

CA318

Advanced Algorithms and AI Search

Introduction to Learning: Nearest Neighbours & Identity Trees



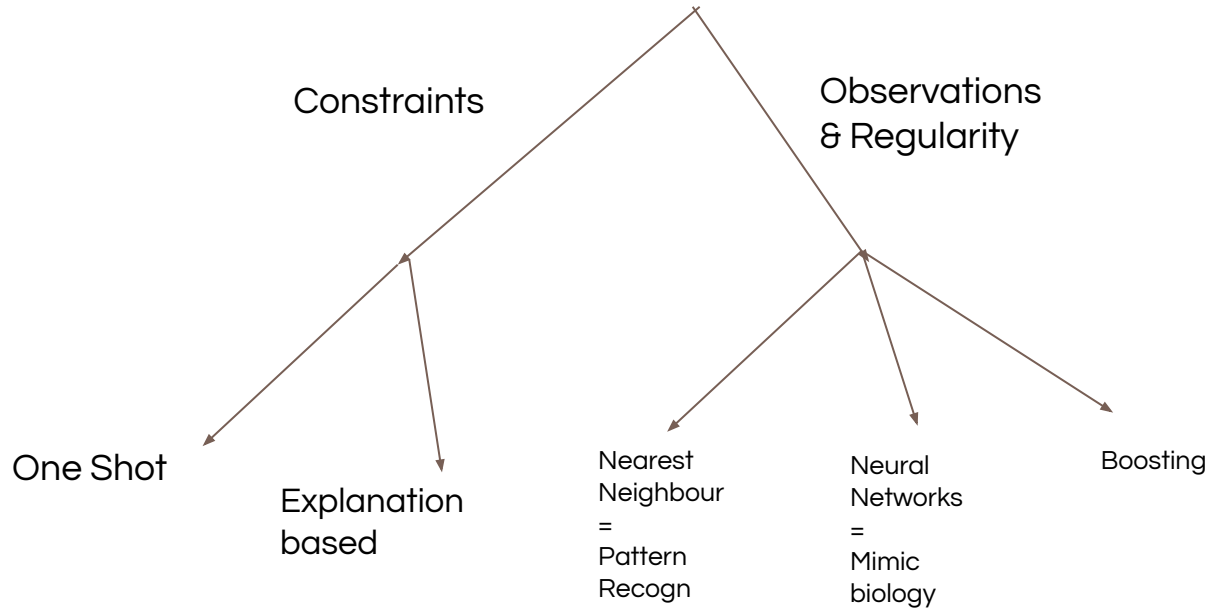
Learning: Part 1

Nearest neighbours; Decision Boundaries

Types of Learning

- Of course, our objective is always to understand how humans learn so well
- We can then mimic this approach
- The objective being to make a machine as capable as learning as a human is
- Our first step then is to characterize learning methods

Types of Learning



Nearest Neighbour Learning

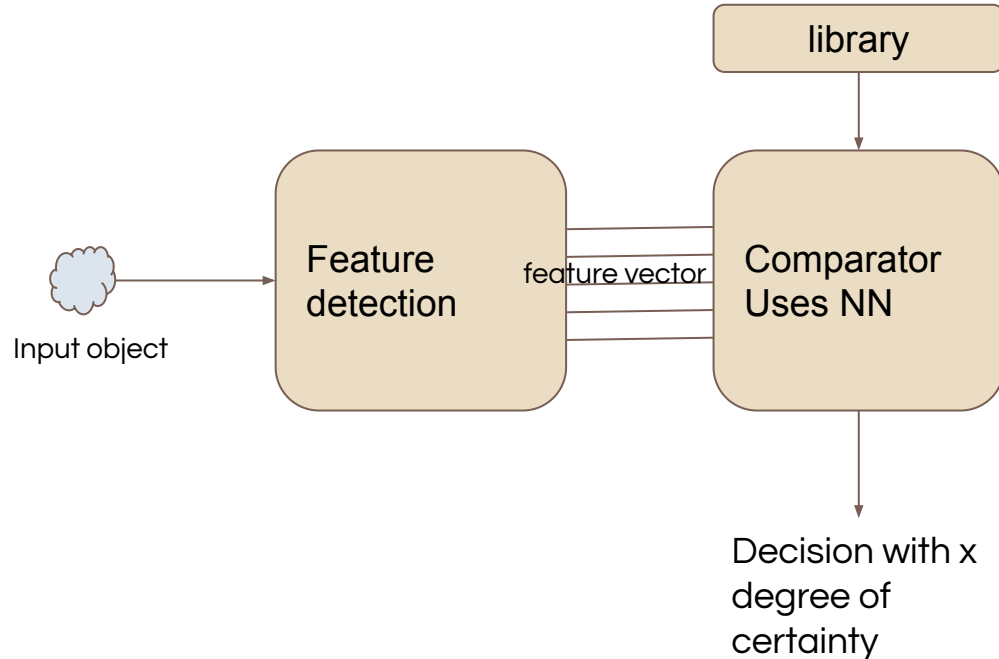
The key components of NNL are:

- Feature Detection
- Feature Comparator
 - Library of vectors
- Recognition with **x** degree of certainty

Decision Flowchart

An input object is scanned and the key features are computed in the form of a feature vector.

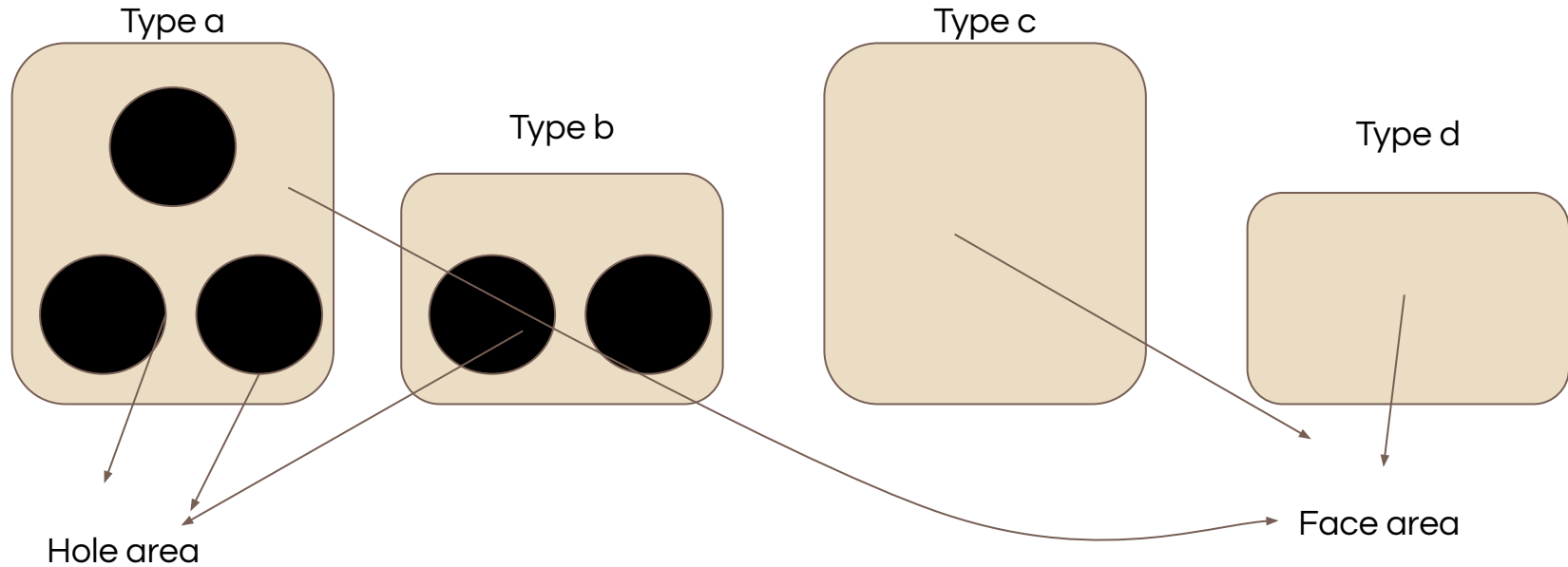
The vector is passed to a comparator, which computes a similarity measure based on NN distance, and a decision with x degree of certainty is computed



