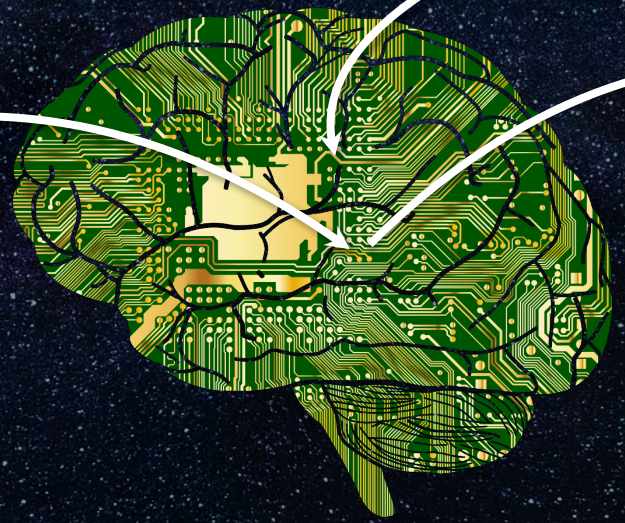
Edge Hill University

# So what is Automated Learning?

Edge Hill University

# Automated Decision Making

- **Machine Learning is concerned with making optimal decisions primarily using the tools of statistics.**




$$E = \{(x_1, y_1), (x_2, y_1), \ldots, (x_n, y_1)\}$$

Edge Hill University

# Automated Decision Making's Flaws

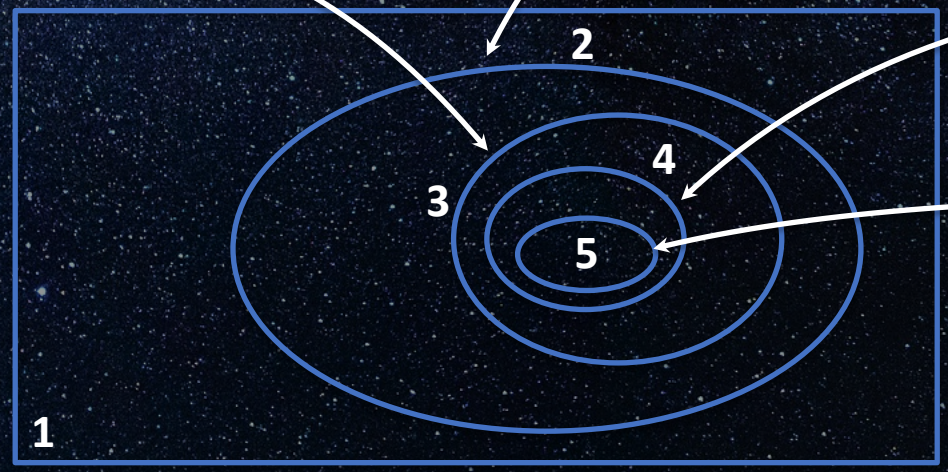
- **Biased input data** -
  For example, suppose you want to teach an algorithm to recognize a specific disease in a group of patients.
- You provide the algorithm with data describing patients who are all aged between 60 to 80.
- You then run the algorithm on patients aged 18 to 30. The algorithm performs badly, as it's knowledge is biased toward recognizing disease in much older patients.

- **Biased cognitive process** -
  For instance suppose we try to teach an algorithm to predict when a train on the Tokyo rail network will be late.
- Trains on this network are exceptionally punctual.
- To achieve the best overall performance, just never predict that a train will be late.

Edge Hill University

# Making Optimal Automated Decisions

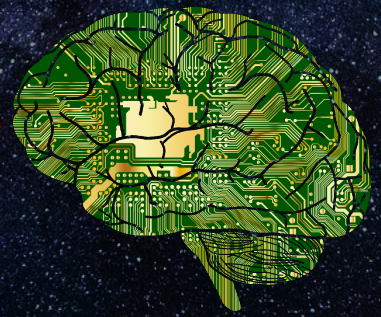
- Algorithms can process more quantifiable data than humans.
- However the data may be biased/incomplete.
- Individual algorithms also have intrinsic biases.
- Algorithms can therefore be just as fallible as humans!



Edge Hill University

# From Experience to Training Data

- Experience is known as "training data".
- An algorithm is ‘taught’ using training data.
- Training data can be labelled or unlabelled.
- We evaluate performance on a disjoint dataset called "test data" - this data must be labelled (ground truth known).



$$Training\ data = \{(x_1, y_1), (x_2, y_1), \ldots, (x_n, y_1)\}$$

Edge Hill University

# Training vs Test Data

**There is an easy way to understand the difference between training and test data.**

- **Training data represents the knowledge given before an exam.**
- **Test data is a disjoint set of information, that represents the exam – we use this to test performance.**

