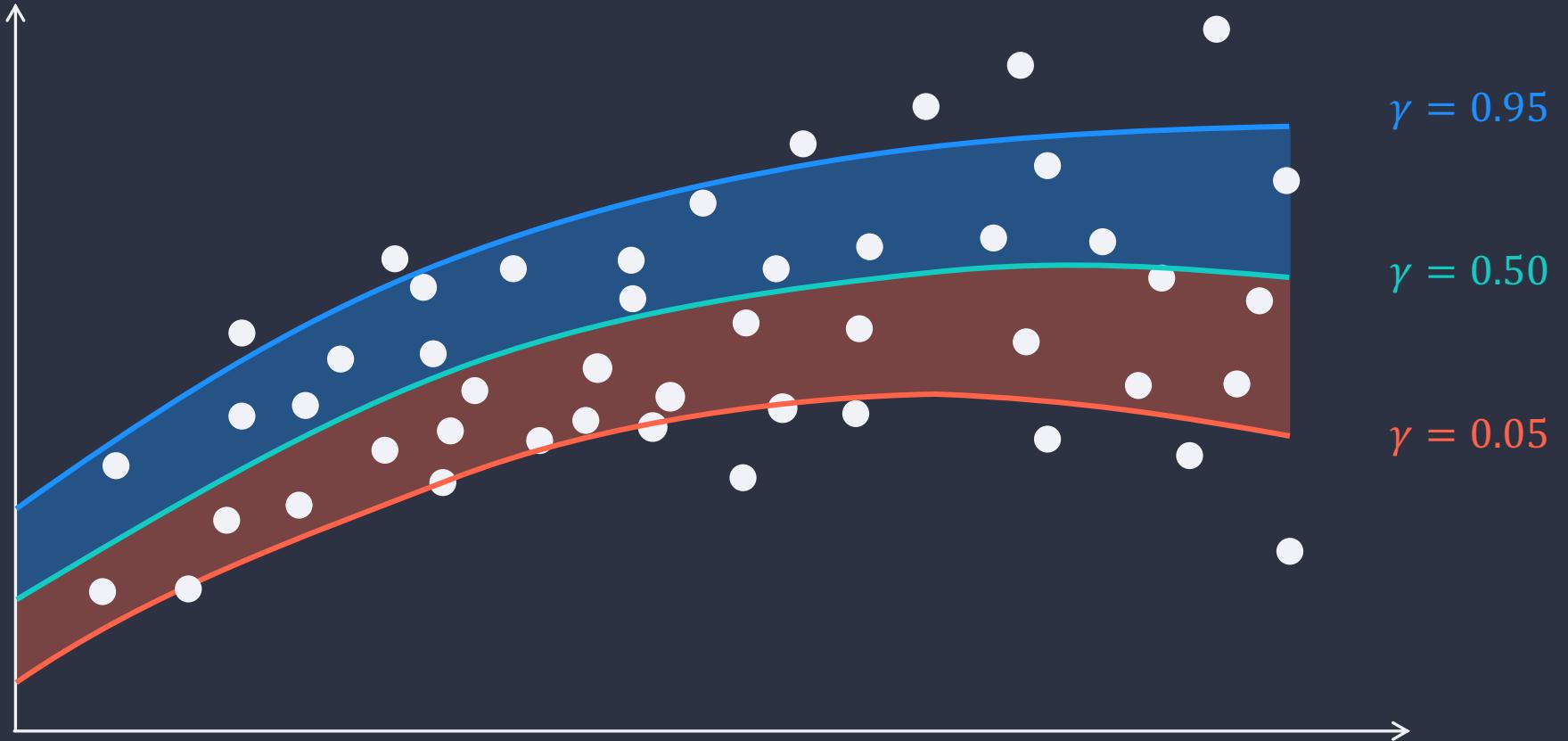
# Quantile Cost Function In Practice

We specify a higher quantile and a lower quantile, i.e., the value for  $\gamma$ , for two cost functions respectively, and we fit the model using both cost functions.

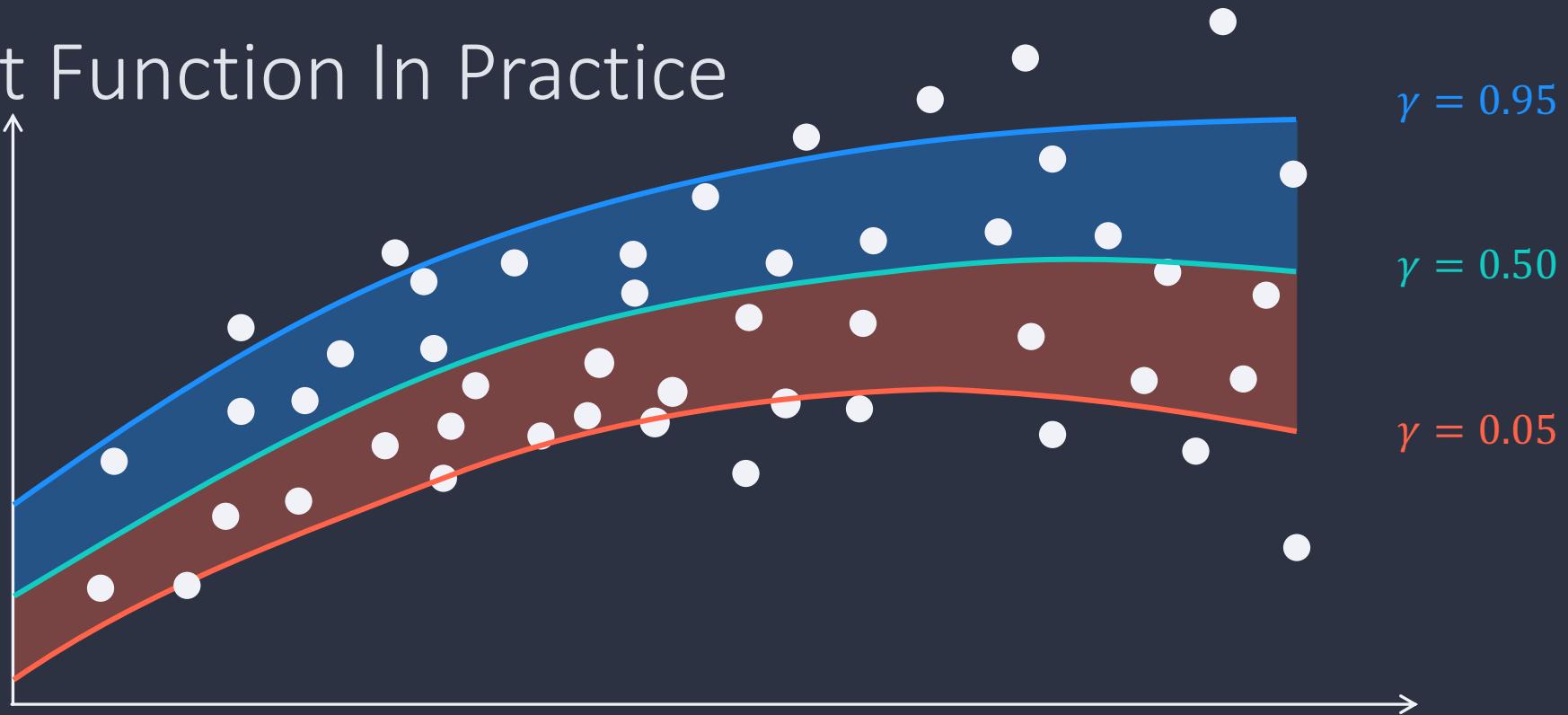


# Quantile Cost Function In Practice

We specify a higher quantile and a lower quantile, i.e., the value for  $\gamma$ , for two cost functions respectively, and we fit the model using both cost functions.



# Quantile Cost Function In Practice



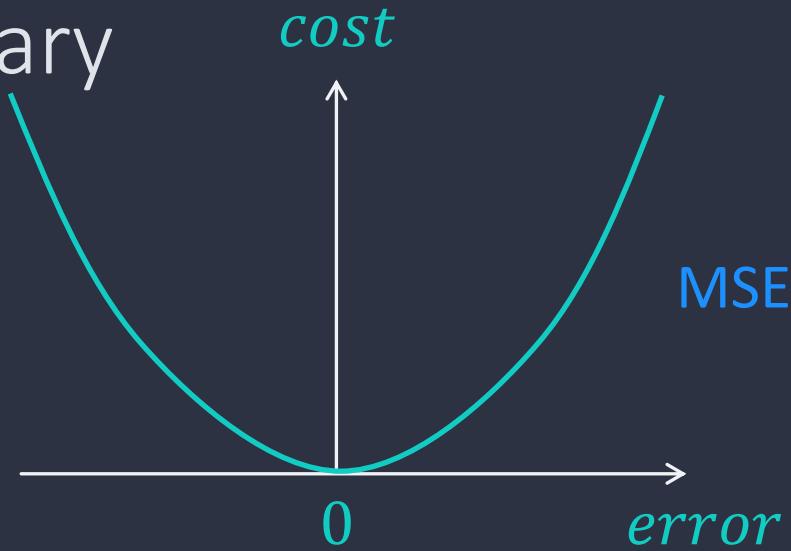
Quantile regression for the 5<sup>th</sup> and 95<sup>th</sup> quantiles tries to find bounds  $y_0(\mathbf{x})$  and  $y_1(\mathbf{x})$ , on the dependent variable  $y$  given predictor variable  $\mathbf{x}$ , such that

$$\mathbb{P}(Y \leq y_0(\mathbf{X})) = 0.05 \quad \mathbb{P}(Y \leq y_1(\mathbf{X})) = 0.95$$

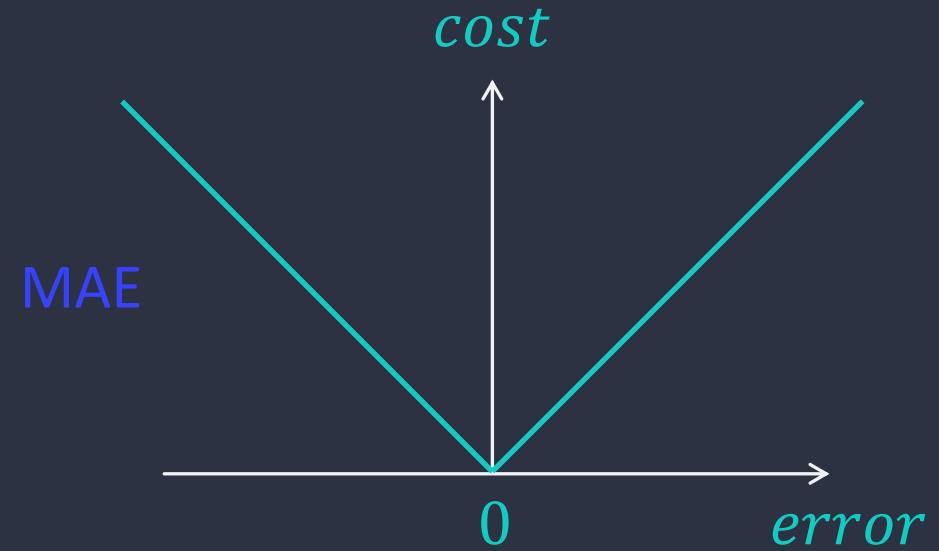
So,  $\mathbb{P}(y_0(\mathbf{X}) \leq Y \leq y_1(\mathbf{X})) = 0.90$  (a 90% prediction interval)

# Summary

# Summary

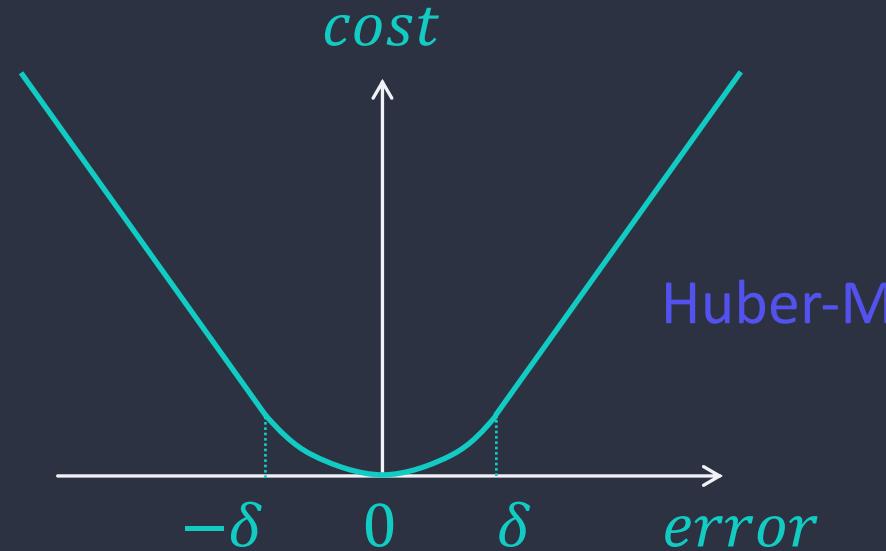


MSE

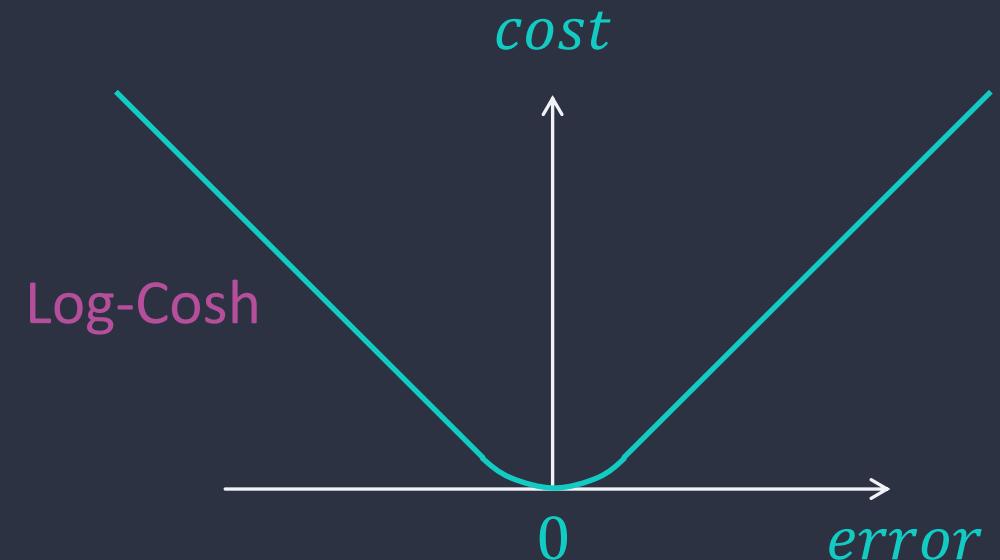


MAE

# Point Prediction

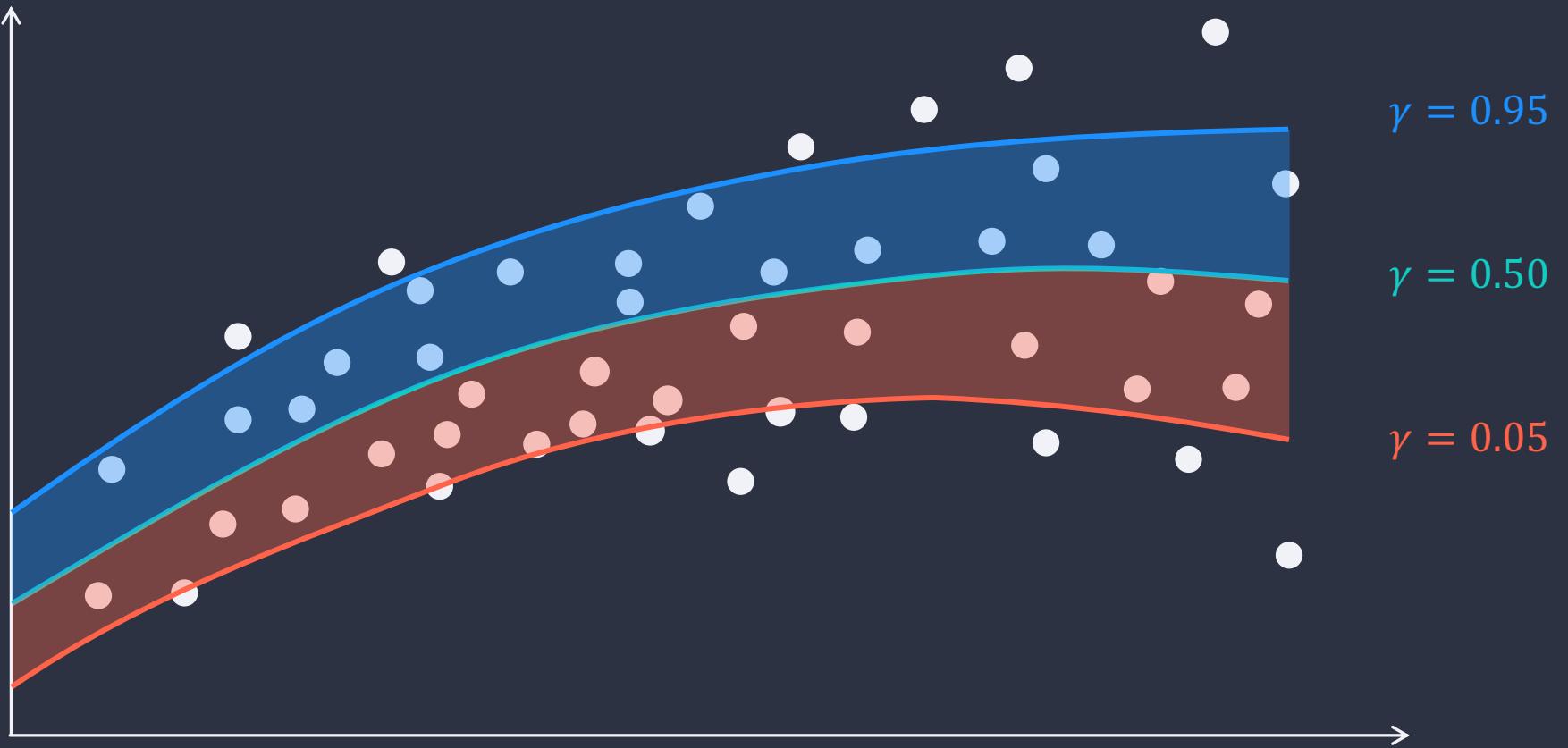


Huber-M



Log-Cosh

# Summary



**COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING**

# Regularisation

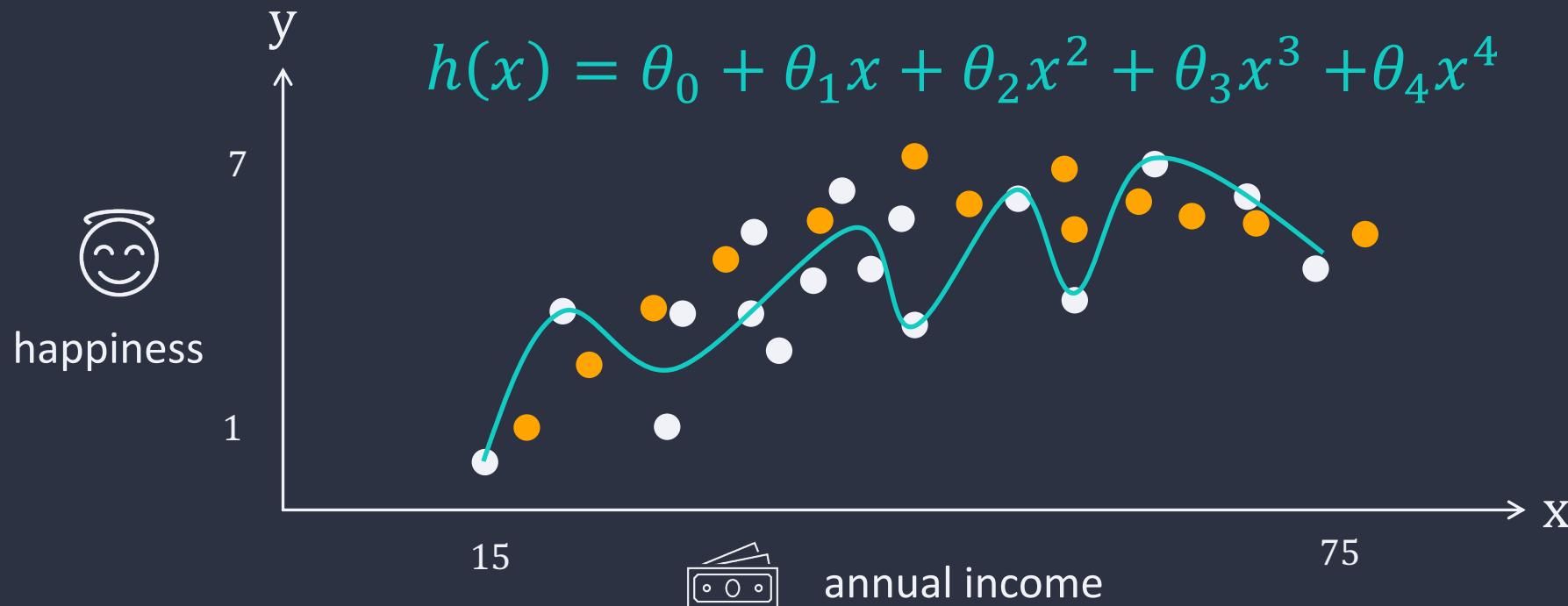
Dr Yang Long

# Previously

## Overfitting problem

It happens when a model learns the detail and noise from the training data to the extent that it negatively impacts the model performance on new data (generalisation).

Fits well on training data but performs poorly on validation/test data.



# Previously

## Tackling Overfitting

### Include more data

- Collect more data
- Data augmentation

### Cross-Validation

- K-fold cross validation
- Leave-one-out cross validation

### Feature selection / reduction

- Manually
- Feature selection algorithms

### Regularisation

- Keep all the features
- Regularise parameters

# Regularisation

# Lecture Overview

1. Intuition
2. Ridge Regression
3. LASSO Regression
4. Ridge versus LASSO

# 1. Intuition

# Intuition

$$h_{\theta}(X) = [\theta_0, \theta_1, \theta_2, \dots, \theta_n] \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

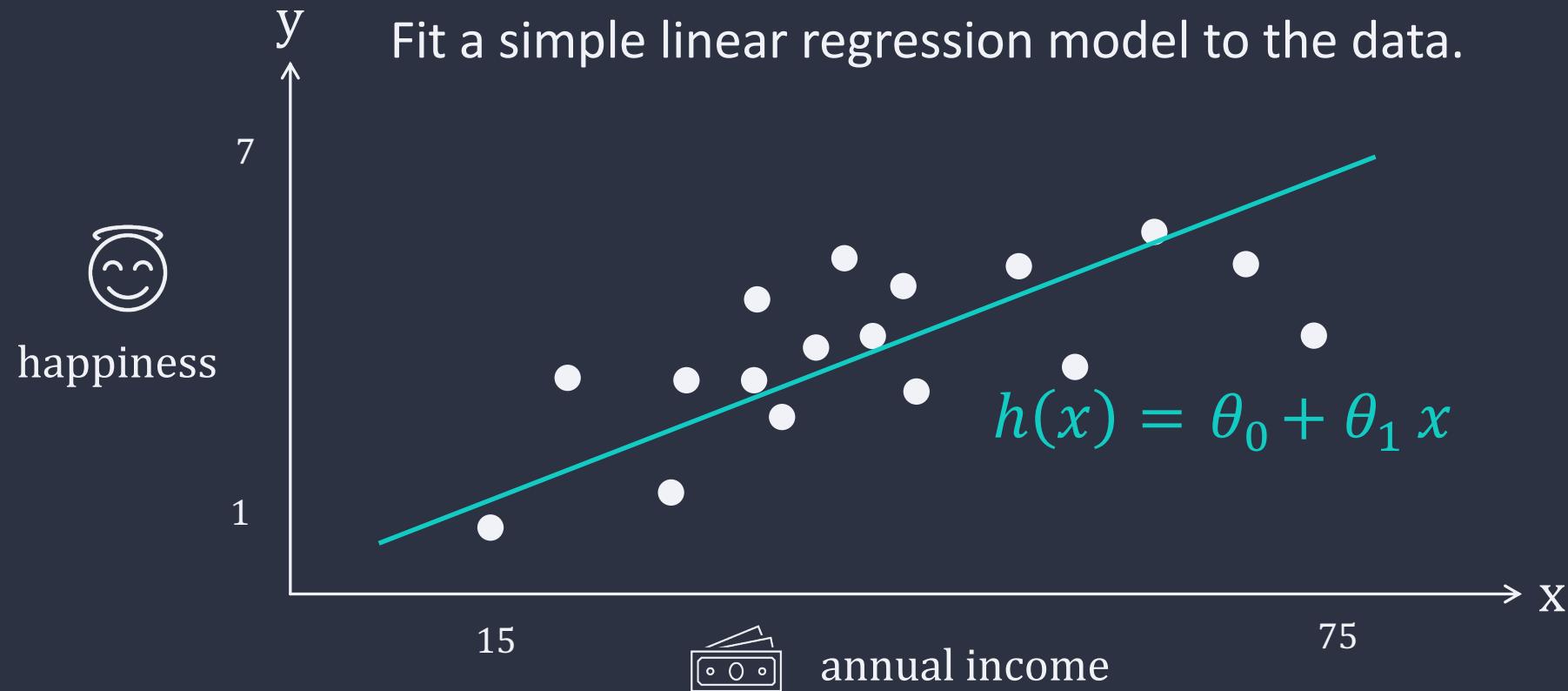
hypothesis      coefficients      features

The smaller the coefficient, the less impact of the corresponding feature.

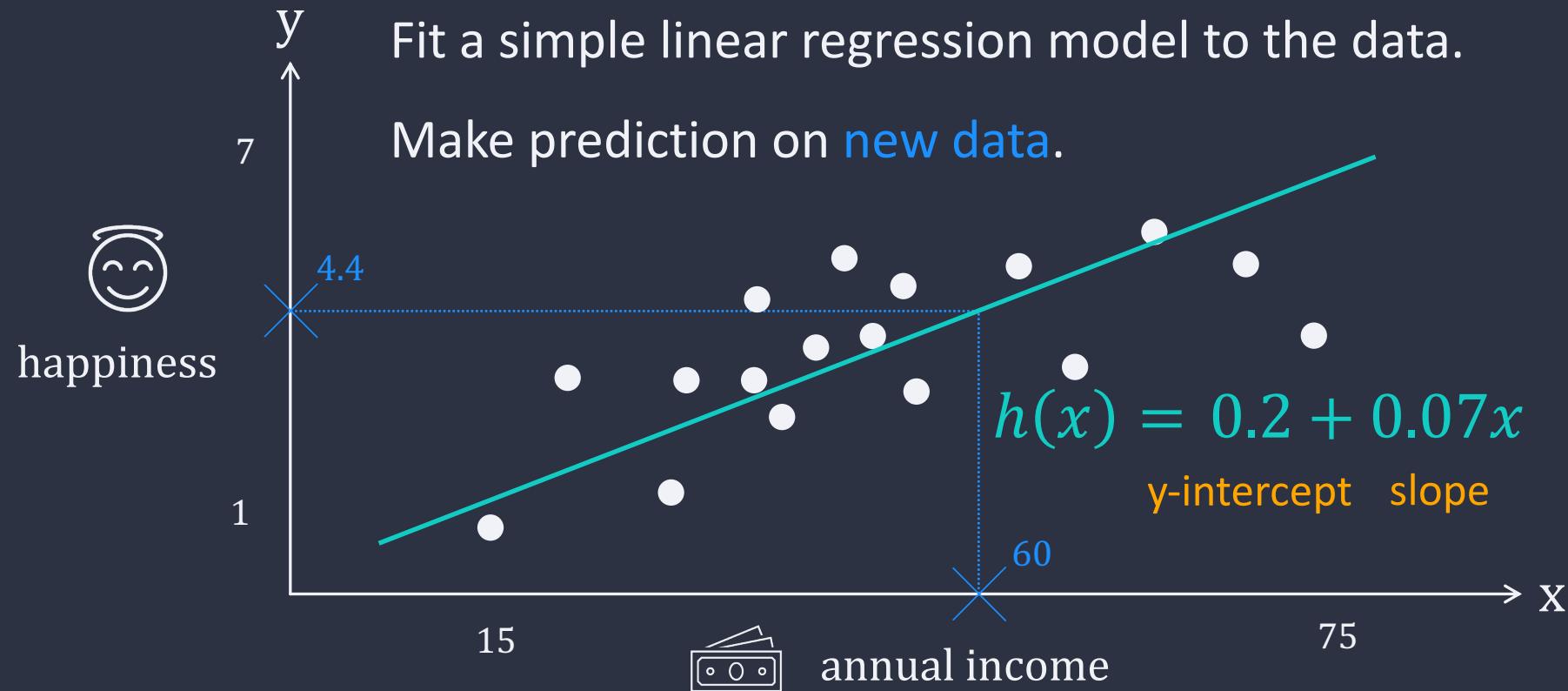
The decrease in coefficients represents the decrease in model flexibility.

To reduce the risk of overfitting, we shall prevent coefficients from rising too high.

# Intuition

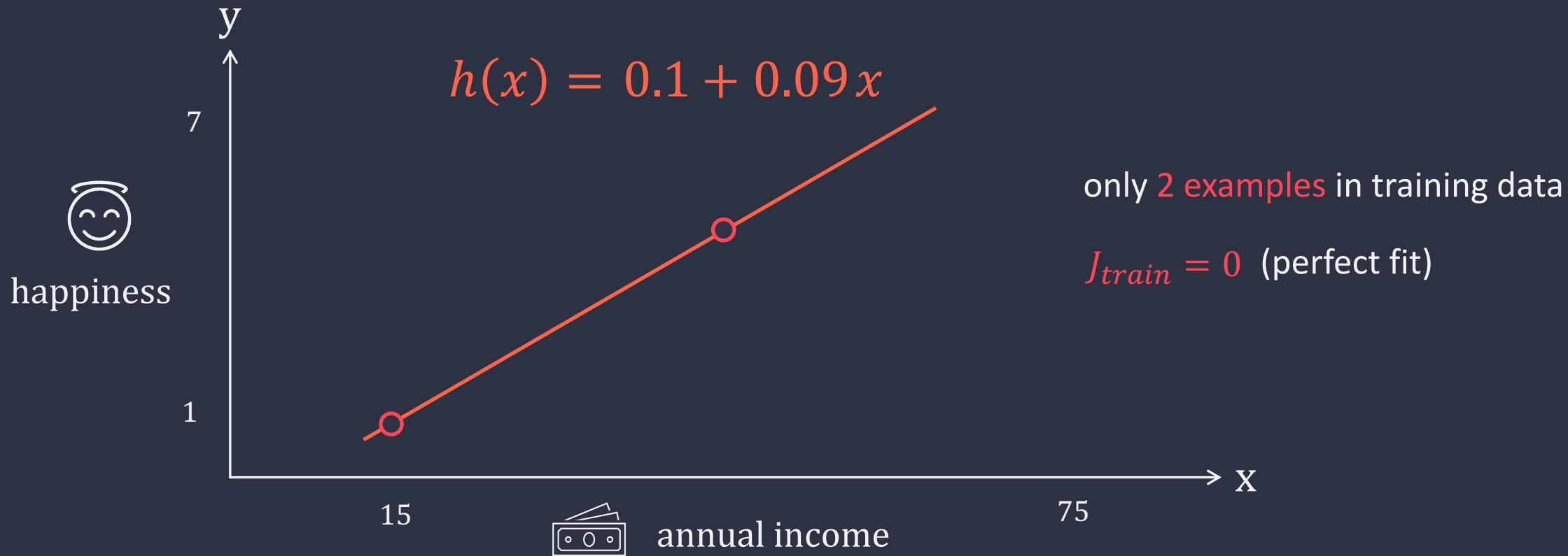


# Intuition

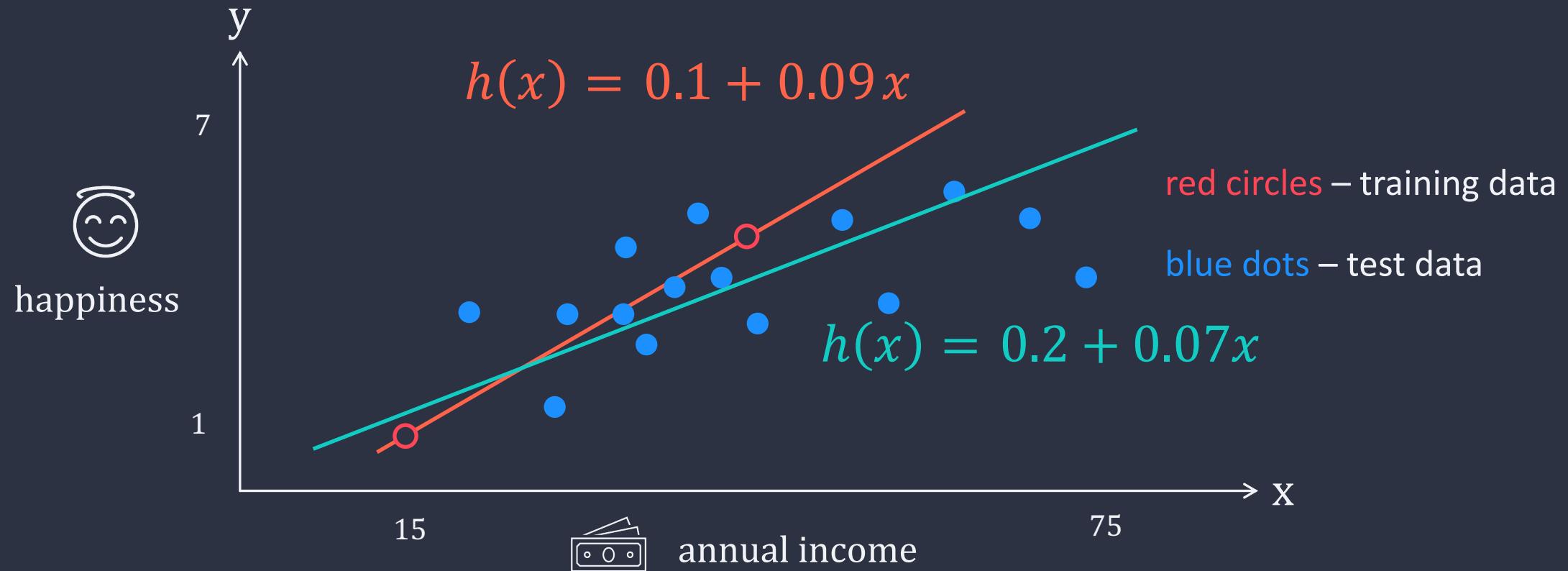


- More training data -> more confidence on model.
- What if there are only 2 examples in training data ?

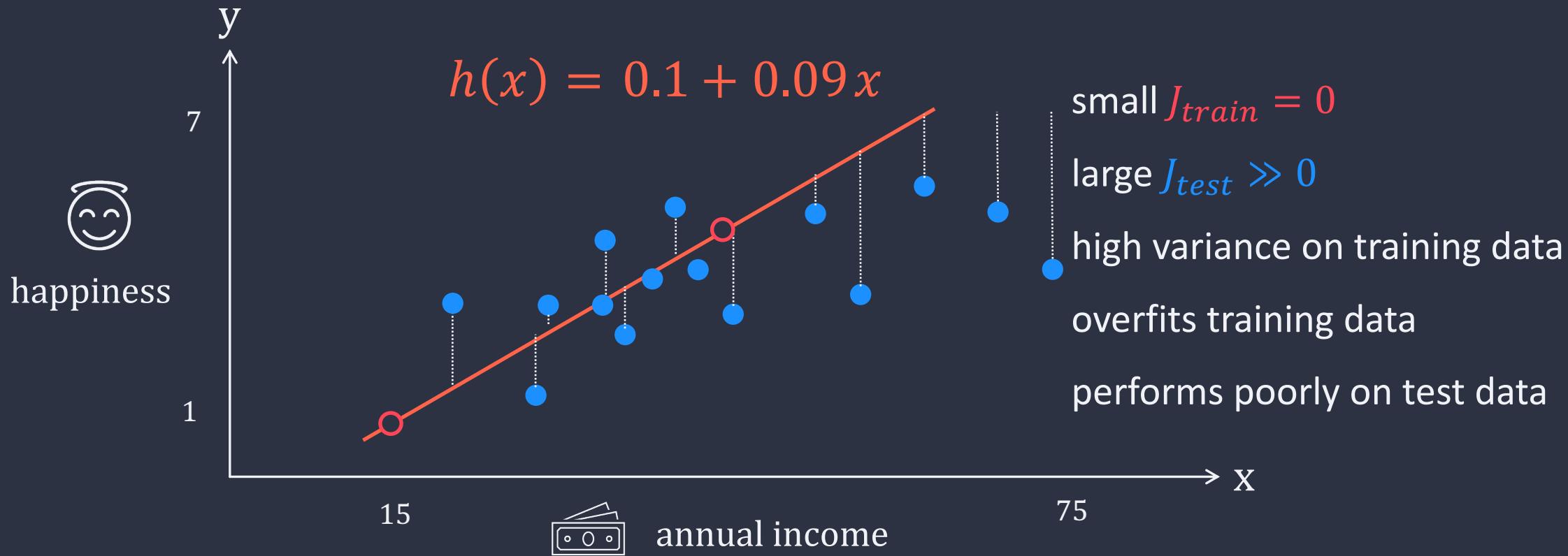
# Intuition



# Intuition

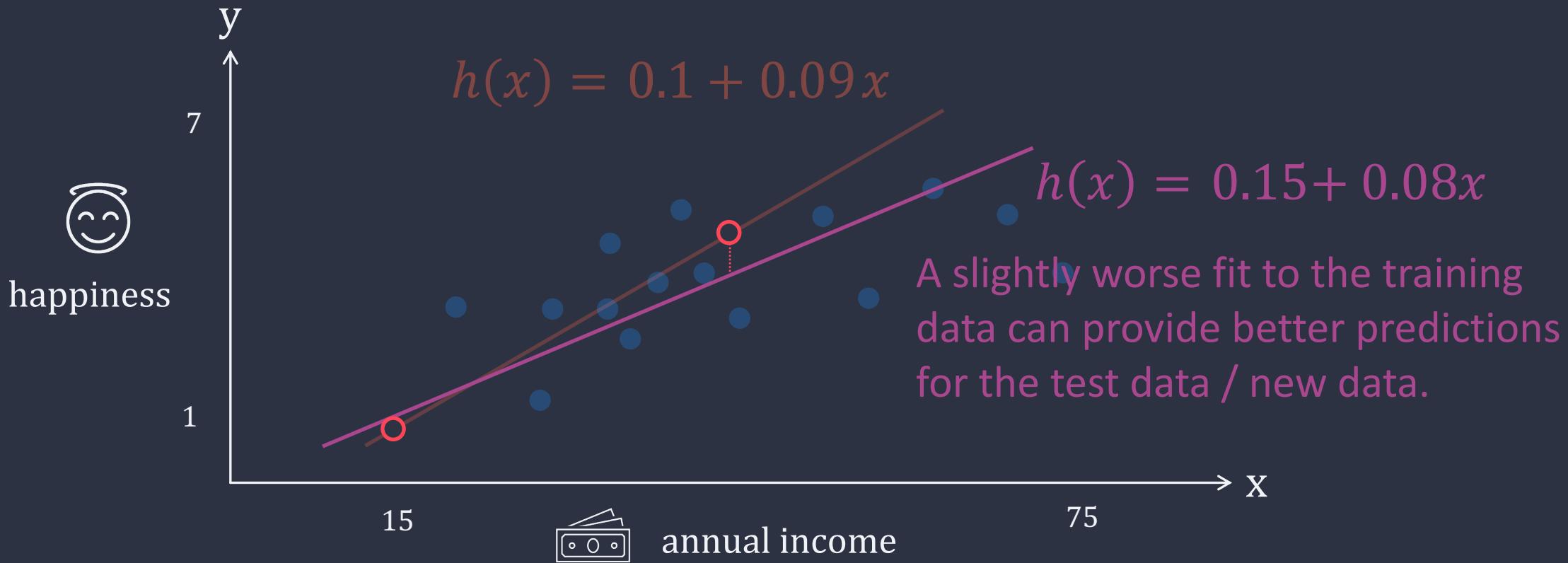


# Intuition

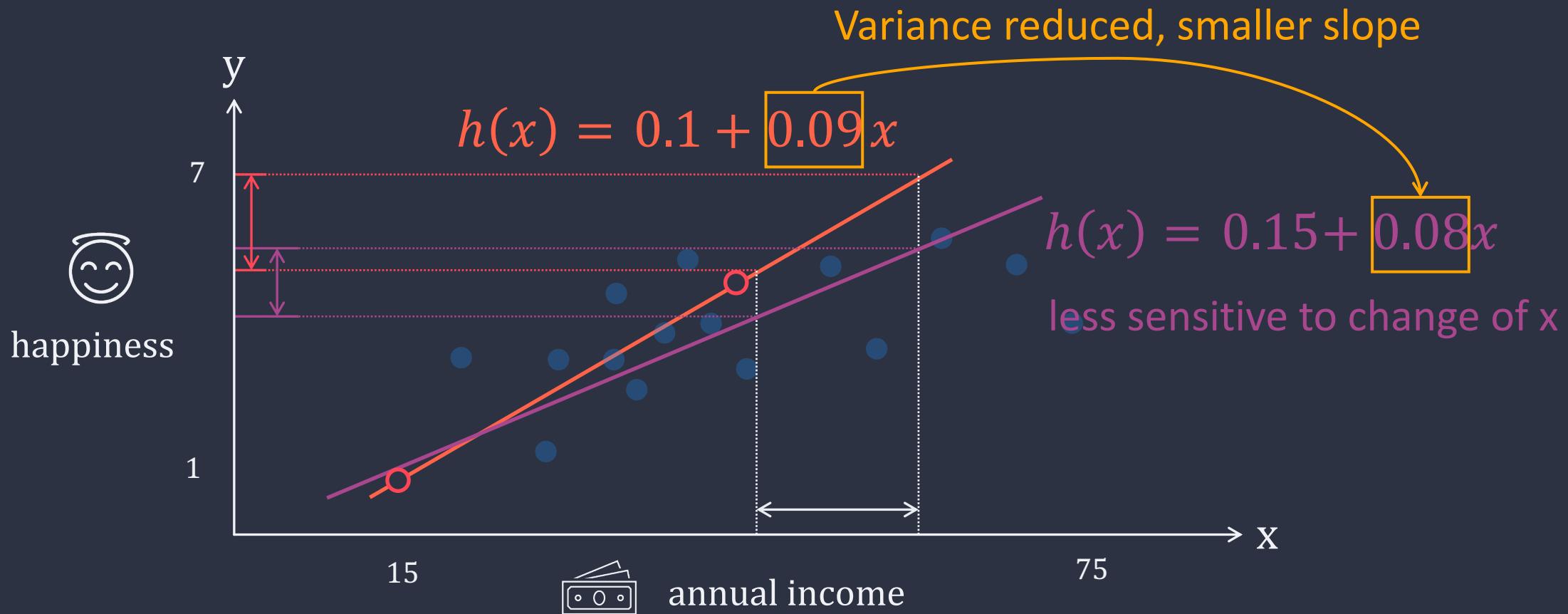


# Intuition

Regularisation: introduces a small amount of bias to build a model that doesn't fit the training data as well, but in turn, that small amount of bias can help drop the variance.



# Intuition



## 2. Ridge Regression

# Ridge Regression

## Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X^{(i)}) - y^{(i)})^2$$

The objective of training a model is to learn from training data to find  $\boldsymbol{\theta}$  that minimises the cost function  $J$ .

# Ridge Regression

## Regularised Cost Function

Shrinkage Penalty

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

The objective of training a model is to learn from training data to find  $\boldsymbol{\theta}$  that minimises the regularised cost function  $J$ .

# Ridge Regression

## Regularised Cost Function

## Shrinkage Penalty

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

$\Sigma$  adds a penalty to the original parameter  $\boldsymbol{\theta}$

$\lambda$  determines how severe the penalty is.

As  $\lambda \rightarrow 0$ , the regularised cost function becomes similar to usual cost function.

Lower constraint (low  $\lambda$ ) on features, model resembles linear regression model.

As  $\lambda \rightarrow \infty$ , risk of underfitting;  $\boldsymbol{\theta}$  to be penalised closer but not exactly to zero.

Do not penalise  $\theta_0$  but all the rest from  $\theta_1$  to  $\theta_n$ .

# Ridge Regression

## Regularised Cost Function

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

$$\frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \quad (j > 0)$$

# Ridge Regression

## Gradient Descent

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X)^{(i)} - y^{(i)}) x_0^{(i)} \right] \quad \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta})$$

$$\theta_1 := \theta_1 - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X)^{(i)} - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1 \right] \quad \frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta})$$

...

$$\theta_n := \theta_n - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(X)^{(i)} - y^{(i)}) x_n^{(i)} + \frac{\lambda}{m} \theta_n \right] \quad \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta})$$

}

# Ridge Regression

## Gradient Descent

Repeat until convergence {

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

$$\theta_1 := \theta_1 - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(X)^{(i)} - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1 \right]$$

...

$$\boxed{\theta_n := \theta_n - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(X)^{(i)} - y^{(i)}) x_n^{(i)} + \frac{\lambda}{m} \theta_n \right]}$$

}

# Ridge Regression

## Gradient Descent

The regularisation term should only be added to the cost function during training. Once the model is trained, we evaluate the model performance using un-regularised performance measure.

$$\theta_n := \theta_n - \alpha \cdot \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(X)^{(i)} - y^{(i)}) x_n^{(i)} + \frac{\lambda}{m} \theta_n \right]$$



$$\theta_n := \boxed{\left(1 - \alpha \cdot \frac{\lambda}{m}\right) \theta_n} - \boxed{\alpha \cdot \frac{1}{m} \sum_{i=1}^m (h_{\theta}(X)^{(i)} - y^{(i)}) x_n^{(i)}}$$

positive slightly smaller than 1      same as without regularisation

The whole equation is meant to shrink  $\theta_n$ .

# Ridge Regression

## Normal Equation

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$
$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

feature values

(feature matrix)

labels

(label vector)

# Ridge Regression

## Normal Equation

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

feature values

(feature matrix)

labels

(label vector)

To  $\min J(\theta)$ : set  $\boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

# Ridge Regression

## Normal Equation

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

feature values

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

labels

$$\text{To } \min J(\theta): \text{ set } \boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X} + \lambda \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix})^{-1} \mathbf{X}^T \mathbf{y} \quad (\lambda > 0)$$

$\in \mathbb{R}^{n+1}$

# Ridge Regression

## Normal Equation

To  $\min J(\theta)$ : set  $\theta = (X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix})^{-1} X^T y$   $(\lambda > 0)$

$$\in \mathbb{R}^{n+1}$$

If  $n > m$  ( $n$ : the number of features;  $m$ : the number of examples)

then  $X^T X$  is non-invertible, so we cannot calculate  $\theta = (X^T X)^{-1} X^T y$ .

However, having the regularisation term, we don't have this issue.

# Ridge Regression

## Coding with scikit-learn

```
from sklearn.datasets import load_boston
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

boston = load_boston()
x, y = boston.data, boston.target
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)
```

# Ridge Regression

## Coding with scikit-learn

```
alphas = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1]
```

```
for a in alphas:  
    model = Ridge(alpha=a, normalize=True).fit(x,y)  
    score = model.score(x, y)  
    pred_y = model.predict(x)  
    mse = mean_squared_error(y, pred_y)  
    print("Alpha:{0:.6f}, R2:{1:.3f}, MSE:{2:.2f}, RMSE:{3:.2f}"  
        .format(a, score, mse, np.sqrt(mse)))
```

```
Alpha:0.000001, R2:0.741, MSE:21.89, RMSE:4.68 Alpha:0.000010, R2:0.741,  
MSE:21.89, RMSE:4.68 Alpha:0.000100, R2:0.741, MSE:21.89, RMSE:4.68  
Alpha:0.001000, R2:0.741, MSE:21.90, RMSE:4.68 Alpha:0.010000, R2:0.740,  
MSE:21.92, RMSE:4.68 Alpha:0.100000, R2:0.732, MSE:22.66, RMSE:4.76  
Alpha:0.500000, R2:0.686, MSE:26.48, RMSE:5.15 Alpha:1.000000, R2:0.635,  
MSE:30.81, RMSE:5.55
```

# Ridge Regression

## Coding with scikit-learn

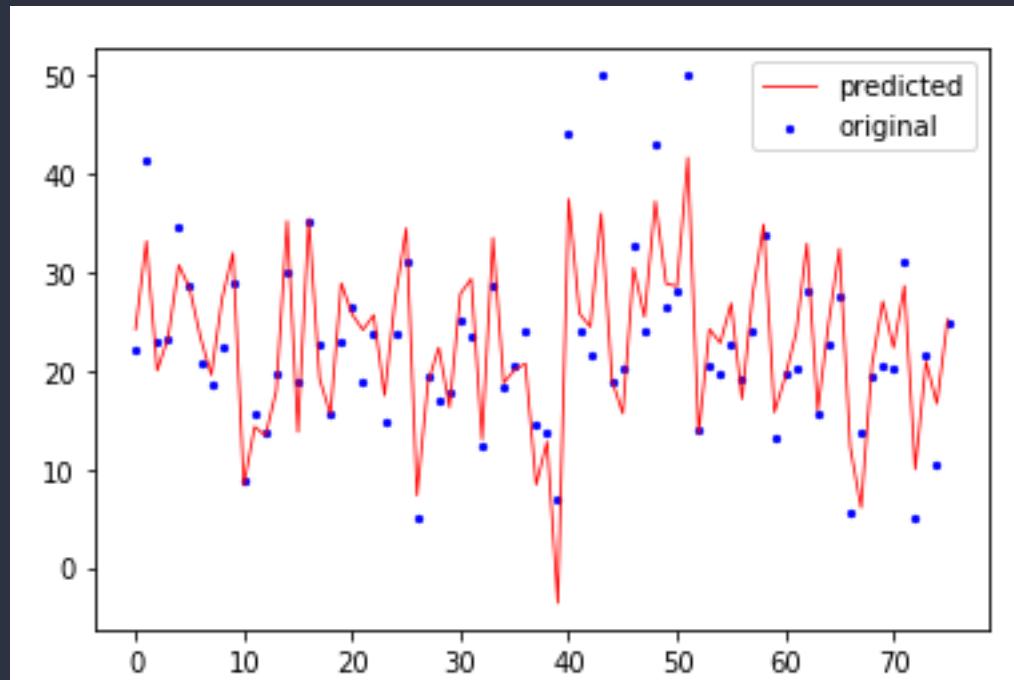
```
ridge_mod=Ridge(alpha=0.01, normalize=True).fit(xtrain,ytrain)
ypred = ridge_mod.predict(xtest)
score = model.score(xtest,ytest)
mse = mean_squared_error(ytest,ypred)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
      .format(score, mse,np.sqrt(mse)))
```

