# Module 4: Building a Simple ML Model

- ML pipeline: problem definition, algorithm selection, training, evaluation
- Overfitting/underfitting, model validation basics
- 📌 Live Demo: Step-by-step model building in Colab using a simple dataset

**IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System**
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

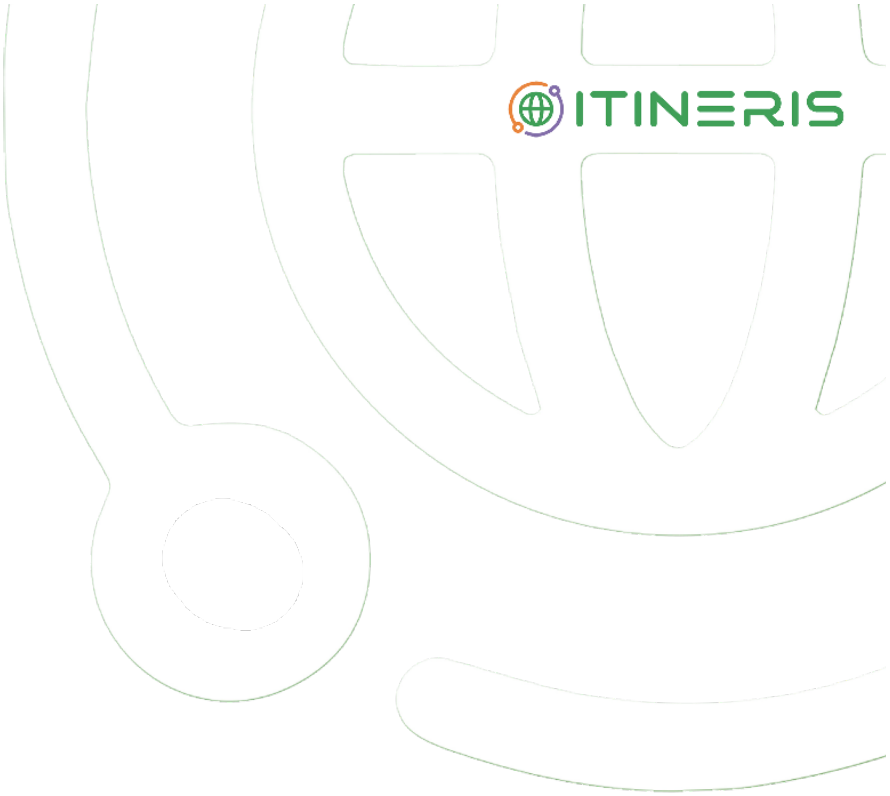
# Link

- https://drive.google.com/drive/folders/1co_PgxnccqMLkXadfZtHjpF5cBbJR6ms?usp=sharing
- Link … https://shorturl.at/5Blm2

# Pattern recognition

- The human brain… the most powerful pattern recognition machine?

# What’s a “chair”?



# What’s a “chair”?

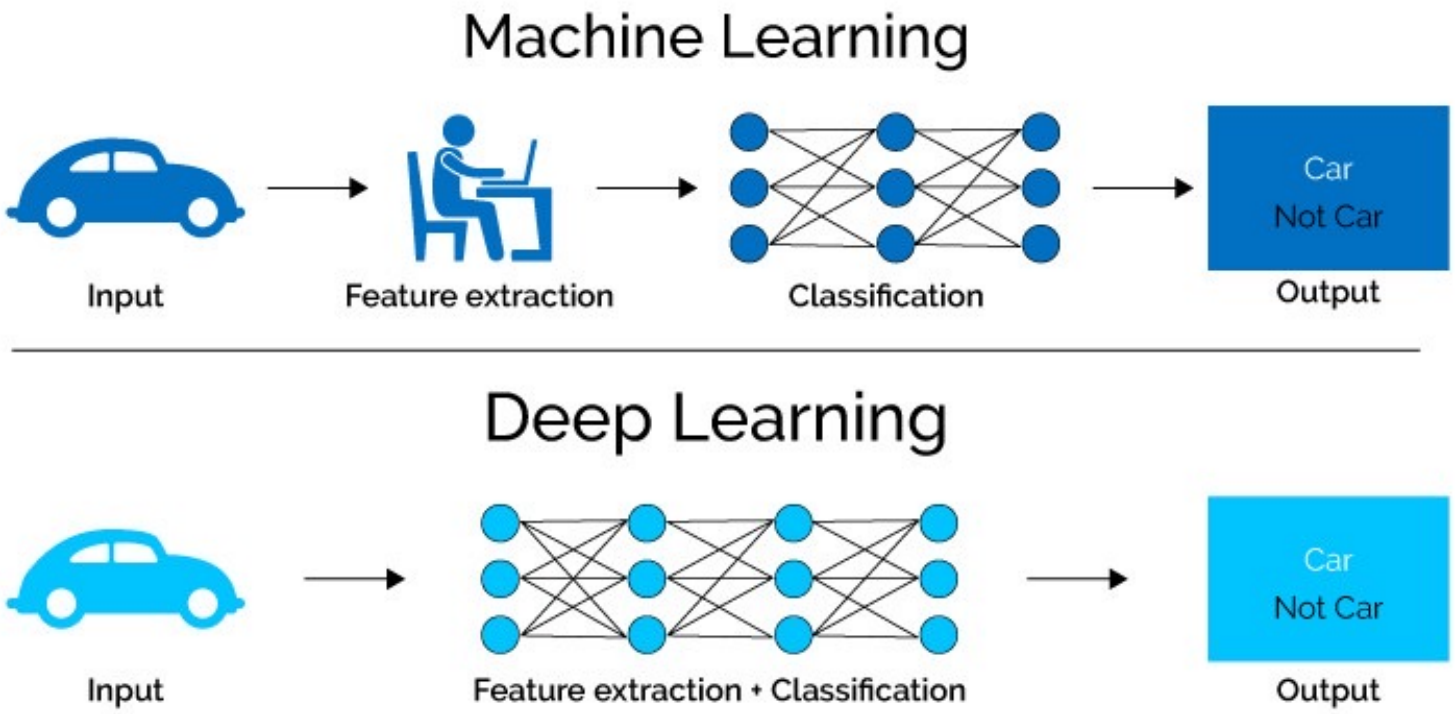
# What’s a “chair”?

# What’s a “chair”?

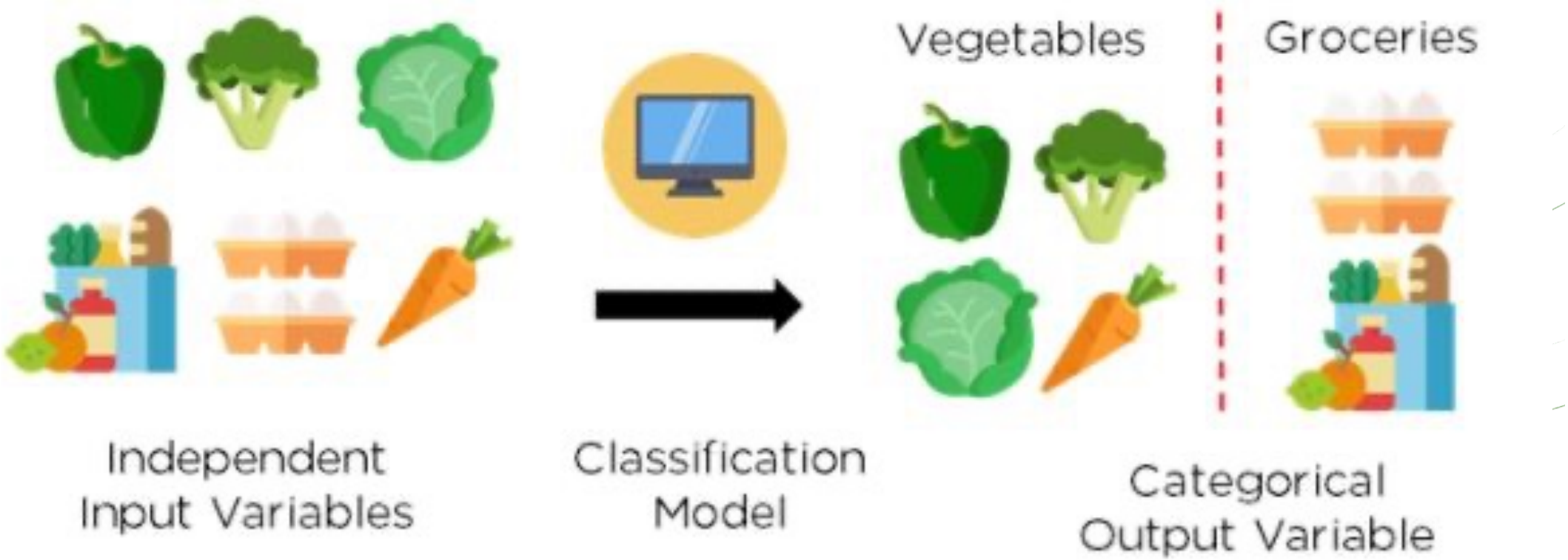
# What’s a “chair”?