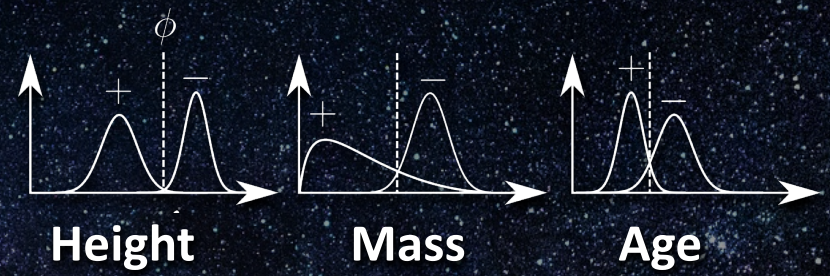

**Set of all knowledge on subject *A***

Edge Hill University

# Features & Labels Revisit

**Features (Variables)**

$$x_i = \{x_i^1, \ldots, x_i^m\}$$




**Labels ( Feedback )**

$$y_i \in \{-1, 0, 1\}$$

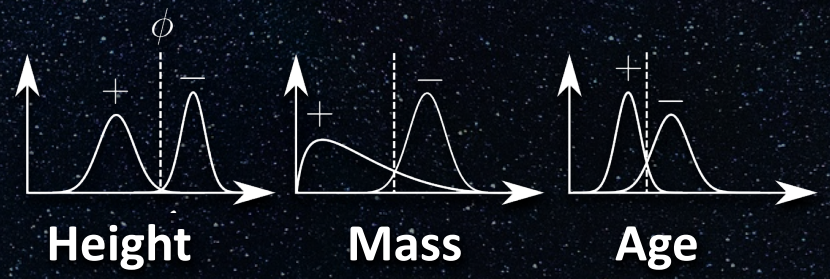
**e.g. -1 = no disease etc**

$$Training\ data = \{(x_1, y_1), (x_2, y_1), \ldots, (x_n, y_1)\}$$

Edge Hill University

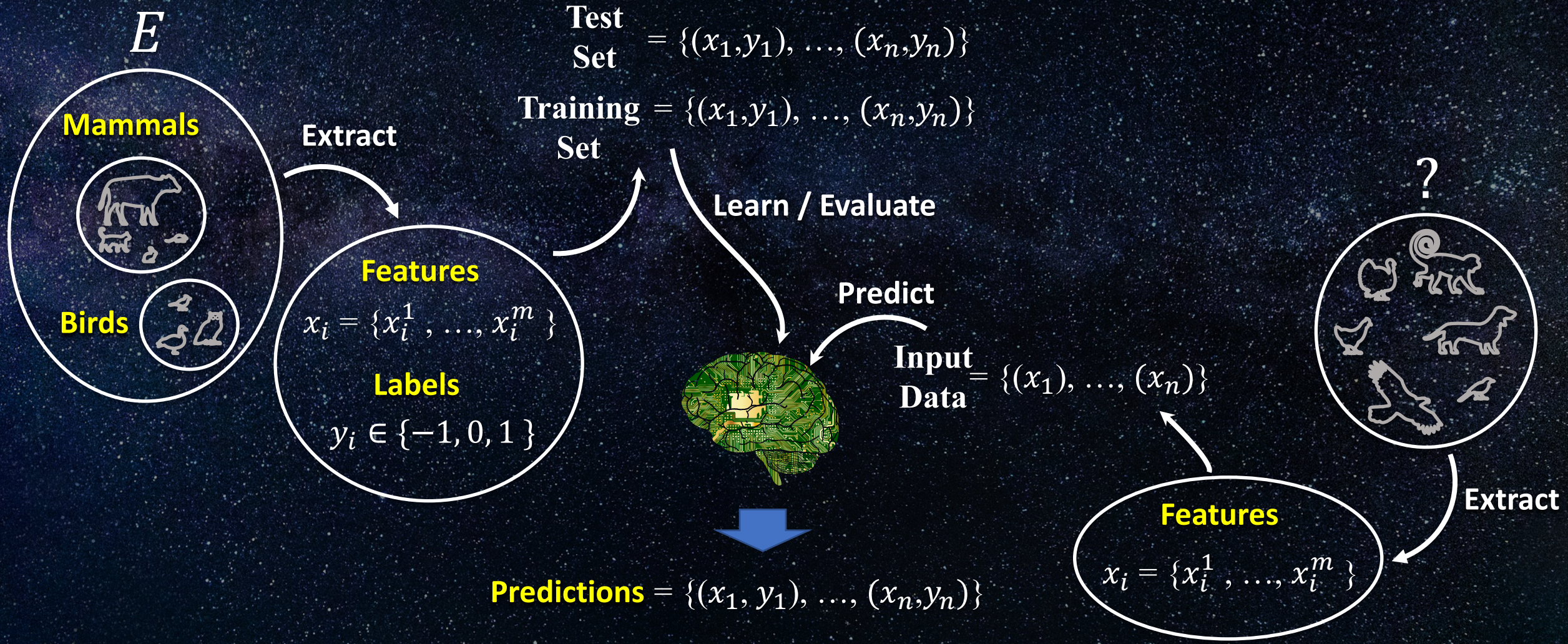
# Feature Design

- In principle feature design involves studying your data.
- Considering it's properties.
- Extracting information you believe will help with class separation.
- In practice this involves,
  1. considering as many candidate features as possible.
  2. Considering their useful ness in turn.
  3. Selecting a sub-set of features you think will be effective.
  4. Testing those, see what happens.
  5. Return to step 1 if results are not as expected.
  6. Collect new data?
  7. Deriving new features?



Edge Hill University

# Classification Process



Edge Hill University

# So How is “Learning” Done?

- **Learning via trial and error – learning from mistakes made.**


**We need to be at work, but don’t know what time the bus arrives to get us there. Work starts at 09:00, and the journey takes roughly 40 minutes.**

**We start to wait at the bus stop each day, recording some details.**

| Day | Arrived at bus stop | Bus arrived | Late for work $y$ |
|---|---|---|---|
| Mon | 08:32 | 08:45 | Yes |
| Tue | 08:17 | 08:30 | Yes |
| Wed | 08:12 | 08:15 | No |

**In this case learning involves finding an arrival time at the bus stop, that lets you make it to work on time. Each time you’re late, you’ve made an error that you learn from.**

Edge Hill University

# So How is "Learning" Done?

- In ML error is quantified and minimised using the tools of mathematics.
- This sort of error minimisation is done using functions.


We can reduce finding the best time to wait at the bus stop, to a mathematical problem, i.e. find a value for $t$ that minimizes the number of times you're late.

Many potential values for $t$ work, but clearly it must be 08:15 or earlier!

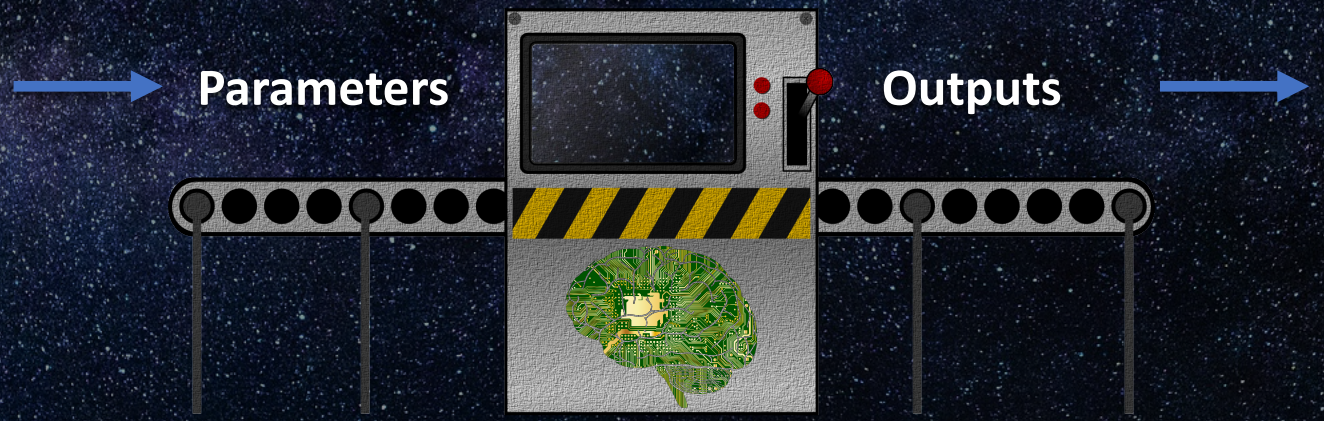
| Day | Arrived at bus stop $t$ | Bus arrived |
|---|---|---|
| Mon | 08:27 | 08:45 |
| Tue | 08:17 | 08:30 |
| Wed | 08:12 | 08:15 |

| Late for work $y$ |
|---|
| Yes |
| Yes |
| No |

Edge Hill University

# Functions

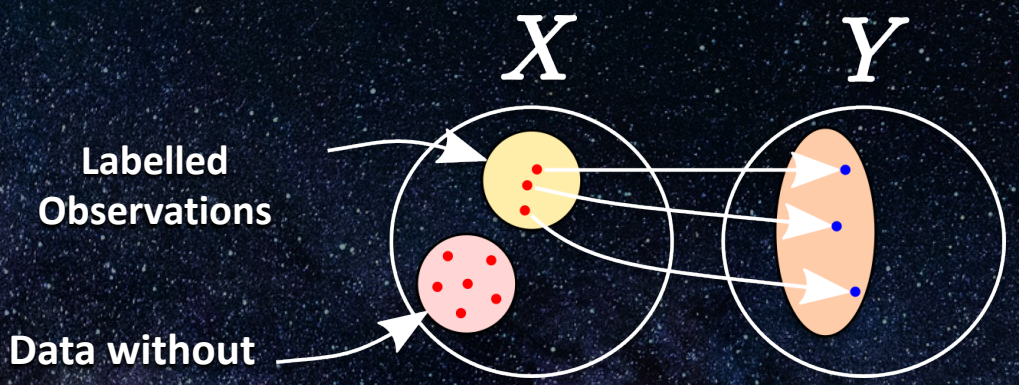
- Functions can be thought of as simple input/output boxes.
- Machine learning algorithms are functions that ingest data, and produce some output.
- How they do this is hard to convey without some basic mathematics!




$f(x, y) = x + y$ e.g. $f(x = 2, y = 4) = 2 + 4 = 6$

Edge Hill University

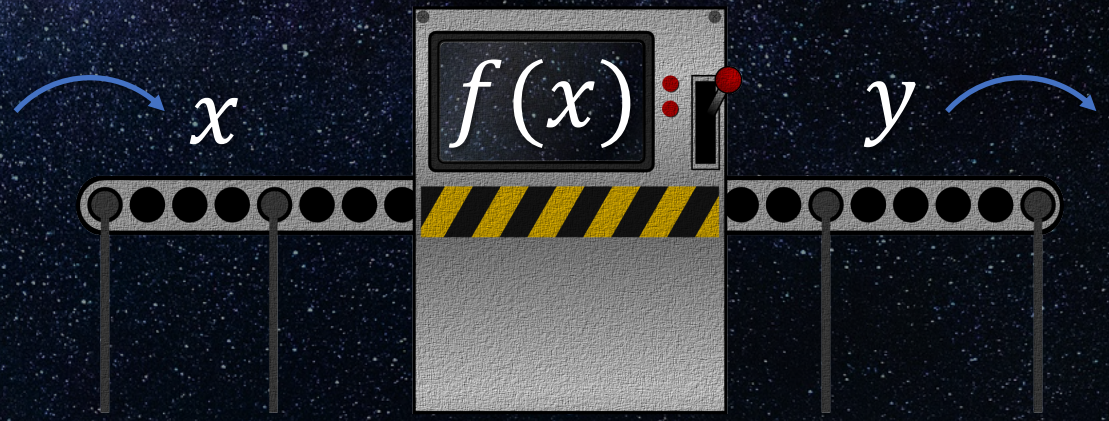
# Functions That "Map"




