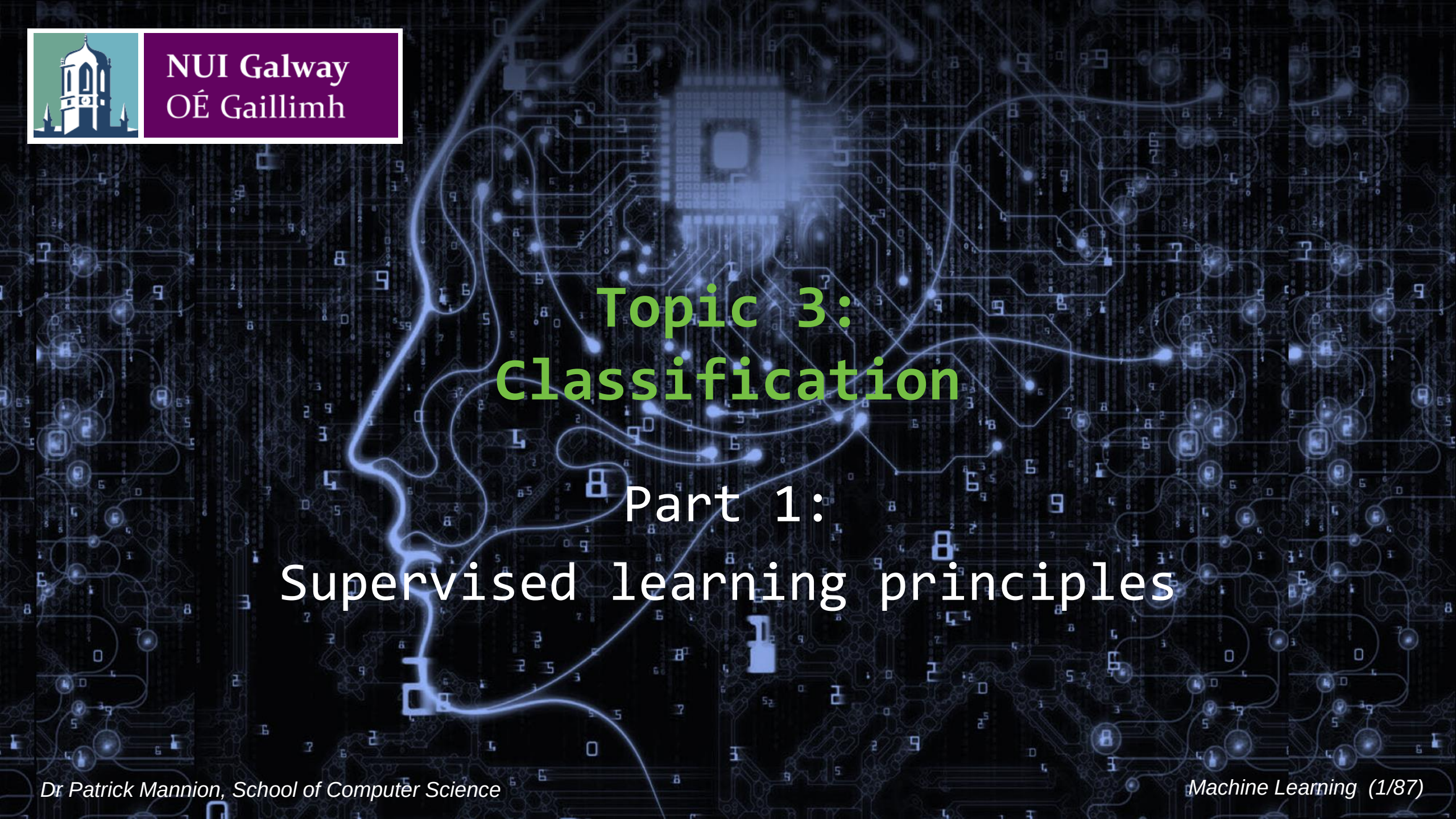


Topic 3: Classification

Part 1: Supervised learning principles

Types of machine learning techniques

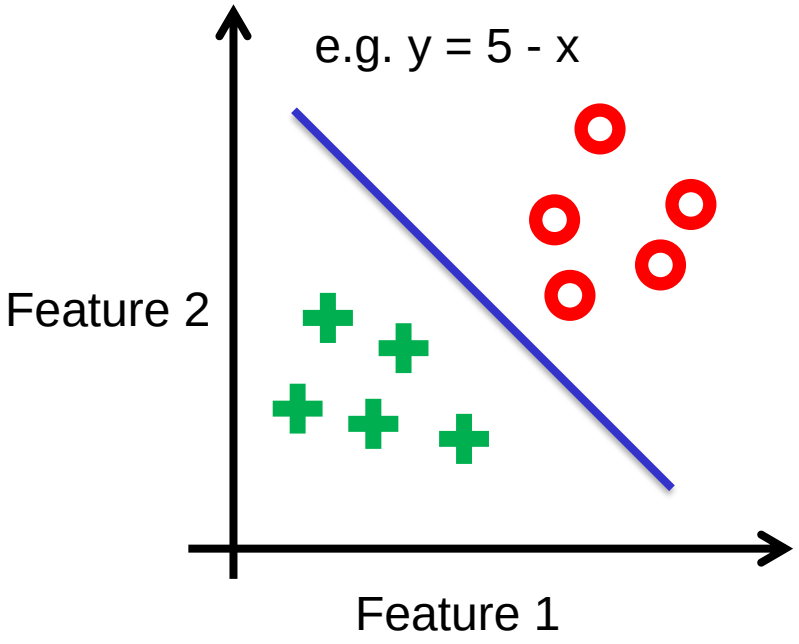
- Recall that in the introduction lecture we saw that there are several main types of machine learning techniques, including **supervised learning**, unsupervised learning, semi-supervised learning and reinforcement learning
- Supervised learning tasks include:
  - **Classification** (the main subject of this lecture)
  - Regression (which we will cover in more detail later)

Supervised learning: motivating examples

1. Estimate sale price of a house, given past data of house sizes, locations and their prices