R2:0.725, MSE:17.31, RMSE:4.16

# Ridge Regression

## Coding with scikit-learn

```
x_ax = range(len(xtest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



### 3. LASSO Regression

# LASSO Regression

## Cost Function for Ridge Regression

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

Shrinkage Penalty

# LASSO Regression

## Cost Function

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

Shrinkage Penalty

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

LASSO: Least Absolute Shrinkage and Selection Operator

# LASSO Regularisation

## Cost Function

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

$\lambda$ : the regularisation hyperparameter – regularises the model and penalises larger model parameter  $\boldsymbol{\theta}$ s (feature coefficients), which helps reduce the model variance and the risk of overfitting.

As  $\lambda \rightarrow 0$ , the solution approaches linear regression.

As  $\lambda \rightarrow \infty$ , the solution approaches the global mean.

# LASSO Regularisation

## Cost Function

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

As  $\lambda \rightarrow 0$  , the solution approaches linear regression.

As  $\lambda \rightarrow \infty$  , the solution approaches the global mean.

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \left[ \sum_{i=1}^m (\theta_0 + \sum_{j=1}^n \theta_j x_j^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right]$$

# LASSO Regularisation

## Cost Function

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

As  $\lambda \rightarrow 0$  , the solution approaches linear regression.

As  $\lambda \rightarrow \infty$  , the solution approaches the global mean.

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \left[ \sum_{i=1}^m (\theta_0 + \sum_{j=1}^n \theta_j x_j^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right]$$

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \sum_{j=1}^n \theta_j x_j^{(i)} - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n |\theta_j|$$

least square term

LASSO regularisation term

# LASSO Regularisation

## Coordinate Descent

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \sum_{j=1}^n \theta_j x_j^{(i)} - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n |\theta_j|$$

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \left[ \sum_{j=1}^n \theta_j x_j^{(i)} \right] - y^{(i)}) x_k^{(i)} + \dots$$

$$= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} + \boxed{\theta_k x_k^{(i)}} - y^{(i)}) x_k^{(i)} + \dots$$

$$= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} - y^{(i)}) x_k^{(i)} + \boxed{\frac{1}{m} \theta_k \sum_{i=1}^m x_k^{(i)2}}$$

$$\frac{\partial}{\partial \theta_k} + \left\{ \quad \right.$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} - y^{(i)}) x_k^{(i)} + \frac{1}{m} \theta_k \sum_{i=1}^m x_k^{(i)2} + \begin{cases} \frac{1}{2m} \lambda & \theta_k > 0 \\ (-\frac{1}{2m} \lambda, \frac{1}{2m} \lambda) & \theta_k = 0 \\ -\frac{1}{2m} \lambda & \theta_k < 0 \end{cases}$$

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m} (c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \\ (\frac{1}{m} \left( c_k - \frac{\lambda}{2} \right), \frac{1}{m} \left( c_k + \frac{\lambda}{2} \right)) & \theta_k = 0 \\ \frac{1}{m} (c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \end{cases}$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} - y^{(i)}) x_k^{(i)} + \frac{1}{m} \theta_k \sum_{i=1}^m x_k^{(i)2} + \begin{cases} \frac{1}{2m} \lambda & \theta_k > 0 \\ (-\frac{1}{2m} \lambda, \frac{1}{2m} \lambda) & \theta_k = 0 \\ -\frac{1}{2m} \lambda & \theta_k < 0 \end{cases}$$

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m} (c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \\ (\frac{1}{m} (c_k - \frac{\lambda}{2}), \frac{1}{m} (c_k + \frac{\lambda}{2})) & \theta_k = 0 \\ \frac{1}{m} (c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \end{cases} \rightarrow \boxed{\begin{aligned} \frac{1}{m} (c_k + \theta_k a_k) + \frac{\lambda}{2m} &= 0 \\ \theta_k &= -\frac{2c_k + \lambda}{2a_k} \\ (\theta_k > 0, a_k > 0) &\rightarrow c_k < -\frac{1}{2}\lambda \end{aligned}}$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} - y^{(i)}) x_k^{(i)} + \frac{1}{m} \theta_k \sum_{i=1}^m x_k^{(i)2} + \begin{cases} \frac{1}{2m} \lambda & \theta_k > 0 \\ (-\frac{1}{2m} \lambda, \frac{1}{2m} \lambda) & \theta_k = 0 \\ -\frac{1}{2m} \lambda & \theta_k < 0 \end{cases}$$

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m} (c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \\ (\frac{1}{m} (c_k - \frac{\lambda}{2}), \frac{1}{m} (c_k + \frac{\lambda}{2})) & \theta_k = 0 \\ \frac{1}{m} (c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \end{cases} \rightarrow \theta_k = -\frac{2c_k + \lambda}{2a_k} \quad \text{if } c_k < -\frac{1}{2}\lambda$$

$$\boxed{\frac{1}{m} (c_k + \theta_k a_k) - \frac{\lambda}{2m} = 0}$$

$$\theta_k = \frac{2c_k - \lambda}{2a_k}$$

$$(\theta_k > 0, a_k > 0) \rightarrow c_k > \frac{1}{2}\lambda$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \sum_{j \neq k} \theta_j x_j^{(i)} - y^{(i)}) x_k^{(i)} + \frac{1}{m} \theta_k \sum_{i=1}^m x_k^{(i)2} + \begin{cases} \frac{1}{2m} \lambda & \theta_k > 0 \\ (-\frac{1}{2m} \lambda, \frac{1}{2m} \lambda) & \theta_k = 0 \\ -\frac{1}{2m} \lambda & \theta_k < 0 \end{cases}$$

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m} (c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \\ (\frac{1}{m} (c_k - \frac{\lambda}{2}), \frac{1}{m} (c_k + \frac{\lambda}{2})) & \theta_k = 0 \\ \frac{1}{m} (c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \end{cases} \rightarrow \begin{cases} \theta_k = -\frac{2c_k + \lambda}{2a_k} & \text{if } c_k < -\frac{1}{2}\lambda \\ \theta_k = 0 & \text{if } -\frac{1}{2}\lambda \leq c_k \leq \frac{1}{2}\lambda \\ \theta_k = \frac{2c_k - \lambda}{2a_k} & \text{if } c_k > \frac{1}{2}\lambda \end{cases}$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m}(c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \\ (\frac{1}{m}\left(c_k - \frac{\lambda}{2}\right), \frac{1}{m}\left(c_k + \frac{\lambda}{2}\right)) & \theta_k = 0 \\ \frac{1}{m}(c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \end{cases} \rightarrow \begin{cases} \theta_k = -\frac{2c_k + \lambda}{2a_k} & \text{if } c_k < -\frac{1}{2}\lambda \\ \theta_k = 0 & \text{if } -\frac{1}{2}\lambda \leq c_k \leq \frac{1}{2}\lambda \\ \theta_k = \frac{2c_k - \lambda}{2a_k} & \text{if } c_k > \frac{1}{2}\lambda \end{cases}$$

Update  $\theta_k$  ( $k = 1, 2, \dots, n$ )

$$\theta_k := \begin{cases} -\frac{2c_k + \lambda}{2a_k} & c_k < -\frac{1}{2}\lambda \\ 0 & -\frac{1}{2}\lambda \leq c_k \leq \frac{1}{2}\lambda \\ \frac{2c_k - \lambda}{2a_k} & c_k > \frac{1}{2}\lambda \end{cases}$$

# LASSO Regularisation Coordinate Descent

$$\frac{\partial}{\partial \theta_k} J(\boldsymbol{\theta}) = \begin{cases} \frac{1}{m}(c_k + \theta_k a_k) + \frac{\lambda}{2m} & \theta_k > 0 \rightarrow \theta_k = -\frac{2c_k + \lambda}{2a_k} \quad \text{if } c_k < -\frac{1}{2}\lambda \\ (\frac{1}{m}(c_k - \frac{\lambda}{2}), \frac{1}{m}(c_k + \frac{\lambda}{2})) & \theta_k = 0 \rightarrow \theta_k = 0 \quad \text{if } -\frac{1}{2}\lambda \leq c_k \leq \frac{1}{2}\lambda \\ \frac{1}{m}(c_k + \theta_k a_k) - \frac{\lambda}{2m} & \theta_k < 0 \rightarrow \theta_k = \frac{2c_k - \lambda}{2a_k} \quad \text{if } c_k > \frac{1}{2}\lambda \end{cases}$$

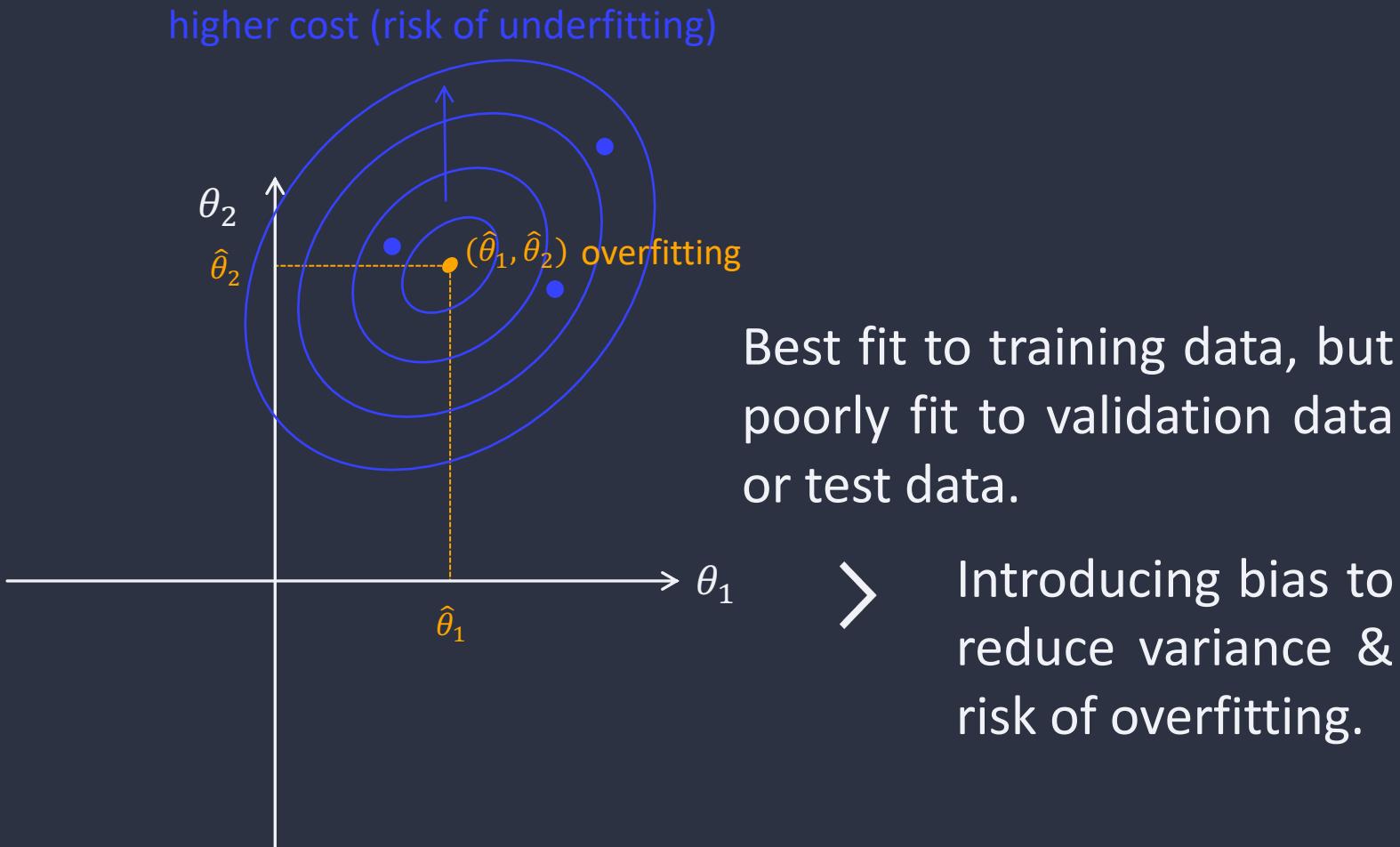
Update  $\theta_k$  ( $k = 1, 2, \dots, n$ )

$$\theta_k := \begin{cases} -\frac{2c_k + \lambda}{2a_k} & c_k < -\frac{1}{2}\lambda \\ 0 & -\frac{1}{2}\lambda \leq c_k \leq \frac{1}{2}\lambda \\ \frac{2c_k - \lambda}{2a_k} & c_k > \frac{1}{2}\lambda \end{cases}$$

Completely eliminates coefficient  
of least important feature  $x_k$  → Feature Selection

## 4. Ridge versus LASSO

Regression task: to find the pair of parameters that minimises the cost function J.

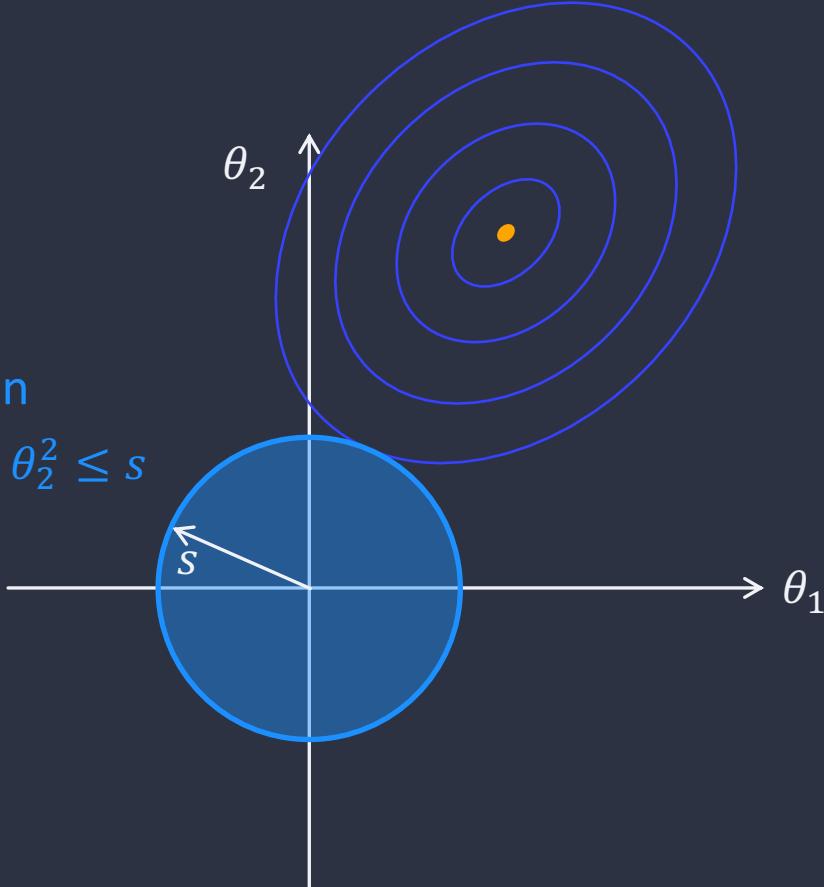


# Ridge Regression

Introducing  $\mathcal{L}_2$  norm regularisation term  $\lambda \sum_{j=1}^n \theta_j^2 >$  Putting a constraint on how far away the optimal parameter pair can be from the minima.

parameter space under regularisation

( $s$  is determined by  $\lambda \in (0, \infty)$ )  $\theta_1^2 + \theta_2^2 \leq s$

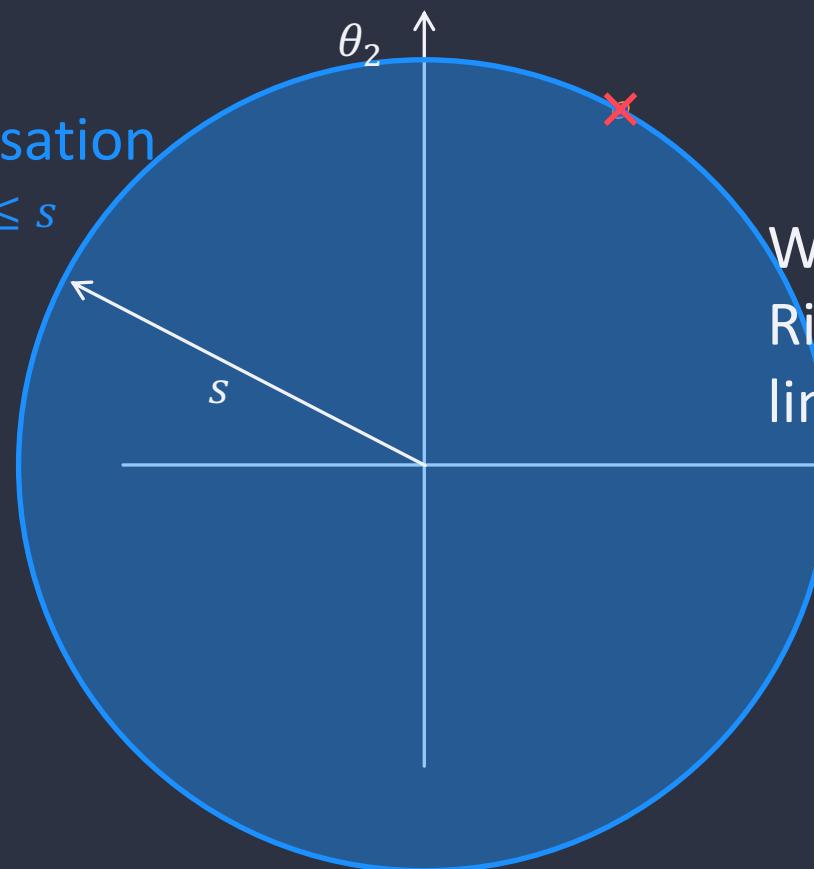


# Ridge Regression

Introducing  $\mathcal{L}_2$  norm regularisation term  $\lambda \sum_{j=1}^n \theta_j^2 >$  Putting a constraint on how far away the optimal parameter pair can be from the minima.

parameter space under regularisation

$$\theta_1^2 + \theta_2^2 \leq s$$

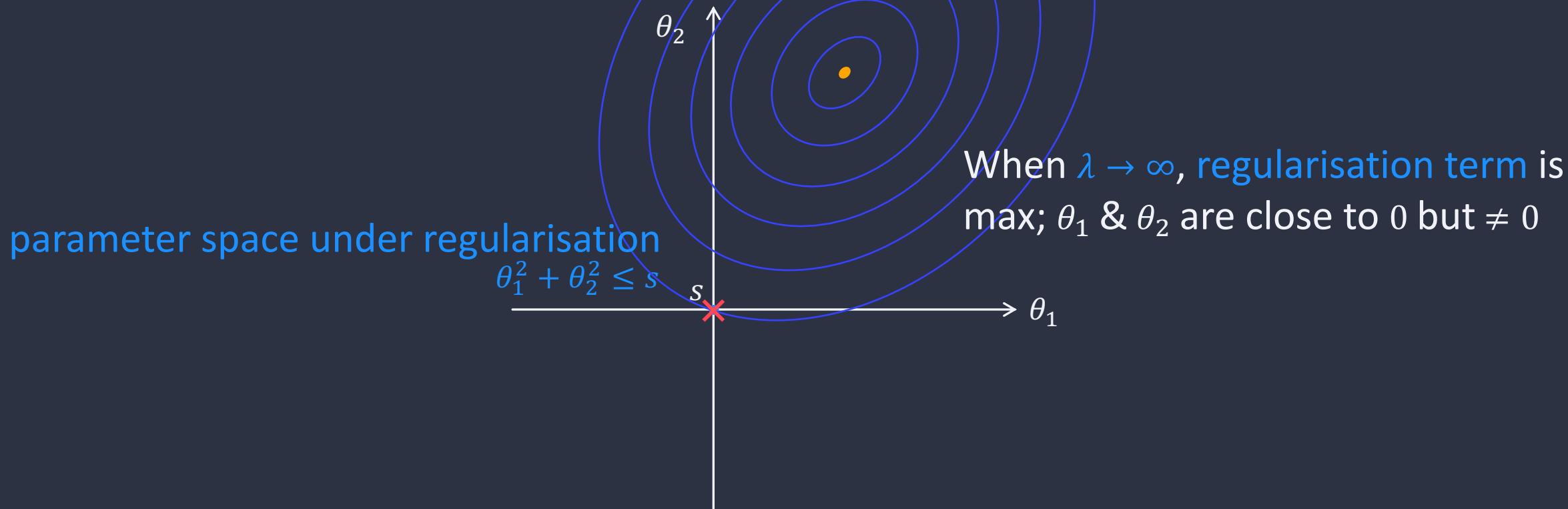


When  $\lambda = 0$ , regularisation term is 0.  
Ridge regression becomes ordinary linear regression.

# Ridge Regression

Introducing  $\mathcal{L}_2$  norm regularisation term  $\lambda \sum_{j=1}^n \theta_j^2 >$

Putting a constraint on how far away the optimal parameter pair can be from the minima.



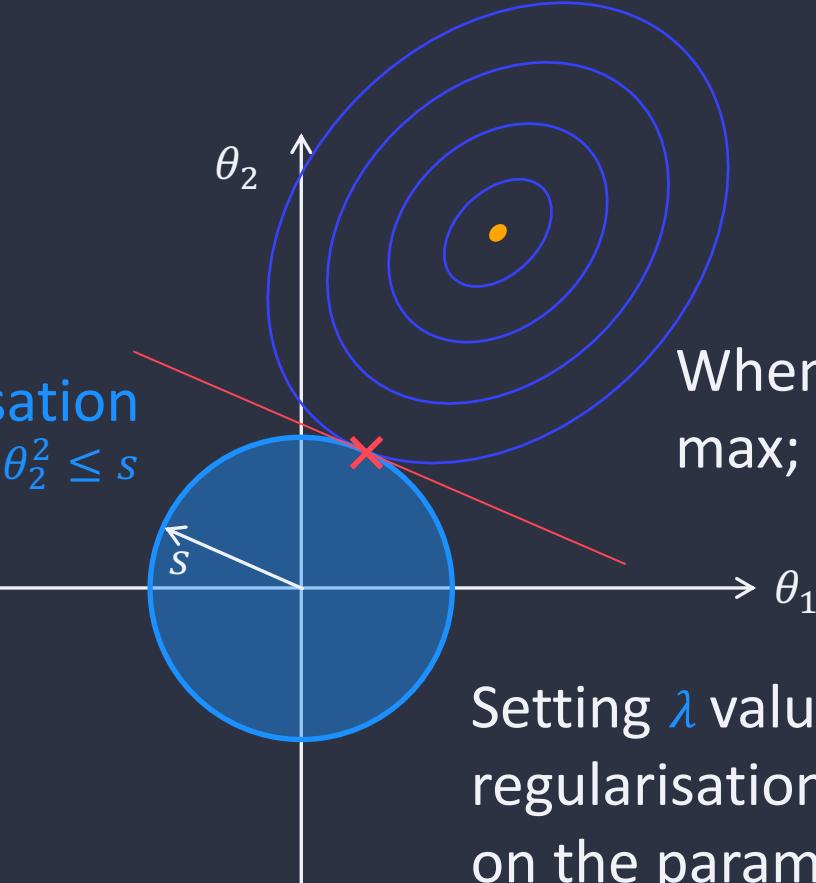
When  $\lambda \rightarrow \infty$ , regularisation term is max;  $\theta_1$  &  $\theta_2$  are close to 0 but  $\neq 0$

# Ridge Regression

Introducing  $\mathcal{L}_2$  norm regularisation term  $\lambda \sum_{j=1}^n \theta_j^2 >$  Putting a constraint on how far away the optimal parameter pair can be from the minima.

parameter space under regularisation

$$\theta_1^2 + \theta_2^2 \leq s$$

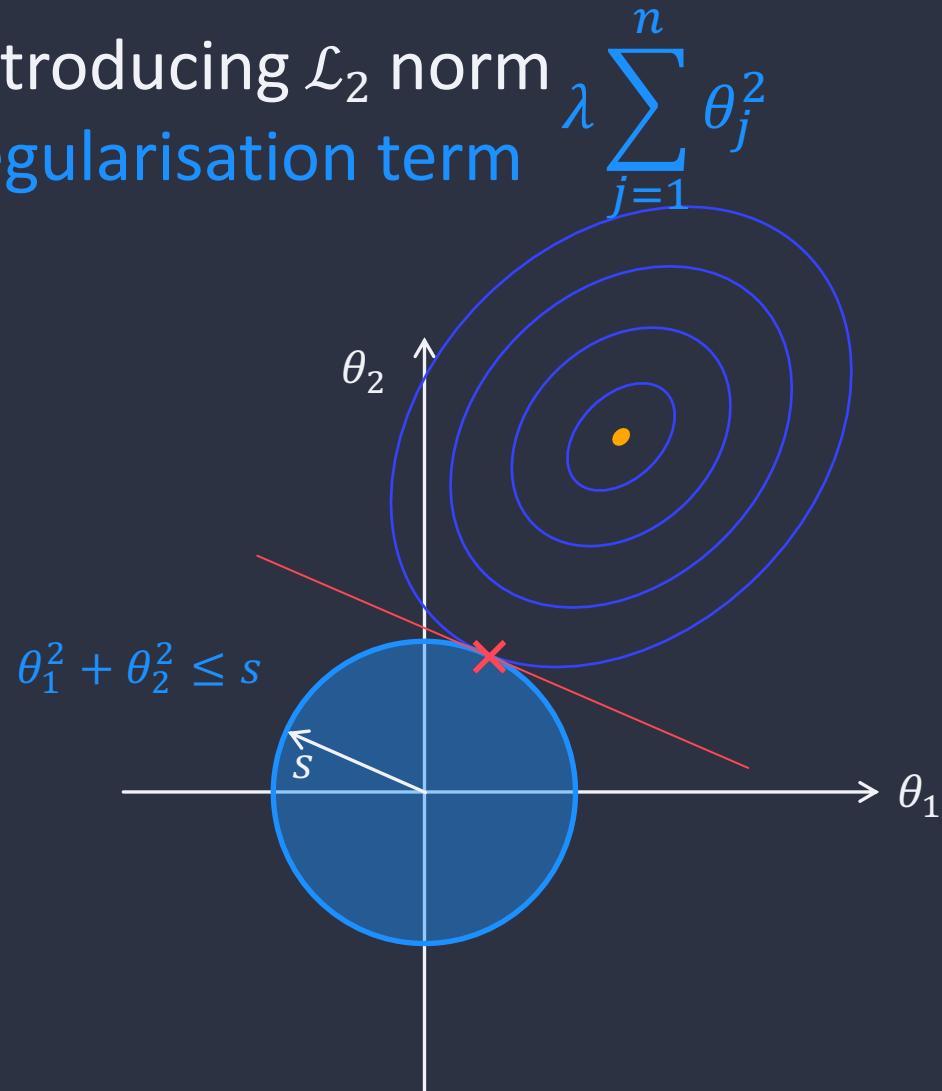


When  $\lambda \rightarrow \infty$ , regularisation term is max;  $\theta_1$  &  $\theta_2$  are close to 0 but  $\neq 0$

Setting  $\lambda$  value, we set the effectiveness of regularisation term, which is the constraint on the parameter space, and it determines where this intersection point can be.

# Ridge Regression

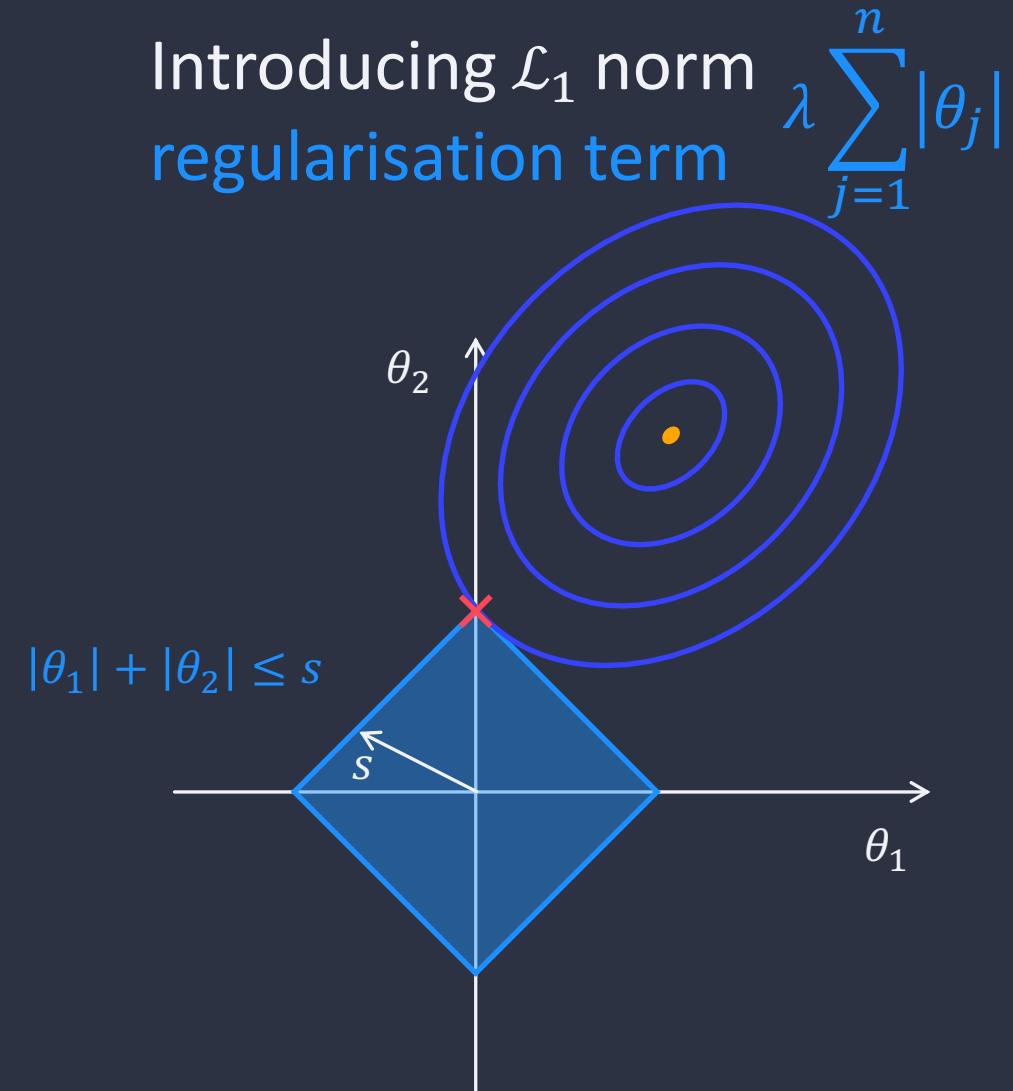
Introducing  $\mathcal{L}_2$  norm regularisation term



Reduce parameters close to zero

# LASSO Regression

Introducing  $\mathcal{L}_1$  norm regularisation term



Reduce parameters to zero -> feature selection  
Useful when there are too many features.

# Summary

# Summary

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

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

Next Lecture

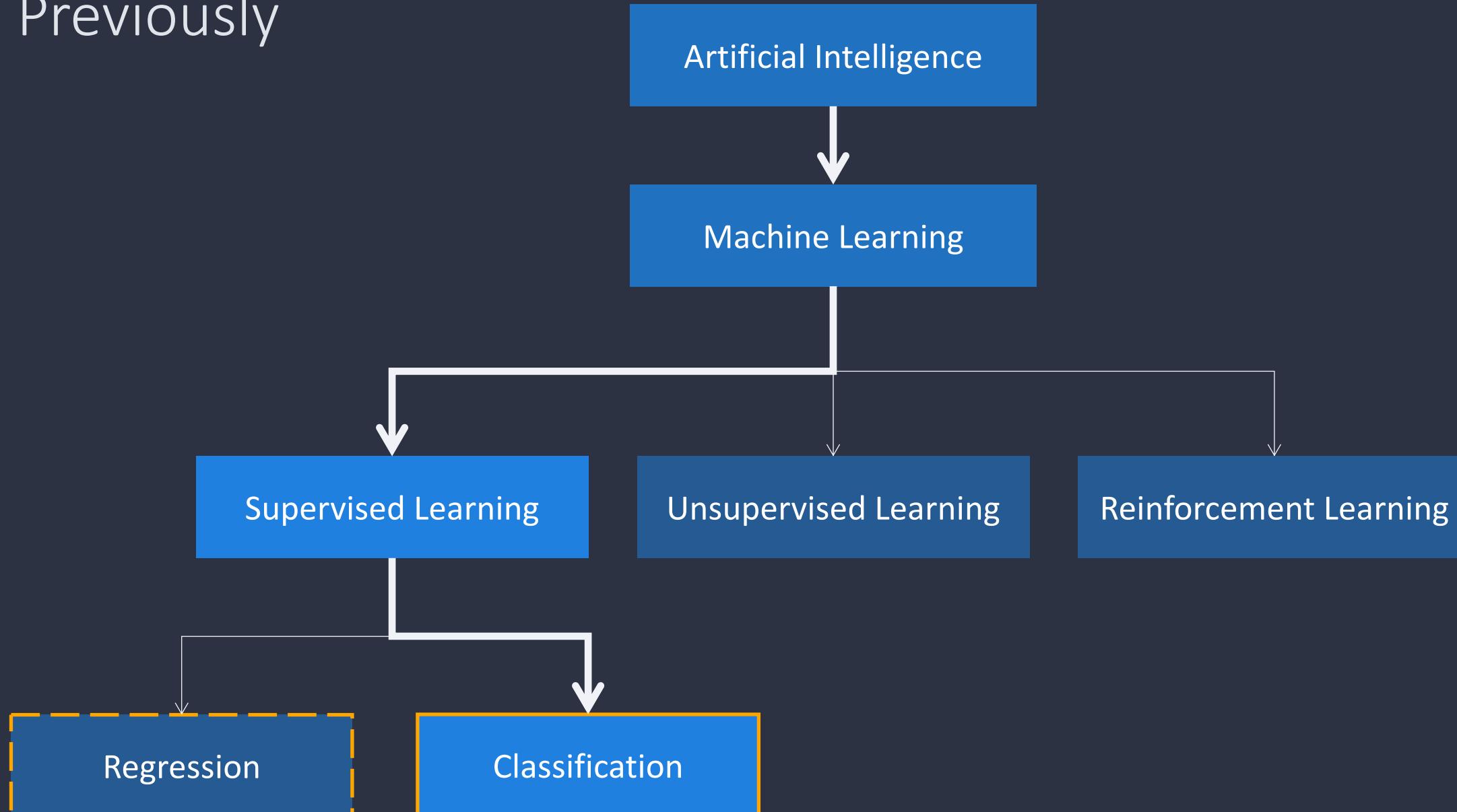
Binary Classification Instance Based Learning

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Binary Classification & Instance-based Learning

Dr Yang Long

# Previously



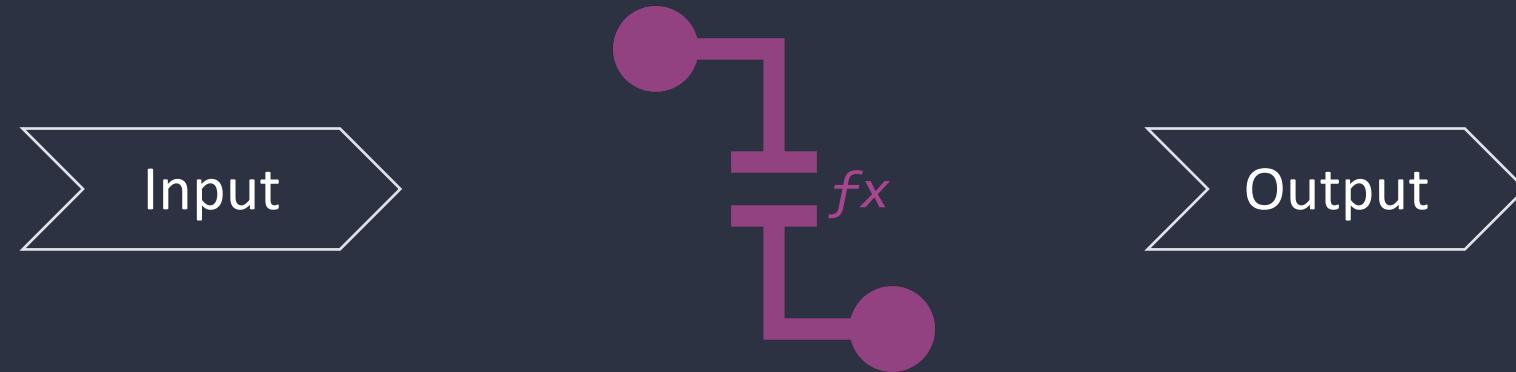
# Lecture Overview

1. Introduction to Classification
2. Binary Classification
3. Binary Classifier for Multiclass Classification
4. Instance-based Learning
5. k Nearest Neighbours

# Introduction to Classification

## A type of Supervised Learning

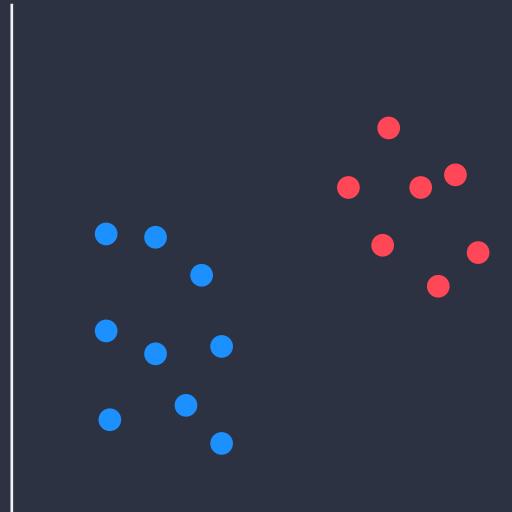
- To learn an approximation function, which is inferred from labelled training data containing a set of training instances.



- The approximation function maps from input variables to categorical and discrete output variables (categories / labels).

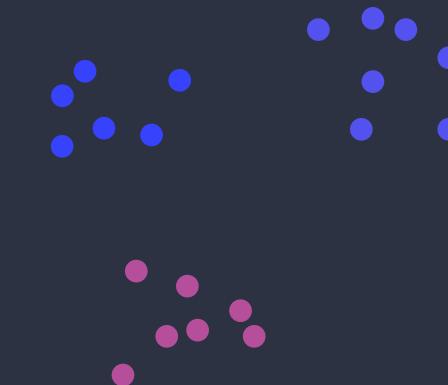
# Introduction to Classification

Output



Binary Classification

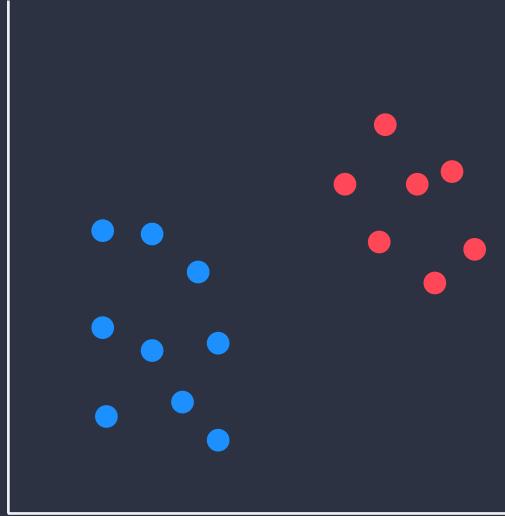
Two categories / labels



Multiclass Classification

More than two categories / labels

# Introduction to Classification



## Binary Classification

Email: spam / not spam

Transaction: fraudulent / not fraudulent

Gold trading: sell / buy

COVID-19 test: positive / negative



## Multiclass Classification

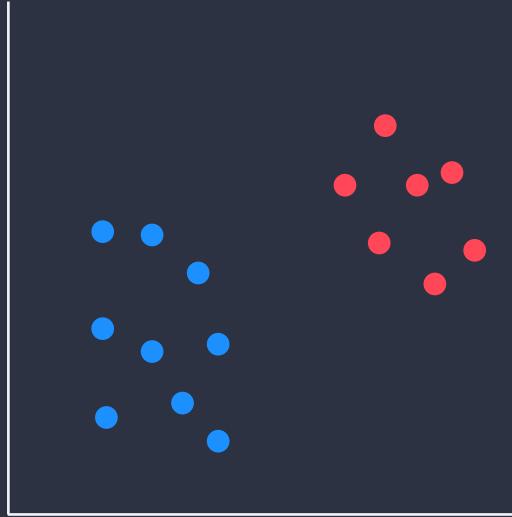
Fruit detection: apple, banana, orange

Handwriting recognition: A~Z

Movie classification: action, comedy, drama, ...

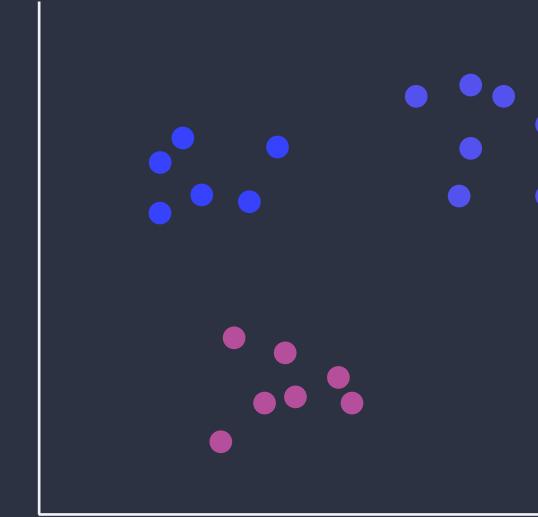
Weather forecast: sunny, rainy, snowy, cloudy, ...

# Introduction to Classification



## Binary Classification

Logistic Regression  
k-Nearest Neighbours  
Decision Trees  
Support Vector Machine  
Naïve Bayes



## Multiclass Classification

k-Nearest Neighbours  
Decision Trees  
Naïve Bayes  
Random Forest  
Gradient Boosting

# Introduction to Classification

- To estimate / evaluate how well a classification model performs:

**accuracy = correct predictions / total predictions**

## EXAMPLE.

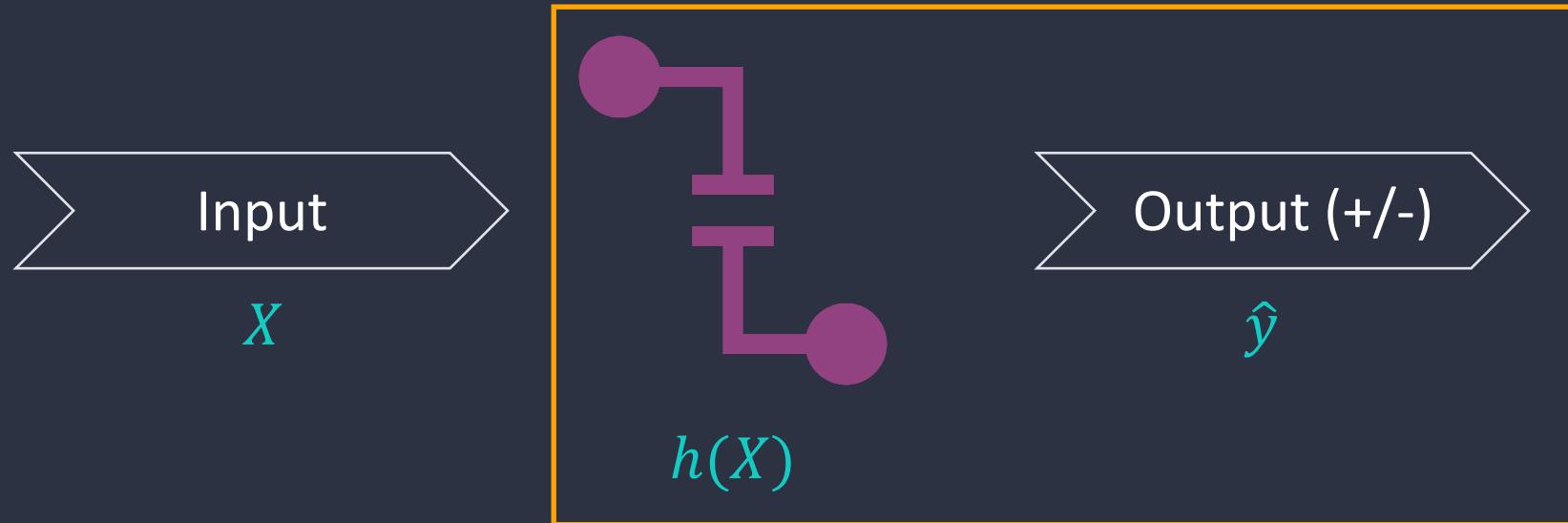
If our model made predictions on 50 students' UG degree award, of which 45 were correct, then the classification accuracy would be: **accuracy = 45 / 50 = 90%**

Other measures: recall, specificity, precision, F1 score, support, ...

## 2. Binary Classification

# Binary Classification

- Surprised -> Learns for  $h(X)$  mapping from  $X$  to  $\hat{y}$
- Classification -> Outputs discrete  $\hat{y}$
- Binary -> Only two possible output values + and -



What's the relationship between  $h(X)$  and  $\hat{y}$ ?

# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

	$X$				$y$
	$x_1$	$x_2$	$\dots$	$x_n$	
1	■	■	...	■	+
2	■	■	...	■	+
3	■	■	...	■	-
...	...	...	...	...	...
m	■	■	...	■	+

$h(X) \rightarrow \hat{y}$

# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

$x$	$y$
■	+
■	+
■	-
...	...
■	+

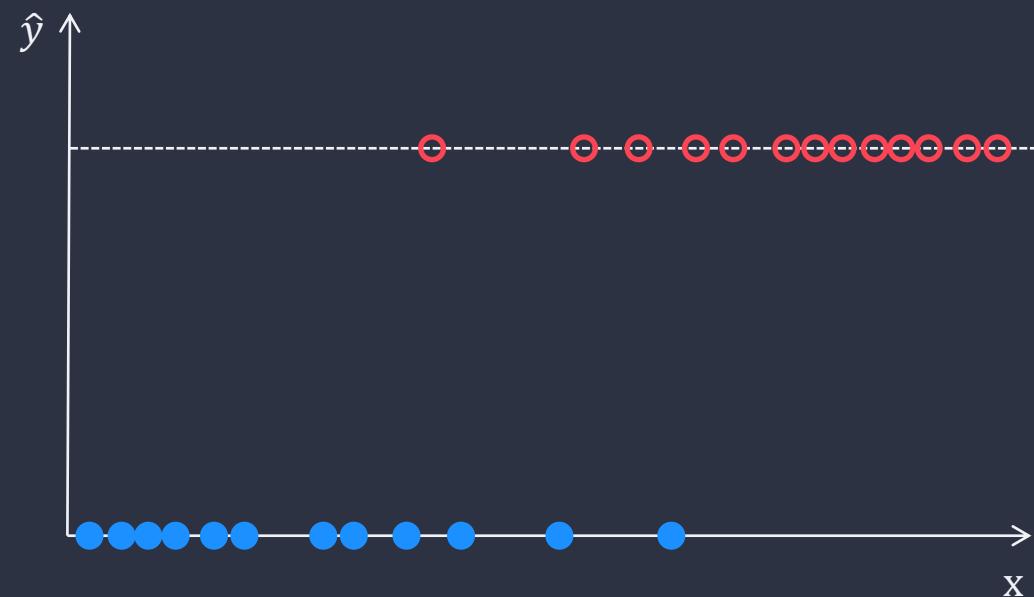


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

$x$	$y$
■	+
■	+
■	-
...	...
■	+

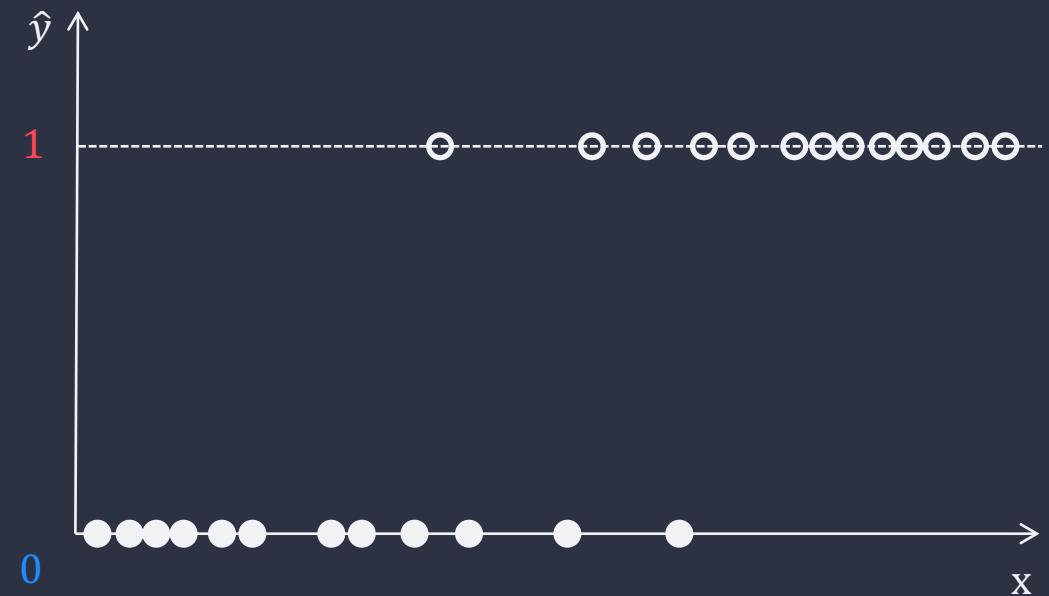


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

$x$	$y$
■	+
■	+
■	-
...	...
■	+

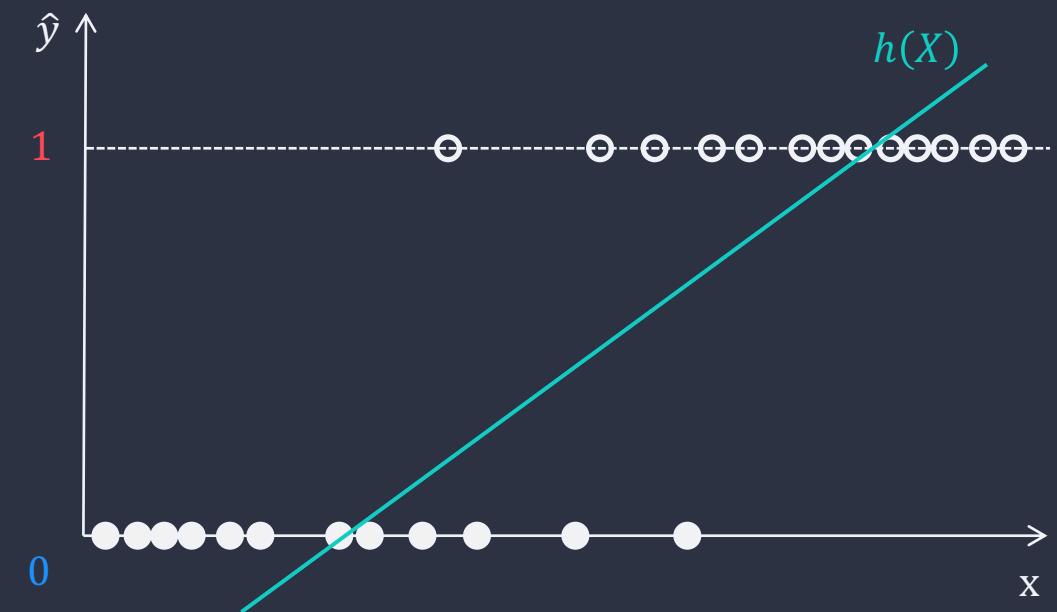


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

$x$	$y$
■	+
■	+
■	-
...	...
■	+

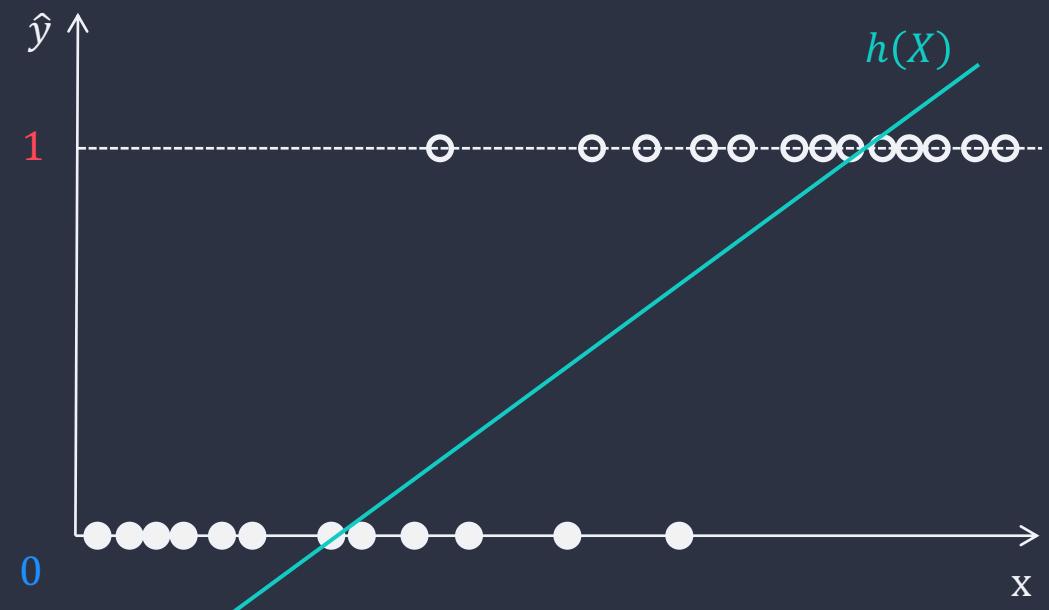


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

## Intuition

$x$	$y$	$h(x)$
■	+	$h_1$
■	+	$h_2$
■	-	$h_3$
...	...	...
■	+	$h_k$

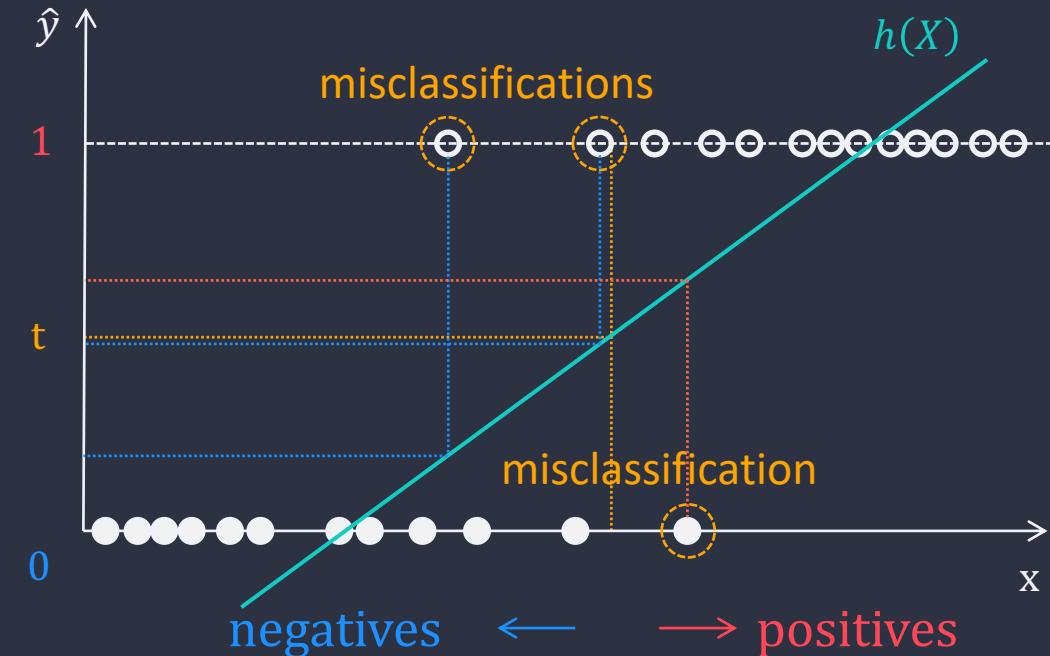


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

- Decision rule:  $\hat{y} = \begin{cases} +, & h(X) \geq t \\ -, & \text{otherwise} \end{cases}$

$x$	$y$	$h(x)$	$\hat{y}$
■	+	$h_1 > t$	1 (+)
■	+	$h_2 < t$	0 (-)
■	-	$h_3 > t$	1 (+)
...	...	...	...
■	+	$h_k < t$	0 (-)

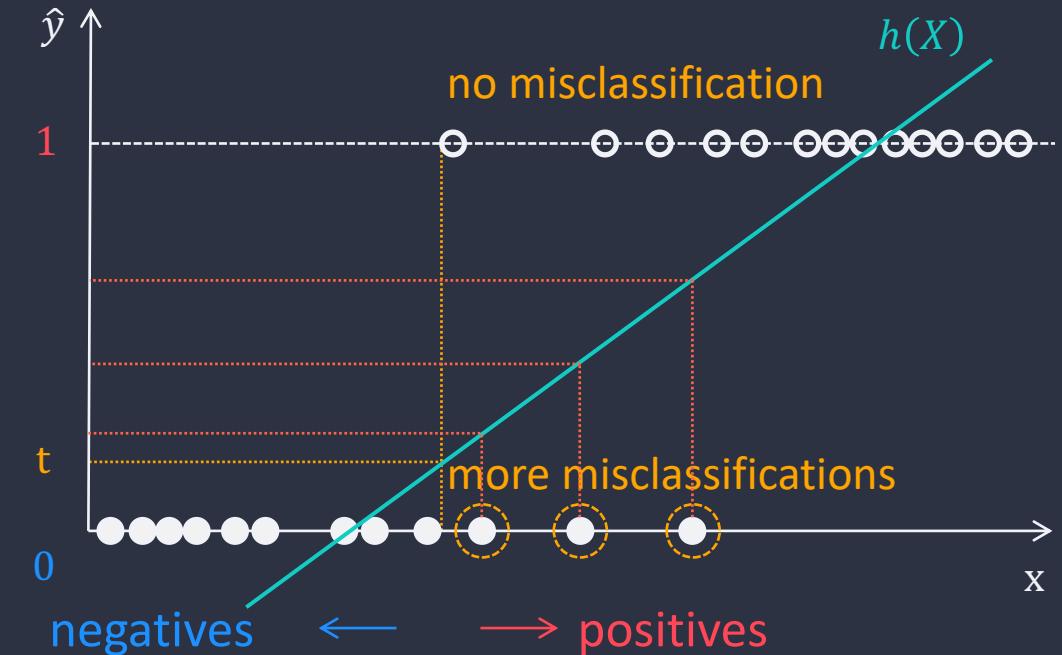


# Binary Classification

What's the relationship between  $h(X)$  and  $\hat{y}$ ?

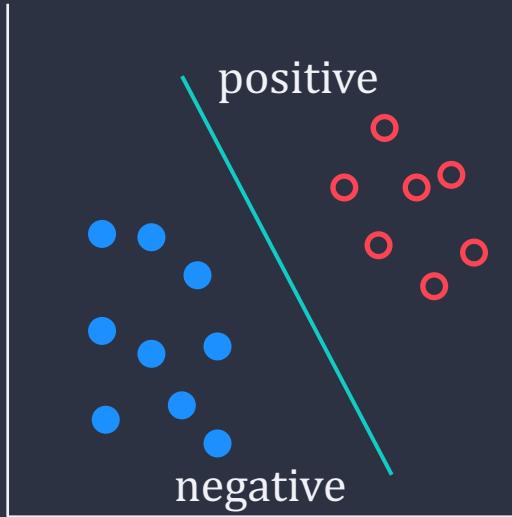
- Decision rule:  $\hat{y} = \begin{cases} +, & h(X) \geq t \\ -, & \text{otherwise} \end{cases}$

$x$	$y$	$h(x)$	$\hat{y}$
■	+	$h_1 > t$	1 (+)
■	+	$h_2 < t$	0 (-)
■	-	$h_3 > t$	1 (+)
...	...	...	...
■	+	$h_k < t$	0 (-)



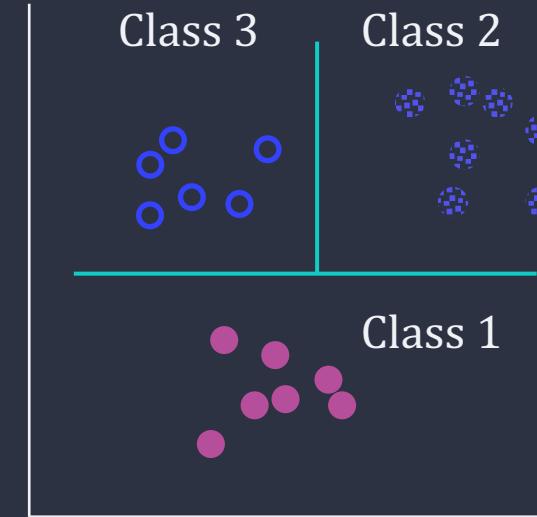
### 3. Binary Classifier for Multiclass Classification

# Binary Classifier for Multiclass Classification



Binary Classification

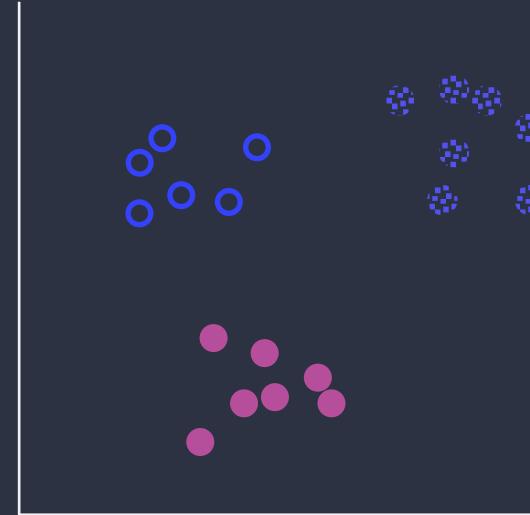
Two categories / labels



Multiclass Classification

More than two categories / labels

# Binary Classifier for Multiclass Classification



Multiclass Classification

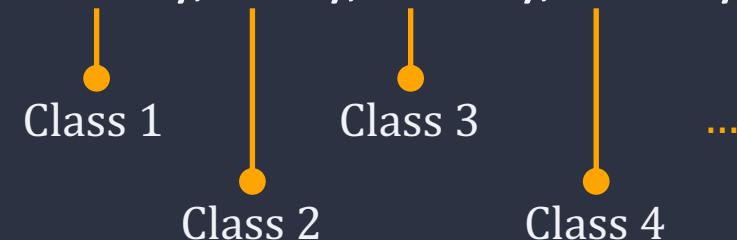
## EXAMPLES:

Fruit detection: apple, banana, orange

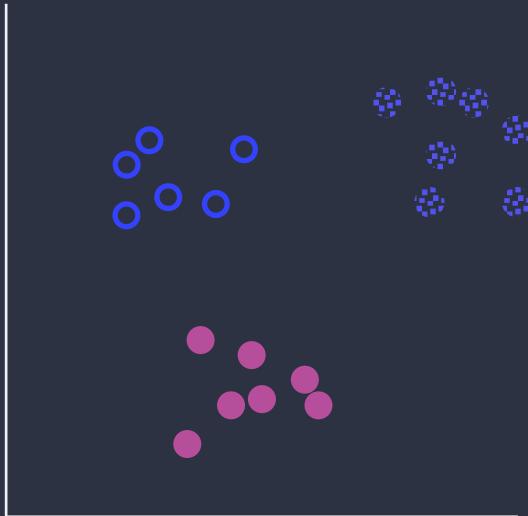
Handwriting recognition: A~Z

Movie classification: action, comedy, drama,...

Weather forecast: sunny, rainy, snowy, cloudy,...



# Binary Classifier for Multiclass Classification



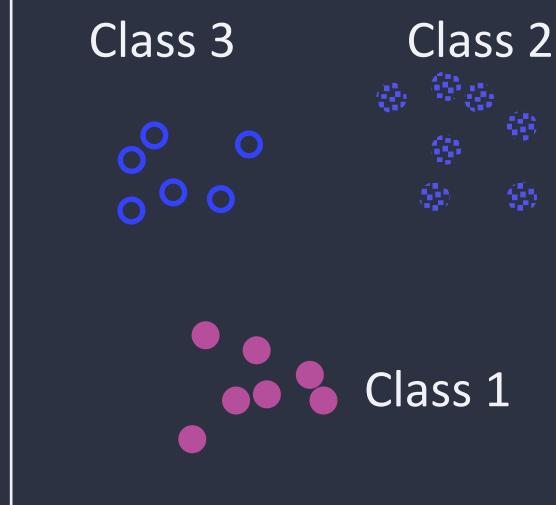
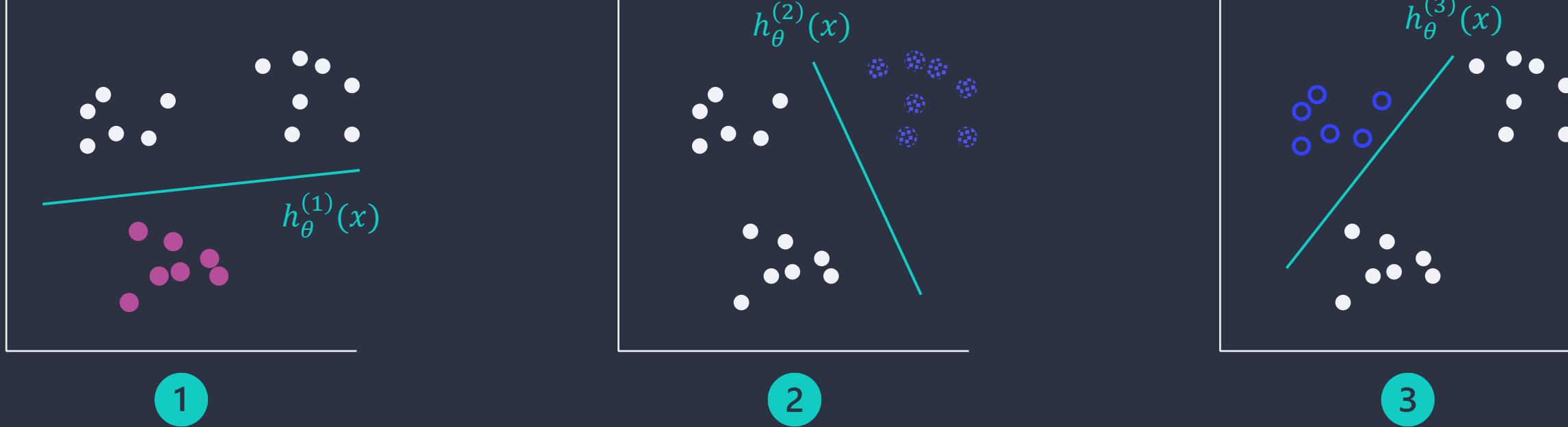
Multiclass Classification

ALGORITHMS:

k-Nearest Neighbours  
Decision Trees  
Naïve Bayes  
Random Forest  
Gradient Boosting

Multiple Binary Classifiers to perform  
Multiclass Classification Tasks

# One-vs-Rest



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

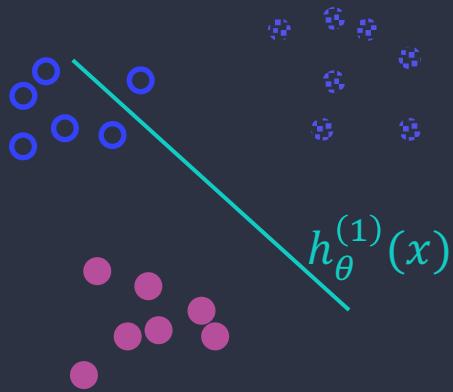
Multiple Binary Classifiers to perform  
Multiclass Classification Tasks

# One-vs-One

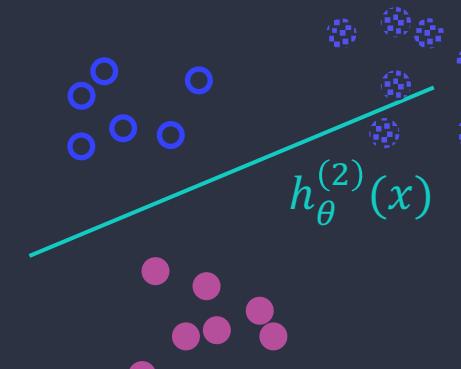


Multiple Binary Classifiers to perform  
Multiclass Classification Tasks

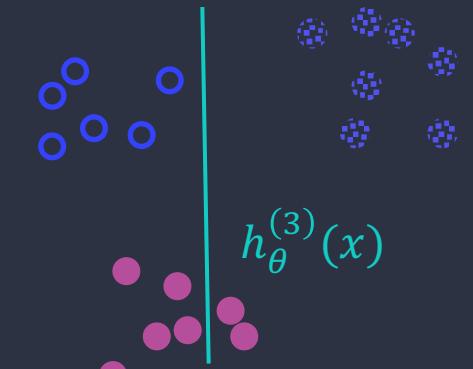
# One-vs-One



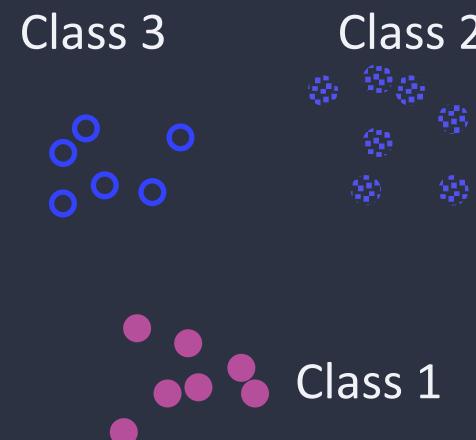
1



2



3



$n \times (n - 1)/2$  binary classifiers

e.g. 3 classes: 3 binary classifiers

10 classes:  $10 \times 9/2 = 45$  binary classifiers

Multiple Binary Classifiers to perform  
Multiclass Classification Tasks

# Machine Learning Algorithms



Regression



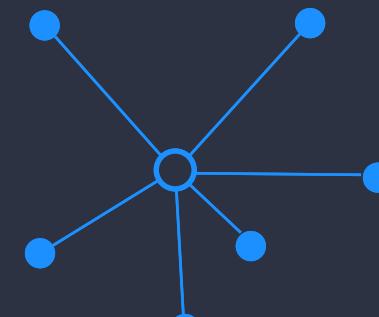
Regularisation



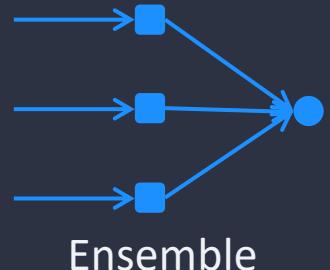
Clustering



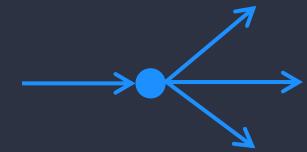
Bayesian



Instance-based



Ensemble



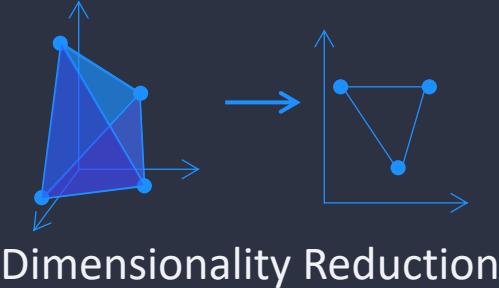
Neural Network



Deep Learning



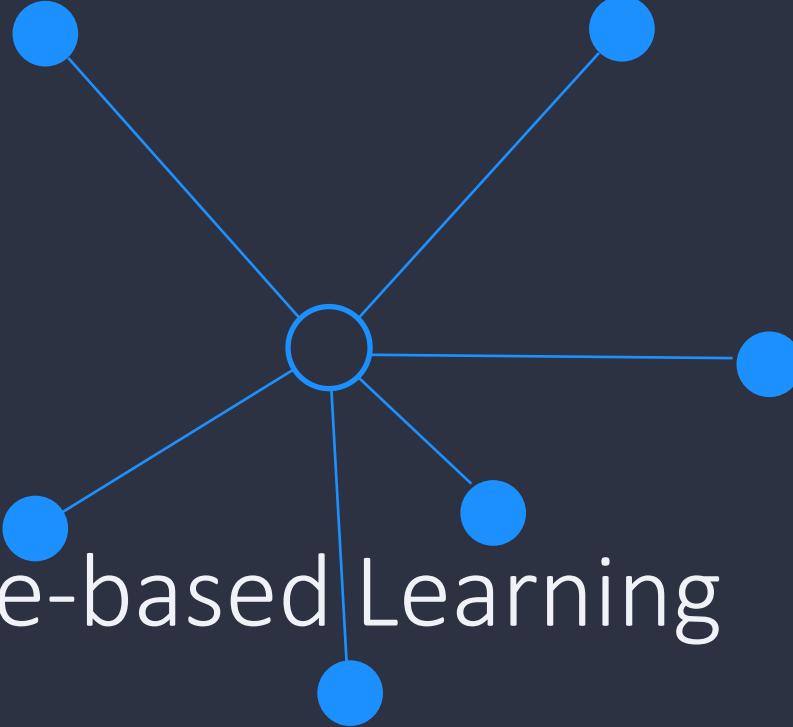
Associated Rule Learning



Dimensionality Reduction



Reinforcement Learning



## 4. Instance-based Learning

# Instance-based Learning vs Model-based Learning

# Instance-based Learning

- Memories (not learns) training examples (memory-based learning).
- Compares new examples with examples seen in training, based on similarity measure.
- Postpones computation until a new example is observed (lazy learning).
- Constructs hypotheses directly from the training examples themselves.

# Model-based Learning

- Learns (not memorise) from training examples and create a model with parameters.
- Uses models to make predictions.
- Takes long time to learn and less time to predict.
- Constructs hypotheses from the patterns in the training examples.

# Instance-based Learning

- k Nearest Neighbours
- Self-Organising Map
- Learning Vector Quantisation
- Locally Weighted Learning

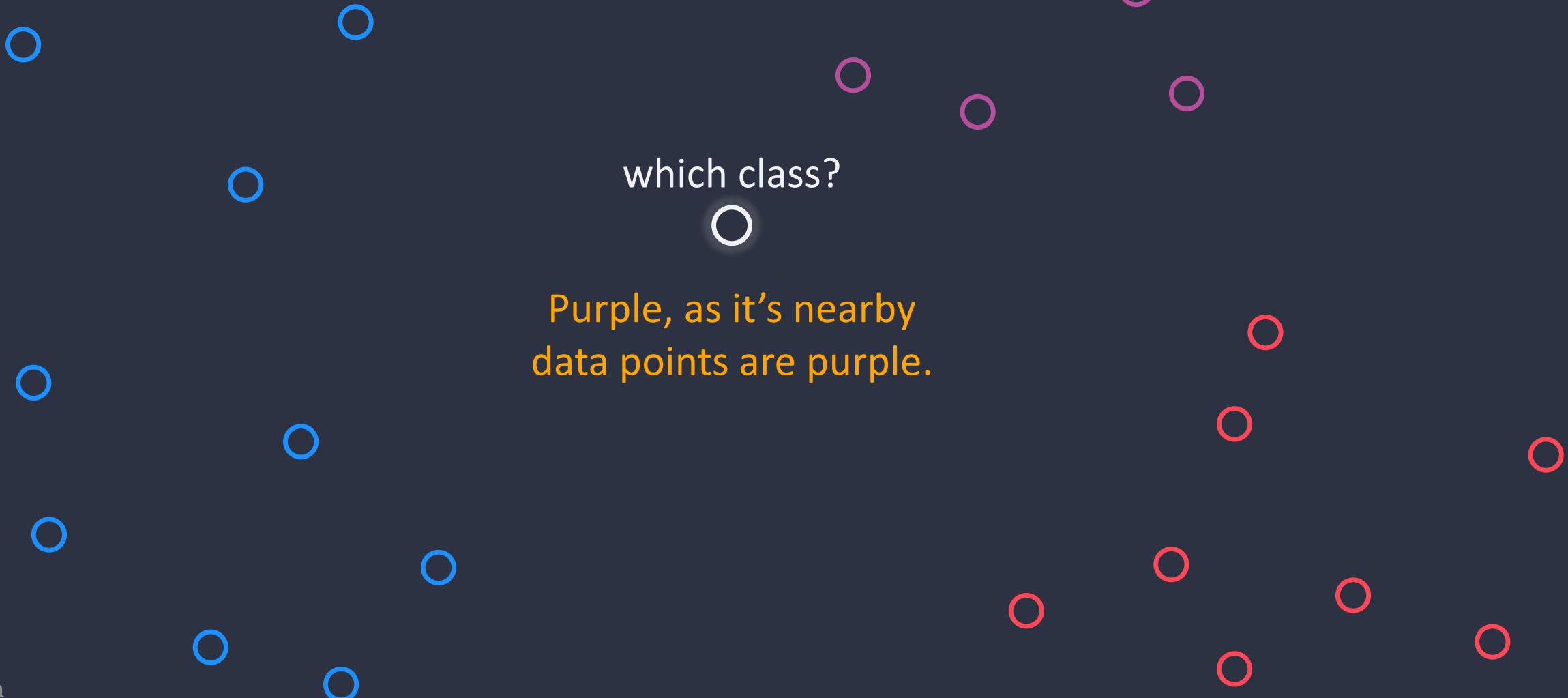
# Model-based Learning

- Linear Regression
- k-means Clustering
- Principal Component Analysis
- Logistic Regression

# 5. k Nearest Neighbours

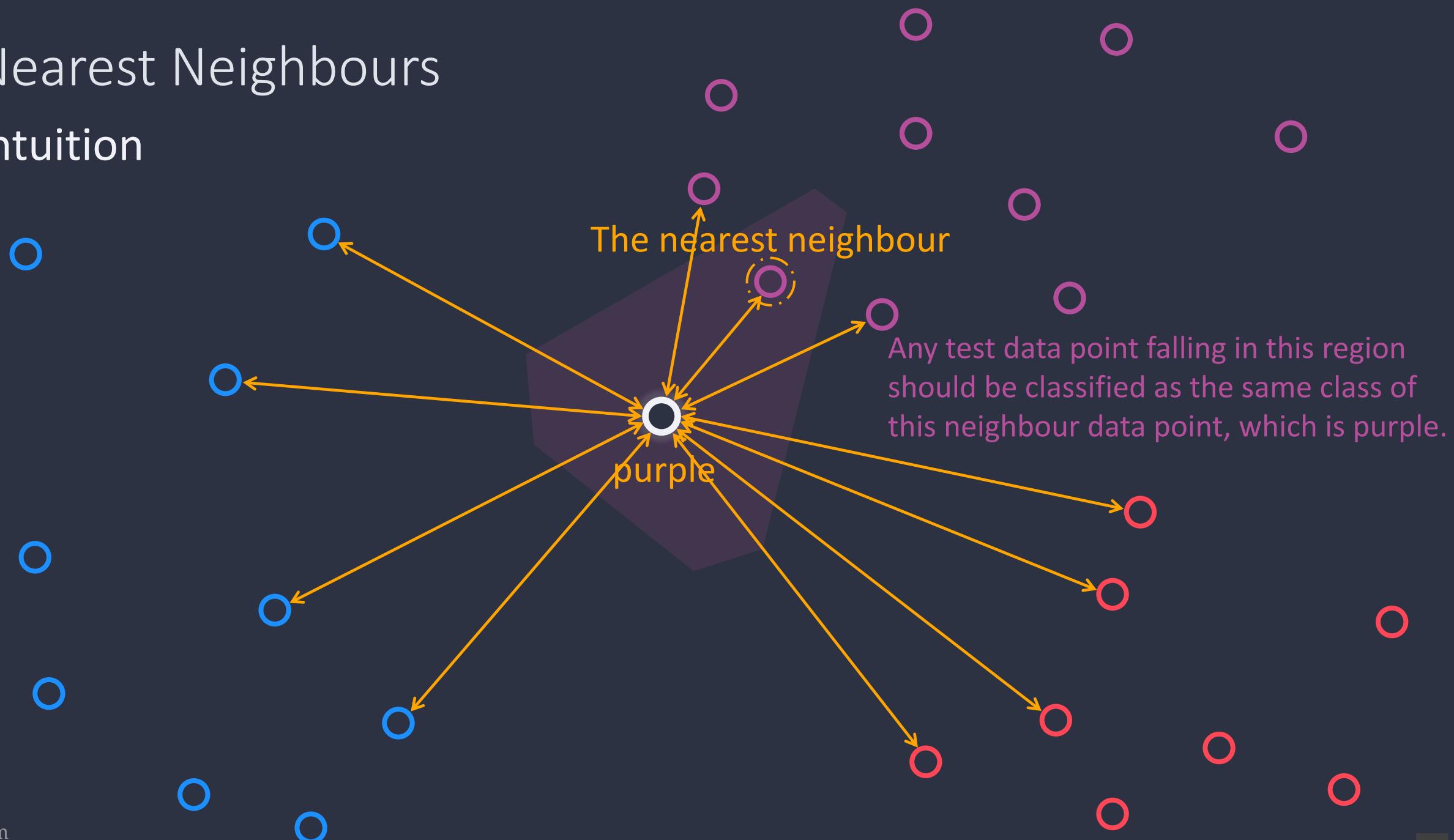
# k Nearest Neighbours

## Intuition



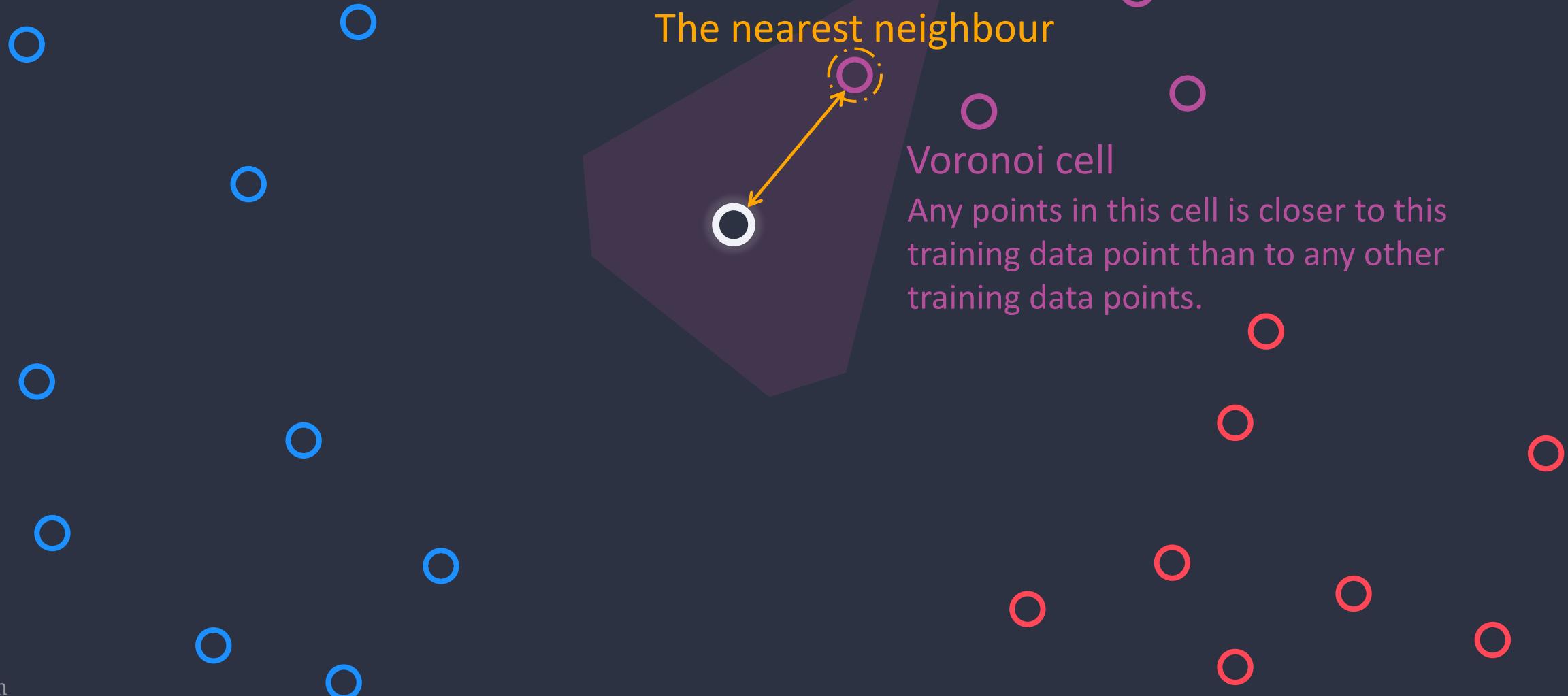
# k Nearest Neighbours

## Intuition



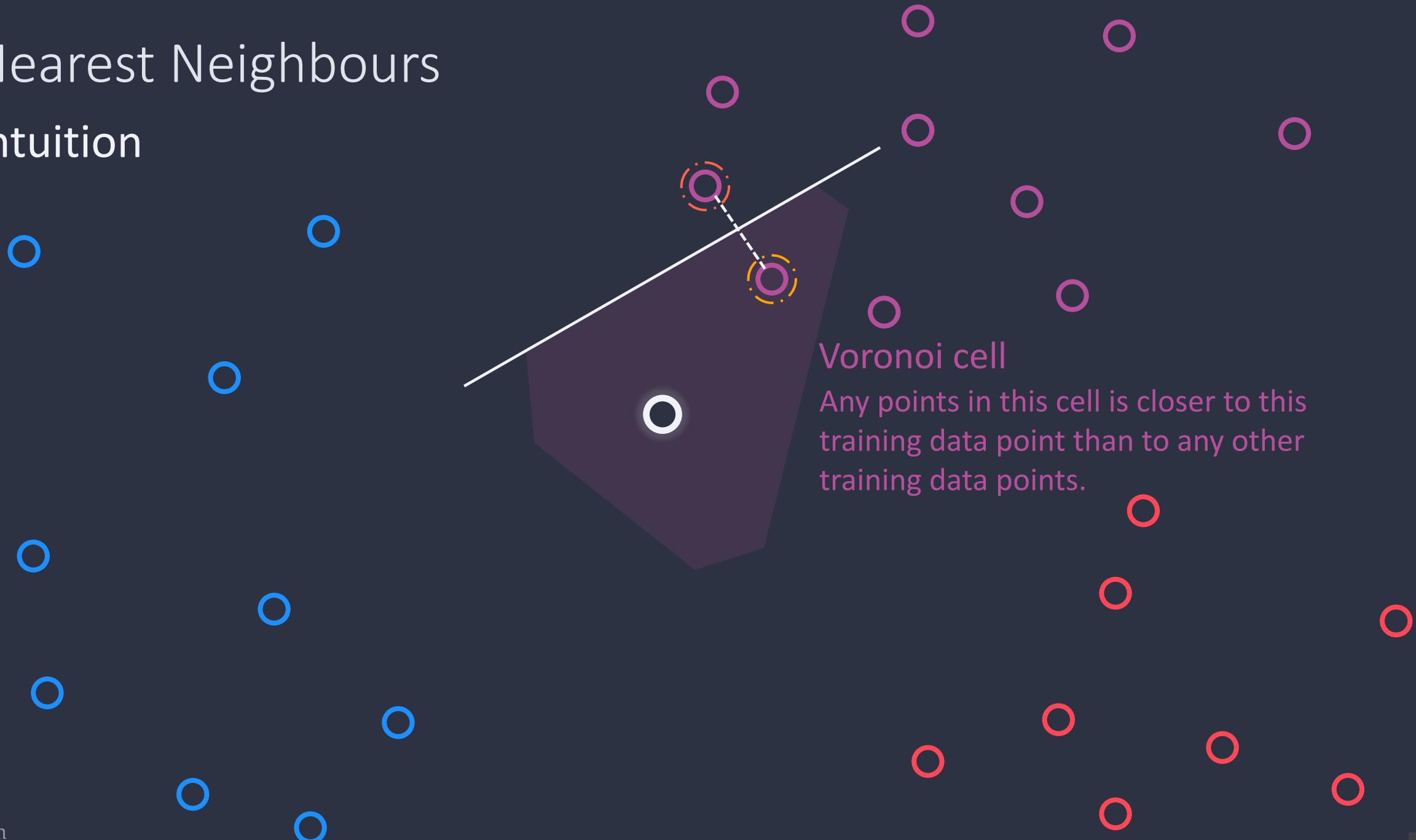
# k Nearest Neighbours

## Intuition



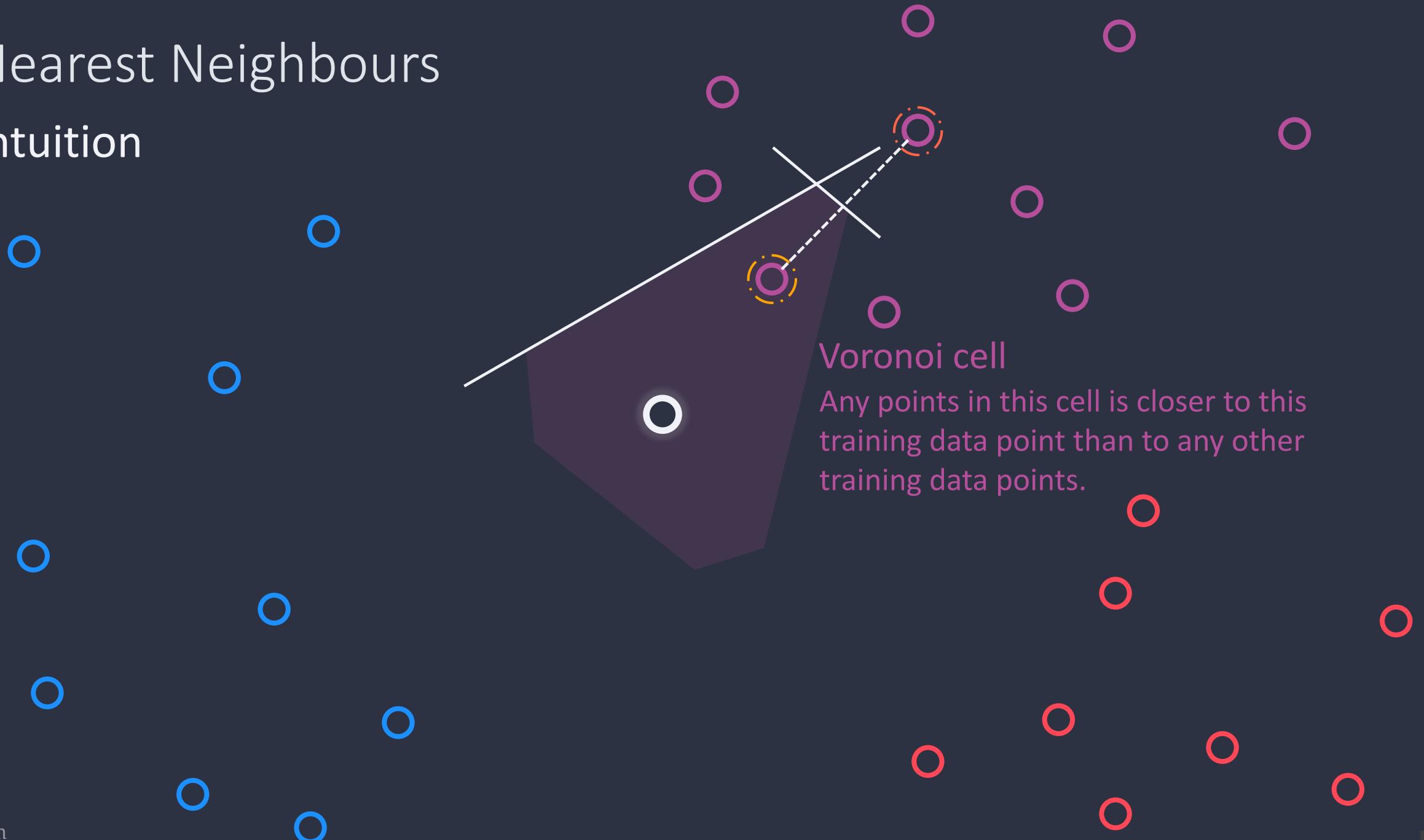
# k Nearest Neighbours

## Intuition



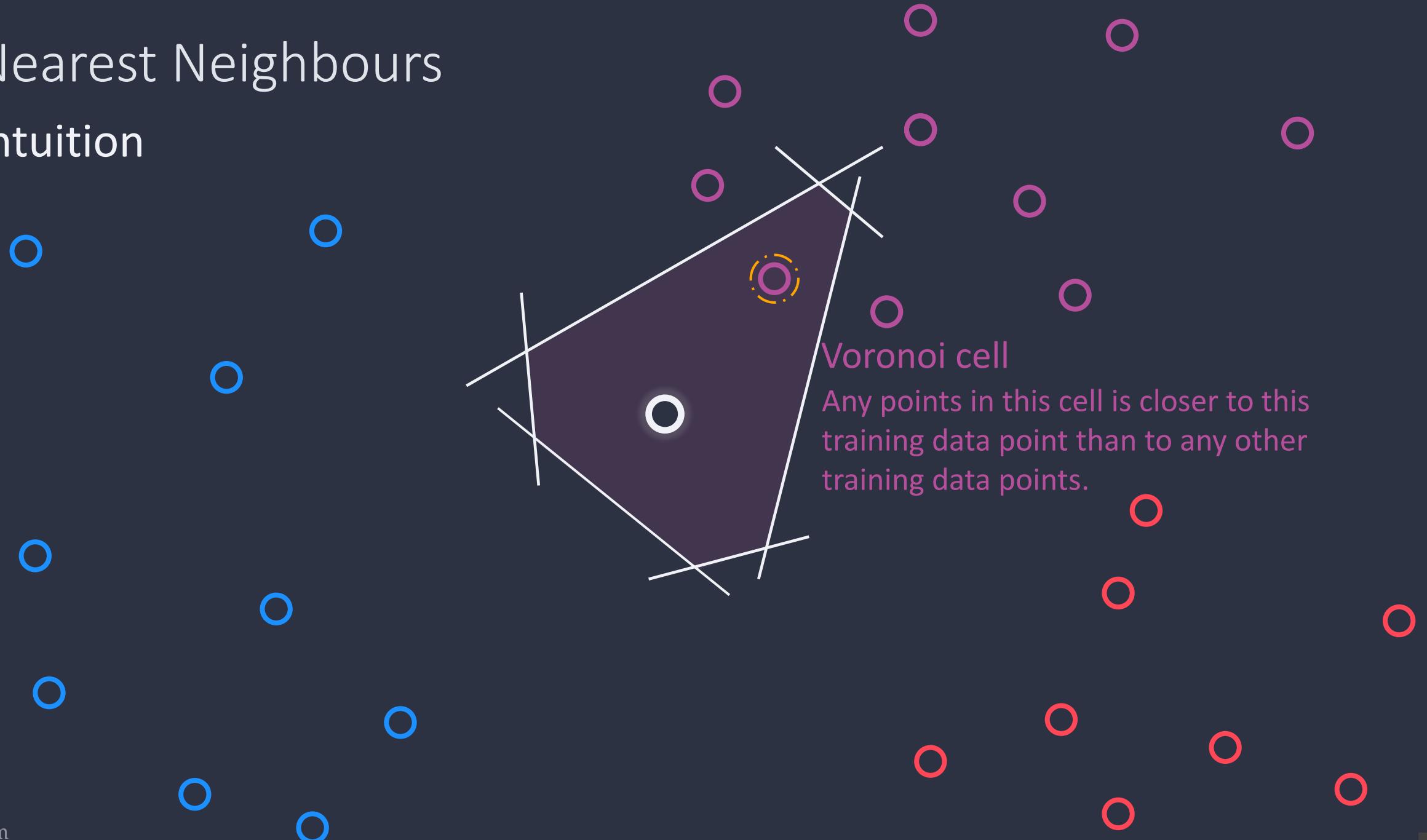
# k Nearest Neighbours

## Intuition



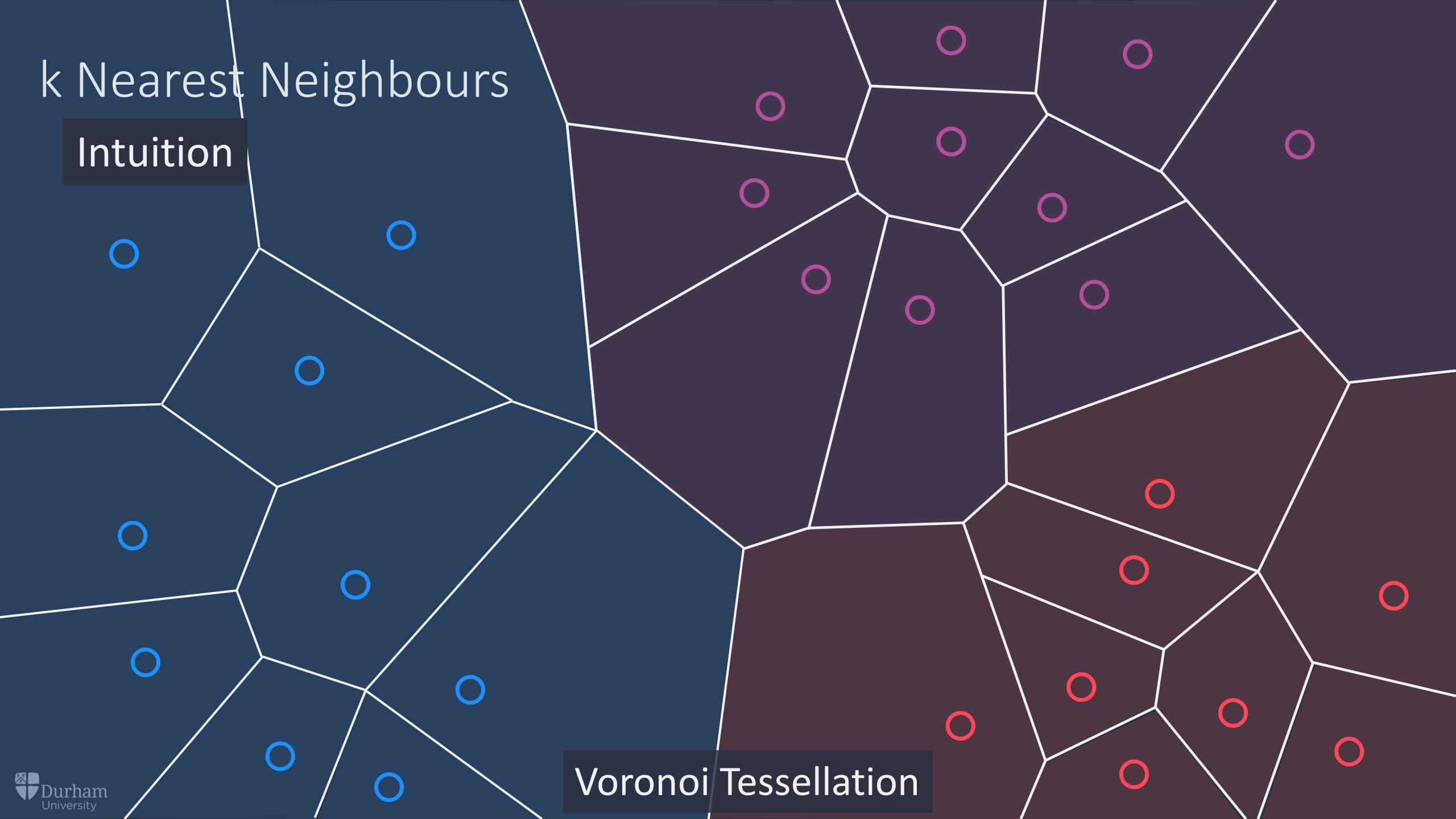
# k Nearest Neighbours

## Intuition



# $k$ Nearest Neighbours

Intuition



Voronoi Tessellation

# k Nearest Neighbours

## Intuition

These two test data points are both classified as purple, but for different reasons, as there are different seed data points that make them purple.

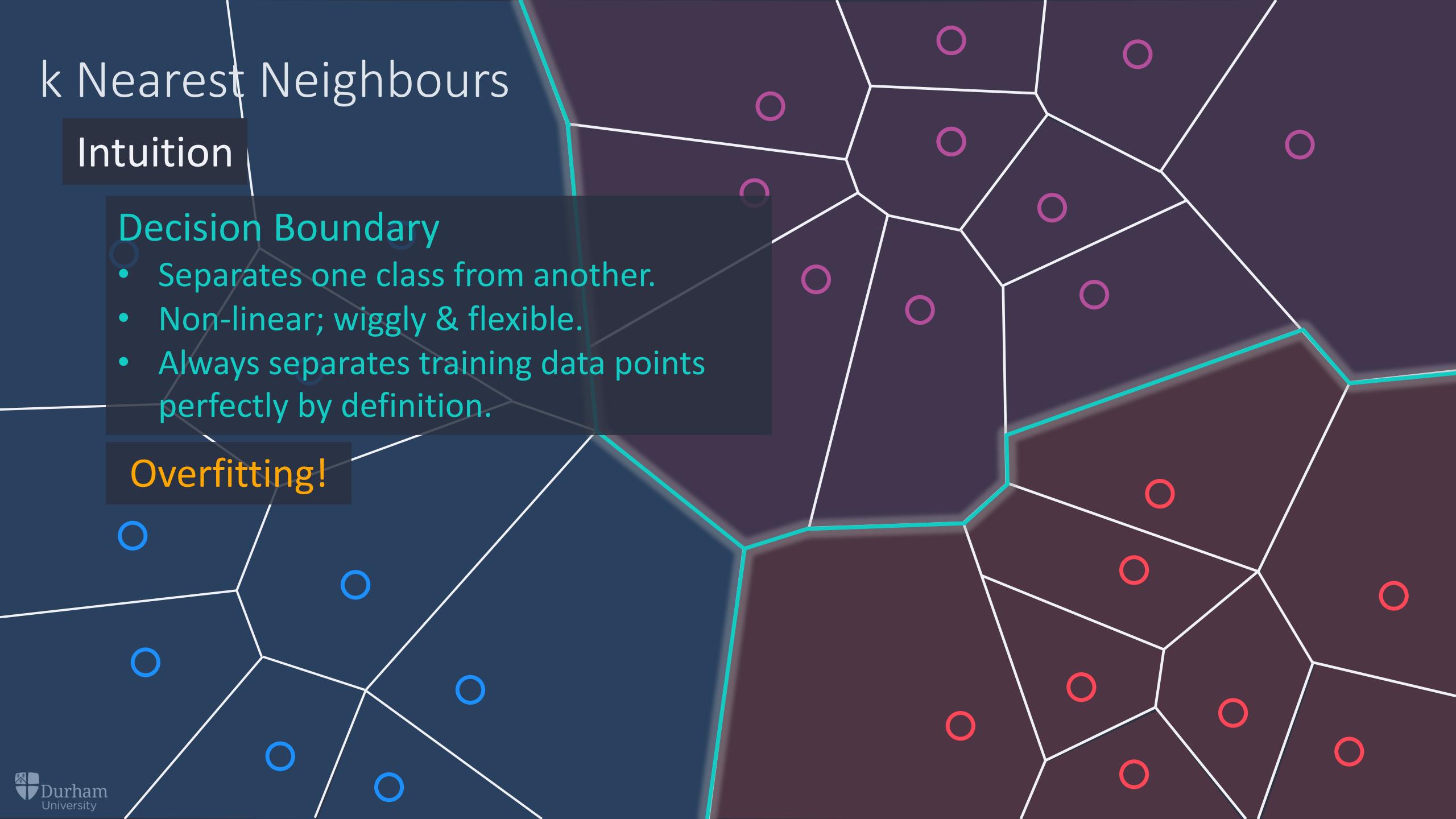
# k Nearest Neighbours

## Intuition

### Decision Boundary

- Separates one class from another.
- Non-linear; wiggly & flexible.
- Always separates training data points perfectly by definition.

Overfitting!



# k Nearest Neighbours

## Intuition

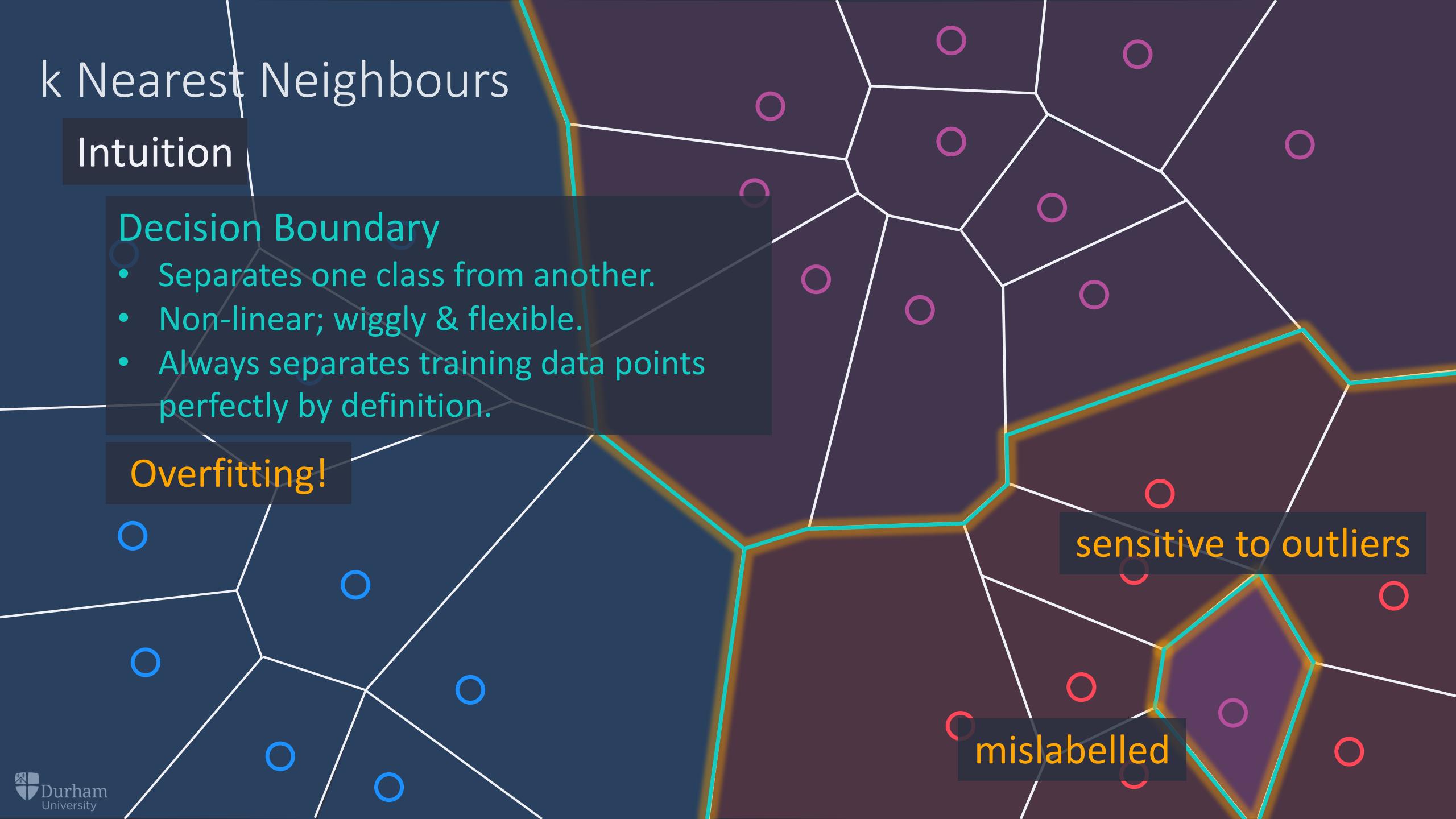
### Decision Boundary

- Separates one class from another.
- Non-linear; wiggly & flexible.
- Always separates training data points perfectly by definition.

Overfitting!

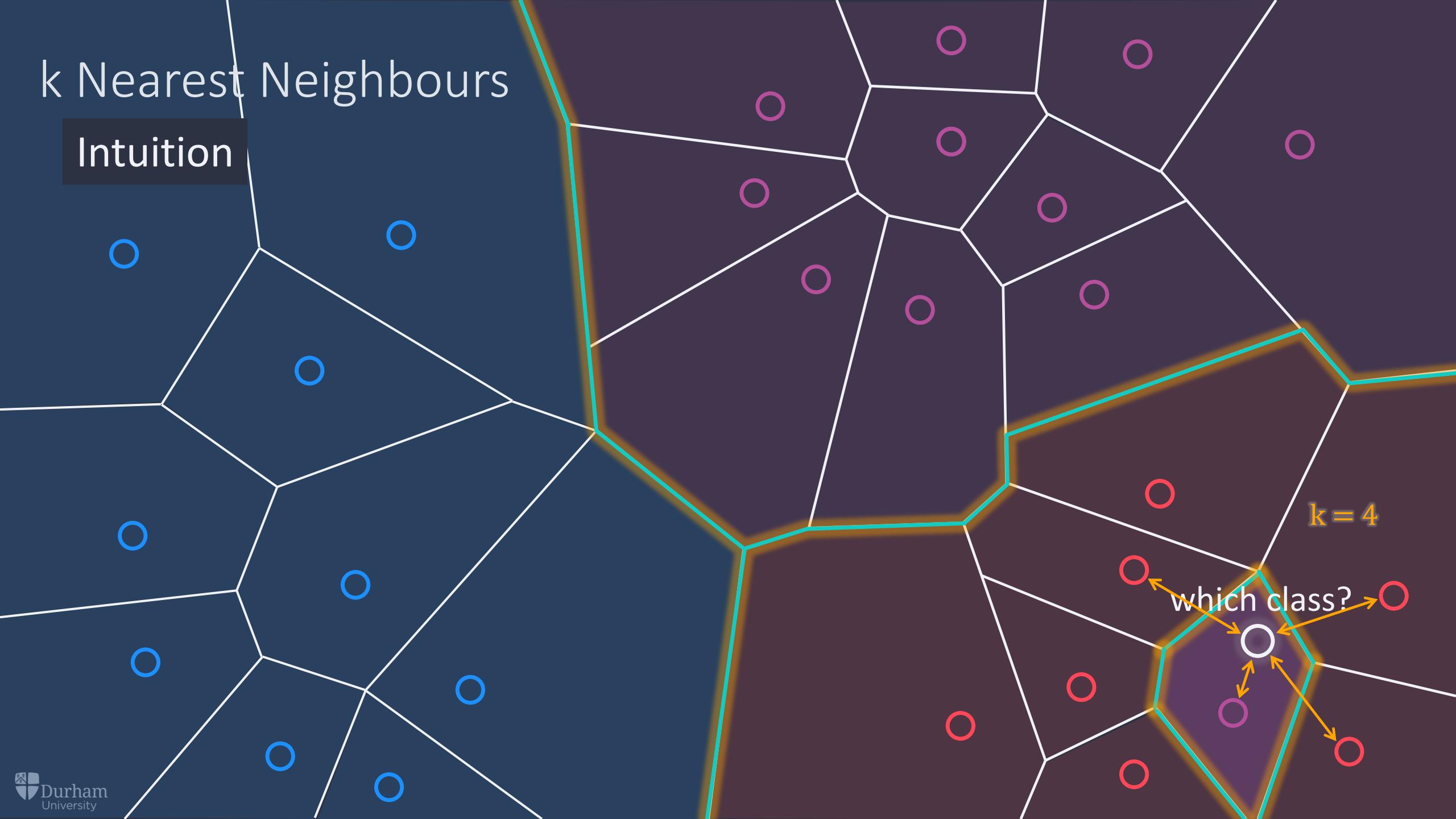
sensitive to outliers

mislabelled



# k Nearest Neighbours

Intuition



# k Nearest Neighbours

## Intuition

The value of  $k$  strongly affects the performance.

### Small $k$

- Is sensitive to outliers
- Overfits training data

### Large $k$

- Approaches a prior classifier
- Outputs the most frequent class

which class?

if  $k = 1$ , then blue  
if  $k = 4$ , then purple  
if  $k = 10$ , then red

# k Nearest Neighbours

## Algorithm

Given

Training examples  $\{(X^{(i)}, y^{(i)}) \mid i \in \{1, 2, \dots, m\}\}$

$X^{(i)}$  feature-value representation of the  $i^{\text{th}}$  data point

$y^{(i)}$  label of the  $i^{\text{th}}$  data point

Test example

$X$ , to be labelled (i.e. to predict  $\hat{y}$ )

How to decide the value of  $k$ ?

How to decide the label  $\hat{y}$ ?

Learning algorithm

Calculate the distance  $d(X, X^{(i)})$  to every training data point  $X^{(i)}$

Choose the  $k$  nearest examples  $X^{(i1)}, \dots, X^{(ik)}$  and their labels  $y^{(i1)}, \dots, y^{(ik)}$

Get the label  $\hat{y}$  which is the most frequent in  $y^{(i1)}, \dots, y^{(ik)}$

Output the mode (for classification) or the mean (for regression) of  $k$  labels

How to decide the value of  $k$ ?

How to decide the label  $\hat{y}$ ?

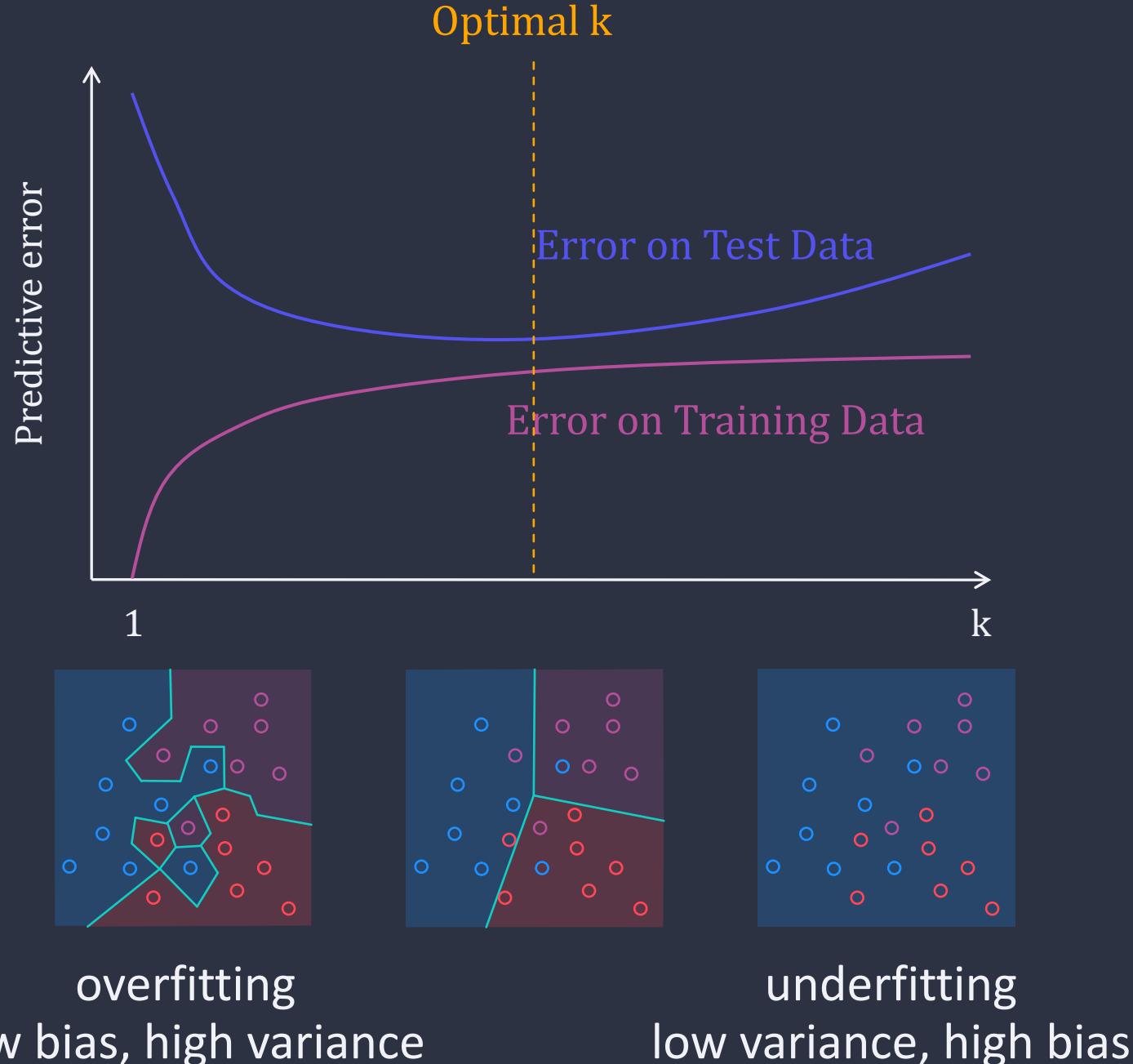
How to calculate the distance  $d(X, X^{(i)})$  ?

# $k$ Nearest Neighbours

How to decide the value of  $k$ ?

## Cross Validation

- To test the model's ability to predict new data that was not used in estimating it, in order to identify overfitting and selection bias problems.
- Try different values for  $k$ , and pick the optimal one for the model.
- Overfitting, if  $k$  is too small.
- Underfitting, if  $k$  is too large.



How to decide the value of  $k$ ?

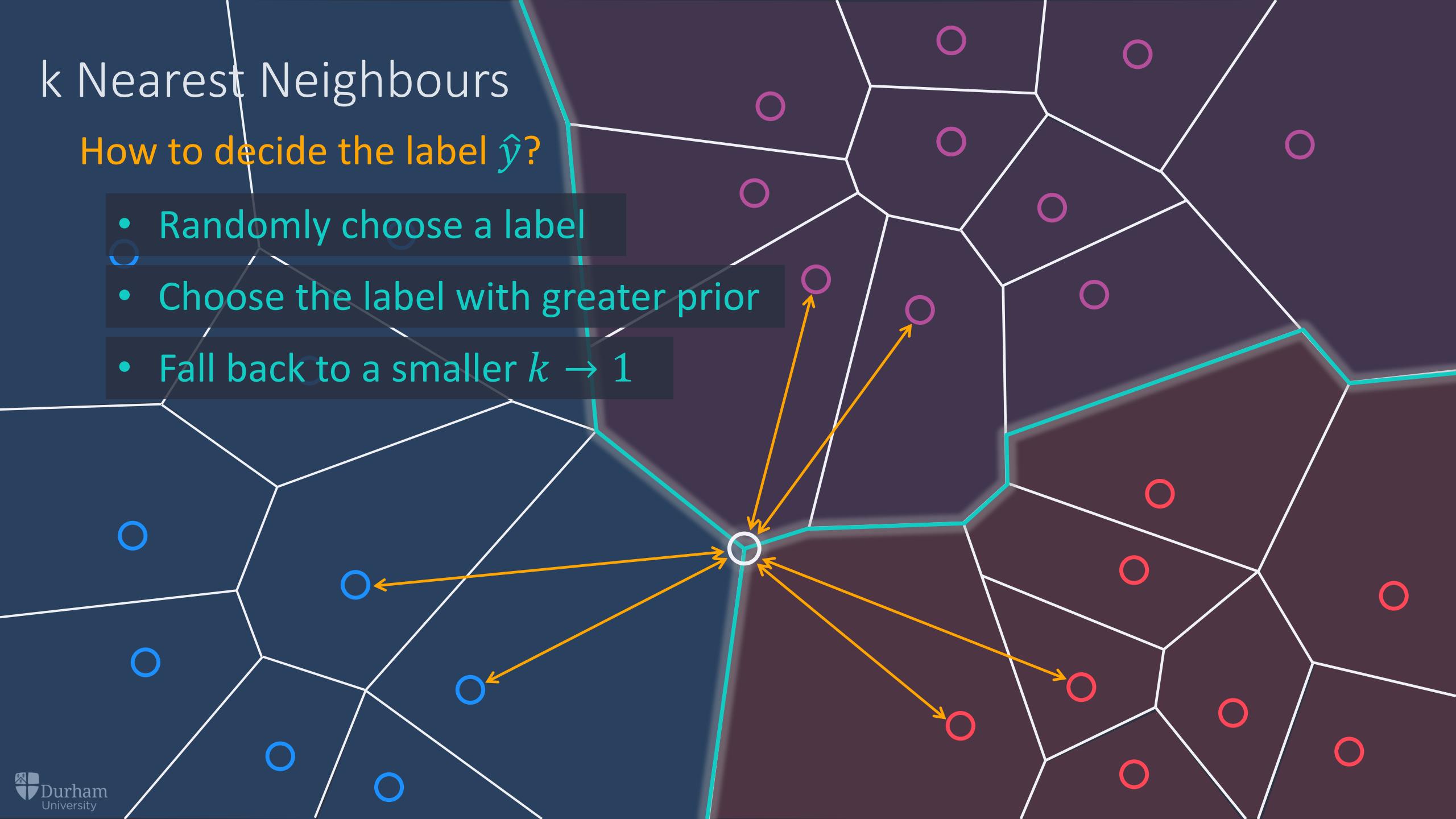
How to decide the label  $\hat{y}$ ?

How to calculate the distance  $d(X, X^{(i)})$  ?

# $k$ Nearest Neighbours

How to decide the label  $\hat{y}$ ?

- Randomly choose a label
- Choose the label with greater prior
- Fall back to a smaller  $k \rightarrow 1$



How to decide the value of  $k$ ?

How to decide the label  $\hat{y}$ ?

How to calculate the distance  $d(X, X^{(i)})$  ?

# k Nearest Neighbours

How to calculate the distance  $d(X, X^{(i)})$  ?

## Distance Measures

# k Nearest Neighbours - Distance Measures

- Determines which examples are nearest neighbours
- Different distance definitions may result in very different classifiers with very different performances

## Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

- For numerical (real-valued) features
- Symmetric, treats all dimensions equally
- Sensitive to extreme differences in single feature

# k Nearest Neighbours - Distance Measures

## Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

- For numerical (real-valued) features
- Symmetric, treats all dimensions equally
- Sensitive to extreme differences in single feature

## Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

- Generalisation of Euclidean distance
- With a parameter called “order” or “p”
- Becomes Euclidean distance, when  $p = 2$
- Becomes Manhattan distance, when  $p = 1$

# k Nearest Neighbours - Distance Measures

## Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

## Minkowski distance

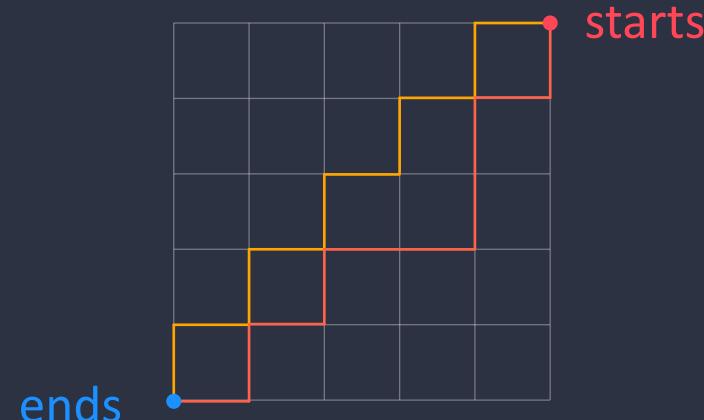
$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

## Manhattan distance

$$d(x, x') = \sqrt[1]{\sum_{i=1}^n |x_i - x'_i|^1}$$

$$= \sum_{i=1}^n |x_i - x'_i|$$

- City block distance / taxicab geometry



# k Nearest Neighbours - Distance Measures

Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

Manhattan distance

$$d(x, x') = \sum_{i=1}^n |x_i - x'_i|$$

Hamming distance

$$\text{dog} \quad x = [0,0,1]$$

$$\text{wolf} \quad x' = [0,1,1]$$

$$\text{cat} \quad x'' = [1,0,0]$$

$$d(x, x') = 1 < 3 = d(x', x'')$$

# k Nearest Neighbours - Distance Measures

Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

Manhattan distance

$$d(x, x') = \sum_{i=1}^n |x_i - x'_i|$$

Hamming distance

$$d(x, x') = \sum_{i=1}^n \mathbb{1}_{x_i \neq x'_i}$$

# k Nearest Neighbours - Distance Measures

Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

Manhattan distance

$$d(x, x') = \sum_{i=1}^n |x_i - x'_i|$$

Hamming distance

$$d(x, x') = \sum_{i=1}^n \mathbb{1}_{x_i \neq x'_i}$$

Missing data?

Must “fill in” – mean/median (continuous); most frequent label (categorical).

Or, use regression/classification models to estimate.

# k Nearest Neighbours - Distance Measures

Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

Manhattan distance

$$d(x, x') = \sum_{i=1}^n |x_i - x'_i|$$

Hamming distance

$$d(x, x') = \sum_{i=1}^n \mathbb{1}_{x_i \neq x'_i}$$

Feature scaling to improve performance

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$$x' = \frac{x - \text{avg}(x)}{\max(x) - \min(x)}$$

$$x' = \frac{x - \bar{x}}{\sigma}$$

$$x' = \frac{x}{\|x\|}$$

# k Nearest Neighbours

## Coding with sci-kit learn