# Fully automated learning




Vegetables
Groceries

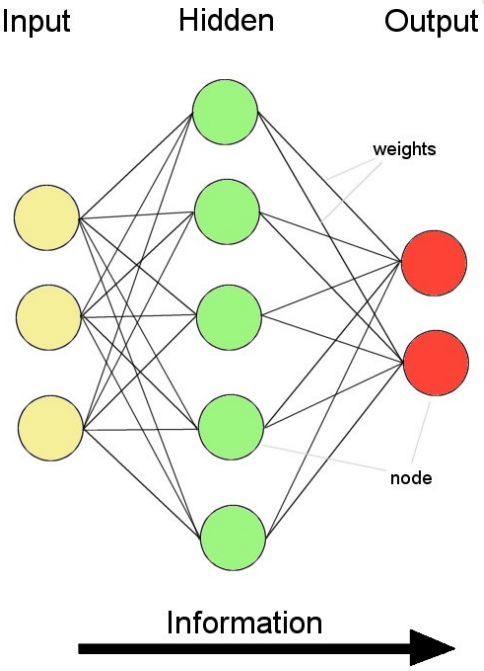

Vegetables
Groceries
Independent
Input Variables
Classification
Model
Categorical
Output Variable

# Some questions

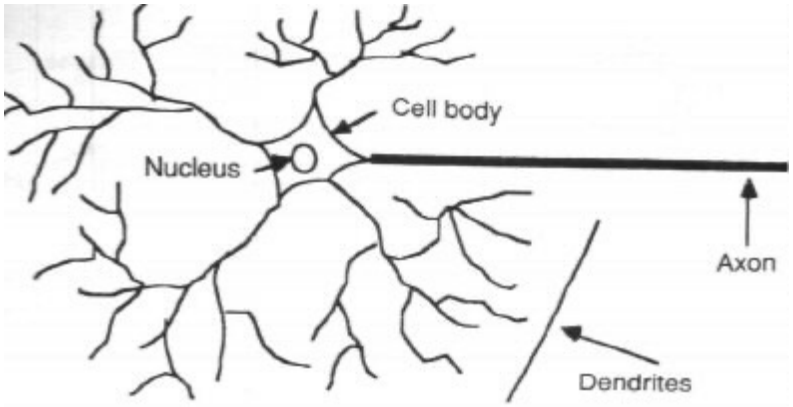
- What is a neural network?
- How does it work and why now?
- Why is it generally better than other methods on image, speech and certain other types of data?

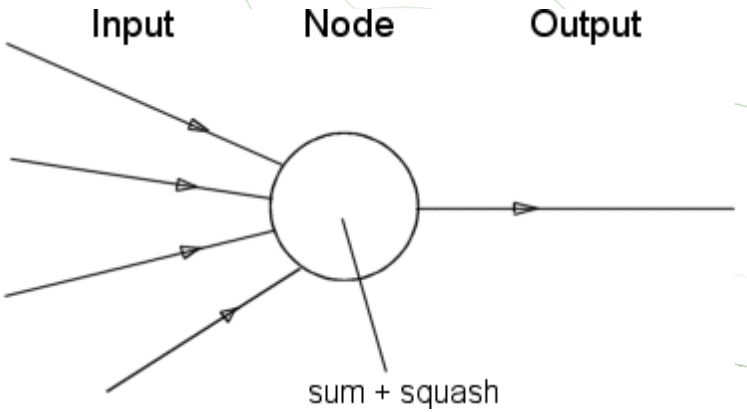
# Artificial Neural Networks (ANNs)

- ANNs incorporate the two fundamental components of biological neural nets:
  - Neurones (nodes)
  - Synapses (weights)

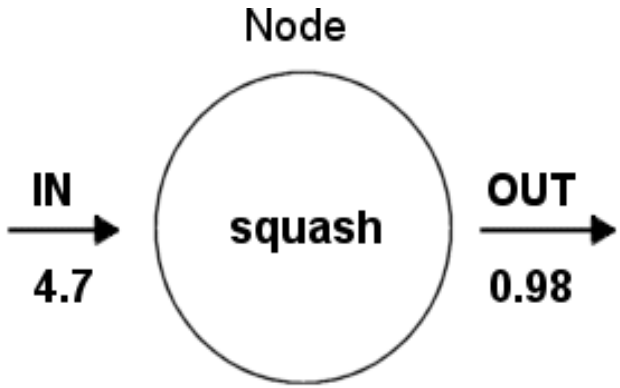


# Neuron vs. Nodes

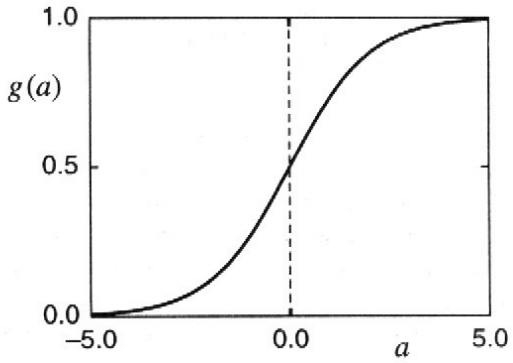


# Structure of a node



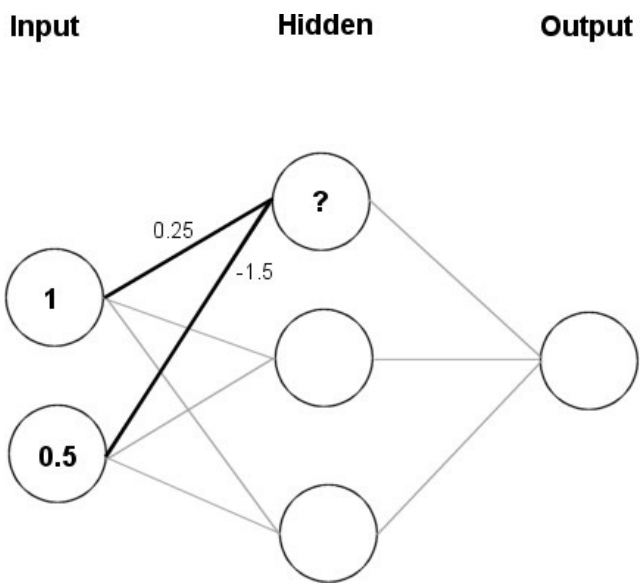

Squashing function limits node output:





132

# Feeding data through the net




$(1 \times 0.25) + (0.5 \times (-1.5)) =$

$0.25 + (-0.75) = -\mathbf{0.5}$

Squashing:

$$\frac{1}{1+e^{0.5}} = 0.3775$$

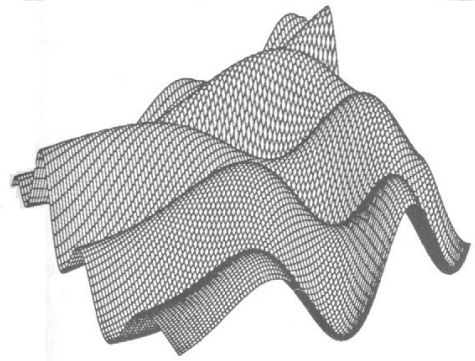
# Weight settings determine the behaviour of a network

→ How can we find the right weights?