**E.g.**

$$f(x) = 0, if\ feature_1 \leq 0$$
$$1, otherwise$$

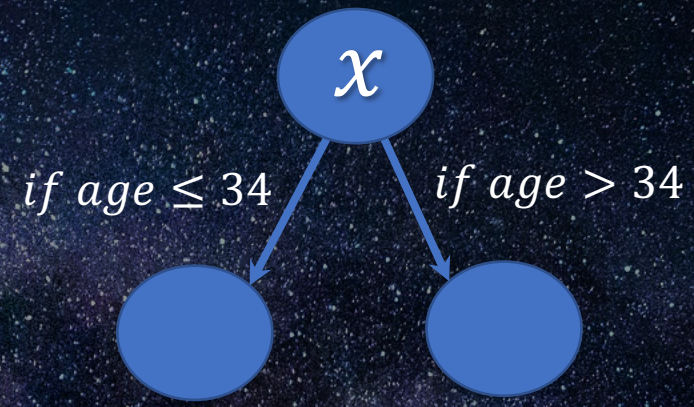
| $feature_1$ | $feature_2$ |
|---|---|
| 7.4 | 29 |
| 6.6 | 24 |
| 3.1 | 11 |
| … | … |
| 0.5 | 2 |

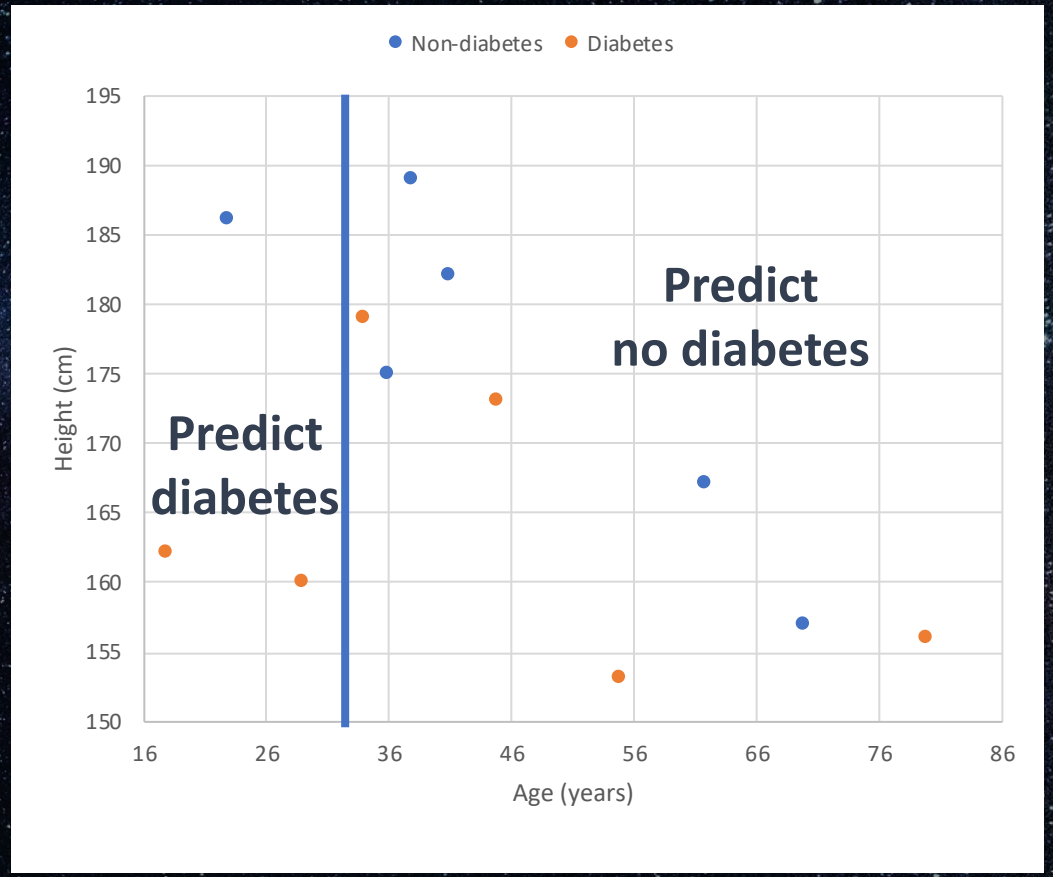



| $y$ |
|---|
| 1= Diabetic |
| 1= Diabetic |
| 0 = Non-diabetic |
| … |
| Unknown |

Edge Hill University

# Decision Stump

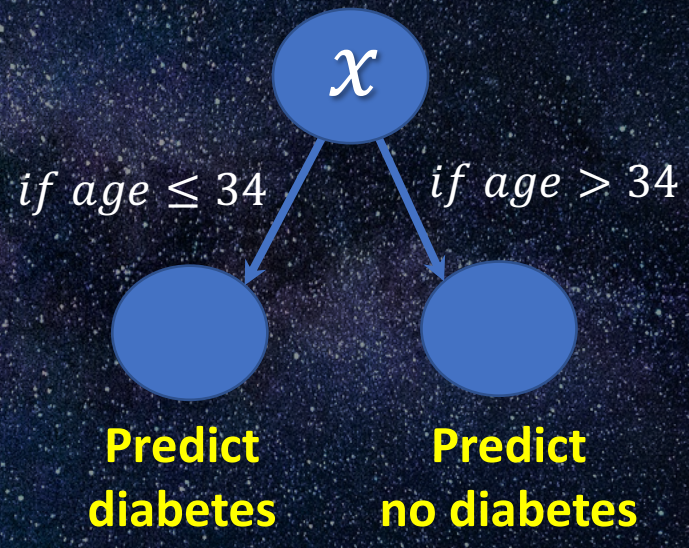



- We've just covered our first ML algorithm - the "Decision stump".
- It is a linear separator, or a linear model.
- It simply looks for a feature to split on, in a optimal way.
- It produces a linear separator between two classes.