```
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
scaler.fit(X_train)
```

```
X_train = scaler.transform(X_train)  
X_test = scaler.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier  
classifier = KNeighborsClassifier(n_neighbors=5, metric='euclidean')  
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
```

# Summary

# Summary

Classification, Binary Classification, Multiclass Classification

Instance-based Learning, k Nearest Neighbours (kNN)

Next Lecture

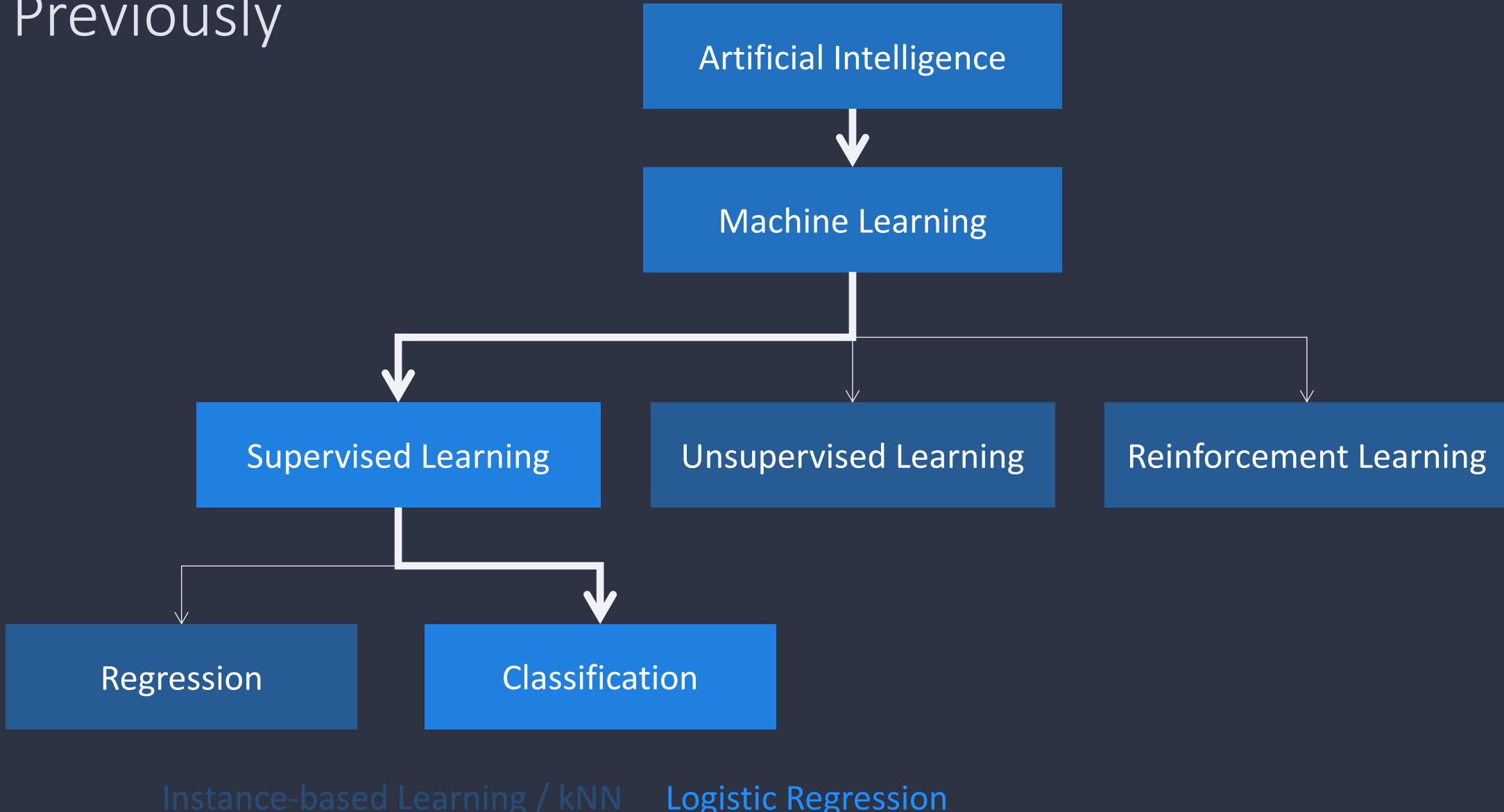
Classifier Evaluation

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Logistic Regression (Part I)

Dr Yang Long

# Previously



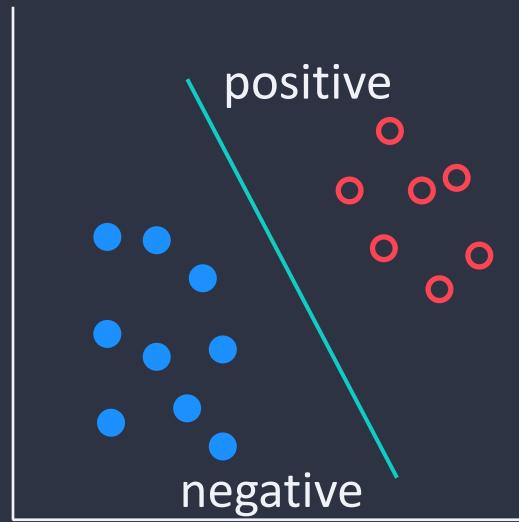
# Lecture Overview (Logistic Regression, Part I)

1. Intuition
2. Odds, Log Odds, Odds Ratio
3. Hypothesis Function
4. Decision Boundary
5. Cost Function
6. Gradient Descent
7. Regularisation
8. Multinomial Logistic Regression (Softmax Regression)

# 1. Intuition

# Intuition

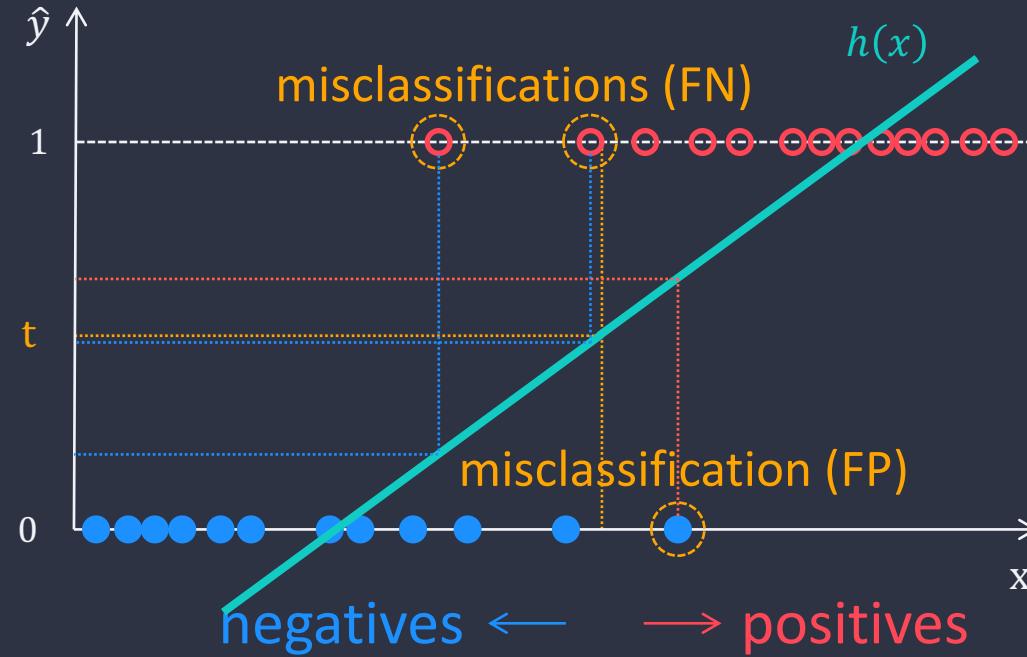
## Binary Classification



Two categories / labels

# Intuition

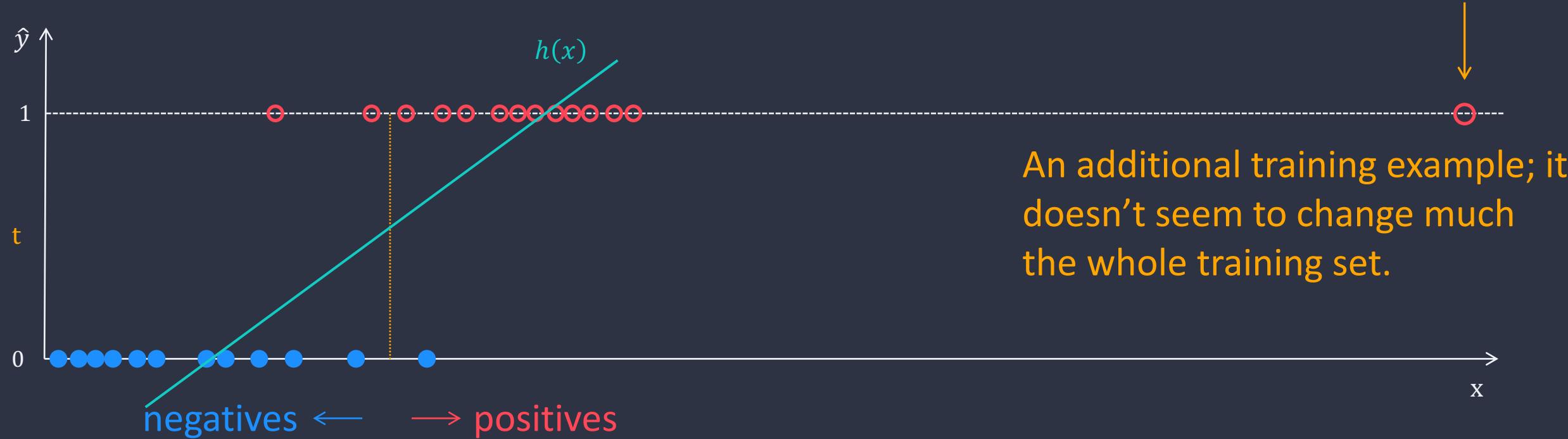
## Linear Regression to solve Binary Classification tasks



Decision rule:  $\hat{y} = \begin{cases} +, & h(x) \geq t \\ -, & \text{otherwise} \end{cases}$

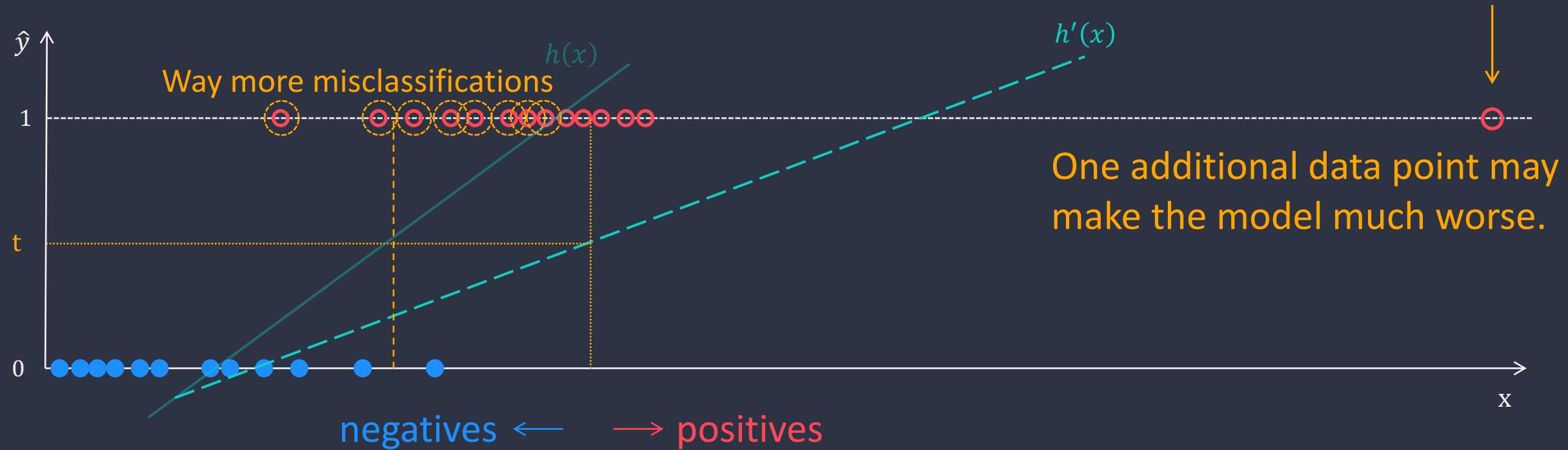
# Intuition

## Problems of using Linear Regression to solve Binary Classification tasks



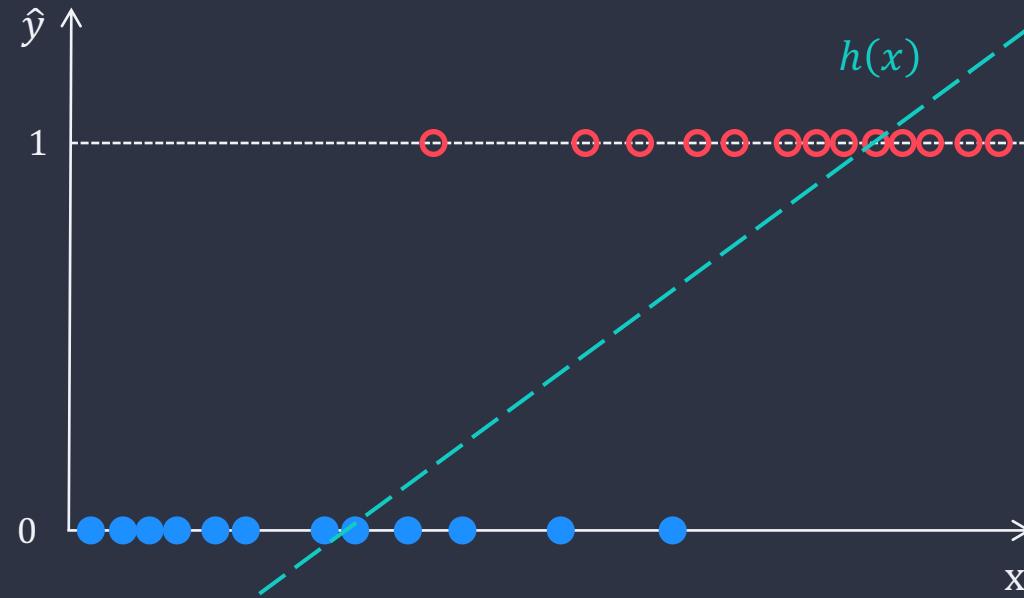
# Intuition

## Problems of using Linear Regression to solve Binary Classification tasks



# Intuition

## Problems of using Linear Regression to solve Binary Classification tasks

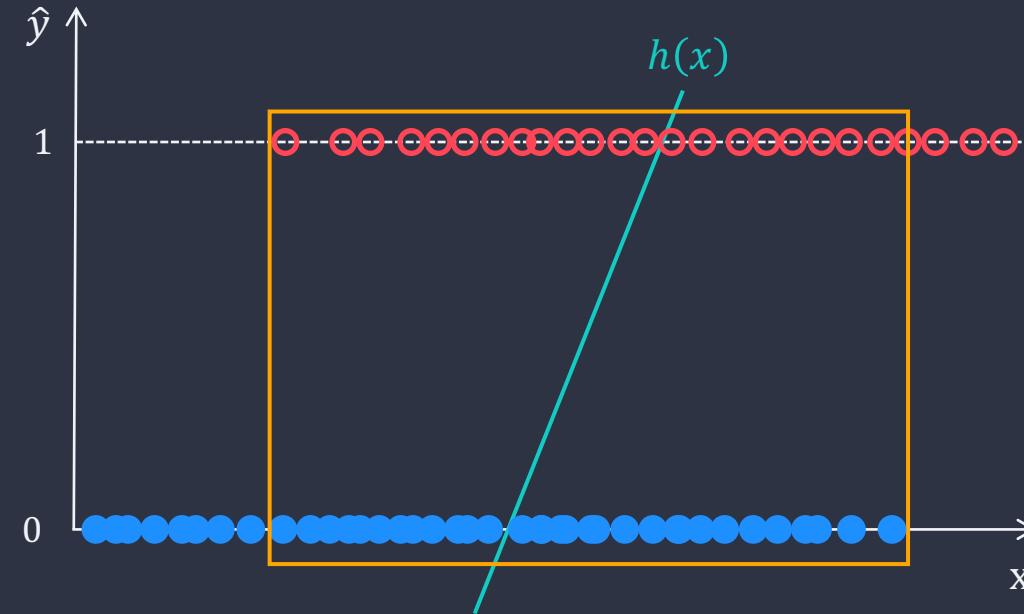


Both input  $x$  and output  $\hat{y} \in (-\infty, +\infty)$ , even if examples in training set have only labels of 0 and 1.

In practice, we would really need a more dedicated classification model.

# Intuition

## Problems of using Linear Regression to solve Binary Classification tasks



Big overlap between positive & negative examples → a linear model would cause too many misclassifications.

# Logistic Regression

# Aims of Logistic Regression

To model

the probability of an event occurring  
depending on the input values.

To estimate

the probability that an event occurs vs  
doesn't occur.

To predict

the effect of a set of input variables  
on a binary output.

To classify

the examples by estimating the  
probability of being a particular  
class/category.

## 2. Odds, Log odds, Odds Ratio

# Odds

# Odds

Odds: a numerical expression, expressed as a pair of numbers.

Odds for (odds of) an event measures the likelihood of the event occurring;

Odds against an event measures the likelihood of the event NOT occurring.

EXAMPLE. 1st-class honours

The odds of students graduating with 1<sup>st</sup>-class honours is 1 to 4:



1 of them graduating with 1<sup>st</sup>-class honours.

4 of them not graduating with 1<sup>st</sup>-class honours.

EXAMPLE. 1st-class honours

The odds of students graduating with 1<sup>st</sup>-class honours is 1 to 4:

Alternatively, we can write this as a fraction       $\frac{1}{4} = 0.25$



EXAMPLE.

1st-class honours



odds

1 : 4

vs



probability

1 : 5

EXAMPLE. 1st-class honours

odds(occurring)

$$\frac{\text{Wreath}}{\text{Wreath Wreath Wreath Wreath}} = \frac{1}{4} = 0.25$$

probability(occurring)

$$\frac{\text{Wreath}}{\text{Wreath Wreath Wreath Wreath Wreath}} = \frac{1}{5} = 0.20$$

probability(not occurring)

$$\frac{\text{Wreath Wreath Wreath Wreath}}{\text{Wreath Wreath Wreath Wreath Wreath}} = \frac{4}{5} = \underline{\underline{0.80}}$$

$$\text{probability(not occurring)} = 1 - \text{probability(occurring)} = 1 - \frac{1}{5} = \frac{4}{5} = \underline{\underline{0.80}}$$

EXAMPLE. 1st-class honours

odds(occurring)

$$\frac{\text{Wreath}}{\text{Wreath Wreath Wreath Wreath}} = \frac{1}{4} = 0.25$$

probability(occurring)

$$\frac{\text{Wreath}}{\text{Wreath Wreath Wreath Wreath Wreath}} = \frac{1}{5} = 0.20 \quad p$$

probability(not occurring)

$$\frac{\text{Wreath Wreath Wreath Wreath}}{\text{Wreath Wreath Wreath Wreath Wreath}} = \frac{4}{5} = 0.80 \quad q$$

$\frac{\text{probability(occurring)}}{\text{probability(not occurring)}}$

$$\frac{p}{q} = \frac{0.20}{0.80} = \underline{0.25} \quad \text{or} \quad \frac{p}{1-p} = \frac{1/\cancel{5}}{4/\cancel{5}} = \frac{1}{4} = \underline{0.25}$$

# Log of Odds

EXAMPLE. 1st-class honours

odds(occurring)



$$= \frac{1}{4} = 0.25$$

EXAMPLE. 1st-class honours

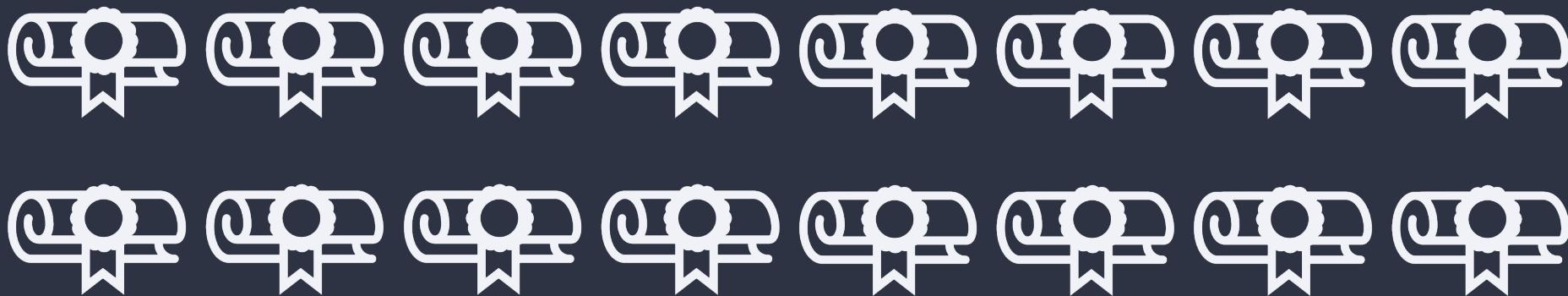
odds(occurring), if students were bad



$$= \frac{1}{8} = 0.125$$

EXAMPLE. 1st-class honours

odds(occurring), if students were terrible



$$= \frac{1}{16} = 0.063$$

EXAMPLE.

1st-class honours

odds(occurring), if students were the worst



---

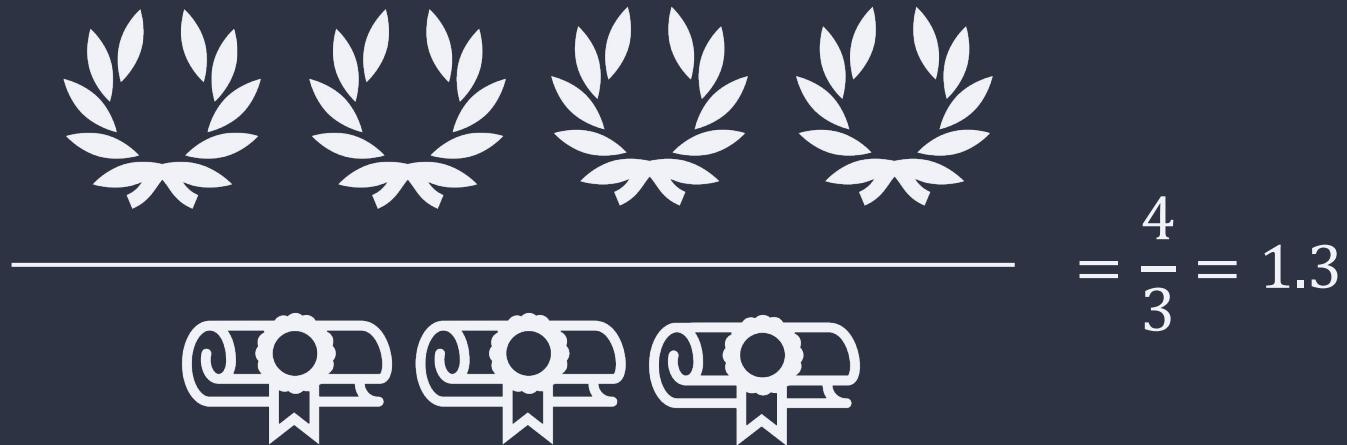
$$= \frac{1}{32} = 0.031$$



Odds against occurring is between 0 and 1

EXAMPLE. 1st-class honours

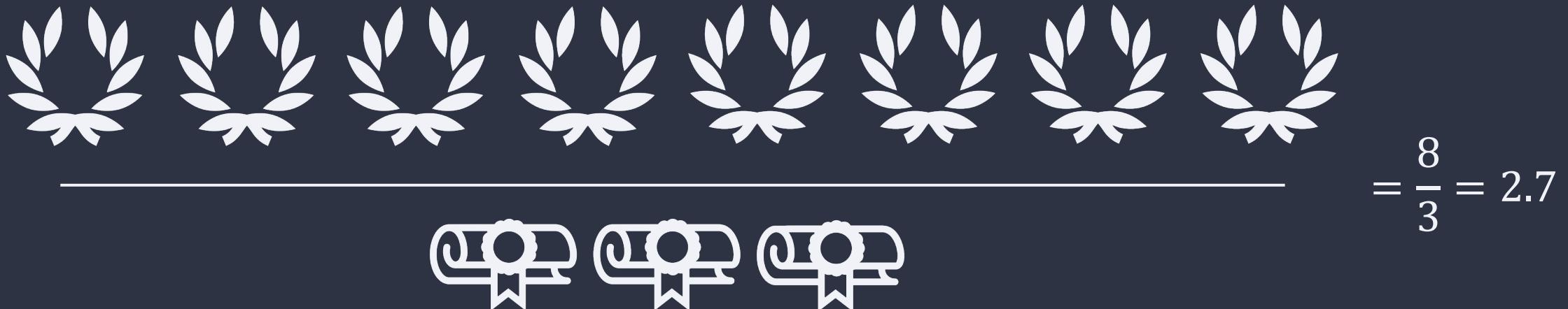
odds(occurring), if students were good

$$\frac{4}{3} = 1.3$$


EXAMPLE.

1st-class honours

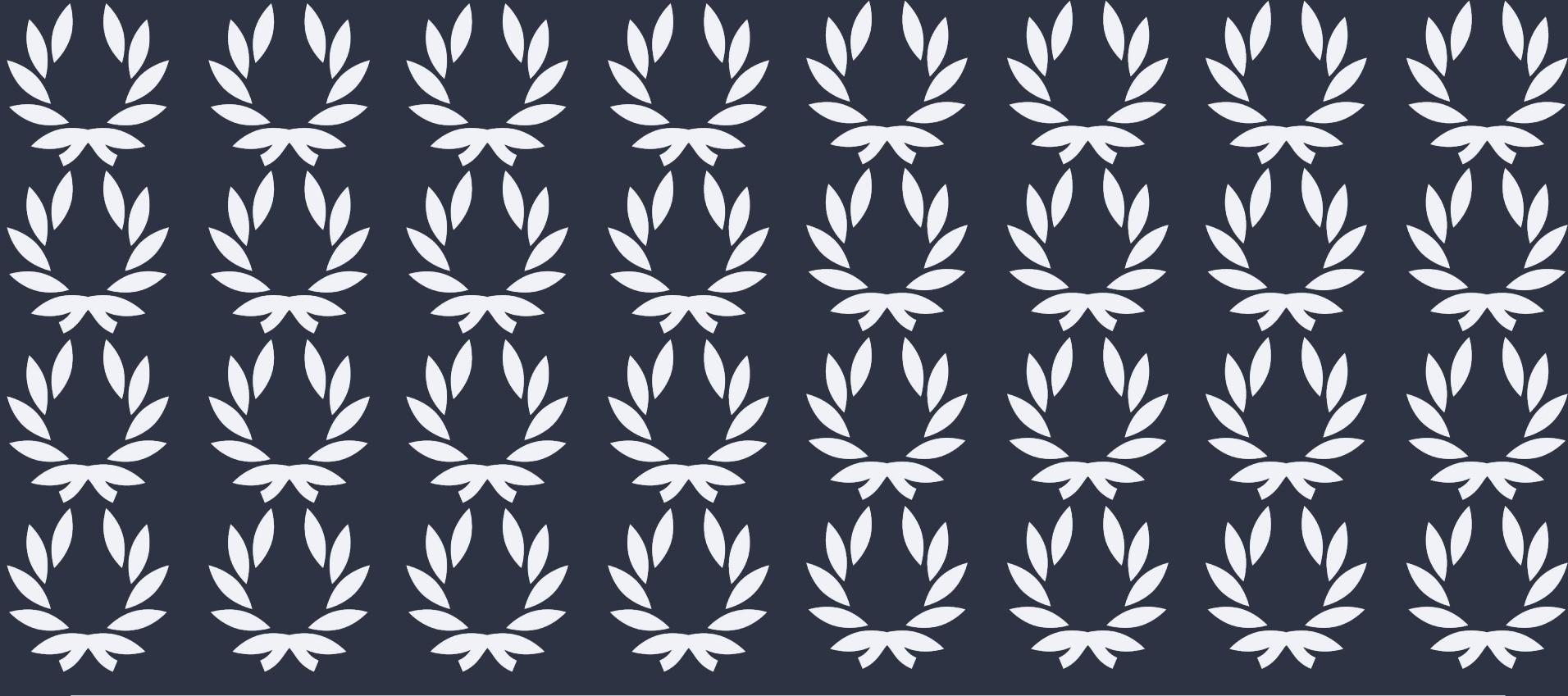
odds(occurring), if students were better



EXAMPLE.

1st-class honours

odds(occurring), if students were really good



$$= \frac{32}{3} = 10.7$$

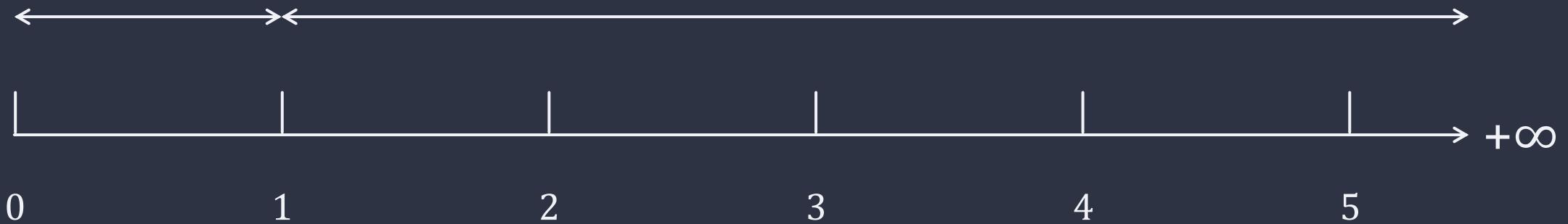


Odds in favour of occurring is between 1 and  $+\infty$

EXAMPLE. 1st-class honours

Odds(not occurring) go from 0 to 1

Odds(occurring) go from 1 to  $+\infty$



The asymmetry makes it difficult to  
compare odds(not occurring) and odds(occurring)

EXAMPLE. 1st-class honours

Taking log of odds (i.e.  $\log(\text{odds})$ ) solves this problem by making everything symmetrical.



e.g. If odds(occurring) is 1:6, then

$$\log(\text{odds}) = \log(1/6) = \log(0.17) = -1.79$$

If odds(occurring) is 6:1, then

$$\log(\text{odds}) = \log(6/1) = \log(6) = 1.79$$

Using  $\log()$ , the distance from the origin, is the same for 1:6 and 6:1 odds.

# Odds, Log of Odds

$$odds(\text{occurring}) = \frac{\text{probability}(\text{occurring})}{\text{probability}(\text{not occurring})} = \frac{\text{probability}(\text{occurring})}{1 - \text{probability}(\text{occurring})}$$

$$odds = \frac{p}{q} = \frac{p}{1-p}$$

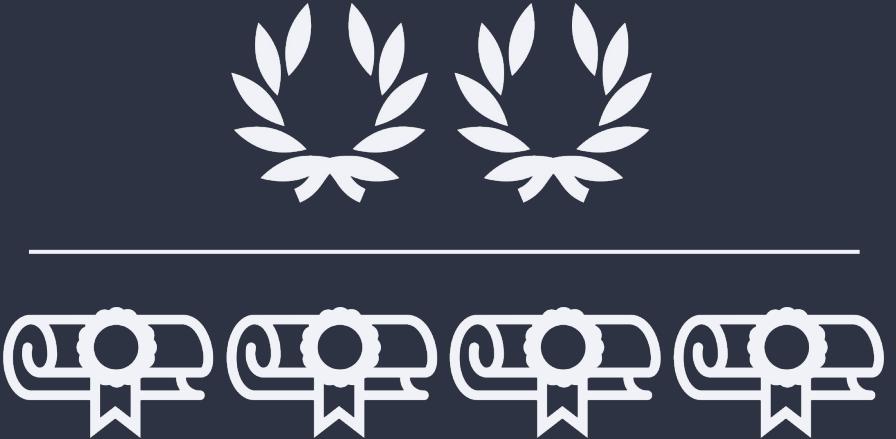
$$\log(\text{odds}) = \log\left(\frac{p}{1-p}\right) \quad \text{Logit equation – forms the basis for Logistic Regression}$$

$\log_e / \ln$   
natural logarithm

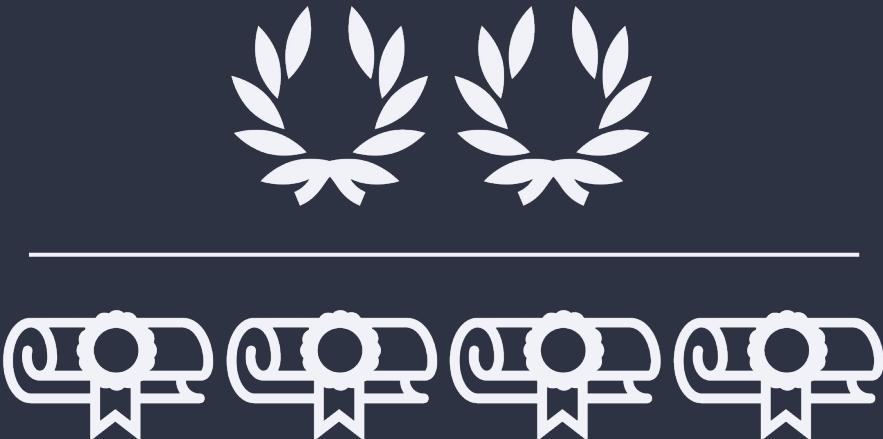
# Odds Ratio

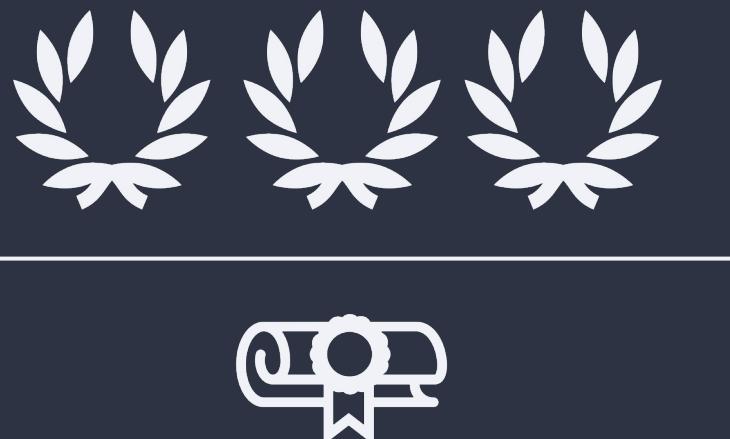
Odds Ratio is just Ratio of Odds

EXAMPLE. 1st-class honours

$$\text{Odds}_1 = \frac{\text{Number of laurels}}{\text{Number of laurels} + \text{Number of diplomas}} = \frac{2}{4} = 0.5$$


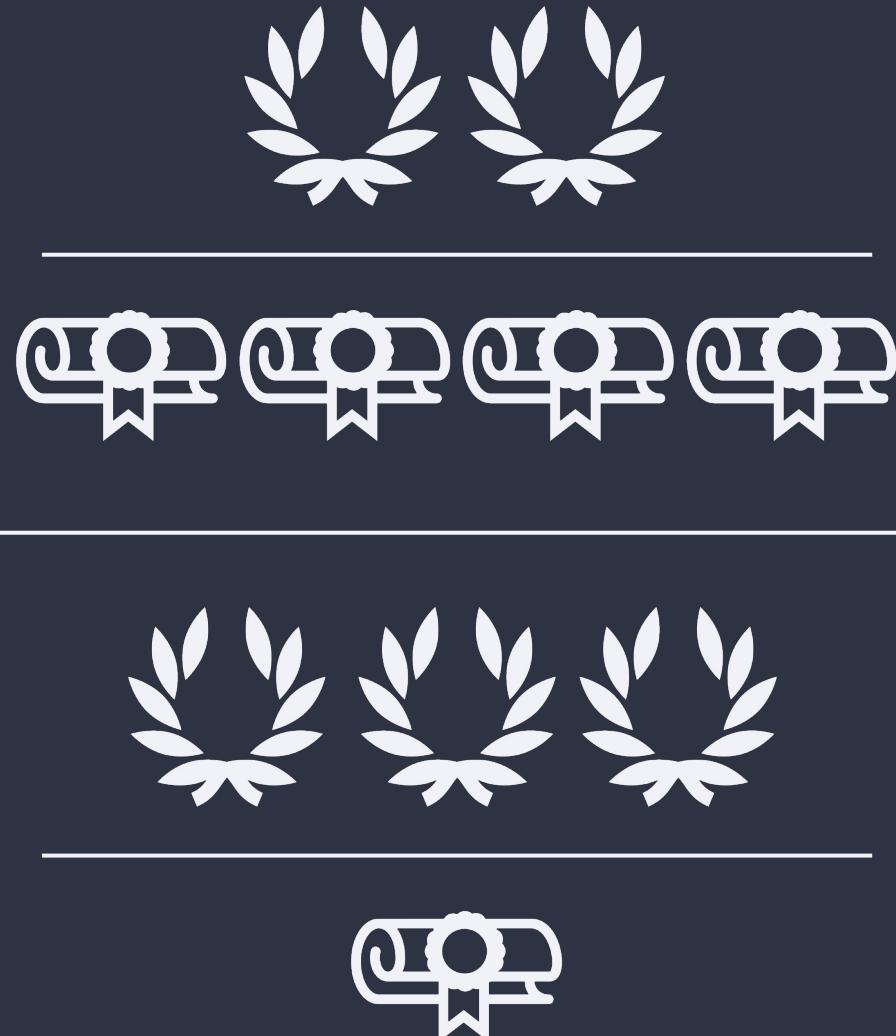
EXAMPLE. 1st-class honours

$$\text{Odds}_1 = \frac{\text{Number of laurels}}{\text{Number of laurels} + \text{Number of diplomas}} = \frac{2}{4} = 0.5$$


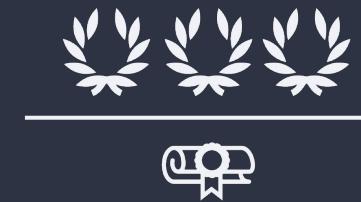
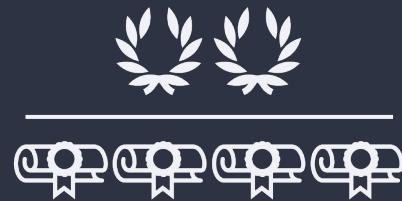
$$\text{Odds}_2 = \frac{\text{Number of laurels}}{\text{Number of laurels} + \text{Number of diplomas}} = \frac{3}{1} = 3$$


EXAMPLE. 1st-class honours

$$\text{Odds Ratio} = \frac{\text{Odds}_1}{\text{Odds}_2} = \frac{0.5}{3} = 0.167$$



EXAMPLE. 1st-class honours



## EXAMPLE. 1st-class honours



Taking log of odds ratio makes things nice and symmetrical.

EXAMPLE. 1st-class honours

How does attending ML lectures predict Graduating with 1<sup>st</sup> Class Honours?

Graduating with 1<sup>st</sup> Class Honours?

Have attended  
more than 75% of  
Machine Learning  
lectures?

Graduating with 1 <sup>st</sup> Class Honours?			
	✓	✗	
✓	27	68	95
✗	8	132	140
	35	200	

EXAMPLE.

1st-class honours

How does attending ML lectures predict Graduating with 1<sup>st</sup> Class Honours?

Graduating with  
1<sup>st</sup> Class Honours?

Have attended  
more than 75% of  
Machine Learning  
lectures?

		Graduating with 1 <sup>st</sup> Class Honours?
		✗
Have attended more than 75% of Machine Learning lectures?	✓	27
	✗	68
		✗
		✓
		8
		132

The odds of graduating with 1<sup>st</sup> Class Honour, if a student has  
27/68 attended >75% ML lectures.

8/132 The odds of not graduating with 1<sup>st</sup> Class Honour, if a student has not attended >75% ML lectures.

EXAMPLE. 1st-class honours

How does attending ML lectures predict Graduating with 1<sup>st</sup> Class Honours?

Graduating with  
1<sup>st</sup> Class Honours?

Have attended  
more than 75% of  
Machine Learning  
lectures?

		Graduating with 1 <sup>st</sup> Class Honours?
		✗
Have attended more than 75% of Machine Learning lectures?	✓	27
	✗	68
		✗
		132
		8

$$\frac{27/68}{8/132} = \frac{0.397}{0.061} = 6.509 \text{ Odds Ratio}$$

$$\log(6.509) = 1.873 > 1 \quad \log(\text{Odds Ratio})$$

Having attended >75% lectures is a strong predictor of a student is graduating with 1<sup>st</sup> Class Honours.

EXAMPLE. 1st-class honours

How does attending ML lectures predict Graduating with 1<sup>st</sup> Class Honours?

		Graduating with 1 <sup>st</sup> Class Honours?	
		✓	✗
Have attended more than 75% of Machine Learning lectures?	✓	27	68
	✗	8	132
		27/68	=

$\frac{27/68}{8/132} = \frac{0.397}{0.061} = 6.509$  Odds Ratio

$\log(6.509) = 1.873 > 1$  log(Odds Ratio)

Odds Ratio > 1, the more likely one event occurring, the more likely the other event occurring.

Odds Ratio = 1 ( $\log(\text{Odds Ratio}) = 0$ ), no relationship / not a predictor

Odds Ratio < 1, the more likely one event occurring, the less likely the other event occurring.

### 3. Hypothesis Function

## What we've known

Event occurring is labelled as 1; event NOT occurring is labelled as 0.

Probability of event occurring is  $p$ ; NOT occurring is  $q = 1 - p$ .

Odds of event occurring is  $odds = \frac{p}{1-p}$

Log of odds is  $\log(odds) = \log(\frac{p}{1-p})$ , called **logit function**.

## What we want

To learn from the training set for a model that estimates  $p(occurring)$  for any given combination of independent variables.

# What we want

To learn from the training set for a model that estimates  $p(\text{occurring})$  for any given combination of independent variables.



$$\left. \begin{aligned} \logit(p) &= \log(\text{odds}) = \log\left(\frac{p}{1-p}\right) \\ \theta^T X &= \logit(p) \end{aligned} \right\} \quad \theta^T X = \log\left(\frac{p}{1-p}\right) \rightarrow p = ?$$

# What we want

$$\theta^T X = \log \left( \frac{p}{1-p} \right) \longrightarrow p = ?$$

$$e^{\theta^T X} = \frac{p}{1-p}$$

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

$$e^{\theta^T X} - pe^{\theta^T X} = p$$

$$e^{\theta^T X} = p + pe^{\theta^T X}$$

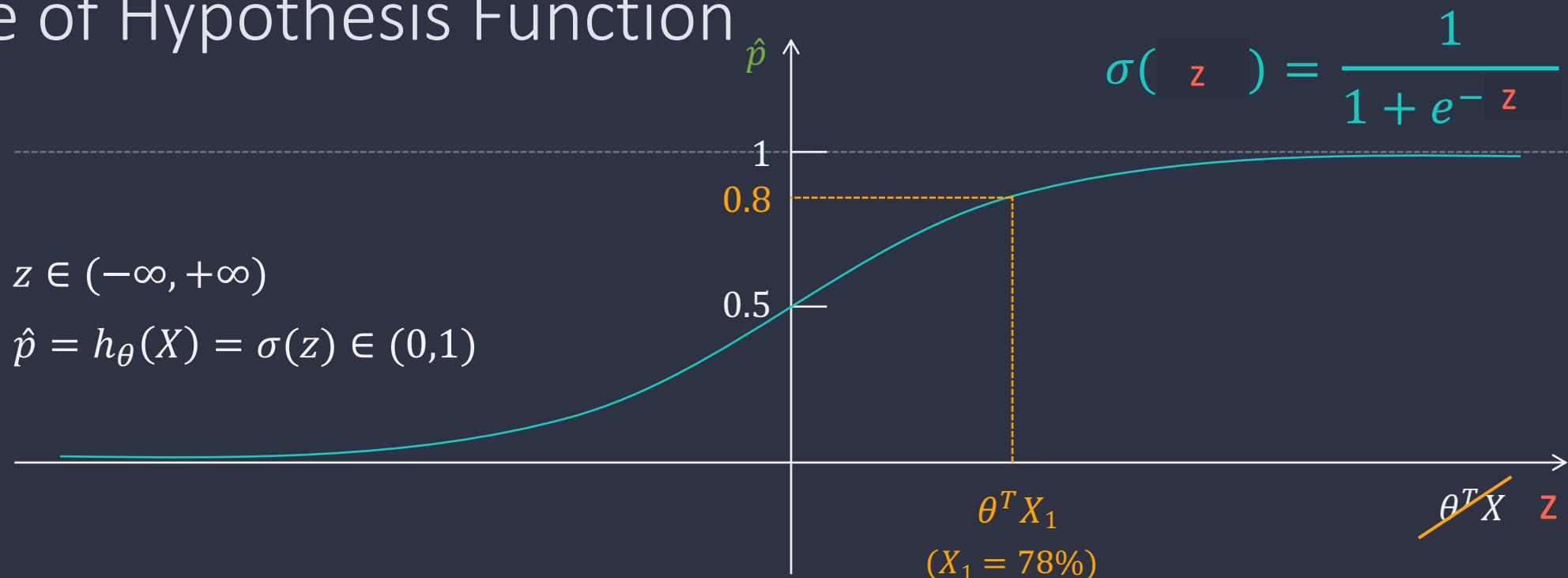
$$e^{\theta^T X} = p(1 + e^{\theta^T X})$$

$$p = \frac{e^{\theta^T X}}{1 + e^{\theta^T X}} = \frac{1}{1 + e^{-\theta^T X}}$$

$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}} \equiv \sigma(\theta^T X)$$

Learning algorithm aims to learn from training set for  $\theta$ , the parameter vector, in order to make predictions.

# Shape of Hypothesis Function



## Interpretation of Hypothesis Output

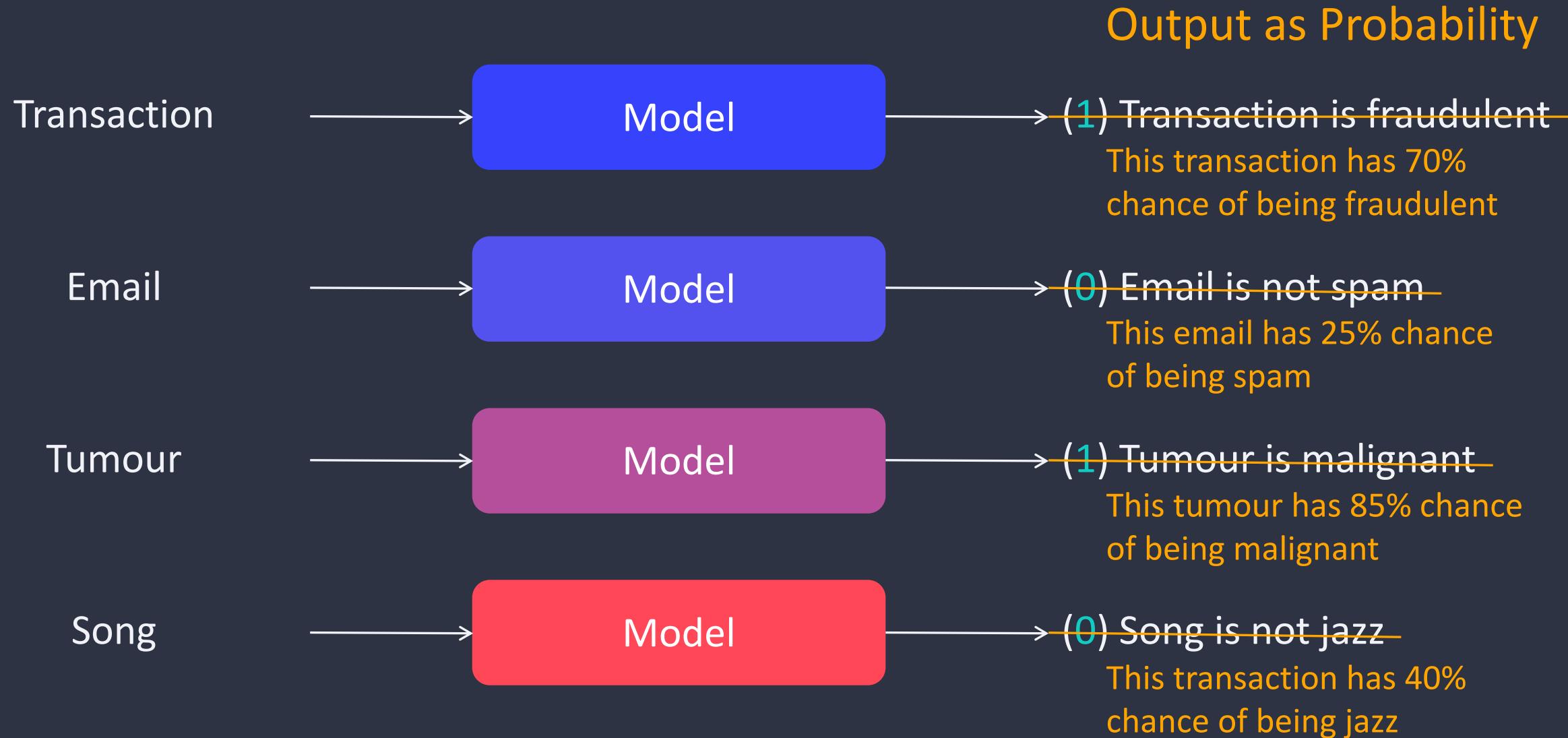
$\hat{p} = h_\theta(X)$  = estimated probability that  $y = 1$  on input  $X$

$\hat{p} = h_\theta(X) = P(y = 1|X; \theta)$  The probability that  $y = 1$ , given  $X$ , parameterised by  $\theta$ .

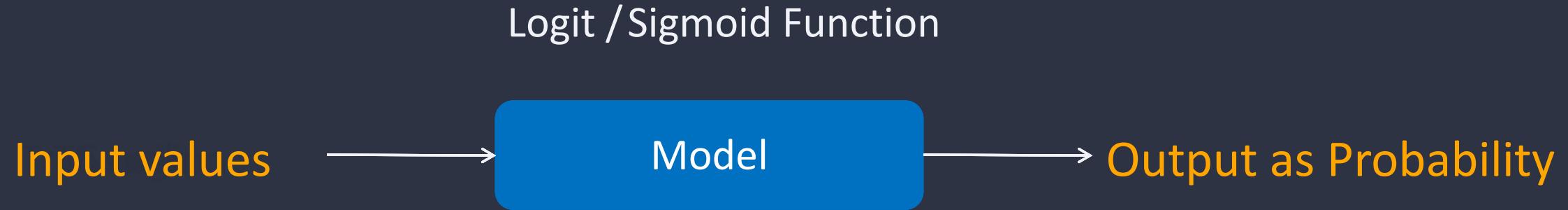
$y \in \{0,1\} \rightarrow P(y = 0|X; \theta) + P(y = 1|X; \theta) = 1 \quad \text{or} \quad P(y = 1|X; \theta) = 1 - P(y = 0|X; \theta)$

## 4. Decision Boundary

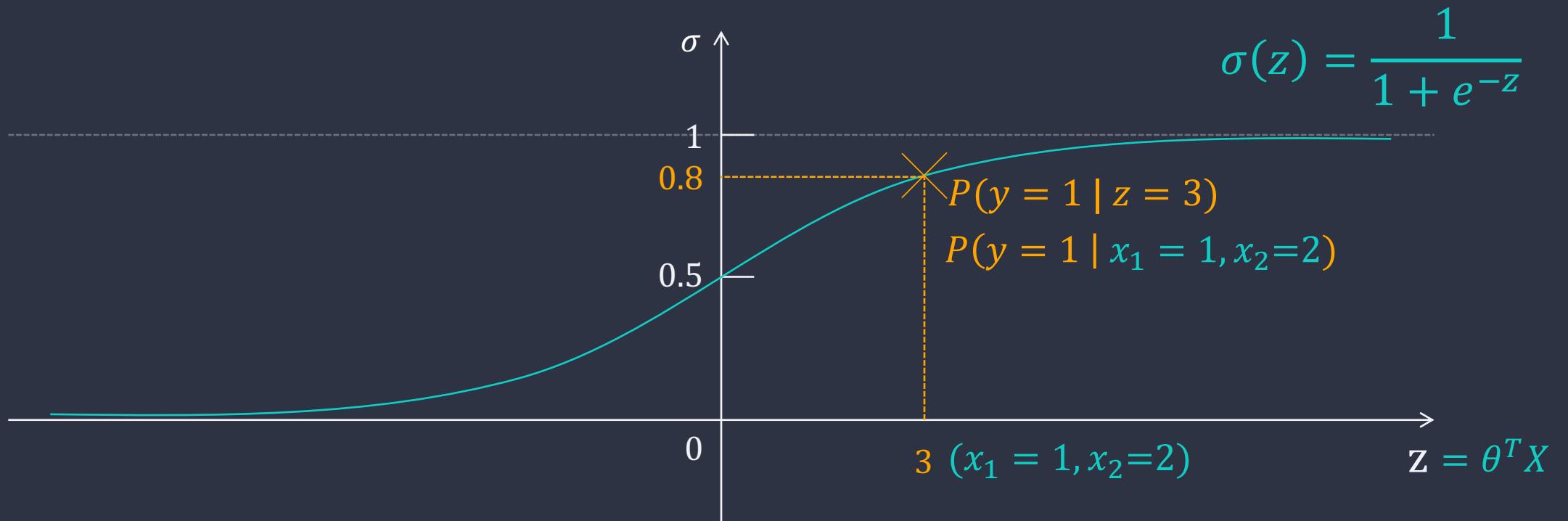
# Logistic Regression for Binary Classification



# Logistic Regression for Binary Classification



# Sigmoid Function

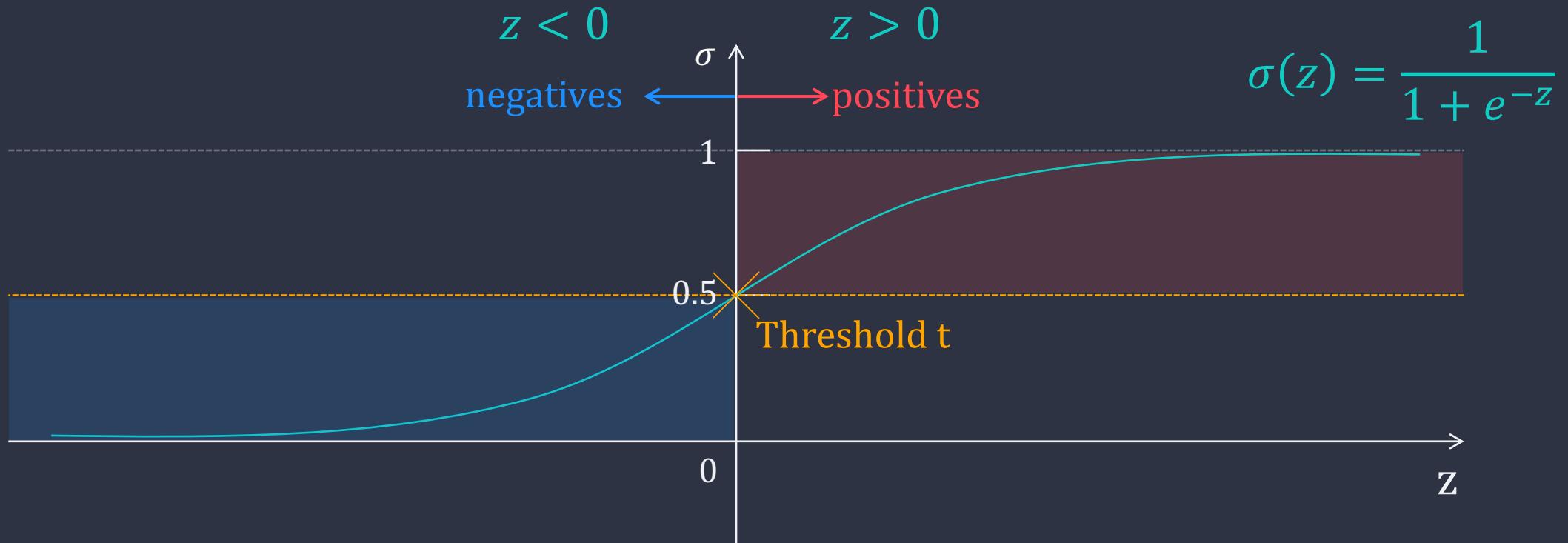


e.g.  $Z = \theta^T X = \theta_0 + \theta_1 x_1 + \theta_2 x_2$      $\theta = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$

$$Z = \theta^T X = -1 + 2x_1 + x_2 = 3$$

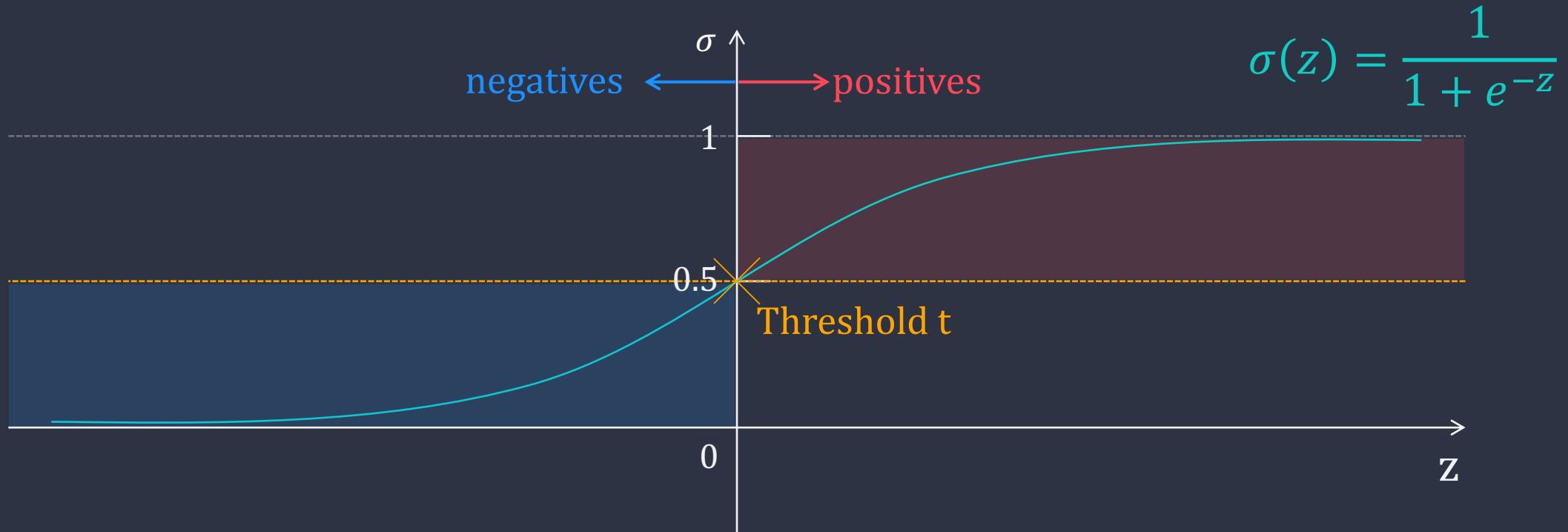
$$x_1 = 1 \quad x_2 = 2$$

# Making Predictions



If we want less false positives, we need to set a \_\_\_\_\_ (larger/smaller) threshold.

# Decision Boundary



Predict  $y = 1$  (positive), if  $z > 0$

$$z = \theta^T X \longrightarrow \theta^T X > 0$$

Predict  $y = 0$  (negative), if  $\theta^T X < 0$

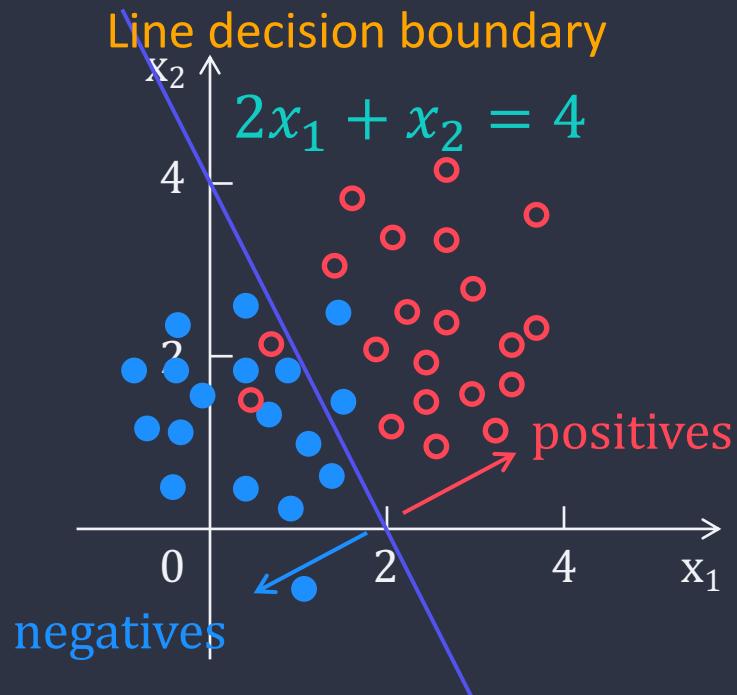
} Decision Boundary  
 $\theta^T X = 0$

# 2-Dimensional Decision Boundary

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

$$\left. \begin{array}{l} \text{Decision Boundary} \\ \theta^T X = 0 \end{array} \right\}$$



$$\begin{aligned} h_{\theta}(X) &= \frac{1}{1 + e^{-\theta^T X}} = \frac{1}{1 + e^{-(-4+2x_1+x_2)}} \\ &= 0 \end{aligned}$$
$$-4 + 2x_1 + x_2 = 0$$

$$2x_1 + x_2 = 4$$

# 1-Dimensional Decision Boundary

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

$$\left. \begin{array}{l} \text{Decision Boundary} \\ \theta^T X = 0 \end{array} \right\}$$

Point decision boundary

$$x = 2$$



negatives  $\longleftrightarrow$  positives

$$\begin{aligned} h_{\theta}(X) &= \frac{1}{1 + e^{-\theta^T X}} = \frac{1}{1 + e^{-(2-x)}} \\ &= 0 &= 0 \end{aligned}$$

$$2 - x = 0$$

$$x = 2$$

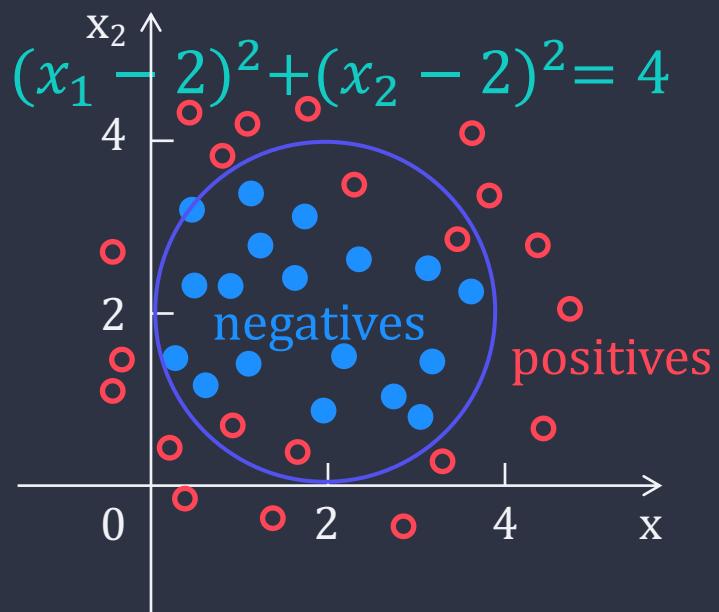
# Non-linear Decision Boundary

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

$$\left. \begin{array}{l} \text{Decision Boundary} \\ \theta^T X = 0 \end{array} \right\}$$

Non-linear decision boundary



$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}} = \frac{1}{1 + e^{-(4 - 4x_1 - 4x_2 + x_1^2 + x_2^2)}} = 0$$
$$4 - 4x_1 - 4x_2 + x_1^2 + x_2^2 = 0$$

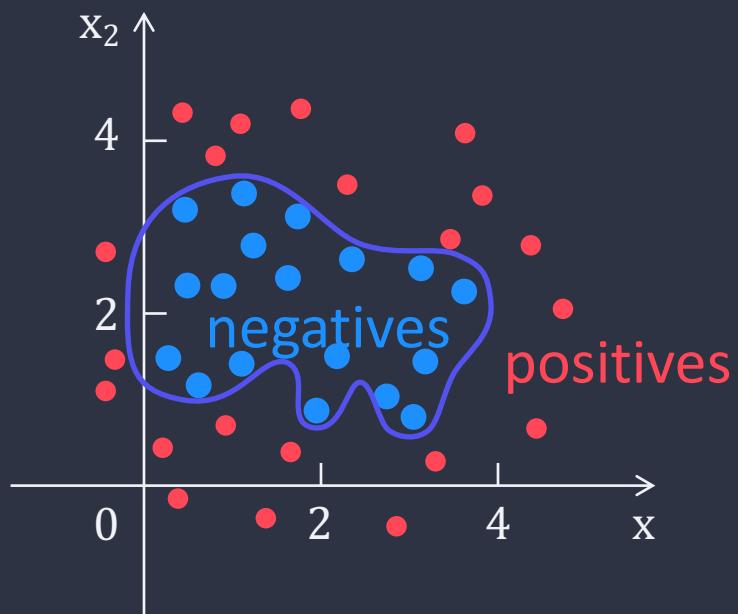
$$(x_1 - 2)^2 + (x_2 - 2)^2 = 4$$

# Non-linear Decision Boundary

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

} Decision Boundary  
 $\theta^T X = 0$



$$h_{\theta}(X) = \frac{1}{1 + e^{-\underline{\theta^T X}}} = 0$$

$$\begin{aligned} \theta^T X &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ &\quad + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \theta_6 x_1^2 x_2^2 \\ &\quad + \theta_7 x_1^3 x_2^2 + \dots \\ &= 0 \end{aligned}$$

# Summary

# Summary (Logistic Regression, Part I)

1. Intuition
2. Odds, Log Odds, Odds Ratio
3. Hypothesis Function
4. Decision Boundary

## Next Lecture (Logistic Regression, Part II)

5. Cost Function
6. Gradient Descent
7. Regularisation
8. Multinomial Logistic Regression (Softmax Regression)

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Logistic Regression (Part II)

Dr Yang Long

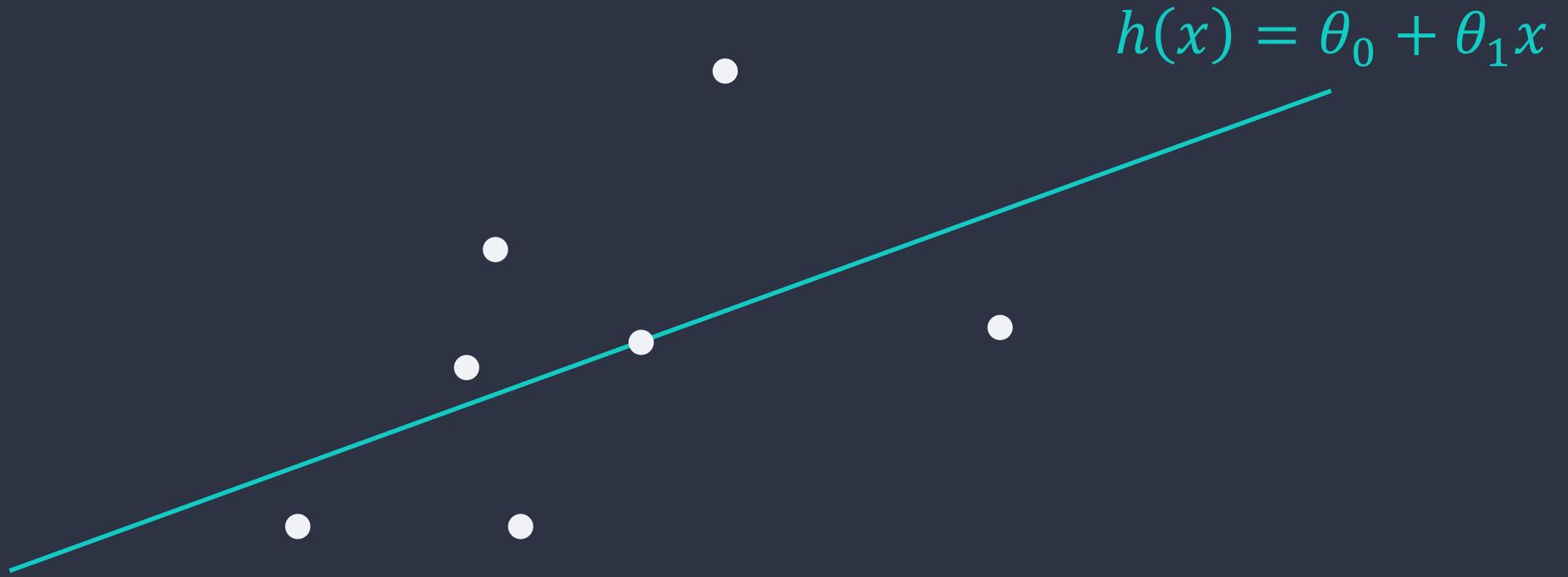
# Lecture Overview (Logistic Regression, Part II)

1. Intuition
2. Odds, Log Odds, Odds Ratio
3. Hypothesis Function
4. Decision Boundary
5. Cost Function
6. Gradient Descent
7. Regularisation
8. Multinomial Logistic Regression (Softmax Regression)

# 1. Cost Function

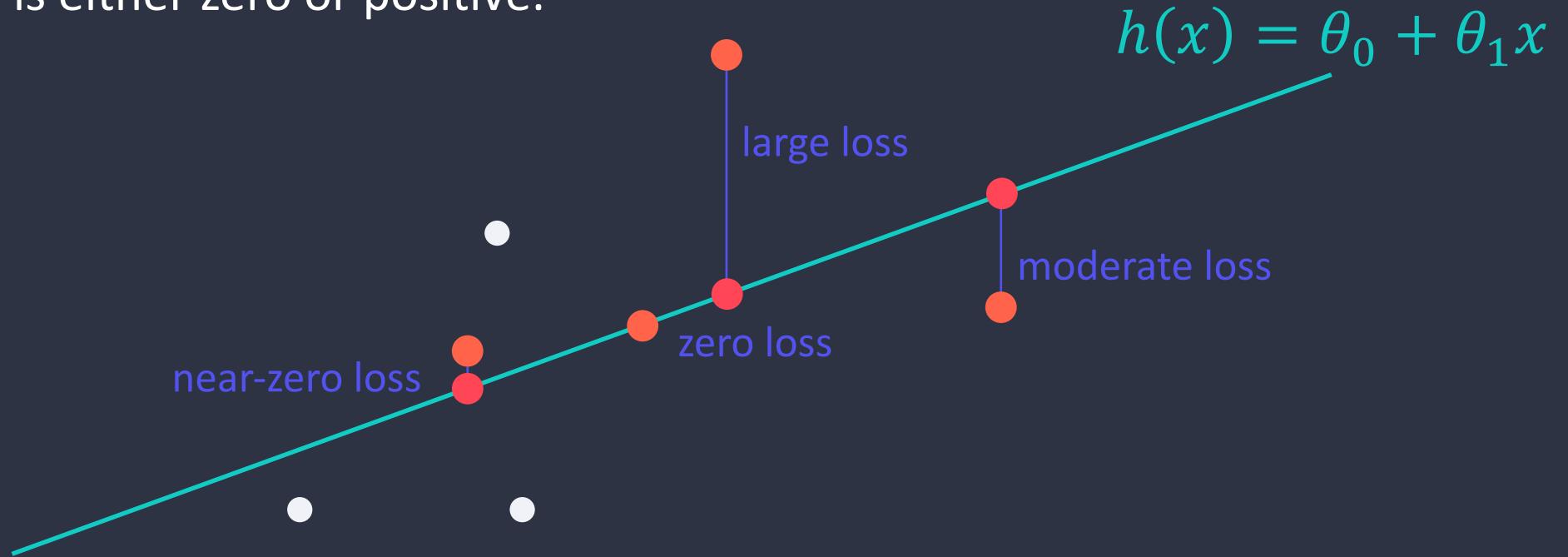
# Cost

- Cost measures how **bad** a **model** performs on the whole training set.



# Cost

- Cost measures how bad a **model** performs on the **whole** training set.
- Loss (error) measures how bad a **model** performs on one **single** training example.
  - The difference between the **prediction** and **actual value** for **a given x**.
  - Value is either zero or positive.



# Cost for Linear Regression

## Mean Squared Error (MSE)

$$J = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

$m$  the number of examples in the training set

$i$  the  $i$ -th example

$y^{(i)}$  the actual value of the  $i$ -th example

$\hat{y}^{(i)} = \theta^T X^{(i)}$  the predicted value for the  $i$ -th example

$loss = y^{(i)} - \hat{y}^{(i)}$  the loss/error for the  $i$ -th example



convergence at the minima

# Cost for Logistic Regression

# Cost for Logistic Regression

Estimation  $\hat{y} = \frac{1}{1 + e^{-\theta^T X}}$  (estimated probability of X occurring)

$loss^{(i)} = \hat{y}^{(i)} - y^{(i)}$  (how close the predicted value (estimation) is to the actual value)

Given  $\{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(m)}, y^{(m)})\}$ , want  $loss^{(i)} \approx 0$

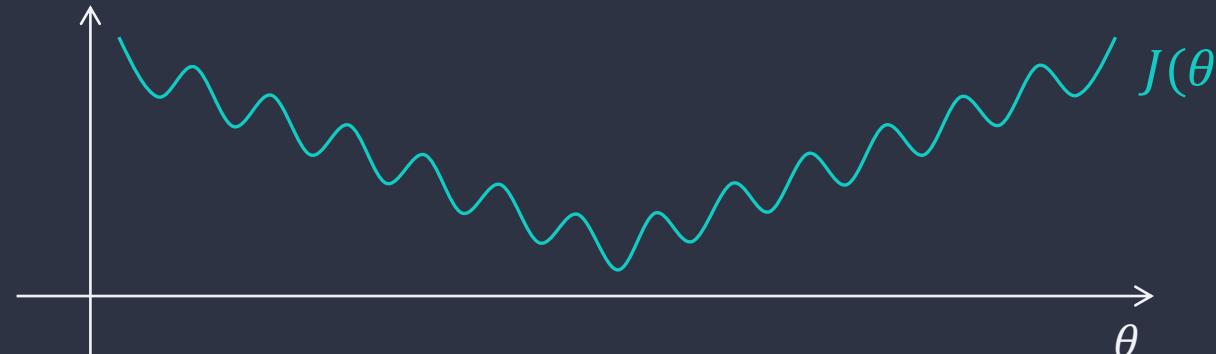
Loss Function ?

$$loss^{(i)} = \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2$$

$$\hat{y} = \frac{1}{1 + e^{-\theta^T X}} \quad (\text{non-linear})$$

Cost Function ?  $J = \frac{1}{m} \sum_{i=1}^m loss^{(i)}$

non-convex



# Logistic Regression

$$\hat{y} = \frac{1}{1 + e^{-\theta^T X}}$$

$$\hat{y} = \begin{cases} 1 & \text{for occurring} \\ 0 & \text{for not occurring} \end{cases}$$

(binary classification)

## Probability Function

- $P(y = 1|X; \theta) = \hat{y}$  Probability of  $y = 1$  (*occurring*), given  $X$ , parameterised by  $\theta$ .
- $P(y = 0|X; \theta) = 1 - \hat{y}$  Probability of  $y = 0$  (*not occurring*), given  $X$ , parameterised by  $\theta$ .

## Likelihood (conditional on $X$ )

$$P(y = y|X; \theta) = \frac{P(y = 1|X; \theta)^y \cdot P(y = 0|X; \theta)^{1-y}}{\hat{y}^y \cdot (1 - \hat{y})^{1-y}}$$

$$P(y = 1|X; \theta) = \hat{y}^1 \cdot (1 - \hat{y})^{1-1} = \hat{y}$$

$$P(y = 0|X; \theta) = \hat{y}^0 \cdot (1 - \hat{y})^{1-0} = 1 - \hat{y}$$

Likelihood (conditional on  $X$ )

$$P(y = y|X; \theta) = \hat{y}^y \cdot (1 - \hat{y})^{1-y}$$

Likelihood (conditional on  $X^{(i)}, i = 1, 2, \dots, m$ )

Assuming independence of examples in training set

$$\mathcal{L}(\theta) = \prod_{i=1}^m \hat{y}^{(i)} y^{(i)} (1 - \hat{y}^{(i)})^{1-y^{(i)}}$$

$$\boxed{\hat{y}^{(i)} = \frac{1}{1 + e^{-\theta^T X^{(i)}}}}$$

$(X^{(i)}, y^{(i)})$  is the  $i$ -th example (data, constant) from the training set.

$\mathcal{L}(\theta)$  is a function of  $\theta$ , i.e.,  $\theta$  is the only independent variable;  $\mathcal{L}$  is the dependent variable.

To fit  $\theta$  that minimises overall cost of logistic regression model  $\ell(y^{(i)}, \hat{y}^{(i)}), i = 1, 2, \dots, m$

→ To fit  $\theta$  that maximises the likelihood for all conditions  $\mathcal{L}(\theta)$

Maximum Likelihood Estimation (MLE)

Likelihood (conditional on  $X^{(i)}, i = 1, 2, \dots, m$ )

$$\mathcal{L}(\theta) = \left[ \prod_{i=1}^m \hat{y}^{(i)y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}} \right]$$

As  $\log$  is monotone increasing, maximising  $\mathcal{L}(\theta)$  is the same as maximising  $\log(\mathcal{L}(\theta))$ .

Log Likelihood (conditional on  $X^{(i)}, i = 1, 2, \dots, m$ )

$$\log(\mathcal{L}(\theta)) = \log \left( \prod_{i=1}^m \hat{y}^{(i)y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}} \right)$$

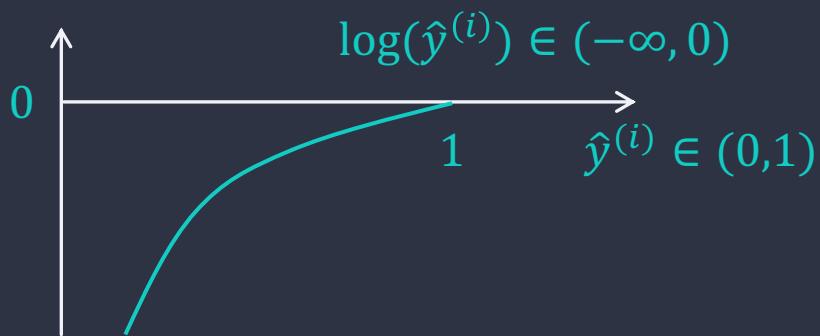
$$= \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

## Likelihood (conditional on $X^{(i)}, i = 1, 2, \dots, m$ )

$$\ell(\theta) = \sum_{i=1}^m \begin{cases} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \\ = 0, \text{ if } y^{(i)} = 0 \\ = 0, \text{ if } y^{(i)} = 1 \end{cases}$$

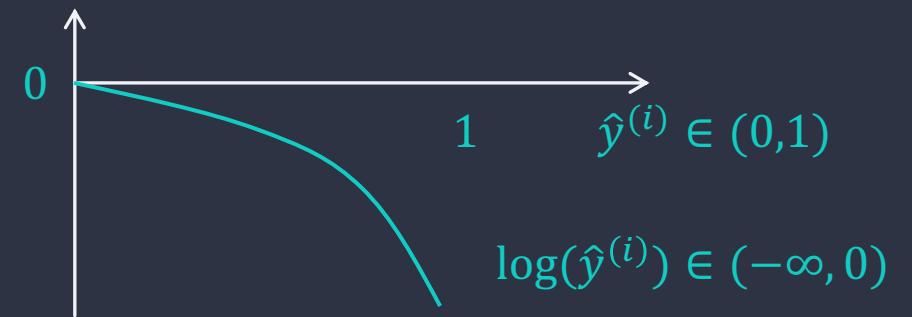
If  $y^{(i)} = 1$ ,  $\ell(\theta) = \sum_{i=1}^m \log(\hat{y}^{(i)})$

If  $y^{(i)} = 0$ ,  $\ell(\theta) = \sum_{i=1}^m \log(1 - \hat{y}^{(i)})$



$\hat{y}^{(i)} = 1$  (estimated)     $y^{(i)} = 1$  (actual)     $\max \ell(\theta)$

$\hat{y}^{(i)} \rightarrow 0$  (estimated)     $y^{(i)} = 1$  (actual)     $\ell(\theta) \rightarrow -\infty$



$\hat{y}^{(i)} = 1$  (estimated)     $y^{(i)} = 0$  (actual)     $\ell(\theta) \rightarrow -\infty$

$\hat{y}^{(i)} \rightarrow 0$  (estimated)     $y^{(i)} = 0$  (actual)     $\max \ell(\theta)$

## Log Likelihood (conditional on $X^{(i)}, i = 1, 2, \dots, m$ )

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \\&= \sum_{i=1}^m y^{(i)} (\log \hat{y}^{(i)} - \log(1 - \hat{y}^{(i)})) + \log(1 - \hat{y}^{(i)}) \\&= \sum_{i=1}^m y^{(i)} \log\left(\frac{\hat{y}^{(i)}}{1 - \hat{y}^{(i)}}\right) + \log(1 - \hat{y}^{(i)}) \\&= \sum_{i=1}^m y^{(i)} \log\left(\frac{1}{1 + e^{-\theta^T X^{(i)}}}\right) + \log\left(1 - \frac{1}{1 + e^{-\theta^T X^{(i)}}}\right) \\&= \sum_{i=1}^m y^{(i)} \log\left(\frac{1}{e^{-\theta^T X^{(i)}}}\right) + \log\left(\frac{e^{-\theta^T X^{(i)}}}{1 + e^{-\theta^T X^{(i)}}}\right) \\&= \sum_{i=1}^m y^{(i)} \log(e^{\theta^T X^{(i)}}) + \log\left(\frac{1}{e^{\theta^T X^{(i)}} + 1}\right) \\&= \sum_{i=1}^m y^{(i)} (\theta^T X^{(i)}) - \log\left(1 + e^{\theta^T X^{(i)}}\right)\end{aligned}$$

$$\hat{y}^{(i)} = \frac{1}{1 + e^{-\theta^T X^{(i)}}}$$

Log Likelihood (conditional on  $X^{(i)}, i = 1, 2, \dots, m$ )

$$\ell(\theta) = \sum_{i=1}^m y^{(i)} (\theta^T X^{(i)}) - \log(1 + e^{\theta^T X^{(i)}})$$

Cost Function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} (\theta^T X^{(i)}) - \log(1 + e^{\theta^T X^{(i)}})$$

$$= \frac{1}{m} \sum_{i=1}^m \log(1 + e^{\theta^T X^{(i)}}) - y^{(i)} (\theta^T X^{(i)})$$

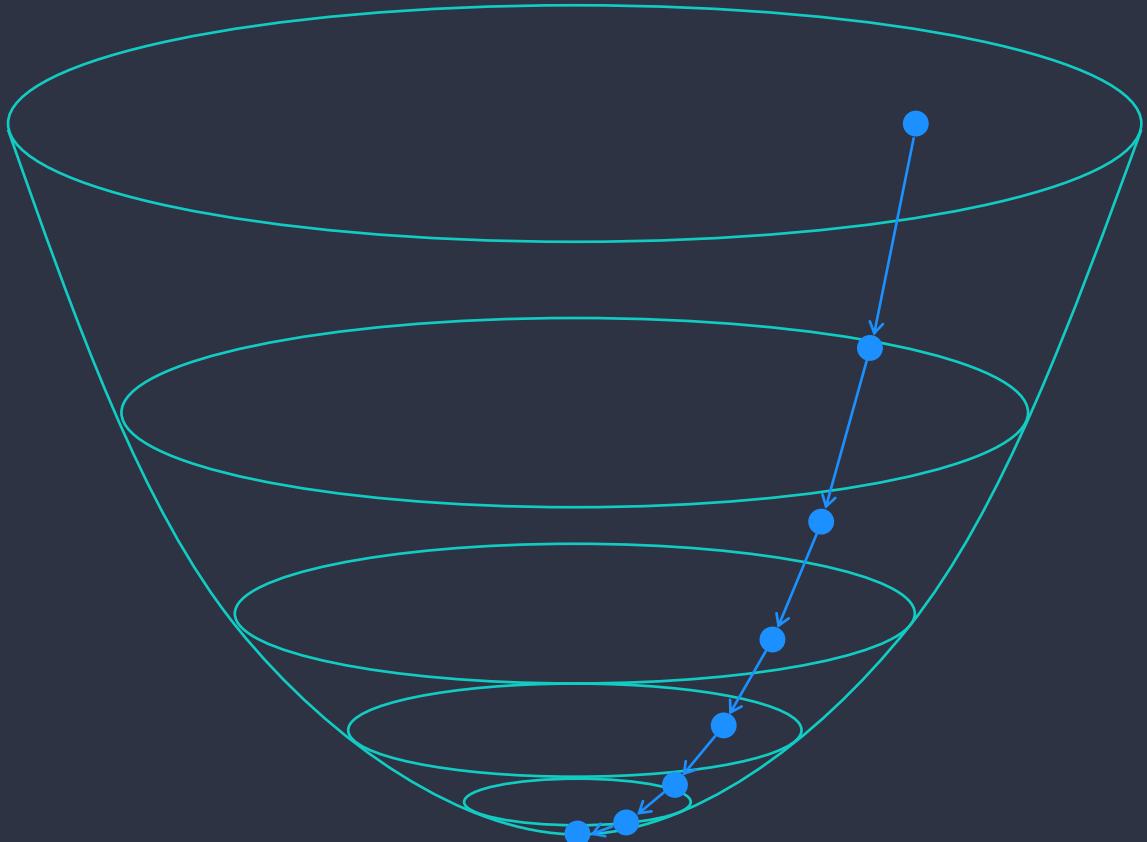
# Cost Function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \log\left(1 + e^{\theta^T X^{(i)}}\right) - y^{(i)}(\theta^T X^{(i)})$$

Aim to fit  $\theta$  that minimises  $J(\theta)$

## 2. Gradient Descent

# Gradient Descent



- In order to find the model parameter vector that minimises the cost function  $J(\theta)$ , we iteratively try different  $\theta$ s in our learning algorithm, and gradient descent can help decide which  $\theta$ s to try and when to stop.
- In order for a gradient descent learning algorithm to converge, i.e. reach the minimum of the cost function, as quickly as possible, we need to change the model parameters in the direction of the steepest slope.

# Gradient Descent for Logistic Regression

- Cost Function for Logistic Regression model is convex.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \log\left(1 + e^{\theta^T X^{(i)}}\right) - y^{(i)}(\theta^T X^{(i)})$$

Convex: has just one minimum; no local minima to get stuck in.

Gradient descent can start any point; guaranteed to find the minimum.

- The magnitude of the amount to update  $\theta$  in gradient descent is the value of the slope

$$\nabla J = \left\langle \frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_n} \right\rangle \text{ weighted by a learning rate } \alpha.$$

# Calculus refresher

- Constant Rule

$$\frac{d}{dx} c = 0$$

- Chain Rule

$$\frac{d}{dx} f(g(x)) = \frac{d}{dx} f(g) \cdot \frac{d}{dx} g(x)$$

- Constant Multiple Rule

$$\frac{d}{dx} cf(x) = c \cdot \frac{d}{dx} f(x)$$

- Derivative of  $e^x$

$$\frac{d}{dx} e^x = e^x$$

- Sum Rule

$$\frac{d}{dx} (f(x) \pm g(x)) = \frac{d}{dx} f(x) \pm \frac{d}{dx} g(x)$$

- Derivative of  $\ln(x)$

$$\frac{d}{dx} \ln(x) = \frac{1}{x}$$

# Partial Derivative

Cost Function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \log\left(1 + e^{\theta^T X^{(i)}}\right) - y^{(i)}(\theta^T X^{(i)})$$

Partial derivative  
with respect to  $\theta_j$

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{1}{m} \sum_{i=1}^m \left[ \frac{\partial}{\partial \theta_j} \log\left(1 + e^{\theta^T X^{(i)}}\right) - \frac{\partial}{\partial \theta_j} y^{(i)}(\theta^T X^{(i)}) \right] \\ &= \frac{1}{m} \sum_{i=1}^m \left[ \frac{e^{\theta^T X^{(i)}}}{1 + e^{\theta^T X^{(i)}}} \cdot \frac{\partial}{\partial \theta_j} \theta^T X^{(i)} - y^{(i)} \cdot \frac{\partial}{\partial \theta_j} (\theta^T X^{(i)}) \right] \\ &= \frac{1}{m} \sum_{i=1}^m \left[ \frac{e^{\theta^T X^{(i)}}}{1 + e^{\theta^T X^{(i)}}} - y^{(i)} \right] \cdot \frac{\partial}{\partial \theta_j} (\theta^T X^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m \left[ \frac{1}{1 + e^{-\theta^T X^{(i)}}} - y^{(i)} \right] \cdot x_j^{(i)}\end{aligned}$$

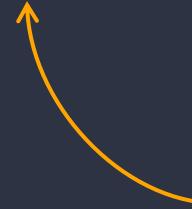
# Gradient Descent for Logistic Regression

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Simultaneously update all  $\theta_j, j = 0, 1, \dots, n$

}



$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[ \frac{1}{1 + e^{-\theta^T X^{(i)}}} - y^{(i)} \right] \cdot x_j^{(i)}$$

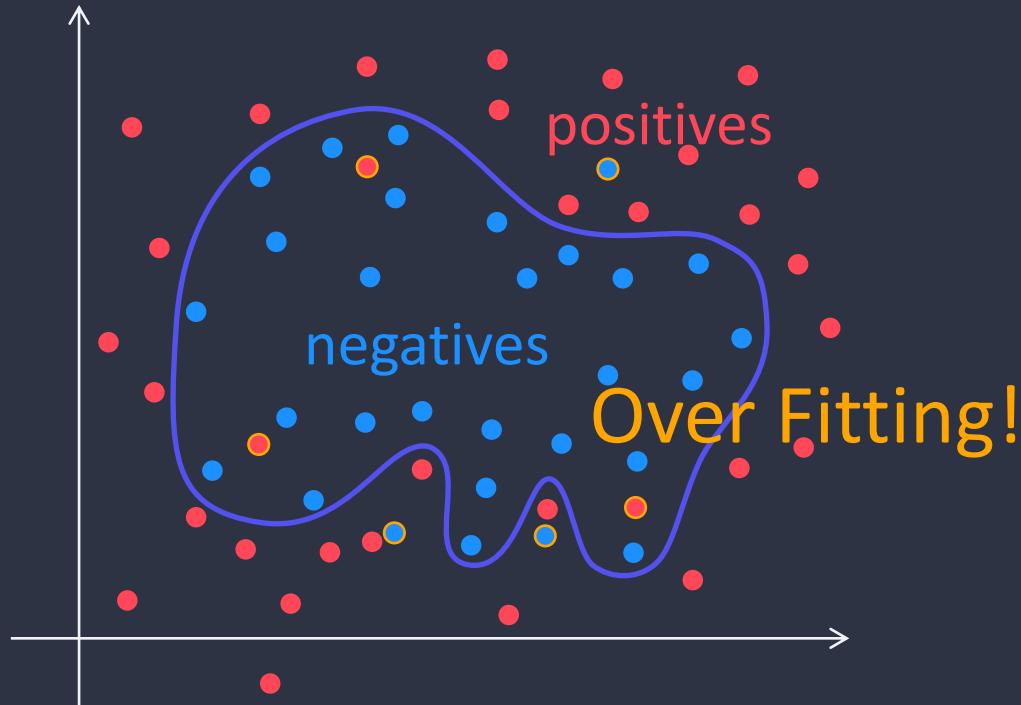
# 3. Regularisation

# Overfitting

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

} Decision Boundary  
 $\theta^T X = 0$



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

$$\begin{aligned}\theta^T X &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ &+ \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \theta_6 x_1^2 x_2^2 \\ &+ \theta_7 x_1^3 x_2^2 + \dots\end{aligned}$$

# Regularisation

Adding regularisation term  $R(\theta)$  to avoid overfitting,

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \log\left(1 + e^{\theta^T X^{(i)}}\right) - y^{(i)}(\theta^T X^{(i)}) + \lambda R(\theta)$$

Ridge Regression / L2 Regularisation  $R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^n \theta_j^2$

Prefers many small parameters.

Easier to optimise – derivative of  $\theta^2$  is  $2\theta$ .

LASSO Regression / L1 Regularisation  $R(\theta) = \|\theta\|_1 = \sum_{j=1}^n |\theta_j|$

Prefer sparse parameters – feature selection.

More complex to optimise – derivative of  $|\theta|$  is non-continuous at 0.

# Gradient Descent with Regularisation

Repeat until convergence {

$$\theta_j := \theta_j - \boxed{\alpha} \cdot \frac{1}{m} \sum_{i=1}^m \left[ \frac{1}{1 + e^{-\theta^T X^{(i)}}} - y^{(i)} \right] \cdot x_j^{(i)}$$

}

Simultaneously update

$$\theta_j, j = 0, 1, \dots, n$$

# Gradient Descent with Regularisation

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^m \left[ \frac{1}{1 + e^{-\theta^T X^{(i)}}} - y^{(i)} \right] \cdot x_0^{(i)}$$

Partial derivative of  $\lambda R(\theta)$   
with respect to  $\theta_j$

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \left[ \sum_{i=1}^m \left[ \frac{1}{1 + e^{-\theta^T X^{(i)}}} - y^{(i)} \right] \cdot x_j^{(i)} + \boxed{\lambda \frac{\partial}{\partial \theta_j} R(\theta)} \right]$$

}

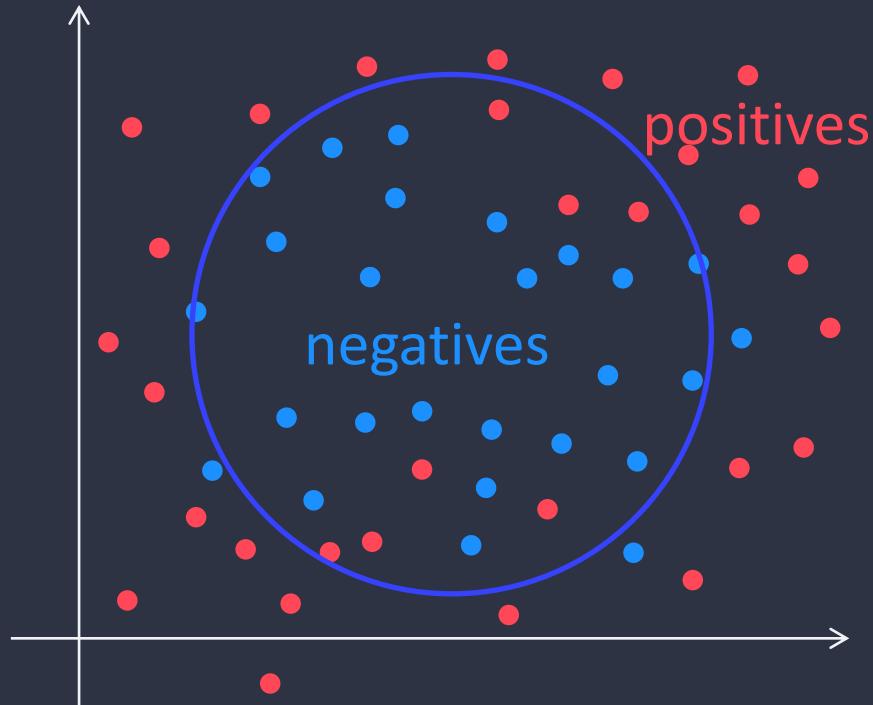
Simultaneously update  
 $\theta_j, j = 0, 1, \dots, n$

# Overfitting avoided

Predict  $y = 1$  (**positive**), if  $\theta^T X > 0$

Predict  $y = 0$  (**negative**), if  $\theta^T X < 0$

} Decision Boundary  
 $\theta^T X = 0$



Less complicated decision boundary

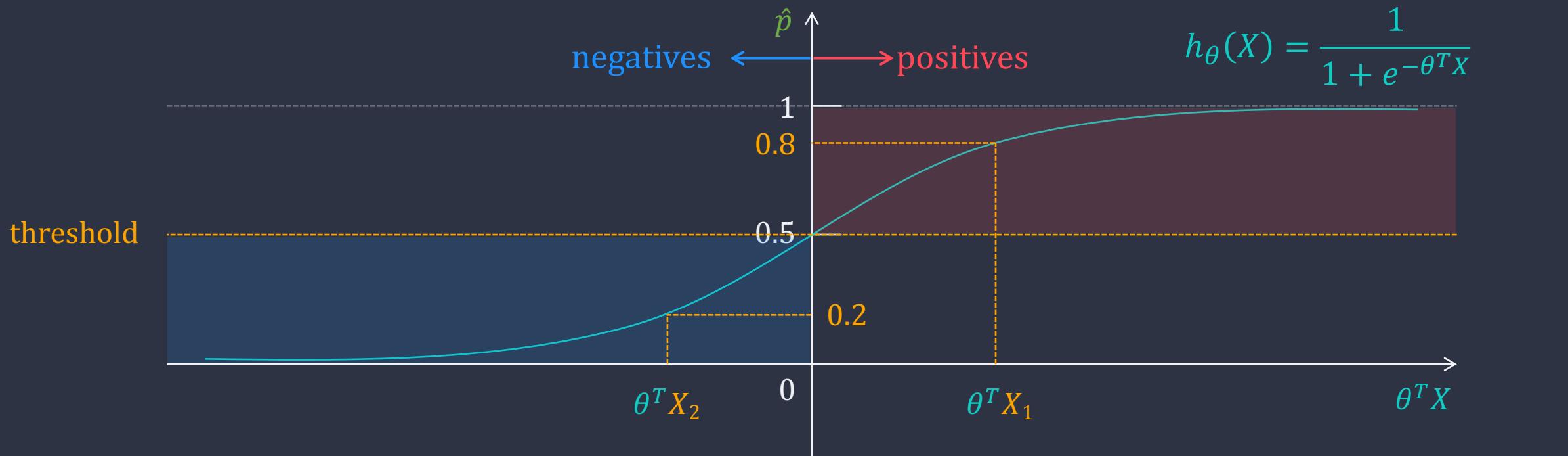
$$h_{\theta}(X) = \frac{1}{1 + e^{-\theta^T X}}$$

$$\begin{aligned}\theta^T X &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ &\quad + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \theta_6 x_1^2 x_2^2 \\ &\quad + \theta_7 x_1^3 x_2^2 + \dots\end{aligned}$$

With penalised  $\theta$ s

# 4. Multinomial Logistic Regression (Softmax Regression)

# Logistic Regression



$\hat{p} = h_{\theta}(X) = p(y = 1|X; \theta)$  The probability that  $y = 1$ , given  $X$ , parameterised by  $\theta$ .

$$\hat{p} = h_{\theta}(X_1) = 0.8 > 0.5 \longrightarrow y = 1 \text{ (positive class)}$$

$$\hat{p} = h_{\theta}(X_2) = 0.2 < 0.5 \longrightarrow y = 0 \text{ (negative class)}$$

Binary Classification  $y \in \{1,0\}$

# Multinomial Logistic Regression / Soft Regression

Generalises Logistic Regression to solve **multi-class classification** problems.

Training Set

$$\{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(m)}, y^{(m)})\} \quad X^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \quad y^{(i)} \in \{0,1\}^K \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow c_2$$

To model the probability  $P(y = c_k | X; \{\theta\})$  of  $y$  being in each potential class  $c_k \in C = \{1, 2, \dots, K\}$ .  $K$  is the number of potential classes (labels).

Multinomial Logistic Classifier uses a generalisation of the sigmoid function, called the **Softmax Function**, to compute the probability.

# Softmax Function

Takes a vector  $\mathbf{z} = [z_1, z_2, \dots, z_K]$  of  $K$  arbitrary values and maps them to a probability distribution, with each value  $(0,1)$ , and all values summing to 1.

For a vector  $\mathbf{z}$  of dimensionality  $K$ , the softmax is

$$S_k = \sigma(z_i) = \frac{e^{z_k}}{\sum_{l=1}^K e^{z_l}} \quad 1 \leq k \leq K \quad z_k = \theta^{(k)^T} X \quad \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$$

normalisation

Model parameters

For an input vector  $\mathbf{z} = [z_1, z_2, \dots, z_K]$ , the softmax is

$$\mathbf{S} = \sigma(\mathbf{z}) = [S_1, S_2, \dots, S_K]^T = \left[ \frac{e^{z_1}}{\sum_{l=1}^K e^{z_l}}, \frac{e^{z_2}}{\sum_{l=1}^K e^{z_l}}, \dots, \frac{e^{z_K}}{\sum_{l=1}^K e^{z_l}} \right]^T$$

e.g.  $\sigma\left(\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}\right) \xrightarrow{\text{Soft-Max}} \begin{bmatrix} .032 \\ .087 \\ .237 \\ .644 \end{bmatrix}$   $(.032 + .087 + .237 + .644 = 1)$

# Softmax Function

$$S_k = \sigma(z_i) = \frac{e^{z_k}}{\sum_{l=1}^K e^{z_l}} \quad 1 \leq k \leq K \quad z_k = \theta^{(k)^T} X$$

$$S = \sigma(z) = [S_1, S_2, \dots, S_K]^T = \left[ \frac{e^{z_1}}{\sum_{l=1}^K e^{z_l}}, \frac{e^{z_2}}{\sum_{l=1}^K e^{z_l}}, \dots, \frac{e^{z_K}}{\sum_{l=1}^K e^{z_l}} \right]^T$$

$$P(y = c_k | X; \{\theta\}) = \sigma \left( \begin{bmatrix} \theta^{(1)^T} X \\ \theta^{(2)^T} X \\ \vdots \\ \theta^{(K)^T} X \end{bmatrix} \right)_k = \frac{e^{(\theta^{(k)^T} X)}}{\sum_{l=1}^K e^{(\theta^{(l)^T} X)}}$$

Different  $\theta$  for each class

$$\begin{bmatrix} \theta^{(1)^T} X \\ \theta^{(2)^T} X \\ \vdots \\ \theta^{(K)^T} X \end{bmatrix} = \begin{bmatrix} -\theta^{(1)} & - \\ -\theta^{(2)} & - \\ \vdots & \\ -\theta^{(K)} & - \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Take the  $k$  element for the probability that  $y$  is in  $c_k$

To find  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$ , given  $\{Y\}, \{X\}$

Parameters of our model

To find  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$ , given  $\{Y\}, \{X\}$

Likelihood

$$\ell(\{\theta\}) = P(\{Y\}|\{X\}; \{\theta\}) = \prod_{n=1}^m \prod_{k=1}^K \left( \frac{e^{(\theta^{(k)T} X^{(n)})}}{\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}} \right)^{y_k^{(n)}} \rightarrow y_k^{(n)} = \begin{cases} 1, & \text{when } y^{(n)} \in c_k \\ 0, & \text{otherwise} \end{cases}$$

To maximise likelihood  $\Leftrightarrow$  to minimise negative log likelihood

**Cost Function**

$$\begin{aligned}
 J(\{\theta\}) &= -\log \ell(\{\theta\}) = -\sum_{n=1}^m \sum_{k=1}^K y_k^{(n)} [\log(e^{(\theta^{(k)T} X^{(n)})}) - \log\left(\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}\right)] \\
 &= -\sum_{n=1}^m \sum_{k=1}^K y_k^{(n)} \theta^{(k)T} X^{(n)} + \sum_{n=1}^m \left[ \sum_{k=1}^K y_k^{(n)} \log\left(\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}\right) \right] \\
 &= -\sum_{n=1}^m \sum_{k=1}^K y_k^{(n)} \theta^{(k)T} X^{(n)} + \sum_{n=1}^m \left[ \log\left(\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}\right) \right]
 \end{aligned}$$

# Cost Function ( negative log likelihood )

$$J(\{\theta\}) = - \sum_{n=1}^m \left[ \sum_{k=1}^K y_k^{(n)} \theta^{(k)T} X^{(n)} \right] + \sum_{n=1}^m \log \left( \sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})} \right)$$

Take  $\nabla_{\theta^{(j)}} J(\{\theta\})$  w.r.t. a particular vector  $\theta^{(j)}$  & perform Gradient Decent

$$\frac{\partial}{\partial \theta^{(j)}} J(\{\theta\}) = - \sum_{n=1}^m y_j^{(n)} X^{(n)} + \sum_{n=1}^m \left( \frac{1}{\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}} \right) e^{(\theta^{(j)T} X^{(n)})} X^{(n)}$$

$$= \sum_{n=1}^m \left[ \left( \frac{e^{(\theta^{(j)T} X^{(n)})}}{\sum_{l=1}^K e^{(\theta^{(l)T} X^{(n)})}} \right) - y_j^{(n)} \right] X^{(n)}$$

$$= \sum_{n=1}^m \left[ P(y^{(n)} \in c_j) - y_j^{(n)} \right] X^{(n)}$$

Difference between predicted value & true value

# Gradient

$$\frac{\partial}{\partial \theta^{(j)}} J(\{\theta\}) = \boxed{\sum_{n=1}^m \left[ \left( \frac{e^{(\theta^{(j)})^T X^{(n)}}}{\sum_{l=1}^K e^{(\theta^{(l)})^T X^{(n)}}} \right) - y_j^{(n)} \right] X^{(n)}}$$

# Gradient Descent

Repeat until convergence {

For every class  $c_j$  { //every  $\theta$  seperately

$$\theta^{(j)} := \theta^{(j)} - \alpha \cdot \boxed{\frac{\partial}{\partial \theta^{(j)}} J(\{\theta\})}$$

Simultaneously update all  $\theta_i^{(j)}, i = 1, \dots, n$

}

}

# To make prediction

After training

$$\text{class}(X^{(new)}) = \text{argmax} \left( \sigma \left( \{\theta\}^{(trained)}^T X^{(new)} \right) \right)$$

$$\begin{aligned}\{\theta\}^{(trained)} &= \begin{bmatrix} -\theta^{(1,trained)} & - \\ -\theta^{(2,trained)} & - \\ \vdots & \\ -\theta^{(K,trained)} & - \end{bmatrix} \\ &= \begin{bmatrix} \theta_1^{(1,trained)} & \dots & \theta_n^{(1,trained)} \\ \vdots & \ddots & \vdots \\ \theta_1^{(K,trained)} & \dots & \theta_n^{(K,trained)} \end{bmatrix}\end{aligned}$$

$$X^{(new)} = \begin{bmatrix} x_1^{(new)} \\ x_2^{(new)} \\ \vdots \\ x_n^{(new)} \end{bmatrix}$$

# Coding with sci-kit learn