134

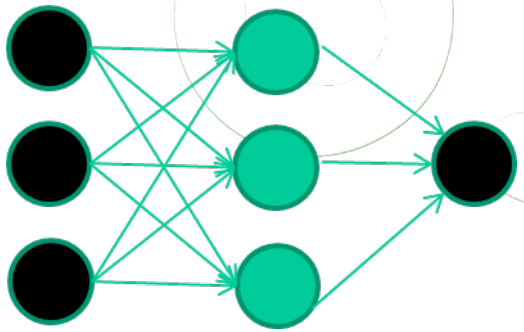
# Training – Backpropagation

- The Learning Phase
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Error is used to adjust weights (supervised learning)
- -> Gradient descent on error landscape



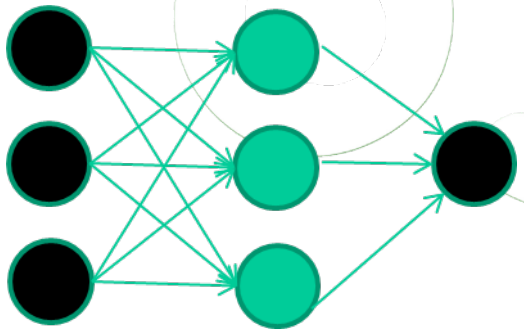
# A dataset

| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training the neural network

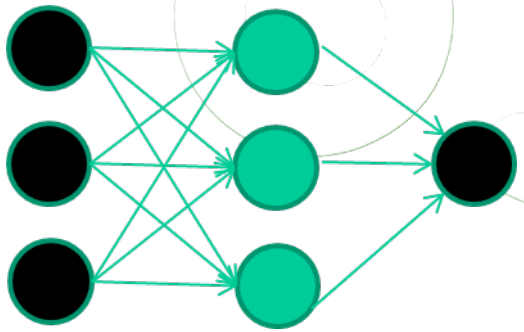
| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training data

- Initialise with random weights

| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |

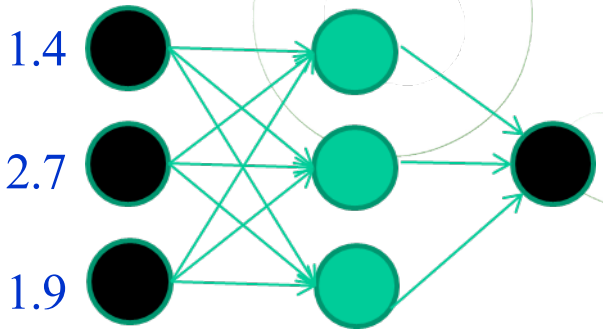



138

# Training data

- Present a training pattern

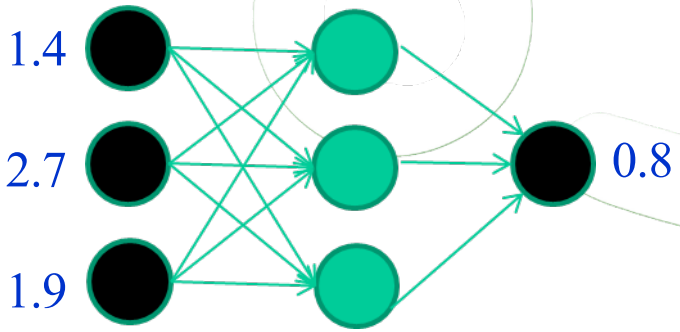
| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training data

- Feed it through to get output

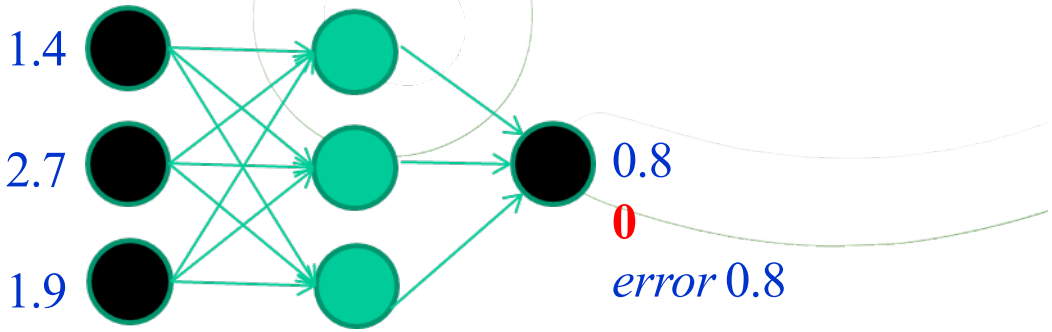
| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training data

- Compare with target output

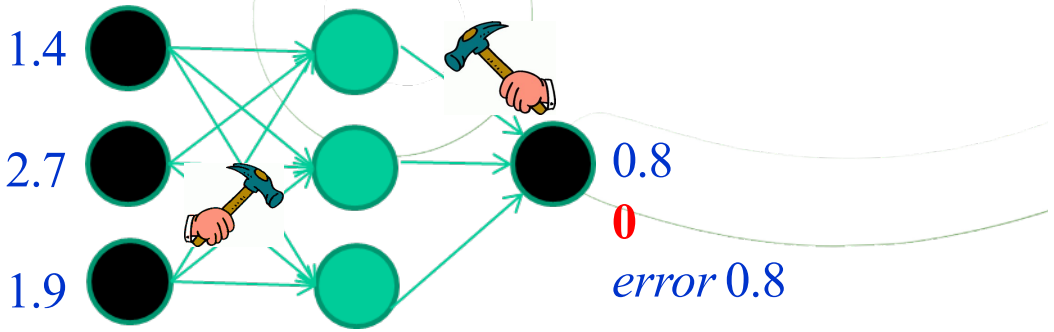
| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training data