2. Before unlocking a tablet, determine whether a known user or somebody else is looking at the webcam
3. Decide whether a chemical spectrum of a mixture has evidence of containing cocaine, based on other spectra with & without cocaine
4. Predict concentration of cocaine in mixture
5. Determine whether objects of interest are present in a scene – if so, what are they? (relevant for autonomous vehicles and robotics, among other domains)

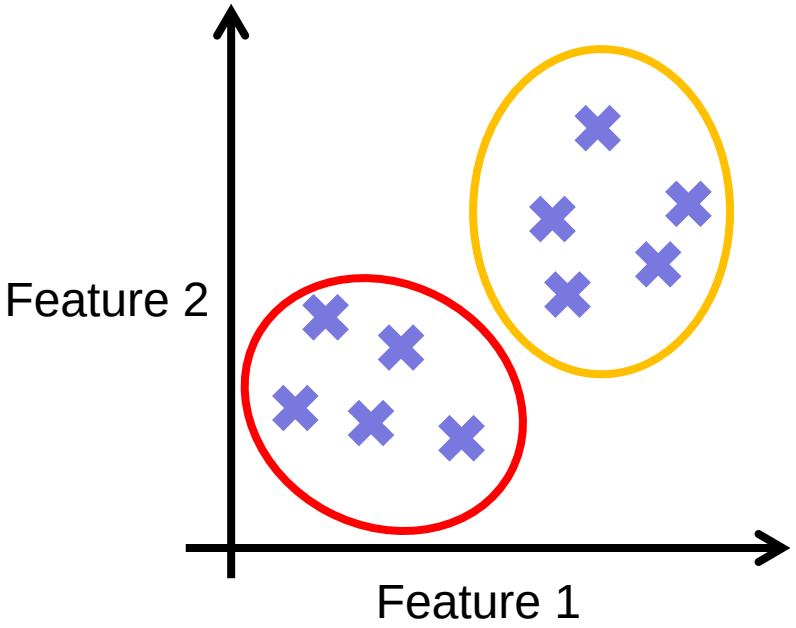
Key feature:
given "right answers" / "ground truth" as start point.
Which tasks are classification, and which are regression?