```
# Import lib
from sklearn.linear_model import LogisticRegression

# Train the model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Evaluate on training set
train_acc = model.score(X_train_scaled, y_train)
print("The Accuracy for Training Set is {}".format(train_acc*100))

# Evaluate on test set
test_acc = accuracy_score(y_test, y_pred)
print("The Accuracy for Test Set is {}".format(test_acc*100))

# Generate classification report
print(classification_report(y_test, y_pred))
```

# Summary

# Summary

1. Intuition
2. Odds, Log Odds, Odds Ratio
3. Hypothesis Function
4. Decision Boundary
5. Cost Function
6. Gradient Descent
7. Regularisation
8. Multinomial Logistic Regression (Softmax Regression)

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Tree-based Learning

Dr Yang Long

# Previously

## Machine Learning Algorithms



Regression



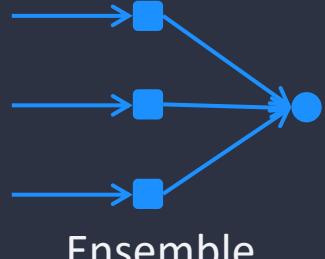
Regularisation



Clustering



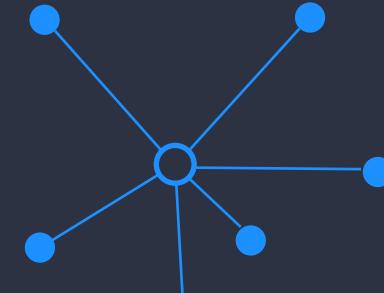
Bayesian



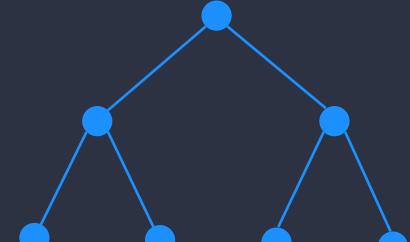
Ensemble



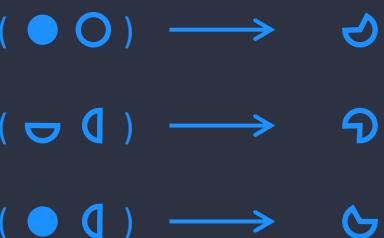
Neural Network



Instance-based



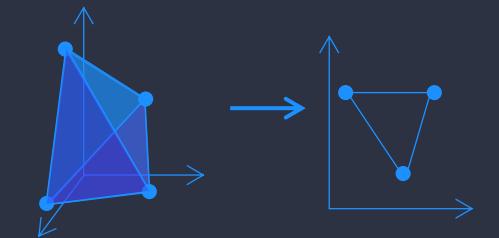
Tree-based



Associated Rule Learning



Deep Learning



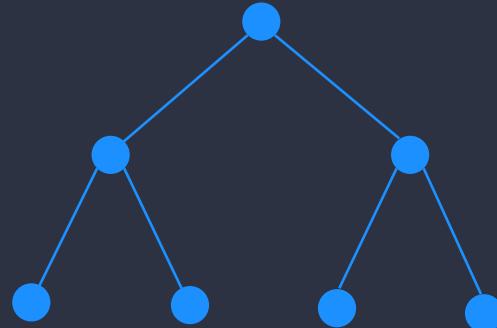
Dimensionality Reduction



Reinforcement Learning

# Lecture Overview

1. Decision Trees
2. Learning a Decision Tree

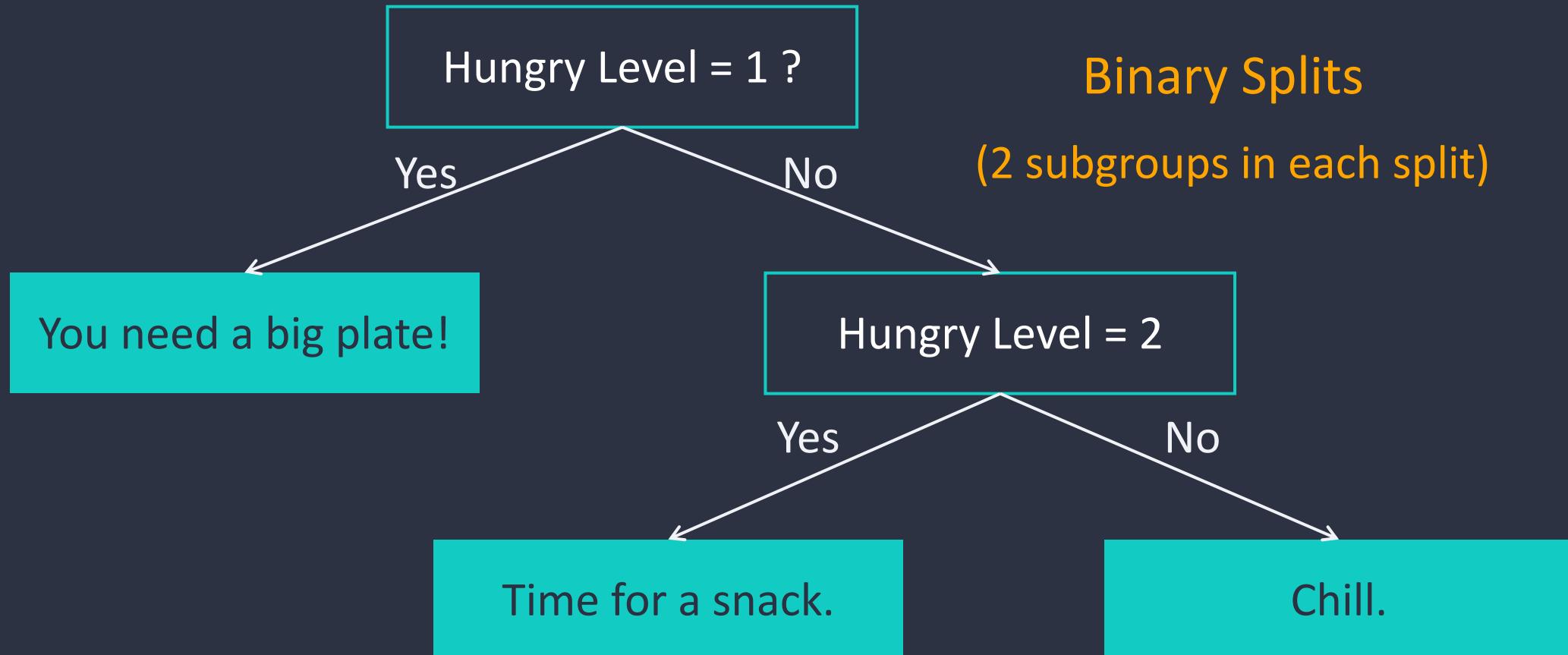


Tree-based

# 1. Decision Trees

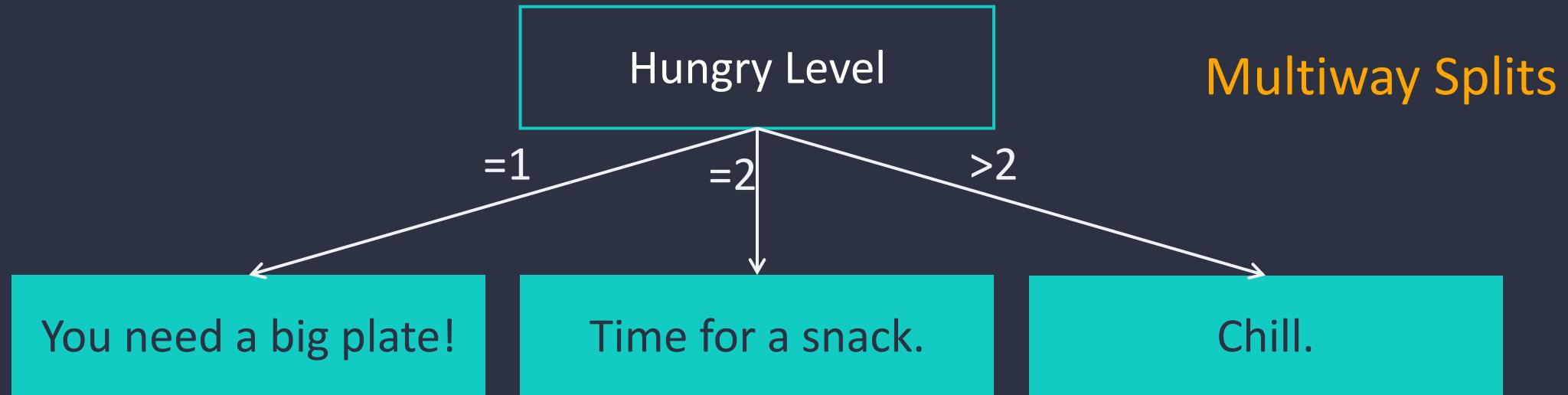
# Decision Trees

EXAMPLE. learn from ranked data



# Decision Trees

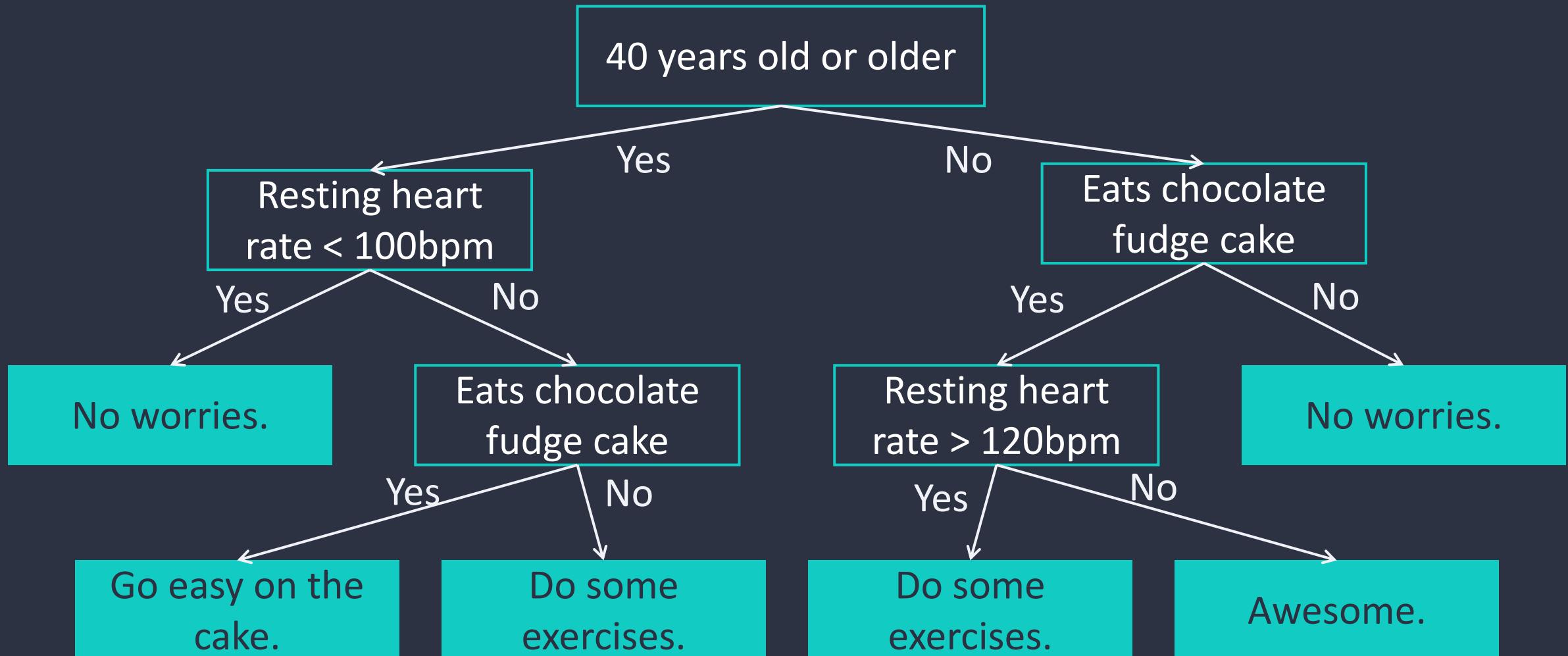
EXAMPLE. learn from ranked data



More decision trees with binary splits, in practice.  
Advantages of multiway splitting: exhaust all info in a nominal feature, i.e., a feature rarely appears more than once in any path from the root note to a leaf note.

# Decision Trees

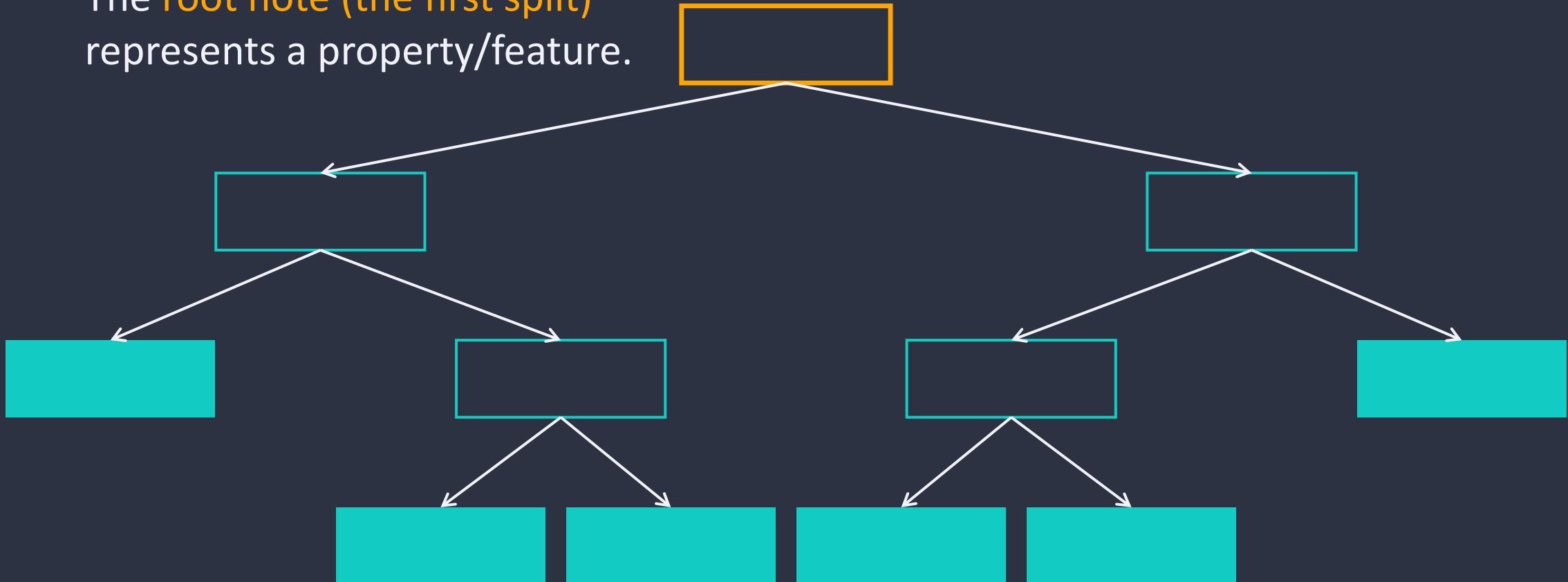
EXAMPLE. learn for a more complicated DT



# Decision Trees

## Some Terminologies

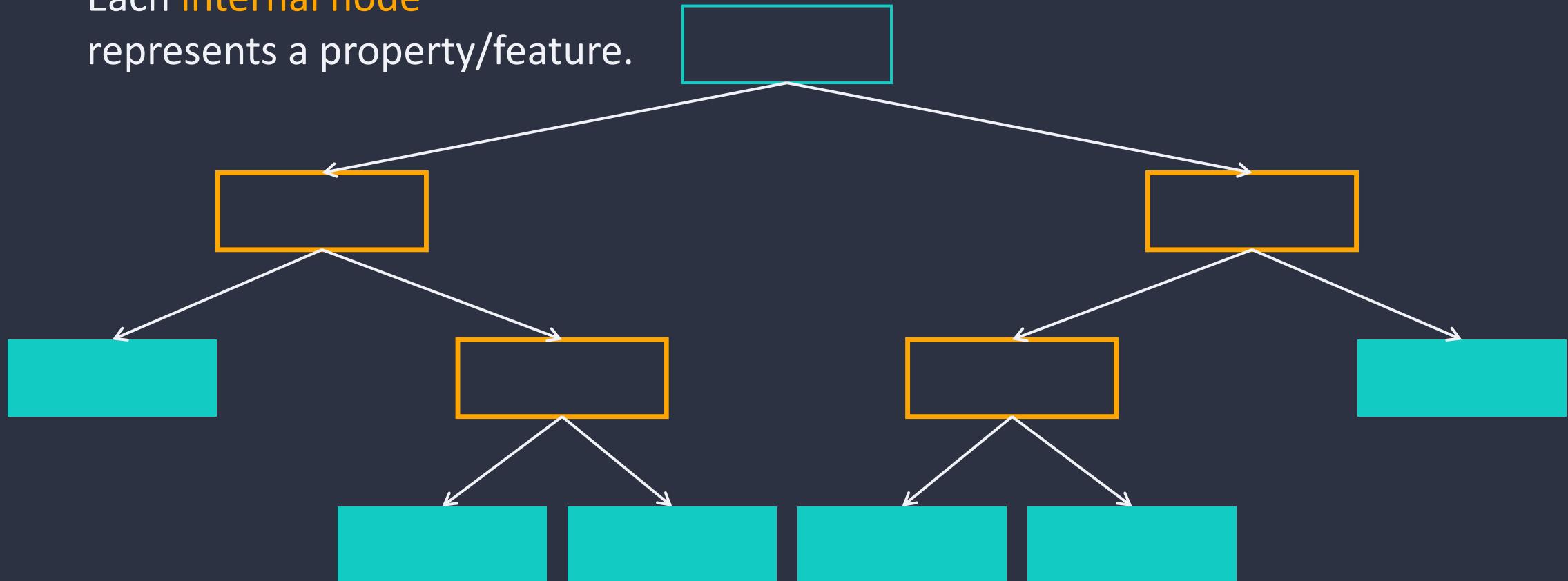
The **root note (the first split)** represents a property/feature.



# Decision Trees

## Some Terminologies

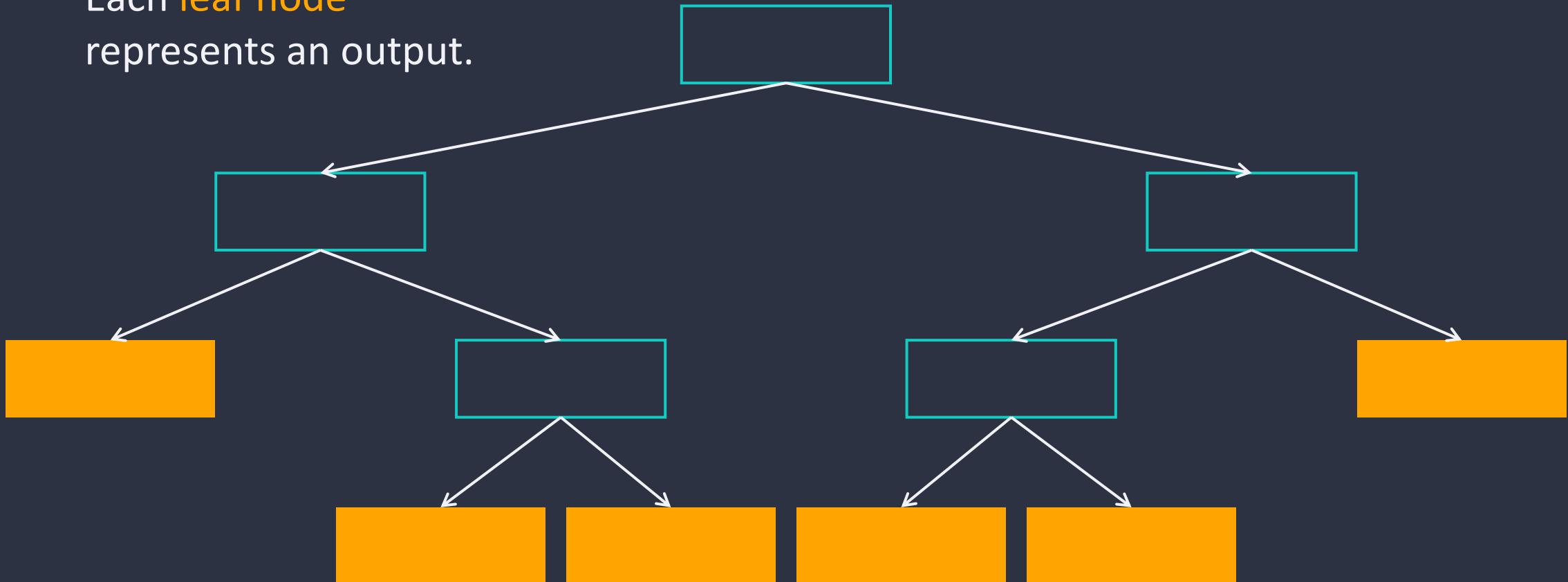
Each **internal node**  
represents a property/feature.



# Decision Trees

## Some Terminologies

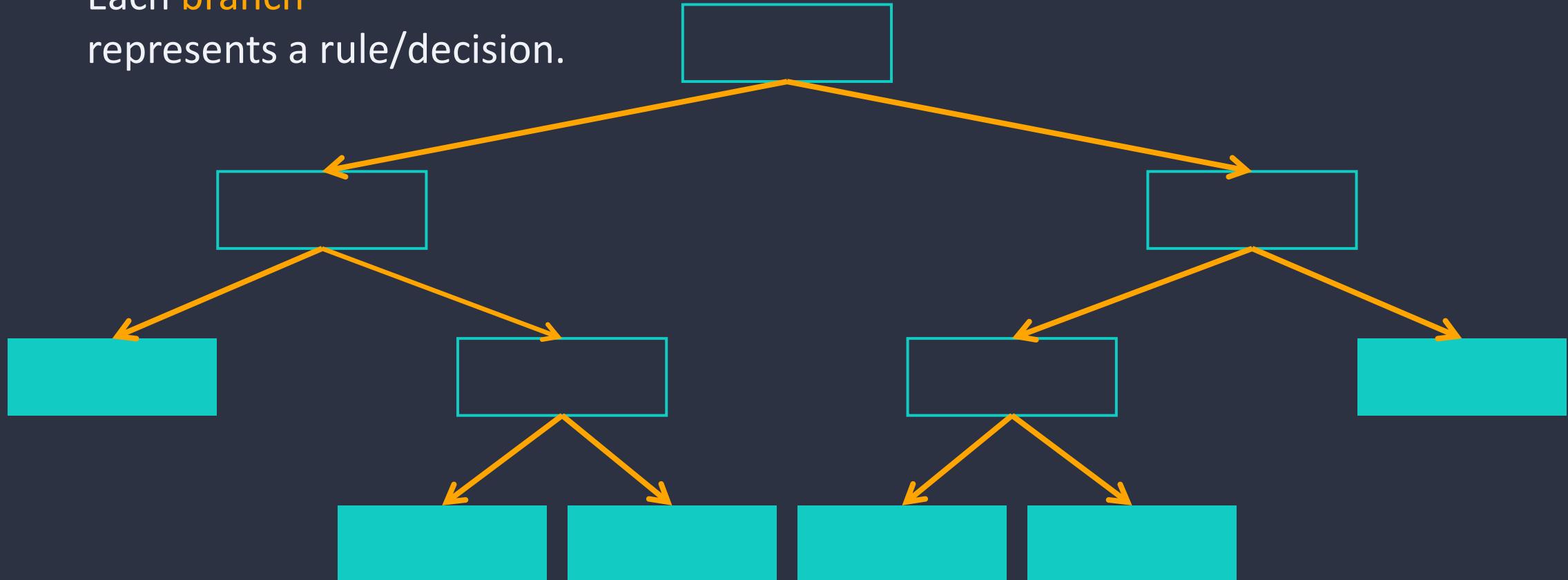
Each **leaf node**  
represents an output.



# Decision Trees

## Some Terminologies

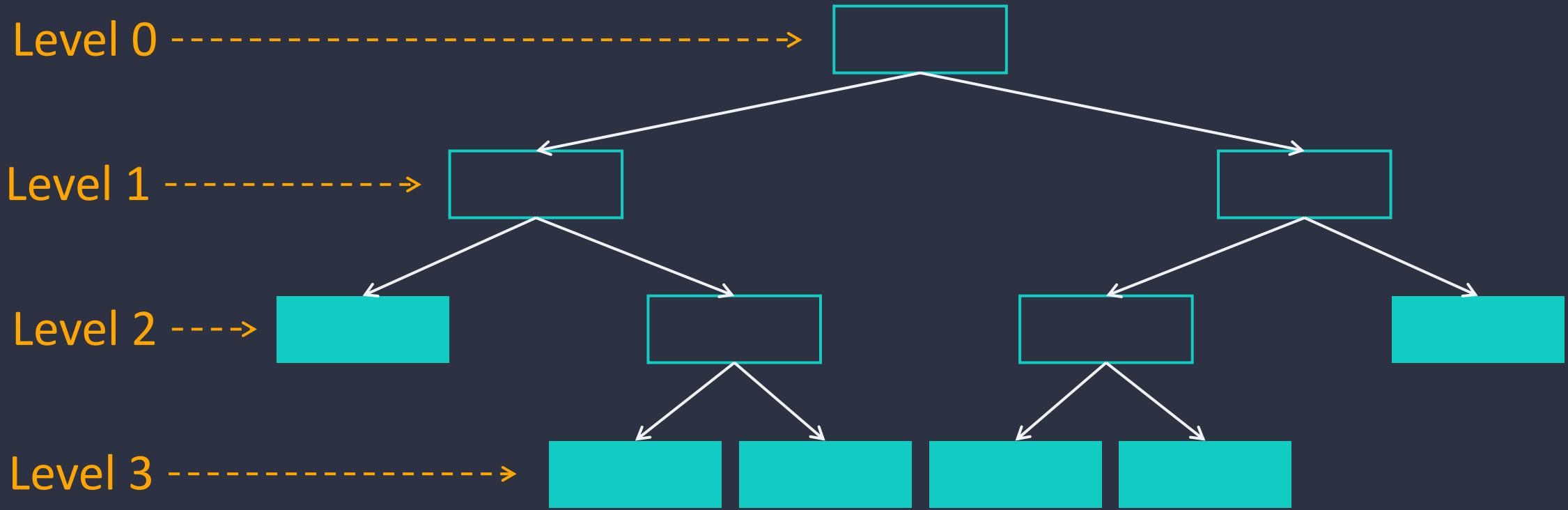
Each **branch**  
represents a rule/decision.



# Decision Trees

## Some Terminologies

Depth is defined by number of levels, excluding the *root node*.



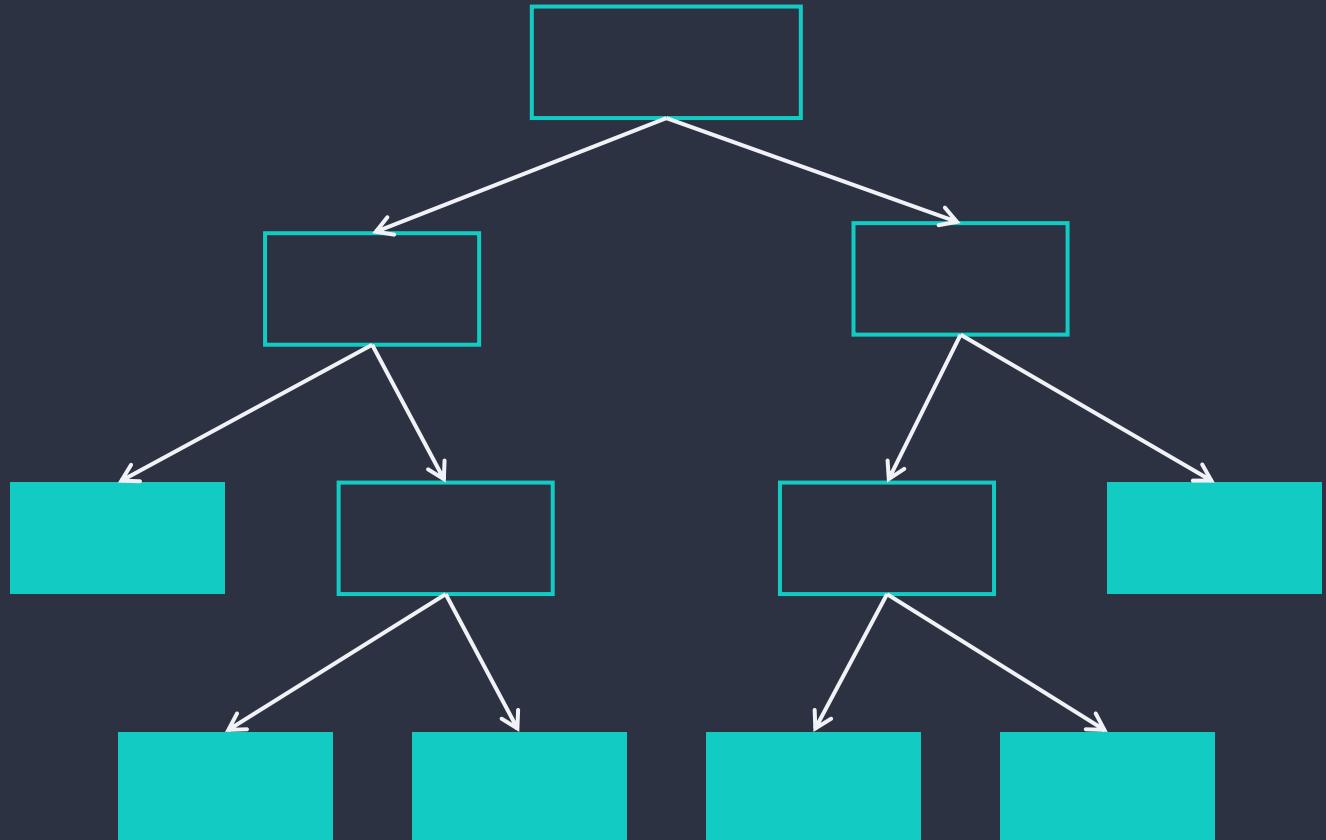
(the depth of this decision tree is 3)

# Decision Trees

To learn for a DT

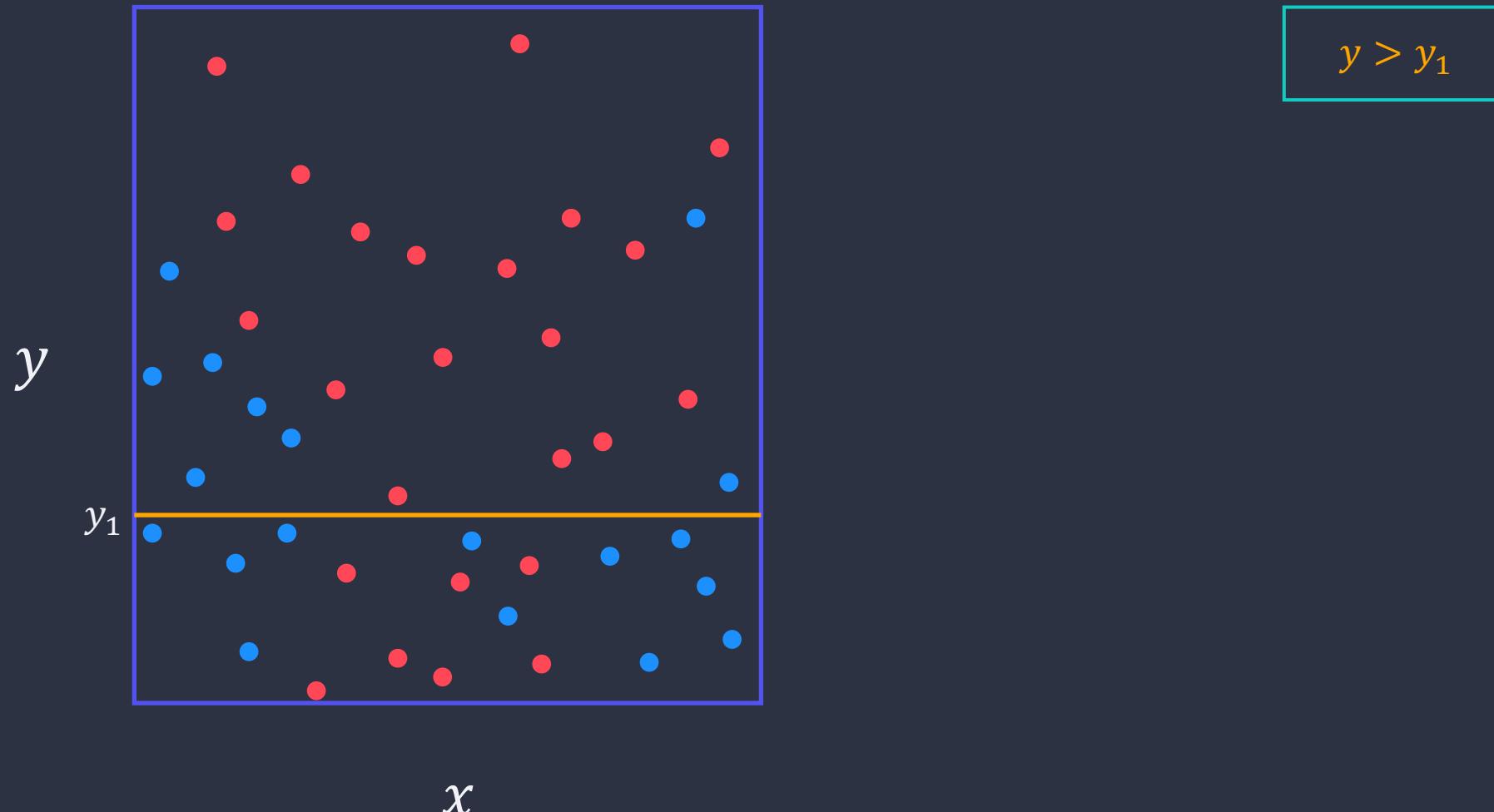
Progressively splits datasets into sub-sets based one descriptive feature, until all sub-sets are small enough to be represented by some label.

Data that fall into the same sub-sets are most similar (homogeneity). Data that fall into different sub-sets are as different as possible (heterogeneity).



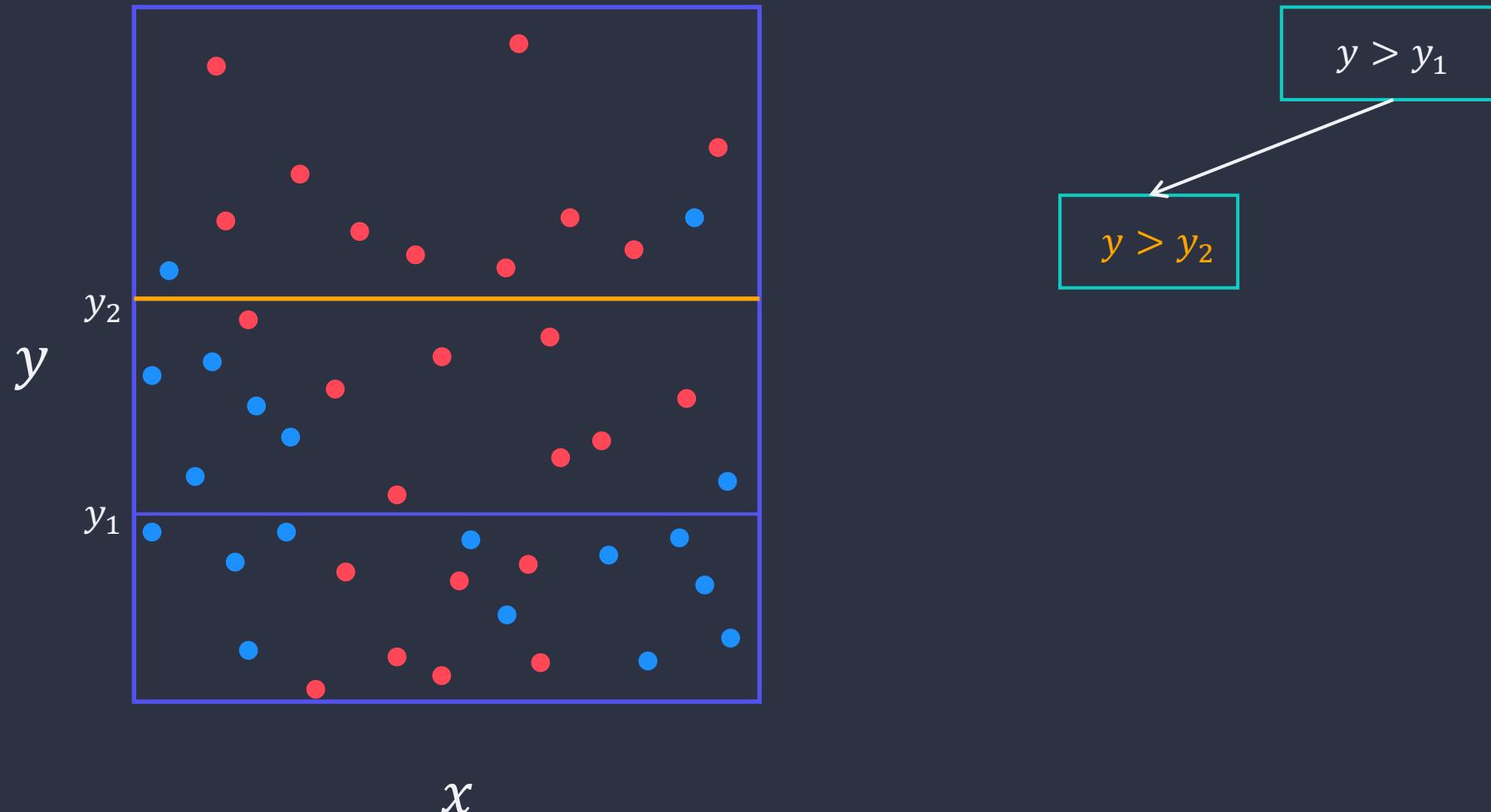
# Decision Trees

To learn for a DT



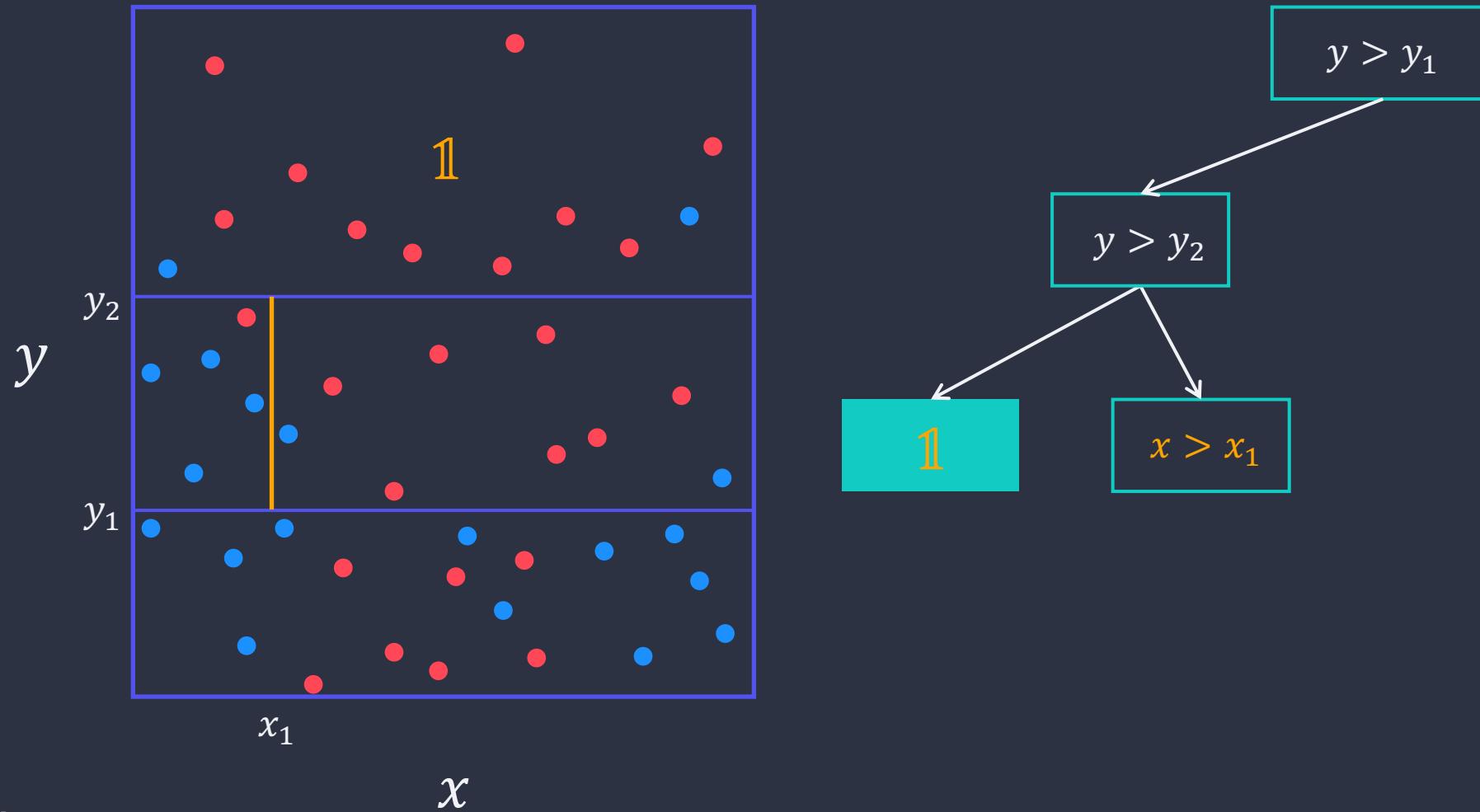
# Decision Trees

To learn for a DT



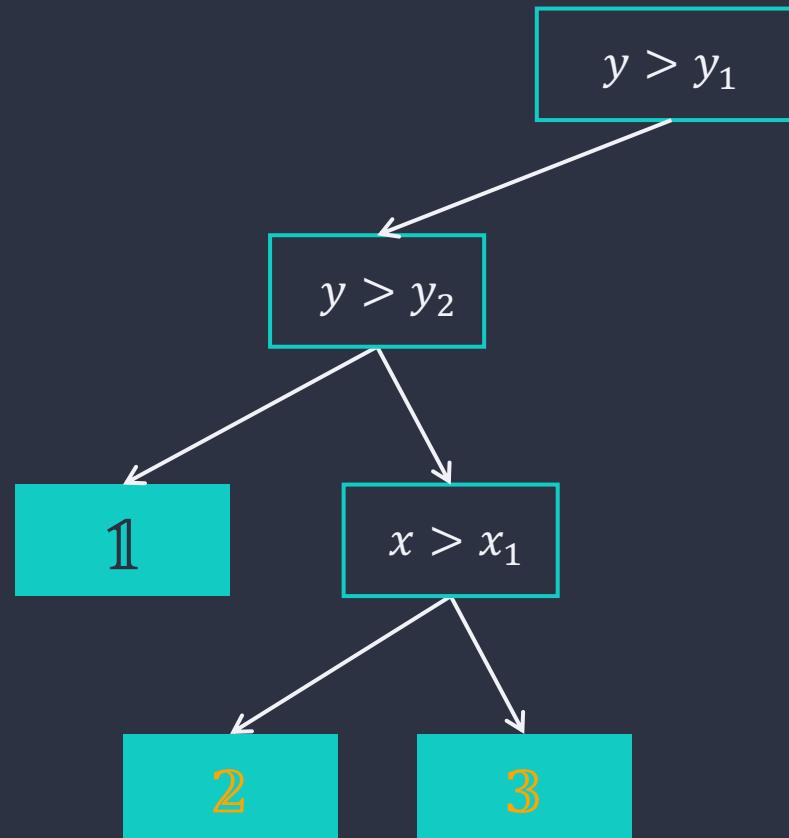
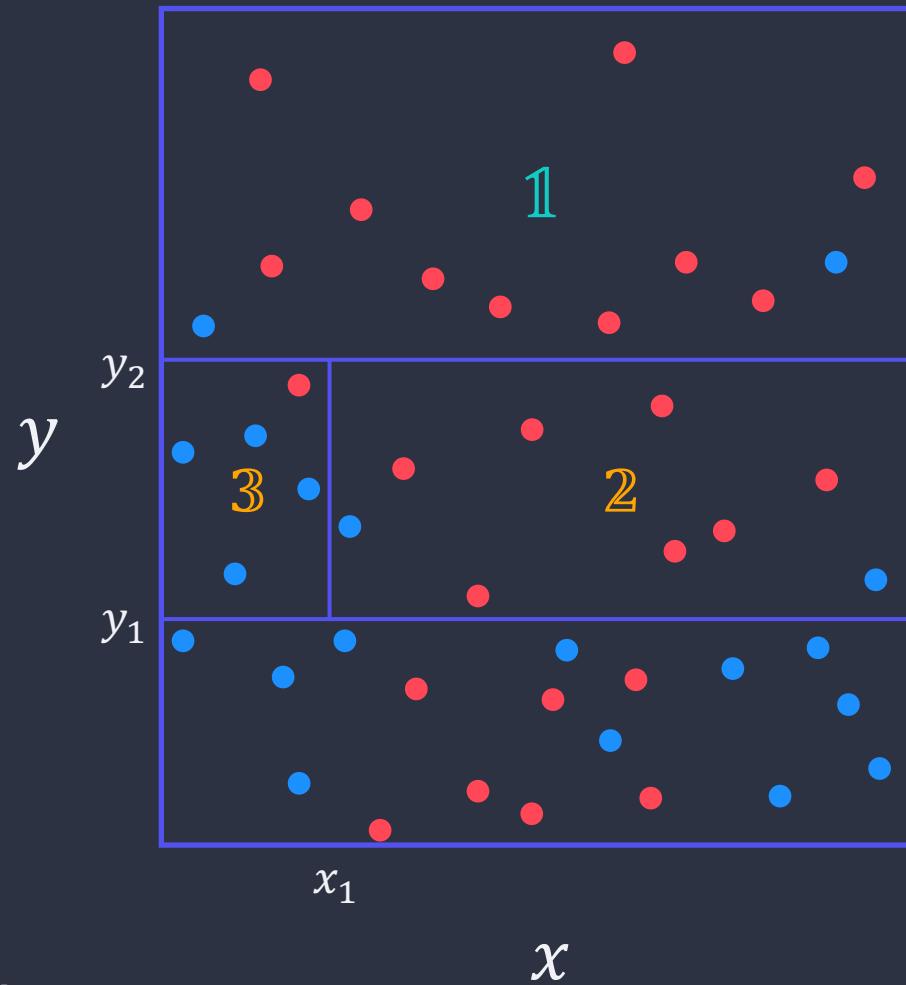
# Decision Trees

To learn for a DT



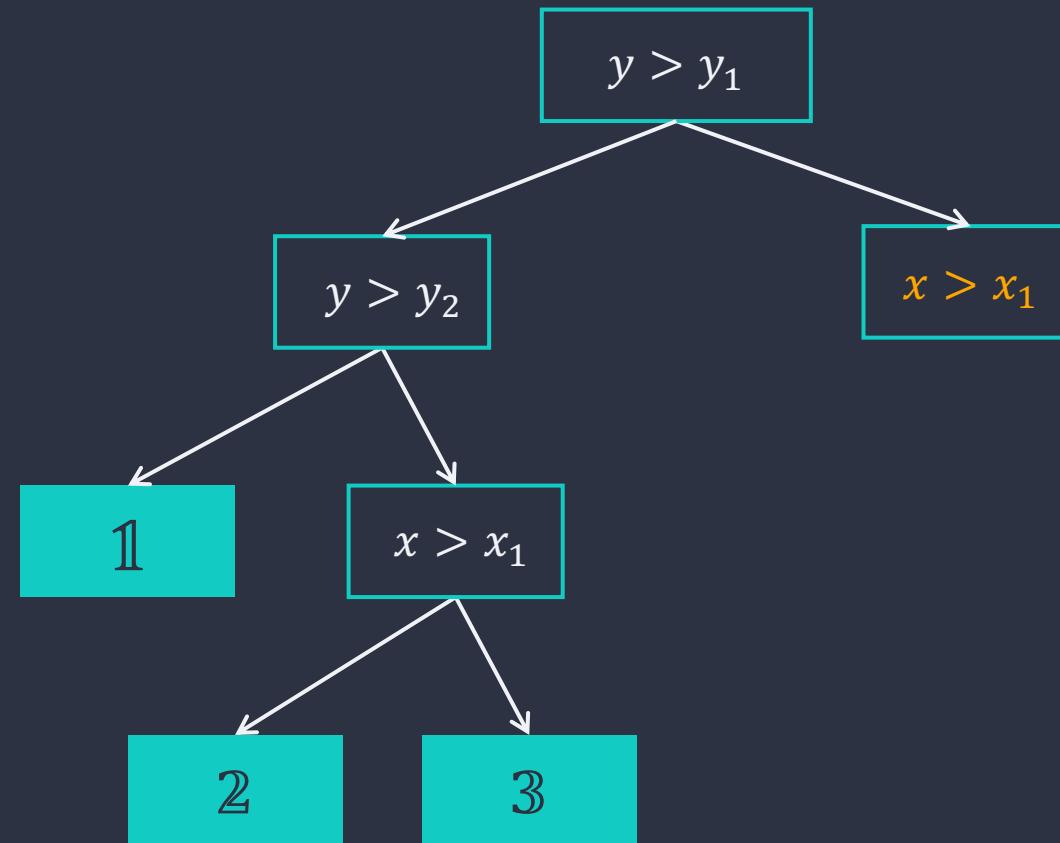
# Decision Trees

To learn for a DT



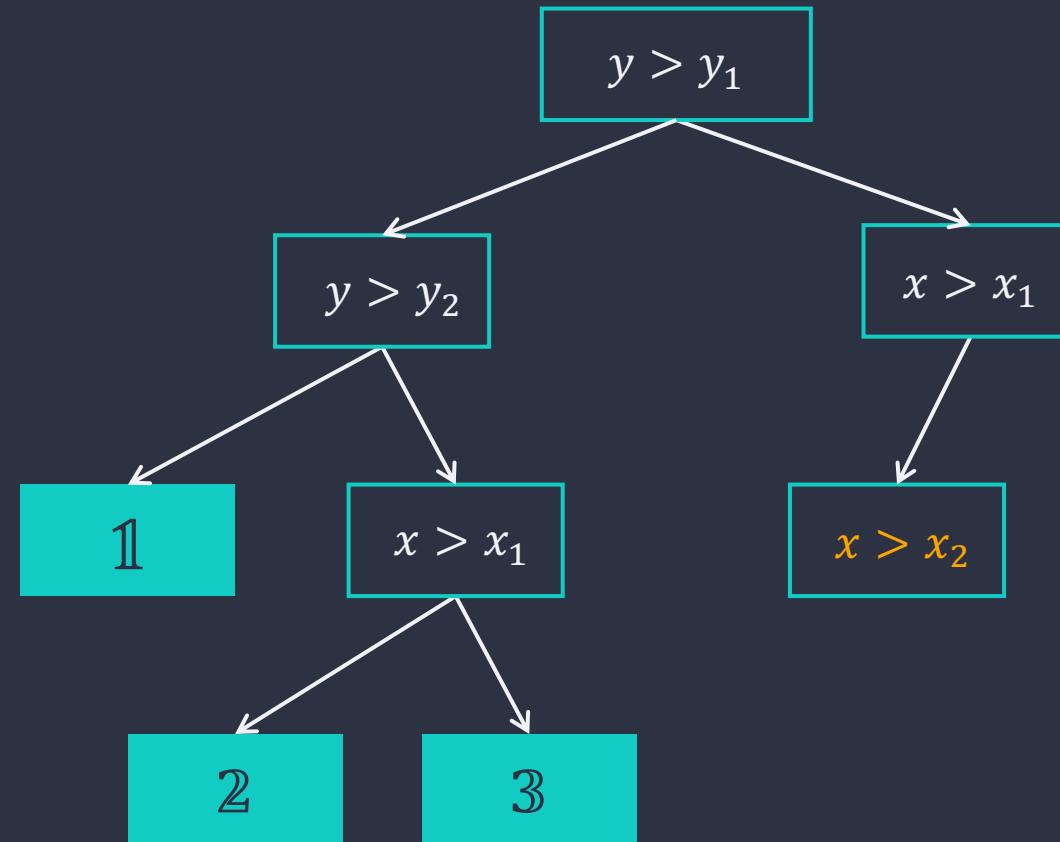
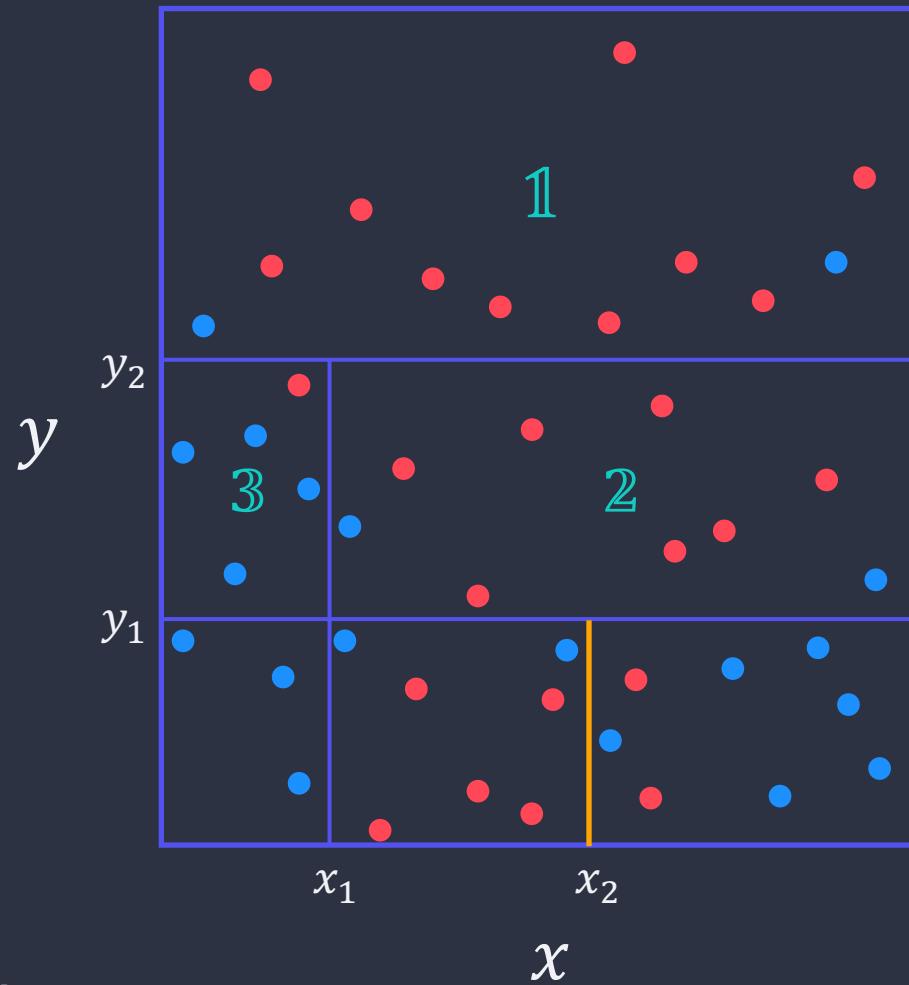
# Decision Trees

# To learn for a DT



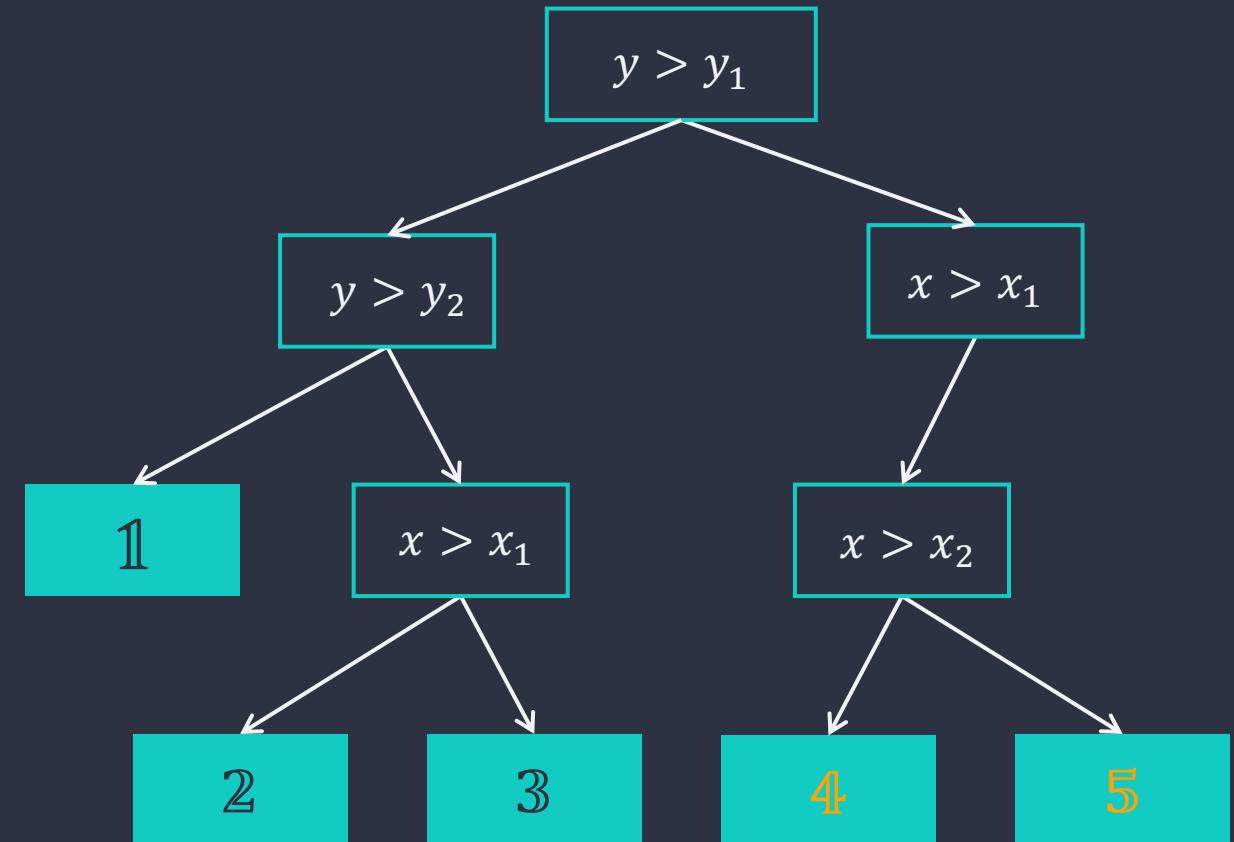
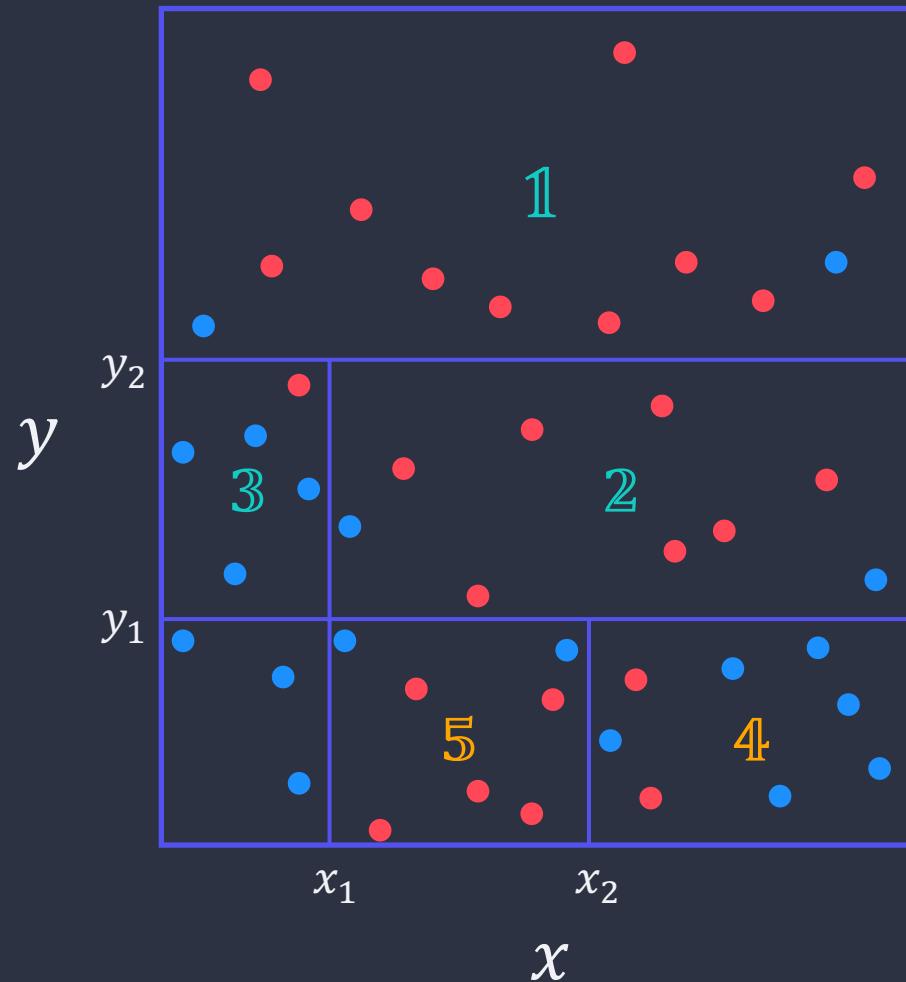
# Decision Trees

To learn for a DT



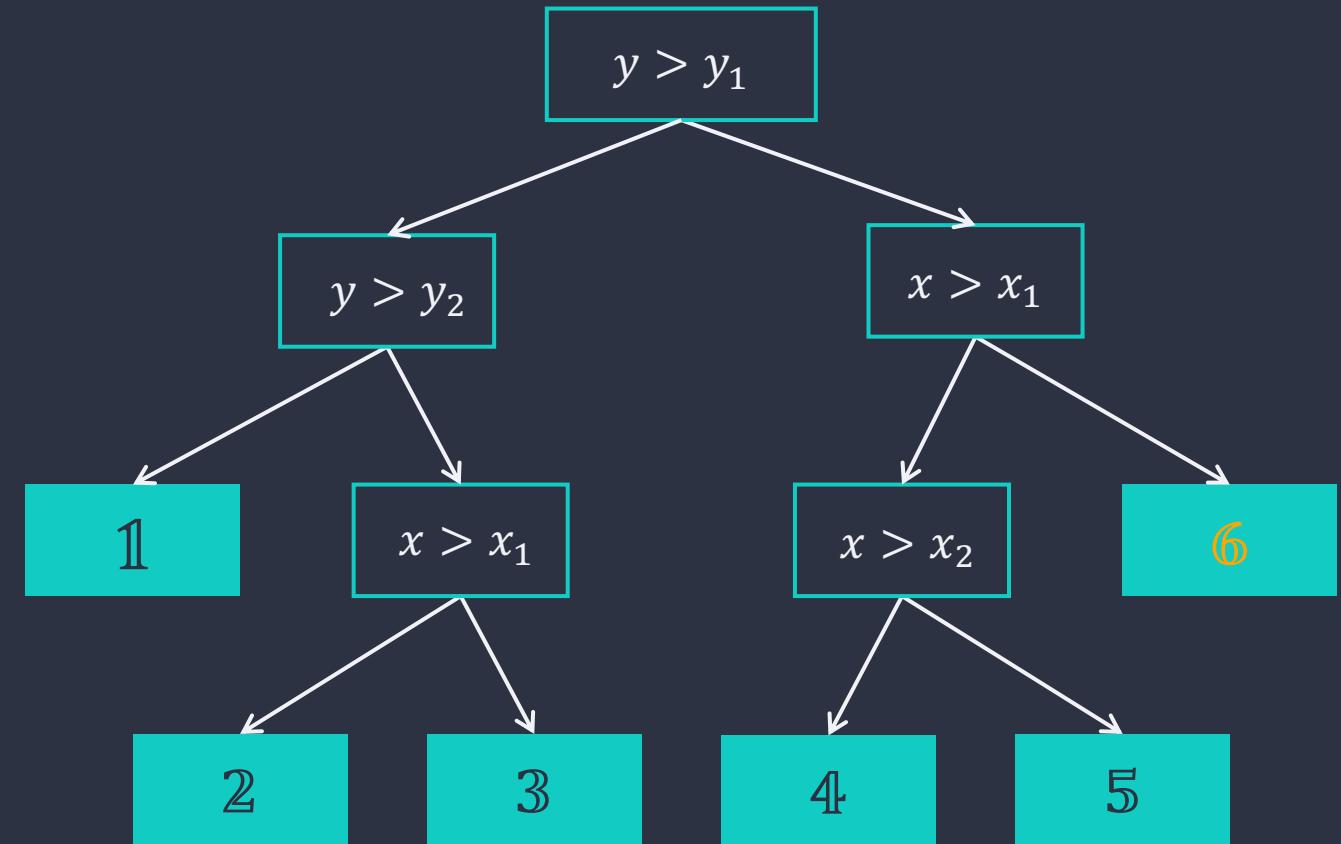
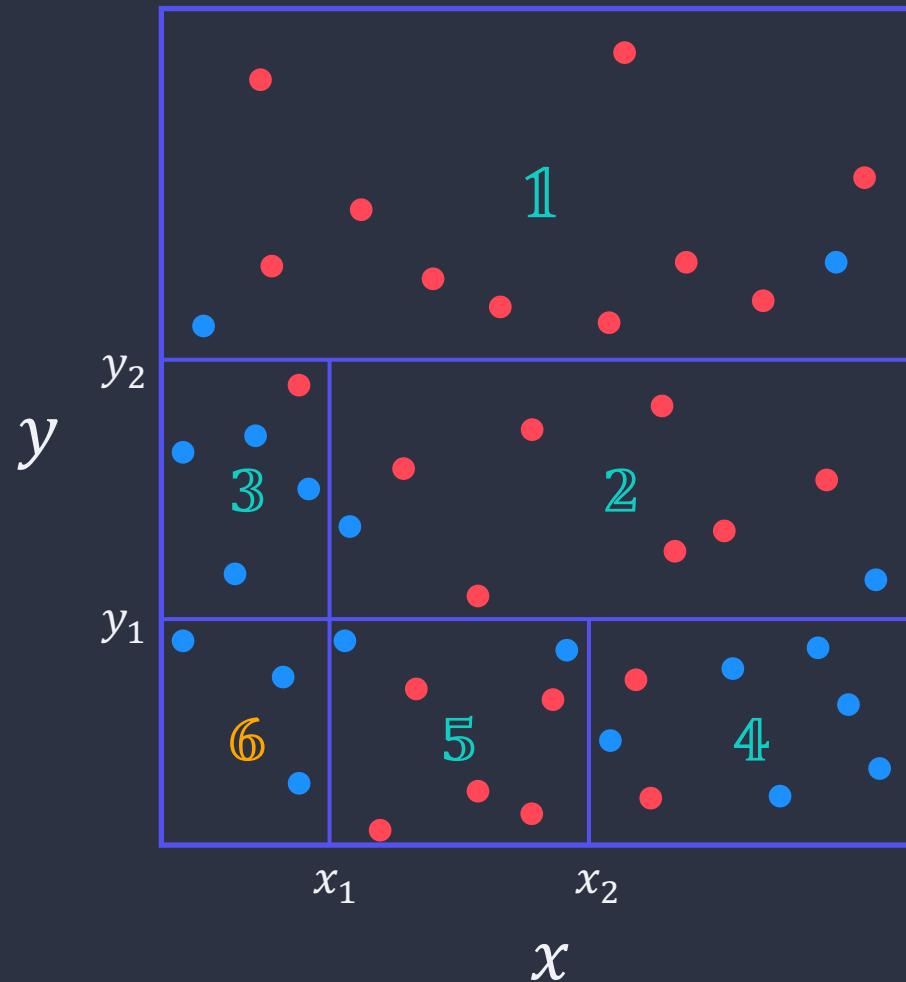
# Decision Trees

To learn for a DT



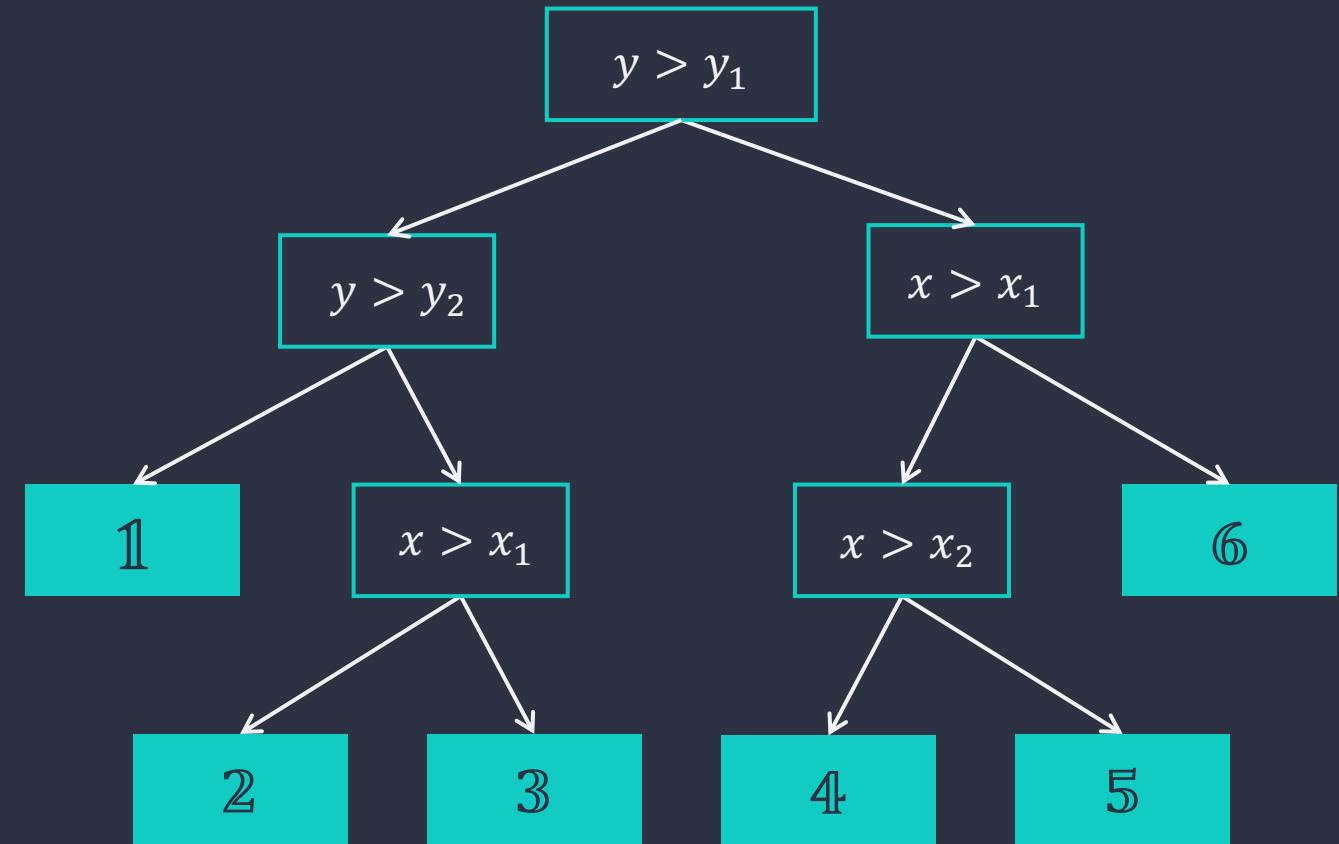
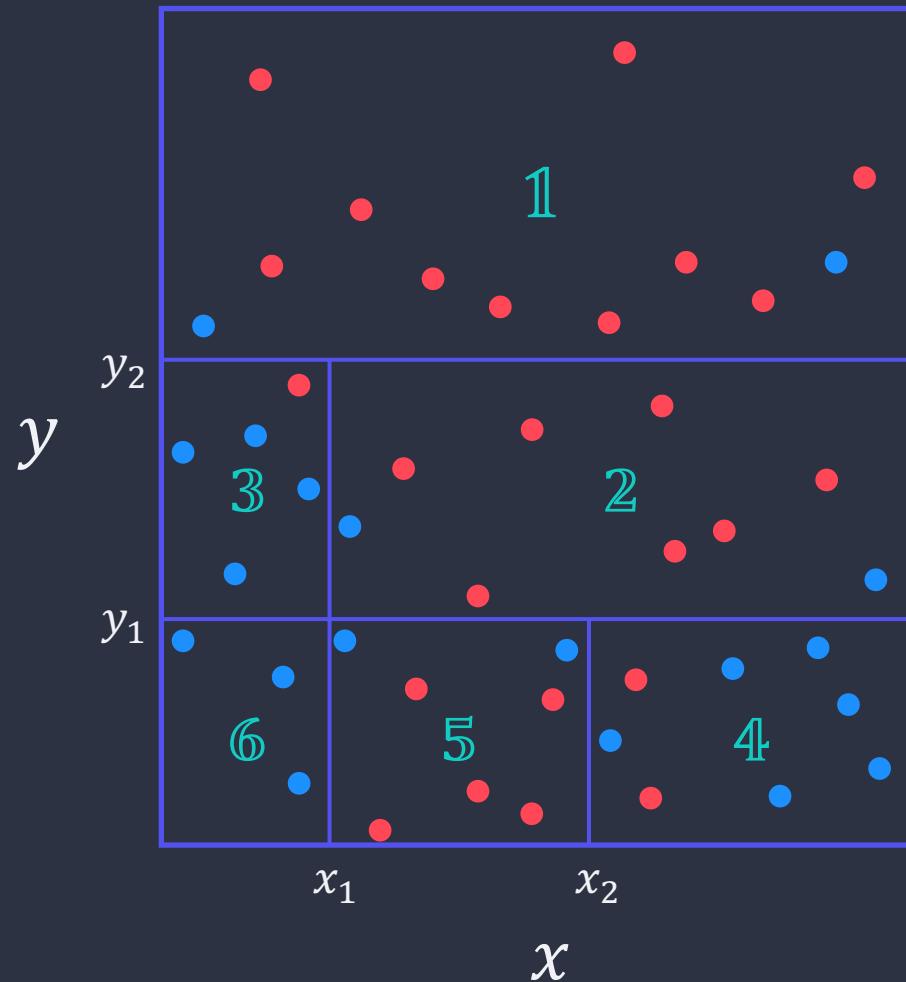
# Decision Trees

To learn for a DT



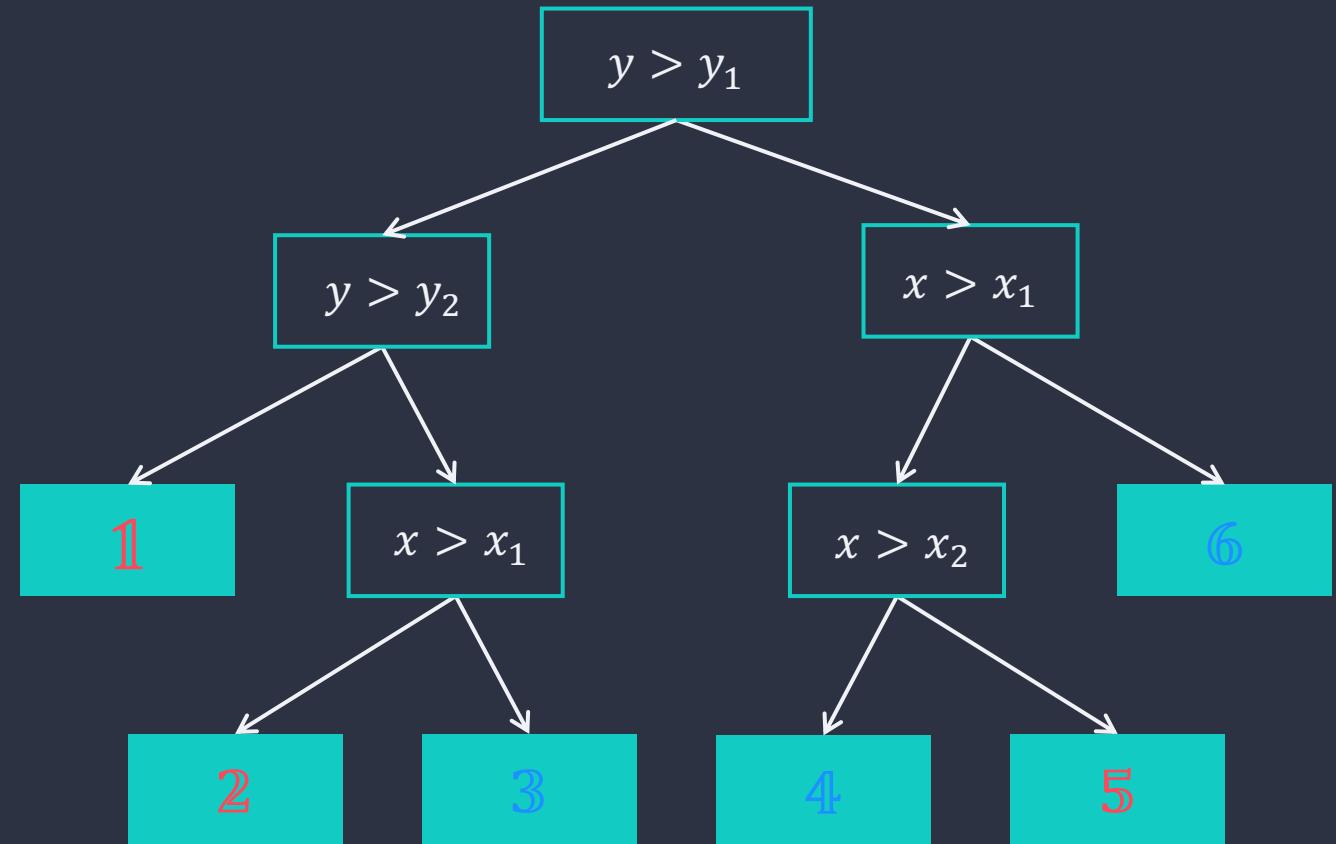
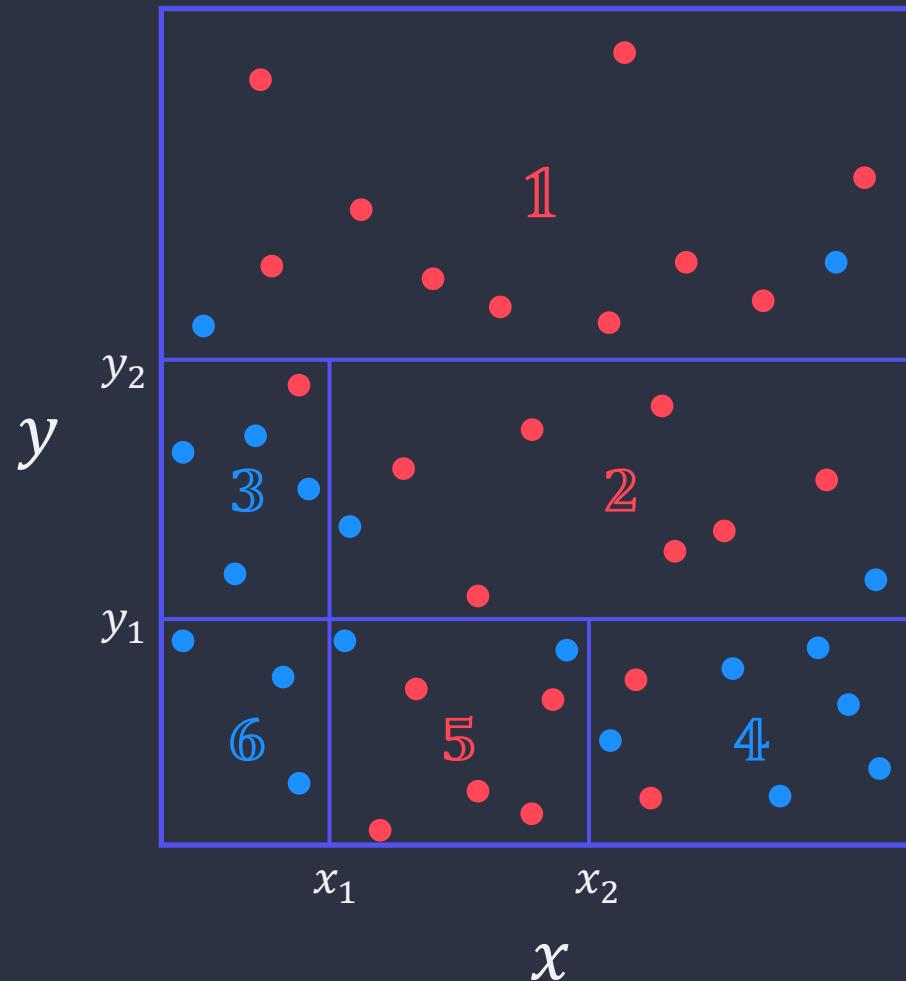
# Decision Trees

To learn for a DT



# Decision Trees

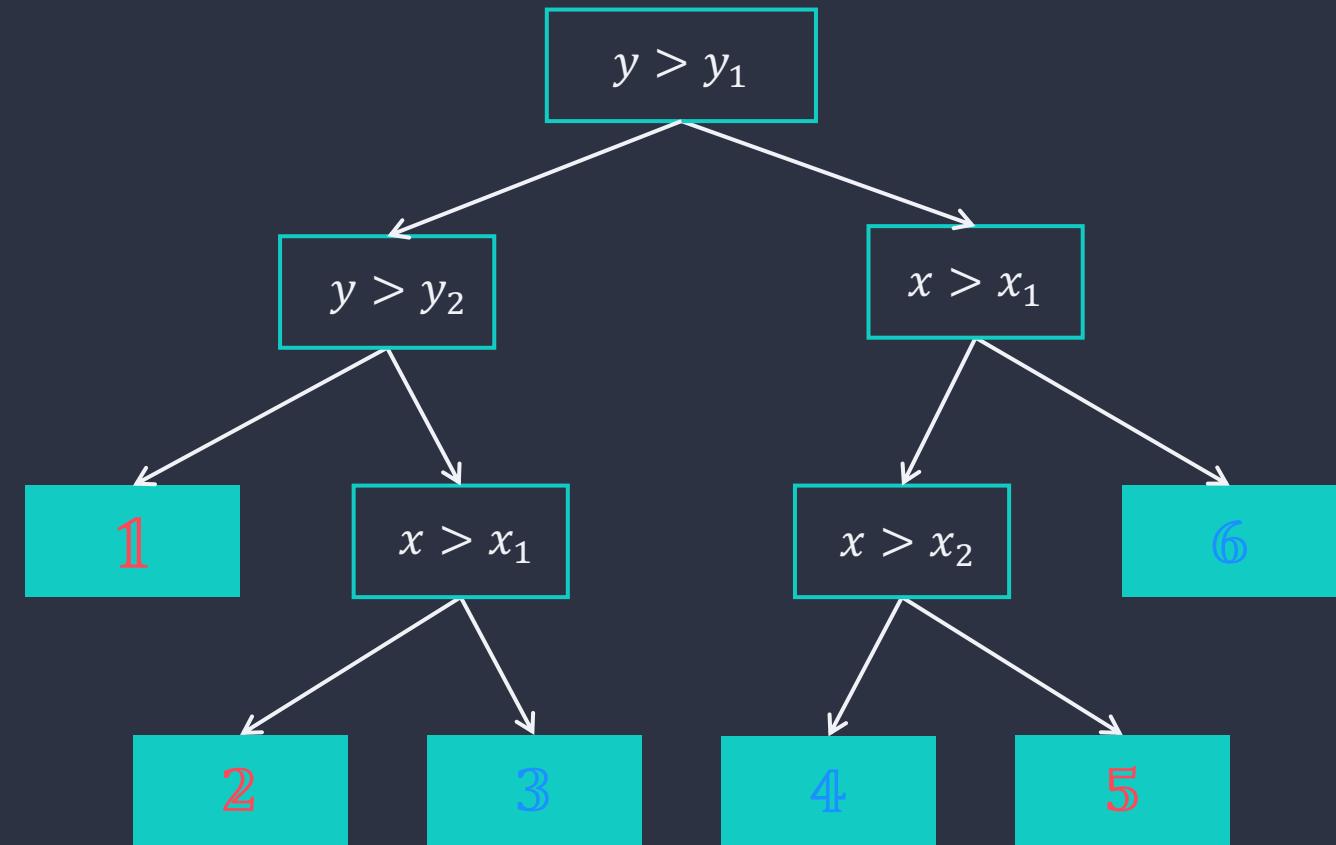
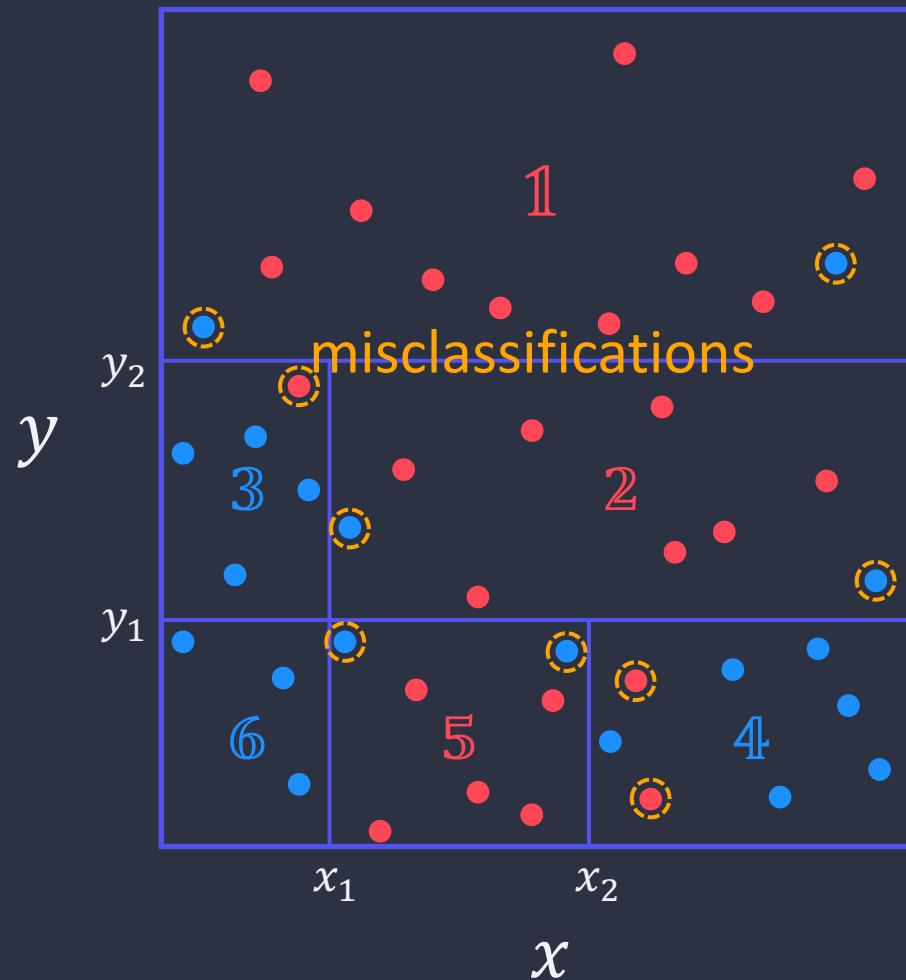
To learn for a DT



Positive class: 1, 2, 5    Negative class: 3, 4 , 6  
A binary classifier

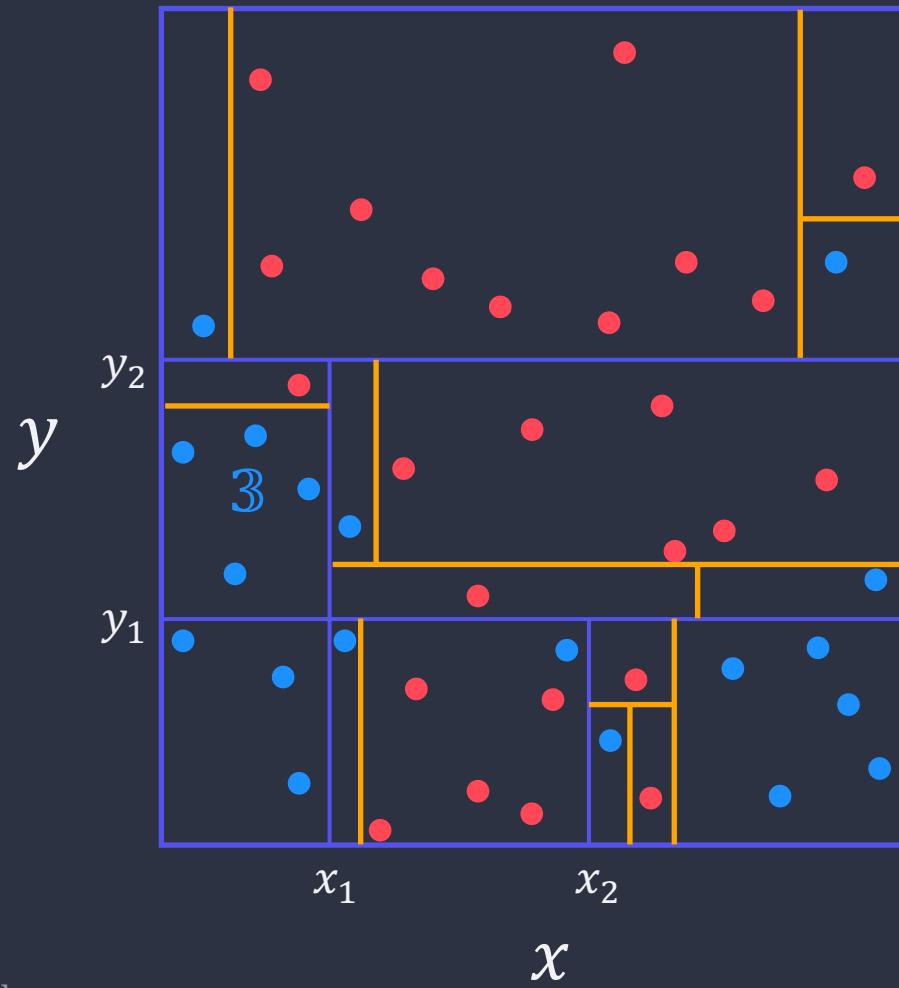
# Decision Trees

To learn for a DT

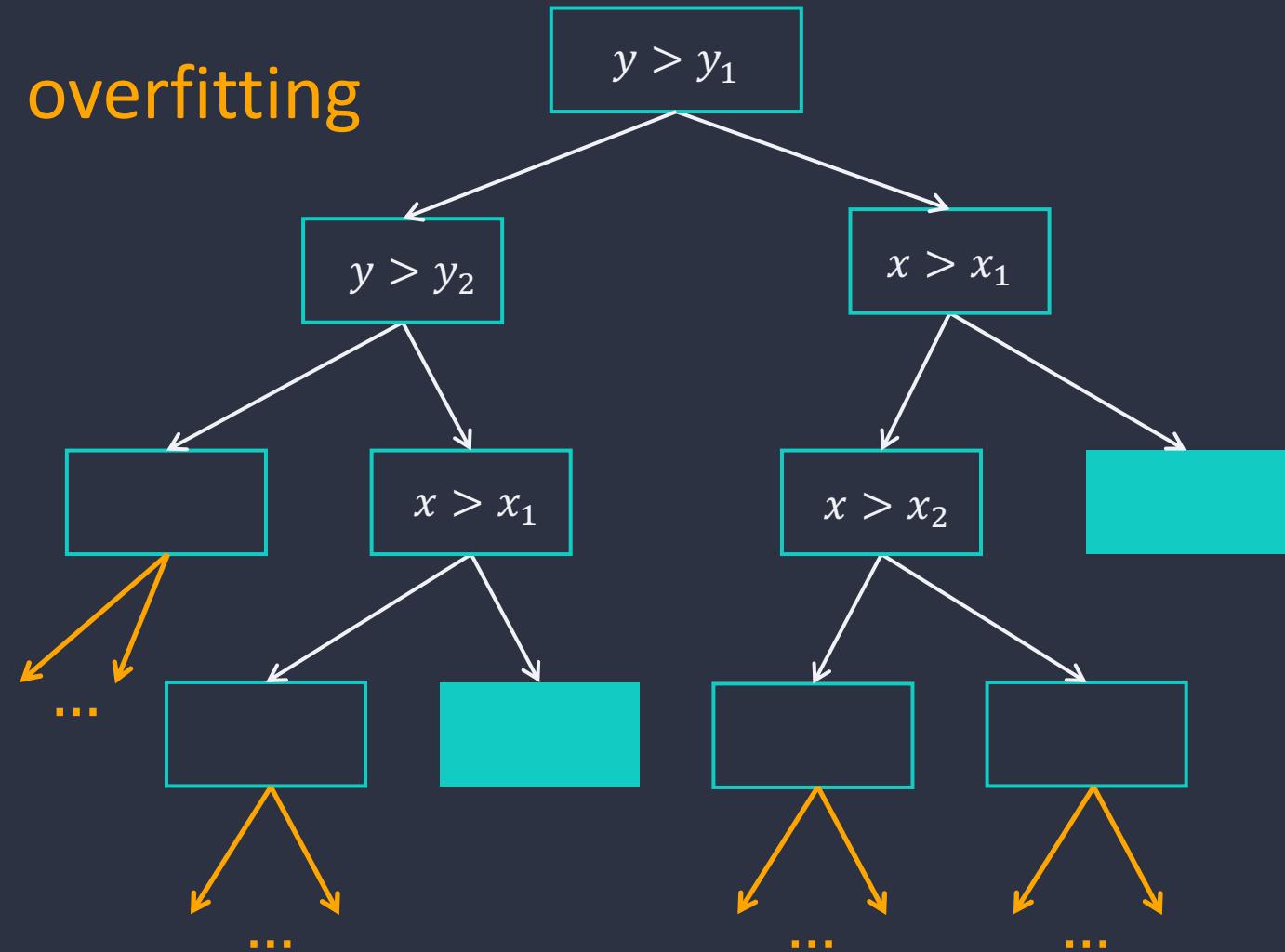


# Decision Trees

To learn for a DT



overfitting



# Decision Trees

Prune a DT to prevent overfitting



Pruning a data compression technique that reduces DT's size by removing its branches, which provide little predictive power.

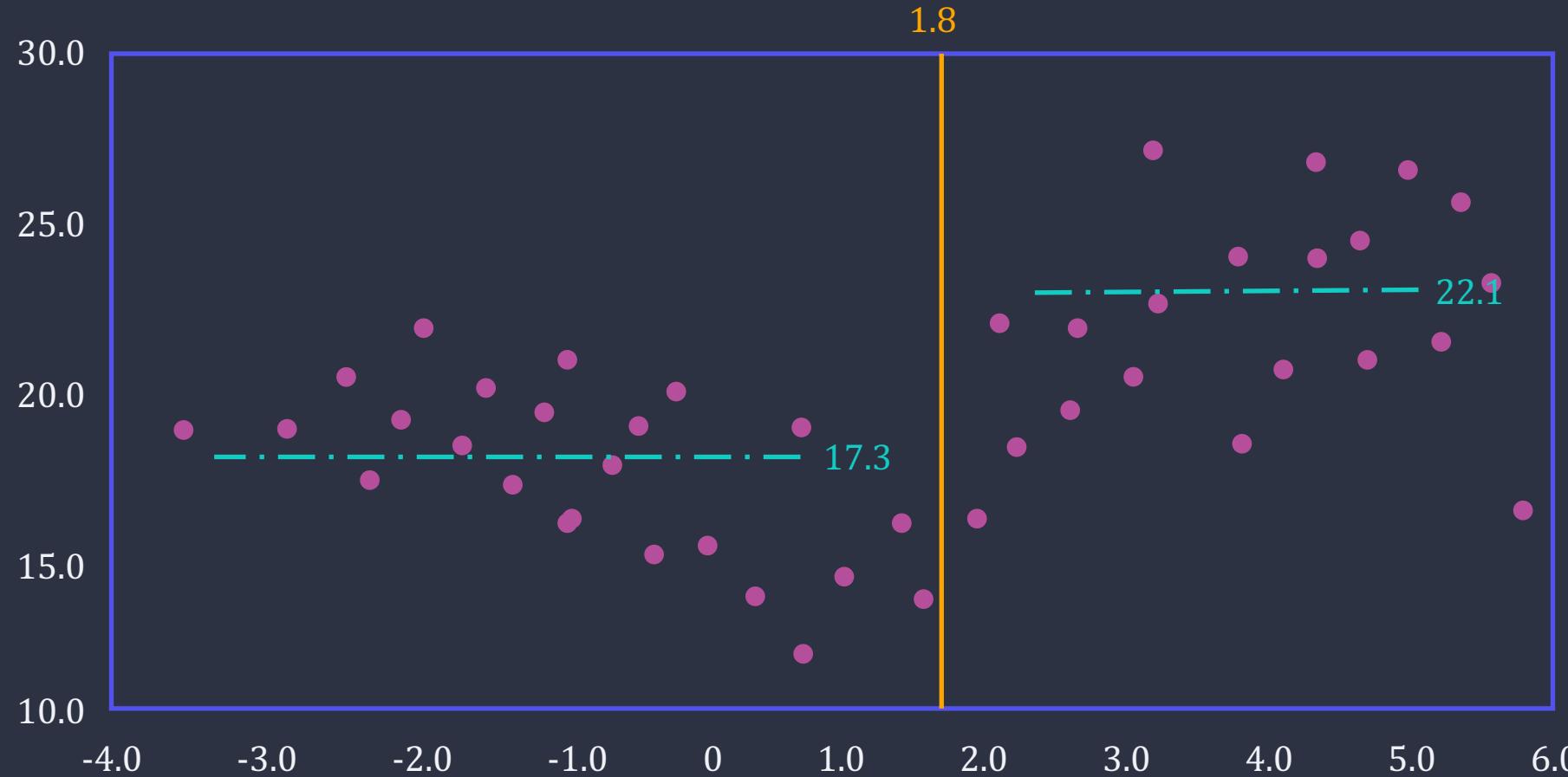
Pre-pruning stop growing a DT before it becomes too complex yet not providing additional information.

Post-pruning stop removing DT branches when no further improvement can be made.

# Decision Trees for Regression

To learn for a Regression Tree

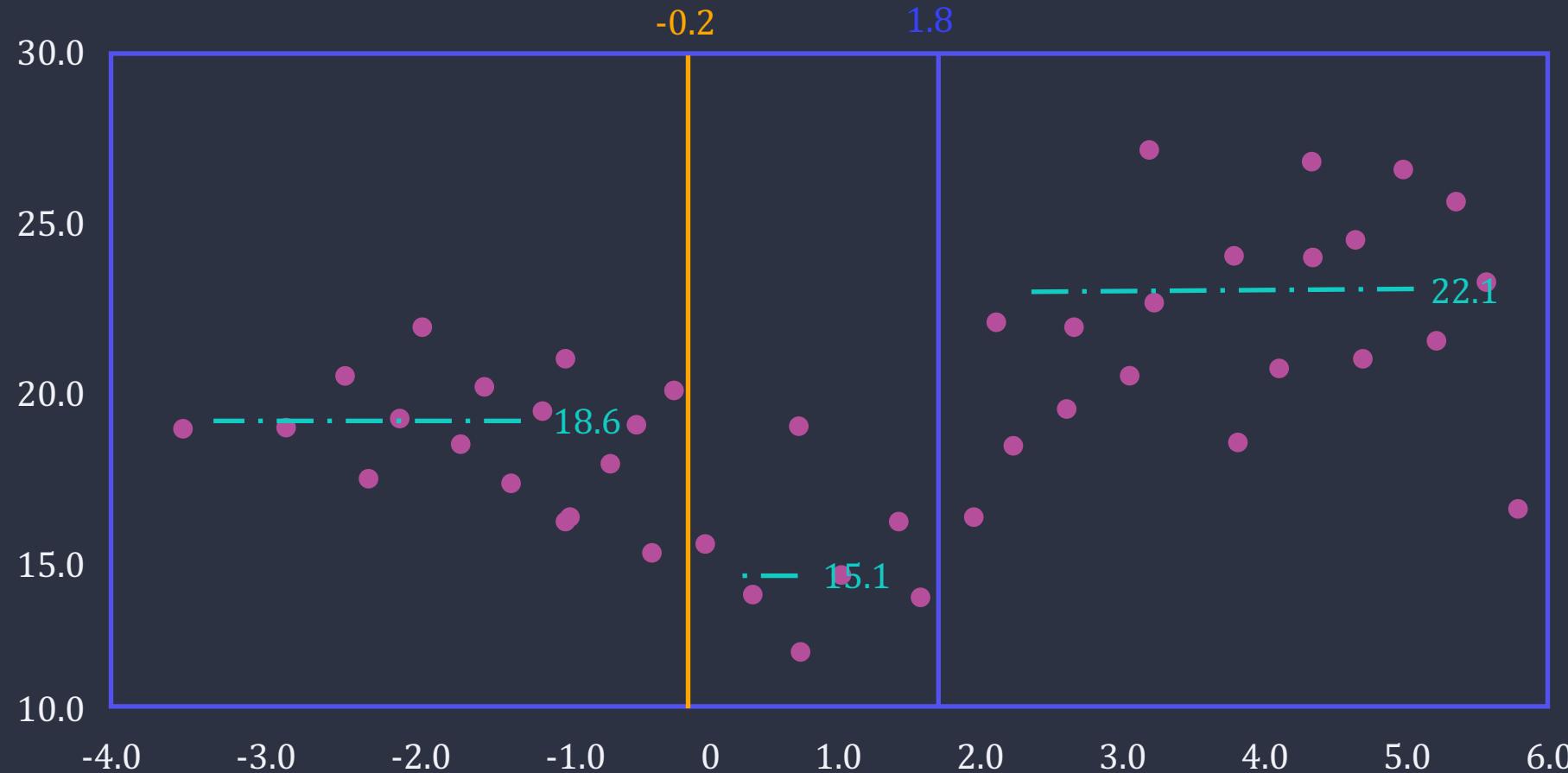
Predict real valued numbers at leaf nodes



# Decision Trees for Regression

To learn for a Regression Tree

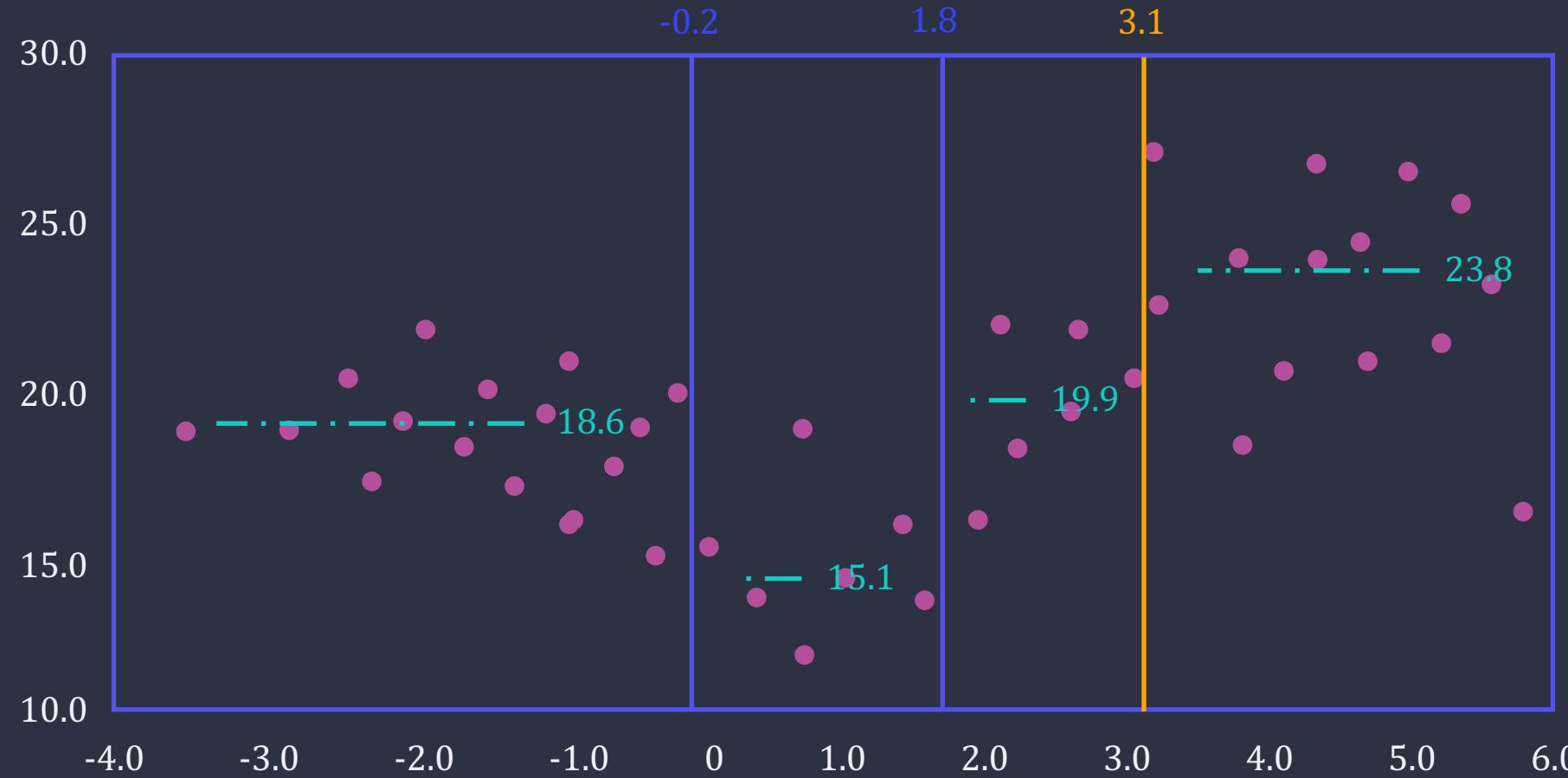
Predict real valued numbers at leaf nodes



# Decision Trees for Regression

To learn for a Regression Tree

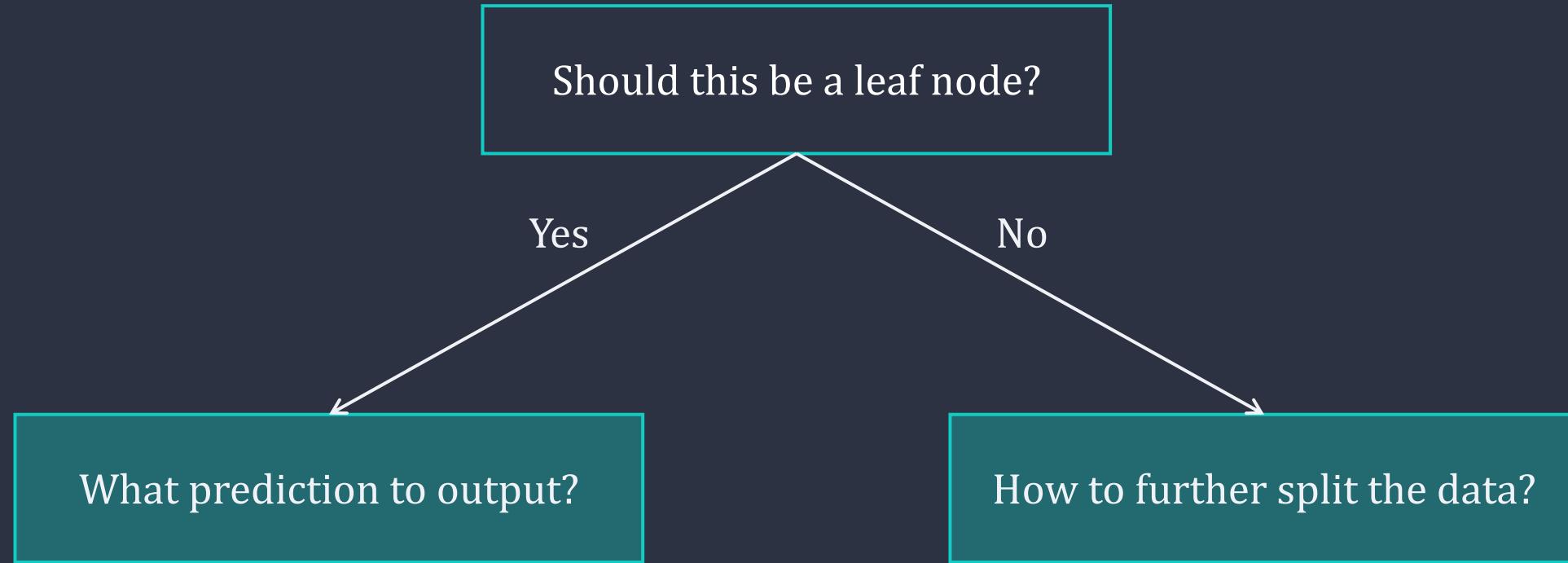
Predict real valued numbers at leaf nodes



## 2. Learning a Decision Tree

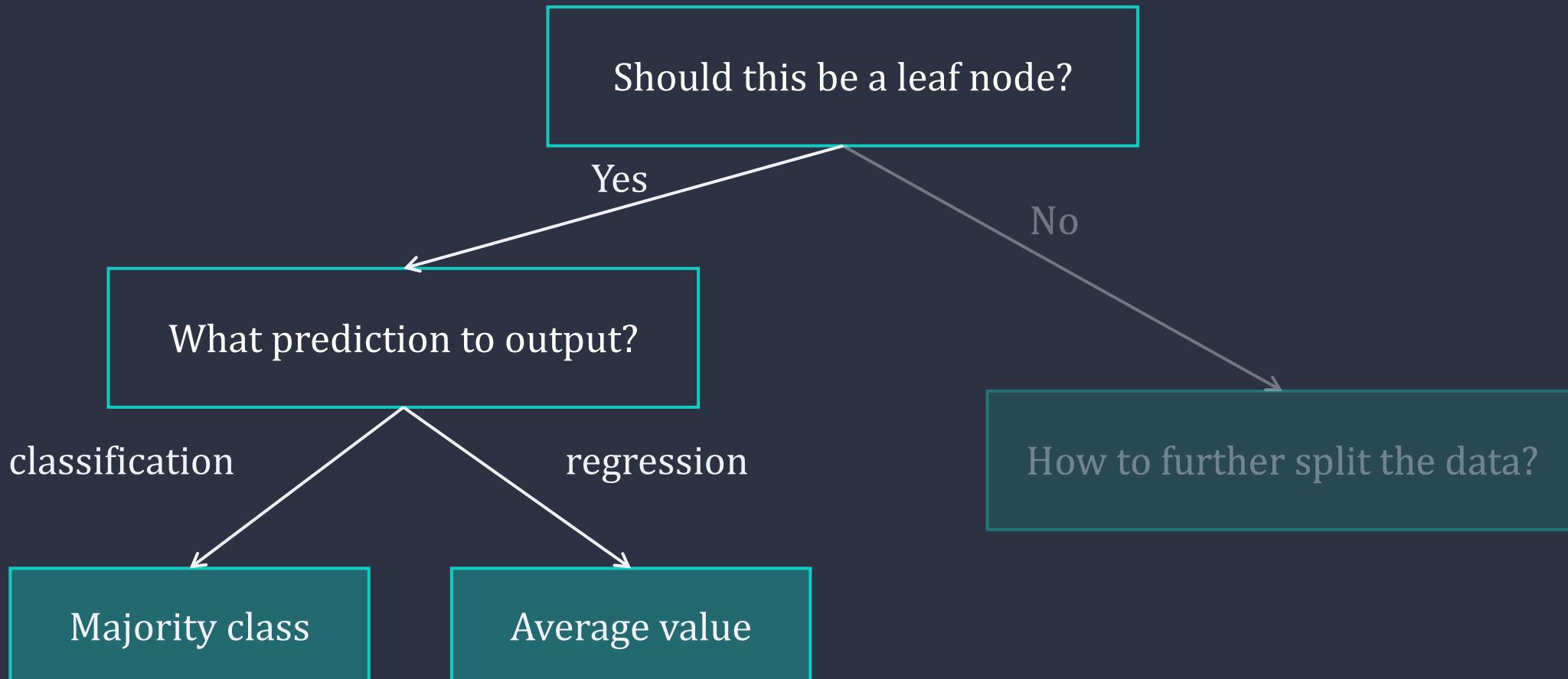
# Learning a Decision Tree

To greedily decide how to split the training data.



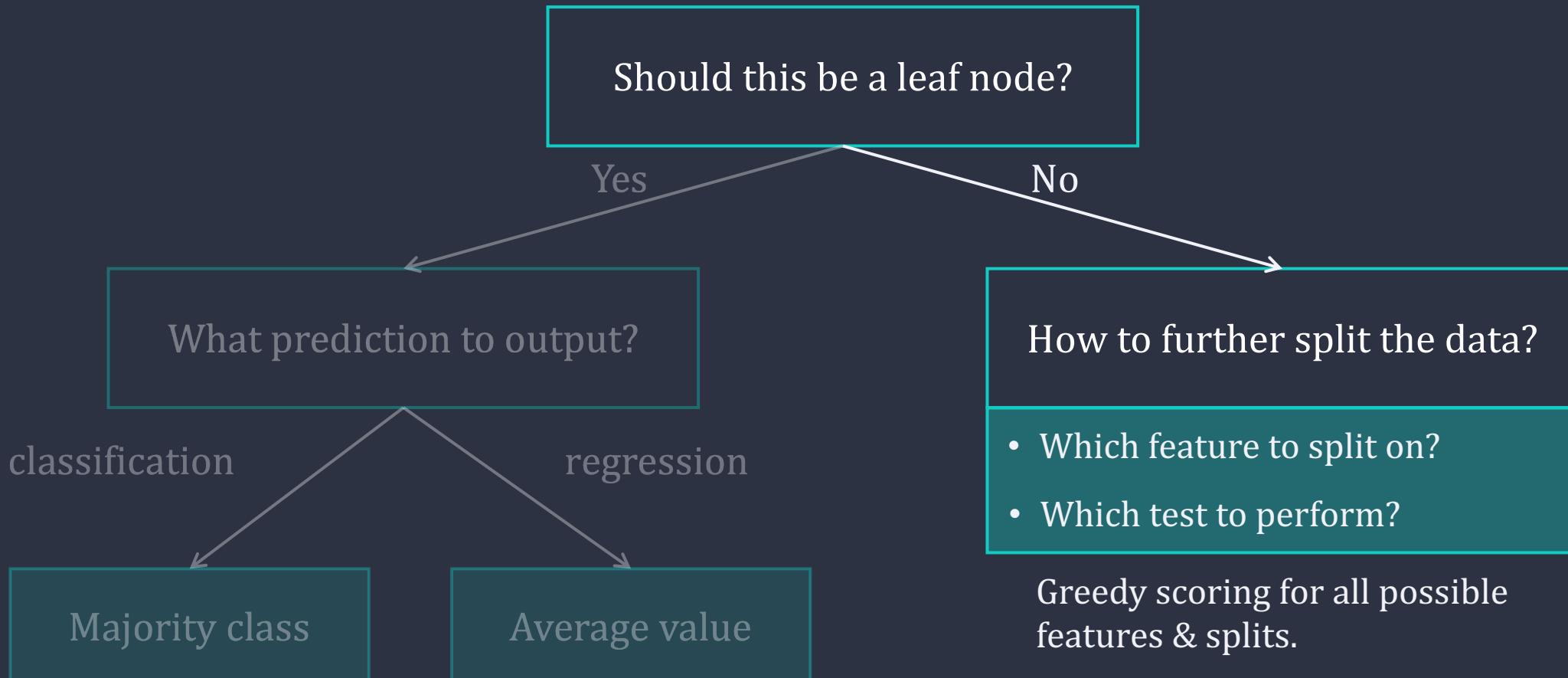
# Learning a Decision Tree

To greedily decide how to split the training data.



# Learning a Decision Tree

To greedily decide how to split the training data.



# Entropy

# Learning a Decision Tree

- A notion from Information Theory
- A measurement of Randomness / Uncertainty / Impurity

## EXAMPLE. Coin tosses

Fair coin – 50% H : 50% T (completely random)

Landing: H T H H T H T T H T T T H T H H H T H T ...

*Entropy = 1*

Biased coin – 80% H : 20% T (somewhat random)

Landing: H H H H H T H T H H H H H T H H T H H H H ...

*Entropy = 0~1*

Enchanted coin – 100% H : 0% T (completely pure)

Landing: H ...

*Entropy = 0*

# Entropy

$$H(X) = - \sum_{i=1}^K P(x_i) \log_2 P(x_i)$$

H, /'i:tə, 'eitə/: uppercase of η

# Entropy

$$H(X) = - \sum_{i=1}^K P(x_i) \log_2 P(x_i)$$

EXAMPLE. Coin tosses

Fair coin – 50% H : 50% T

Maximum entropy for 2 possible outcomes Impure

$$H(X) = -(0.5 \times \log_2 0.5 + 0.5 \times \log_2 0.5) = \log_2 2 = 1 \text{ bit}$$

Biased coin – 80% H : 20% T

$$H(X) = -(0.8 \times \log_2 0.8 + 0.2 \times \log_2 0.2) \approx 0.772 \text{ bits}$$

Enchanted coin – 100% H : 0% T

Minimum entropy for 1 possible outcome Pure

$$H(X) = -(1 \times \log_2 1 + 0 \times \log_2 0) = 0 \text{ bits}$$

# Information Gain

- A notion from Information Theory
- A measurement of Reduction in Entropy caused by splitting a dataset based on a random variable with a range of values.

$$IG(S, a) = H(S) - H(S|a)$$

$IG(S, a)$ : the information gain for dataset  $S$  for the random variable  $a$

$H(S)$ : priori entropy (entropy for dataset  $S$  before any split)

$H(S|a)$ : conditional entropy for dataset  $S$  given the variable  $a$

The  $IG(S, a)$  formula describes the information gained in bits when splitting the dataset  $S$  using the random variable  $a$ .

Information Gain  $IG(S, a) = H(S) - H(S|a)$

$H(S|a)$ : conditional entropy for dataset  $S$  given the variable  $a$

$$H(S|a) = \sum_{v \in values(a)} \frac{|S_a(v)|}{|S|} \cdot H(S_a(v))$$

$v \in values(a)$  defines a *split of dataset S* into mutually exclusive and all-inclusive subsets, for each observed value  $v$  of the feature  $a$ .