Edge Hill University

# Decision Stump

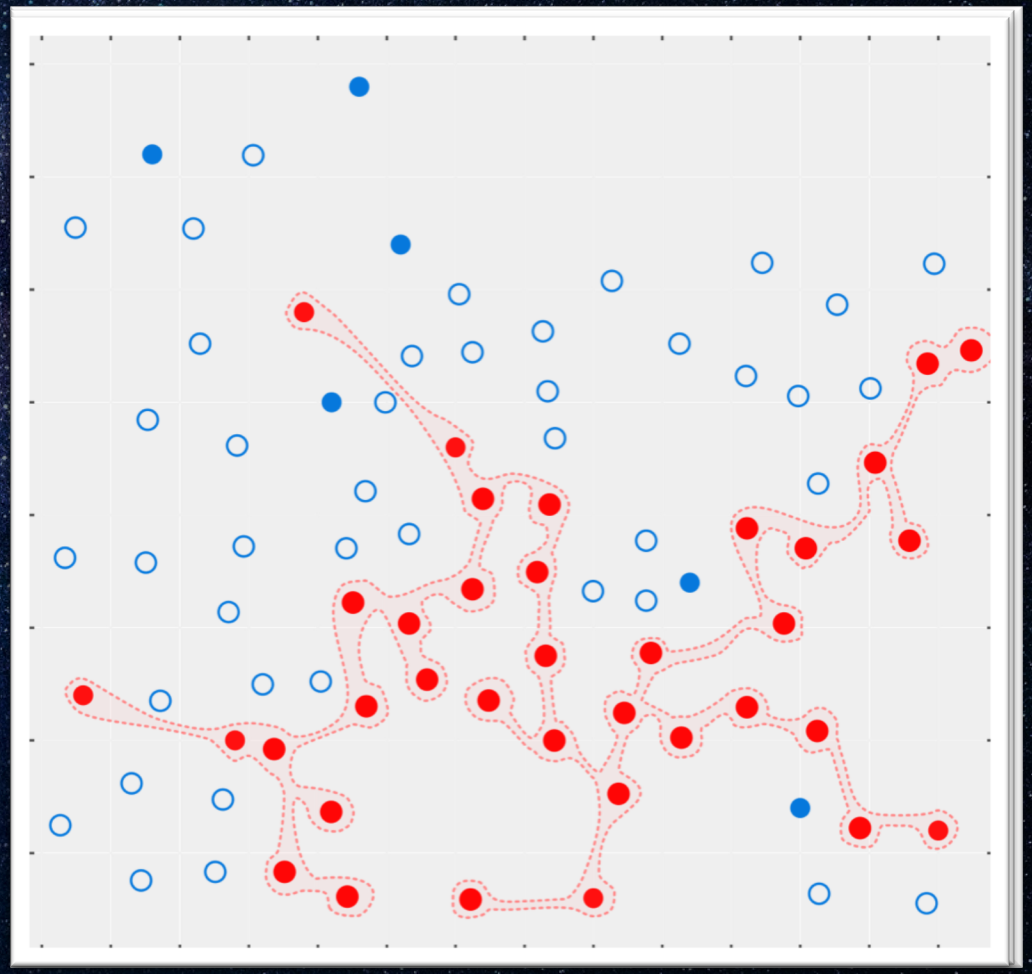



- For each feature in the data, the stump will search for a split-point, that minimises the error rate.
- In our example we used age as a feature, and our threshold = 34.

Edge Hill University

# Decision Boundary

Shaded region predicts diabetes



- You've now seen a linear model in action, and learned what a decision boundary is.
- Real-world problems are complex - so too are the decision boundaries needed to accurately separate data.
- This means linear models don't always work well. More complex ML algorithms are required.
- When we generalise well we can produce effective decision boundaries.
- However sometimes we don't perform well in practice.
- We can underfit to our data.
- We overfit to our data.
- Decision boundaries allow us to visualise this happening.

Edge Hill University

# Learning in Name Only…

- What we've talked about so far - this isn't really learning right?
- It's a search for parameters that minimise some error rate.
- Learning effectively reduced to a parameter search!
- When this works well it provides the illusion of intelligence – but this isn't how you or I learn.

Edge Hill University

# Classification Algorithms

- Classification algorithms come in all shapes and sizes.
- Inside they are comprised of functions, that attempt to perform the mapping from $x$ to $y$ in different ways.
  (Examples to labels)
- What they have in common:

All are evaluated against some ***feedback*** metric,
which guides their learning - call this the error rate.

They must choose (find) parameter values
that minimise the error rate.

Edge Hill University

# Classification Algorithms

Edge Hill University

# Types of Classifier

## Discriminative

- Learns the differences between the examples in a training set.
- Use those differences to classify and make predictions.
- We often think similarly, like in the Fossa example.

## Generative

- Learns a “model” of the classes in the training set.
- This model can be then compared against.
- We don’t usually think this way, e.g. given all the cats we’ve seen, what is the probability that this animal is a cat?