Example

- To make this process more concrete, let's consider the situation of a company that makes widgets.
- These widgets have two basic features
 - Area of the widget face A_f
 - Area of the holes in the widget A_h
- Suppose 4 widgets **a,b,c,d** are created on an assembly line
 - We want to **classify** each widget
 - Then we sort the widgets for packing and distribution

New Type of Widget



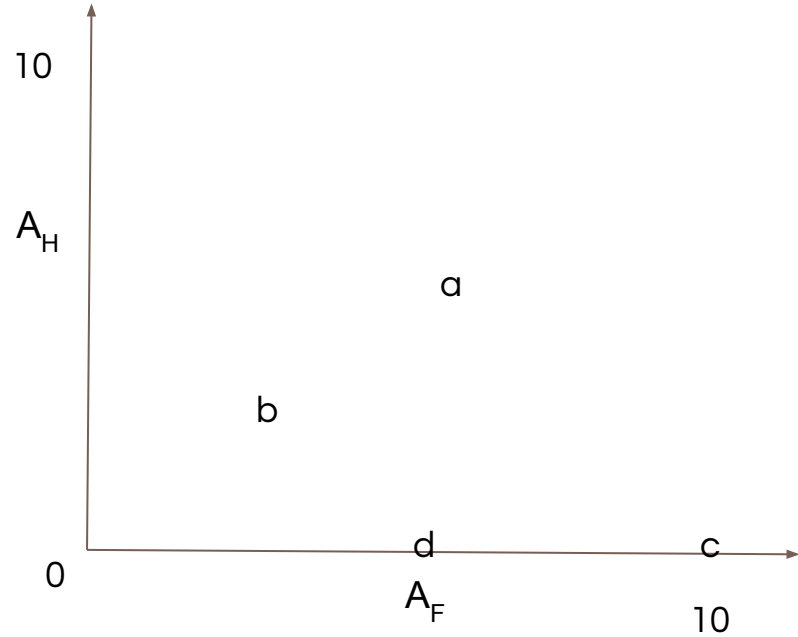
Example Features

Widget feature vector

Widget	A_F	A_H
a	5	5
b	2	2
c	10	0
d	5	0

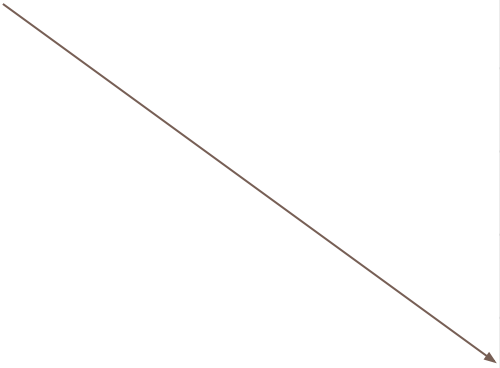
Plotting the vectors

These vectors are 2 dimensional, so we plot these on the standard cartesian plane:



A New Widget **e**

Suppose a new widget (e)
is scanned and we
find its feature vector as
follows:



Widget	A_F	A_H
a	5	5
b	2	2
c	10	0
d	5	0
e	4	1

Classifying **e**

Our first solution is to compute how far **e** is from all the other feature vectors and see which one is its nearest neighbour:

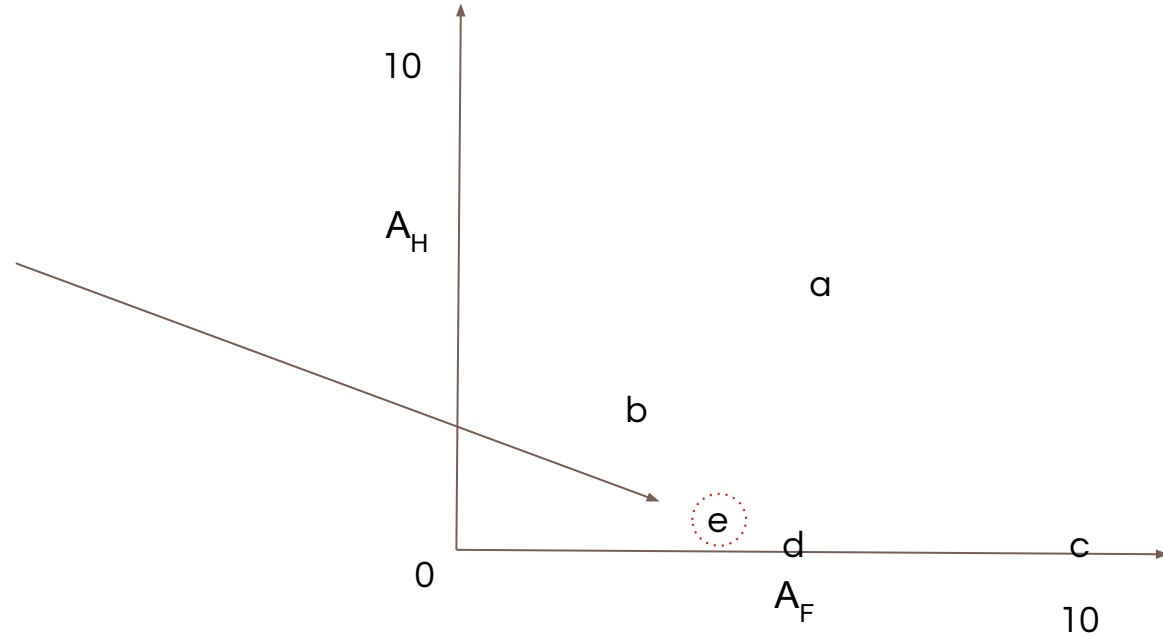
Widget	$A_F(x)$	$A_H(y)$	d_e
a	5	5	4.12
b	2	2	2.23
c	10	0	6.08
d	5	0	1.41
e	4	1	0