$|S_a(v)|/|S|$  is the ratio of data points in each subset out of the entire dataset  $S$ , with variable  $a$  having value  $v$ .

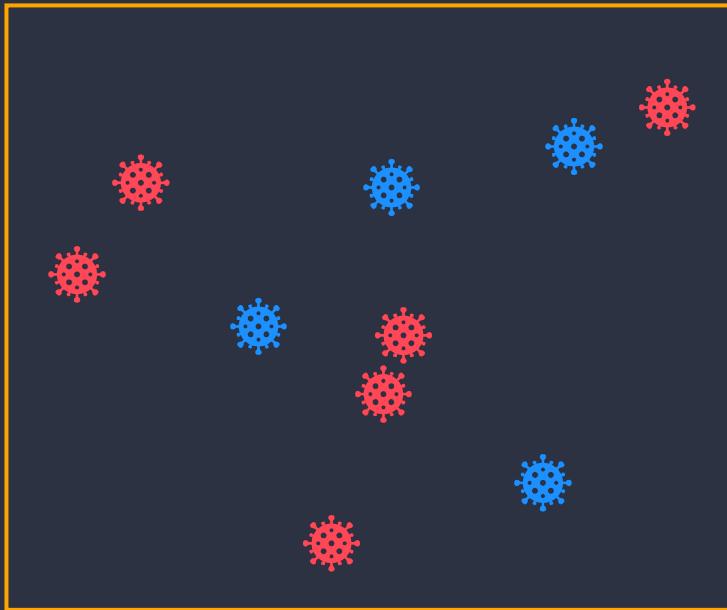
$H(S_a(v))$  is the entropy of the subset of data points , with variable  $a$  having value  $v$ .

## EXAMPLE. COVID-19 affected

index (useless)	features (candidate split factor)			output/prediction
patient	high temperature	continuous cough	loss of sense of smell	Affected (classification)
1	T	T	T	T 
2	T	F	T	T 
3	T	T	F	F 
4	F	T	T	T 
5	T	F	F	F 
6	F	F	F	F 
7	T	T	T	T 
8	T	F	T	F 
9	T	T	T	T 
10	F	T	T	T 

## EXAMPLE. COVID-19 affected

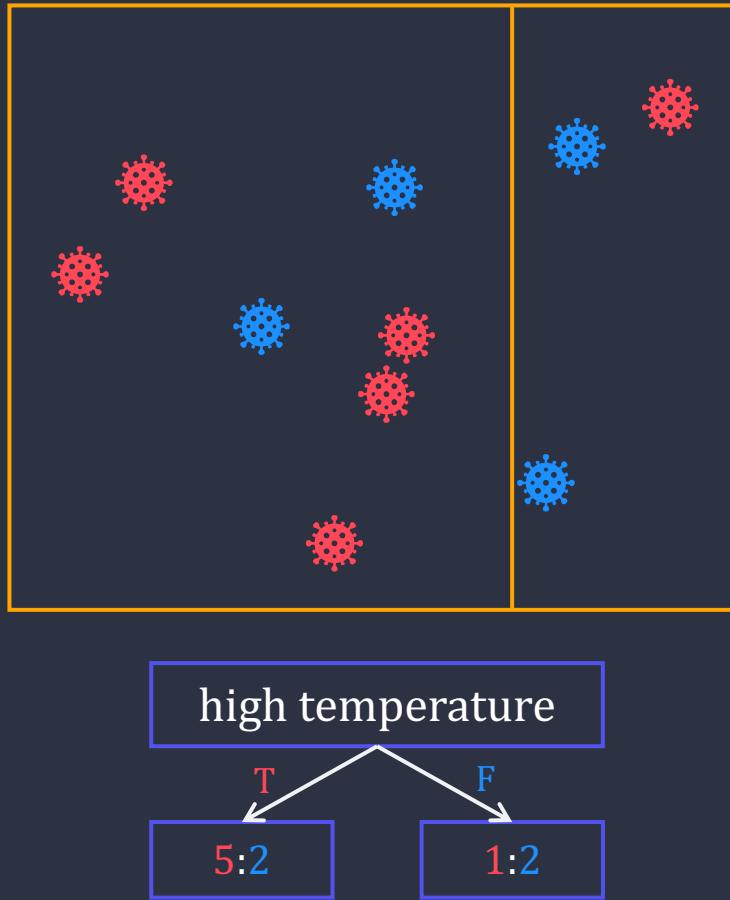
Entropy of the whole dataset, before any split.



$$\begin{aligned} H(X) &= - \sum_{i=1}^K P(x_i) \log_2 P(x_i) \\ &= - \left( \frac{6}{10} \log_2 \frac{6}{10} + \frac{4}{10} \log_2 \frac{4}{10} \right) \\ &= -((-0.442) + (-0.529)) \\ &= 0.971 \text{ (bits)} \end{aligned}$$

## EXAMPLE. COVID-19 affected

Entropy of the whole dataset, before any split.  $H(X) = 0.971$  (bits)



$$H(X_a(v)) = -(P(T) \log_2 P(T) + P(F) \log_2 P(F))$$

$$H(X_{hi\_temp}(T)) = -\left(\frac{5}{7} \log_2 \frac{5}{7} + \frac{2}{7} \log_2 \frac{2}{7}\right) = 0.863 \text{ (bits)}$$

$$H(X_{hi\_temp}(F)) = -\left(\frac{1}{3} \log_2 \frac{1}{3} + \frac{2}{3} \log_2 \frac{2}{3}\right) = 0.918 \text{ (bits)}$$

$$H(X|a) = \frac{|X_{hi\_temp}(T)|}{|X|} H(X_{hi\_temp}(T)) + \frac{|X_{hi\_temp}(F)|}{|X|} H(X_{hi\_temp}(F))$$

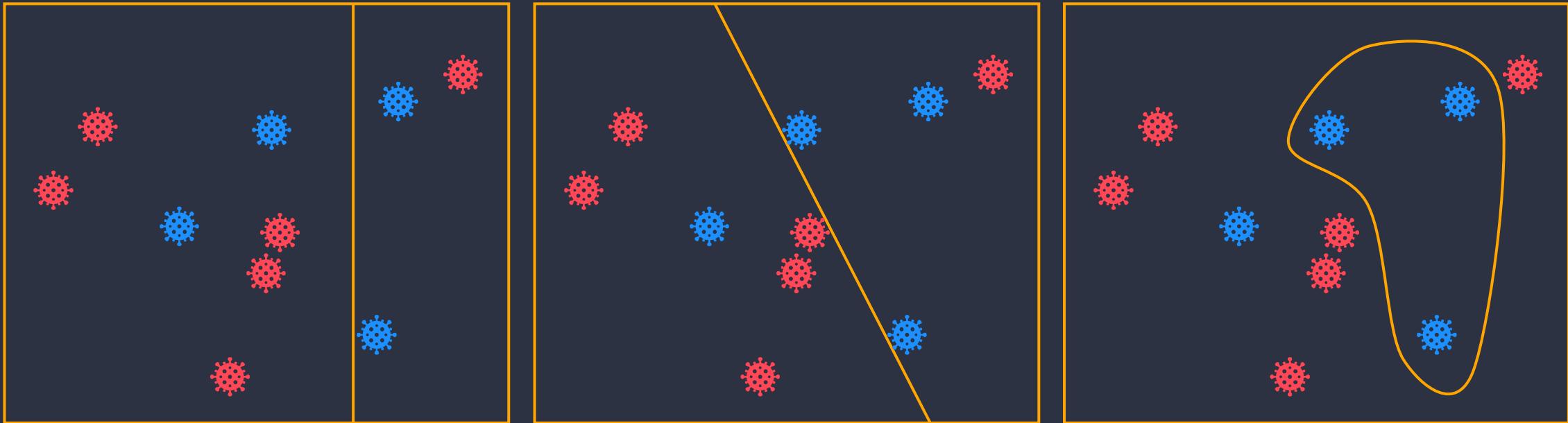
$$= \frac{7}{10} \times 0.863 + \frac{3}{10} \times 0.918 = 0.880 \text{ (bits)}$$

$$IG(X, hi\_temp) = H(X) - H(X|a) = 0.971 - 0.880 = 0.091 \text{ (bits)}$$

(IG very small: using this feature to split the data isn't ideal)

## EXAMPLE. COVID-19 affected

Entropy of the whole dataset, before any split.  $H(X) = 0.971$  (bits)

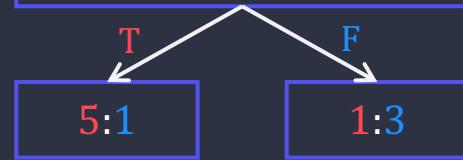


high temperature



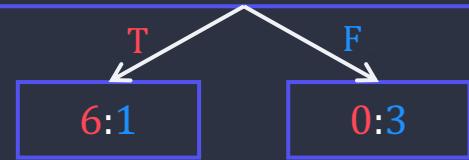
$$IG(S, hi\_temp) = 0.091 \text{ (bits)}$$

continuous cough



$$IG(S, con\_cou) = 0.257 \text{ (bits)}$$

loss of sense of smell



$$IG(S, lo\_sesm) = 0.557 \text{ (bits)}$$

Largest Information Gain

## EXAMPLE. COVID-19 affected

Level 0

loss of sense of smell

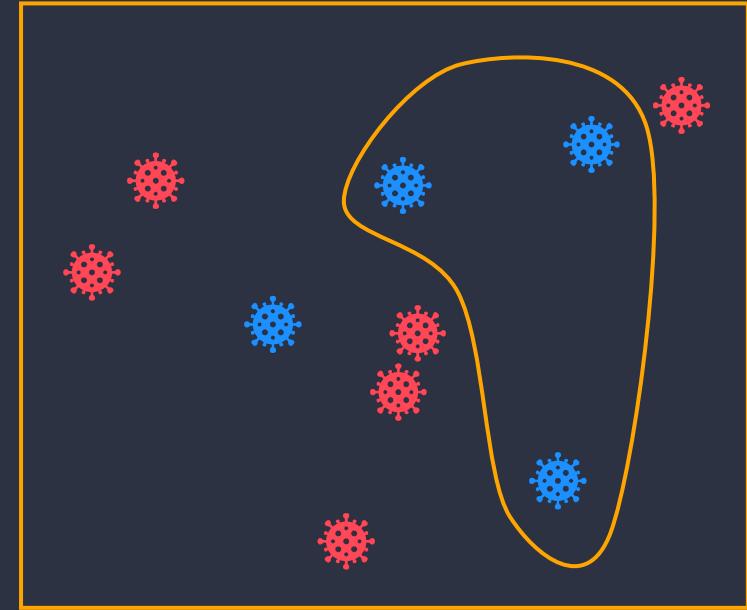
T

F

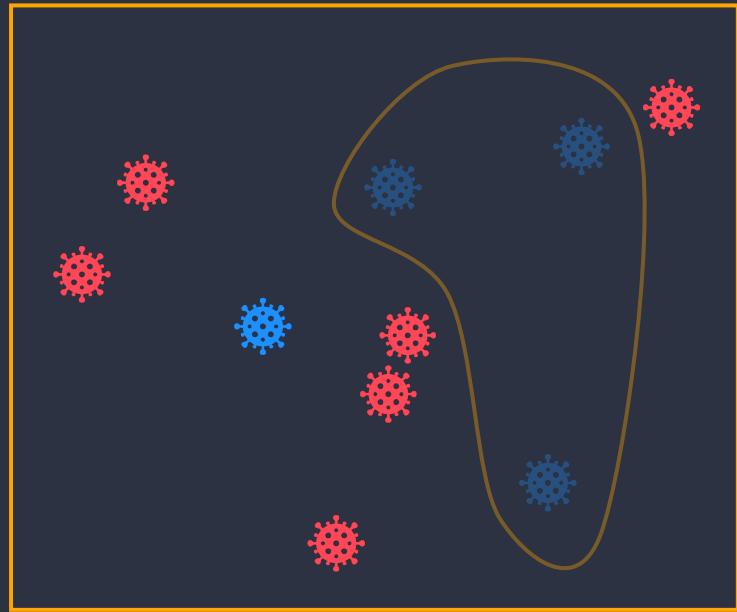
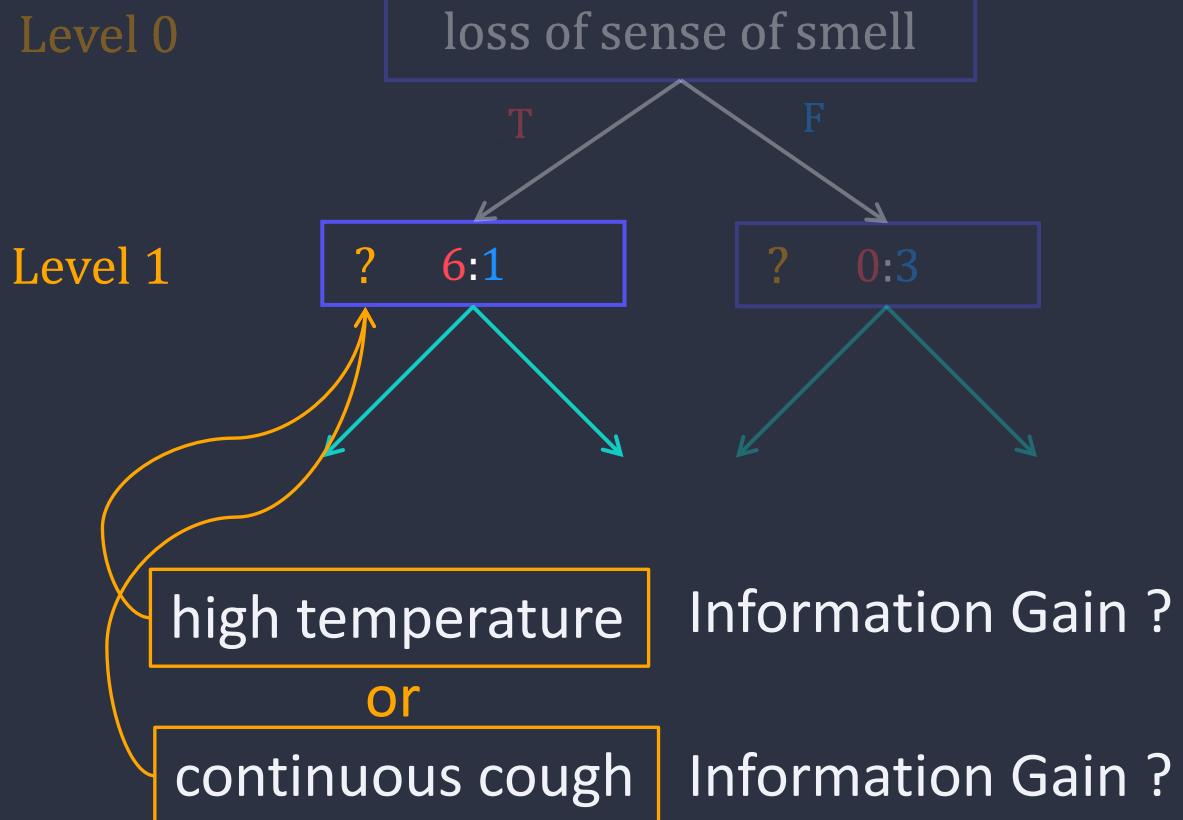
Level 1

? 6:1

? 0:3



## EXAMPLE. COVID-19 affected



## EXAMPLE. COVID-19 affected

Level 0

loss of sense of smell

T

F

Level 1

? 6:1

? 0:3

high temperature

or

continuous cough

Information Gain ?

Information Gain ?



# Gini Impurity

# Gini Impurity

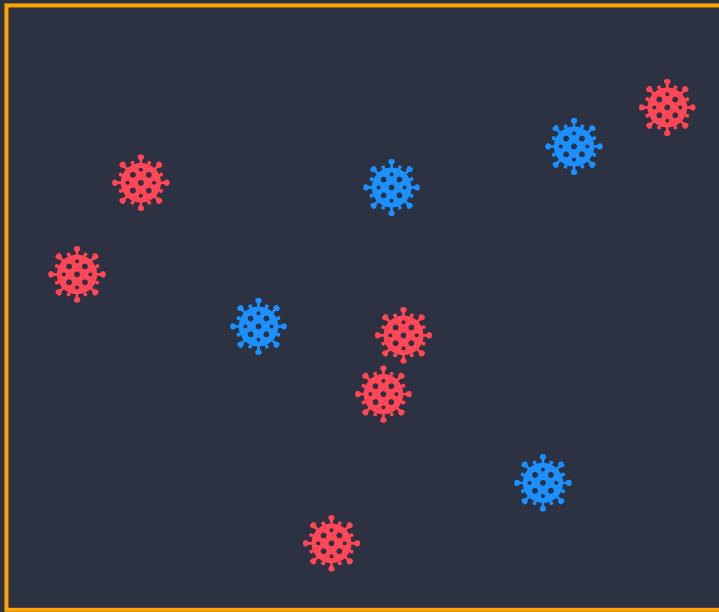
- An alternative to Information Gain
- Measures variance (instead of entropy)

$$\begin{aligned} I_G(X) &= \sum_{i=1}^K \left( P(x_i) \sum_{j \neq i} P(x_j) \right) = \sum_{i=1}^K P(x_i)(1 - P(x_i)) \\ &= \sum_{i=1}^K \left( P(x_i) - (P(x_i))^2 \right) = \sum_{i=1}^K P(x_i) - \sum_{i=1}^K (P(x_i))^2 \\ &= 1 - \sum_{i=1}^K (P(x_i))^2 \end{aligned}$$

EXAMPLE.

COVID-19 affected Gini Index to decide which feature to use for split

Gini Impurity of the whole dataset, before any split.



$$\begin{aligned}I_G(X) &= 1 - \sum_{i=1}^K (P(x_i))^2 \\&= 1 - \left( \left(\frac{6}{10}\right)^2 + \left(\frac{4}{10}\right)^2 \right) \\&= 1 - (0.36 + 0.16)\end{aligned}$$

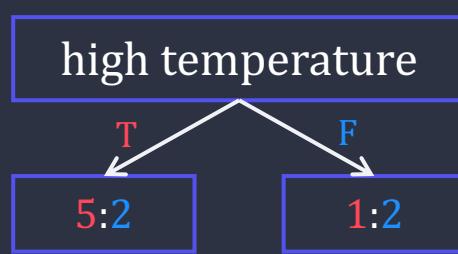
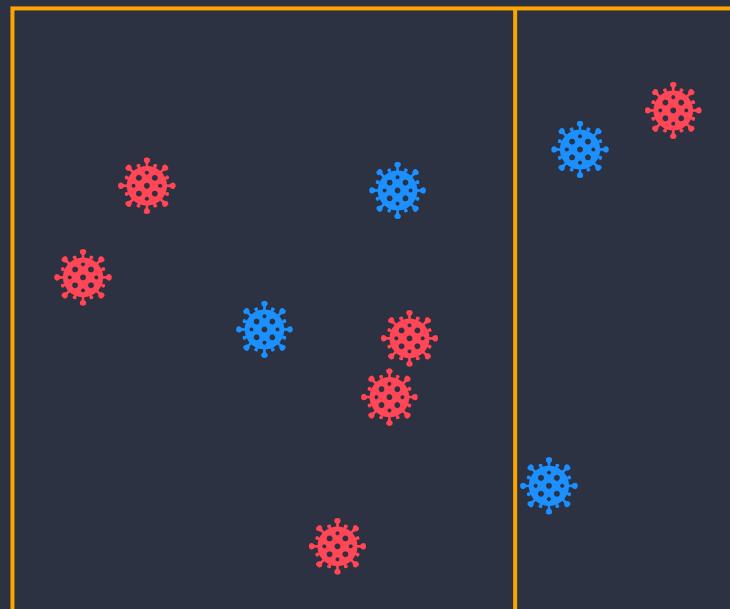
$$= 0.48$$

EXAMPLE.

COVID-19 affected

Gini Index to decide which feature to use for split

Gini Impurity of the whole dataset, before any split.  $I_G(X) = 0.48$



$$I_G(X) = 1 - (P(T)^2 + P(F)^2)$$

$$I_G(X_{hi\_temp}(T)) = 1 - \left( \left(\frac{5}{7}\right)^2 + \left(\frac{2}{7}\right)^2 \right) = 0.408$$

$$I_G(X_{hi\_temp}(F)) = 1 - \left( \left(\frac{1}{3}\right)^2 + \left(\frac{2}{3}\right)^2 \right) = 0.445$$

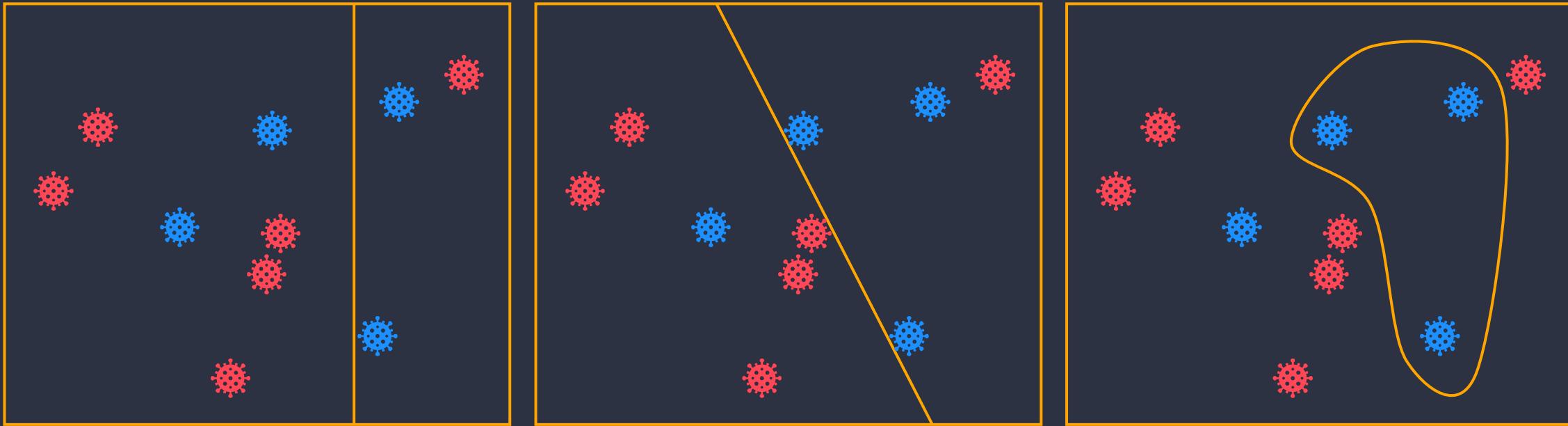
$$I_G(X|a) = \frac{|X_{hi\_temp}(T)|}{|X|} H(X_{hi\_temp}(T)) + \frac{|X_{hi\_temp}(F)|}{|X|} H(X_{hi\_temp}(F))$$

$$= \frac{7}{10} \times 0.408 + \frac{3}{10} \times 0.445 = 0.419$$

$$GI(X, hi\_temp) = I_G(X) - I_G(X|a) = 0.480 - 0.419 = 0.061$$

EXAMPLE. COVID-19 affected Gini Index to decide which feature to use for split

Gini Impurity of the whole dataset, before any split.  $I_G(X) = 0.48$

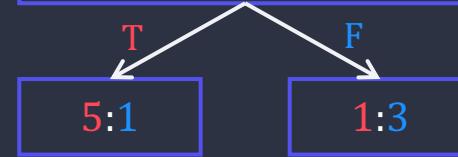


high temperature



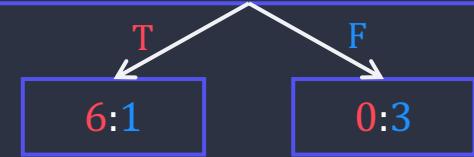
$$GI(X, hi\_temp) = 0.061$$

continuous cough



$$GI(X, con\_cou) = 0.171$$

loss of sense of smell



$$GI(S, lo\_sesm) = 0.309$$

Largest Gini Index

# Feature Value

# Feature Value - Categorical Feature

Covid-19 Affected

Features: continuous cough

T / F

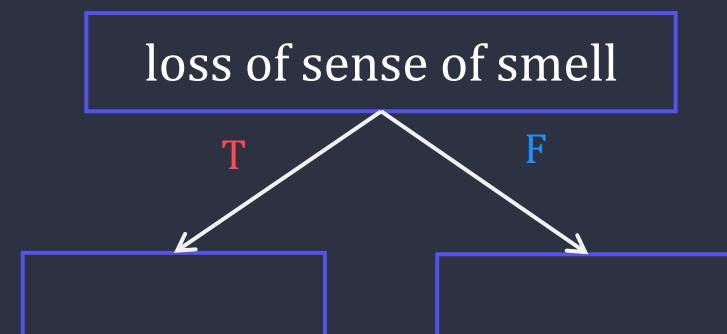
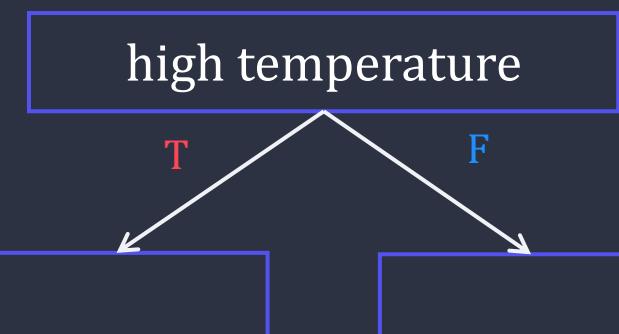
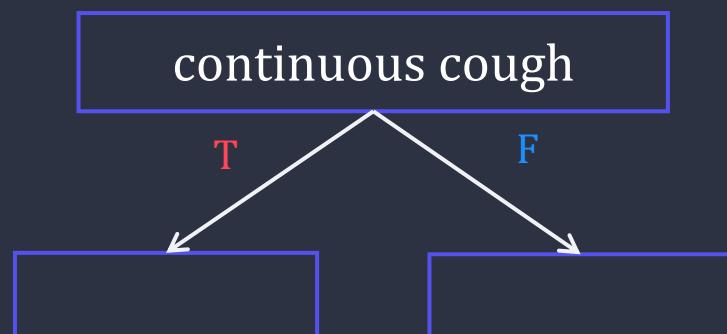
high temperature

T / F

loss of sense of smell

T / F

Split: Directly use the value of the feature



# Feature Value - Numerical Feature

EXAMPLE. weight predicts heart disease

weight (kg)	heart disease
120	T
130	F
110	F
180	T
150	T

# Feature Value - Numerical Feature

EXAMPLE. weight predicts heart disease

	weight (kg)	heart disease
highest	180	T
	150	T
	130	F
	120	T
lowest	110	F



STEP 1

Sort patient by weight,  
from the highest to the  
lowest

# Feature Value - Numerical Feature

EXAMPLE. weight predicts heart disease

	weight (kg)	heart disease
candidate threshold 1	165	T
candidate threshold 2	140	T
candidate threshold 3	125	F
candidate threshold 4	115	T
	180	
	150	
	130	
	120	
	110	

STEP 2

Calculate the average  
weight for all adjacent  
patients.

# Feature Value - Numerical Feature

EXAMPLE. weight predicts heart disease

	weight (kg)	heart disease	
Impurity ?	165	T	STEP 3
Impurity ?	140	T	Calculate the impurity values for each average weight.
Impurity ?	125	F	(Entropy / Gini Impurity)
Impurity ?	115	T	
	180	F	
	150		
	130		
	120		
	110		

# Feature Value - Numerical Feature

EXAMPLE. weight predicts heart disease

	weight (kg)	heart disease	
	180	T	STEP 4
Max?	Impurity ? ← 165	T	Calculate the impurity change for each candidate
	150	F	Choose the largest change
	Impurity ? ← 140	T	(Information Gain / Gini Index)
	130	F	
	Impurity ? ← 125	T	
	120	F	
	Impurity ? ← 115	F	
	110		

# Feature Value - Ranked Feature

EXAMPLE. rating on Lei's ML module predicts fist class honours

rating on ML module

first class honours



F



T



F



T



T

# Feature Value - Ranked Feature

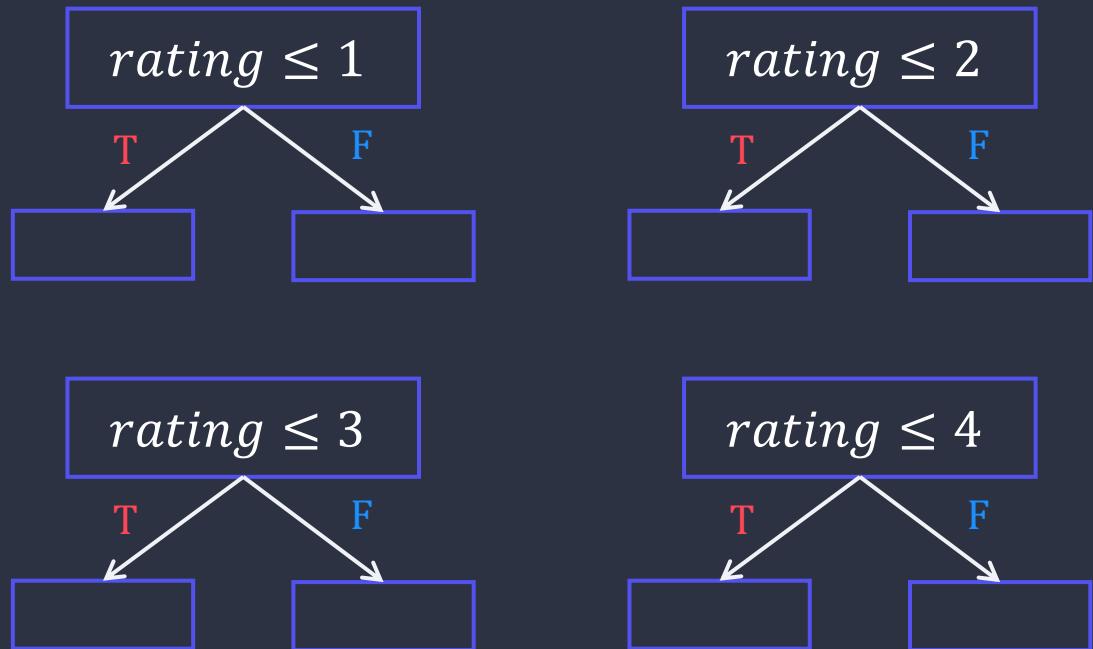
EXAMPLE. rating on Lei's ML module predicts fist class honours

rating on ML module



first class honours

F T F T F T



# Feature Value - Multiple Choice Feature

## EXAMPLE. favourite cuisine predicts tea lover

# favourite cuisine

# is tea lover

## Japanese



F

Chinese



T

## French



F

# Japanese



T

# Chinese



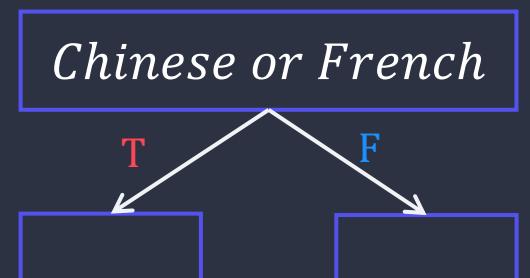
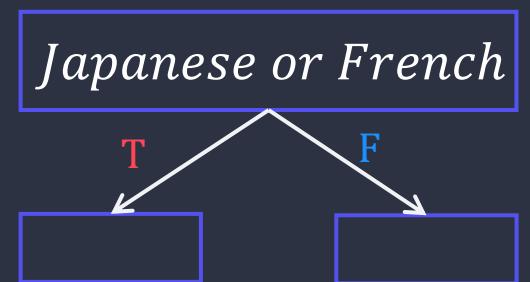
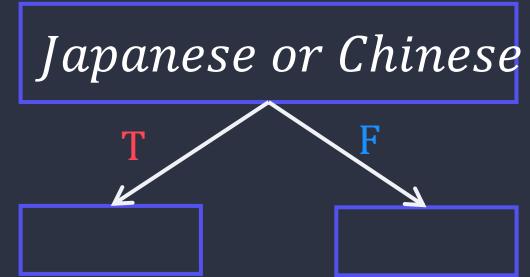
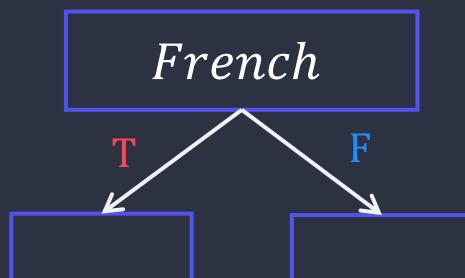
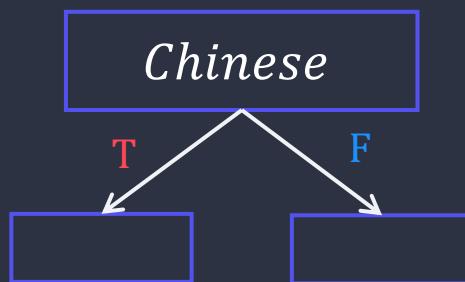
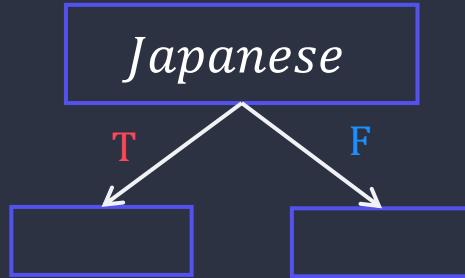
T

# Feature Value - Multiple Choice Feature

EXAMPLE. favourite cuisine predicts tea lover

favourite cuisine      is tea lover

Japanese		F
Chinese		T
French		F
Japanese		T
Chinese		T



# Coding with sci-kit learn

## Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import classification_report, confusion_matrix  
  
clf = DecisionTreeClassifier()  
clf.fit(X_train, y_train)  
  
y_pred = clf.predict(X_test)  
  
print(confusion_matrix(y_test,y_pred))  
print(classification_report(y_test,y_pred))
```