Edge Hill University

# Generative Decision Making

**Probability of mystery disease**

$$P(disease) = 0.1$$

**Probability of red spotty rash**

$$P(symptom) = 0.1$$

**Probability of red rash for those with disease**

$$P(symptom|disease) = 0.8$$

$$P(hypothesis|data) = \frac{P(data|hypothesis) \times P(hypothesis)}{P(data)}$$

**Bayes' Theorem**

$$P(disease|symptoms) = \frac{0.8 \times 0.1}{0.1}$$

$$= 0.80 = 80\%$$

**So the probability of having the mystery disease is 80%, if you have a red rash.**

Edge Hill University

# Generative Decision Making

**Probability of mystery disease**

$$P(disease) = 0.1$$

**Probability of red spotty rash**

$$P(symptom) = 0.1$$

**Probability of red rash for those with disease**

$$P(symptom|disease) = 0.8$$

**40 to 60 year old's**

$$P(disease) = 0.001 = 0.1\%$$

**Mystery disease = Chicken pox!**

$$P(disease|symptoms) = \frac{0.8 \times 0.001}{0.1}$$

$$= 0.8\%$$

**This shows that the data used during training, must be representative of the data to be faced in the real-world.**

**We've actually just learned our next ML algorithm - Naïve Bayes. It makes predictions using probability.**

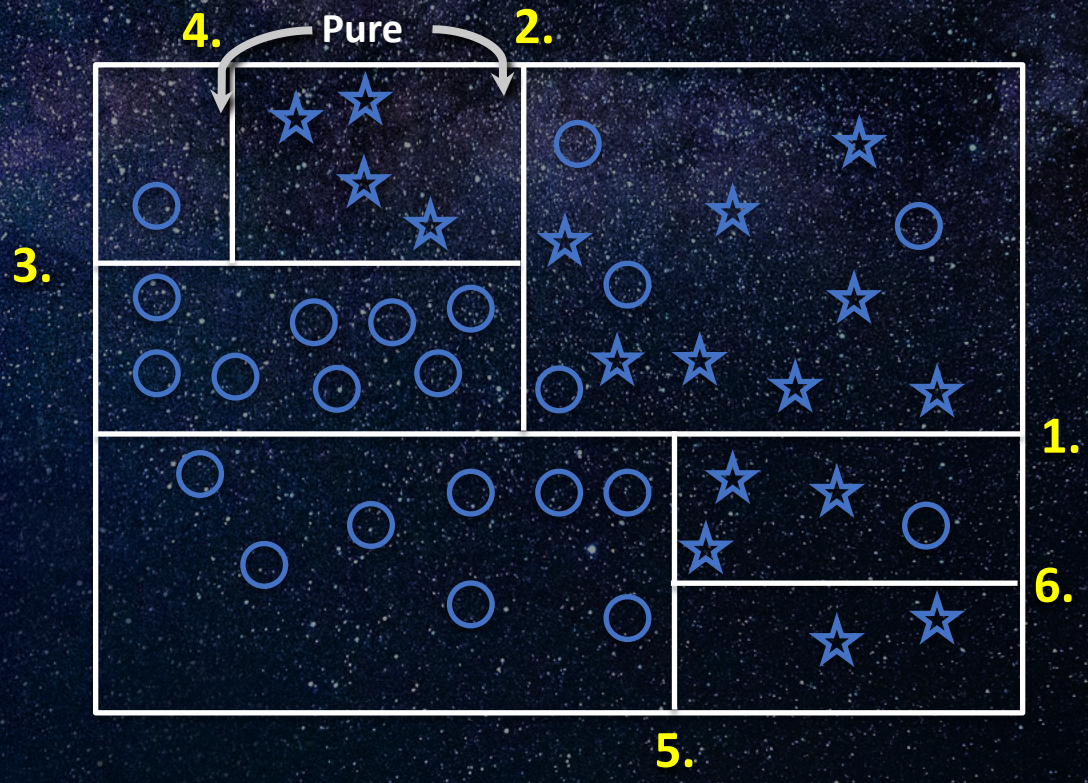
Edge Hill University

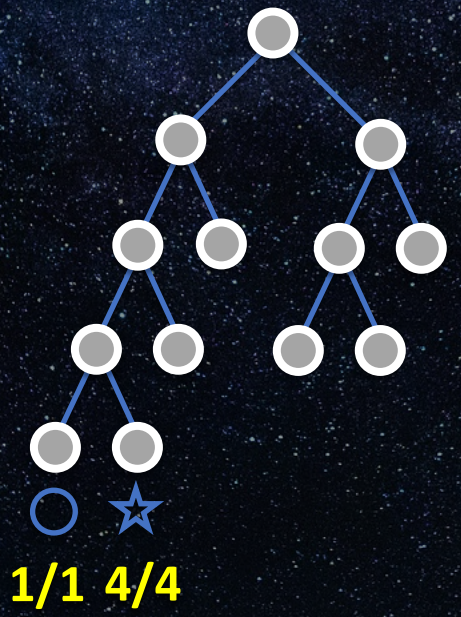
# Tree Learning – Discriminative Model

**Suppose we have a dataset containing 2 classes:**

- **Stars**
- **Circles**

**Goal is the separate them as accurately as possible.**

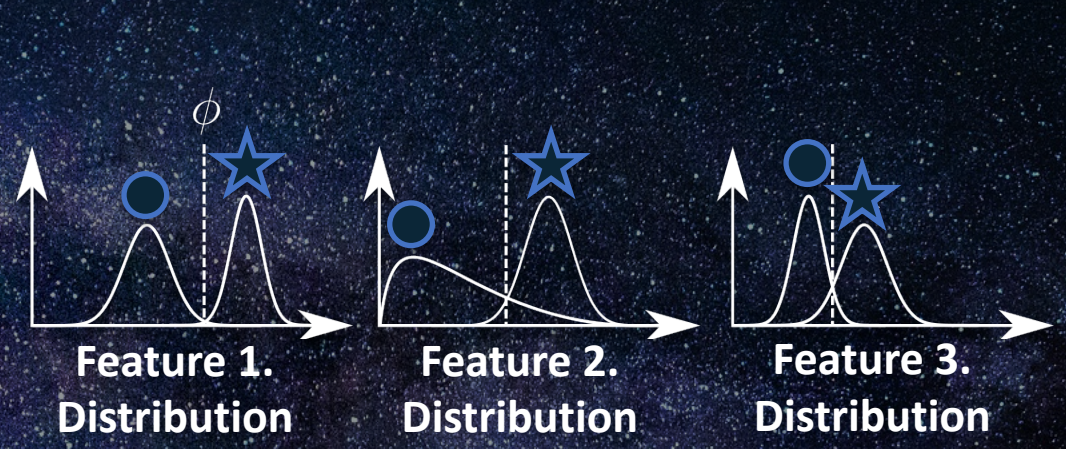


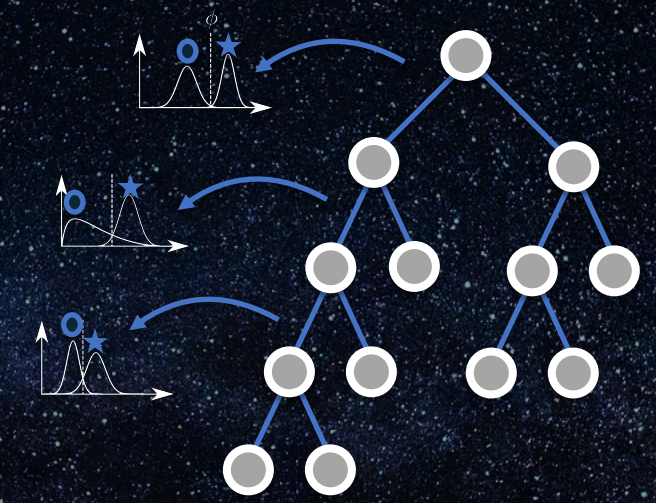

- **Non-linear model.**
- **Successively discriminates into classes via partitioning the data.**
- **Each partition forms an extra decision boundary.**
- **Key question – which feature to split on at any given time?**

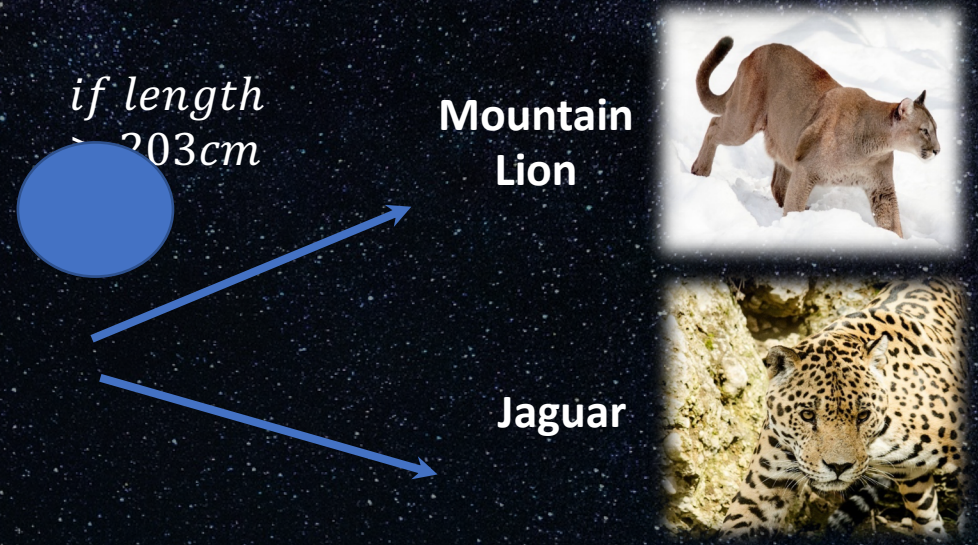
Edge Hill University

# Tree Split Choice



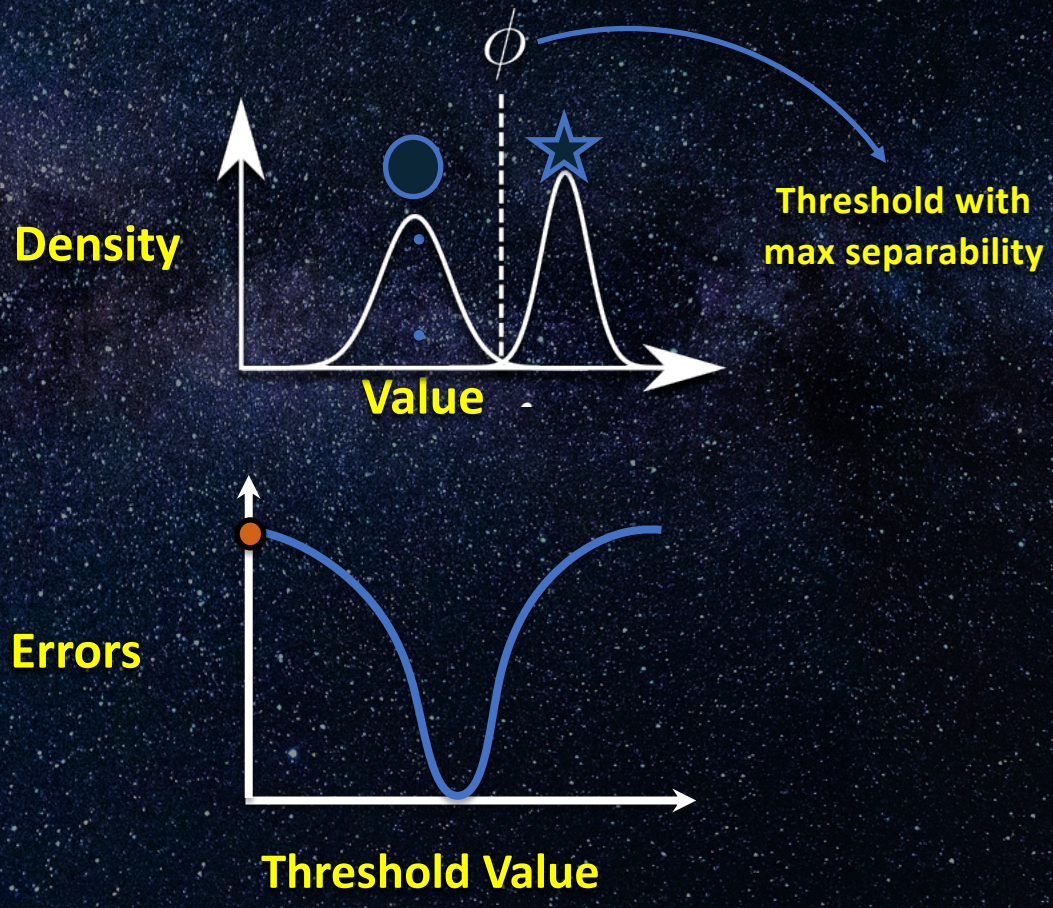

- At each spilt point, find the feature that maximises the separation between each class.
- Done by considering the statistical distribution of the features.
- Via an optimisation method (search over the possible split-points, find the best).