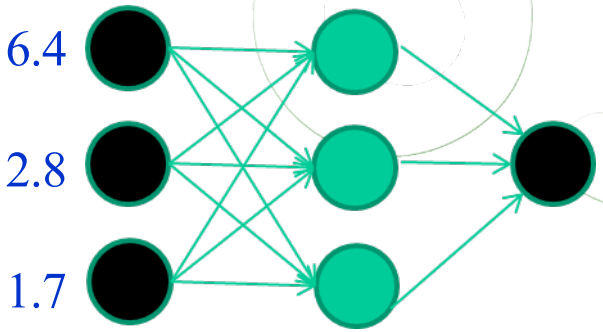
- Adjust weights based on error

| Fields | | | class |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



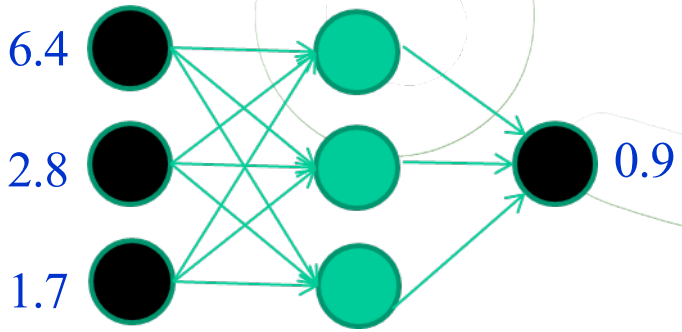
# Training data

| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



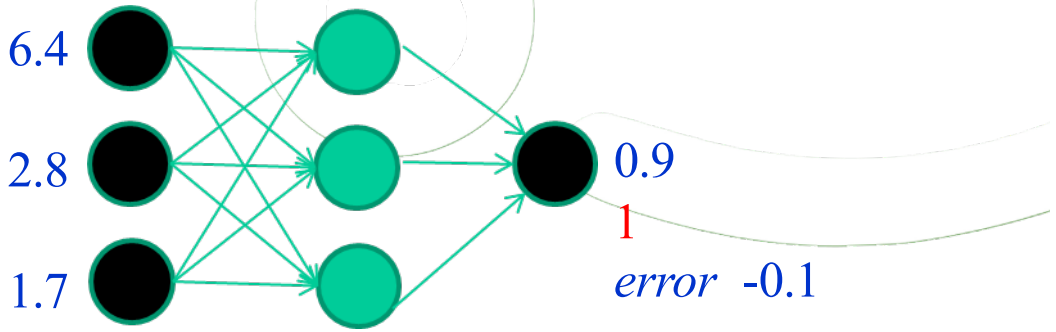
# Feed it through to get output

| Fields | | | class |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



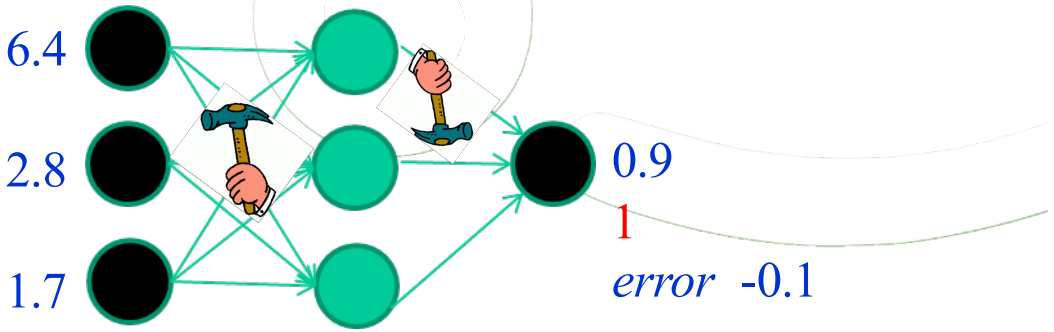
# Training data

| *Fields* | | | *class* |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |



# Training data

| Fields | | | class |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |

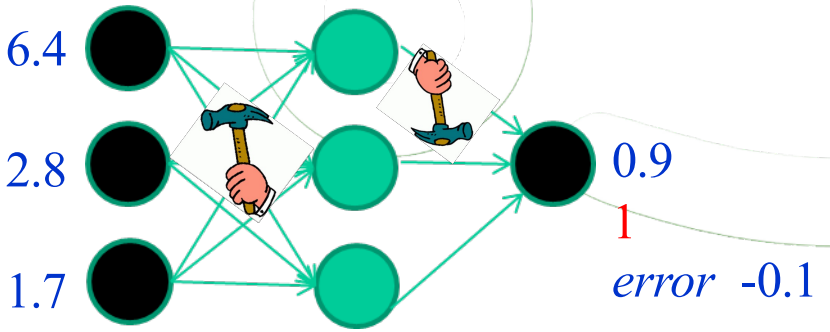


# Training data

- Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

*Training data*

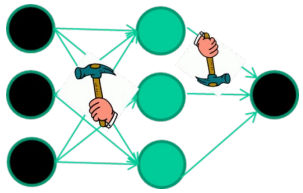
| Fields | | | class |
|---|---|---|---|
| 1.4 | 2.7 | 1.9 | 0 |
| 3.8 | 3.4 | 3.2 | 0 |
| 6.4 | 2.8 | 1.7 | 1 |
| 4.1 | 0.1 | 0.2 | 0 |
| etc … | | | |




**And so on ….**

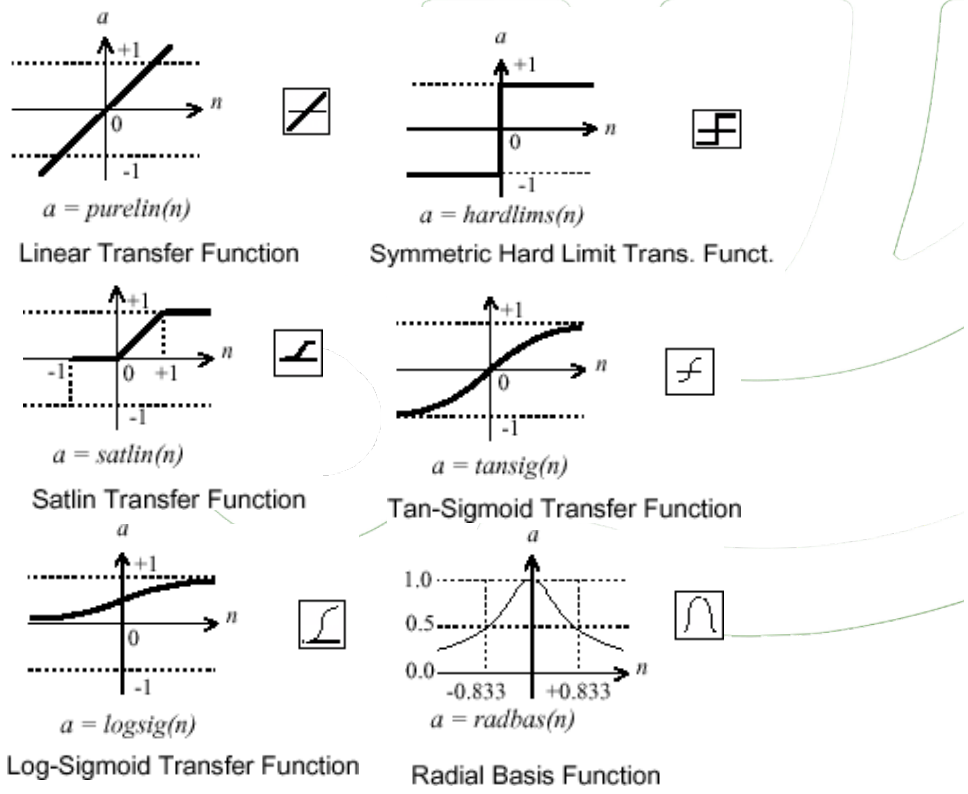
# Dumbness and volume

- Weight-learning algorithms for ANNs are dumb
- They work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others
- But, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications



# Activation functions

- The activation function is generally non-linear.
- Linear functions are limited because the output is simply proportional to the input.



# Some other points

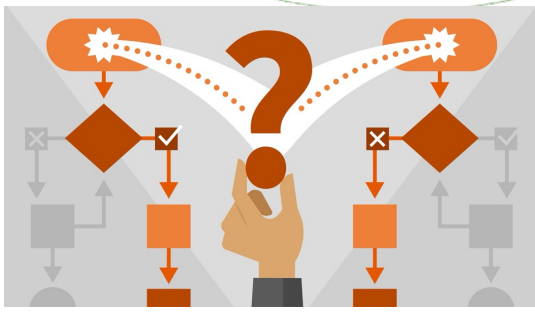
- If f(x) is non-linear, a network with 1 hidden layer can, in theory, learn perfectly any classification problem.
- A set of weights exists that can produce the targets from the inputs.
- The problem is finding them.

# GOAL and MEANS

- You must choose the right method
- You must run statistical validity tests

# The basic of “Modeling”

- We do it naturally in life, maybe without knowing it
  - Basic task for ”data miners”
  - Statisticians have been doing it for many years
  - It takes many different form
  - Today, all managers should have at least a basic understanding



# Modeling: A simple “supervised” example

| Age | Income | **Out** |
|---|---|---|
| 30 | 65k | Y |
| 68 | 83k | Y |
| 43 | 61k | N |
| 30 | 25k | Y |
| 51 | 82k | N |
| 78 | 67k | Y |

# Modeling: A simple “supervised” example

| Age | Income | **Out** |
|---|---|---|
| 30 | 65k | Y |
| 68 | 83k | Y |
| 43 | 61k | N |
| 30 | 25k | Y |
| 51 | 82k | N |
| 78 | 67k | Y |

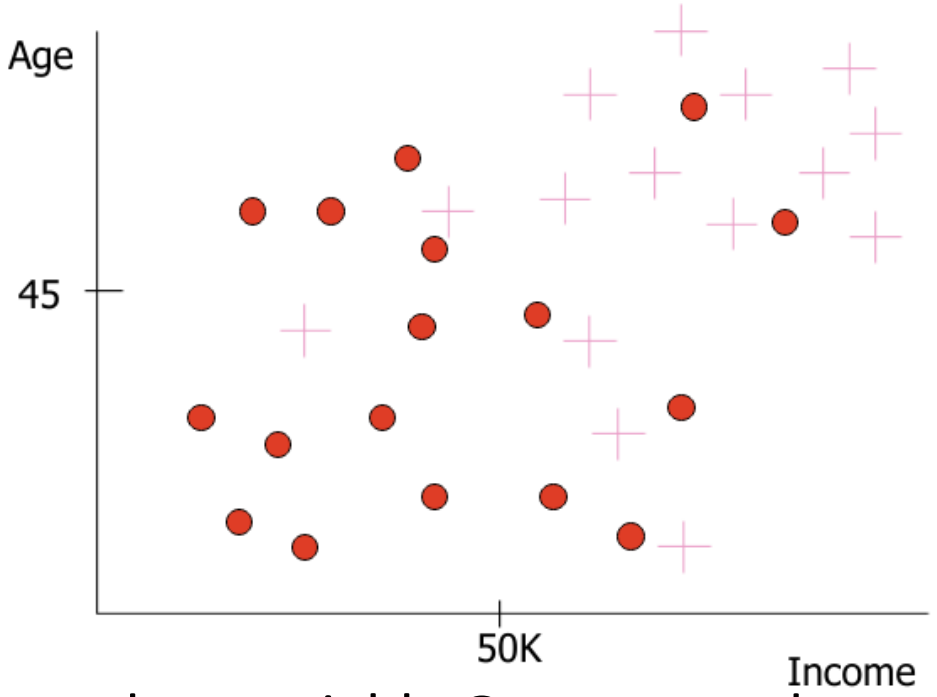
- Goal: Build a model to predict the dependent variable Outcome