# Coding with sci-kit learn

## Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics

regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print(metrics.mean_absolute_error(y_test, y_pred))
print(metrics.mean_squared_error(y_test, y_pred))
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

# Summary

# Summary

1. Decision Trees
2. Learning a Decision Tree

Next Lecture

Ensemble Methods

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Ensemble Methods

Dr Yang Long

# Previously

## Machine Learning Algorithms



Regression



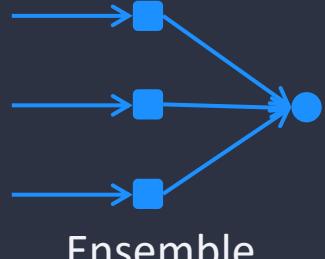
Regularisation



Clustering



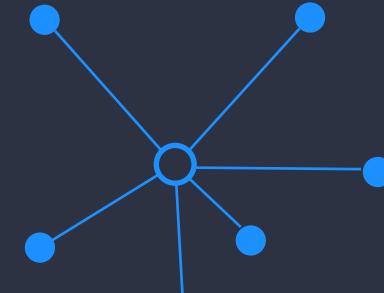
Bayesian



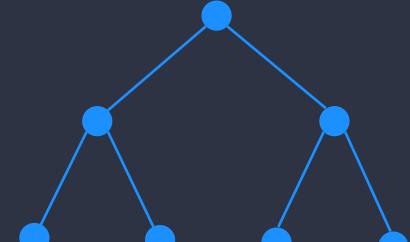
Ensemble



Neural Network



Instance-based



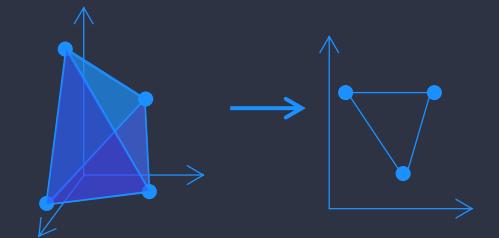
Tree-based



Associated Rule Learning



Deep Learning



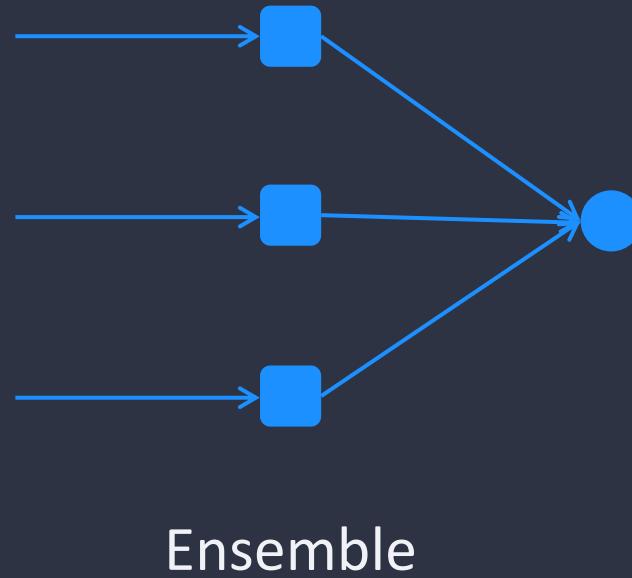
Dimensionality Reduction



Reinforcement Learning

# Lecture Overview

1. Intuition
2. Bagging
3. Boosting
4. Stacking
5. Random Forest



# Intuition

“

## wisdom of crowds

”

The aggregate decisions made by a large group will often be better than those of its individual expert member, i.e., collective knowledge is often better than knowledge of the few.

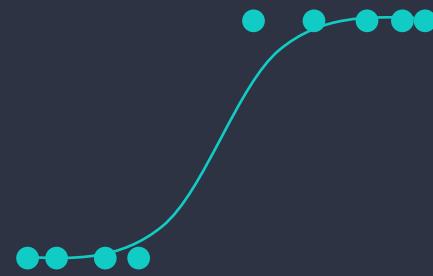
The essence of ensemble methods is to let a group (ensemble) of methods work collectively, to achieve a better prediction result.

# Intuition

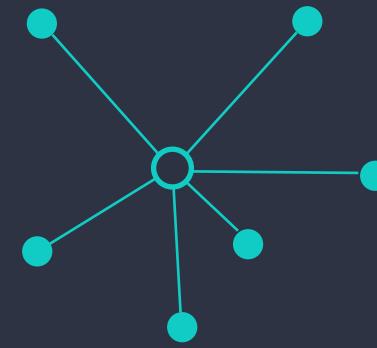
## Single Model



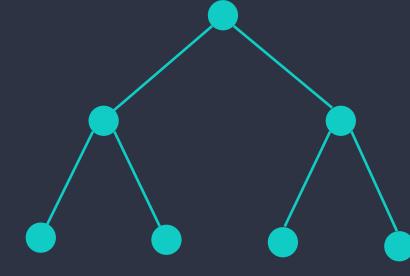
Linear Regression



Logistic Regression



K Nearest Neighbours



Decision Tree

...

Using one single model to make predictions.

# Intuition

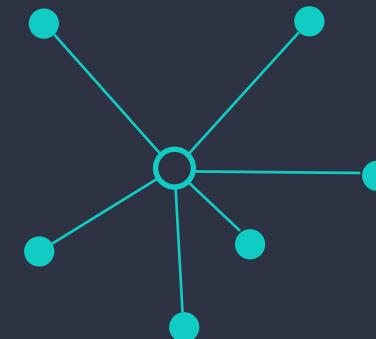
## Single Model



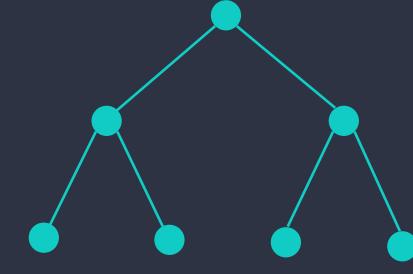
Linear Regression



Logistic Regression

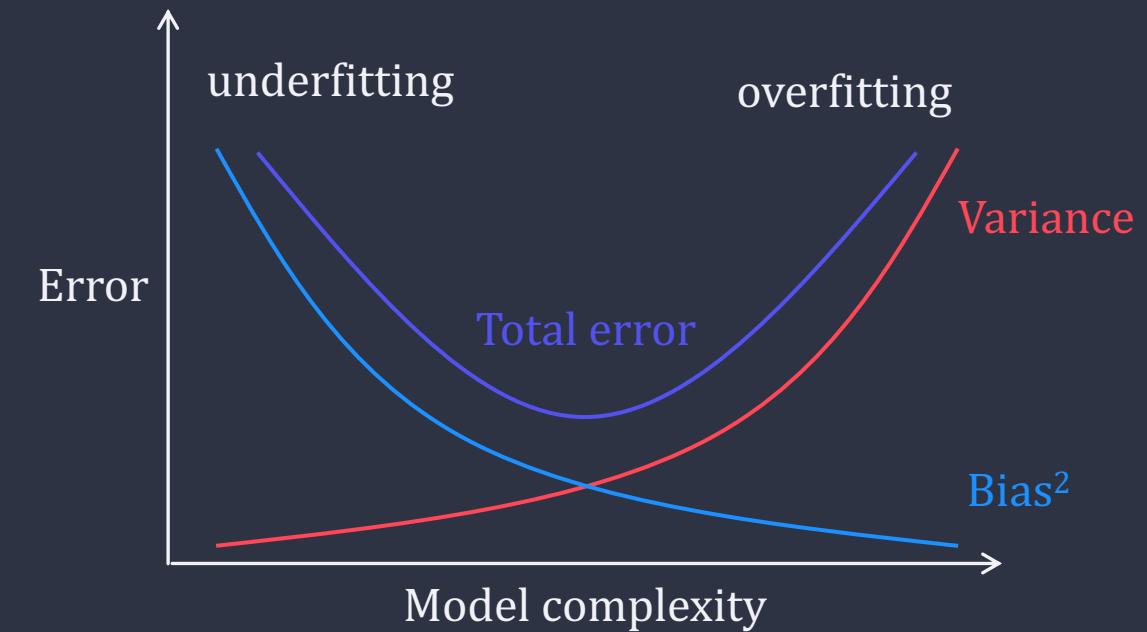


K Nearest Neighbours



Decision Tree

...

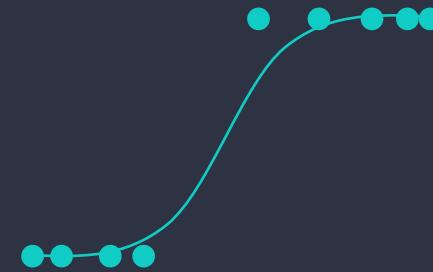


# Intuition

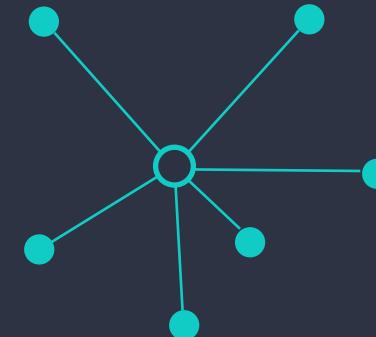
## Single Model



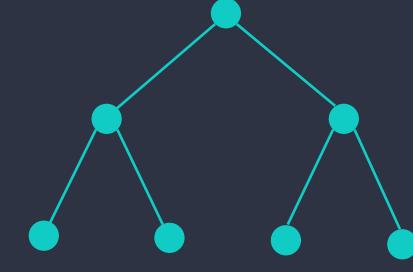
Linear Regression



Logistic Regression



K Nearest Neighbours



Decision Tree

...

## Weak Learners (base learners/models)

- Not performing well individually due to possible high variance or high bias.
- Can be used as building blocks to form a stronger (more complex) learner/model.

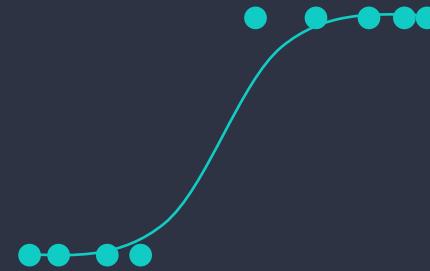
# Intuition

## Single Model

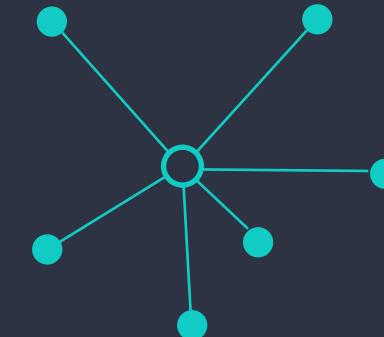


Linear Regression

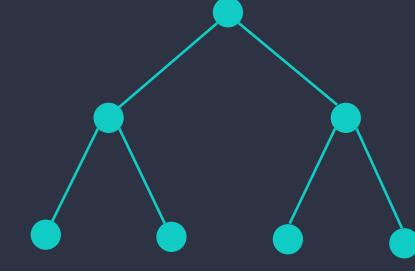
## Weak Learners (base learners/models)



Logistic Regression



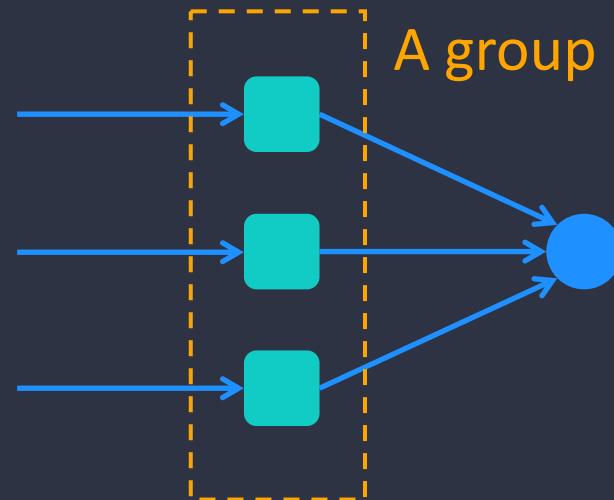
K Nearest Neighbours



Decision Tree

...

## Ensemble Model



A group of Base Learners working collectively

e.g. Random Forests, using a group of decision tree models to make predictions.

# Intuition

Combining **Weak Learners** (base learners/models)



Selecting base learners

- Homogenous → **Bagging** **Boosting**
- Heterogenous → **Stacking**

Base learner selection should be coherent with how they are aggregated

- Base learners with high variance → Combination method to reduce variance
- Base learners with high bias → Combination method to reduce bias

# Intuition

## Ensemble methods (meta-algorithms)

Bagging

Boosting

Stacking

# Intuition

## Ensemble methods (meta-algorithms)

Bagging

- often considers **homogeneous** base learners
- learns them **independently** and in parallel
- combines them using **deterministic algorithm**

Boosting

- often considers **homogeneous** base learners
- learns them **sequentially** and in adaptative way
- combines them using **deterministic algorithm**

Stacking

- often considers **heterogeneous** base learners
- learns them **independently** and in parallel
- combines them using **meta-learning algorithm**

## 2. Bagging

# Bagging

Bootstrap

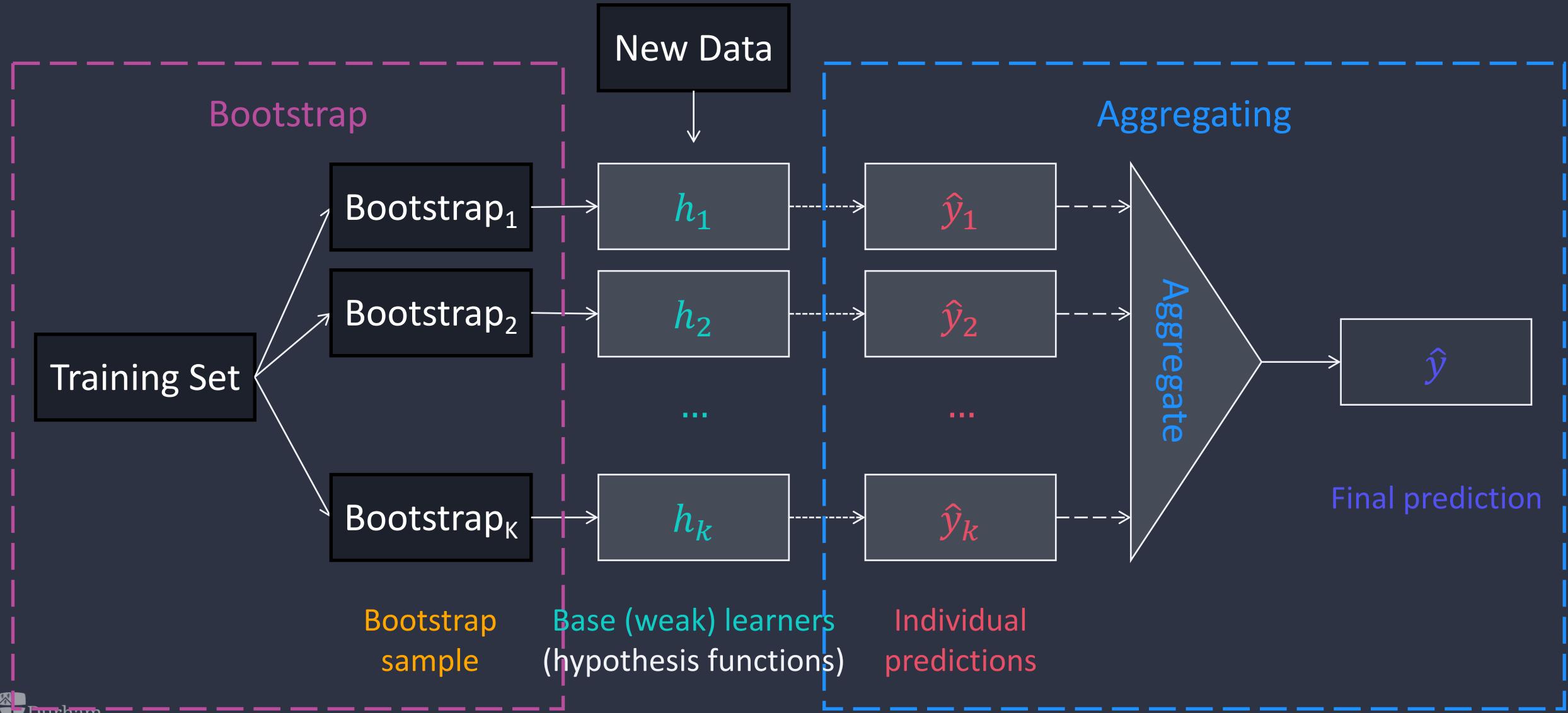
+

Aggregating

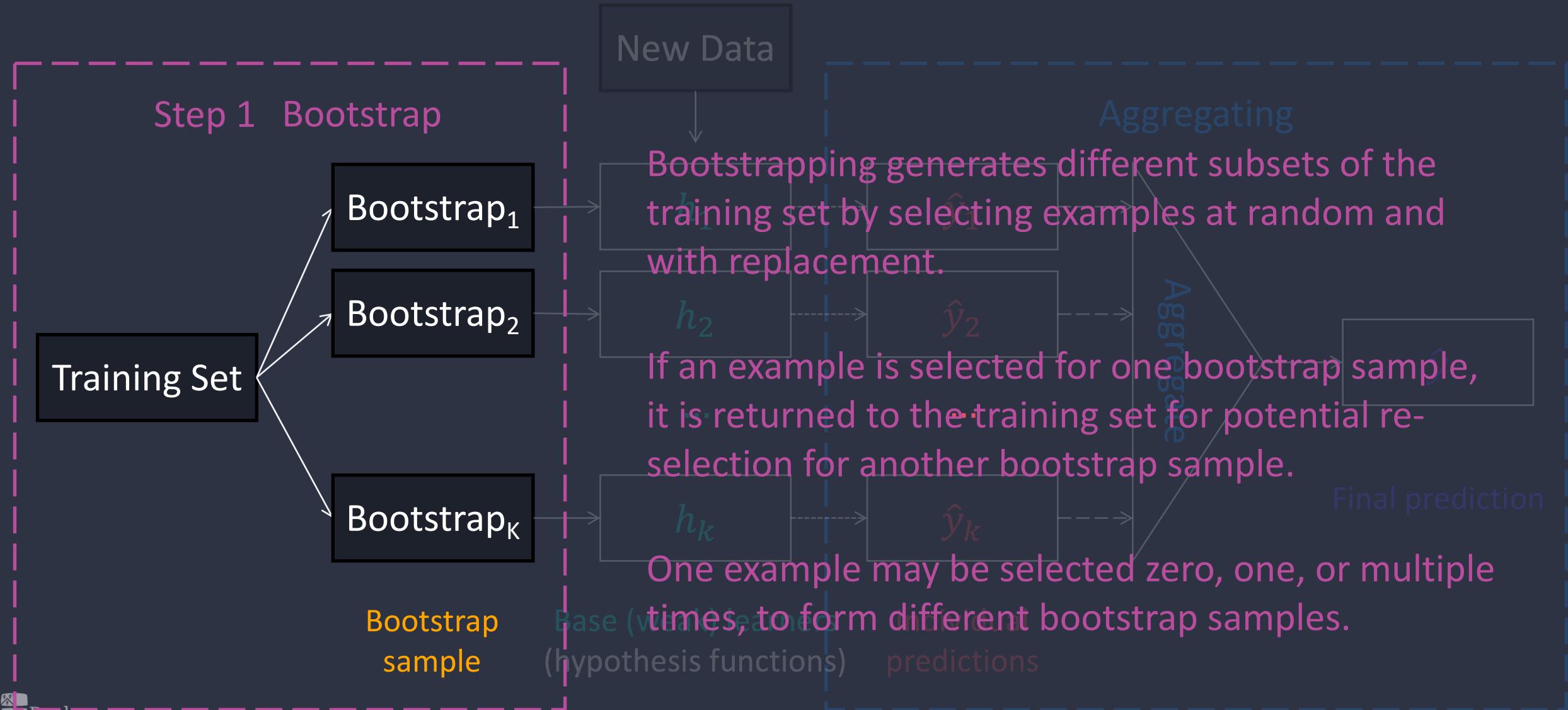
Bootstrapping sampling  
(resampling) technique to create  
diverse samples.

Average or majority of the  
individual predictions to output a  
more accurate estimate.

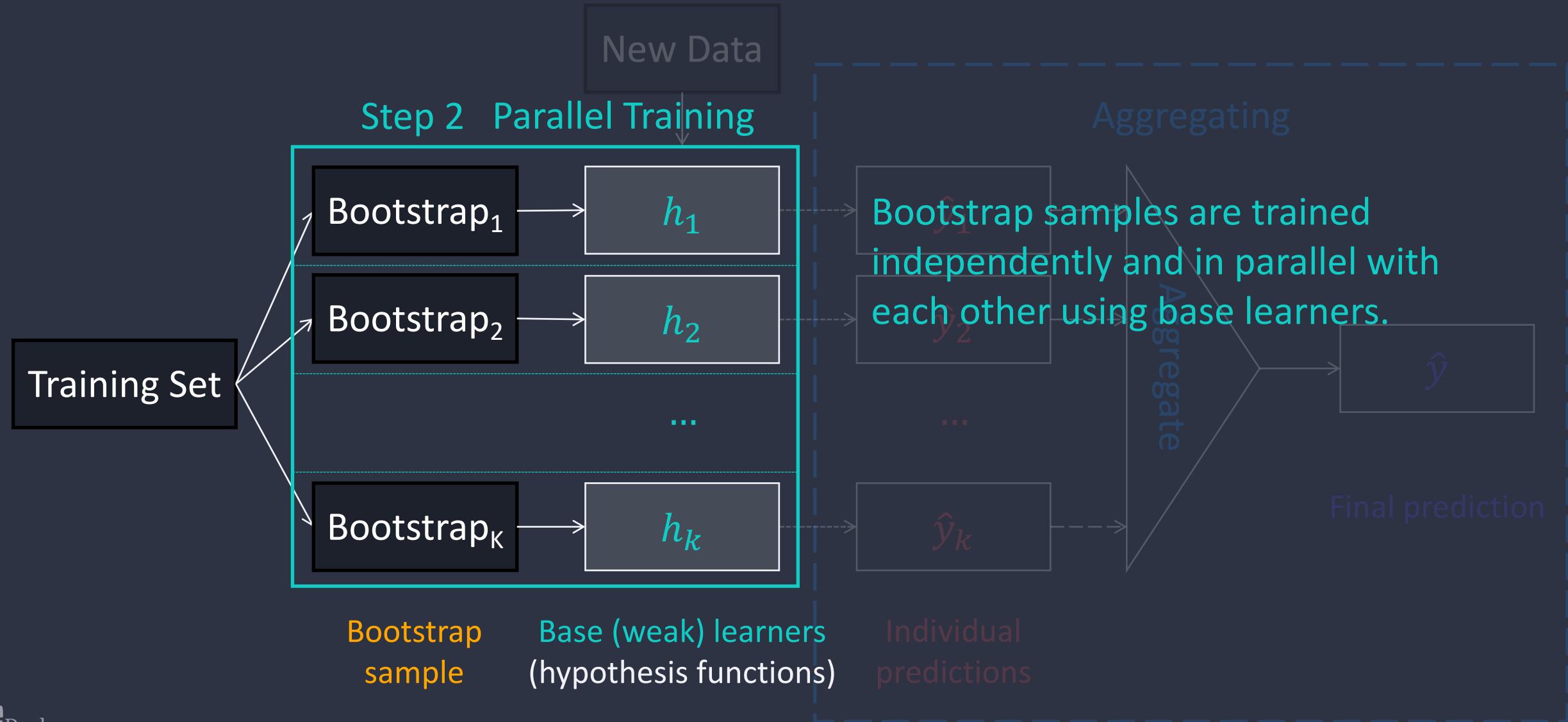
# Bagging



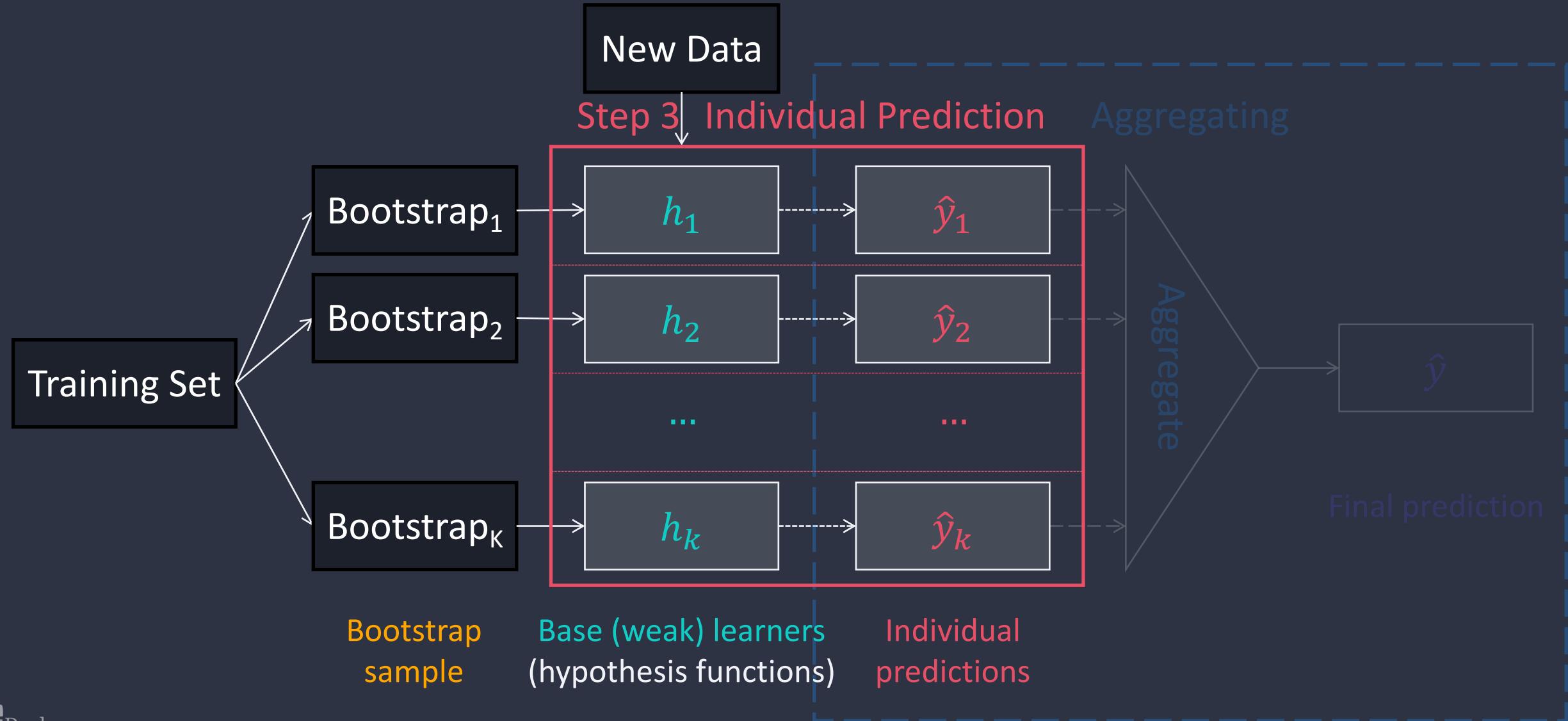
# Bagging



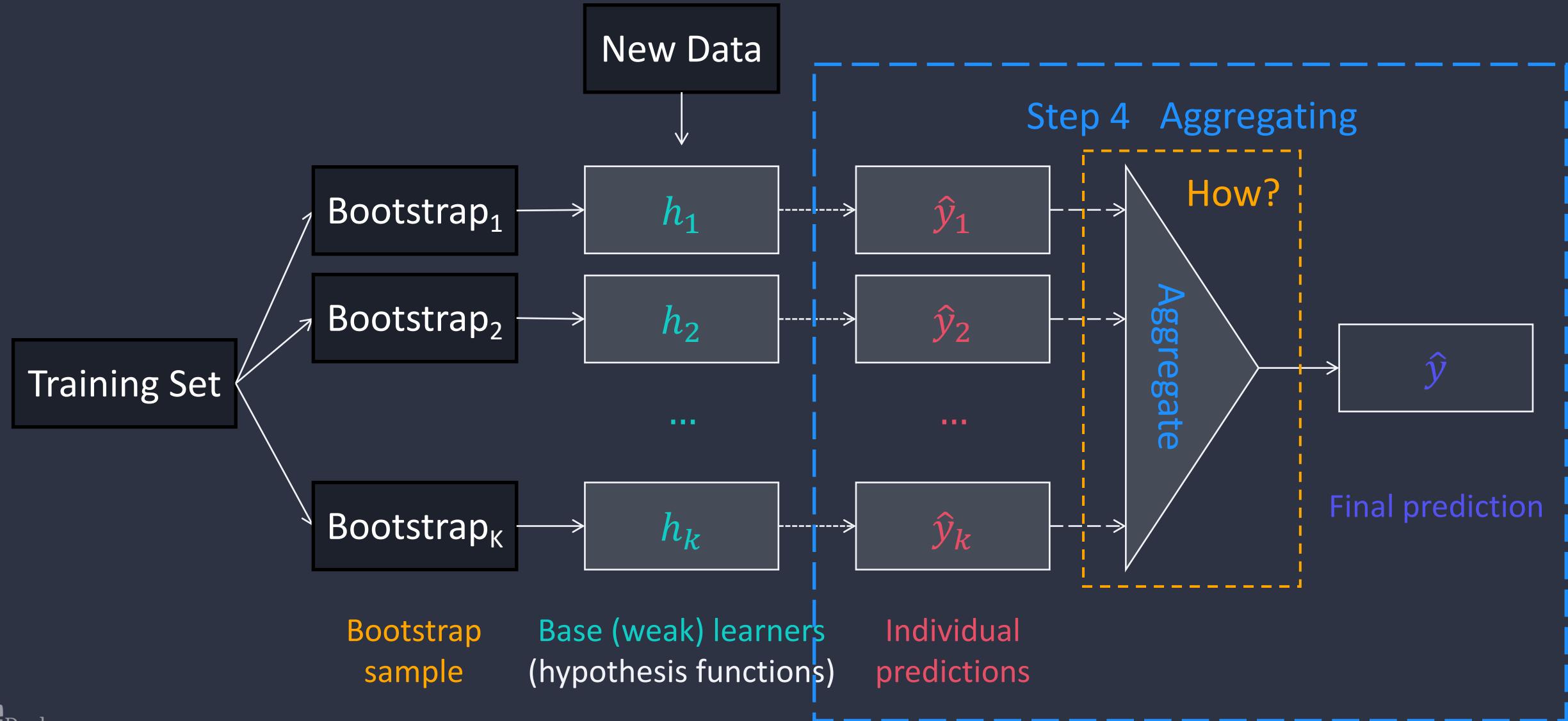
# Bagging



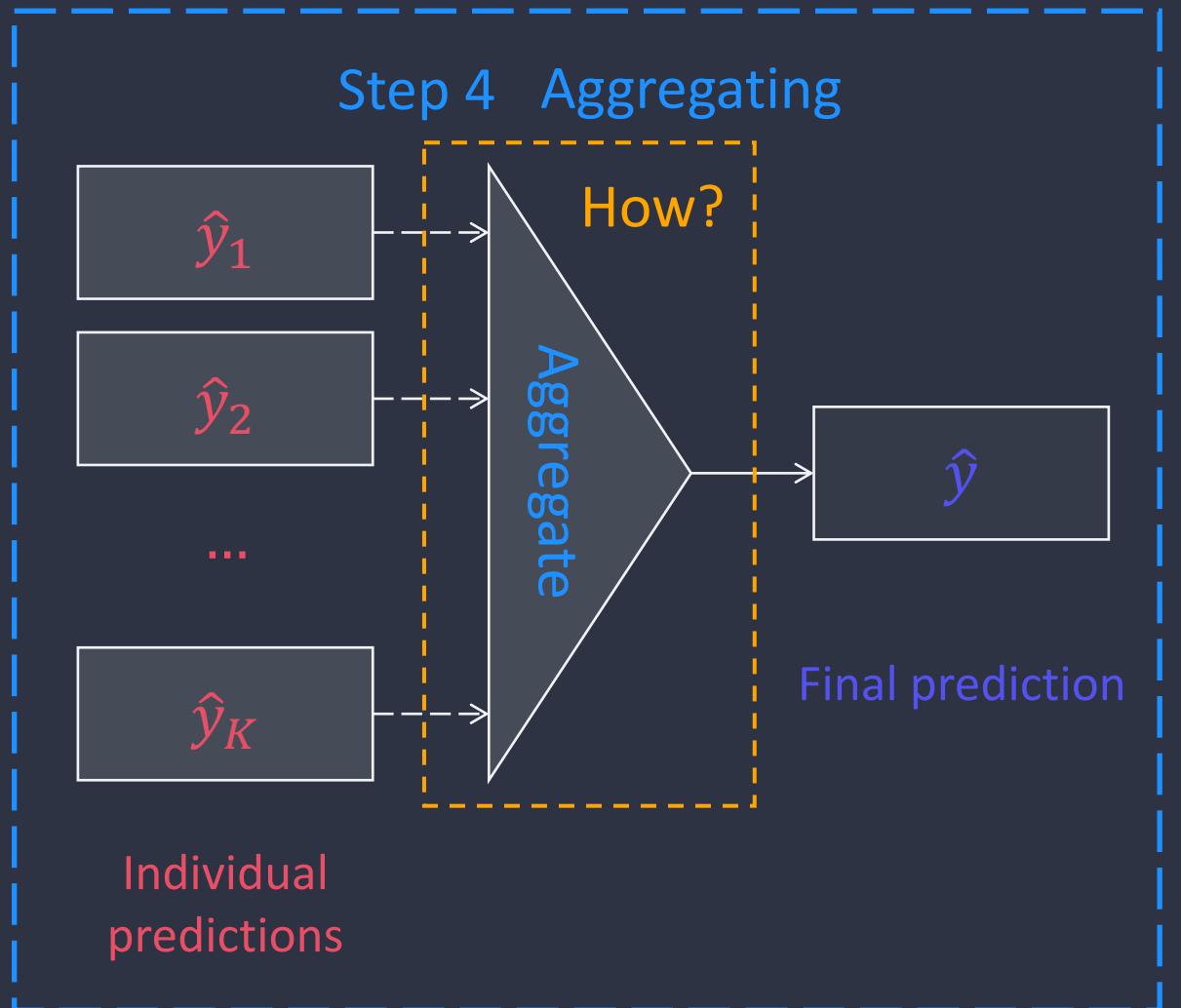
# Bagging



# Bagging

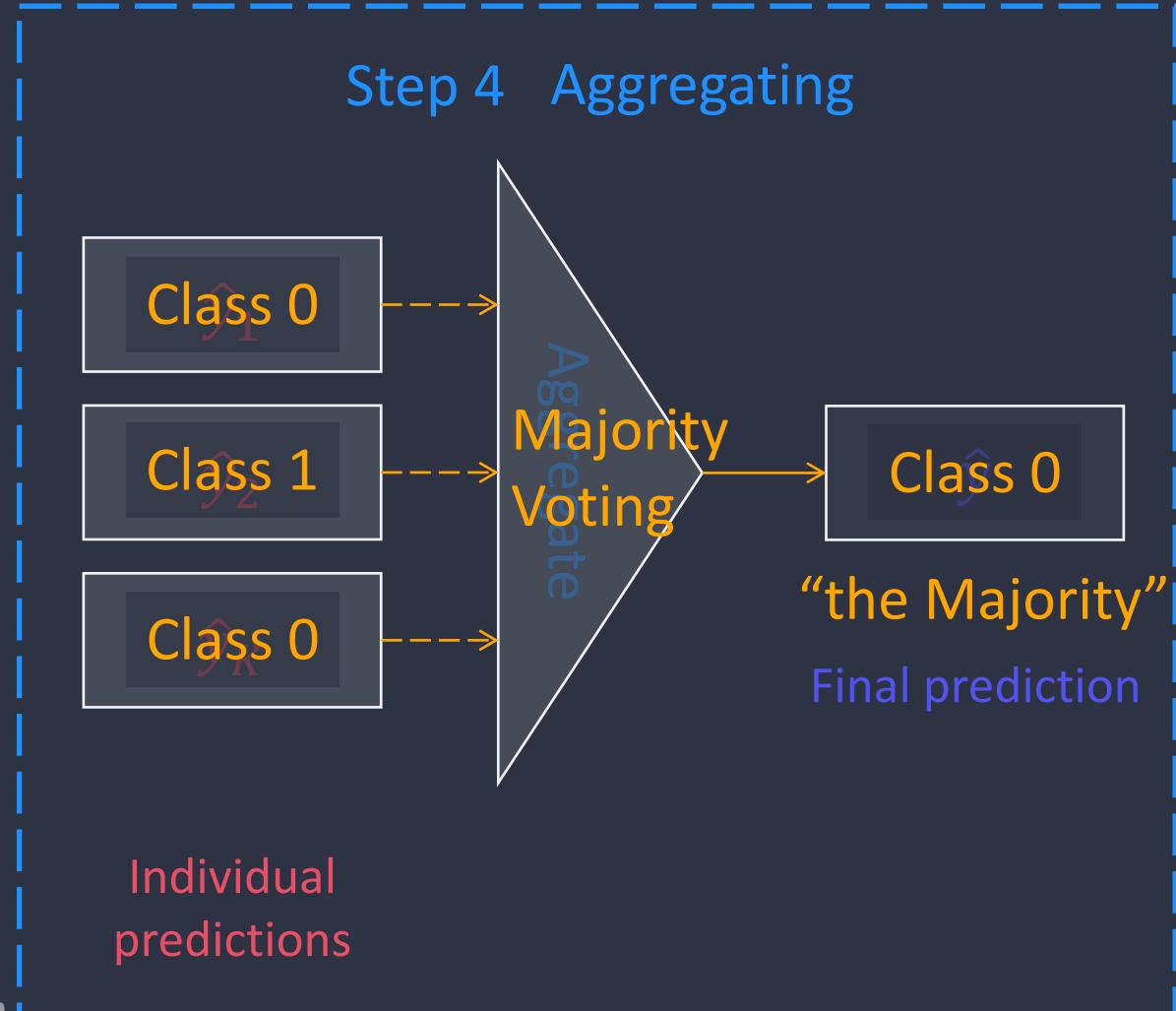


# Bagging



Majority Voting / Averaging  
Classification      Regression

# Bagging



## Majority Voting

EXAMPLE.

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), \dots, h_K(x)\}$$

$$h_1(x) \longrightarrow \text{Class 0}$$

$$h_2(x) \longrightarrow \text{Class 1}$$

$$h_3(x) \longrightarrow \text{Class 0}$$

$$\hat{y} = \text{mode}\{0,1,0\} = 0 \quad (\text{Class 0})$$

# Bagging

## Aggregating / Majority Voting

$$\hat{y} = \text{mode}\{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_K(\mathbf{x})\}$$

Equal weight

$$\hat{y} = \arg \max_i \sum_{j=1}^K \omega_j \chi_A(h_j(\mathbf{x}) = i)$$

Weighted

### EXAMPLE.

$$h_1(\mathbf{x}) \longrightarrow i_0 \text{ (Class 0)}$$

weights {0.1, 0.6, 0.3}

$$h_2(\mathbf{x}) \longrightarrow i_1 \text{ (Class 1)}$$

$$\hat{y} = \arg \max_i \{0.1 \times i_0, 0.6 \times i_1, 0.3 \times i_0\}$$

$$h_3(\mathbf{x}) \longrightarrow i_0 \text{ (Class 0)}$$

$$= i_1 \text{ (Class 1)}$$

# Bagging

## Aggregating / Majority Voting

- Hard Voting: final prediction as the (exact) class that has been most frequently predicted by the base learners.

$$\hat{y} = \text{mode}\{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_K(\mathbf{x})\}$$

Equal weight

$$\hat{y} = \arg \max_i \sum_{j=1}^K \omega_j \chi_A(h_j(\mathbf{x}) = i)$$

Weighted

- Soft Voting: final prediction based on the probabilities of all the predictions made by individual base learners.

$$\hat{y} = \arg \max_i \sum_{j=1}^K \omega_j p_{ij}$$

# Bagging

## Aggregating / Majority Voting

- Soft Voting: final prediction based on the probabilities of all the predictions made by individual base learners.

$$\hat{y} = \arg \max_i \sum_{j=1}^K \omega_j p_{ij}$$

EXAMPLE. Assuming a binary classification task with class labels  $i \in \{0,1\}$ , if

$$h_1(\mathbf{x}) \longrightarrow \begin{matrix} p(i_0|\mathbf{x}) & p(i_1|\mathbf{x}) \\ [0.2, 0.8] \end{matrix}$$

$$h_2(\mathbf{x}) \longrightarrow [0.4, 0.6]$$

$$h_3(\mathbf{x}) \longrightarrow [0.7, 0.3]$$

$$p(i_0|\mathbf{x}) = \frac{1}{3} \times (0.2 + 0.4 + 0.7) = 0.43$$

$$p(i_1|\mathbf{x}) = \frac{1}{3} \times (0.8 + 0.6 + 0.3) = 0.57$$

$$\hat{y} = \arg \max_i [p(i_0|\mathbf{x}), p(i_1|\mathbf{x})] = 1 \text{ (Class 1)}$$

Equal weight

$$p(i_0|\mathbf{x}) = 0.1 \times 0.2 + 0.2 \times 0.4 + 0.7 \times 0.7 = 0.59$$

$$p(i_1|\mathbf{x}) = 0.1 \times 0.8 + 0.2 \times 0.6 + 0.7 \times 0.3 = 0.41$$

$$\hat{y} = \arg \max_i [p(i_0|\mathbf{x}), p(i_1|\mathbf{x})] = 0 \text{ (Class 0)}$$

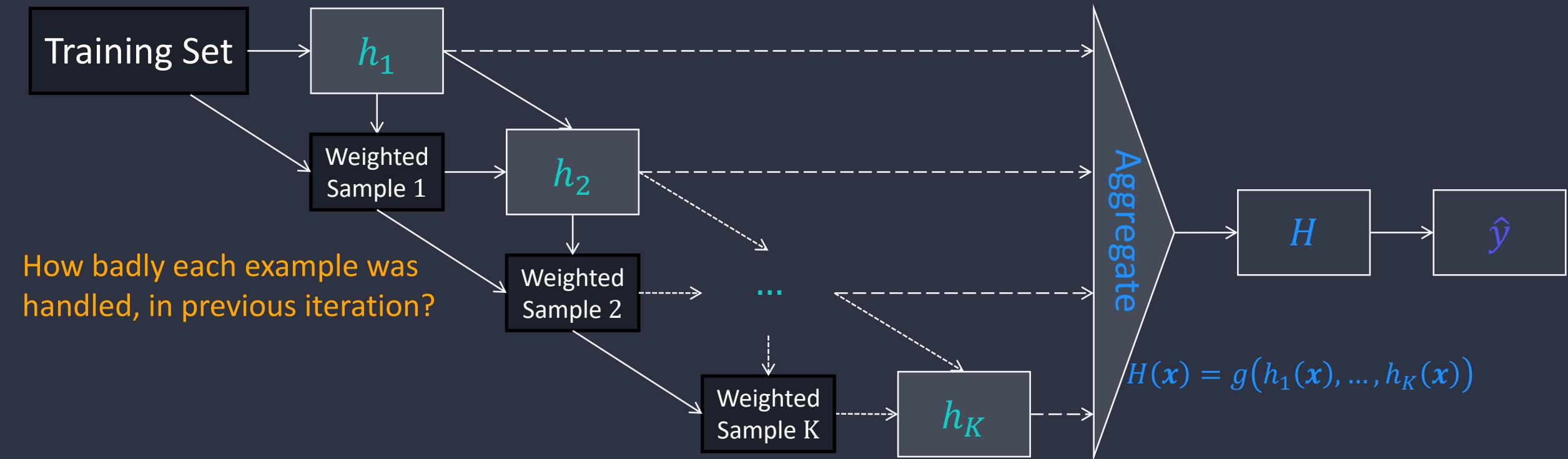
Weighted  
as  $\{0.1, 0.2, 0.7\}$

# Bagging

- Short for Bootstrap Aggregating
- An ensemble learning method seeking a diverse group of ensemble members by varying the training data.
- Designed to improve the stability and accuracy of machine learning algorithms for classification and regression.
- Designed to reduce variance and avoid overfitting.

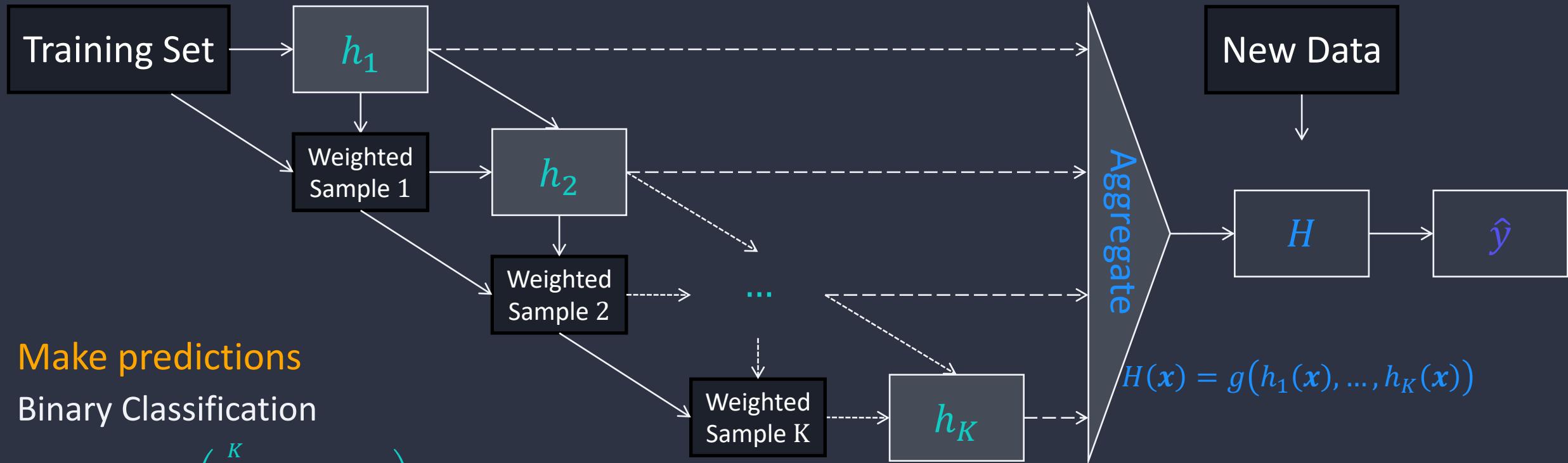
# 3. Boosting

# Boosting



Each base learner in the sequence is trained giving more importance to examples in the training set that were badly handled by the previous base model in the sequence.

# Boosting



Make predictions

Binary Classification

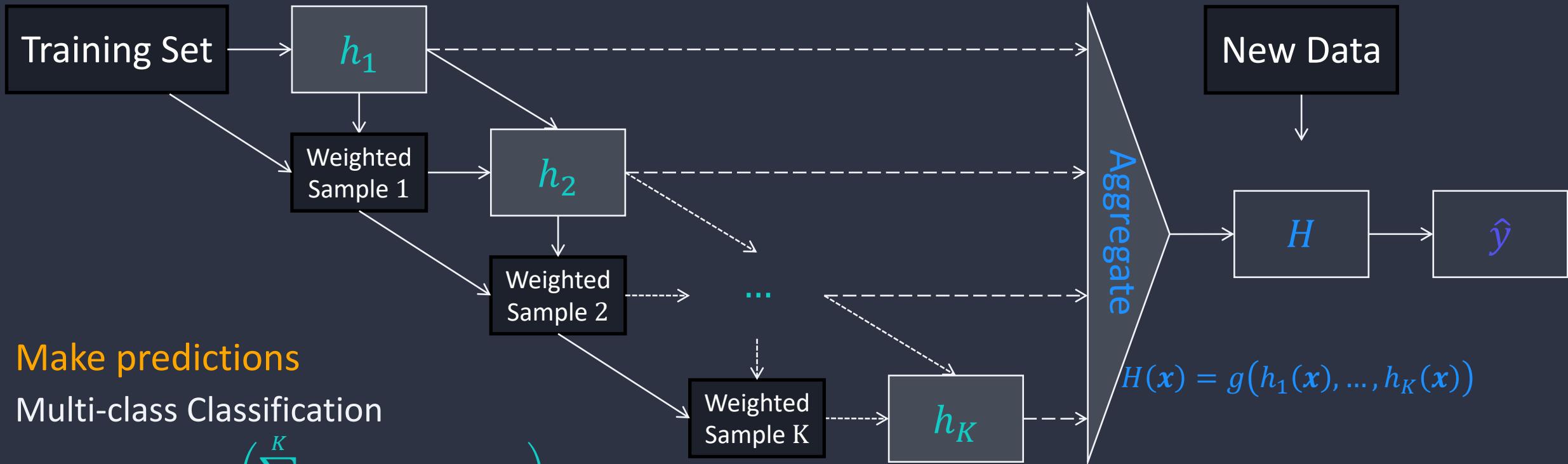
$$H(\mathbf{x}) = \text{sign} \left( \sum_{k=1}^K \alpha_k \times h_k(\mathbf{x}) \right) \text{ for } h_k(\mathbf{x}) \in \{-1, 1\}$$

e.g. if  $k = 4$   $h_1(\mathbf{x}) = 1$   $h_2(\mathbf{x}) = -1$   $h_3(\mathbf{x}) = 1$

then  $h_4(\mathbf{x}) = \text{sign}(0.2 \times 1 + 0.7 \times (-1) + 0.1 \times 1) = \text{sign}(-0.4) = -1$

$$H(\mathbf{x}) = g(h_1(\mathbf{x}), \dots, h_K(\mathbf{x}))$$

# Boosting



**Make predictions**

**Multi-class Classification**

$$H(\mathbf{x}) = \arg \max_c \left( \sum_{k=1}^K \alpha_k \times \mathbb{1}[h_k(\mathbf{x}) = c] \right) \text{ for } h_k(\mathbf{x}) = c, c \in \{1, \dots, C\}$$

e.g. if  $K = 4, C = 3 \rightarrow c \in \{1, 2, 3\}$

then when  $c = 1$

$$h_1(\mathbf{x}) \rightarrow 1 = 1? \quad \checkmark \rightarrow 1$$

$$h_2(\mathbf{x}) \rightarrow 2 = 1? \quad X \rightarrow 0$$

$$h_3(\mathbf{x}) \rightarrow 1 = 1? \quad \checkmark \rightarrow 1$$

when  $c = 2$

$$h_1(\mathbf{x}) \rightarrow 1 = 2? \quad X \rightarrow 0$$

$$h_2(\mathbf{x}) \rightarrow 2 = 2? \quad \checkmark \rightarrow 1$$

$$h_3(\mathbf{x}) \rightarrow 1 = 2? \quad X \rightarrow 0$$

when  $c = 3$

$$h_1(\mathbf{x}) \rightarrow 1 = 3? \quad X \rightarrow 0$$

$$h_2(\mathbf{x}) \rightarrow 2 = 3? \quad X \rightarrow 0$$

$$h_3(\mathbf{x}) \rightarrow 1 = 3? \quad X \rightarrow 0$$

**New Data**

$$H$$

$$\hat{y}$$

**Aggregate**

$$H(\mathbf{x}) = g(h_1(\mathbf{x}), \dots, h_K(\mathbf{x}))$$

Assuming equal weight,  
when  $c = 1$ , we get max.  
so,  $\hat{y} \rightarrow \text{class 1}$

# Boosting

## General procedure

*init a weight vector with equal weights*

{

*train base (weak) learner on weighted sample (examples)*

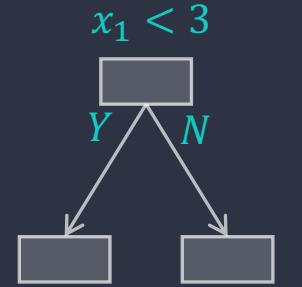
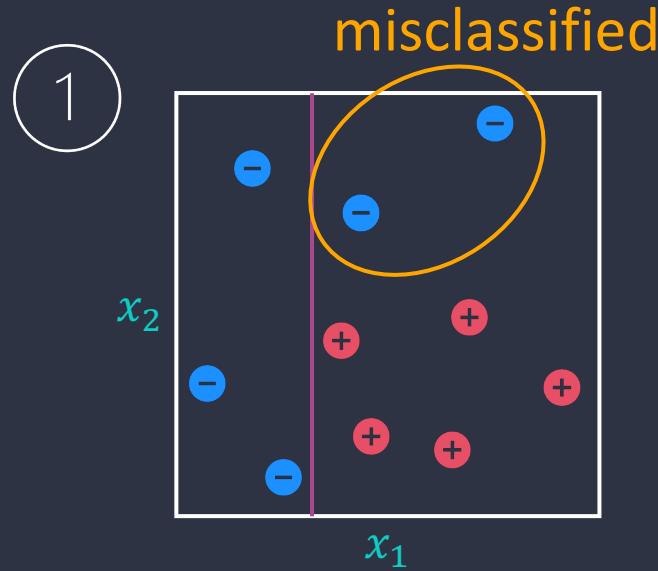
*increase weight for misclassified (large error/cost) example*

}

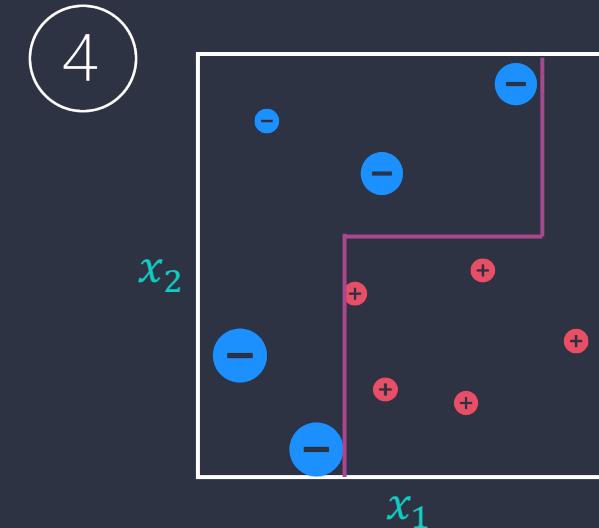
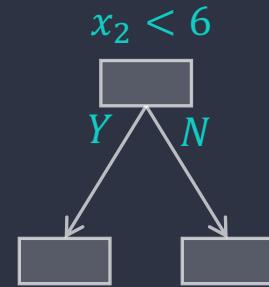
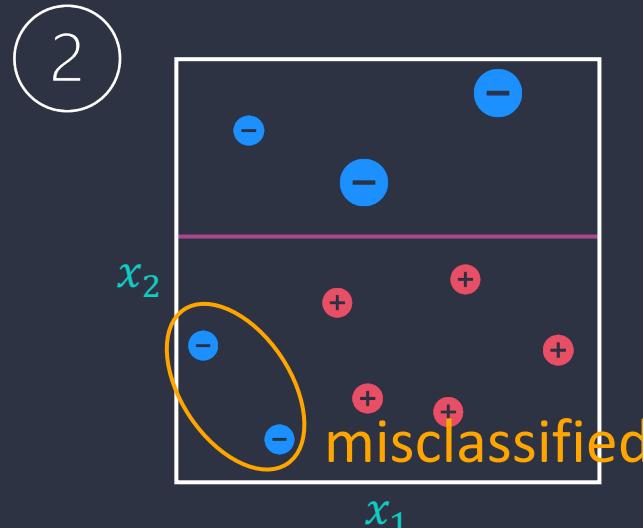
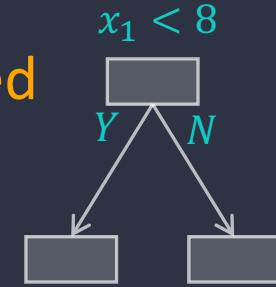
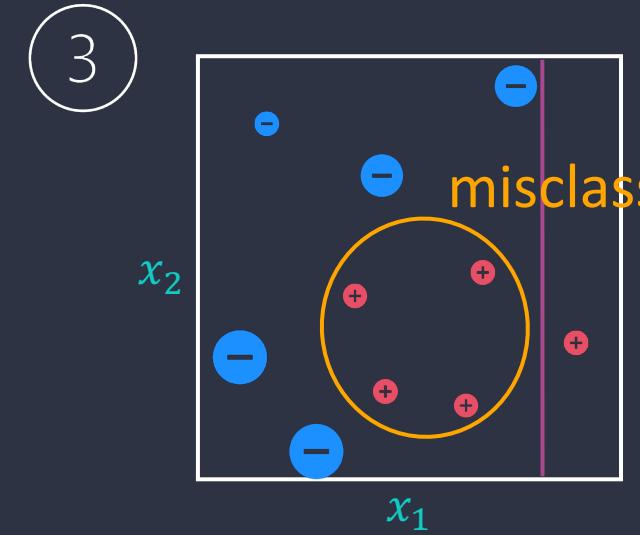
*weighted majority voting on trained base learners*

- How are these base learners trained?
- How are these base learners aggregated?

# Boosting - AdaBoost (Adaptive Boosting)



A Decision Stump



Combined classifier

# Boosting - AdaBoost (Adaptive Boosting)

*init*  $K$ : the number of AdaBoost rounds

*init*  $\mathcal{D}$ : the training set, where  $\mathcal{D} = \{(X^{(i)}, y^{(i)})\}_{i=1}^m, X^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{-1, 1\}$

*init*  $\mathbf{w}_1$ : weights for each example in the 1st round, where  $\mathbf{w}_1 = \{\omega_1^{(i)} = 1/m\}_{i=1}^m$  and  $\mathbf{w}_1 \in \mathbb{R}^m$

for  $k = 1:K$

$h_k := \text{train}(\mathcal{D}, \mathbf{w}_k);$

$$\epsilon_k := \frac{\sum_{i=1}^m \omega_k^{(i)} \mathbb{1}[h_k(X^{(i)}) \neq y^{(i)}]}{\sum_{i=1}^m \omega_k^{(i)}}$$

if  $\epsilon_k > 0.5$  then stop

$$\alpha_k := 0.5 \cdot \ln \frac{(1 - \epsilon_k)}{\epsilon_k}$$

for  $i = 1:m$

$$\omega_{k+1}^{(i)} := \omega_k^{(i)} e^{\alpha_k \mathbb{1}[h_k(X^{(i)}) \neq y^{(i)}]}$$

Predict:  $H(\mathbf{x}) = \arg \max_c \left( \sum_{k=1}^K \alpha_k \times \mathbb{1}[h_k(\mathbf{x}) = c] \right)$

# Boosting - AdaBoost (Adaptive Boosting)

## Early Stopping

More classifiers added in -> the ensemble model becomes more complex -> overfitting



# Boosting - AdaBoost (Adaptive Boosting)

## Early Stopping

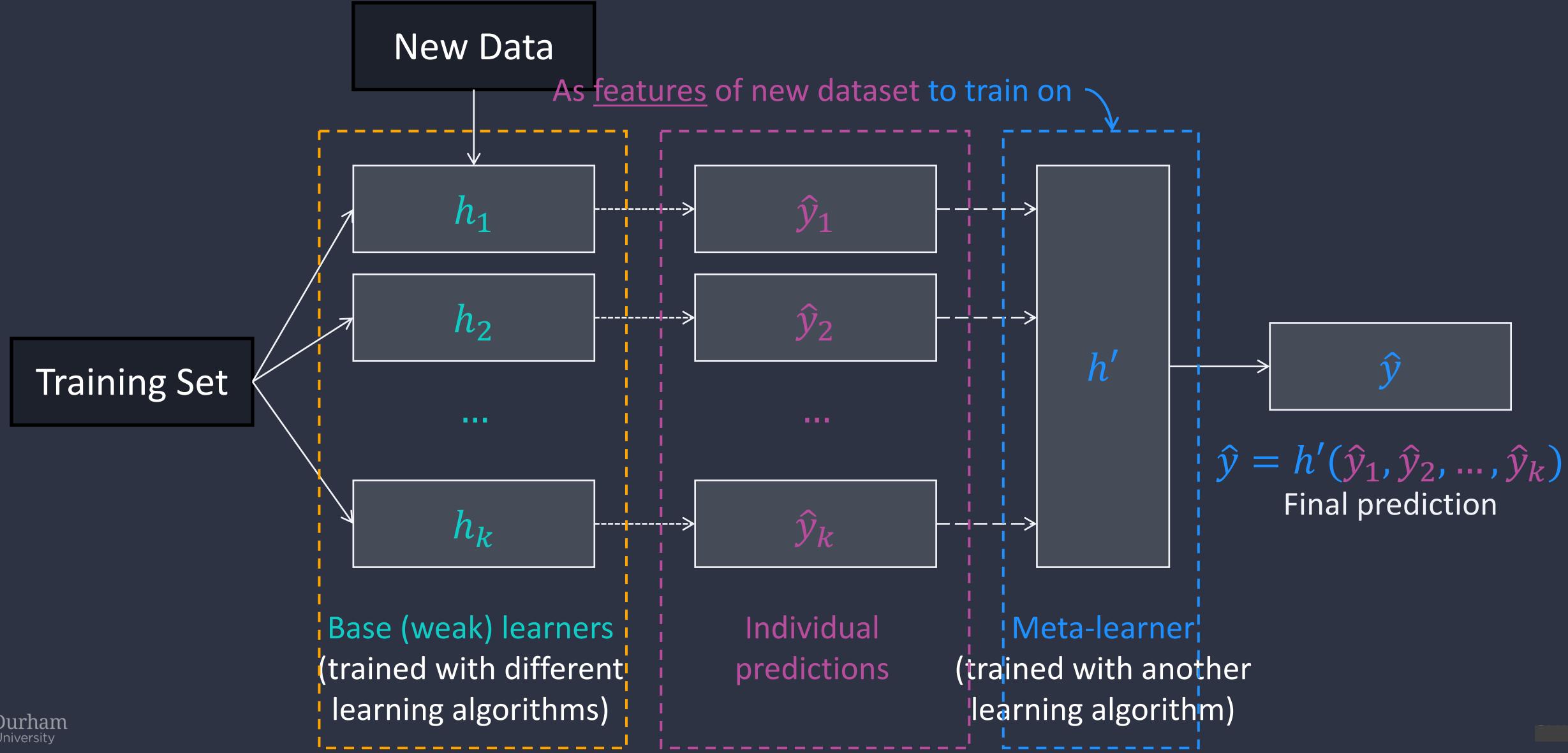


# 4. Stacking

# Stacking / Stacked Generalisation

	Bagging & Boosting	Stacking
Base/weak learners	homogeneous (same learning algorithm)	heterogeneous (different learning algorithms)
Final (combined) prediction	deterministic algorithms (same input -> same output)	meta-learning (learning about learning)

# Stacking / Stacked Generalisation



## 5. Random Forest

# What is a Random Forest ?



- Consists of a number of individual decision trees that operate as an ensemble.
- Is an ensemble learning method for classification and regression tasks.
- Outputs the class that is the mode of the classes (classification); or average prediction (regression) of individual decision trees.

# What is a Random Forest ?



- Decision Trees are easy to build, use and interpret, but...
- Decision Trees are not flexible when classifying new instances.
- A Random Forest combines the simplicity of Decision Trees with flexibility.
- “Wisdom of the Crowd”

# How to build a Random Forest?

# How to build a Random Forest?

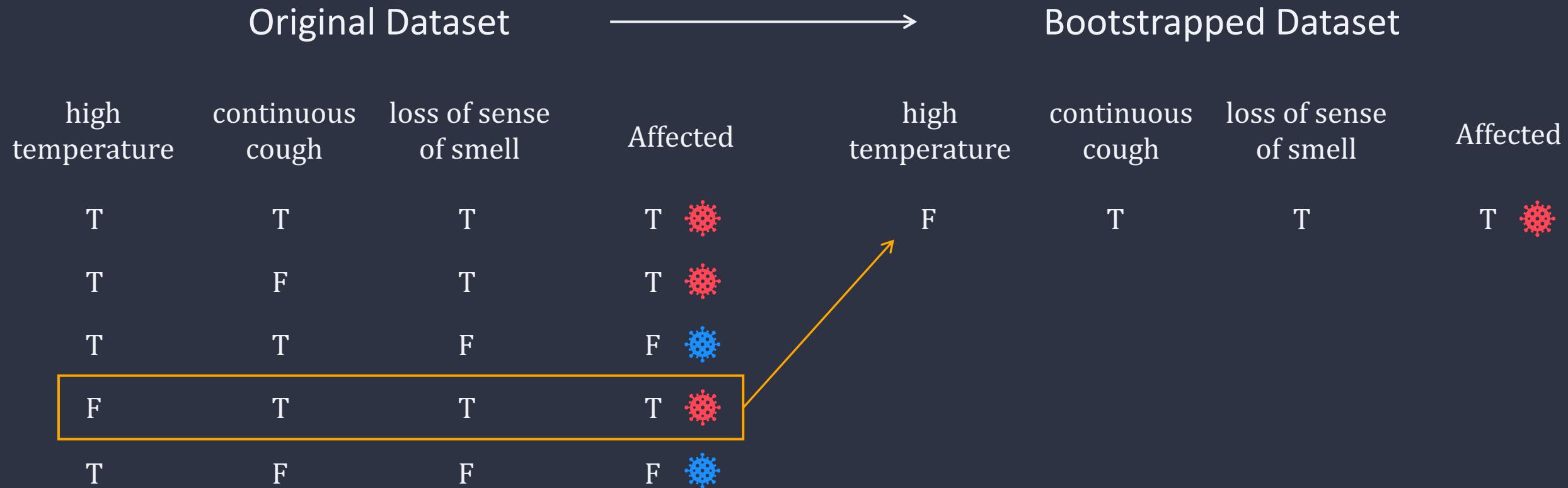
STEP 1. create a Bootstrapped Dataset from the Original Dataset.

Original Dataset					Bootstrapped Dataset			
high temperature	continuous cough	loss of sense of smell	Affected	high temperature	continuous cough	loss of sense of smell	Affected	
T	T	T	T 					
T	F	T	T 					
T	T	F	F 					
F	T	T	T 					
T	F	F	F 					

 • Same size as the original dataset.  
• Randomly selected examples from the original dataset.  
• Examples can be selected more than once.

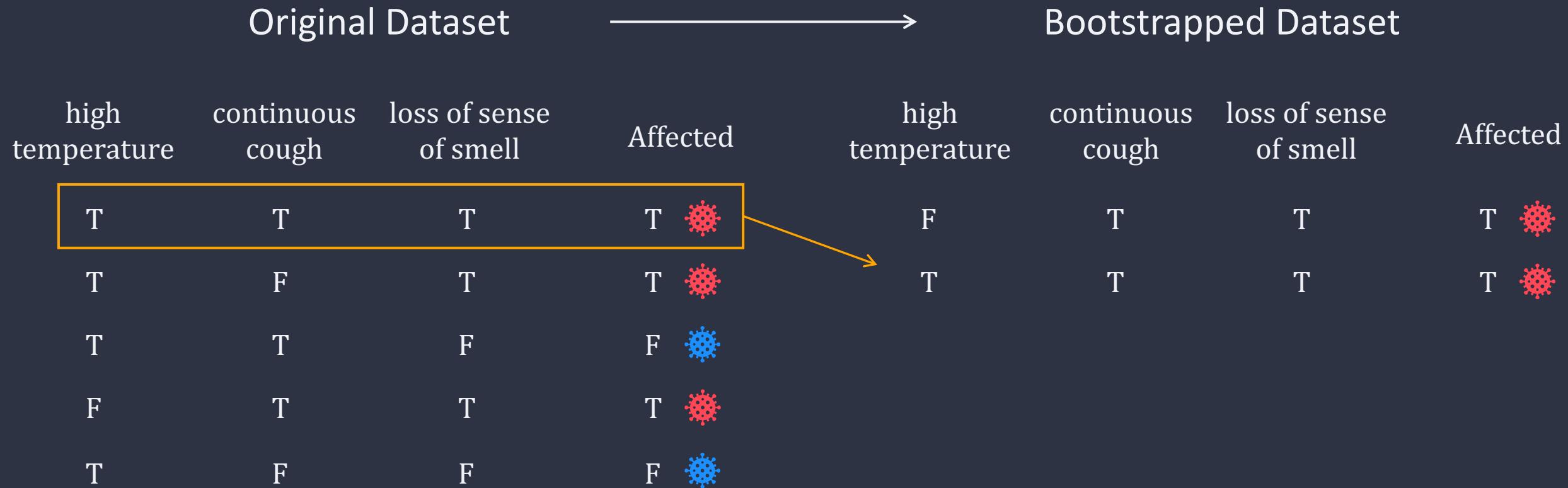
# How to build a Random Forest?

STEP 1. create a Bootstrapped Dataset from the Original Dataset.



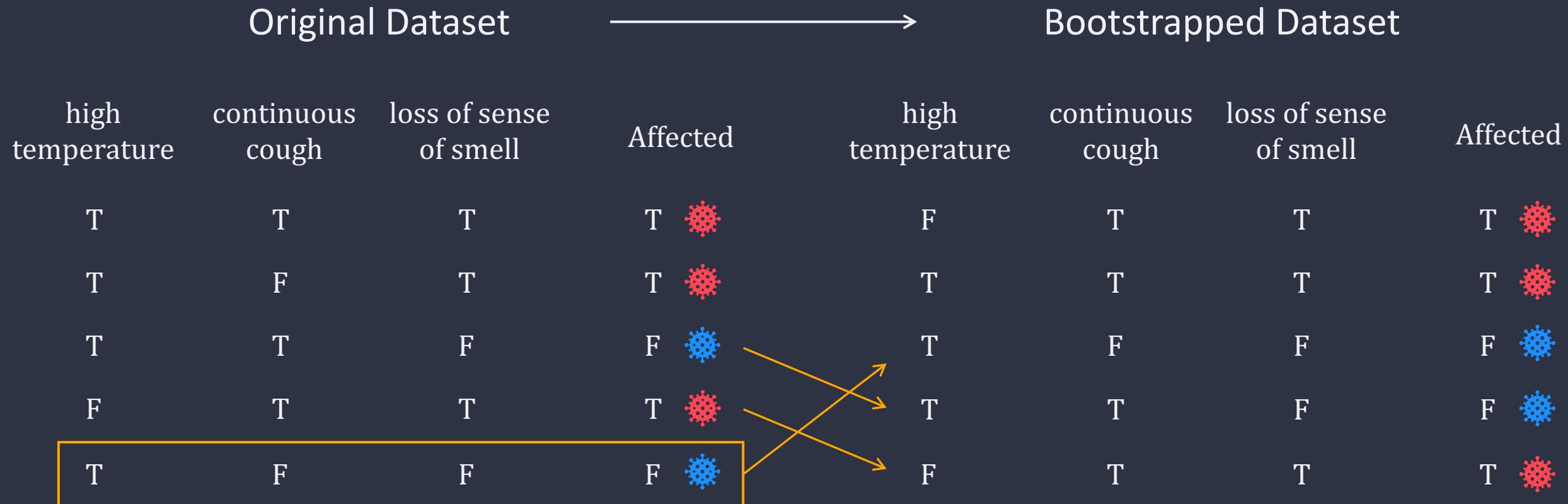
# How to build a Random Forest?

STEP 1. create a Bootstrapped Dataset from the Original Dataset.



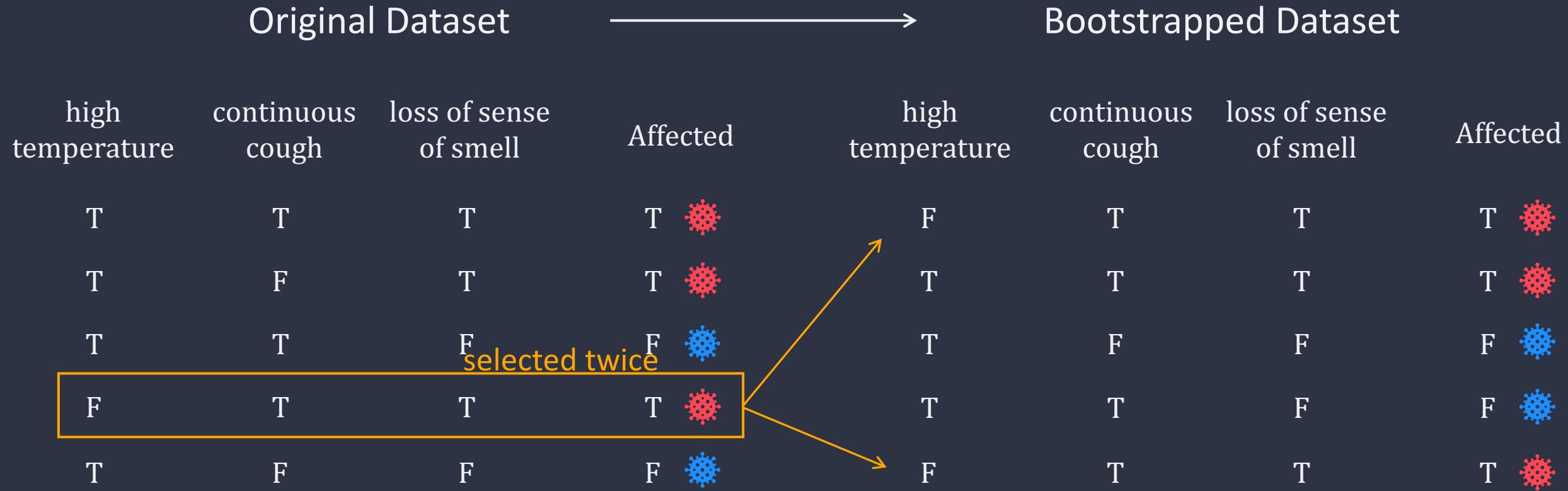
# How to build a Random Forest?

STEP 1. create a Bootstrapped Dataset from the Original Dataset.



# How to build a Random Forest?

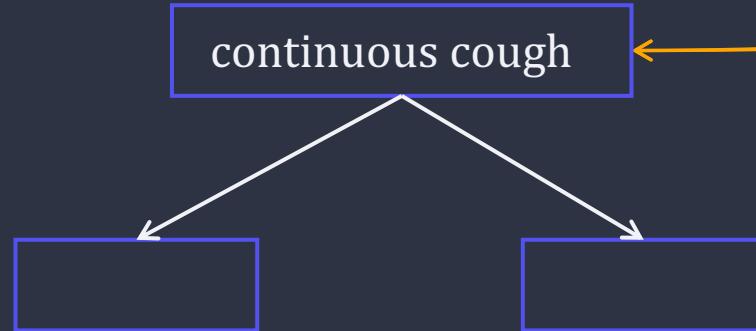
STEP 1. create a Bootstrapped Dataset from the Original Dataset.



# How to build a Random Forest?

STEP 2. create a DT from Bootstrapped Dataset, using random subset of features.

continuous cough did the best job splitting instances.