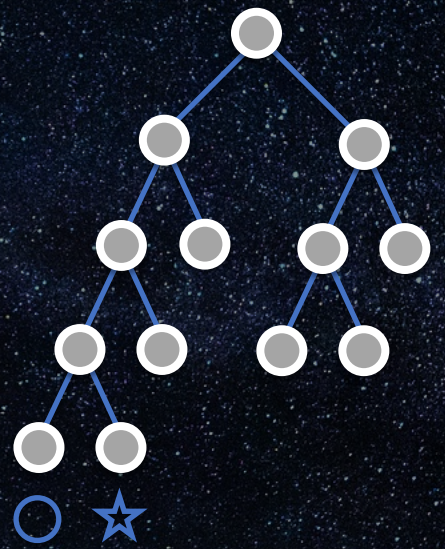


Edge Hill University

# Splitting Guided by Error

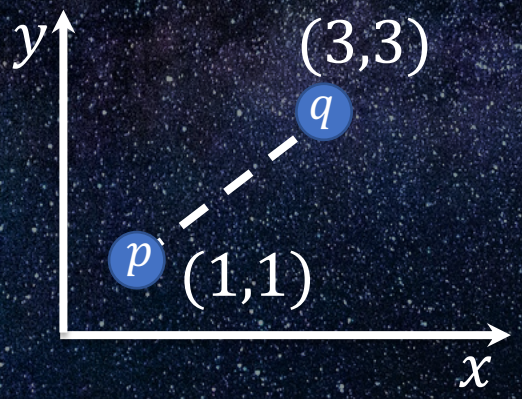


Edge Hill University

# Unsupervised learning

- To make a prediction for a data point, unsupervised algorithms try to group them together with the points closest to them in the training set – i.e. group according to the "nearest neighbors".
- Closest / nearest neighbours are computed using distance measures e.g. Euclidean distance:




$$d(p,q) = \sqrt{(q_1 - p_1)^2 + \ldots + (q_n - p_n)^2}$$

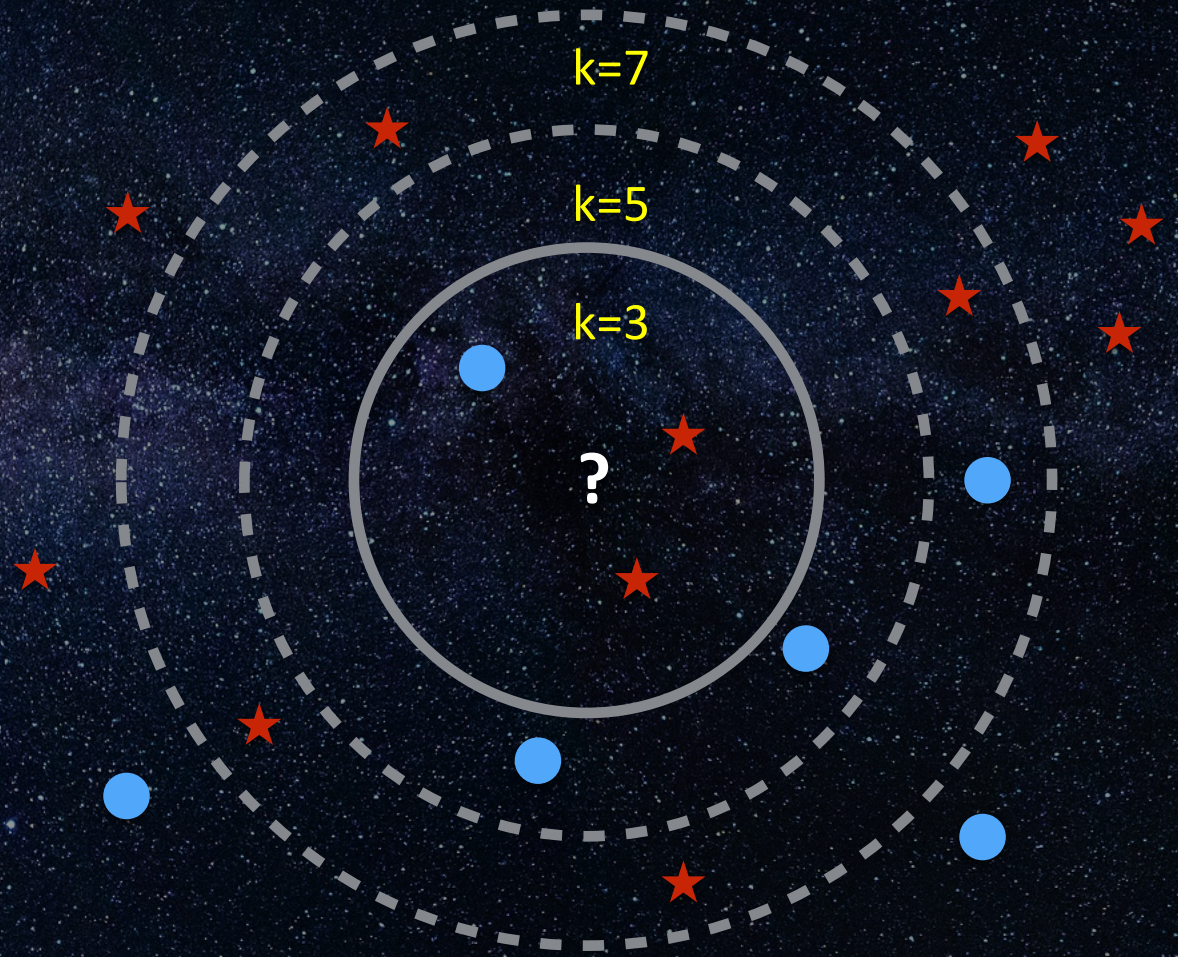
$$= \sqrt{(3-1)^2 + (3-1)^2}$$

$$= \sqrt{(2)^2 + (2)^2} = \sqrt{4+4} = 2.828 \text{ to } 3.\text{d.p}$$

- Once we find the nearest neighbours of a data point via a method like above, we determine the most popular label amongst it's nearest neighbours.
- We assign the majority label of the neighbours to the unlabeled data point.
- We need to determine how many of it's nearest neighbours, $k$, we should use to decide.

Edge Hill University

# $K$-nearest Neighbours




| K value | Prediction |
|---|---|
| 3 | ★ |
| 5 | ● |
| 7 | ★ |

Edge Hill University

# Evaluation

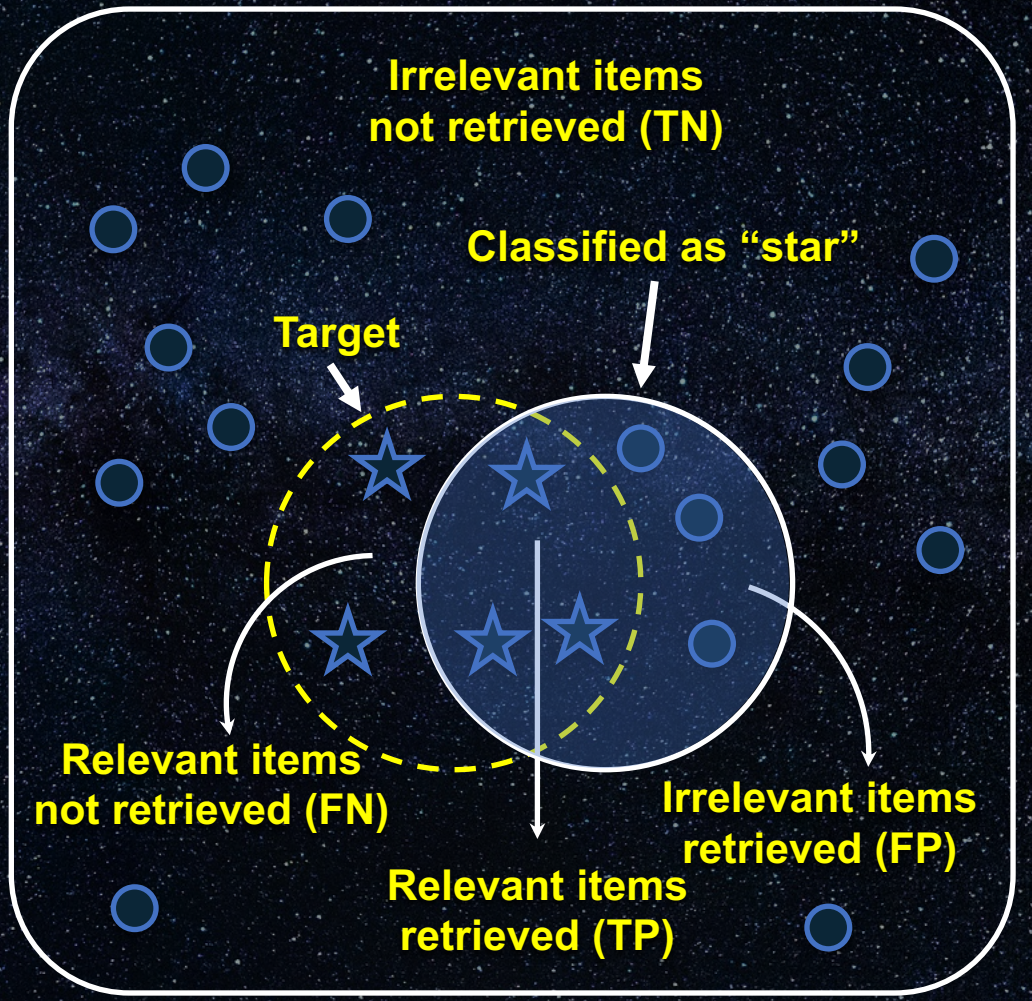
By counting the outcomes, we can evaluate performance. To do this we use "metrics" that quantify performance.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