# Modeling: A simple “supervised” example

| Age | Income | **Out** |
|---|---|---|
| 30 | 65k | Y |
| 68 | 83k | Y |
| 43 | 61k | N |
| 30 | 25k | Y |
| 51 | 82k | N |
| 78 | 67k | Y |

- **Out** is a categorical variable
- *Age* and *Income* are the independent variables

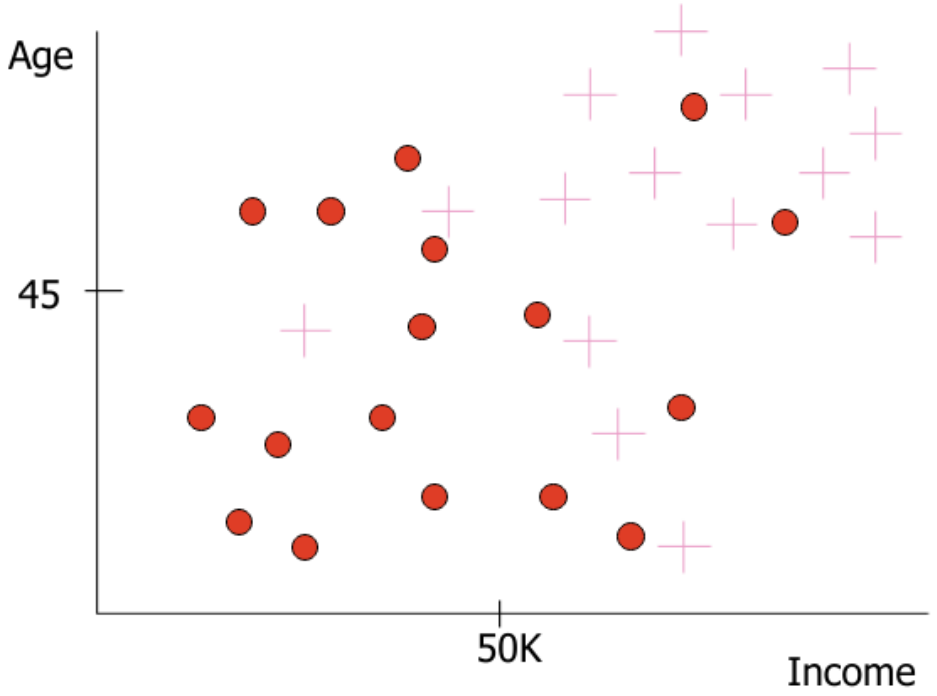
# Let's model



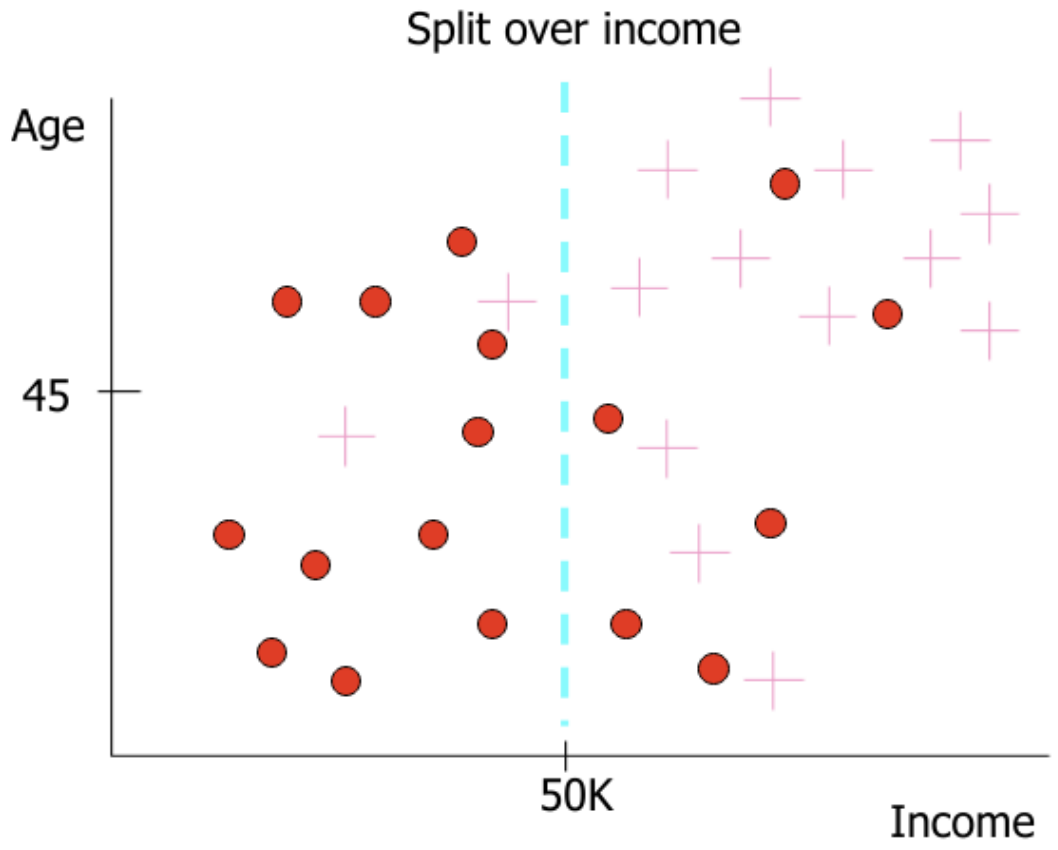

- We project the dependent variable **Out**, as + and ∘, on a two-dimensional space

# Let’s model

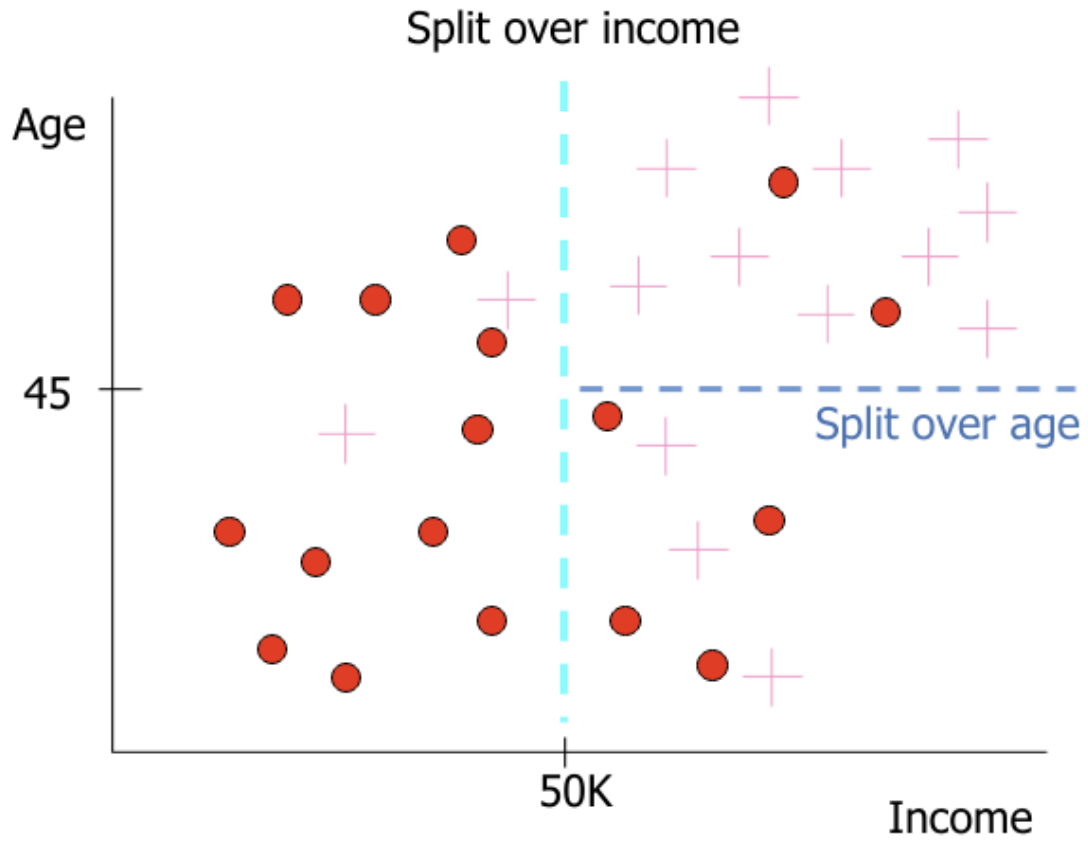
- Do we notice anything?