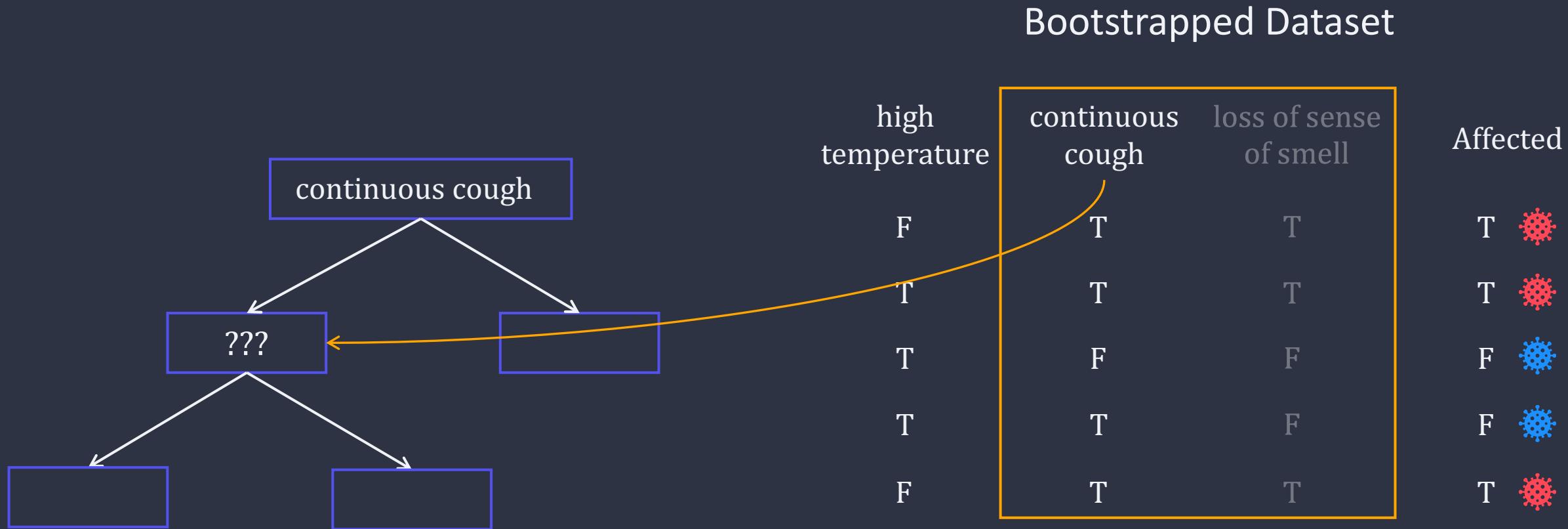


Bootstrapped Dataset

	high temperature	continuous cough	loss of sense of smell	Affected
	F	T	T	T
	T	T	T	T
	T	F	F	F
	T	T	F	F
	F	T	T	T

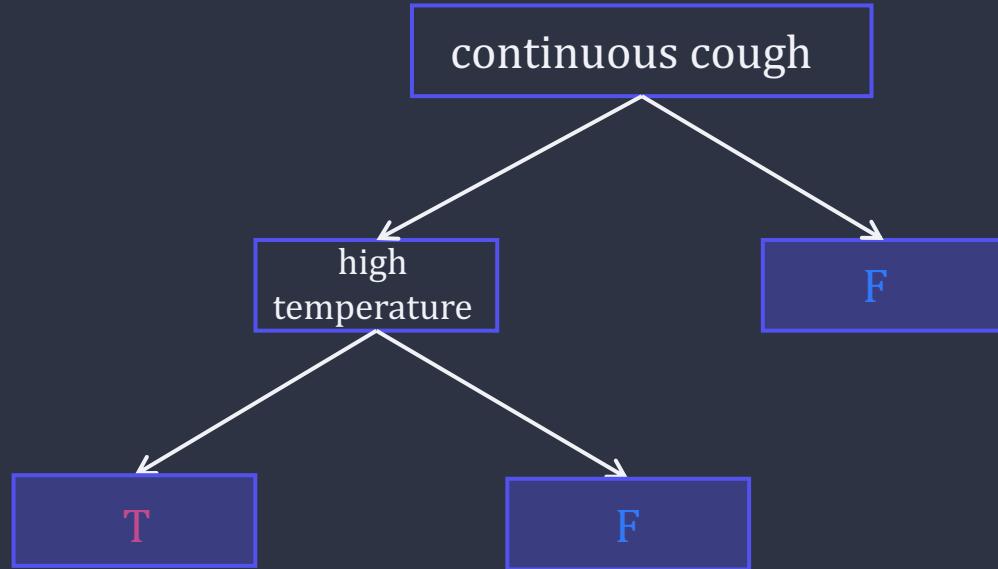
# How to build a Random Forest?

STEP 2. create a DT from Bootstrapped Dataset, using random subset of features.



# How to build a Random Forest?

STEP 2. create a DT from Bootstrapped Dataset, using random subset of features.

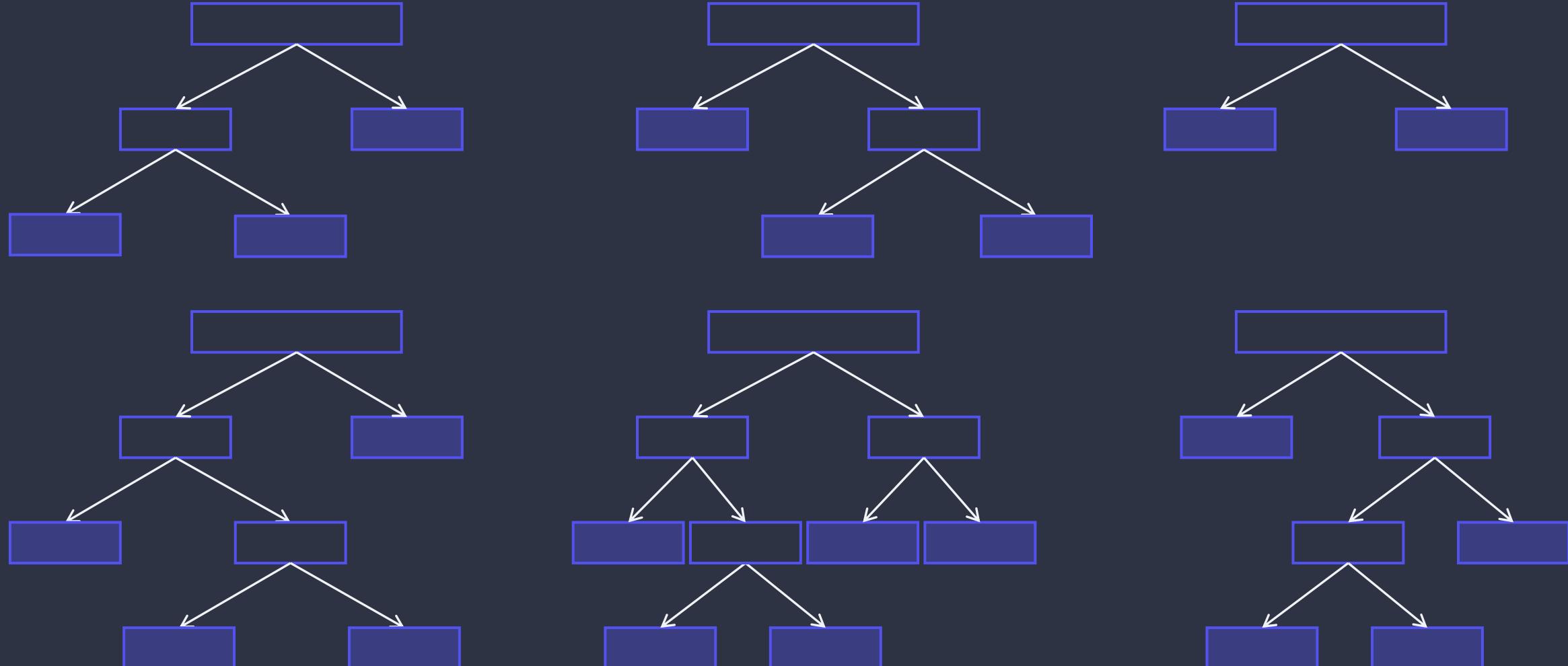


Bootstrapped Dataset

	high temperature	continuous cough	loss of sense of smell	Affected
	F	T	T	T
	T	T	T	T
	T	F	F	F
	T	T	F	F
	F	T	T	T

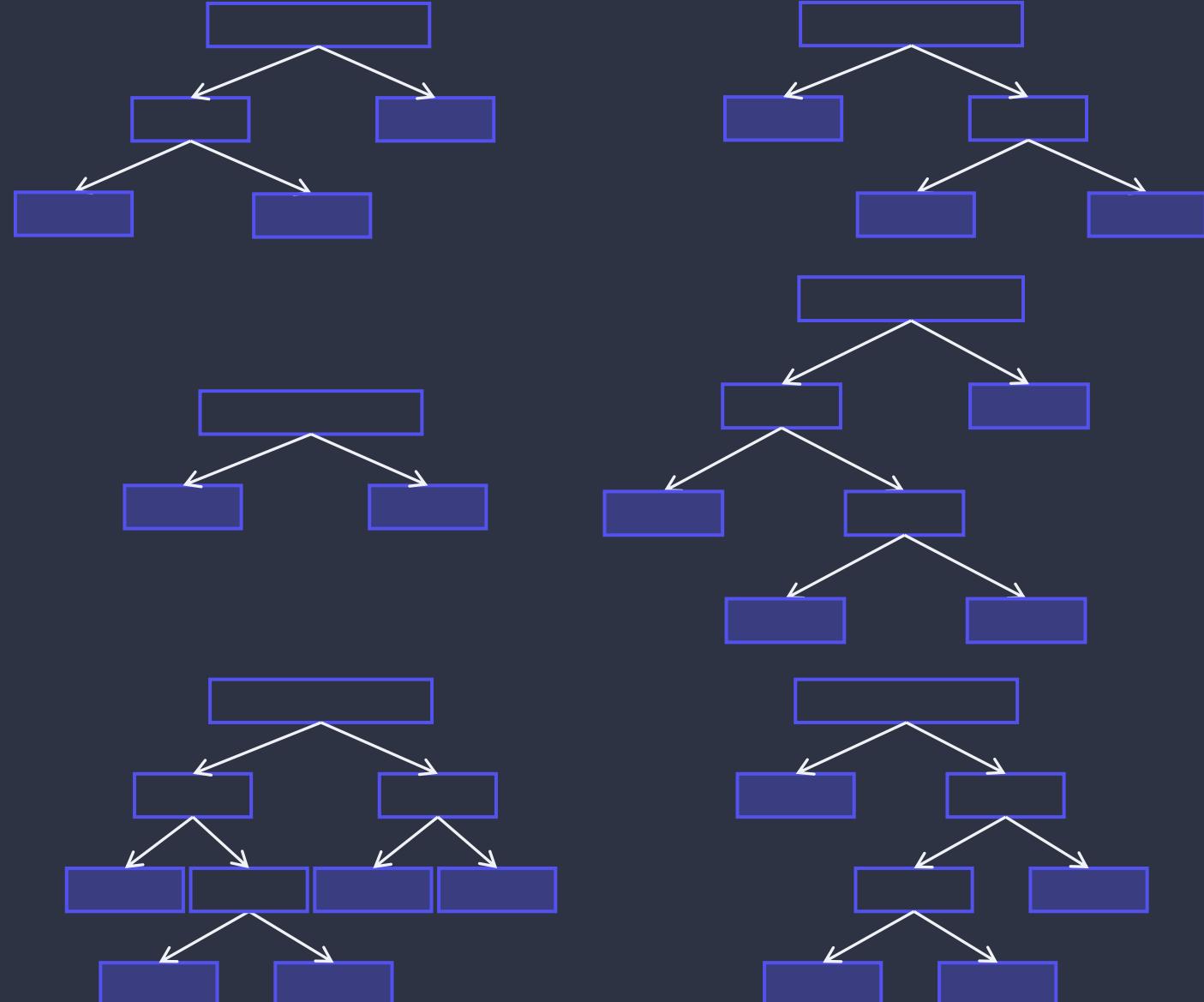
# How to build a Random Forest?

STEP 3. go back to STEP 1 and repeat.



# How to build a Random Forest?

- Using bootstrapped examples and considering only a subset of features can result in a wide variety of Decision Trees.
- The variety makes Random Forests more effective than individual Decision Trees.



# How does a Random Forest make predictions?

# How does a Random Forest make predictions?



a new patient

high continuous loss of sense  
temperature cough of smell

T

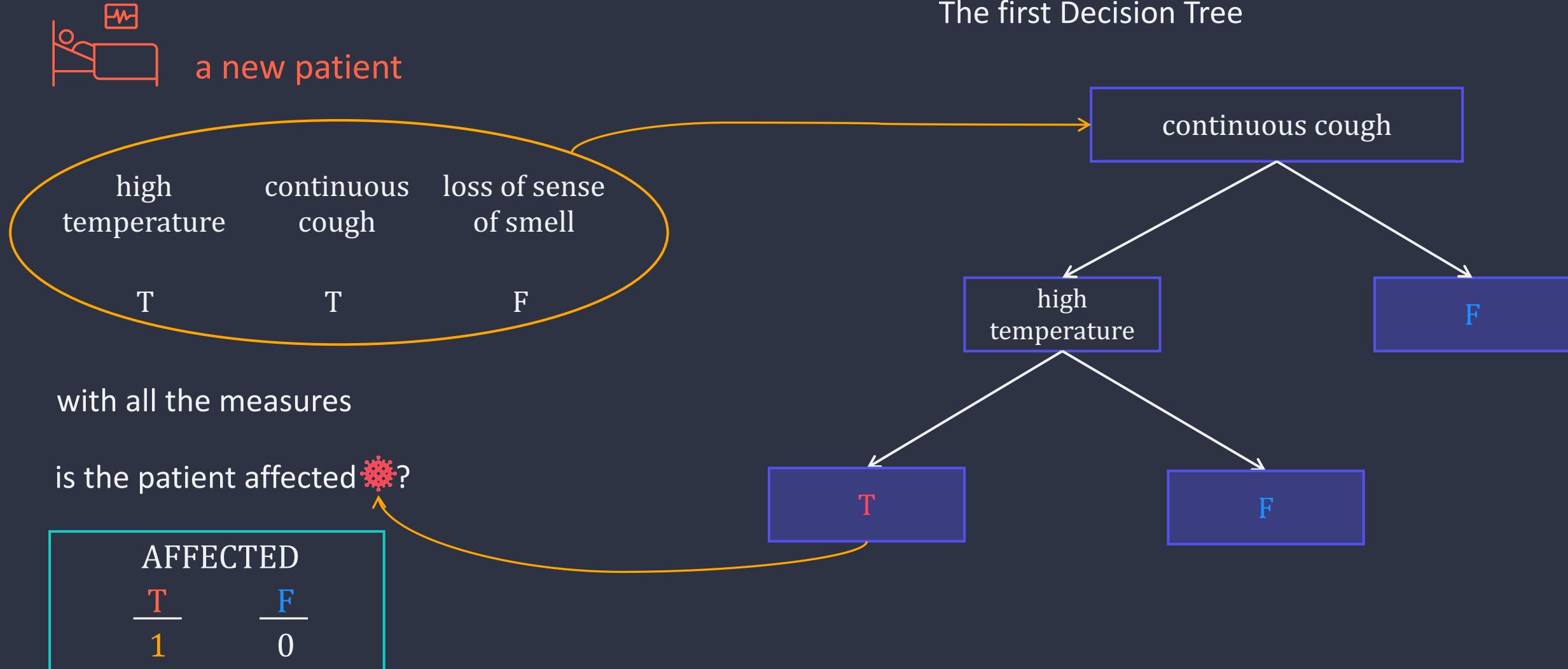
T

F

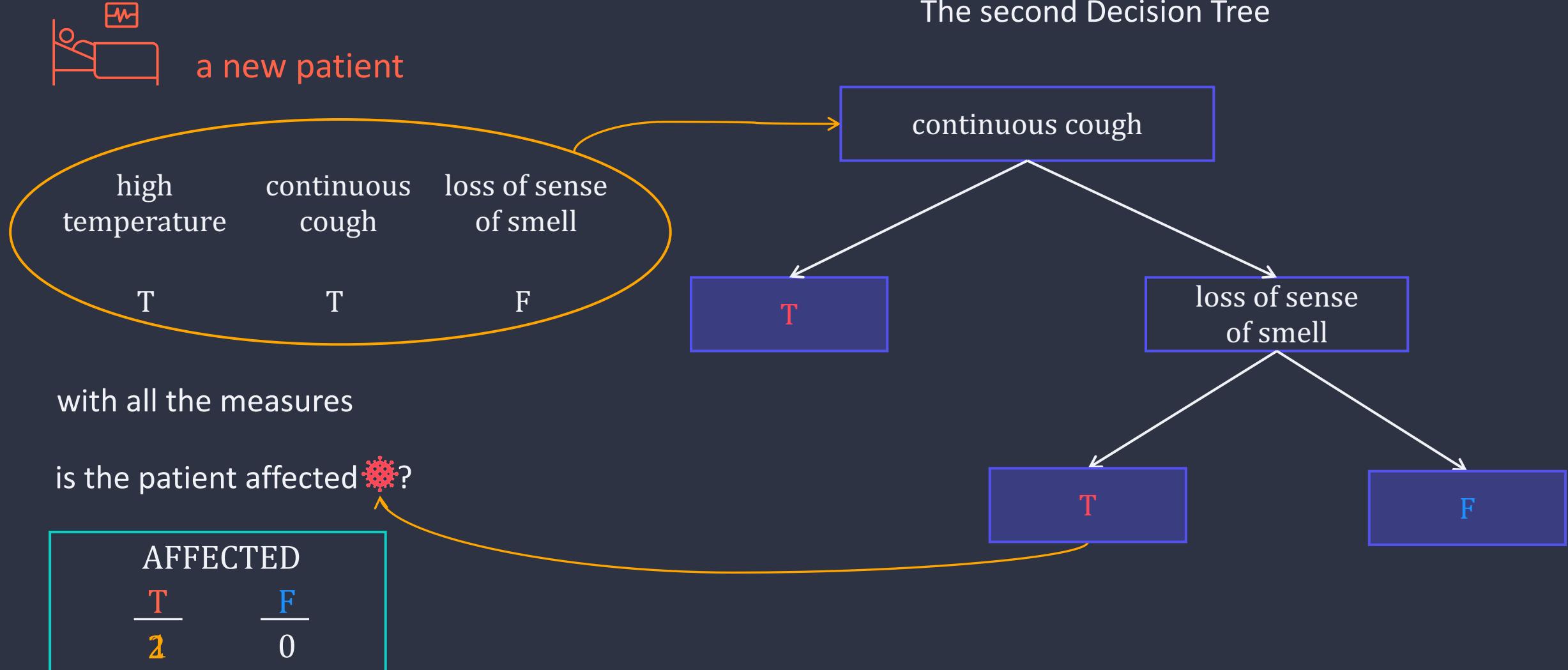
with all the measures

is the patient affected? 

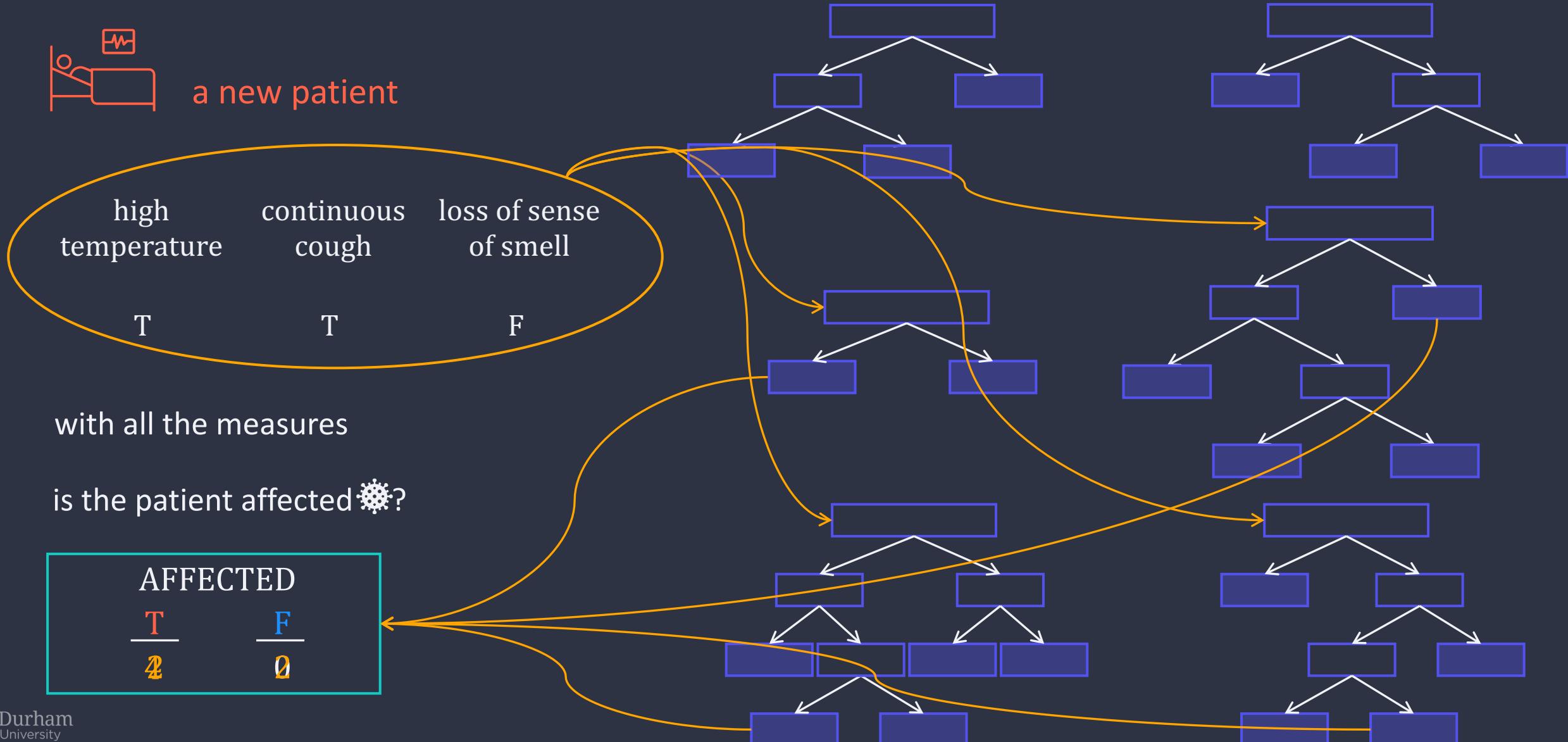
# How does a Random Forest make predictions?



# How does a Random Forest make predictions?



# How does a Random Forest make predictions?



# How does a Random Forest make predictions?



a new patient

high temperature continuous cough loss of sense of smell

T

T

F

with all the measures

is the patient affected? 🌫️

conclude: the patient is affected. ←

AFFECTED

$$\frac{T}{4} \quad \frac{F}{2}$$

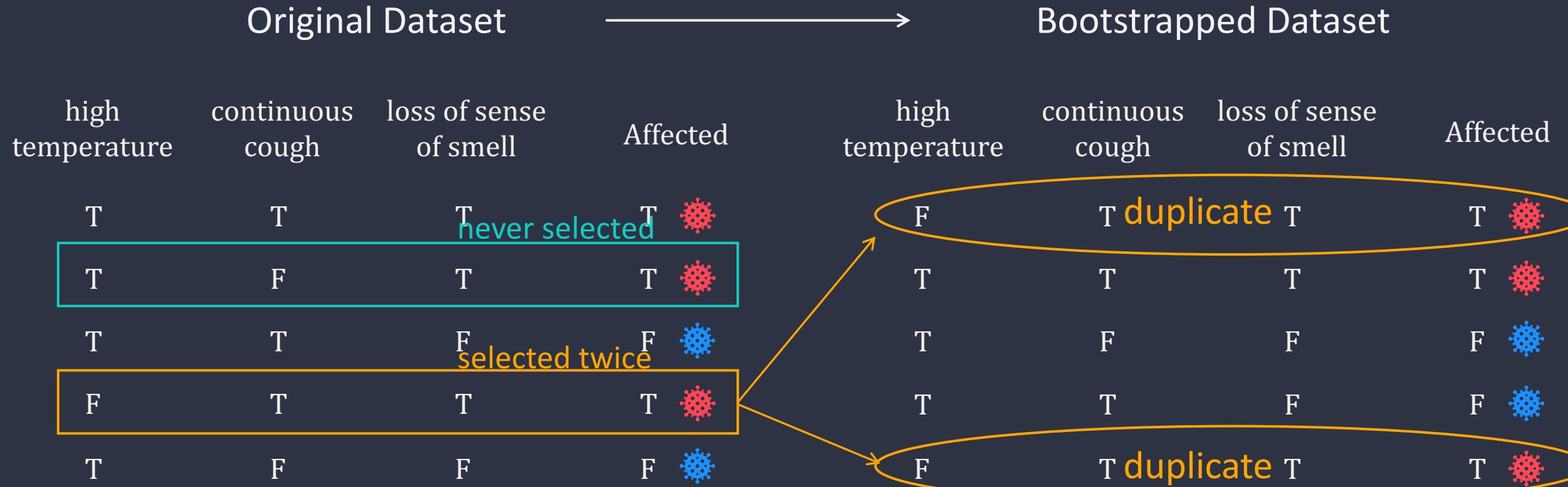
received more votes

Bagging!

# Is this Random Forest any good?

# Is this Random Forest any good?

Creating a Bootstrapped Dataset from the Original Dataset.



# Is this Random Forest any good?

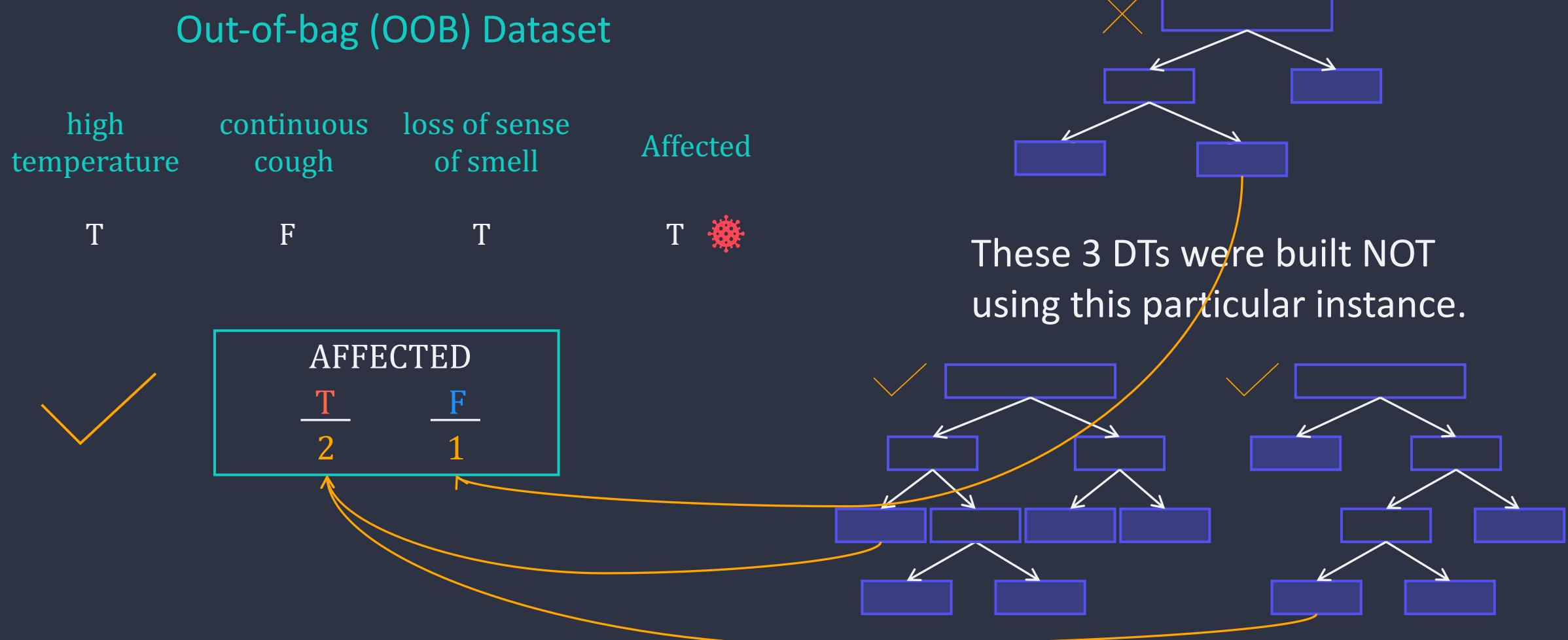
Creating a Bootstrapped Dataset from the Original Dataset.

Original Dataset					Out-of-bag (OOB) Dataset				
high temperature	continuous cough	loss of sense of smell	Affected		high temperature	continuous cough	loss of sense of smell	Affected	
T	T	T never selected	T	*	T	F	T	T	*
T	F	T	T	*					
T	T	F	F	*					
F	T	T	T	*					
T	F	F	F	*					

There would be more examples if the original dataset was larger

# Is this Random Forest any good?

Test each Decision Tree that was built NOT using an instance in the OOB dataset.



# Is this Random Forest any good?

## STEP 1

Test each Decision Tree that was built NOT using an example in the OOB dataset.

## STEP 2

Measure accuracy of Random Forest is by the proportion of Out-Of-Bag examples that were correctly classified by the Random Forest.

## Out-Of-Bag Error

The proportion of Out-Of-Bag examples that were incorrectly classified.

Now, we know how to

Create a Random Forest

Use a Random Forest

Estimate the Accuracy of a Random Forest

# Coding with sci-kit learn

## Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  
  
clf = RandomForestClassifier(n_estimators=20)  
clf.fit(X_train, y_train)  
  
y_pred = regressor.predict(X_test)  
  
print(confusion_matrix(y_test,y_pred))  
print(classification_report(y_test,y_pred))  
print(accuracy_score(y_test, y_pred))
```

# Coding with sci-kit learn

## Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics

regressor = RandomForestRegressor(n_estimators=20)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print(metrics.mean_absolute_error(y_test, y_pred))
print(metrics.mean_squared_error(y_test, y_pred))
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

# Summary

# Summary

1. Intuition
2. Bagging
3. Boosting
4. Stacking
5. Random Forest

Next Lecture

Clustering and K-Means

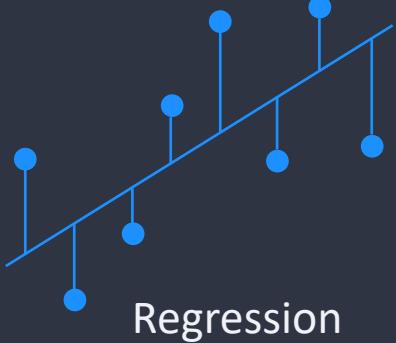
COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Clustering and K-Means

Dr Yang Long

# Previously

## Machine Learning Algorithms



Regression



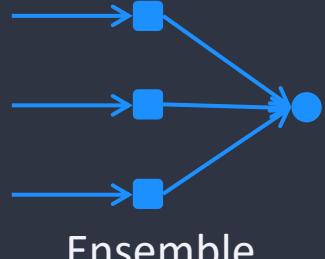
Regularisation



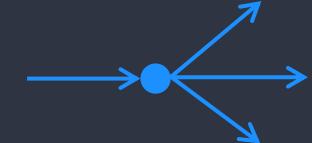
Clustering



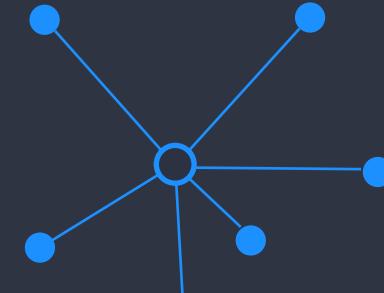
Bayesian



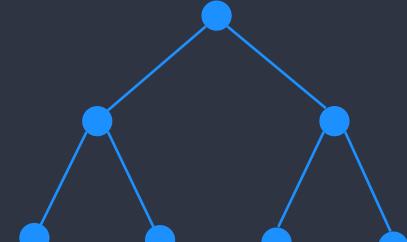
Ensemble



Neural Network



Instance-based



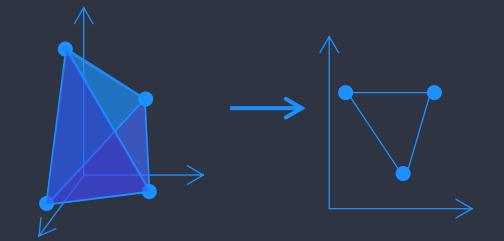
Tree-based



Associated Rule Learning



Deep Learning



Dimensionality Reduction

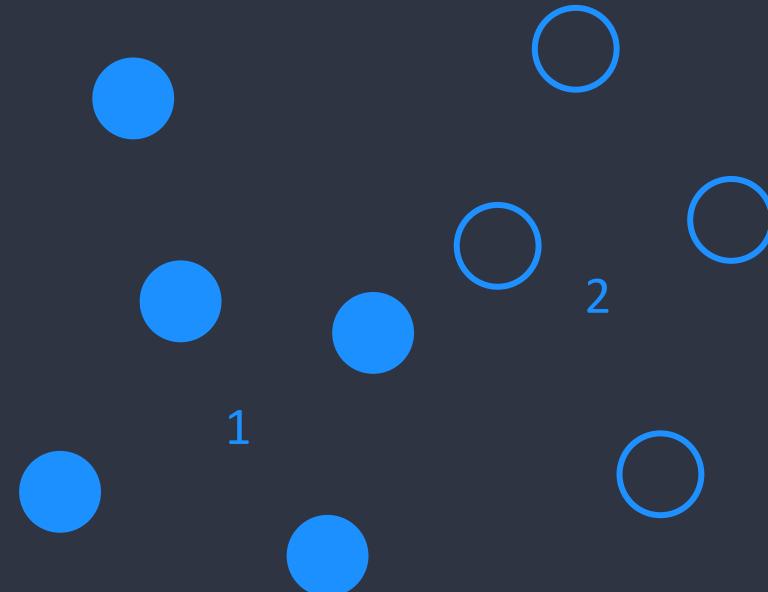


Reinforcement Learning

# Lecture Overview

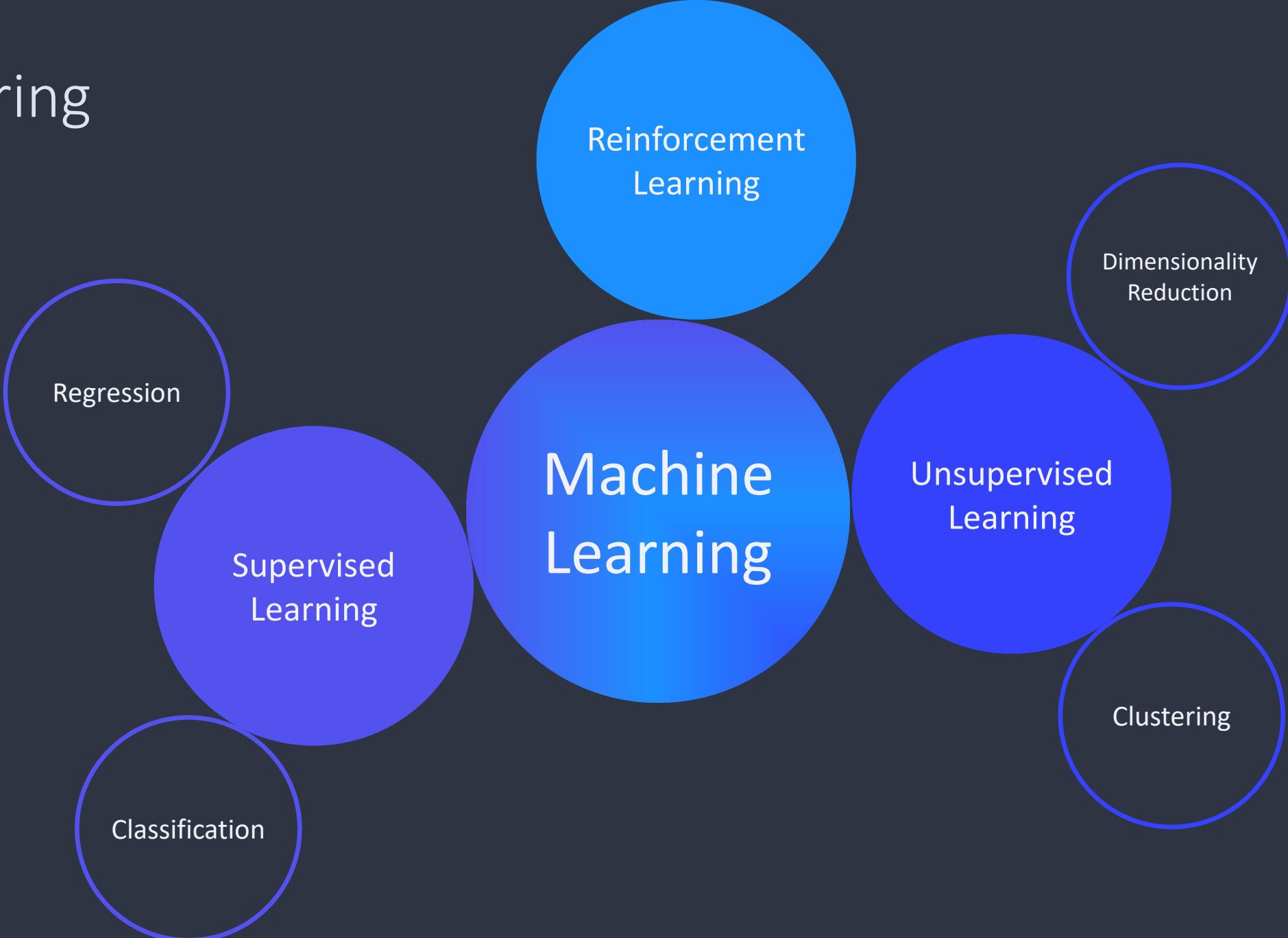
1. Clustering

2. K-Means

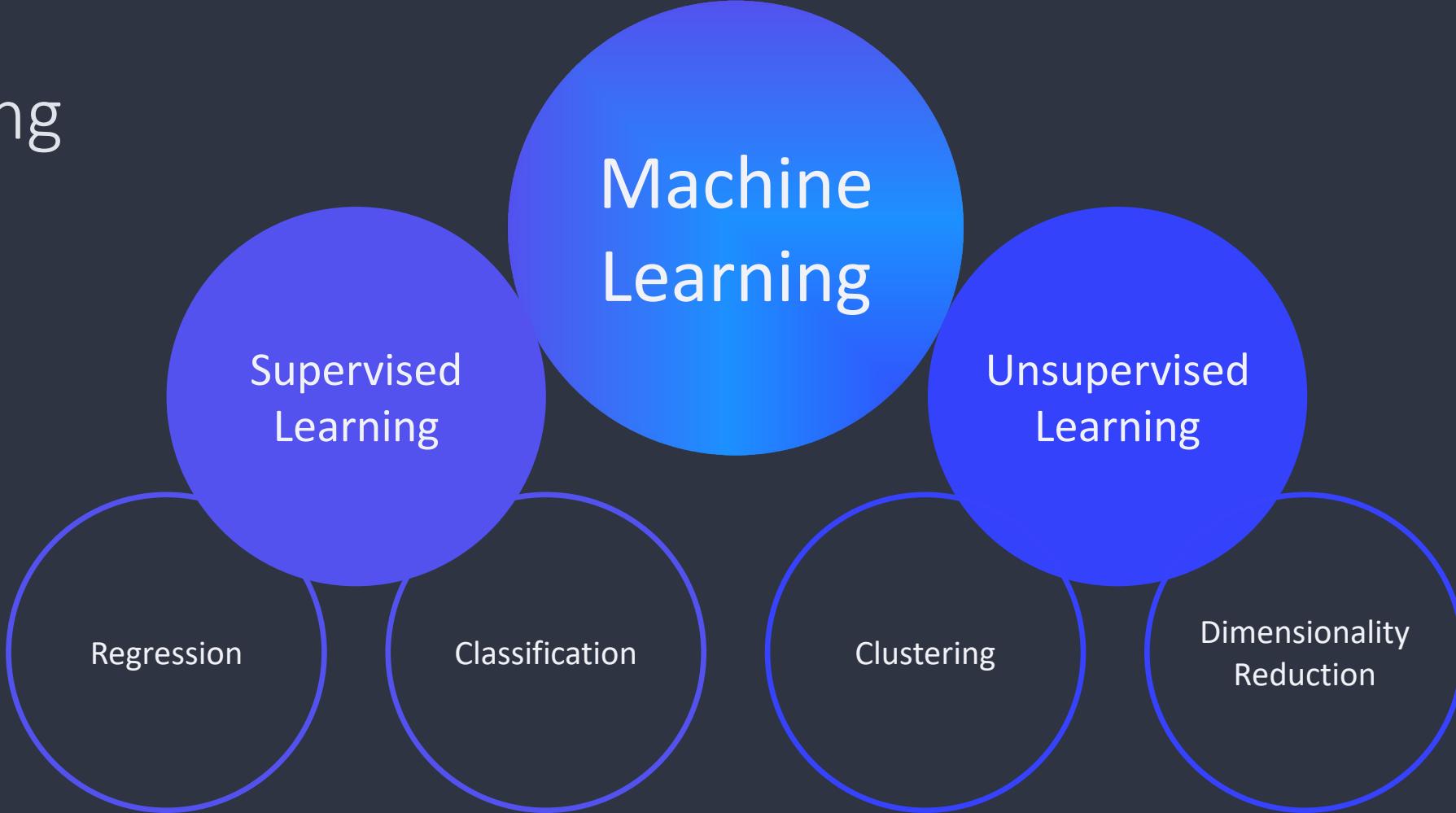


# 1. Clustering

# Clustering

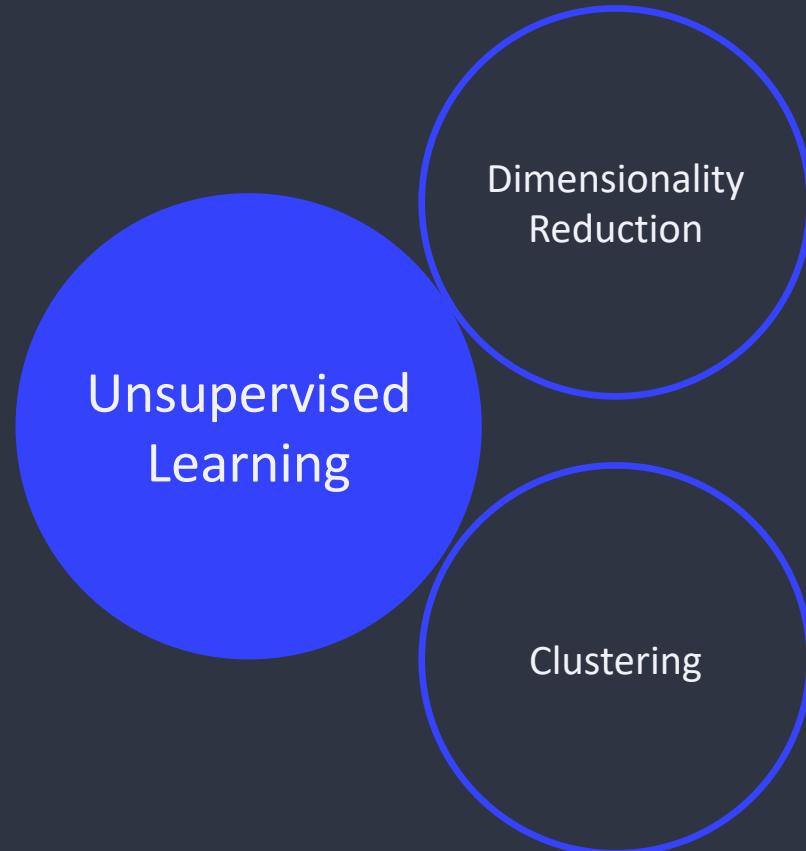


# Clustering



- To learn the mapping between inputs and outputs
- Labelled data (correct answers) is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.
- To learn hidden pattern from a set of inputs (no output).
- Unlabelled data is provided of past input (not a input & output pair) during the learning process. (no correct answers)

# Clustering

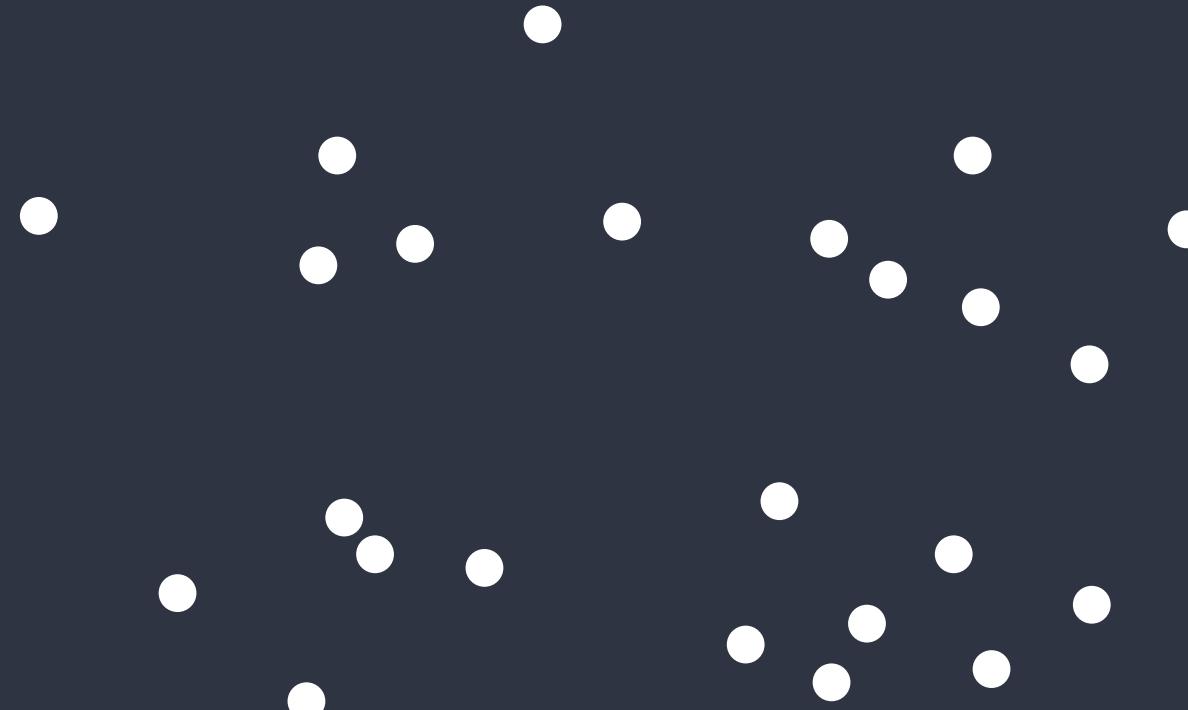


is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

# Clustering

is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).



training data

# Clustering

is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).



# Clustering

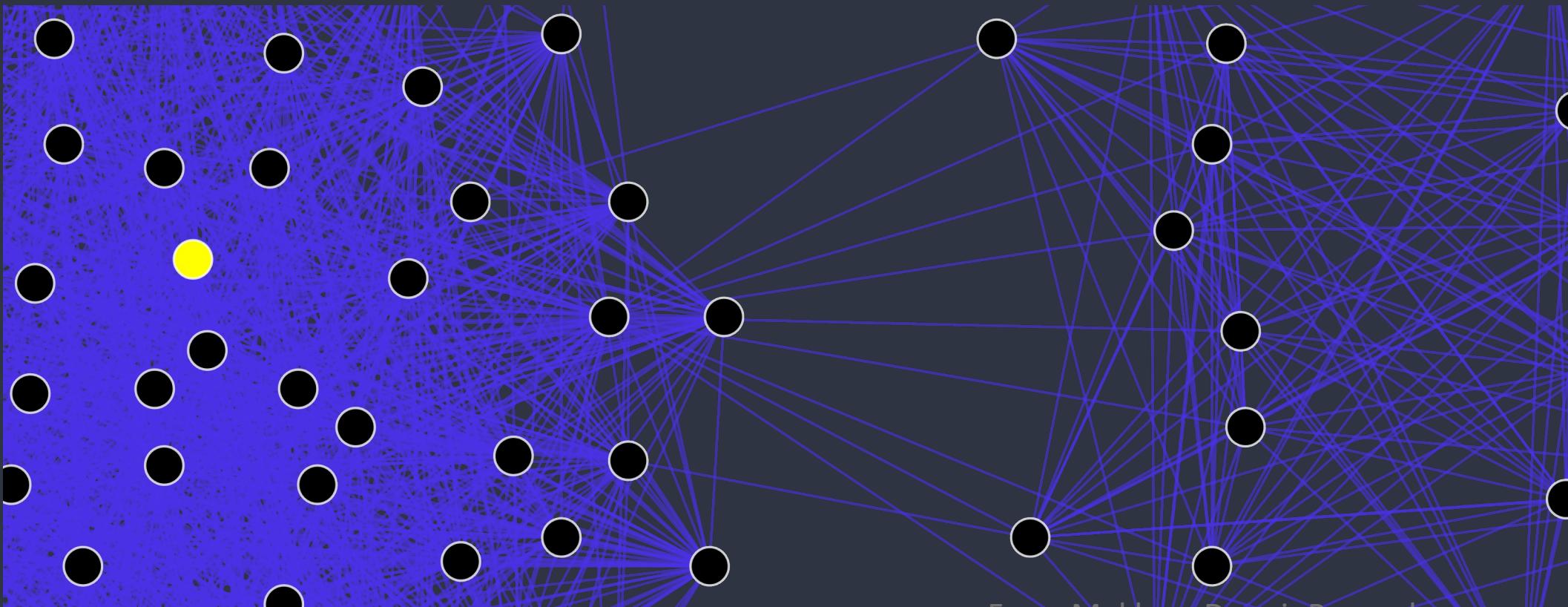
is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).



# Clustering

Is commonly used to explore a dataset.

- Social networks can be clustered to identify communities and to suggest missing connections between people.



# Clustering

Is commonly used to explore a dataset.

- Social networks can be clustered to identify communities and to suggest missing connections between people.
- In biology, clustering is used to find groups of genes with similar expression patterns.
- Recommender systems sometimes employ clustering to identify products or media that might appeal to a user.
- In marketing, clustering is used to find segments of similar consumers.

## 2. K-Means

# K-Means

Given  $\{\mathbf{x}_i\}_{i=1}^m \in \mathbb{R}^d$ , we want to find  $K$  centroids such that the total distance between these data points and their corresponding centroids is minimised.

More precisely, letting  $\{\mathbf{c}_k\}_{k=1}^K$  be the centroids and  $\{\Gamma_k\}_{k=1}^K$  be their corresponding partition. Here  $\{\Gamma_k\}_{k=1}^K$  are mutually disjoint and  $\bigcup_{k=1}^K \Gamma_j = [m]$ .

Total sum of distance (*k-means cost (lost) function*, depending on  $\{\mathbf{c}_k\}_{k=1}^K$  and  $\{\Gamma_k\}_{k=1}^K$ ):

$$J(\mathbf{c}_k, \Gamma_k) := \sum_{k=1}^K \left[ \sum_{i \in \Gamma_k} \boxed{d(\mathbf{x}_i, \mathbf{c}_k)} \right]$$

“distance”  
distortion of the cluster

# K-Means

$d(\mathbf{x}_i, \mathbf{c}_j)$  “distance”, measuring the similarity between  $\mathbf{x}_i$  and  $\mathbf{c}_j$ .

## Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

## Minkowski distance

$$d(x, x') = \sqrt[p]{\sum_{i=1}^n |x_i - x'_i|^p}$$

## Manhattan distance

$$d(x, x') = \sum_{i=1}^n |x_i - x'_i|$$

## Hamming distance

$$d(x, x') = \sum_{i=1}^n \mathbb{1}_{x_i \neq x'_i}$$

...

# K-Means

## Intuition

- Find  $K$  centroids which represent the original data set.
- The performance is measured by the cost function  $J(c_k, \Gamma_k)$ .  
The smaller the value of cost function is, the better the performance is.

## Aim

- To find a good choice of centroids and partition by solving  $\min_{\{c_k, \Gamma_k\}_{k=1}^K} J(c_k, \Gamma_k)$   
which is an optimisation programme.
- Ideally we want to find the global minimiser of this optimisation programme.

# K-Means

## Lloyd's Algorithm (benchmark)

1. Clusters the data into  $K$  groups ( $K$  is predefined).
2. Initialisation: select  $K$  points as cluster centroids.
3. Cluster Assignment: Assign each data point to its closest cluster centroids.
4. Centroid Calculation : (Re-)calculate the centroid or mean of all objects in each cluster.
5. Repeat steps (3) and (4) until the same data points are assigned to each cluster in consecutive rounds.

### Question:

- Is it possible that the algorithm never stops?
- How to determine the number of clusters  $K$ ?

# K-Means

Lloyd's Algorithm (benchmark)

K-means with Euclidean distance

$$\sum_{k=1}^K \sum_{i \in \Gamma_k} d(\mathbf{x}_i, \mathbf{c}_k) \longrightarrow \sum_{k=1}^K \sum_{i \in \Gamma_k} \|\mathbf{x}_i - \mathbf{c}_k\|^2$$

# K-Means

Lloyd's Algorithm (benchmark)

K-means with Euclidean distance  $\sum_{k=1}^K \sum_{i \in \Gamma_k} \|x_i - c_k\|^2$

Assume the centroids  $\{c_k^{(t)}\}_{k=1}^K$  at time  $t$ .

**Cluster Assignment**: indices of data are assigned to  $\Gamma_k^{(t)}$ , which is associated with  $c_k^{(t)}$ , if

$$\Gamma_k^{(t)} := \{i : \|x_i - c_k^{(t)}\| \leq \|x_i - c_j^{(t)}\|, \forall j \neq k\}.$$

In fact,  $\Gamma_k^{(t)}$  is the minimiser to  $J(c_k^{(t)}, \Gamma_k)$ , where centroid  $c_k^{(t)}$  is fixed:

$$J(c_k^{(t)}, \Gamma_k) := \sum_{k=1}^K \sum_{i \in \Gamma_k} \|x_i - c_k\|^2.$$

# K-Means

Lloyd's Algorithm (benchmark)

K-means with Euclidean distance  $\sum_{k=1}^K \sum_{i \in \Gamma_k} \|\mathbf{x}_i, \mathbf{c}_k\|^2$

Assume the centroids  $\{\mathbf{c}_k^{(t)}\}_{k=1}^K$  at time  $t$ .

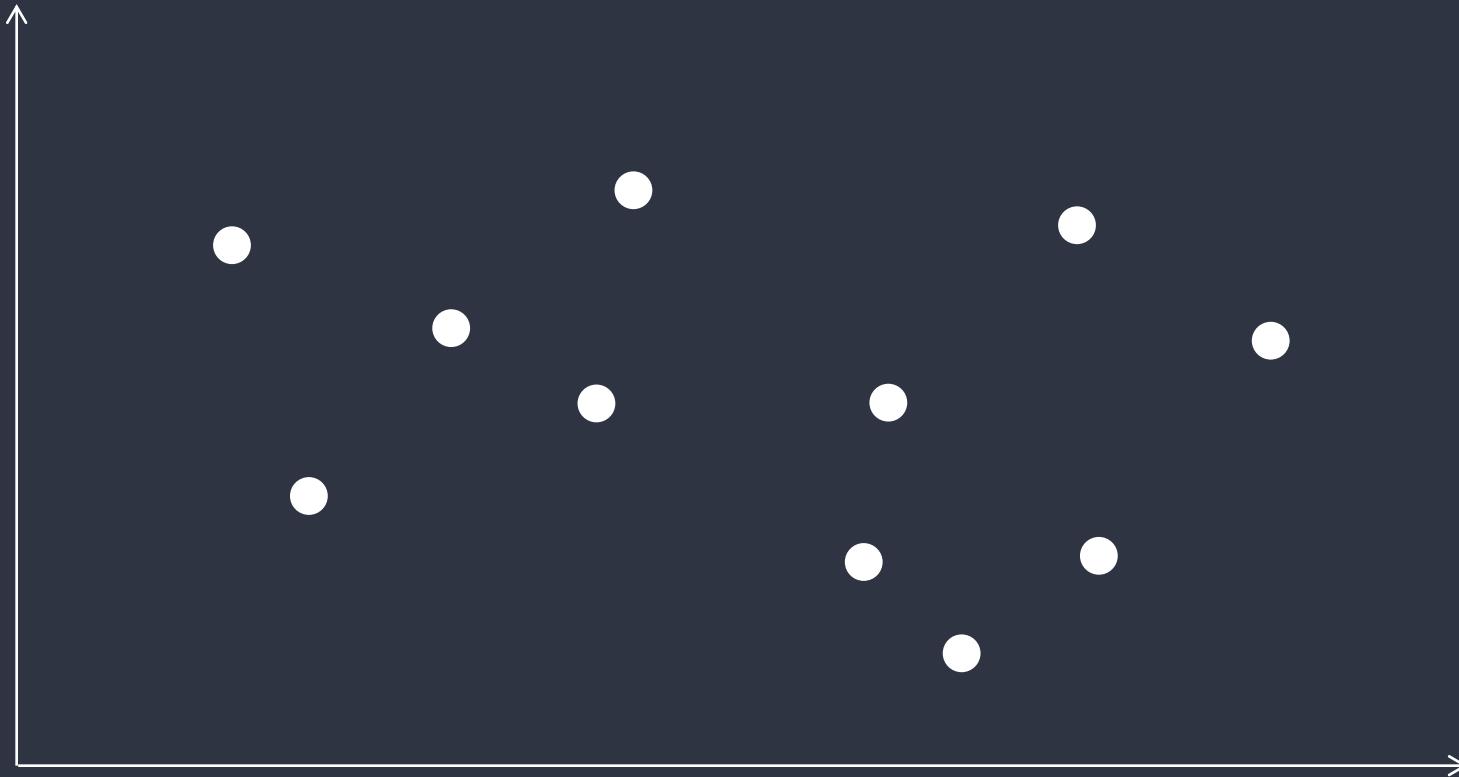
**Centroid Calculation:** by definition, the centroid for the cluster  $\Gamma_k$  is

$$\mathbf{c}_k^{(t+1)} := \operatorname{argmin}_{\mathbf{c}_k \in \mathbb{R}^d} \sum_{i \in \Gamma_k^{(t)}} \|\mathbf{x}_i - \mathbf{c}_k\|^2$$

The minimiser has a closed form solution, which is exactly the sample average of data point in  $\Gamma_k^{(t)}$ , i.e.,  $\mathbf{c}_k^{(t+1)} = \frac{1}{|\Gamma_k^{(t)}|} \sum_{i \in \Gamma_k^{(t)}} \mathbf{x}_i$ .

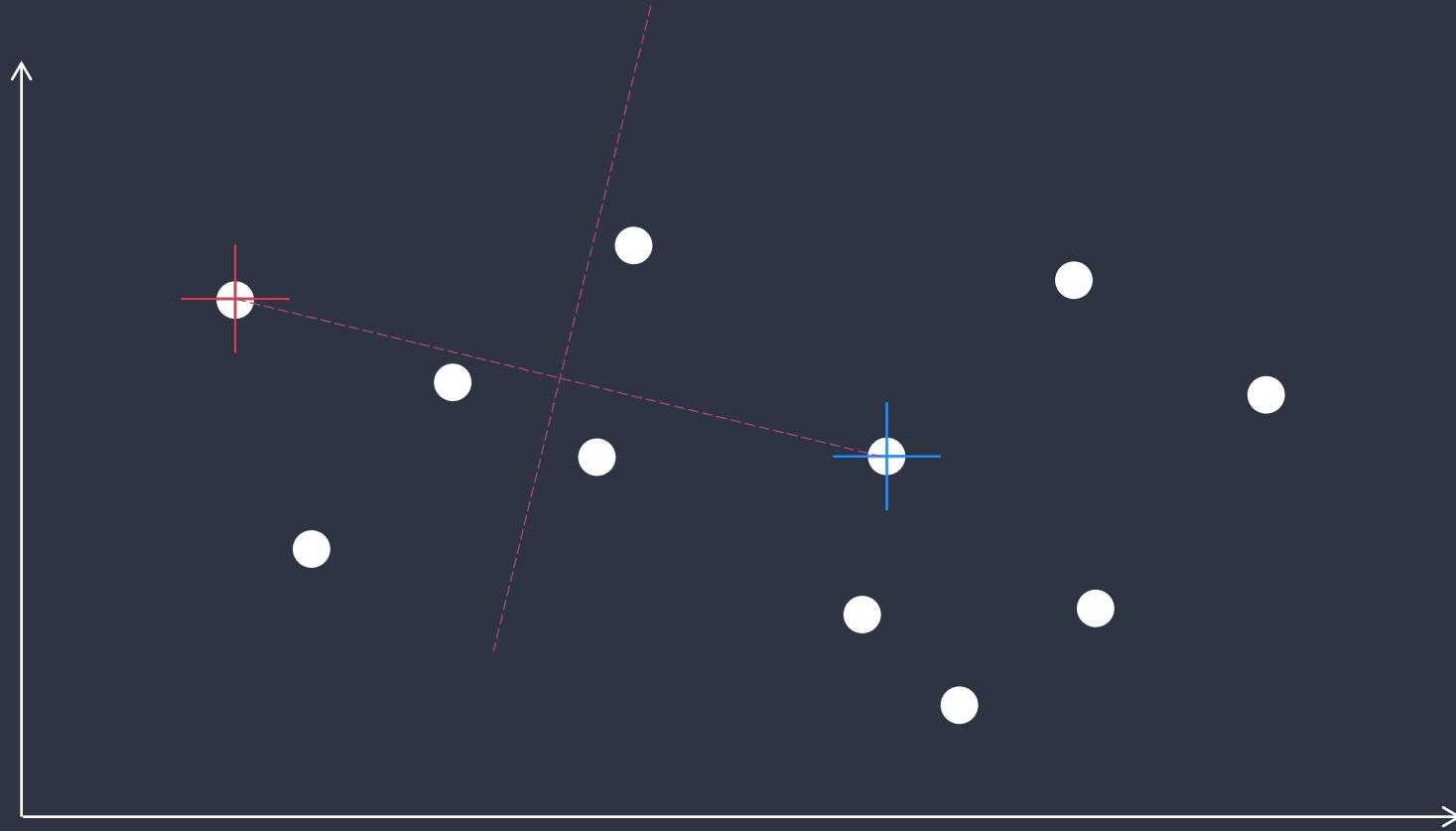
# K-Means

## Intuition - Training



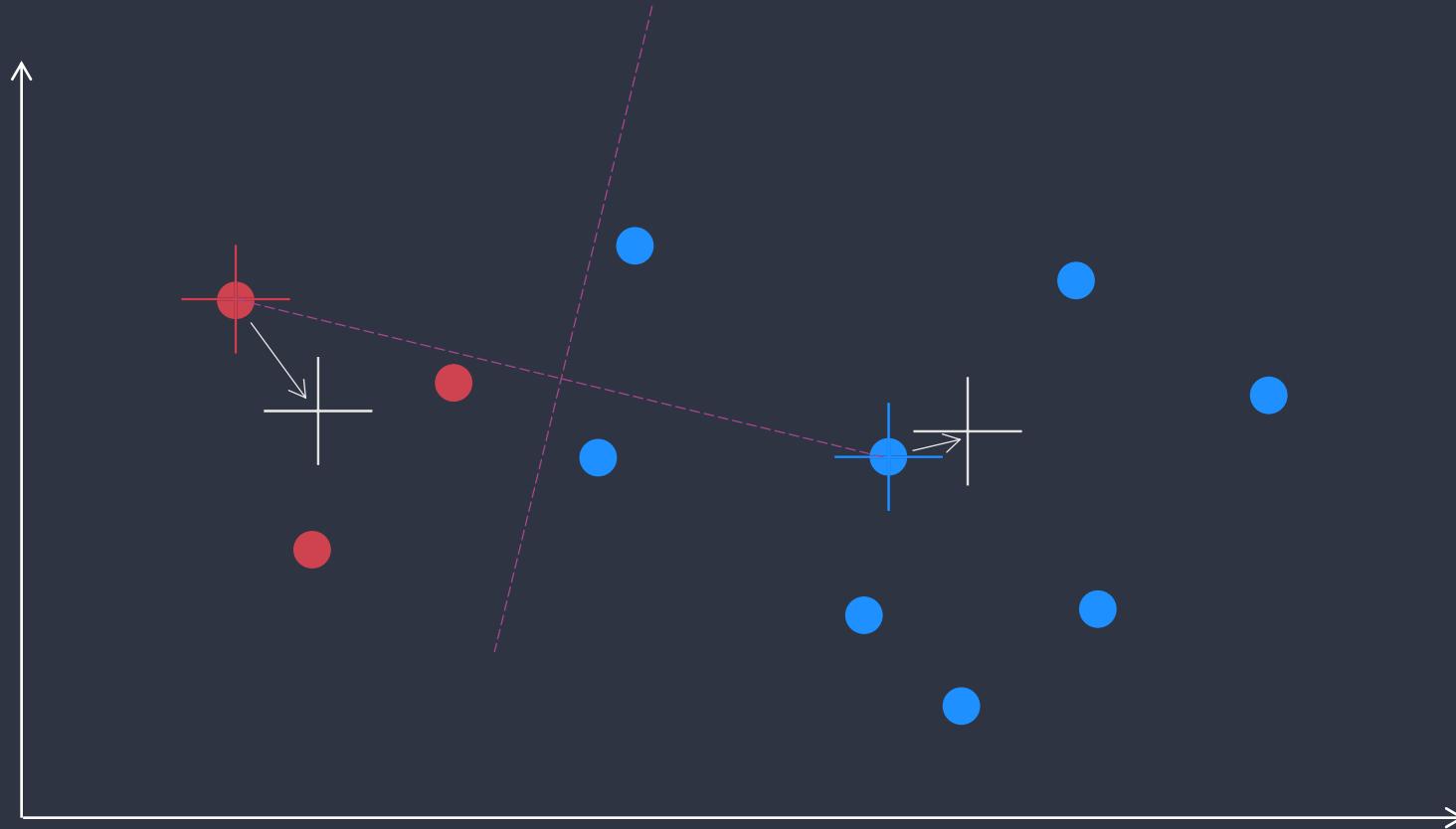
# K-Means

## Intuition - Training



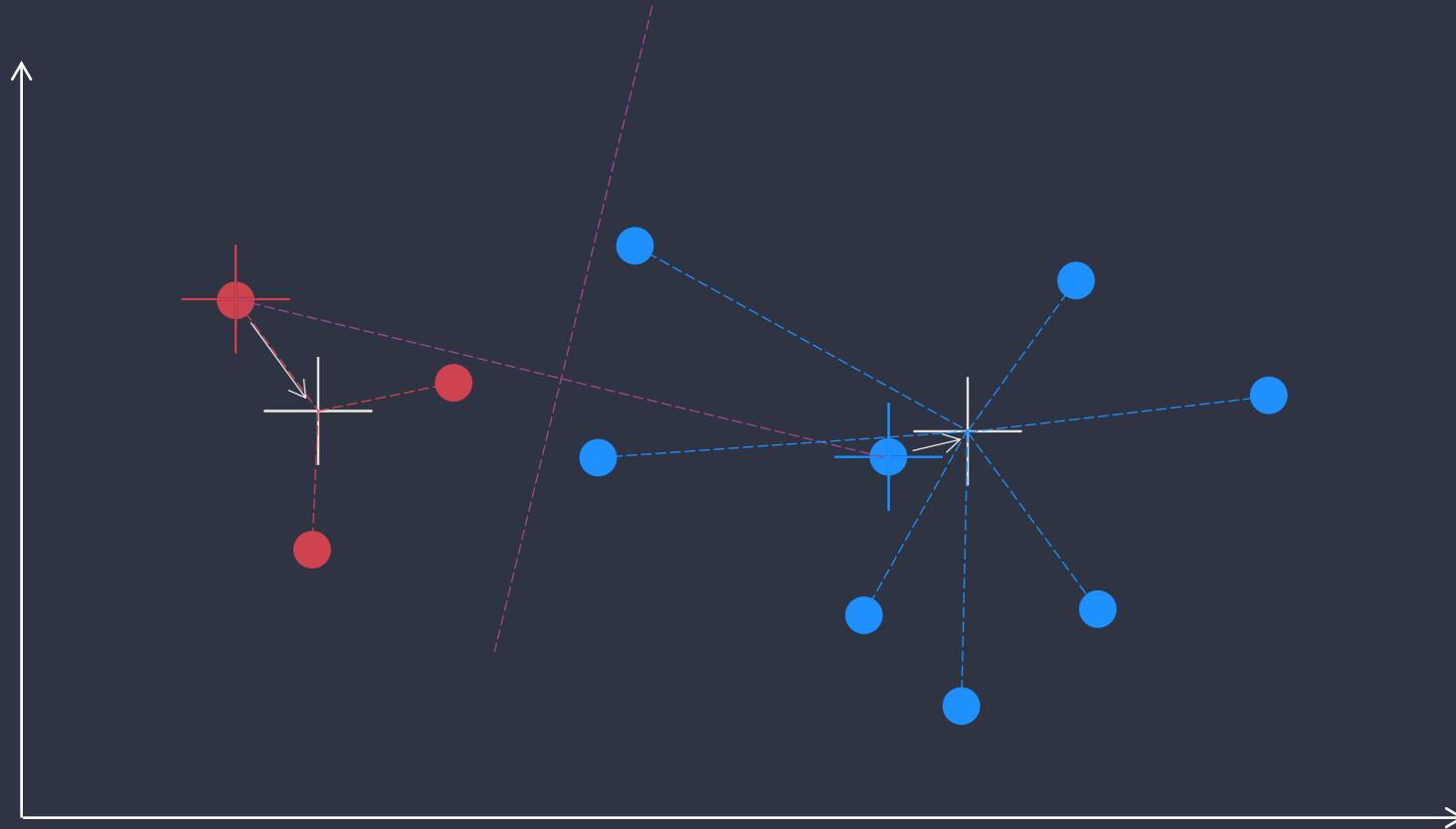
# K-Means

## Intuition - Training



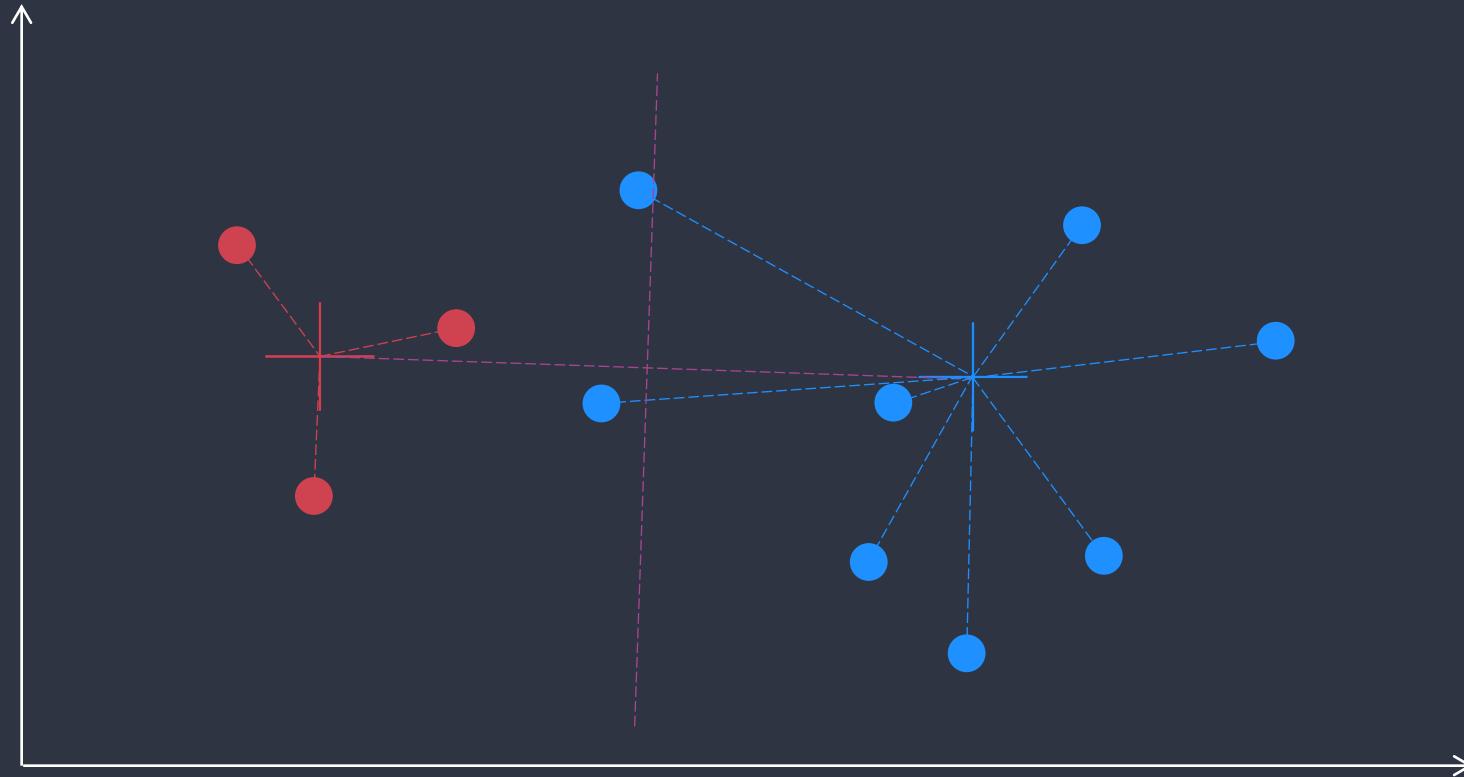
# K-Means

## Intuition - Training



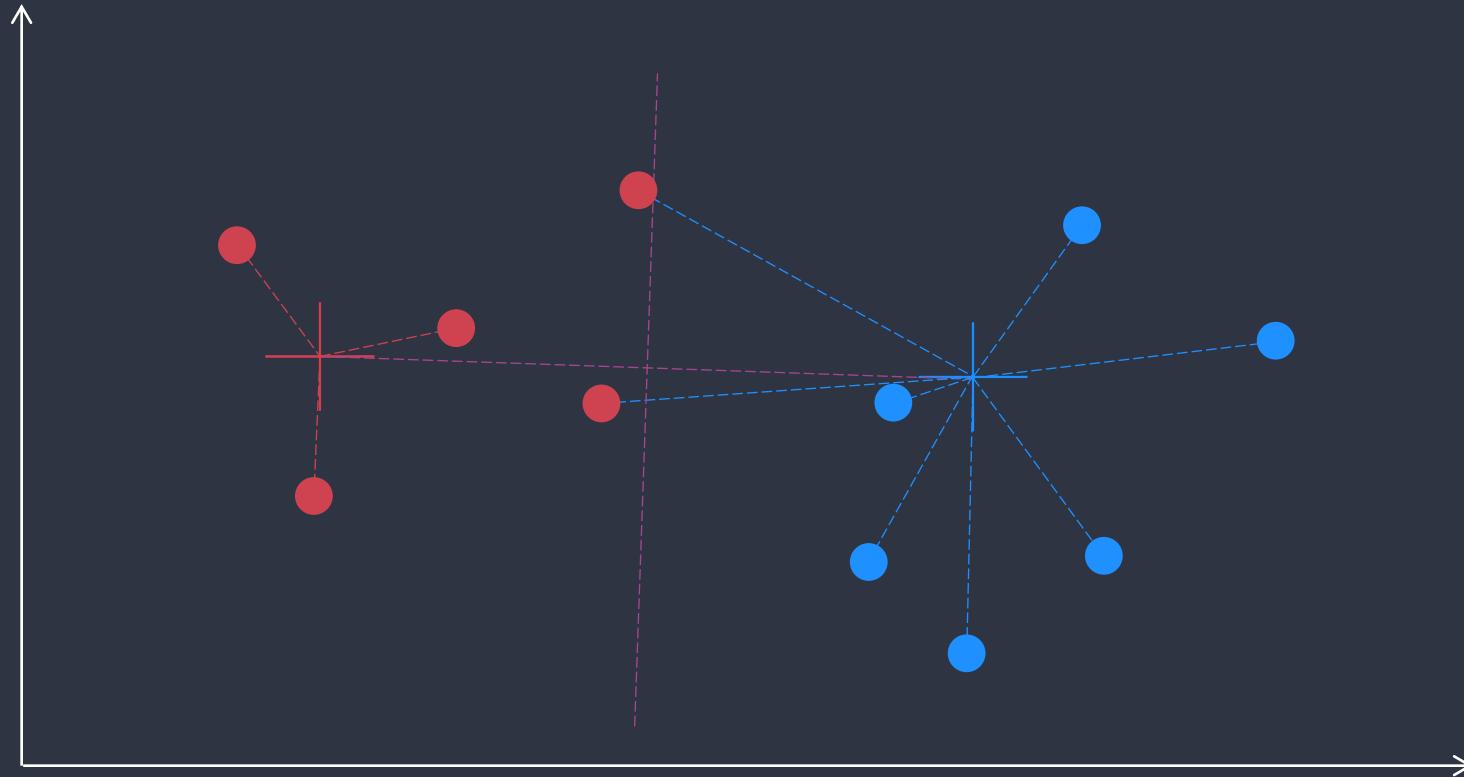
# K-Means

## Intuition - Training



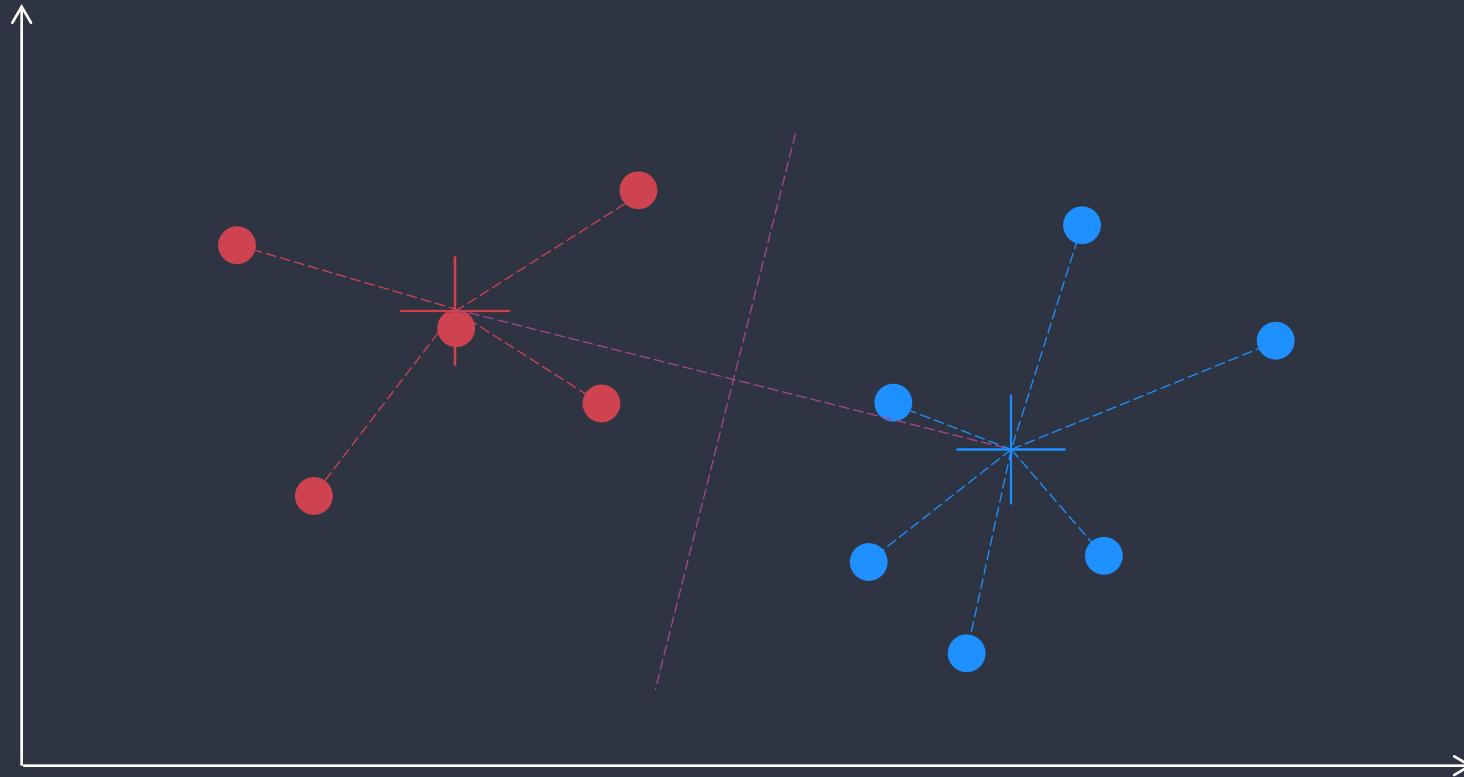
# K-Means

## Intuition - Training



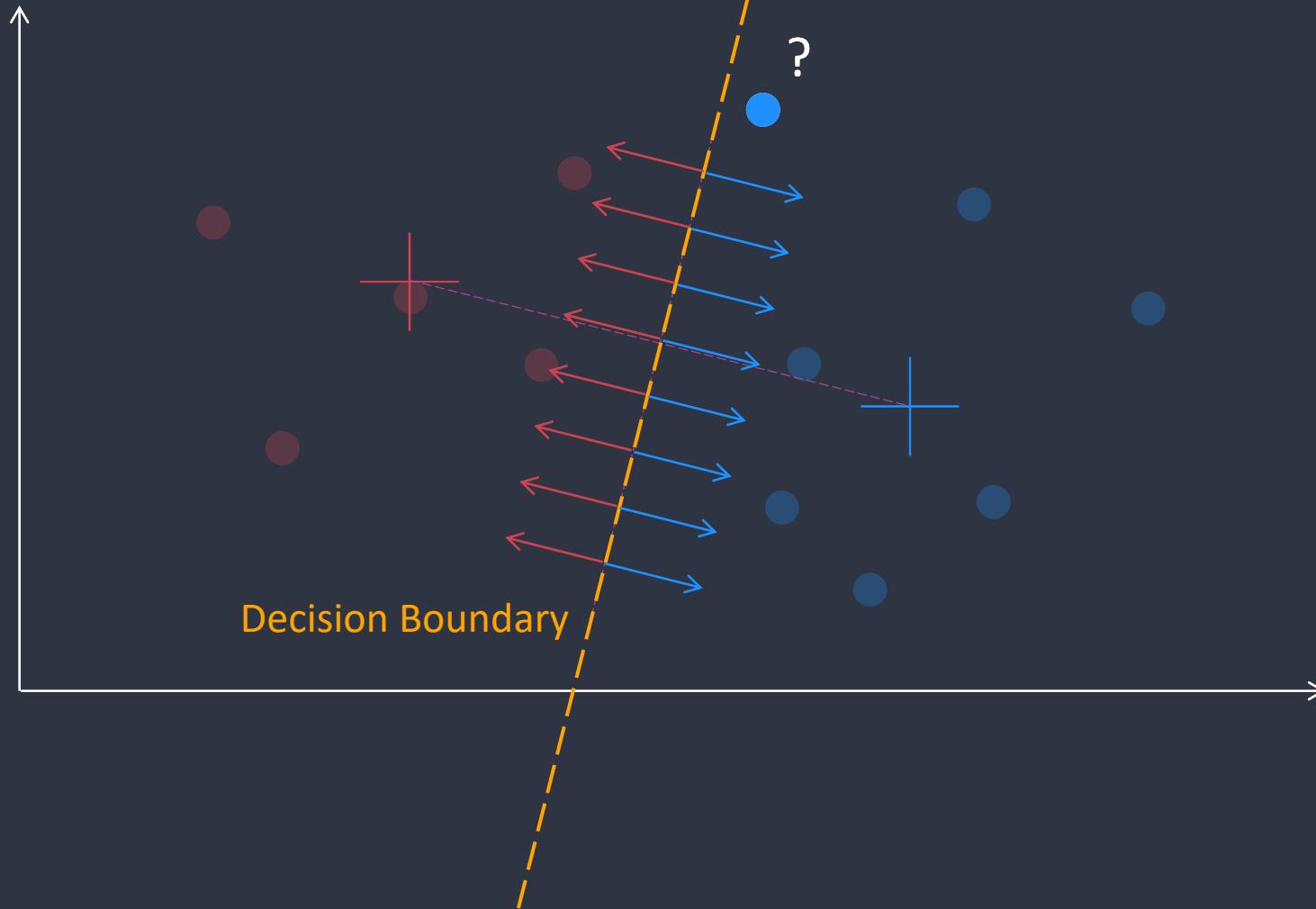
# K-Means

## Intuition - Training



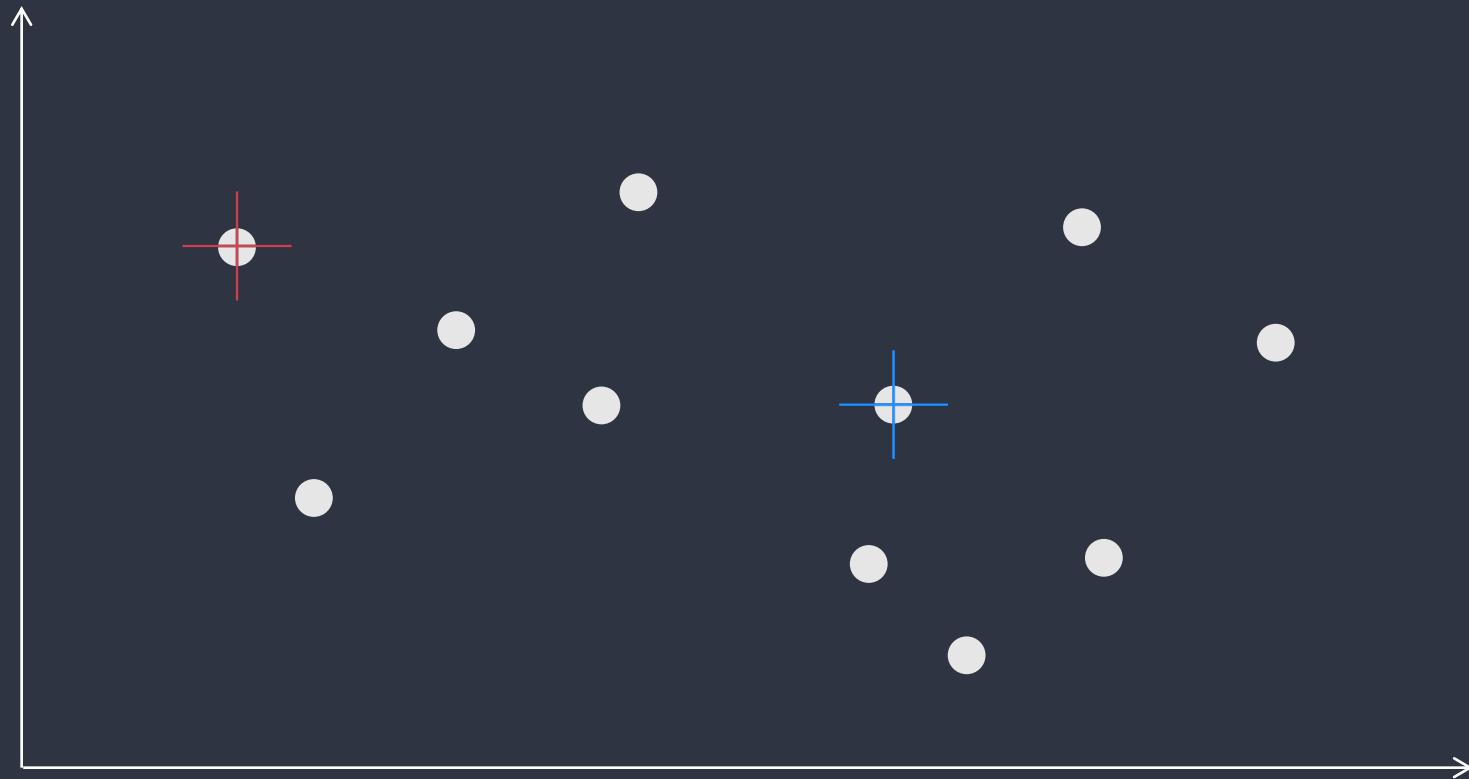
# K-Means

## Intuition - Making predictions



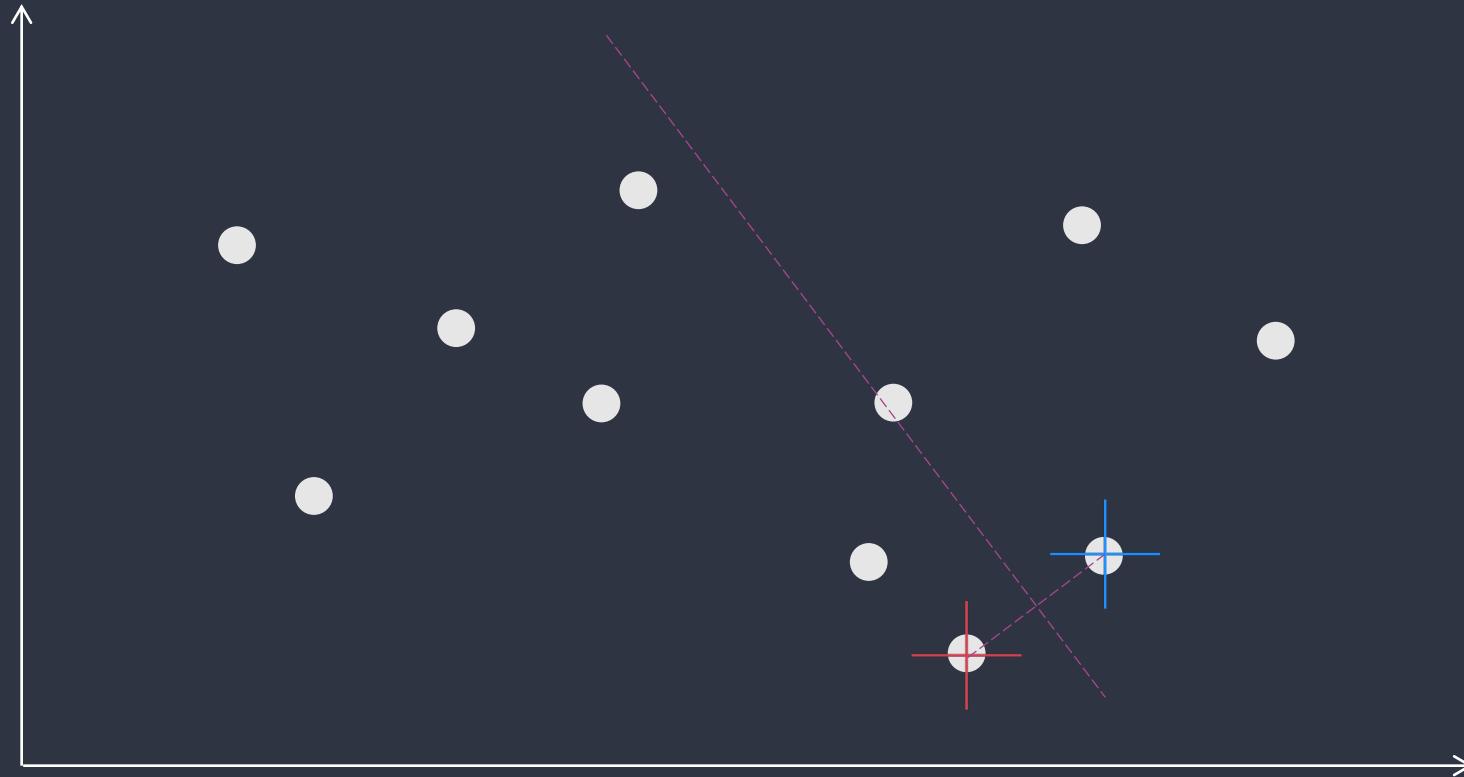
# K-Means

## Intuition - Another example



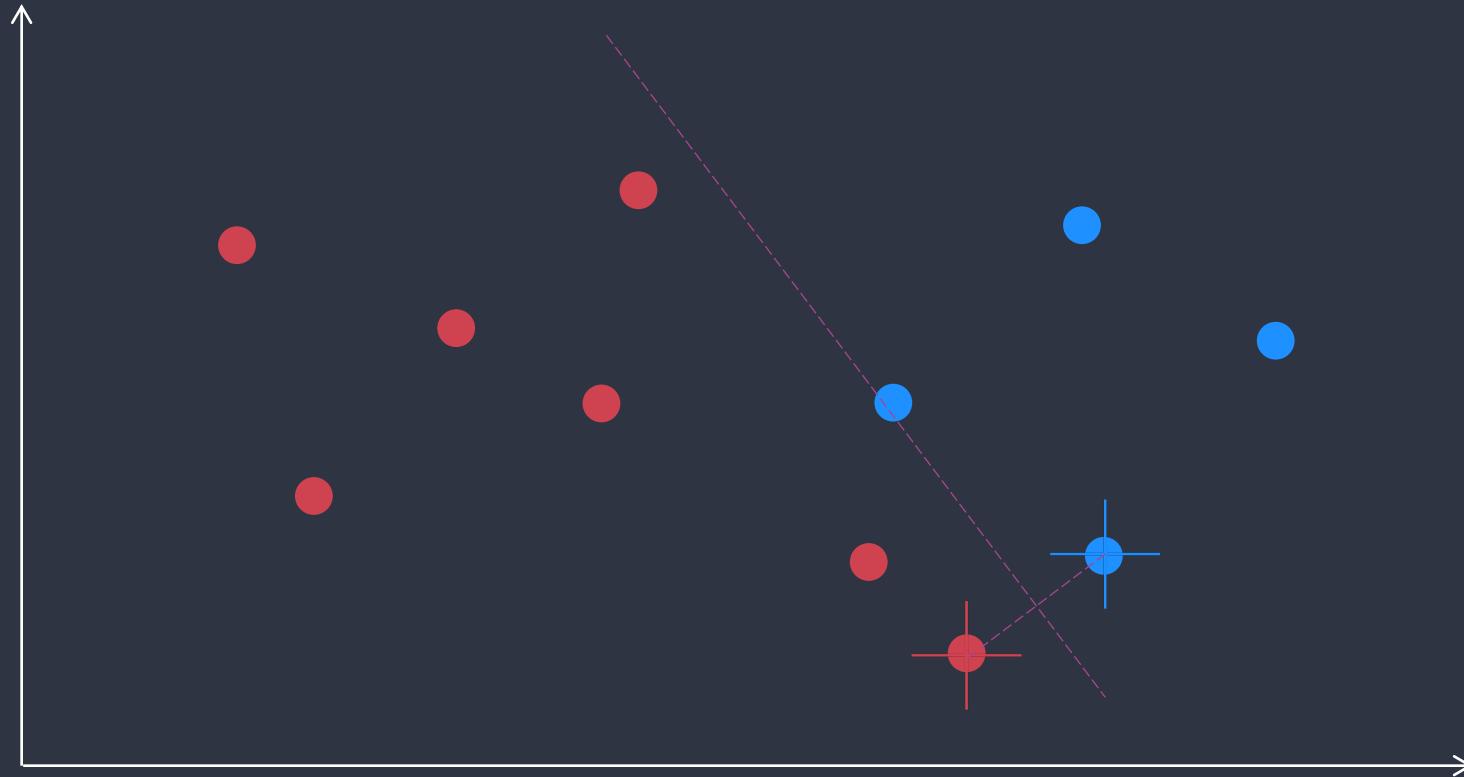
# K-Means

## Intuition - Another example



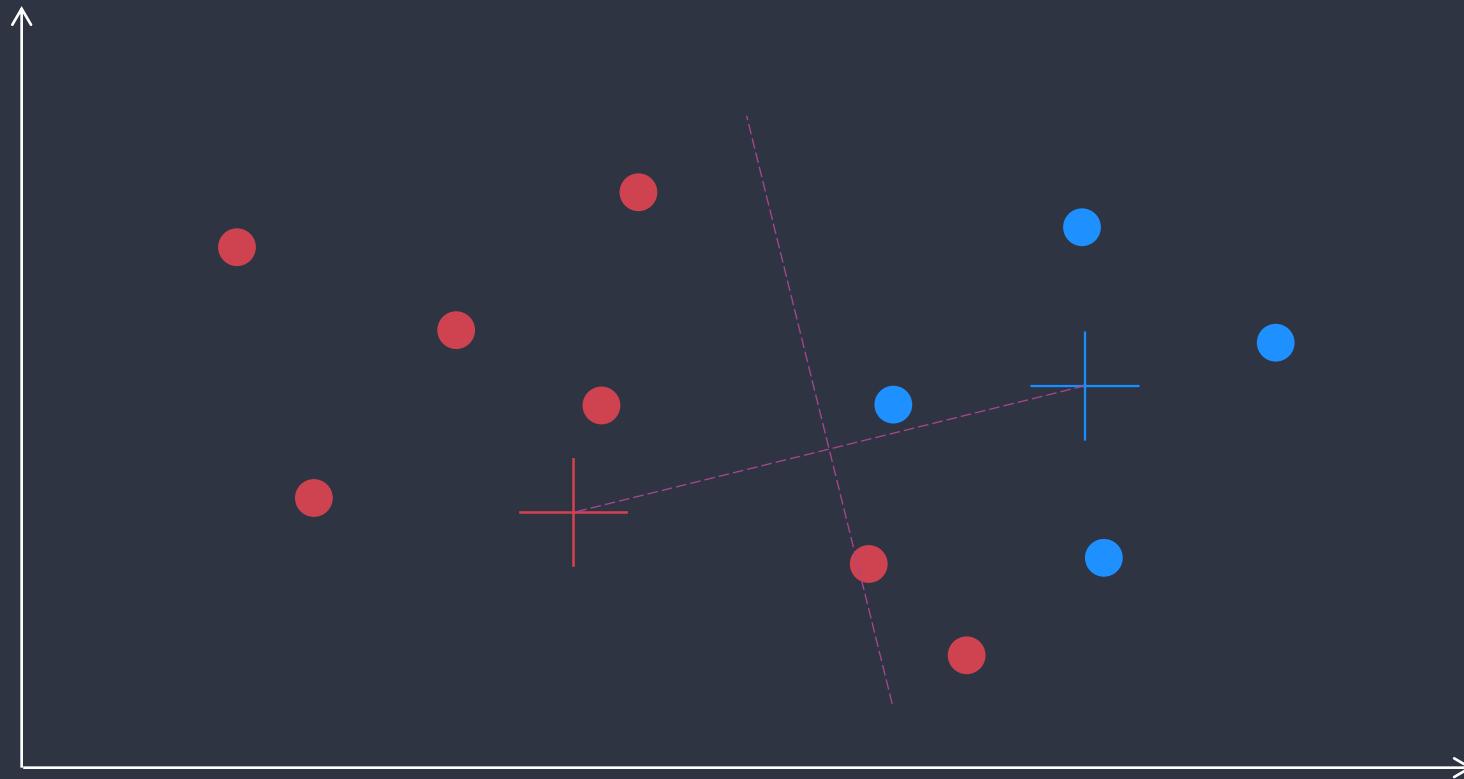
# K-Means

## Intuition - Another example



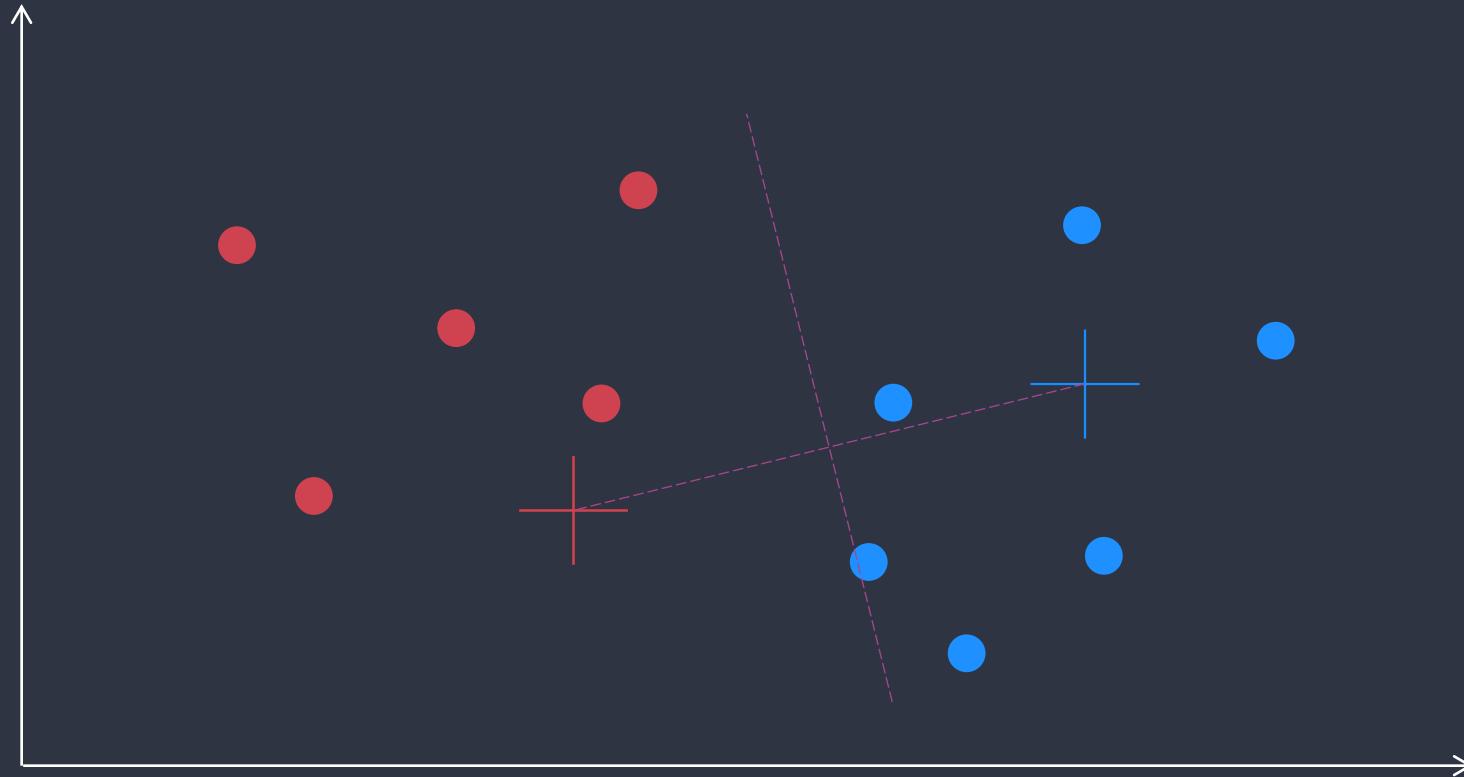
# K-Means

## Intuition - Another example



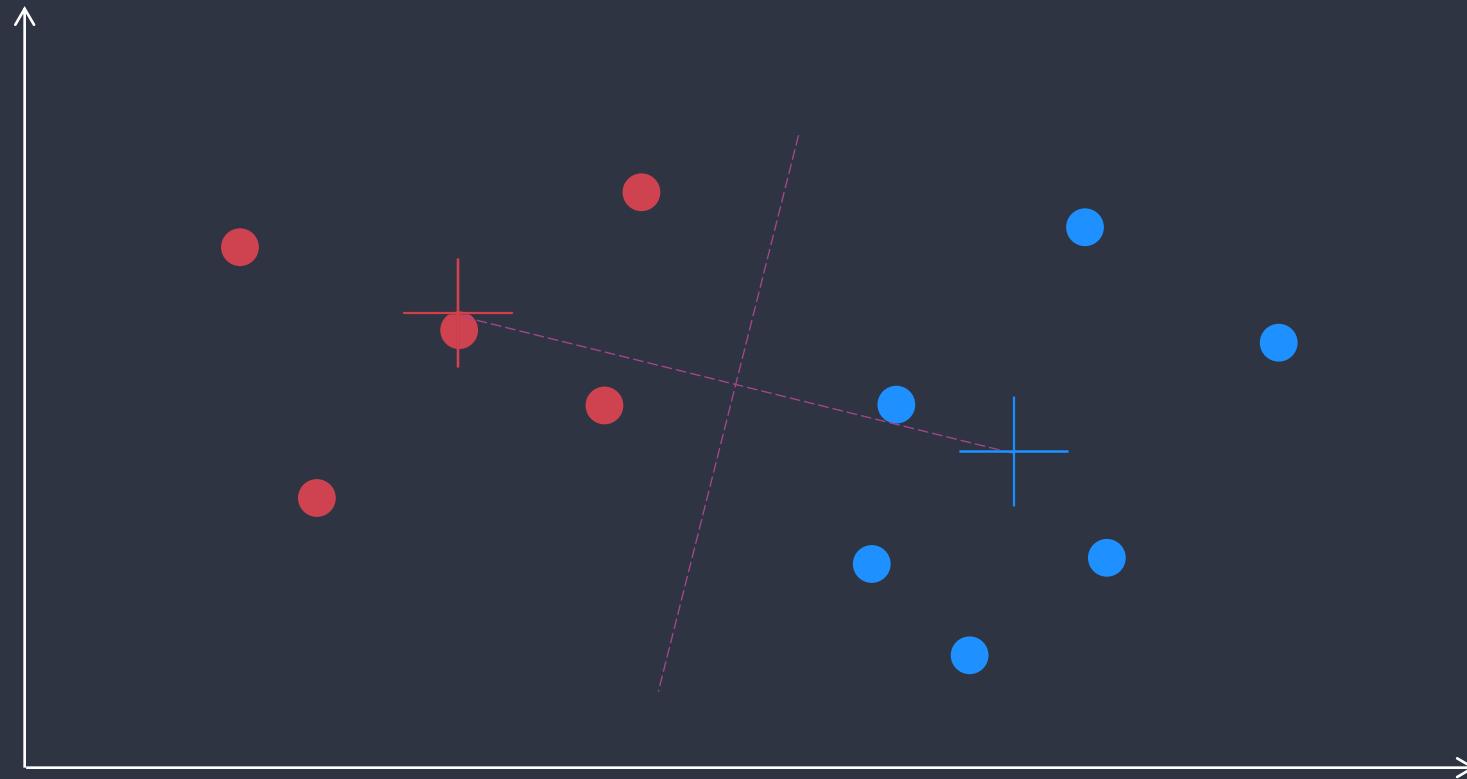
# K-Means

## Intuition - Another example



# K-Means

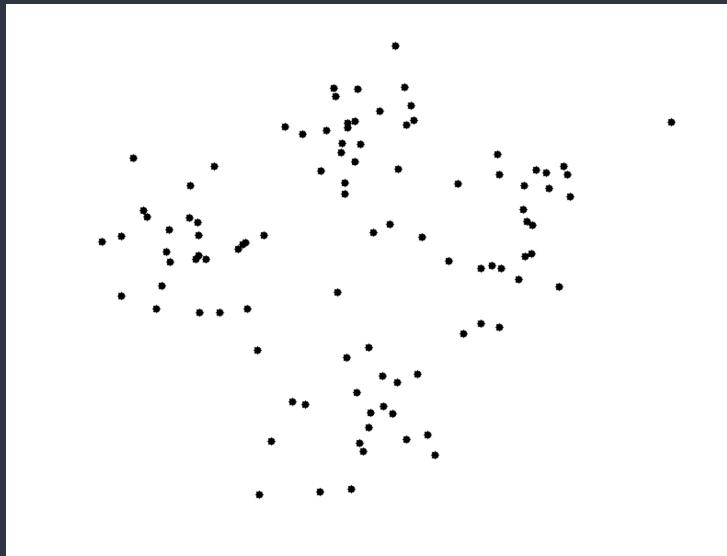
## Intuition - Another example



The result is the same as the previous example

# K-Means

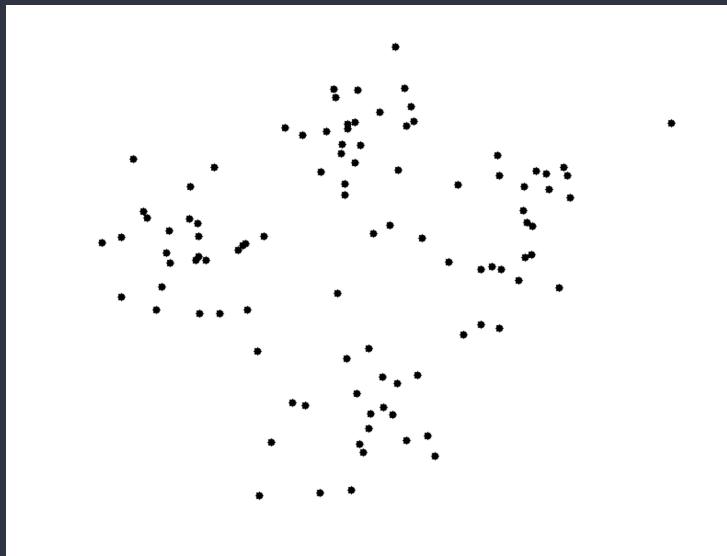
## Local Optima



(Starting with 4 left-most points)



(Starting with 4 right-most points)



(Starting with 4 top points)



(Starting with 4 bottom points)

# K-Means

## Local Optima

To avoid local optima, K-Means is often repeated dozens or even hundreds of times. In each iteration, it is randomly initialised to different starting cluster positions.

The initialisation that minimises the cost function best is selected.

# K-Means

## The Elbow Method

- $K$
- A hyperparameter that specifies the number of clusters to be created.
  - Cannot be determined by k-means.
  - Must be a positive integer that is less than the number of instances in training set.
  - Could be specified by the clustering problem's context.
  - Some problems may not require a specific  $K$ , and the optimal  $K$  might be ambiguous.

# K-Means

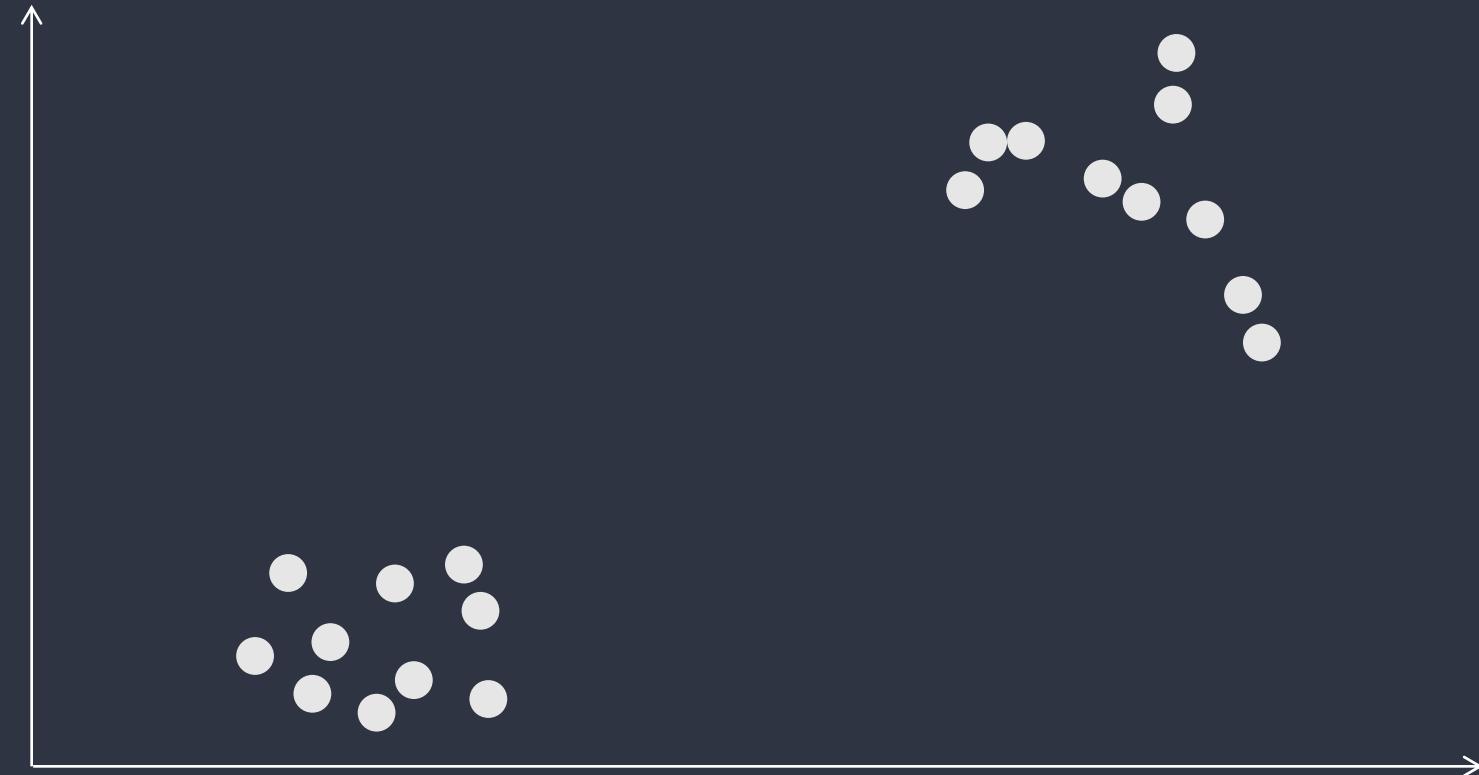
## The Elbow Method

- Some problems may not require a specific  $K$ , and the optimal  $K$  might be ambiguous.  
In this case, the optimal  $K$  can be estimated using the elbow method.
  - Plots the value of the cost function provided by different value of  $K$ .
  - As  $K$  increases, the average distortion will decrease; each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids.
  - However, the improvements to the average distortion will decline as  $K$  increases. The value of  $K$  at which the improvement to the distortion declines the most is called the elbow.

# K-Means

## The Elbow Method

EXAMPLE.



# K-Means

## The Elbow Method

EXAMPLE.