Many, many more metrics!




True Positives (TP)

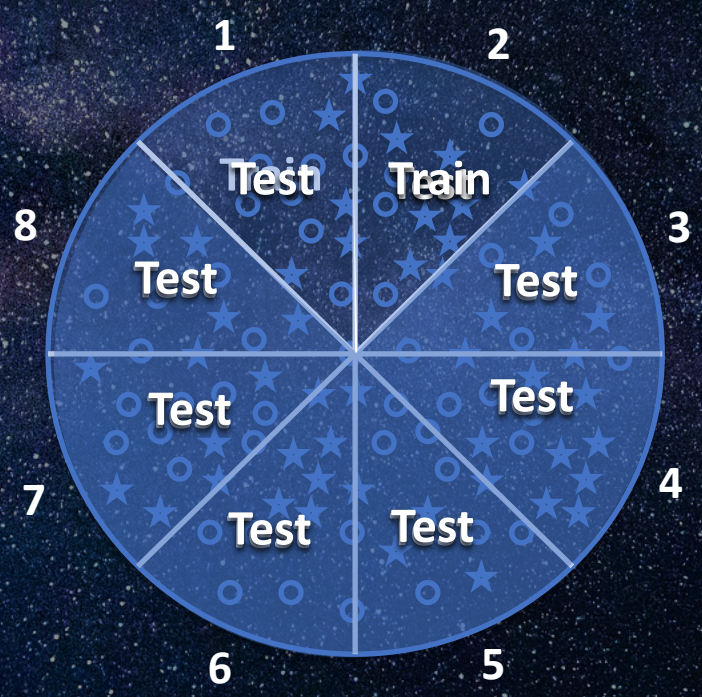
False Negatives (FN)

False Positives (FP)

True Negatives (TN)

Edge Hill University

# Practical Evaluation

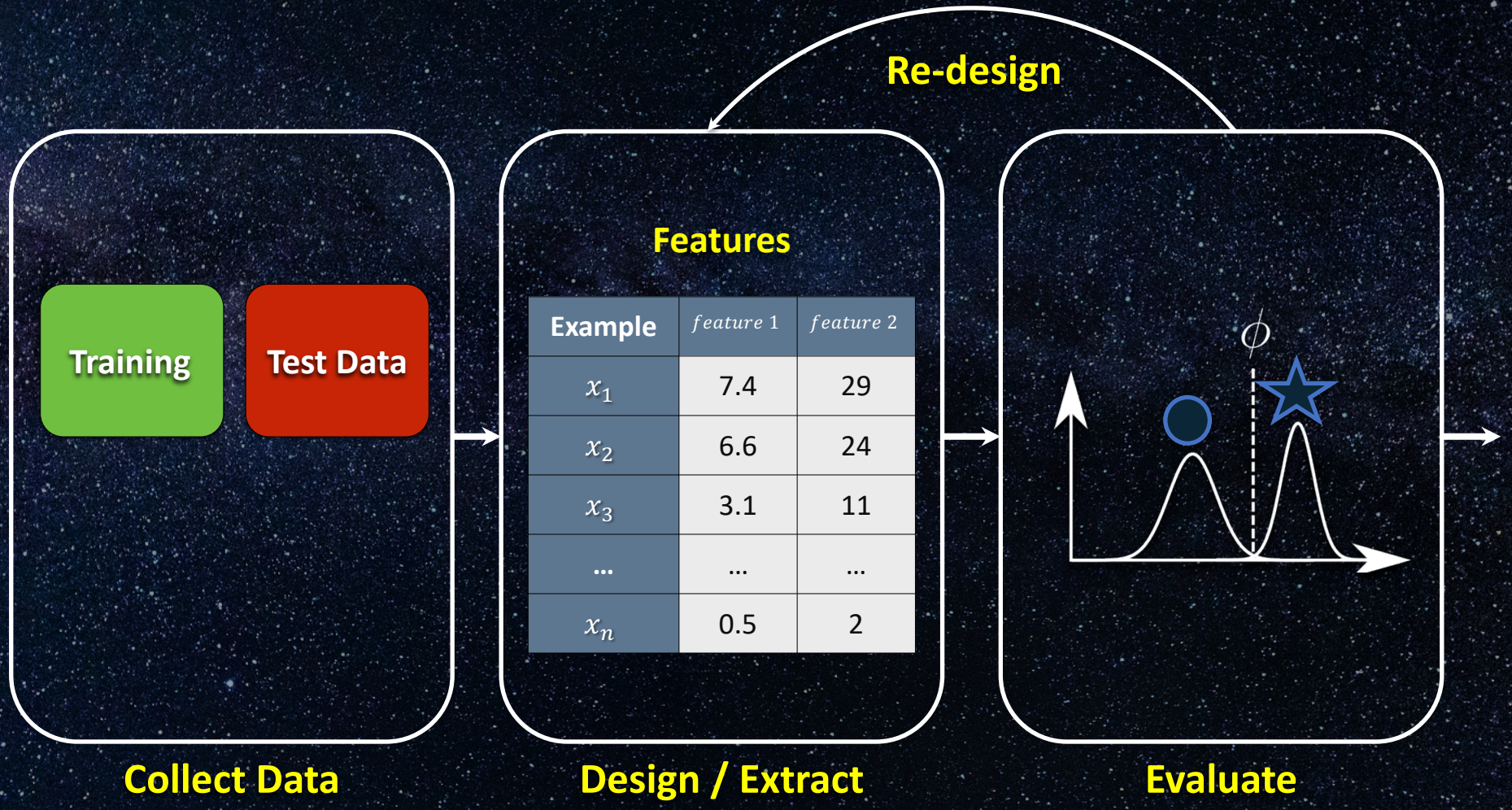



N-fold Stratified Cross Validation

1. Randomly split labelled data in to $n$ equally sized portions called ”folds”. Ensure each fold has the correct proportion of examples for each class, when compared to the complete labelled data set.
2. Label the folds using numbers.
3. Use the first fold to train a model, then test on the remaining folds.
4. Record the result achieved.
5. Now use the second fold to train the model. All other folds (including the the first) now used for testing.
6. Repeat this process until all folds have been used for training.
7. Aggregate the results on each fold, and compute averages. This allows a more rounded impression of performance.