Supervised vs. unsupervised learning



A classification task

A clustering task



Supervised learning: task definition

- Given examples, return function *h* (*hypothesis*) that approximates some 'true' function *f* that (hypothetically) generated the labels for the examples
  - Have set of examples, the **training data**:
    each has a **label** and a set of **attributes** that have known **values**
  - Consider *labels* (classes) to be *outputs* of some function *f*; the observed *attributes* are its *inputs*
  - Denote the attribute value inputs ***x,*** labels are their corresponding outputs ***f(x)***
  - An example is a pair **(*x, f(x)*)**
  - Function ***f*** is *unknown*; want to discover an approximation of it, ***h***
  - Can use ***h*** to **predict** labels of new data: generalisation

Also known as Pure Inductive Learning – why?

Classification task - training data example

Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
	overcast	cool	normal	true	yes

Task Description

Attributes/ Dimensions

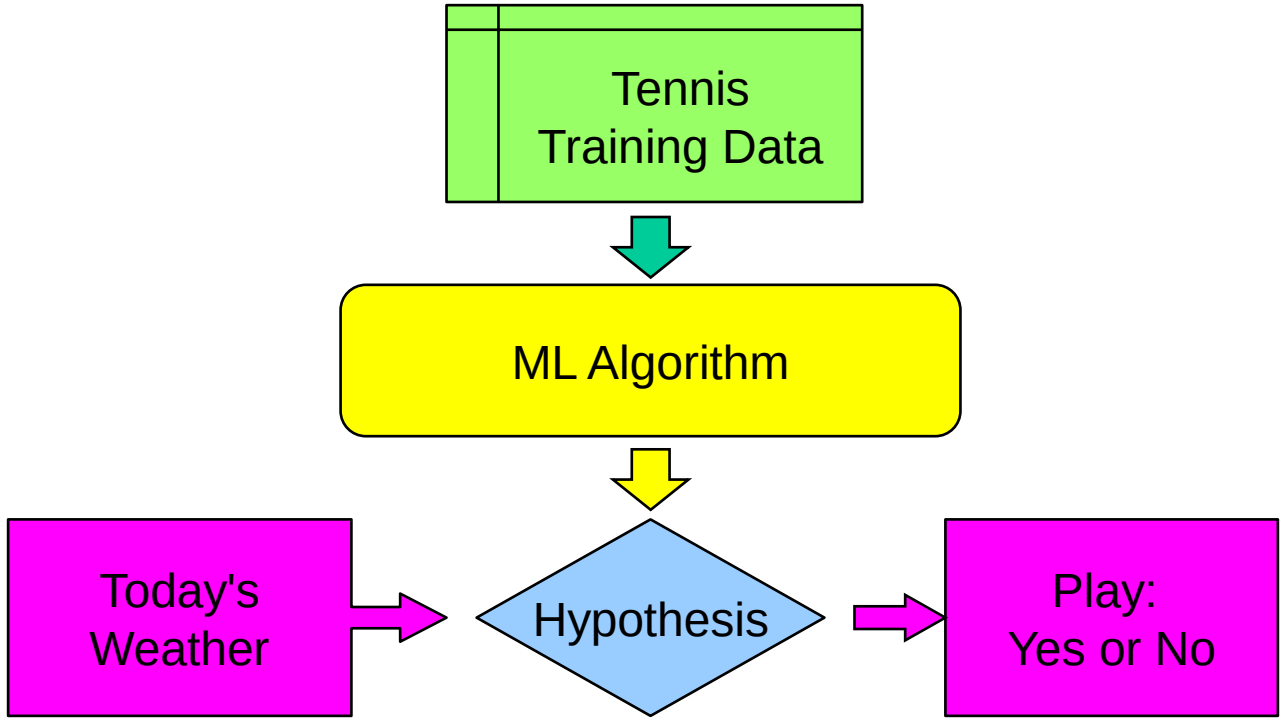