```
import numpy as np
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt

cluster1 = np.random.uniform(0.5, 1.5, (2, 10))
cluster2 = np.random.uniform(3.5, 4.5, (2, 10))
X = np.hstack((cluster1, cluster2)).T
X = np.vstack((x, y)).T

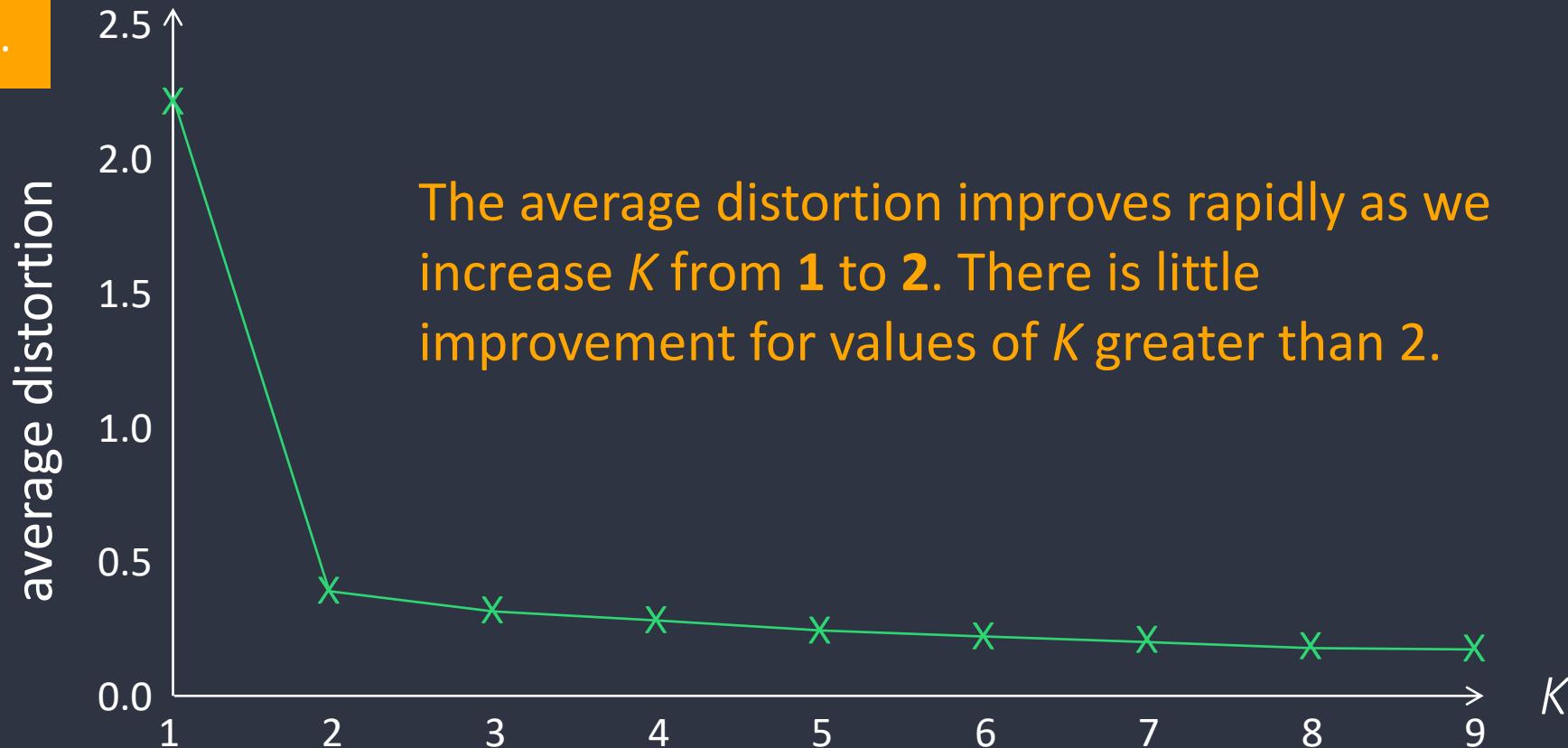
K = range(1, 10)
meandistortions = []
for k in K:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    meandistortions.append(sum(np.min(cdist(X,
                                              kmeans.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])

plt.plot(K, meandistortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Average distortion')
plt.title('Selecting k with the Elbow Method')
plt.show()
```

# K-Means

## The Elbow Method

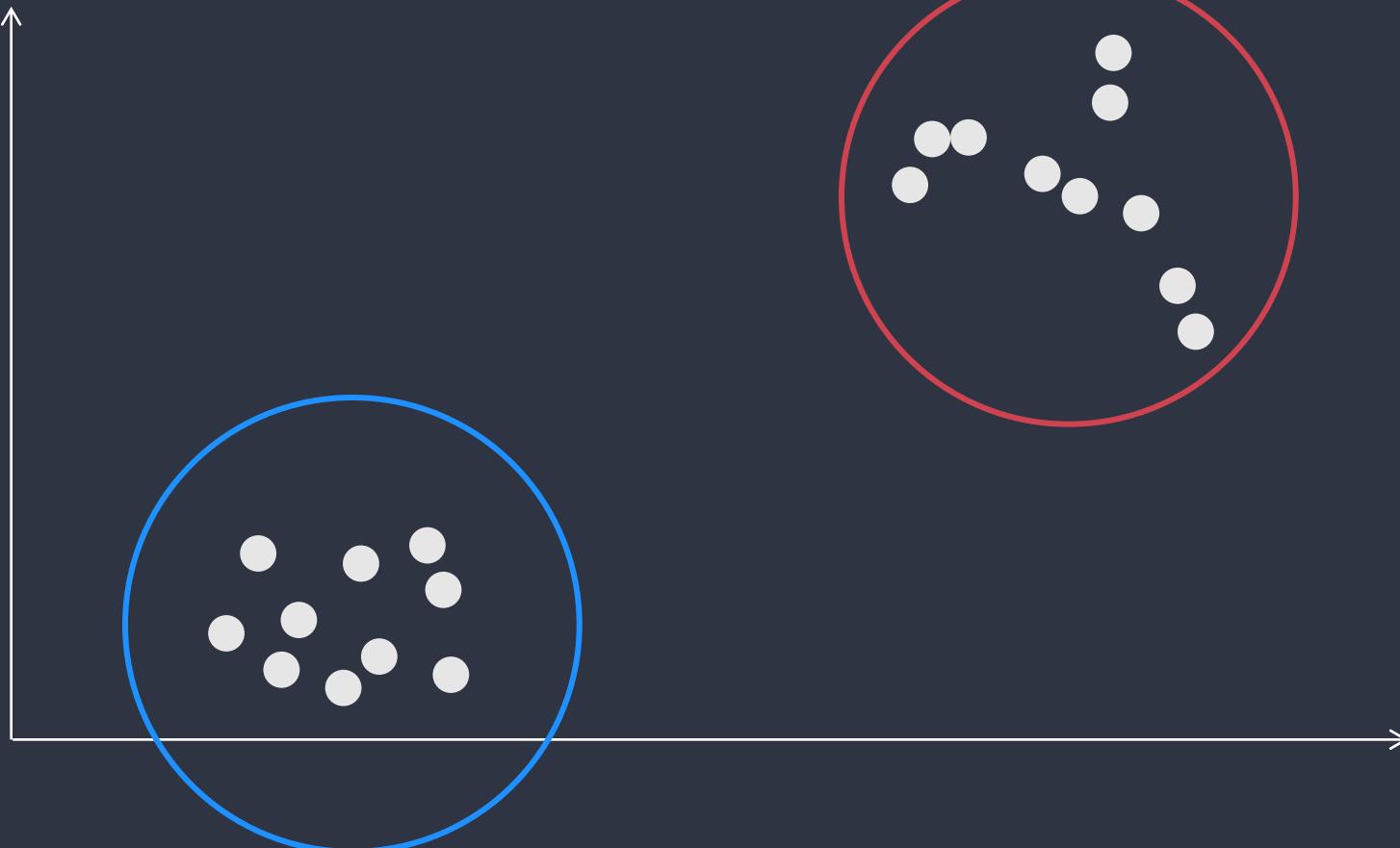
EXAMPLE.



# K-Means

## The Elbow Method

EXAMPLE.



# Summary

# Summary

## Unsupervised Learning: Clustering & Dimensionality Reduction

Clustering: the task of grouping a set of objects in such way that objects in same group (cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

K-Means: aim, Lloyd's algorithm, local optima, the elbow method

## Next Lecture

Dimensionality Reduction

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Dimensionality Reduction

Dr Yang Long

# Previously

## Machine Learning Algorithms



Regression



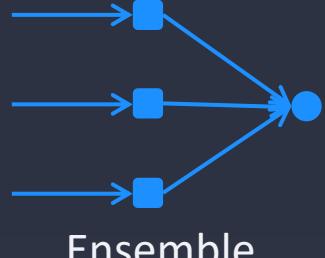
Regularisation



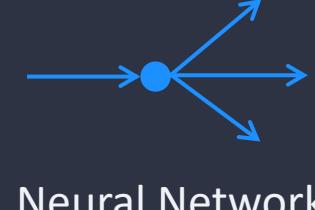
Clustering



Bayesian



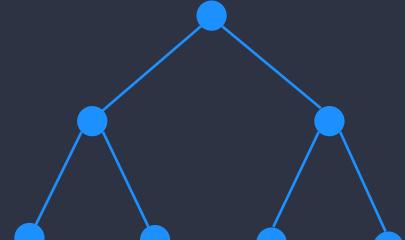
Ensemble



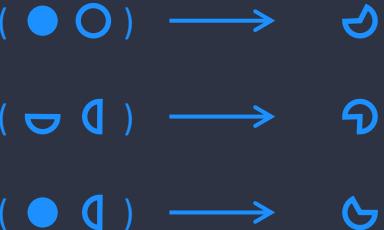
Neural Network



Instance-based



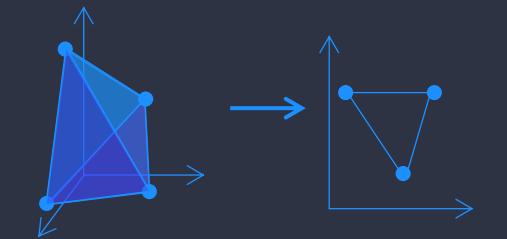
Tree-based



Associated Rule Learning



Deep Learning



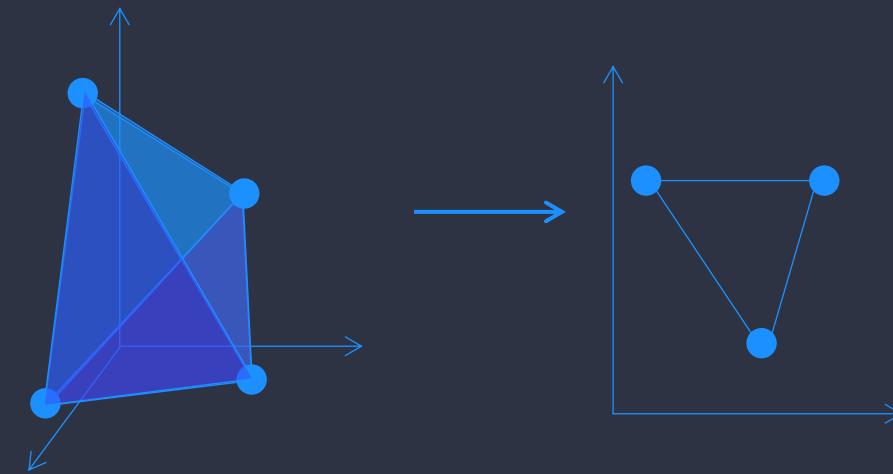
Dimensionality Reduction



Reinforcement Learning

# Lecture Overview

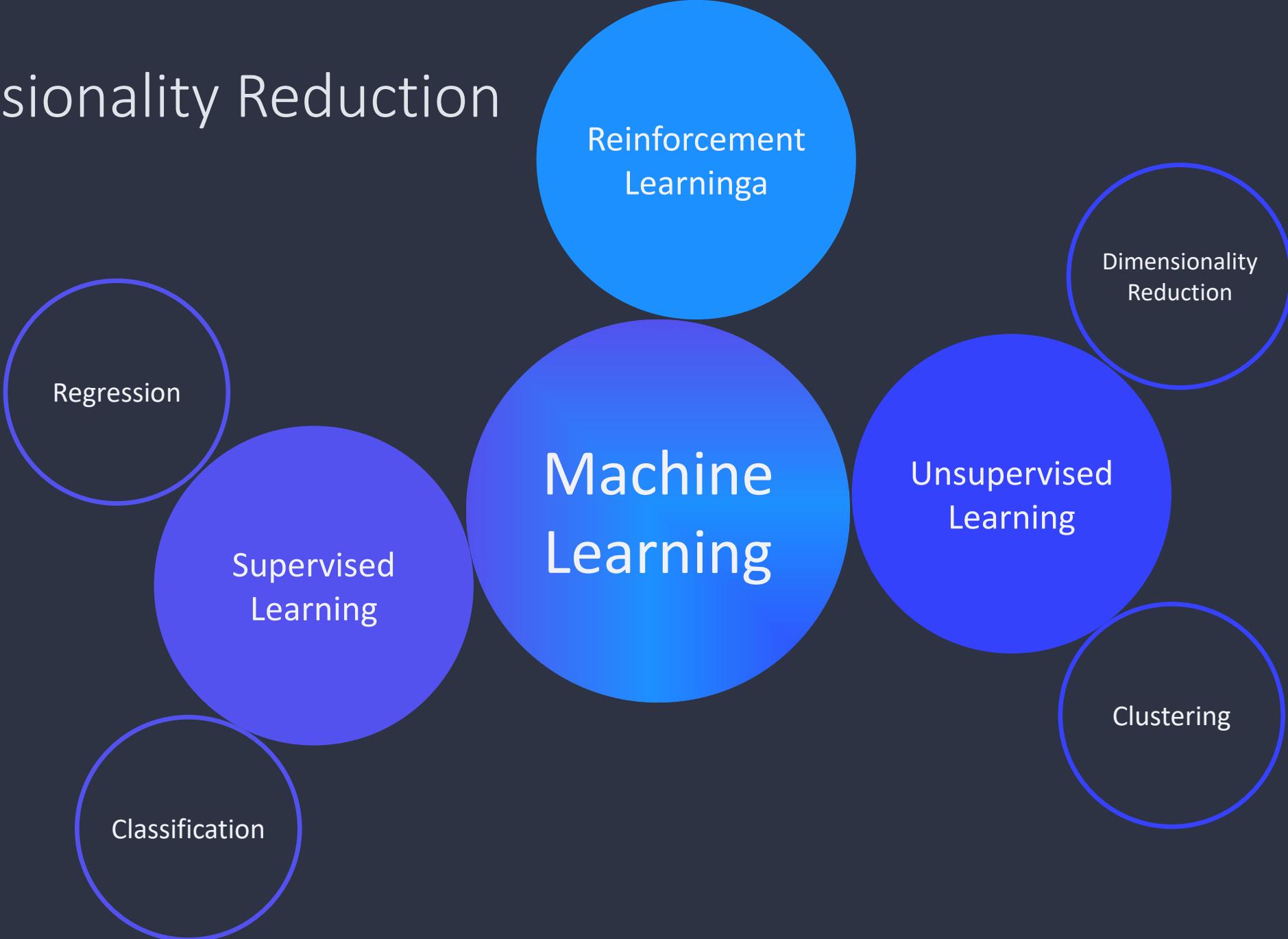
1. Dimensionality Reduction
2. Principal Component Analysis



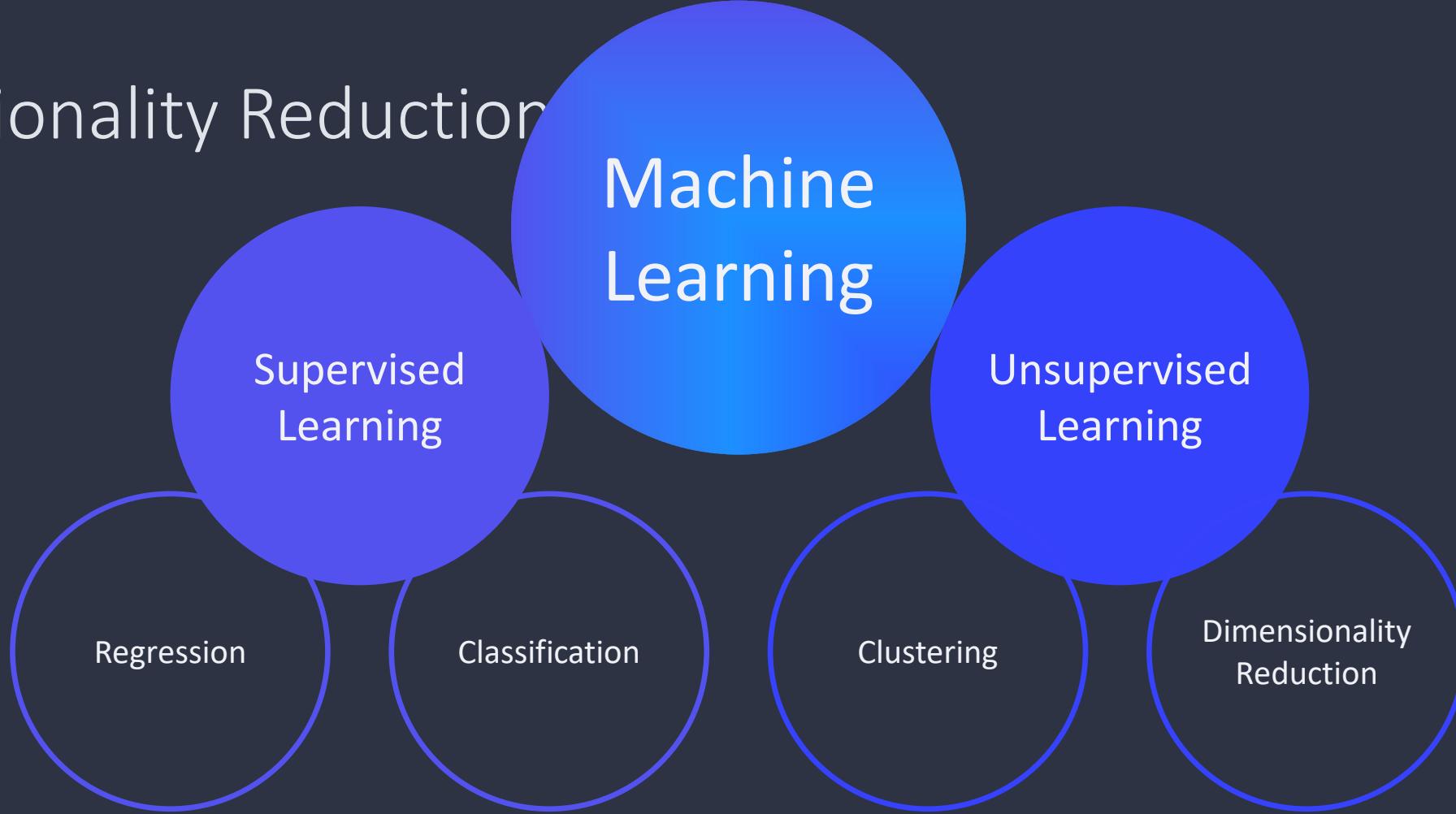
Dimensionality Reduction

# 1. Dimensionality Reduction

# Dimensionality Reduction



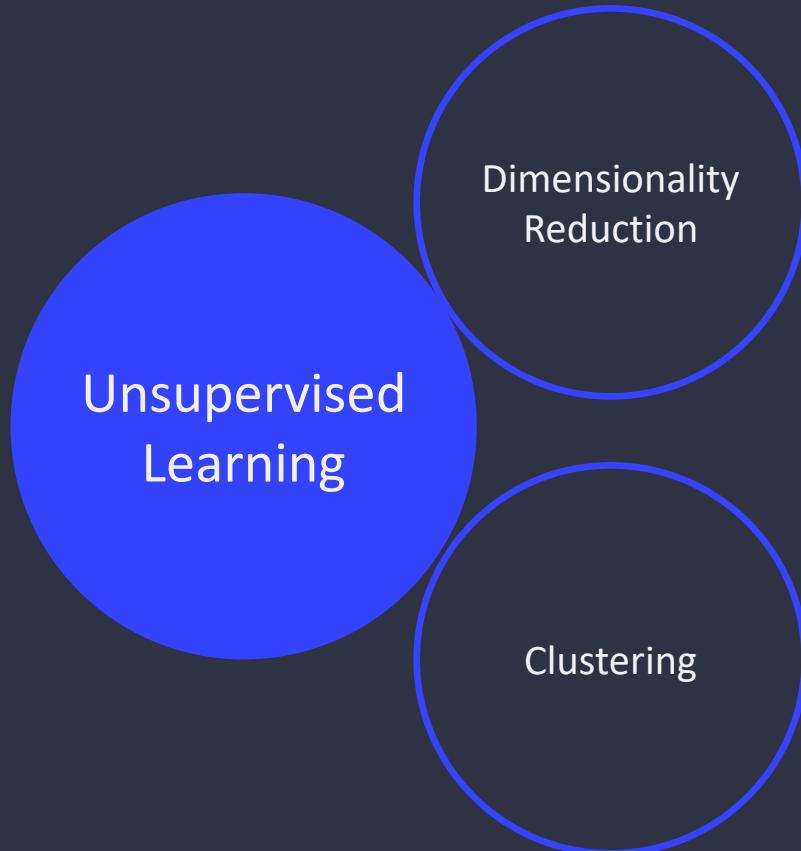
# Dimensionality Reduction



- To learn the mapping between inputs and outputs
- Labelled data (correct answers) is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.

- To learn hidden pattern from a set of inputs (no output).
- Unlabelled data is provided of past input (not a input & output pair) during the learning process. (no correct answers)

# Dimensionality Reduction



is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

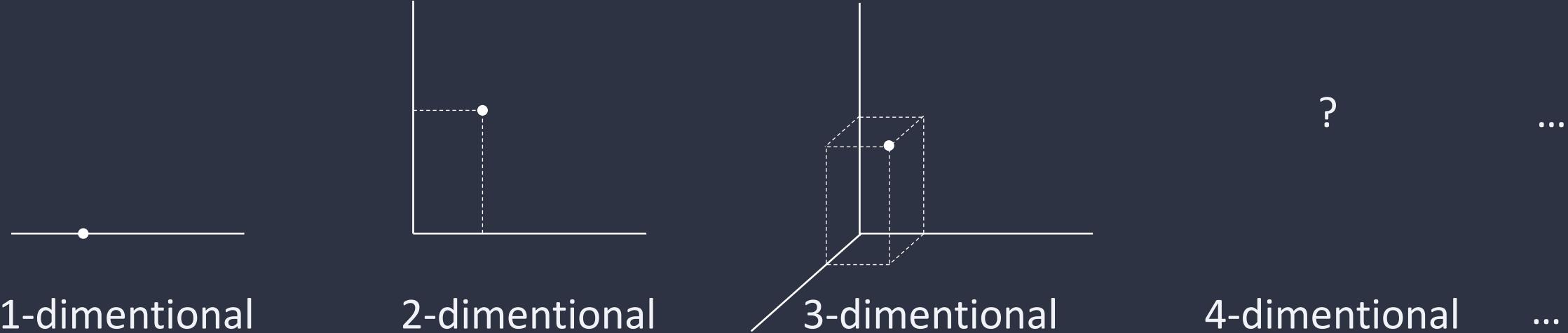
# Dimensionality Reduction

is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

# Dimensionality Reduction - Motivation

## Curse of Dimensionality

refers to various phenomena that arise when analysing and organising data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience.



# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

To learn a “state-of-nature” from a finite number of data samples in a high-dimensional feature space with each feature having a range of possible values.

Typically, an enormous amount of data is required to ensure that there are several samples with each combination of values.

As the number of features or dimensions grows, the amount of data needed to generalise accurately grows exponentially.

# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

EXAMPLE.

Classification: a supervised learning task to organise data points into discrete categories called classes.

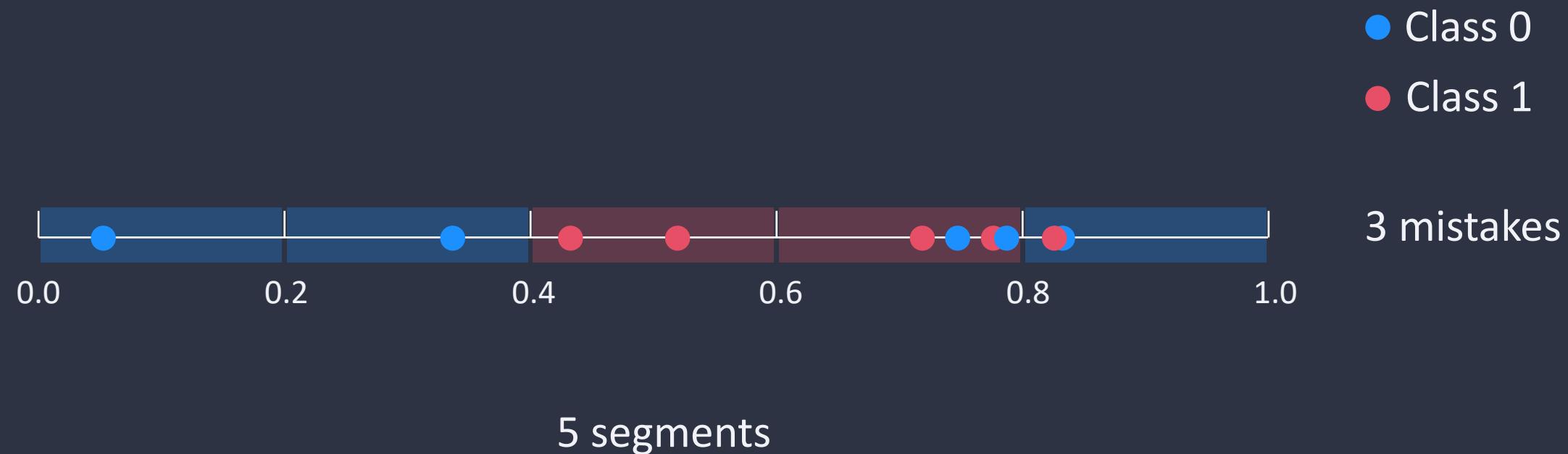
```
import numpy as np
X = np.array([
    [0.33, 0.88, 0.11],
    [0.74, 0.54, 0.62],
    [0.79, 0.07, 0.31],
    [0.83, 0.24, 0.47],
    [0.05, 0.42, 0.47],
    [0.82, 0.70, 0.10],           Input variable
    [0.51, 0.76, 0.51],
    [0.71, 0.92, 0.59],
    [0.78, 0.19, 0.05],
    [0.43, 0.53, 0.53]
])
y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])           Target variable
```

# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

EXAMPLE.

Plot the points using only the first feature:



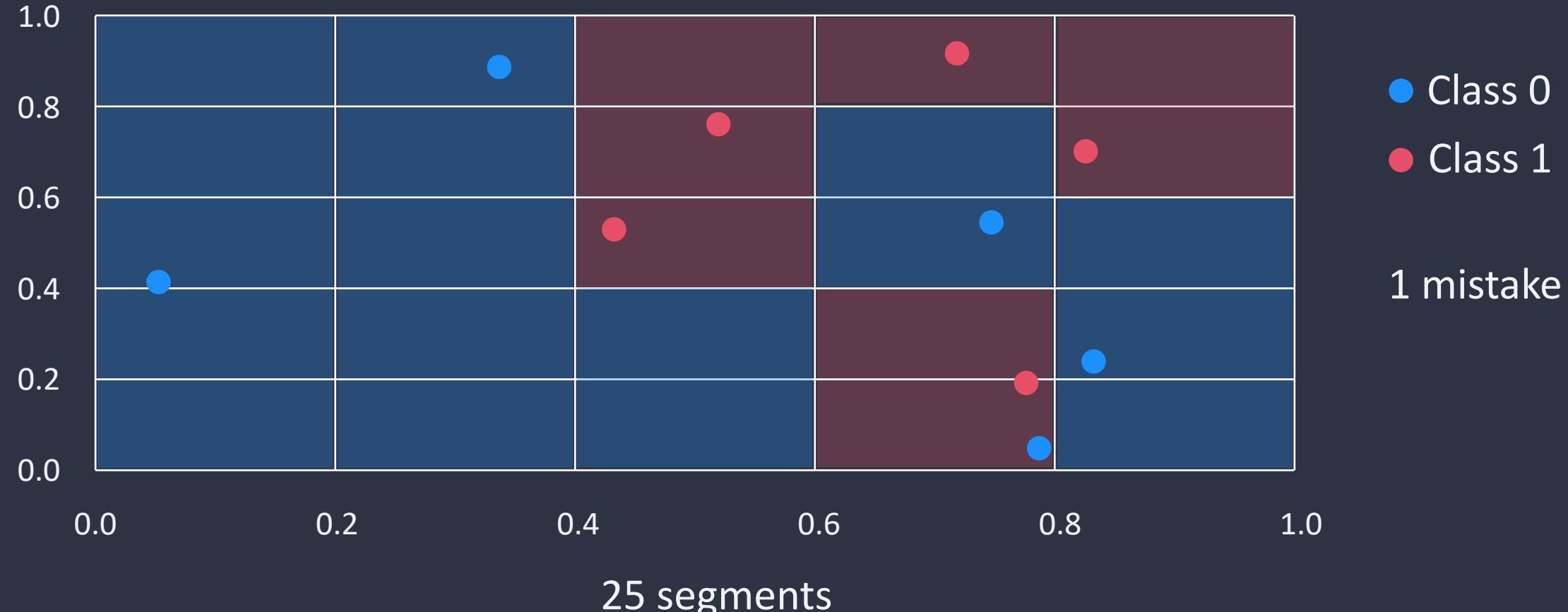
# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

EXAMPLE.

Plot the points using the first two features:

$0.2 \times 0.2$  tiles



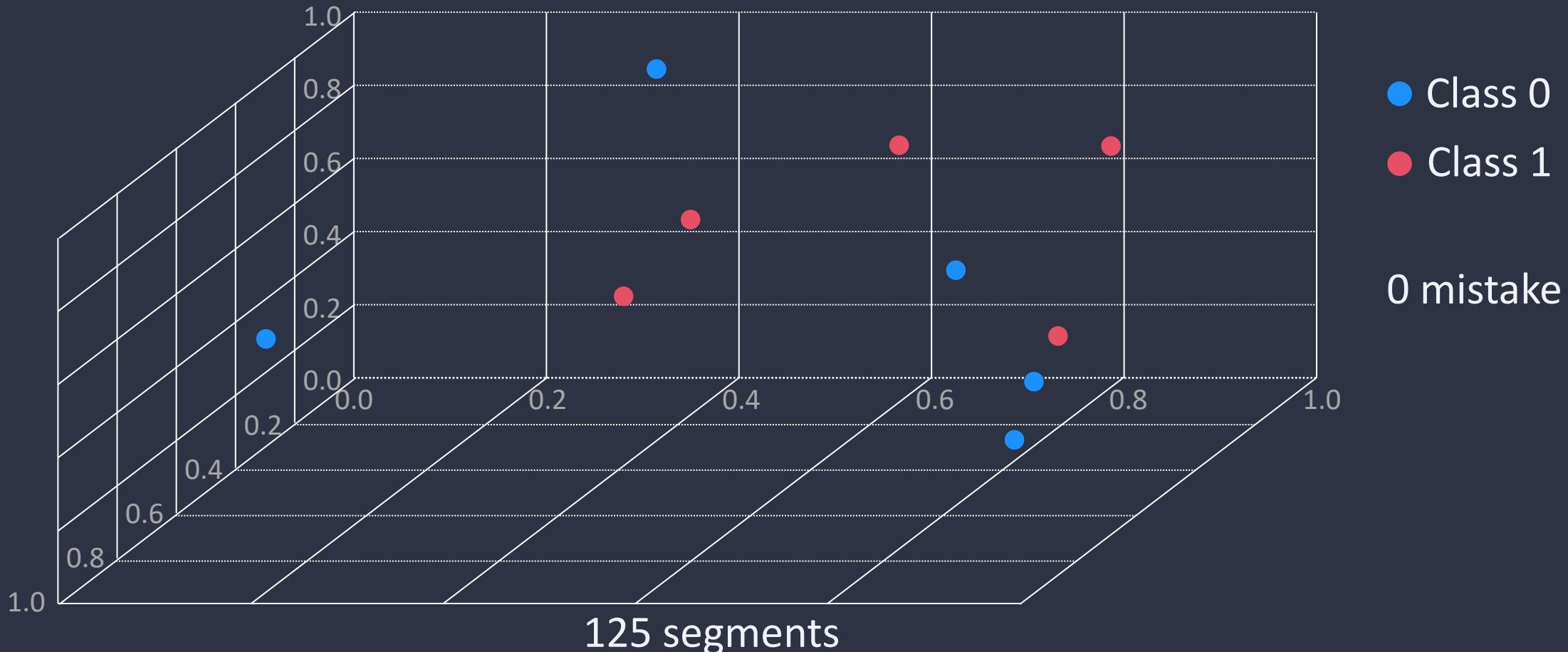
# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

EXAMPLE.

Plot the points using all the three features:

$0.2 \times 0.2 \times 0.2$  cubes



# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

### EXAMPLE.

The issue relates to the proportion of our data points compared to our classification segments:

- with 1 feature: 10 points and 5 sections, i.e.  $10/5=2$  points per segment
- with 2 features:  $10/(5\times5)=0.4$  points per segment
- with 3 features:  $10/(5\times5\times5)=0.08$  points per segment

Adding more features, the available data points in feature space become exponentially sparser, which makes it easier to separate data points. Not because of any pattern in data.

The “pattern” we fit to isn’t actually there at all!

# Dimensionality Reduction - Motivation

## Curse of Dimensionality in Machine Learning

Because of this inherent sparsity we end up **overfitting**, when we add more features to our data, which means we need more data to avoid sparsity — and that's the curse of dimensionality: **as the number of features increase, our data become sparser, which results in overfitting, and we therefore need more data to avoid it.**

## Cure for the Curse

1. Adding more training data points (not always possible...)
2. Reducing the dimension of the data

# Principal Component Analysis

# Principal Component Analysis (PCA)

- Also known as the Karhunen-Loeve Transform
- To search for patterns in high-dimensional data.
- To explore and visualise high-dimensional data.
- To compress data and process data before it is used by another estimator.
- Reduces a set of possibly-correlated, high-dimensional variables to a lower-dimensional set of linearly uncorrelated synthetic variables, called **principle components**.
- The lower-dimensional data will **preserve** as much of the **variance** of the original data as possible.

# Principal Component Analysis (PCA)

- PCA reduces the dimensions of a data set by projecting the data onto a lower-dimensional subspace.

E.g., 2-dimensional data -> onto a line

(an instance is represented by a single value rather than a pair of values.)

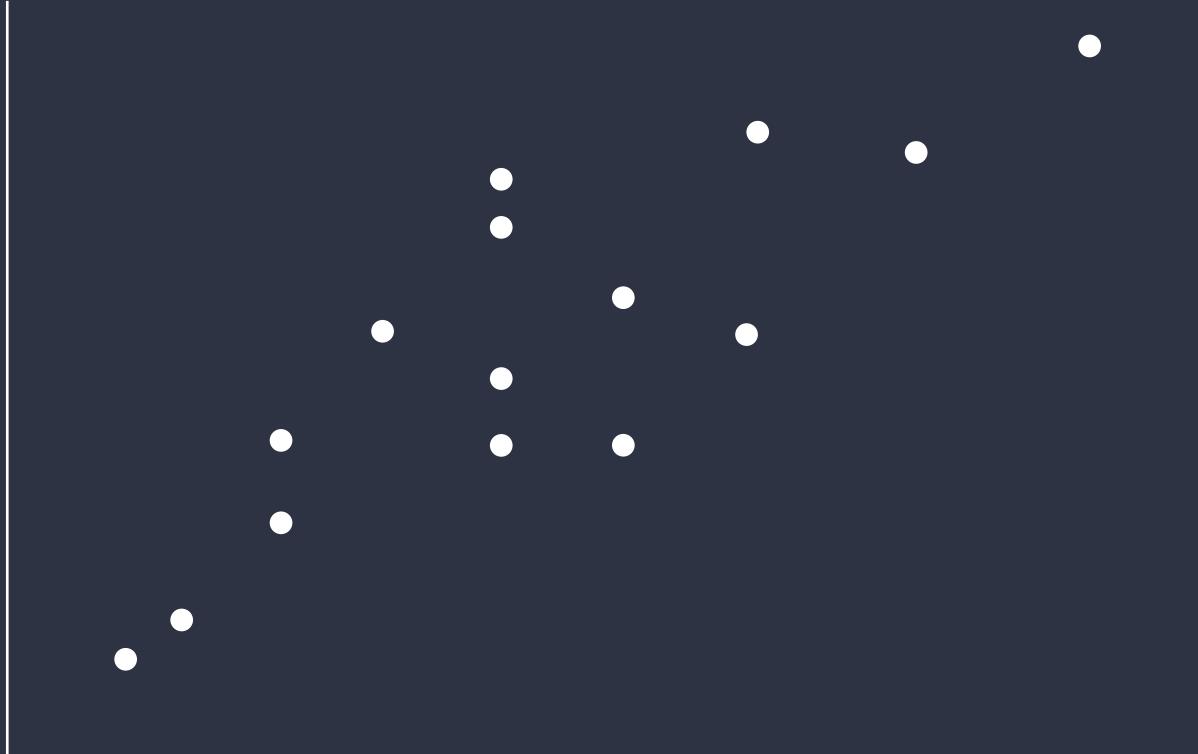
3-dimensional data -> onto a plane

(an instance is represented by a pair of values rather than a triple (3 values).)

- In general, an n-dimensional dataset can be reduced by projecting the dataset onto k-dimensional subspace, where k is less than n.

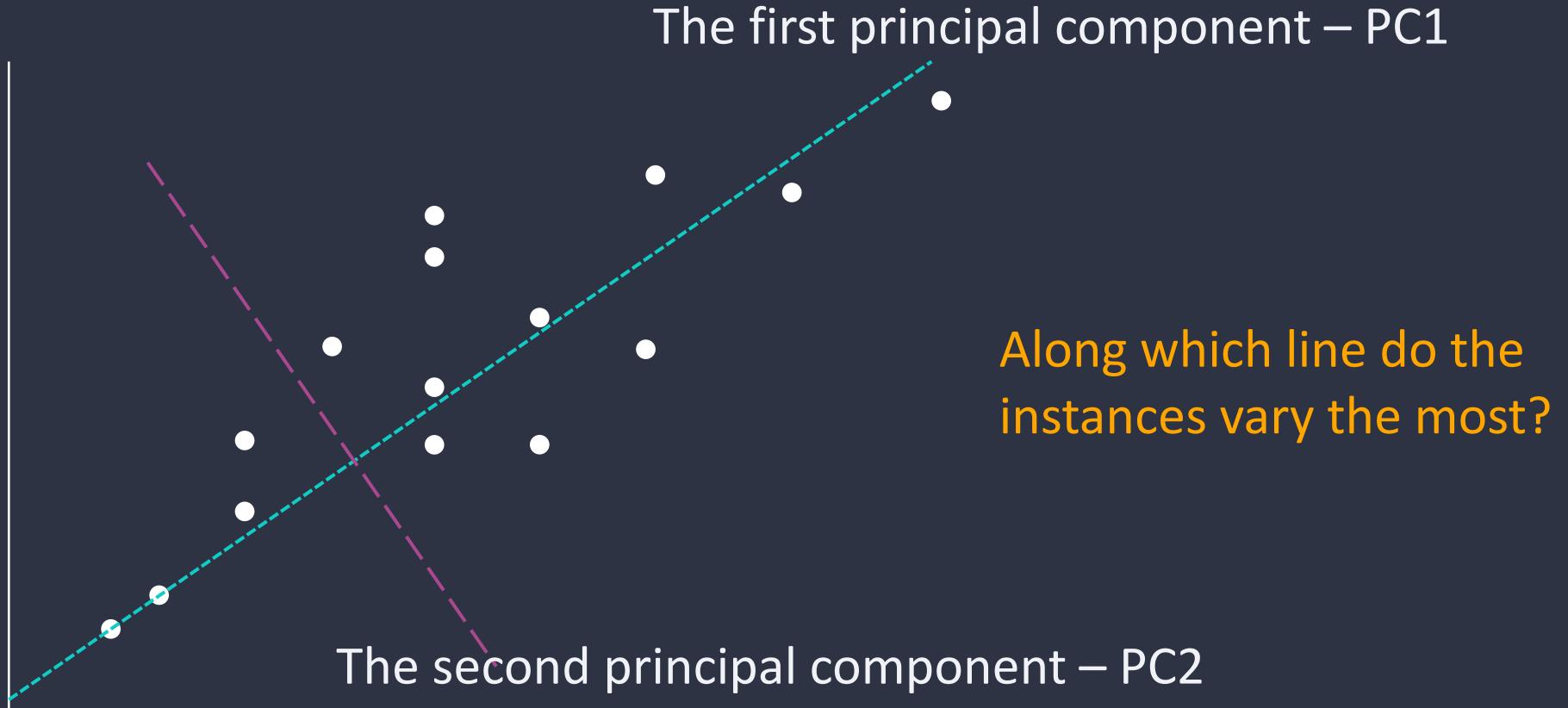
# Principal Component Analysis (PCA)

## Intuition



# Principal Component Analysis (PCA)

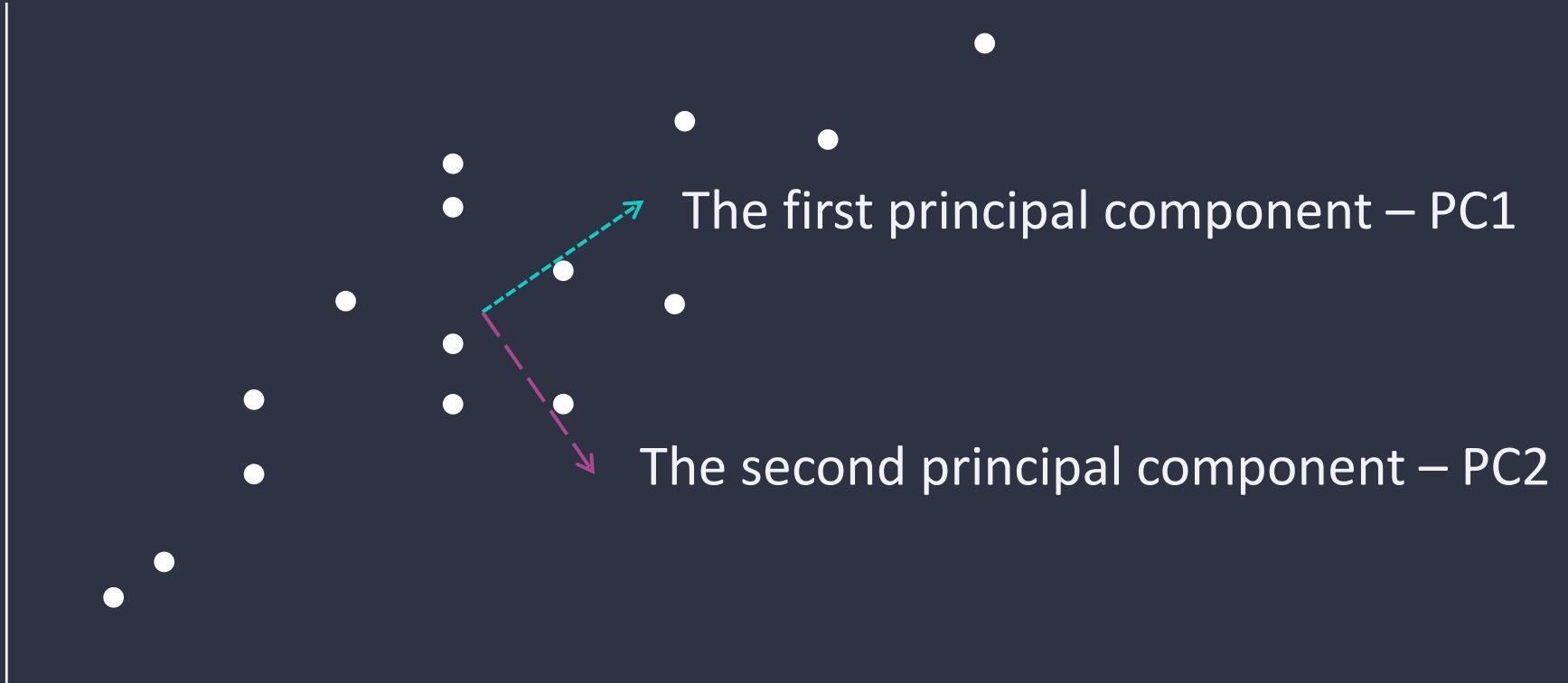
## Intuition



PC2 must be statistically independent, and will appear to be perpendicular to PC1 when it is plotted

# Principal Component Analysis (PCA)

## Intuition



Each subsequent principal component preserves the maximum amount of the remaining variance; the only constraint is that each must be orthogonal to the other principal components.

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

**Variance ( $\sigma_X^2$ ):** a measure of how a set of values are spread out.

$$\sigma_X^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

$n$ : the number of instances

$\bar{X}$ : the mean of the variable  $X$  (represented as a vector)

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

**Covariance**  $\sigma(X, Y)$ : a measure of how much two variables ( $X$  and  $Y$ ) change together (the strength of the correlation between two sets of variables).

$$\sigma(X, Y) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})$$

- If the covariance of two variable is zero, the variables are uncorrelated.
- If the covariance is nonzero, the sign indicates whether the variables are positively or negatively correlated.
- The variance of  $\sigma_X^2$  of a variable  $X$  can be also expressed as the covariance with itself by  $\sigma(X, X)$ .

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

**Covariance Matrix** (a square matrix  $\mathcal{C}$ ): describes the covariance values between each pair of dimensions (features) in a data set.

$$\mathcal{C} = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^T \quad \mathcal{C} = \begin{bmatrix} \sigma(x_1, x_1) & \sigma(x_1, x_2) & \sigma(x_1, x_3) \\ \sigma(x_2, x_1) & \sigma(x_2, x_2) & \sigma(x_2, x_3) \\ \sigma(x_3, x_1) & \sigma(x_3, x_2) & \sigma(x_3, x_3) \end{bmatrix}$$

- $\mathcal{C}_{i,j} = \sigma(x_i, x_j)$ , where  $\mathcal{C} \in \mathbb{R}^{n \times n}$  and  $n$  describes the dimension of the data
- $\mathcal{C}$  is symmetric, as  $\sigma(x_i, x_j) = \sigma(x_j, x_i)$ .
- The diagonal entries of  $\mathcal{C}$  are the variances; the other entries are the covariances.

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

EXAMPLE.

Calculate the covariance matrix for the following data set:

2.15	-5.43	9.12
2.49	-4.83	8.76
5.28	-2.34	2.48
4.96	-1.99	2.52

mean	3.72	-3.65	5.72
------	------	-------	------

$$C = \begin{bmatrix} 1.987 & 2.087 & -4.525 \\ 2.087 & 2.258 & -4.799 \\ -4.525 & -4.799 & 10.385 \end{bmatrix}$$

```
import numpy as numpy
X = [[2.15,-5.43,9.12]
      [2.49,-4.83,8.76]
      [5.28,-2.34,2.48]
      [4.96,-1.99,2.52]]

print np.cov(np.array(X).T)
```

```
[[1.987,2.087,-4.525]
 [2.087,2.258,-4.799]
 [-4.525,-4.799,10.385]]
```

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

A vector is described by a direction and magnitude (or length).

An **eigenvector** of a matrix is a non-zero vector that satisfies:

$\vec{v}$ : an eigenvector

$A\vec{v} = \lambda\vec{v}$        $A$ : a *square* matrix

$\lambda$ : a scalar, called **eigenvalue**

- The direction of  $\vec{v}$  remains the same after it has been transformed by  $A$ .
- Only its magnitude has changed, as indicated by  $\lambda$ , i.e., multiplying a matrix by one of its eigenvectors is equal to scaling the eigenvector.

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

$$\mathbf{A}\vec{v} = \lambda\vec{v}$$

$$\mathbf{A}\vec{v} - \lambda\vec{v} = 0$$

$$\mathbf{A}\vec{v} - \lambda\mathbf{I}\vec{v} = 0$$

$$(\mathbf{A} - \lambda\mathbf{I})\vec{v} = 0$$

If  $\vec{v}$  is nonzero, this equation will only have a solution if

$$\mathbf{A} - \lambda\mathbf{I} = 0$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE. Calculate the eigenvectors and eigenvalues of the following matrix:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$|\mathbf{A} - \lambda\mathbf{I}| = \left| \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

$$\left| \begin{bmatrix} -\lambda & 1 \\ -2 & -3 - \lambda \end{bmatrix} \right| = \lambda^2 + 3\lambda + 2 = 0$$

The two eigenvalues:  $\lambda_1 = -1$      $\lambda_2 = -2$

All that's left is to find the two eigenvectors.

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

The two eigenvalues:  $\lambda_1 = -1$      $\lambda_2 = -2$

$$\begin{aligned} (A - \lambda I)\vec{v} &= 0 \\ \left( \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right) \vec{v} &= 0 \end{aligned}$$

$\xrightarrow{\quad} \begin{bmatrix} -\lambda & 1 \\ -2 & -3 - \lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$

$$\left\{ \begin{array}{l} -\lambda v_1 + v_2 = 0 \\ -2v_1 - (3 + \lambda)v_2 = 0 \end{array} \right\}$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$\begin{cases} -\lambda v_1 + v_2 = 0 \\ -2v_1 - (3 + \lambda)v_2 = 0 \end{cases}$$

$$\text{when } \lambda_1 = -1$$

$$\begin{cases} v_1 + v_2 = 0 \\ -2v_1 - 2v_2 = 0 \end{cases}$$

$$v_1 + v_2 = 0$$

$$v_1 = -v_2$$

$$\text{when } \lambda_2 = -2$$

$$\begin{cases} 2v_1 + v_2 = 0 \\ -2v_1 - v_2 = 0 \end{cases}$$

$$2v_1 + v_2 = 0$$

$$2v_1 = -v_2$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

when  $\lambda_1 = -1$   
( $v_1 = -v_2$ )

when  $\lambda_2 = -2$   
( $2v_1 = -v_2$ )

The corresponding eigenvector:  $\vec{v}_1 = k_1 \begin{bmatrix} +1 \\ -1 \end{bmatrix}$        $\vec{v}_2 = k_2 \begin{bmatrix} +1 \\ -2 \end{bmatrix}$

( $k_1$  and  $k_2$  are arbitrary constants  
i.e., only their ratio is important.)

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

when  $\lambda_1 = -1$   
( $v_1 = -v_2$ )

when  $\lambda_2 = -2$   
( $2v_1 = -v_2$ )

The corresponding eigenvector:  $\vec{v}_1 = k_1 \begin{bmatrix} +1 \\ -1 \end{bmatrix}$

$$\vec{v}_2 = k_2 \begin{bmatrix} +1 \\ -2 \end{bmatrix}$$

PCA requires unit eigenvectors, or eigenvectors that have a length equal to 1.

We normalise an eigenvector by dividing it by its norm  $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$

So,  $\vec{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

$$\vec{v}_2 = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

EXAMPLE.

Calculate the eigenvectors and eigenvalues of the following matrix:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

Solution:

$$\text{Eigenvalues: } \lambda_1 = -1 \quad \lambda_2 = -2$$

$$\text{Eigenvectors: } \vec{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \quad \vec{v}_2 = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$$

# Principal Component Analysis (PCA)

## Performing PCA

Recall from Linear Algebra

### Eigenvectors and Eigenvalues

- Eigenvectors and eigenvalues can only be derived from square matrices, and not all square matrices have eigenvectors or eigenvalues.
- If a matrix does have eigenvectors and eigenvalues, it will have a pair for each of its **covariance matrix**, ordered by their corresponding **eigenvalues**.
- The eigenvector with the greatest eigenvalue is the **first Principle Component (PC)**; the eigenvector with the second greatest eigenvalue is the **second PC**, etc.
- While reducing the dimension of the data, we eliminate the eigenvectors which have near-zero eigenvalues since the variance explained by them is very low.

# Principal Component Analysis (PCA)

## Performing PCA

### Steps to calculate PCA

- Subtract mean from all the observations, so our data is zero centred.
- Calculate the covariance matrix.
- Next, we calculate the eigenvalues and eigenvectors of the covariance matrix.
- Now use the top k eigenvectors (sorted by descending order of eigenvalues) to construct a projection matrix.
- Multiply the original data with the projection matrix to get the new k-dimensional subspace.

# Principal Component Analysis (PCA)

## Performing PCA

### Steps to calculate PCA

EXAMPLE.

Principal Component Analysis in python from scratch using numpy

Import the iris dataset

```
from sklearn import datasets
iris = datasets.load_iris()
X=pd.DataFrame(iris['data'],columns=iris['feature_names'])
```

Subtract the mean from all the instances so that the data will be zero centred.

```
import numpy as np
M = np.mean(X.T, axis=1)
X_scaled = X - M
```

# Principal Component Analysis (PCA)

## Performing PCA

Steps to calculate PCA

EXAMPLE.

Principal Component Analysis in python from scratch using numpy

Construct the covariance matrix and calculate it's eigenvalues and eigenvectors

```
from numpy.linalg import eig
S = np.cov(X_scaled.T)
eig_vals, eig_vecs = eig(S)
```

Sort the eigenpairs in descending order of eigenvalues so that the top eigenvalues will explain the maximum variance.

```
sort_index = eig_vals.argsort()[:-1]
values_sorted = eig_vals[sort_index]
vectors_sorted = eig_vecs[:, sort_index]
```

# Principal Component Analysis (PCA)

## Performing PCA

### Steps to calculate PCA

EXAMPLE.

Principal Component Analysis in python from scratch using numpy

Print the variance explained by each of the eigenvalues

```
explained_variance_ratio = values_sorted/values_sorted.sum()  
print(explained_variance_ratio)
```

OUTPUT:

```
[ 0.92461621  0.05301557  0.01718514  0.00518309]
```

The first two principal components explain more than 95% (the most) of the variance in the data.

# Principal Component Analysis (PCA)

## Performing PCA

Steps to calculate PCA

EXAMPLE.

Principal Component Analysis in python from scratch using numpy

Use the first 2 principal components to construct the projection matrix (p).

```
p = np.hstack((vectors_sorted[0][:,np.newaxis], vectors_sorted[1][:,np.newaxis]))
```

Multiply the original data with the projection matrix, to transform the original data onto the lower-dimensional subspace

```
X_pca = X_scaled.dot(p)
print("Original size of the data:",X_scaled.shape)
print("After applying PCA:",X_pca.shape)
```

```
Original size of the data: (150, 4)
After applying PCA: (150, 2)
```

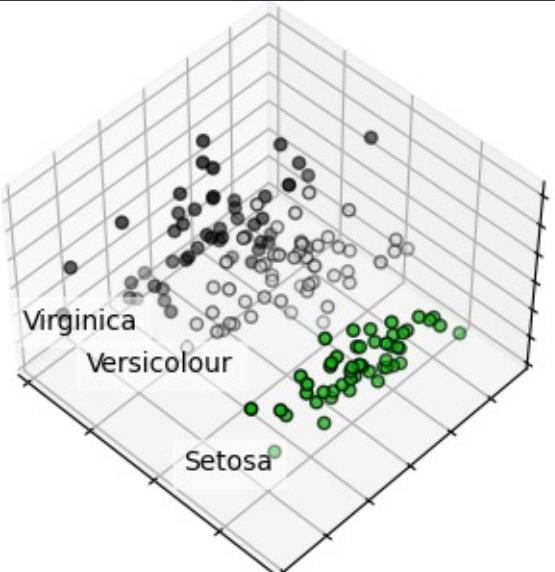
The size of the data is reduced from (150,4) to (150,2) while preserving most of the information about the data.

# Principal Component A

## Performing PCA

EXAMPLE.

With sklearn



```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import decomposition
from sklearn import datasets

np.random.seed(5)

iris = datasets.load_iris()
X = iris.data
y = iris.target

fig = plt.figure(1, figsize=(4, 3))
plt.clf()

ax = fig.add_subplot(111, projection="3d", elev=48, azim=134)
ax.set_position([0, 0, 0.95, 1])

plt.cla()
pca = decomposition.PCA(n_components=3)
pca.fit(X)
X = pca.transform(X)

for name, label in [("Setosa", 0), ("Versicolour", 1), ("Virginica", 2)]:
    ax.text3D(
        X[y == label, 0].mean(),
        X[y == label, 1].mean() + 1.5,
        X[y == label, 2].mean(),
        name,
        horizontalalignment="center",
        bbox=dict(alpha=0.5, edgecolor="w", facecolor="w"),
    )
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(float)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.nipy_spectral, edgecolor="k")

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])

plt.show()
```

# Summary

1. Dimensionality Reduction
2. Principal Component Analysis (PCA)

Next Lecture

Cross Validation and Hyperparameter Tuning