$$d_e = \sqrt{(x_i - x_e)^2 + (y_i - y_e)^2}$$

Choose the minimum

Plotting **e**

As you can see, **e** is clearly closest to **d**



Problems with distance d

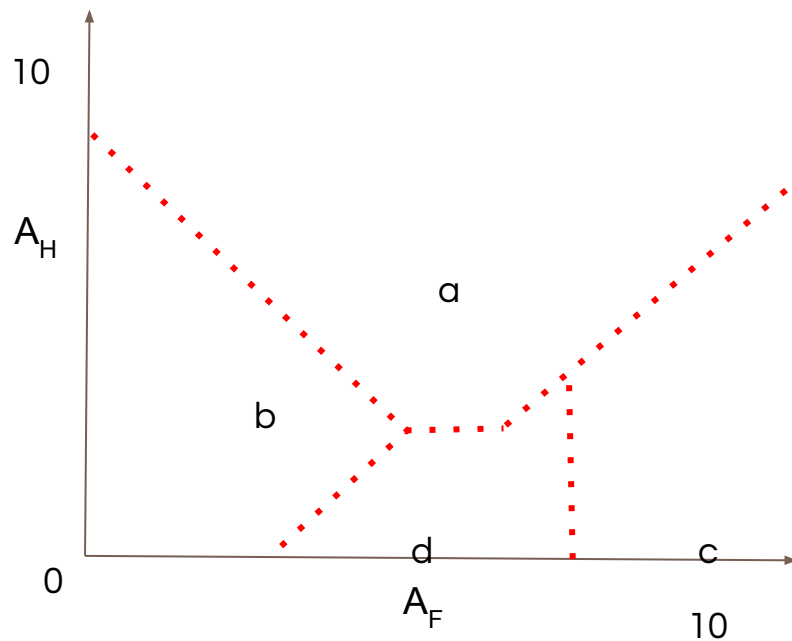
One of the problems with the previous approach, where we used the formula:

$$d_e = \sqrt{(x_i - x_e)^2 + (y_i - y_e)^2}$$

- Is that for the most part, the widgets themselves are not going to be exactly located in the n -dimensional space of features.
- They will have some degree of variance
- So instead of exact points we should use a bounding box around the features instead

Bisect the space

So we construct
perpendicular
bisectors
between each of
the pairs of
features thus:

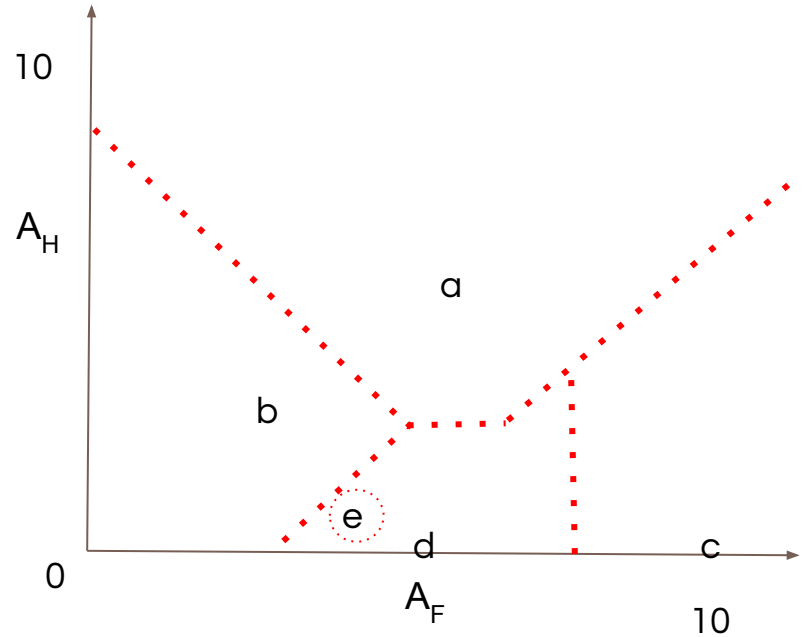


Decision Boundaries

Each of these bisections is called a decision boundary.

Now we have another way to answer the question **which widget does e belong to...** what is it?

Do we get the same answer? - **yes**



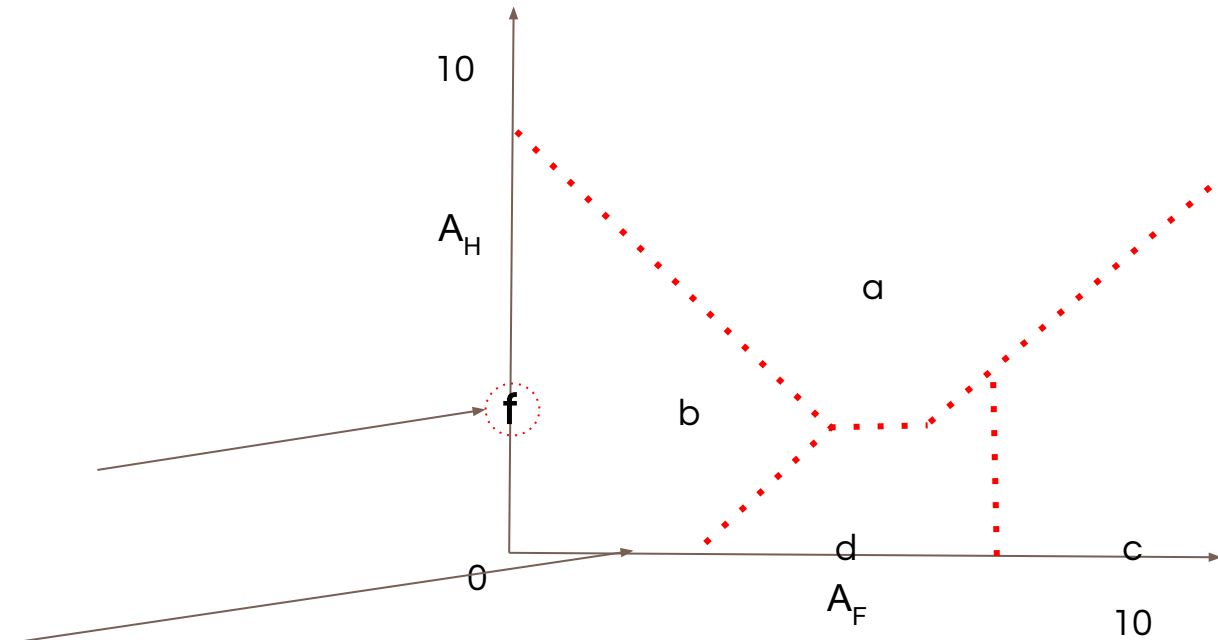
Another widget! (**f**)

- Suppose yet one more widget is scanned but we instead of getting a complete vector of information, we only get the A_H data
- Suppose it turns out that for the widget **f** $A_H = 2$
- What could we say about A_F ?
- We could say that
 - “If something is similar in some respects, its likely to be similar in other respects”
 - So therefore, [weakly] we could say for **f** that $A_F=2$

Computing A_F for \mathbf{f}

“If something is similar in some respects, its likely to be similar in other respects”

=> the area of the face of \mathbf{f} is likely to be 2



Education and Learning

- Medical cases
- Legal cases
- Business cases
- -etc-

We will often apply the principle: “If something is similar in some respects its likely to be similar in other respects”

This is one of the key features of education and learning



Throwing a ball

Its easy...right?