One Training Case

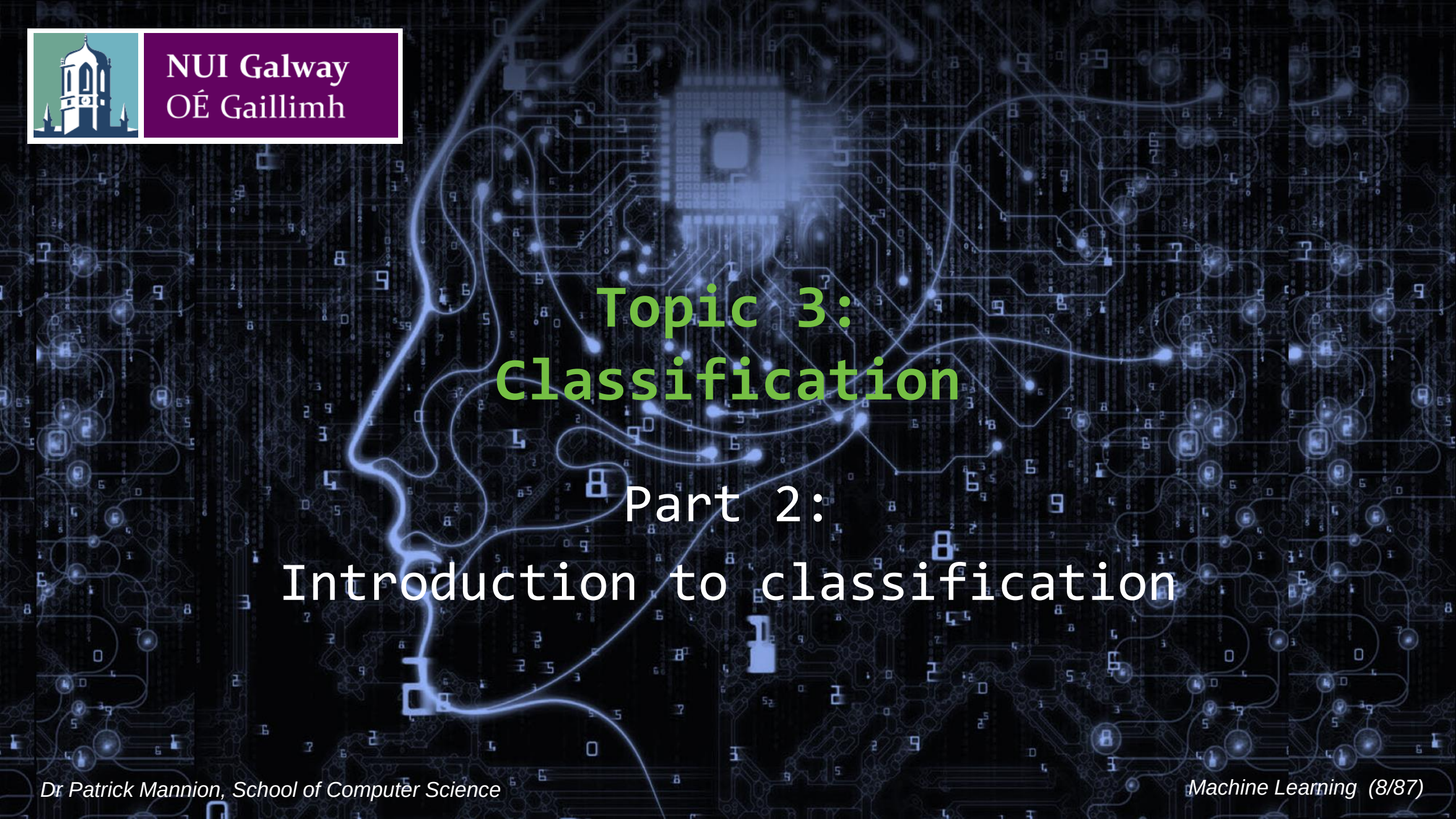
Identifier (not used in learning)

Attribute values = Independent variables

Labels/Classes/Target Attribute = Dependent variable

Overview of the supervised learning process





Topic 3: Classification

Part 2: Introduction to classification

Classification tasks

- The simplest type of classification task is where instances are assigned to one of two categories
- This is referred to as a **binary classification task** / two-class classification task
- Many popular machine learning problems fall into this category:
  - Is cancer present in a scan? (yes / no)
  - Should this loan be approved (yes / no)
  - Sentiment analysis in text reviews of products (positive review / negative review)
  - Face detection in images (face present / not present)
  - …
- The more general form of classification task is **multi-class classification** where the number of classes is >= 3

Example dataset for a binary classification task

- College athletes dataset
  - Two attributes:

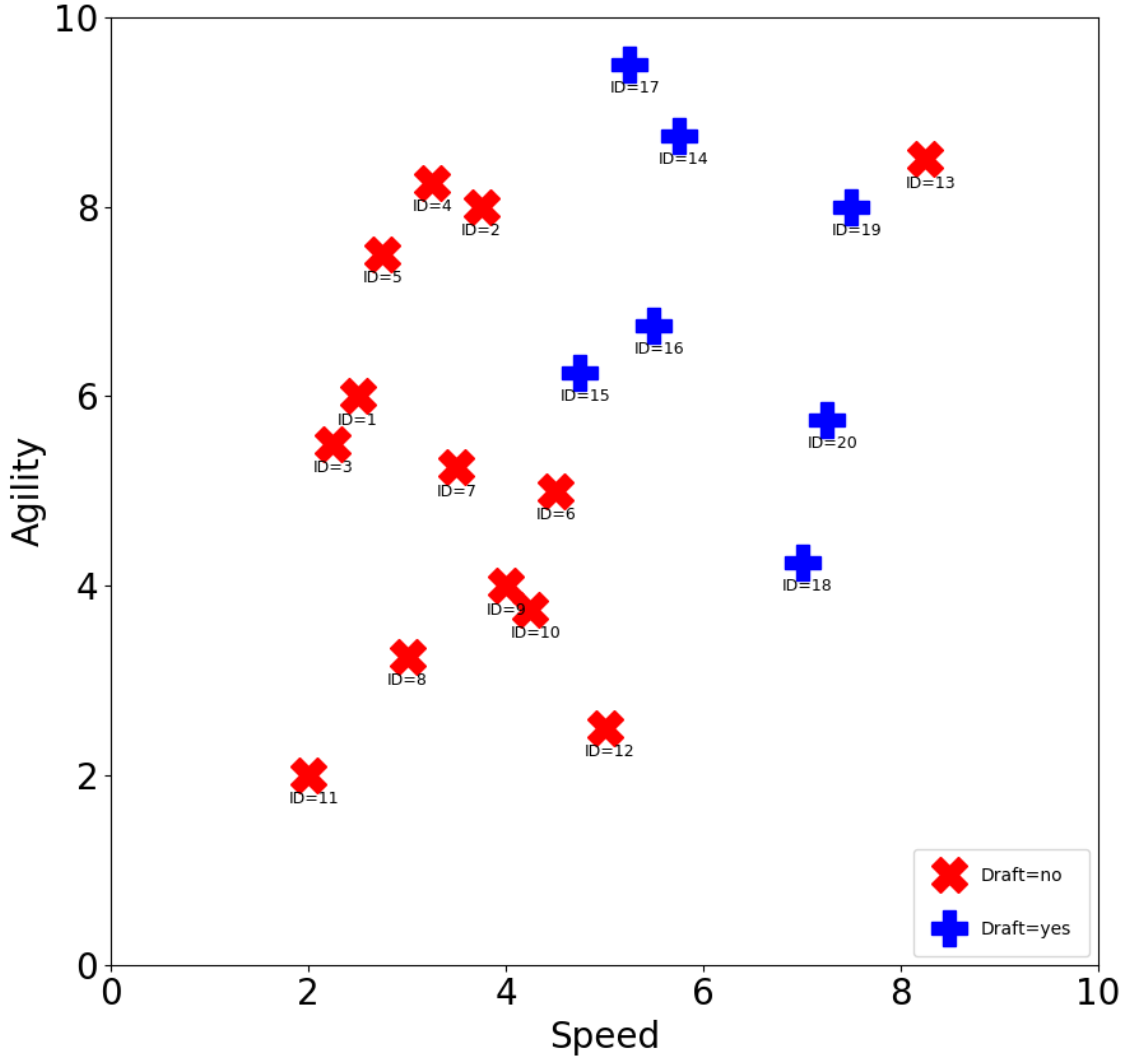
    Speed - continuous variable

    Agility - continuous variable
  - Data on whether or not each college athlete was drafted to a professional team. Target: **Draft** - yes/no
  - 20 examples in dataset
  - See **college_athletes.csv** on Blackboard