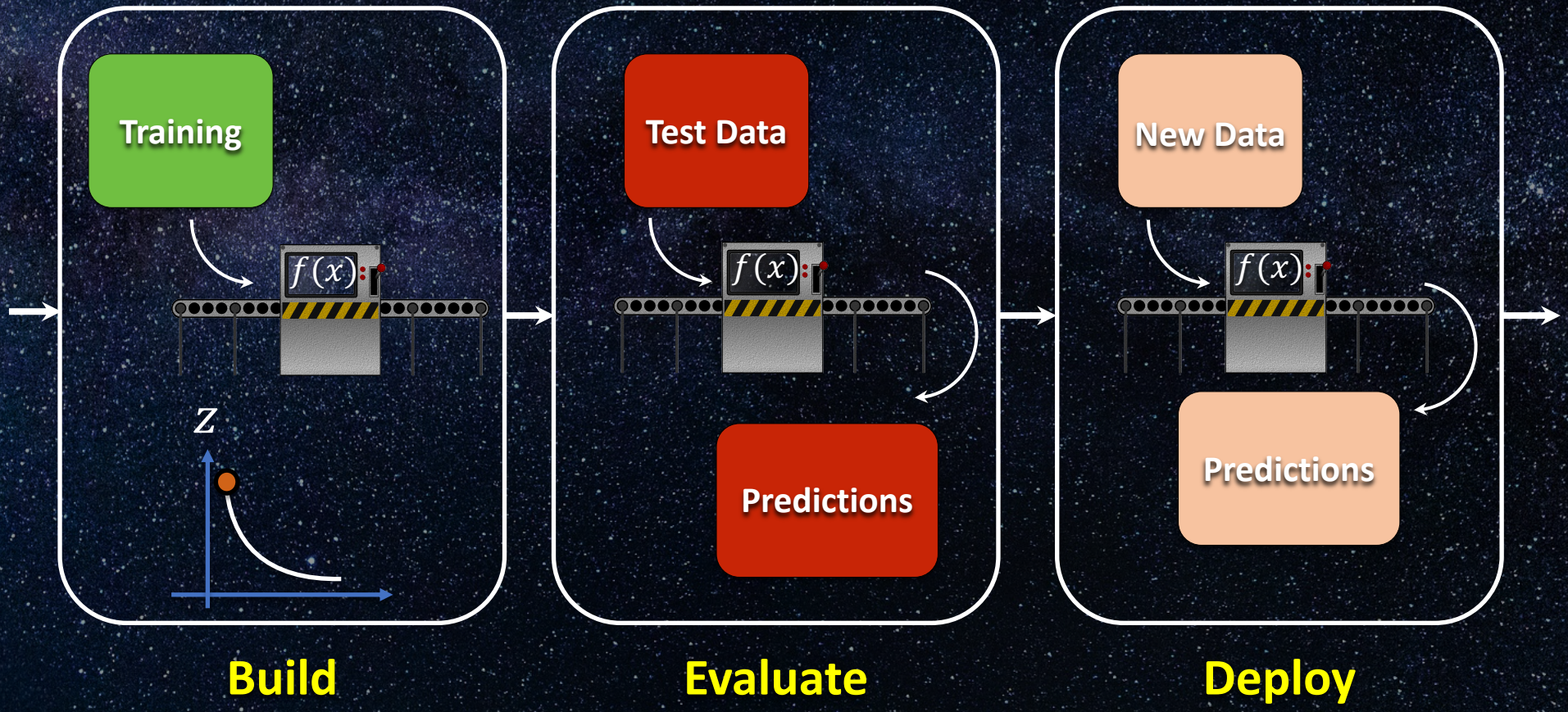
Edge Hill University

# Summary of Process (1)



Edge Hill University

# Summary of Process (2)



Edge Hill
University

# Python

Edge Hill University

# Types of Languages

- There are two different types of coding language:

"Low Level" Languages

- Hard for humans to read.
- Easy for machines to read.
- Can be processed efficiently by CPUs.
- Often written in binary or simple numerical format.
- Can be used to control specific parts of the CPU.
- The total lines of code required to do simple tasks can become very large!

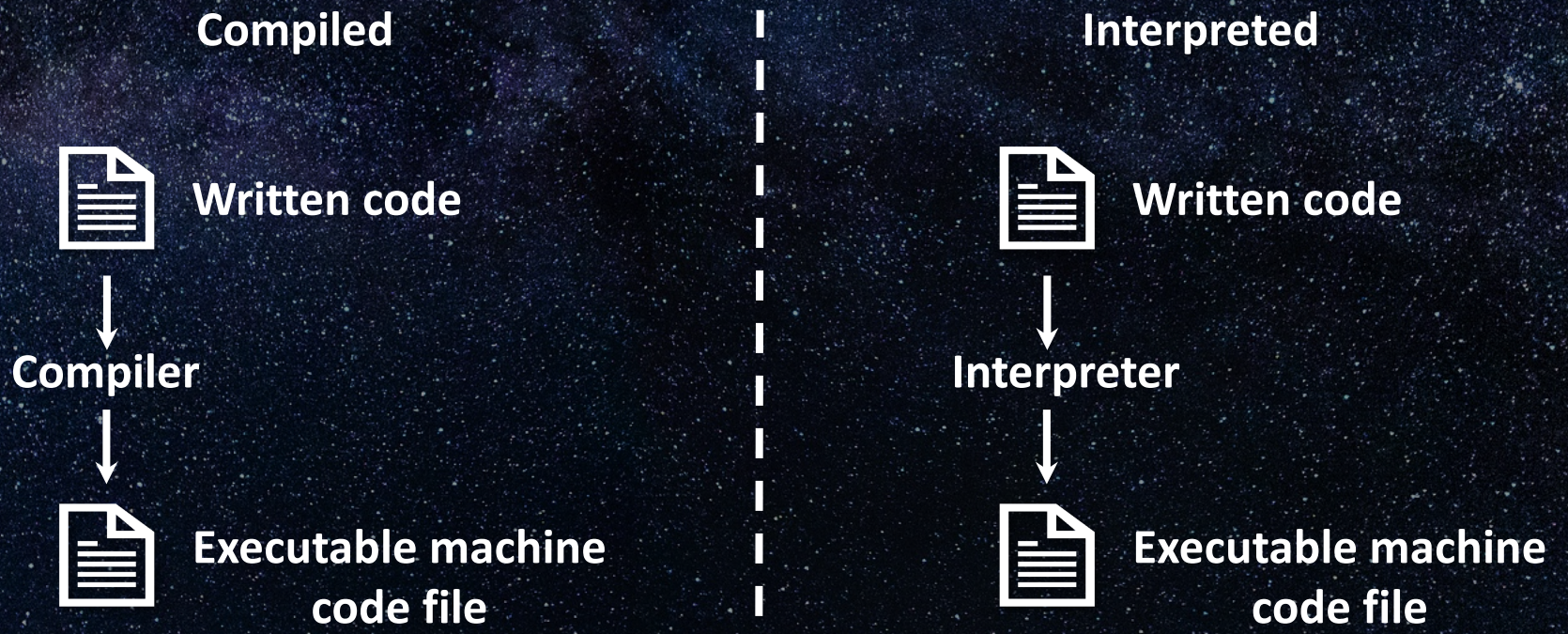
"High Level" Languages

- Easy for humans to read.
- Hard for machines to read.
- Cannot be processed efficiently by CPUs.
- Often written using a combination of human language and numbers.
- The total lines of code required to do simple tasks is small!
- Human language is after all more expressive.

Edge Hill University

# Translating to Machine Code

- Machine code clearly doesn't need translating for the CPU to understand it.
- All other programming languages must be translated. There are two main approaches for doing this.



Edge Hill University

# Python

- Python is an interpreted programming language.
- When you run Python code in code academy, a python interpreter reads each line, converts it to machine code, and this is run by the CPU.
- You've been learning Python 3. Thus, you've been using an interpreter capable of translating Python 3 code to machine code.
- Programming languages improve over time. If you wrote code Python 2 code, a Python 3 interpreter would *not* understand it. Thus ,you'd get errors.
- What's important to understand here:
  - Python is interpreted, you therefore need an interpreter to run Python code.
  - You must use an interpreter that understand the code your writing.
  - Any computer with a suitable Python interpreter will be able to run your code.

Edge Hill
University

# To the Colab!

Edge Hill University

# Questions & Thank For Listening

**@scienceguyrob** **robert.lyon@edgehill.ac.uk**

## Acknowledgements


- Uncredited images obtained from https://pixabay.com (Creative Commons License, no attribution required).


## More Information:

https://github.com/scienceguyrob/AI4AstroMasterclass