Newton's 2nd Law

In order to compute the force required to throw a ball we could reference Newton's 2nd Law:

Constant Mass

For objects and systems with constant mass,^{[10][11][12]} the second law can be re-stated in terms of an object's acceleration.

$$\mathbf{F} = \frac{d(m\mathbf{v})}{dt} = m \frac{d\mathbf{v}}{dt} = m\mathbf{a},$$

But for a computer to really throw an actual ball, the maths is much more complicated

Learning by maths models

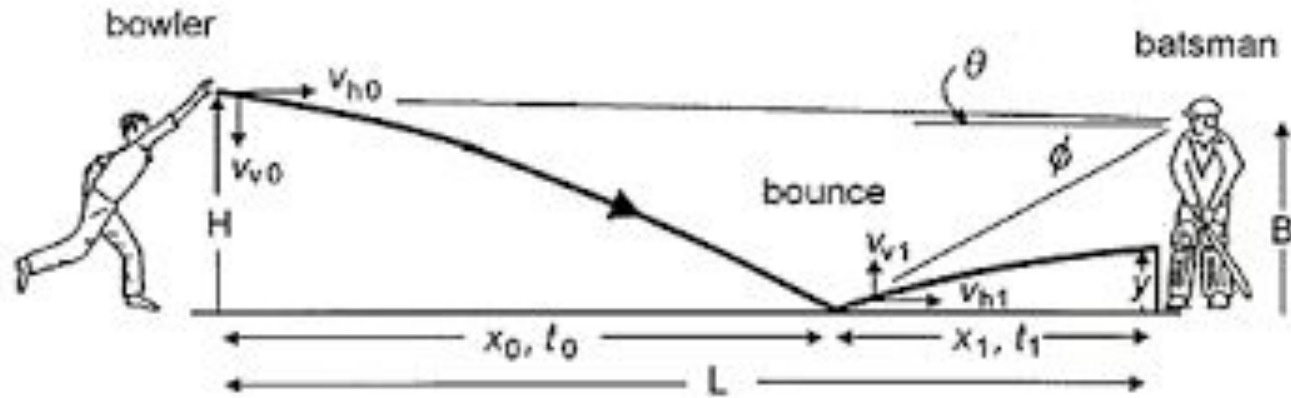
The equation to model how a computer controlled arm should throw a ball a certain distance is given to us by this rather complex mathematical formula

$$\begin{aligned}\tau_1 = & \ddot{\theta}_1 \left(I_1 + I_2 + m_2 l_1 l_2 \cos \theta_2 + \frac{m_1 l_1^2 + m_2 l_2^2}{4} + m_2 l_1^2 \right) \\ & + \ddot{\theta}_2 \left(I_2 + \frac{m_2 l_2^2}{4} + \frac{m_2 l_1 l_2}{2} \cos \theta_2 \right) \\ & - \dot{\theta}_2^2 \frac{m_2 l_1 l_2}{2} \sin \theta_2 \\ & - \dot{\theta}_1 \dot{\theta}_2 m_2 l_1 l_2 \sin \theta_2, \\ \tau_2 = & \ddot{\theta}_1 \left(I_2 + \frac{m_2 l_1 l_2}{2} \cos \theta_2 + \frac{m_2 l_2^2}{4} \right) \\ & + \ddot{\theta}_2 \left(I_2 + \frac{m_2 l_2^2}{4} \right) \\ & + \dot{\theta}_1^2 \frac{m_2 l_1 l_2}{2} \sin \theta_2.\end{aligned}$$

Learning by maths models

- But when you were a child and learned to throw a ball, you did not do it by referencing a Newton's 2nd Law,
 - Or the more complex mathematical model
- So how do we do learn how to it?
- Humans learn by subconsciously populating a table of values from their experience
- Each time we learn a new experience we store this in a table
- When we want to reproduce the result, we lookup the table

Cricket Time



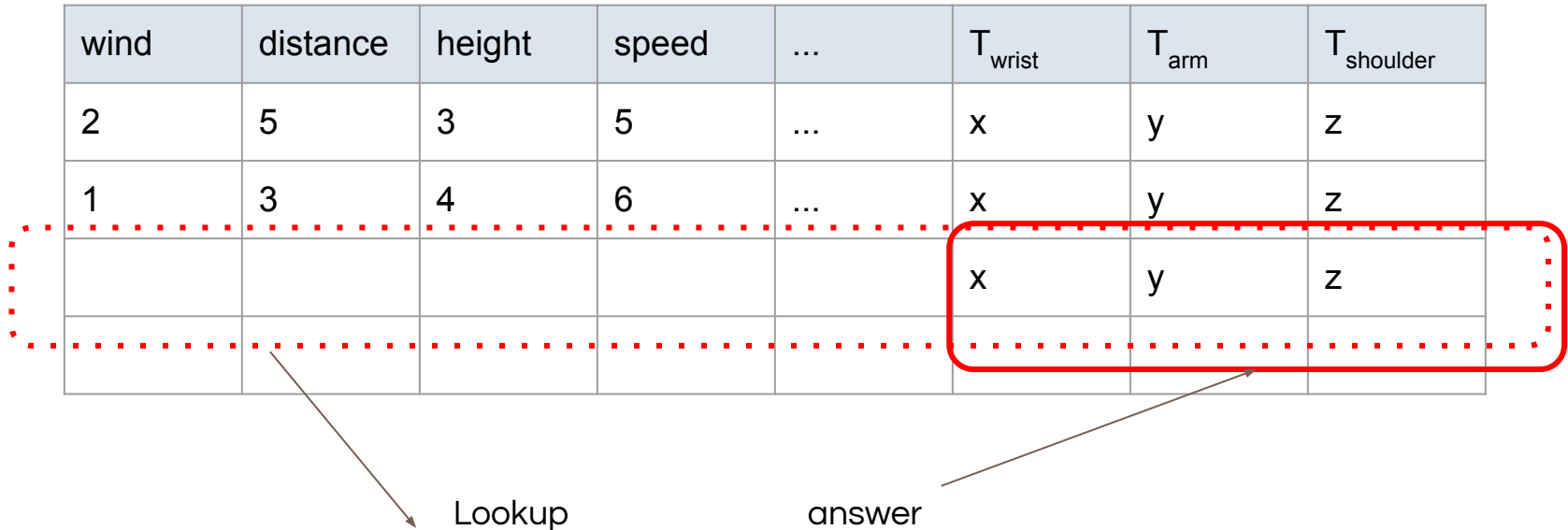
Bowler Parameters

This would be a huge table for a professional athlete

wind	distance	height	speed	...	T _{wrist}	T _{arm}	T _{shoulder}
2	5	3	5	...	x	y	z
1	3	4	6	...	x	y	z
					x	y	z

Lookup

answer



The diagram shows a table with 8 columns: wind, distance, height, speed, ..., T_{wrist}, T_{arm}, T_{shoulder}. The first three rows contain numerical data. The fourth row is highlighted with a red dotted border. The fifth row is highlighted with a red solid border. An arrow labeled 'Lookup' points to the 'distance' column of the fifth row. An arrow labeled 'answer' points to the 'T_{arm}' column of the fifth row.