# Let’s model

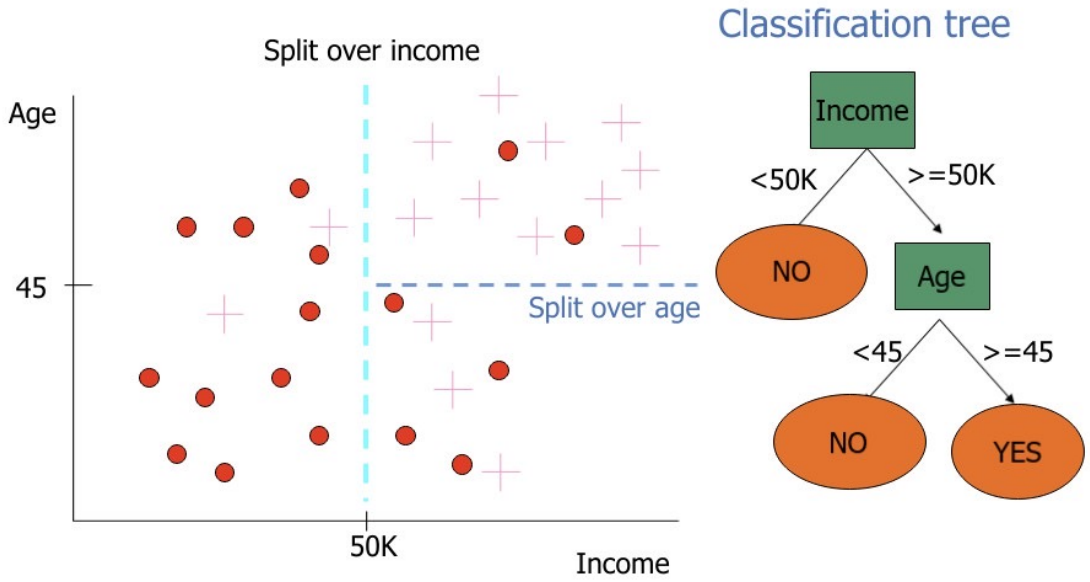


# Let’s model



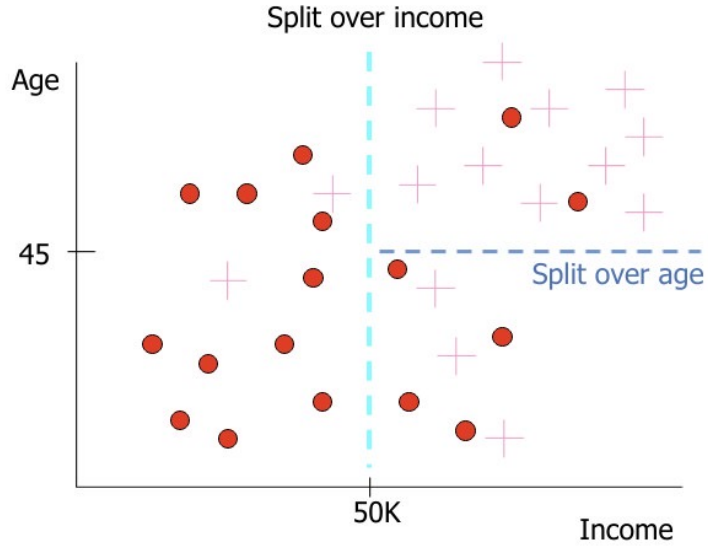
# Let’s model

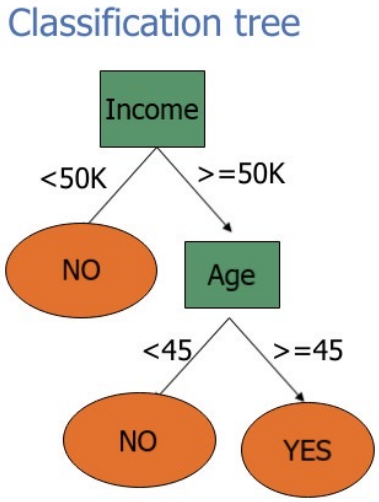
- We can model it through a Classification tree