Look up the table

- Traditionally, it was felt that these parameter tables would be too huge for quick computer look up
- So the idea of having the computer learn parameters was considered too costly in space and time
 - But nowadays we can store infinite quantities of data with very little latency in look ups
- So we can teach a computer to do something by having it store previous successful vales
 - And just look them up

Tables are huge

- Realistically, such tables of parameters could be 10^{12} in size
- But our brains are capable of storing at least 10^{14} so of course humans can comfortably do this.

Q: How much SSD storage is needed to store a 10^{12} table of bytes? - is it feasible on a desktop computer?

Complexity - again

What would be the complexity of a table look up with n rows, assuming we had m parameters?

- Bad design: ?
- Good design: ?



Learning: Part 2

Identification Trees, Disorder and Entropy

Nearest Neighbours again

Nearest neighbour calculations is perfectly feasible if we can create a numeric vector of features:

v is a vector of features
in 4-dimensional space

$$v = \begin{pmatrix} 3 \\ 4 \\ 1 \\ 7 \end{pmatrix}$$

Feature Space

- The problem here is that many features sets cannot be adequately described in terms of vectors of integers
- Remember, each integer in \mathbf{v} is a point in n -dimensional space
- So in nearest neighbours we can easily subtract the \mathbf{v}_i and \mathbf{u}_i to get the distance between the i -th point in \mathbf{u} and \mathbf{v}
- But many feature sets cannot be described numerically like this

Problem

- Suppose we wish to identify an airplane based on characteristics.
- Specifically, we want to know the important characteristics needed to identify a Boeing 787
- Lets suppose there is some uncertainty
 - For example, suppose different iterations of the B787 had somewhat different characteristics
- Suppose we visit the airport and collect some data on airplanes

Airplane Feature Set

Narrow Body	Engine Shape	Winglets	Engine Noise	B787
Yes	cerated	?	quiet	No
Yes	straight	Yes	quiet	No
No	straight	?	quiet	Yes
No	curved	No	loud	Yes
No	curved	?	medium	Yes
No	cerated	Yes	loud	No
No	straight	Yes	loud	No
Yes	curved	?	medium	No

No distances here

Clearly we cannot do any form of distance measurements - but we still want to perform recognition of a **B787 with some degree of certainty.**

How do we recognize here?

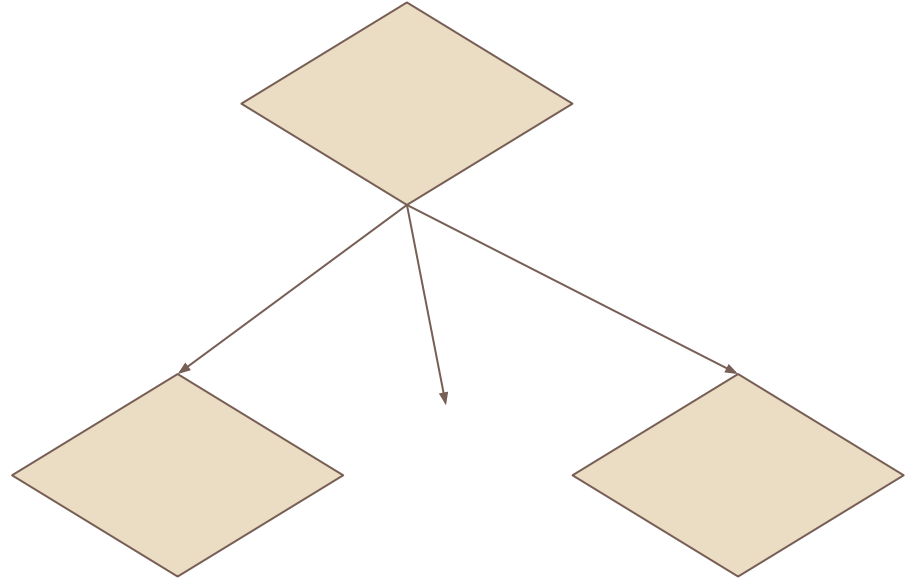
- We want to recognize the B787
- But our data is non numeric
- Therefore distance metrics from NNL does not apply
- We also have data where the answer is unknow (?)
- Some features are more significant than others
- Cost - collecting some of these data is more costly than others
 - eg: Engine noise
- What can we do?

Tests

- Each of the features in the database can be considered a **test** of some kind
- We take our samples, and we arrange our tree of tests
- We process each sample and test the outcome
 - **We rate each test according to the number of individuals it puts into homogeneous sets**
 - We pick the test with the best rating
- We branch according to the outcome of the tests

Identification Tree

We arrange the tests like this - and check each sample against the logic. This is called an **identification tree**