# Let’s model

- We can model it through a Classification tree

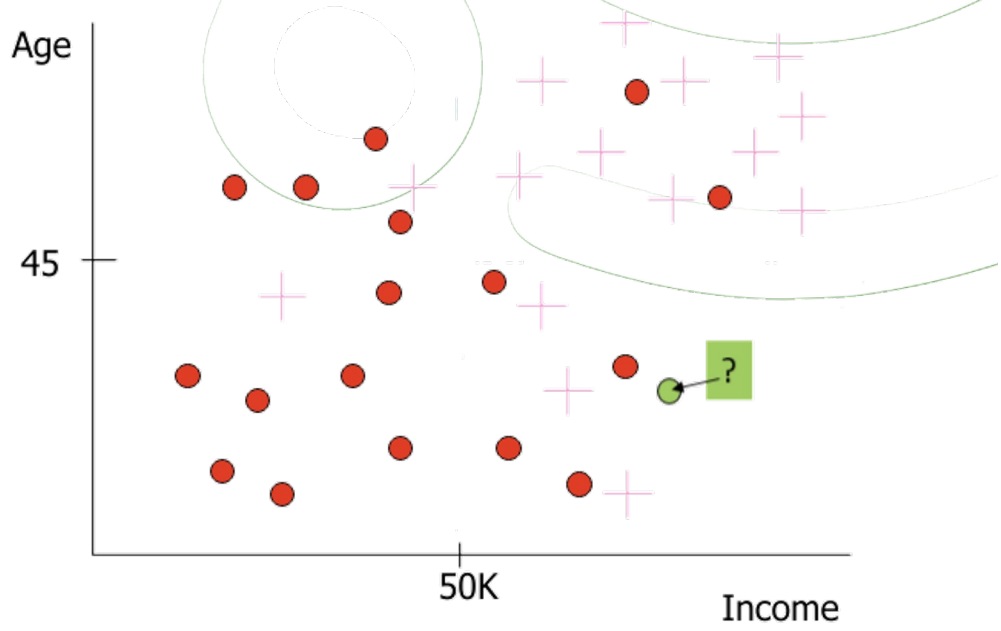


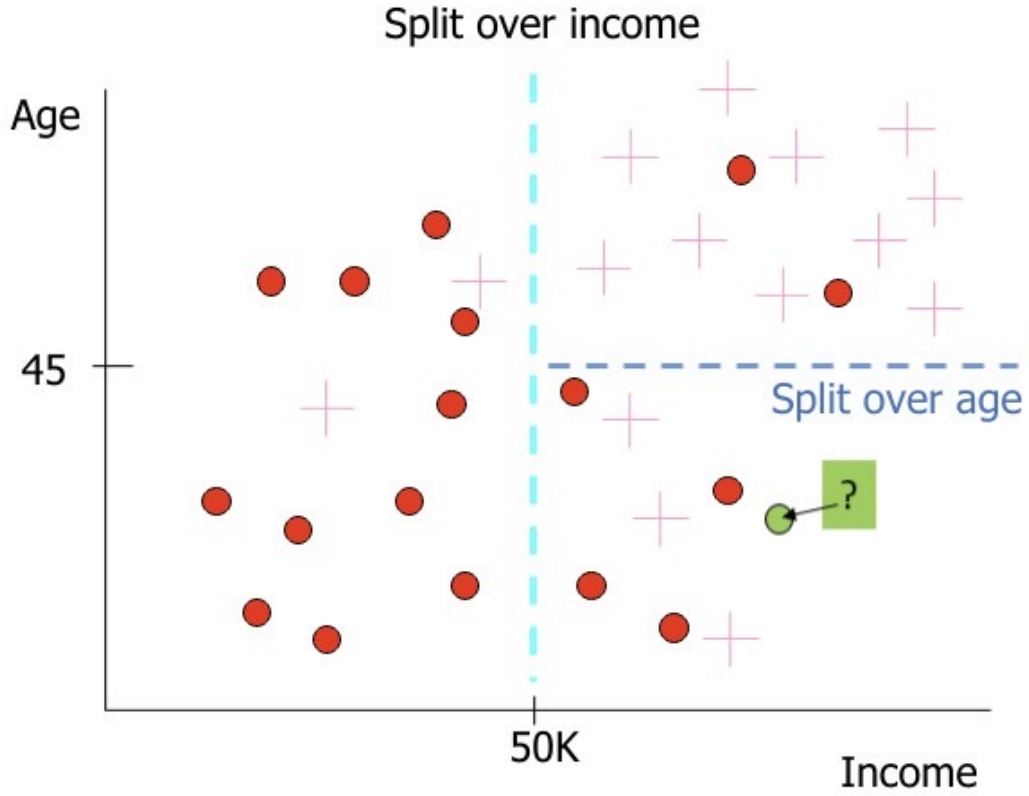

- The algorithm finds the optimal splits
- It maximizes prediction confidence

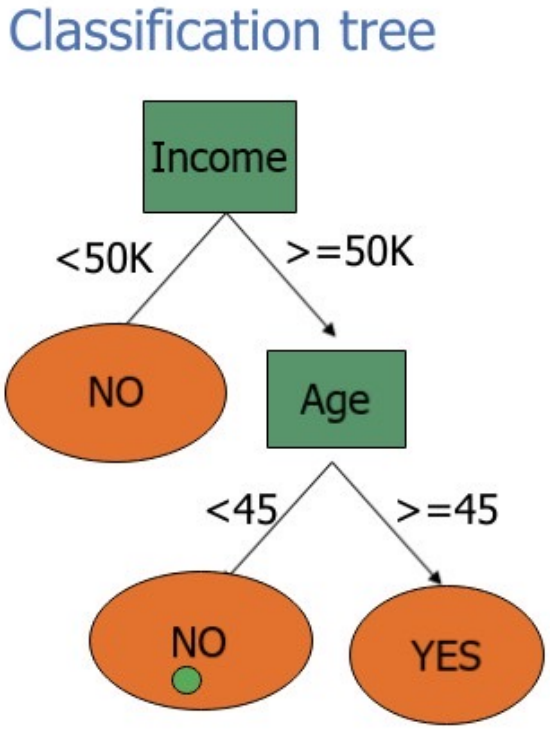
# Let’s apply the model now

- Let’s now generalize the model
- Say, we receive new data and we want to predict the **Out** variable
  - For instance, a new person 25 years old and with an income of 70k



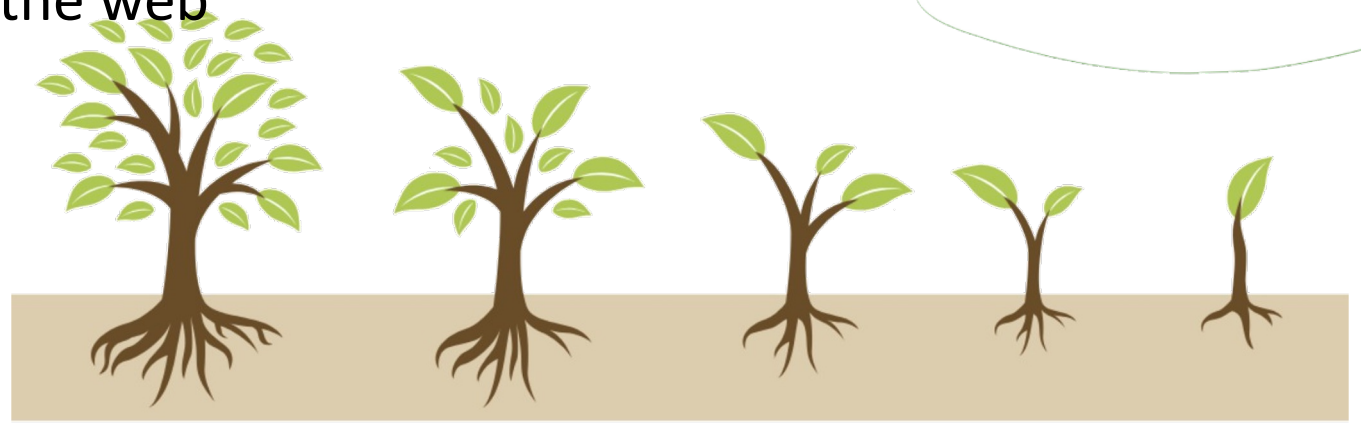
# We can now predict



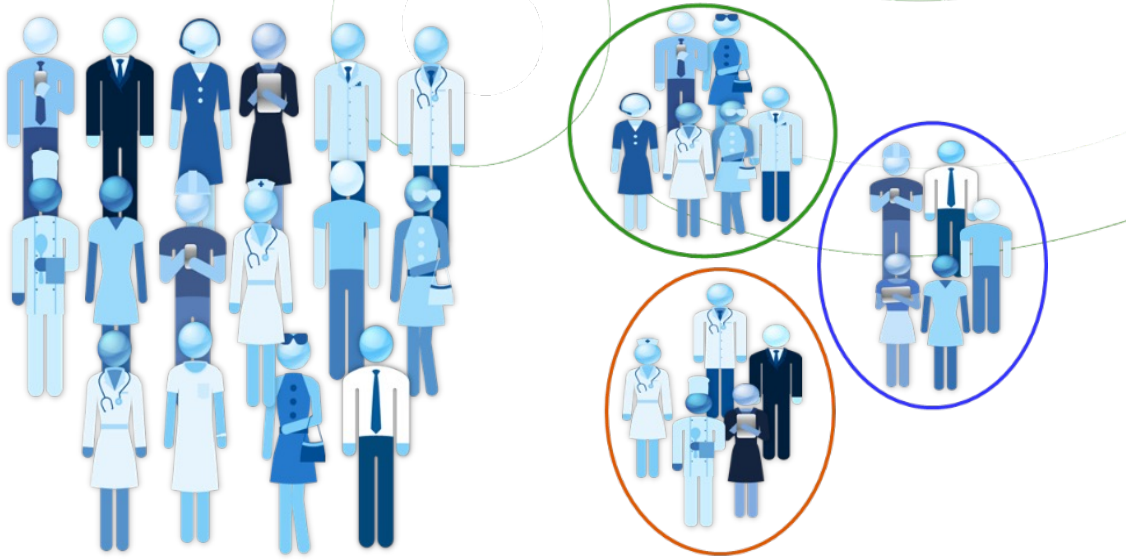
# Classification tree considerations

- C4.5 was the original idea
- Old technique very well established
- Works for categorical dependent variable
- It works with both continuous and categorical independent variables
- Fast to execute
- Easy to find free code on the web



# Clustering

- **Unsupervised learning model**
- Widely used in business/marketing applications
- Gives a structure to unsorted data points
- In simpler words: Aggregate similar items