Identification Tree

We need to make a **good** identification tree

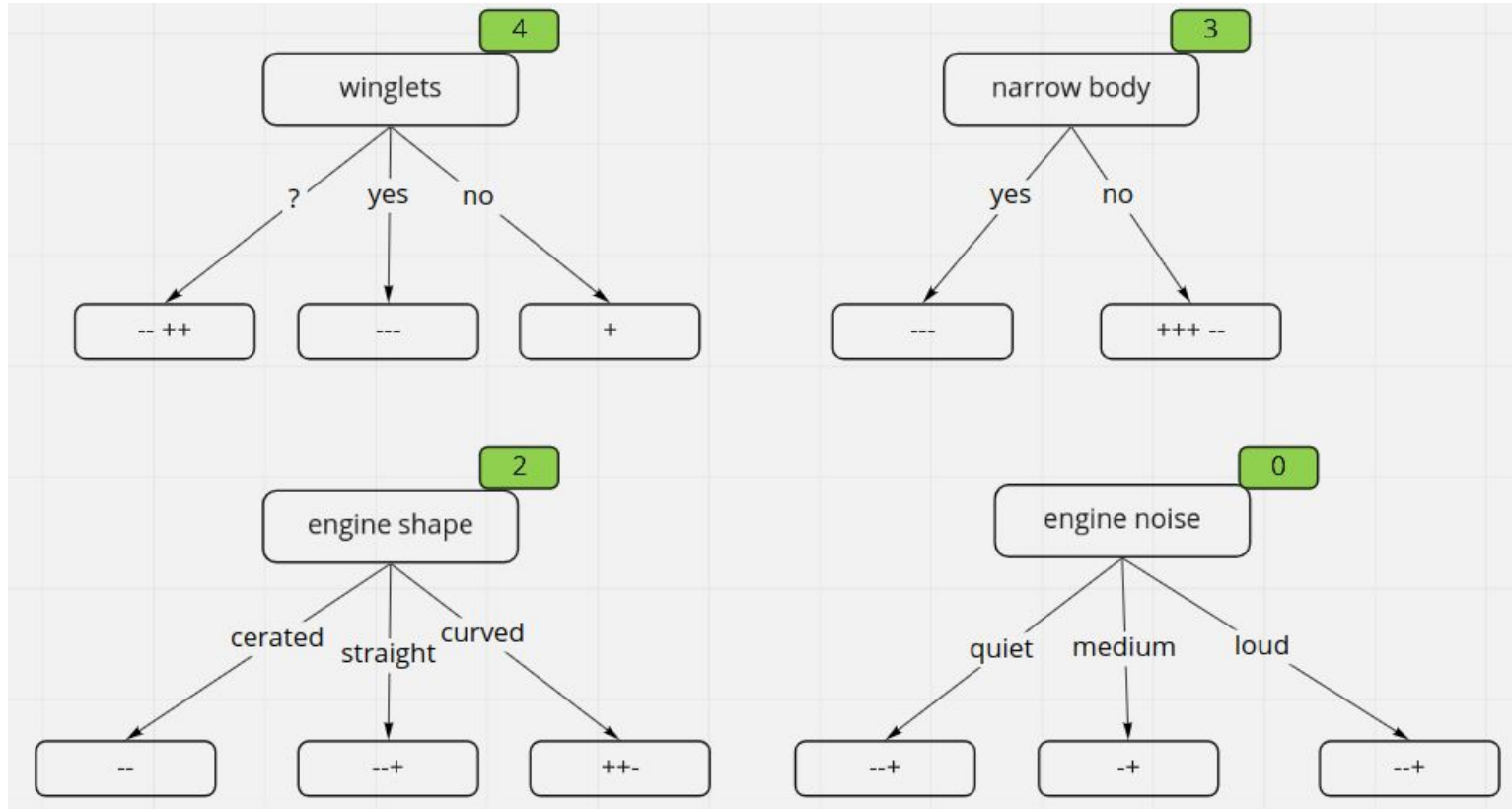
- Small
 - Occams Razor
- Low cost
- Uniform subsets at leaf level
 - At the leaf level, we should have all the B787 aircraft identified together
 - And all the non B787 aircraft identified together

Brute force or Heuristic Methods

- Even though the dataset shown is tiny, in general, these trees cannot be enumerated in a brute force way
- Because trees are generally speaking exponential problems
- So we need a better approach
- Perhaps, a **heuristic**** approach

**of course, heuristics can be dangerous

Each Test with Homogeneous Count



Scores

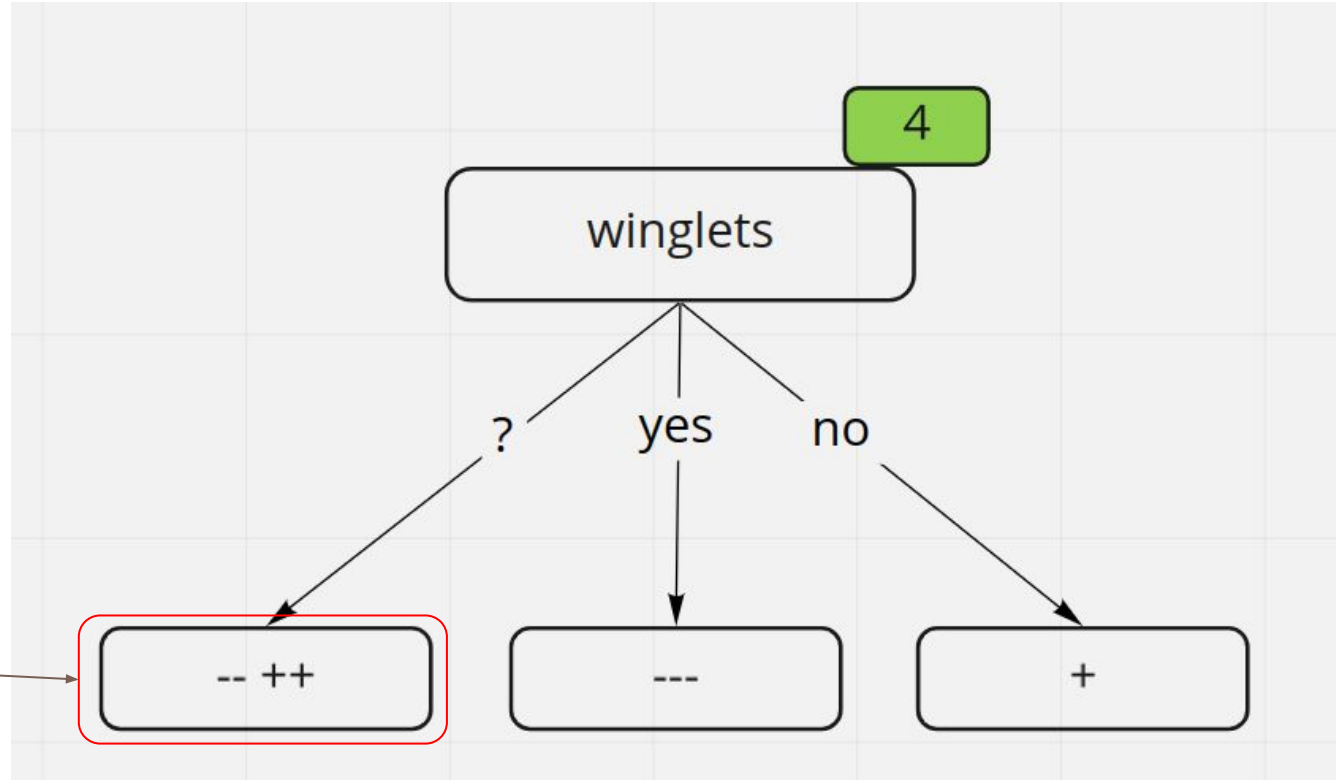
- Winglets: 4
- Narrow Body: 3
- Engine Shape: 2
- Engine Noise: 0

The **winglets** test produced the most homogeneous solution

Identity Tree 1/2

So we now start with the winglets test, eliminate **yes** and **no** rows from the table, and iterate again

non-uniform

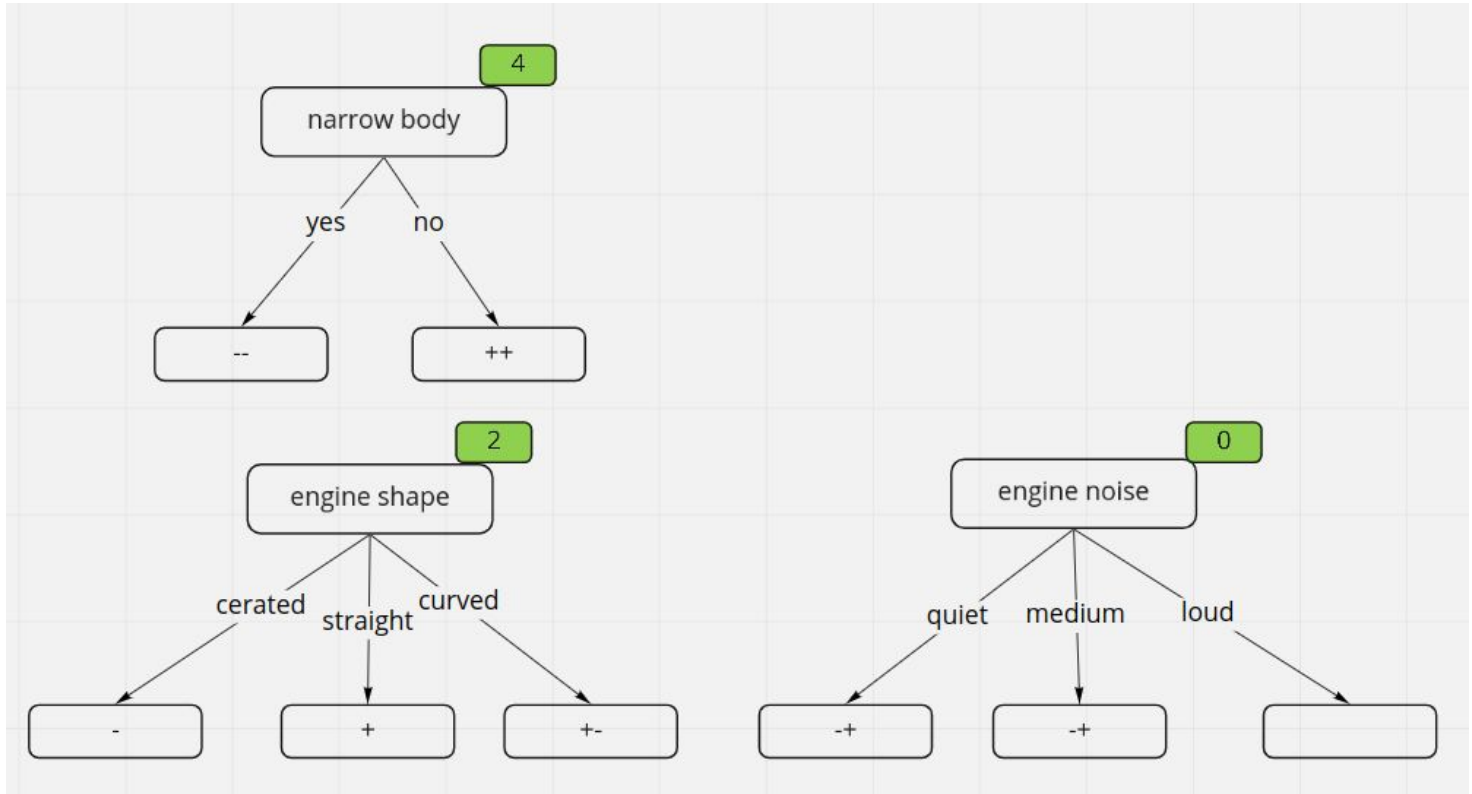


Reduce the test table

Narrow Body	Engine Shape	Winglets	Engine Noise	B787
Yes	cerated	?	quiet	No
Yes	straight	Yes	quiet	No
No	straight	?	quiet	Yes
No	curved	No	loud	Yes
No	curved	?	medium	Yes
No	cerated	Yes	loud	No
No	straight	Yes	loud	No
Yes	curved	?	medium	No

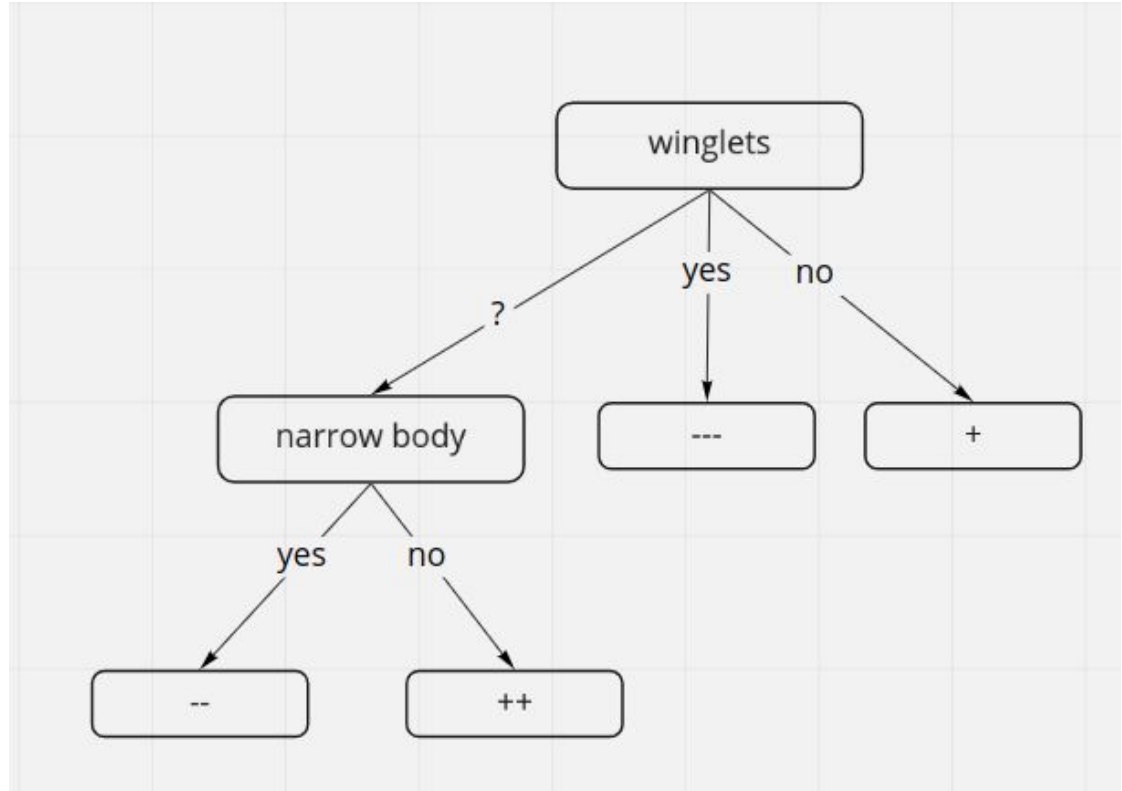
Run the 3 tests again

Select the best test, which in this case is the **narrow body** test



Identity Tree 2/2

Now we have
a measure of
certainty,
which will
guide us in
the
classification
task