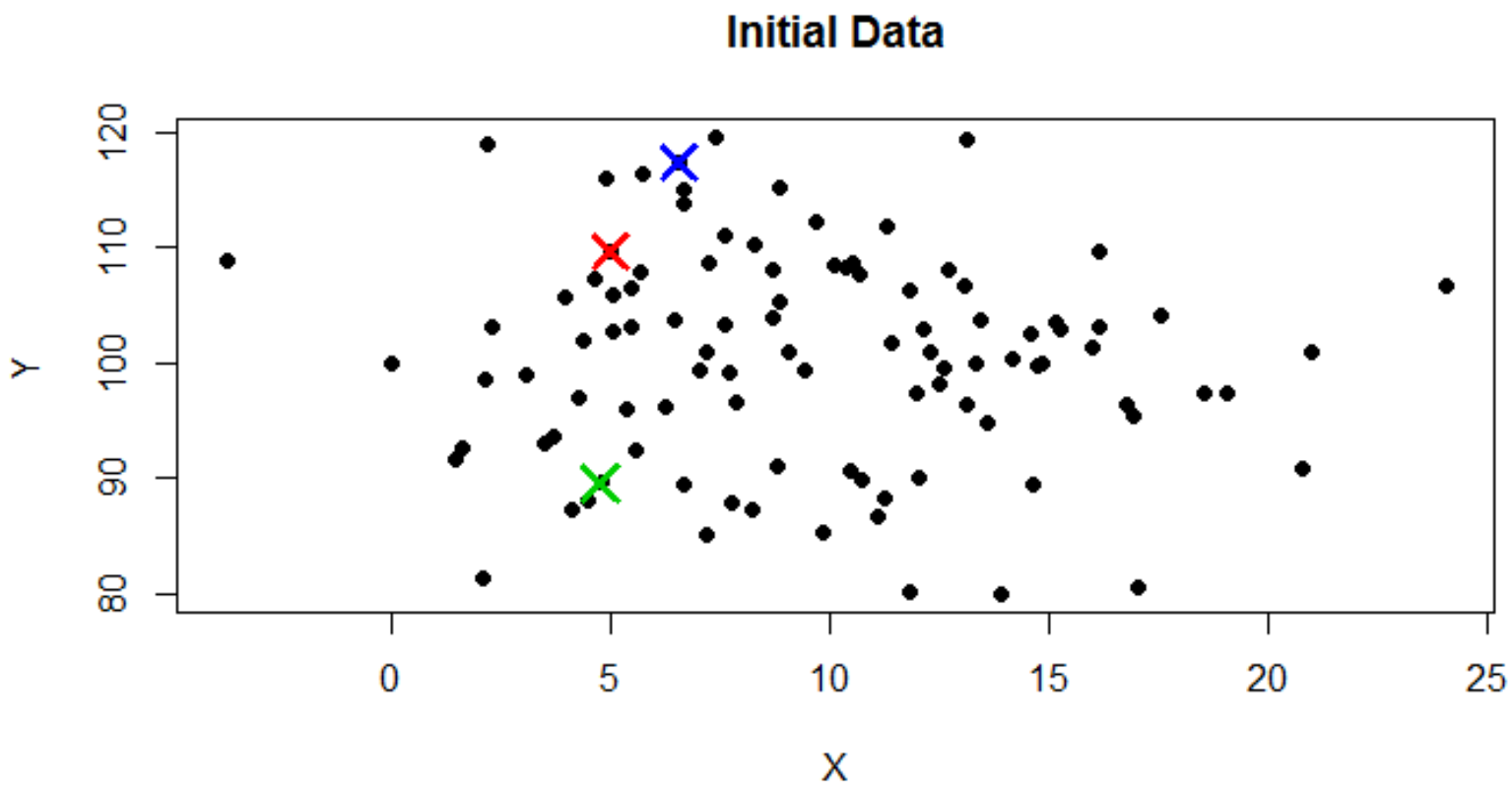


# k-means Clustering algorithm

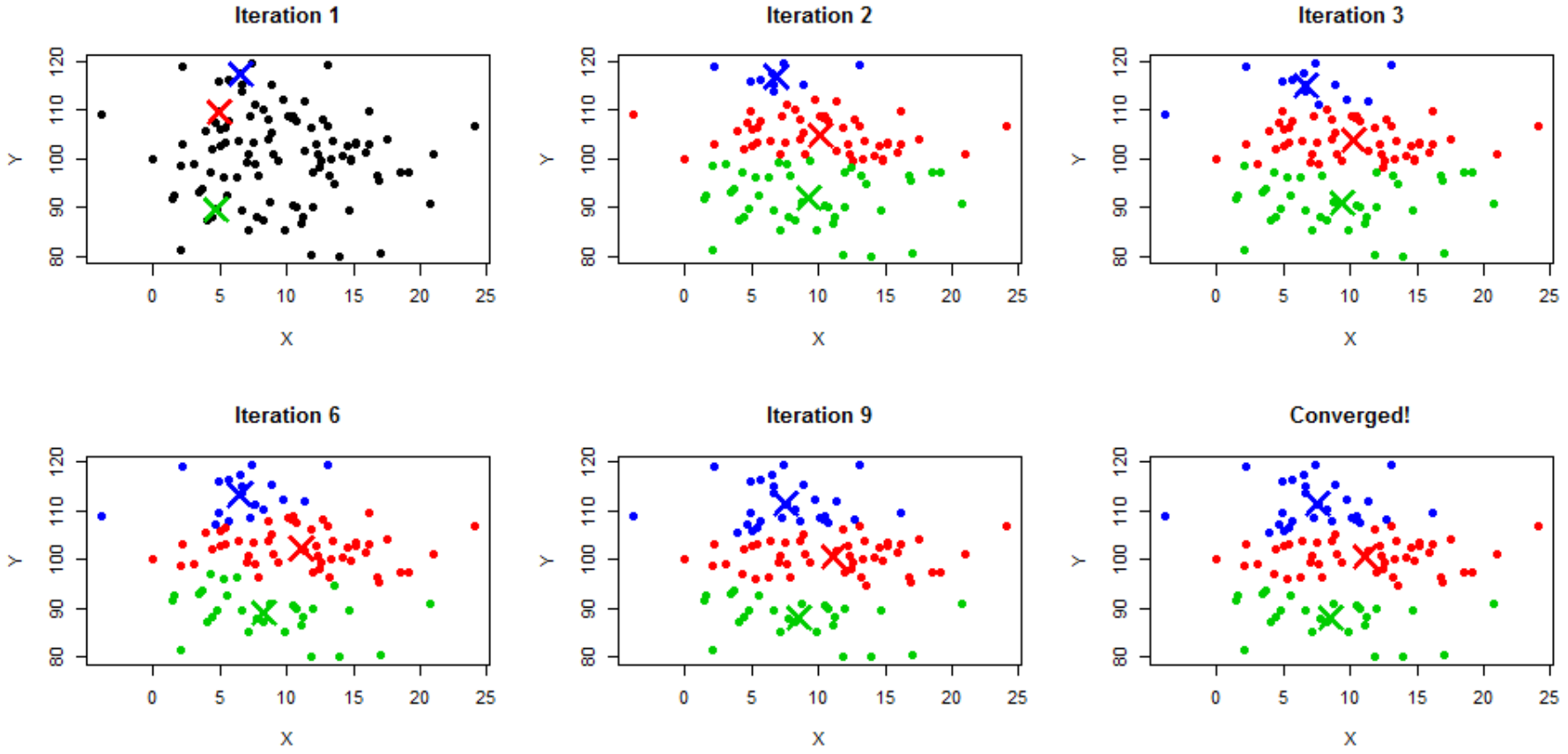
ITINERIS

- Step 0: Initialize K random centroids (just pick randomly K data points)
- Step 1 For every data point:
  - assign it to the closest centroid (any distance metric works);
- Step 2 For every centroid
  - move the centroid to the average among all its points;
- Repeat Step 1 and Step 2 until all centroids do not change anymore;
- The algorithm converged

# k-means example… Initial step



# k-means example… Iterations



# k-means, some considerations

- Easy to apply
- Various algorithms available
  - You can find free, ready to use, code on the web
- Not suitable for categorical variables
- Need to normalize variable for scale uniformity
  - Otherwise, distance calculation may be affected
- It scales on Big Data, that is, it can be parallelized
- How to pick initial K?

# Associations

*What can we infer just by observing?*

# Association rules

- Market-basket analysis: Understanding meaningful patterns by analysing baskets
- A basket is a generic set of items
  - Pattern: A set of items
  - Frequent pattern: A pattern that appears frequently
  - We infer rules such as: $A, B, \ldots, C \Longrightarrow E$
  - Beer and diapers on friday evening?!
  - It may depend on the context: ”Have kids?”, ”Travelling fo

# What Is the ML Pipeline?

- Structured process for building ML models
- Ensures repeatability and clarity
- Common across all ML workflows
- Steps: problem → data → model → evaluation

# Step 1 – Problem Definition

- Understand the business or scientific objective
- Define input/output clearly
- Set measurable success criteria
- Decide on ML type: supervised, unsupervised, or reinforcement

# Step 2 - Data Collection & Preparation

- Collect relevant data from appropriate sources
- Clean the data: remove duplicates, handle missing values
- Feature engineering: extract and format meaningful attributes
- Normalize/standardize data if needed

# Step 3 - Algorithm Selection

- Choose based on task: classification, regression, clustering
- Consider:
  - Data size and quality
  - Training time
  - Interpretability vs performance
- Start simple (e.g., linear models) and iterate

# Step 4 – Model Training

- Use the training data to teach the model
- Fit parameters to minimize error
- Adjust model settings (hyperparameters)
- Watch for convergence or signs of overfitting

# Step 5 – Model Evaluation

- Test on unseen data (test set)
- Use metrics:
  - Classification: accuracy, precision, recall, F1
  - Regression: RMSE, MAE, $R^2$
- Compare multiple models for best performance

# Overfitting vs Underfitting

- Overfitting: memorizes training data, **fails on new data**
- Underfitting: **too simple**, misses important patterns
- Visualization: training vs test accuracy curve
- Ideal: good generalization to new, unseen data

# Model Validation Basics

- Use a validation set or cross-validation
- Helps tune hyperparameters and assess performance
- K-Fold Cross-Validation = robust, avoids random split bias
- Use validation before final test

# Improving Generalization

- Techniques:
  - Regularization (L1/L2)
  - Dropout (in neural nets)
  - Pruning (in trees)
  - Ensembling (bagging, boosting)
- Collect more or better data
- Perform feature selection or engineering

# Summary & Q&A

- ML pipeline provides structure to model building
- Each stage impacts final performance
- Validation helps build robust, trustworthy models
- Avoid both overfitting and underfitting