Conclusion: Probabilities

Winglets=**Y**:

$P(3/8)$ it is not B787

Winglets=**N**:

$P(1/8)$ it is B787

Winglets=**?** & Narrow Body=**Y**

$P(2/8)$ it is not B787

Winglets=**?** & Narrow Body=**N**

$P(2/8)$ it is B787



Disorder

Working with large data sets

Large data sets

- One of the problems we face when dealing with a large data set is that of how do we start?
- How do we find a homogeneous partition in a large data set
- This is not easy to do
- We can use a measure called **disorder**

Information Theory: Set Entropy

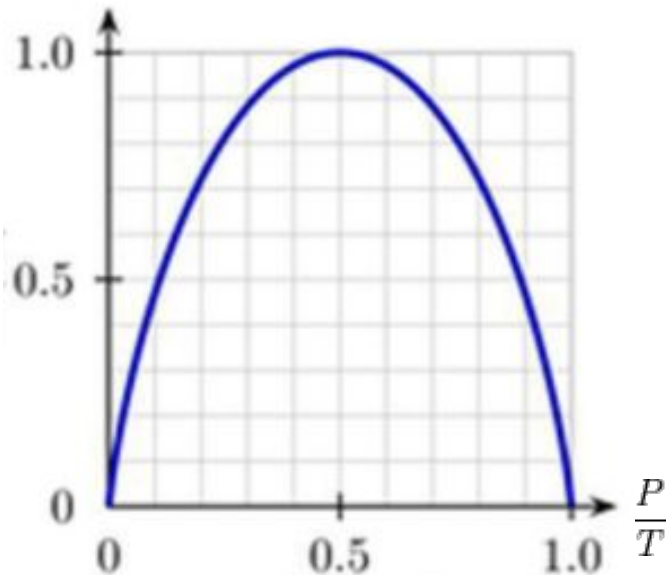
$D(S)$ = disorder of a set S :

$$D(S) = -\frac{P}{T} \log_2 \frac{P}{T} - \frac{N}{T} \log_2 \frac{N}{T}$$

P = total positives in S

N = total negatives in S

T = total in S



Finally

When we use set entropy

$$D(S) = -\frac{P}{T} \log_2 \frac{P}{T} - \frac{N}{T} \log_2 \frac{N}{T}$$

Weighted for the number of edges as a percentage of the total in the decision set, we find the same test partition logic as we did for the previous methods.

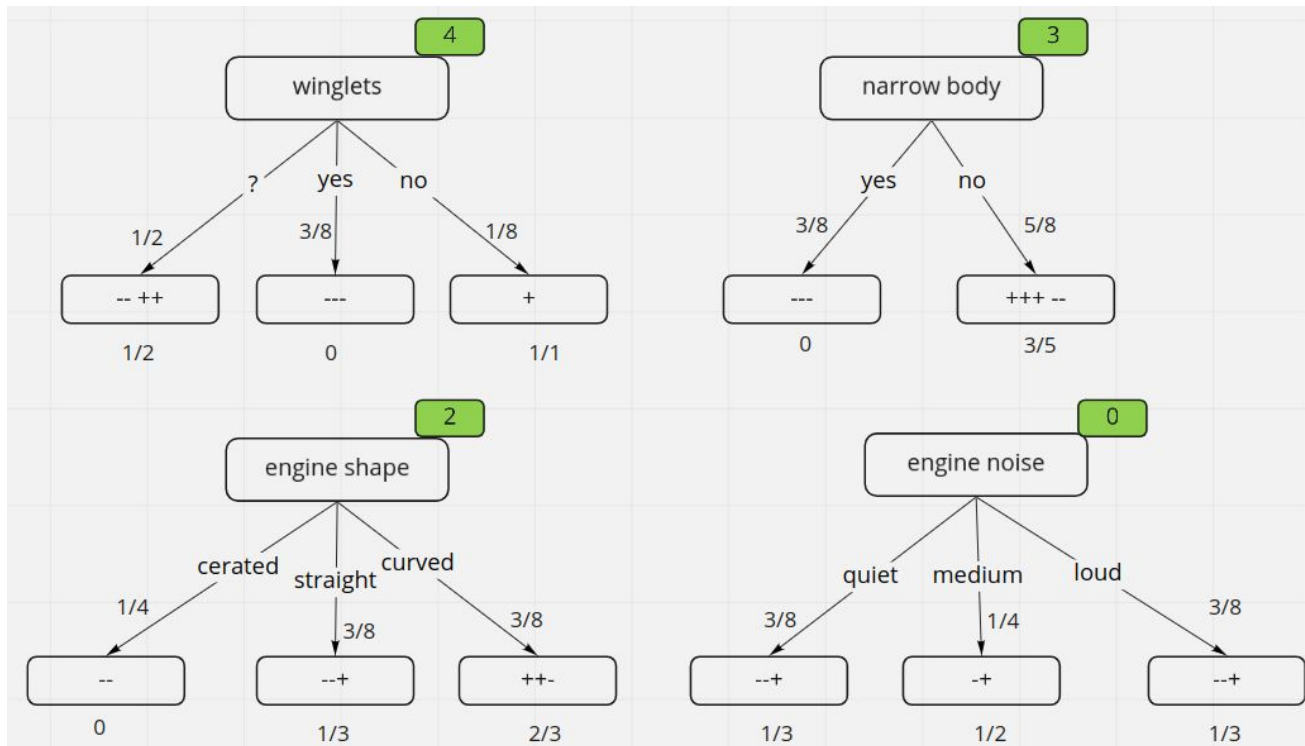
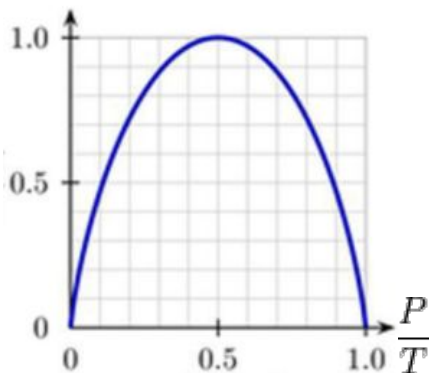
Worked example

- We can now use the density function in the previous example
- to rank our tests and
- validate our decision to choose the **winglets** test
- We will use graph itself rather than the log function calculation

Worked example

Each test outcome is weighted in proportion to the number of elements in the outcome divided by the total number of elements (N)

In the example N=8



Worked example



We will complete this in class

Finally

- When using Identification Trees we need to create tests with the lowest disorder
- For this we use the entropy density function
- Weighted for the number of elements in the set divided by the set size
- This is the best method for bootstrapping Identification Trees

Thank You



Questions