- Objective:
  - Build a binary classifier to predict whether a new previously unknown athlete who did not feature in the dataset should be drafted

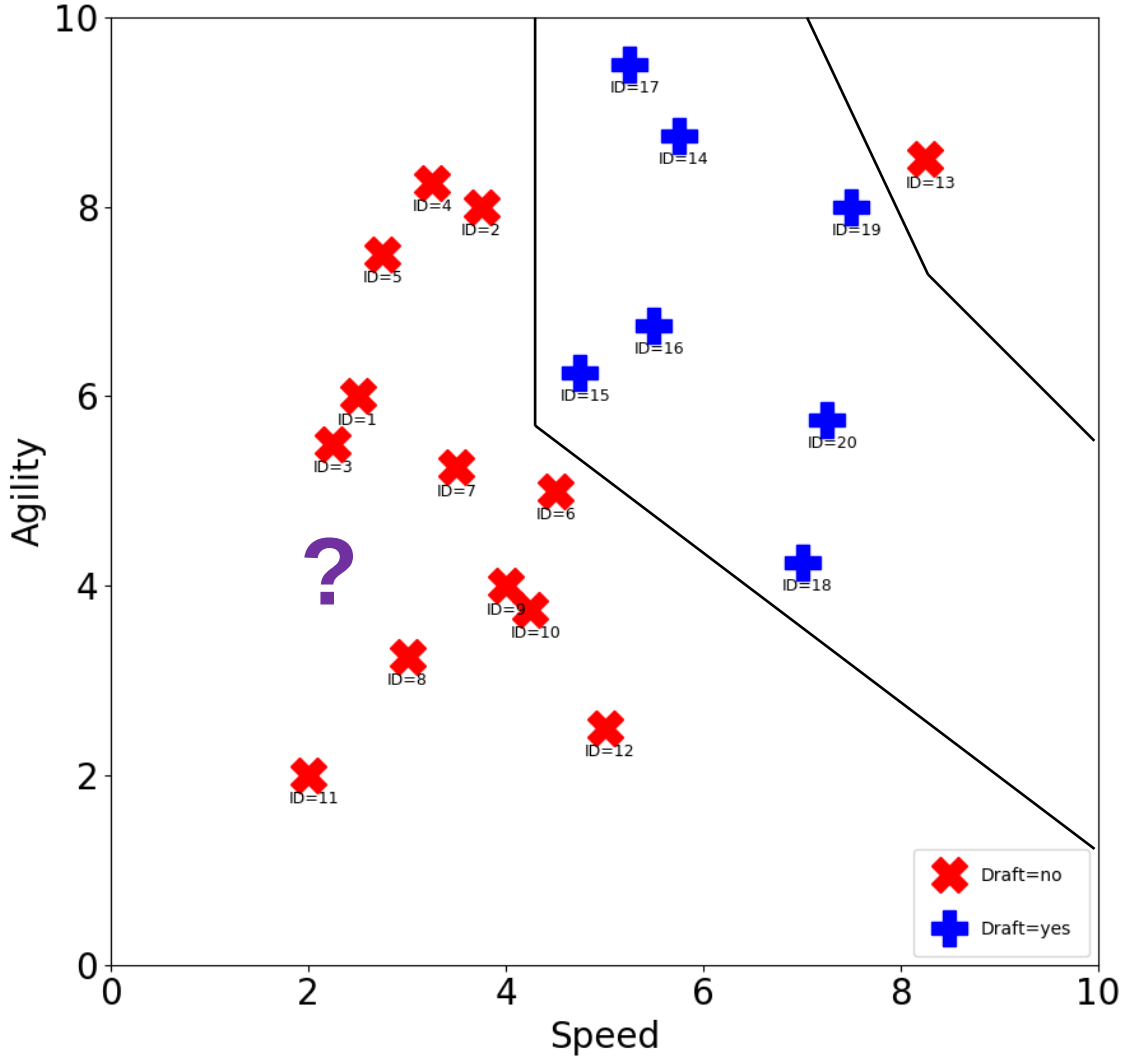
College Athletes			
ID	Speed	Agility	Draft
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

Feature space plot for the college athletes dataset



College Athletes			
ID	**Speed**	**Agility**	**Draft**
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

What is a reasonable decision boundary?



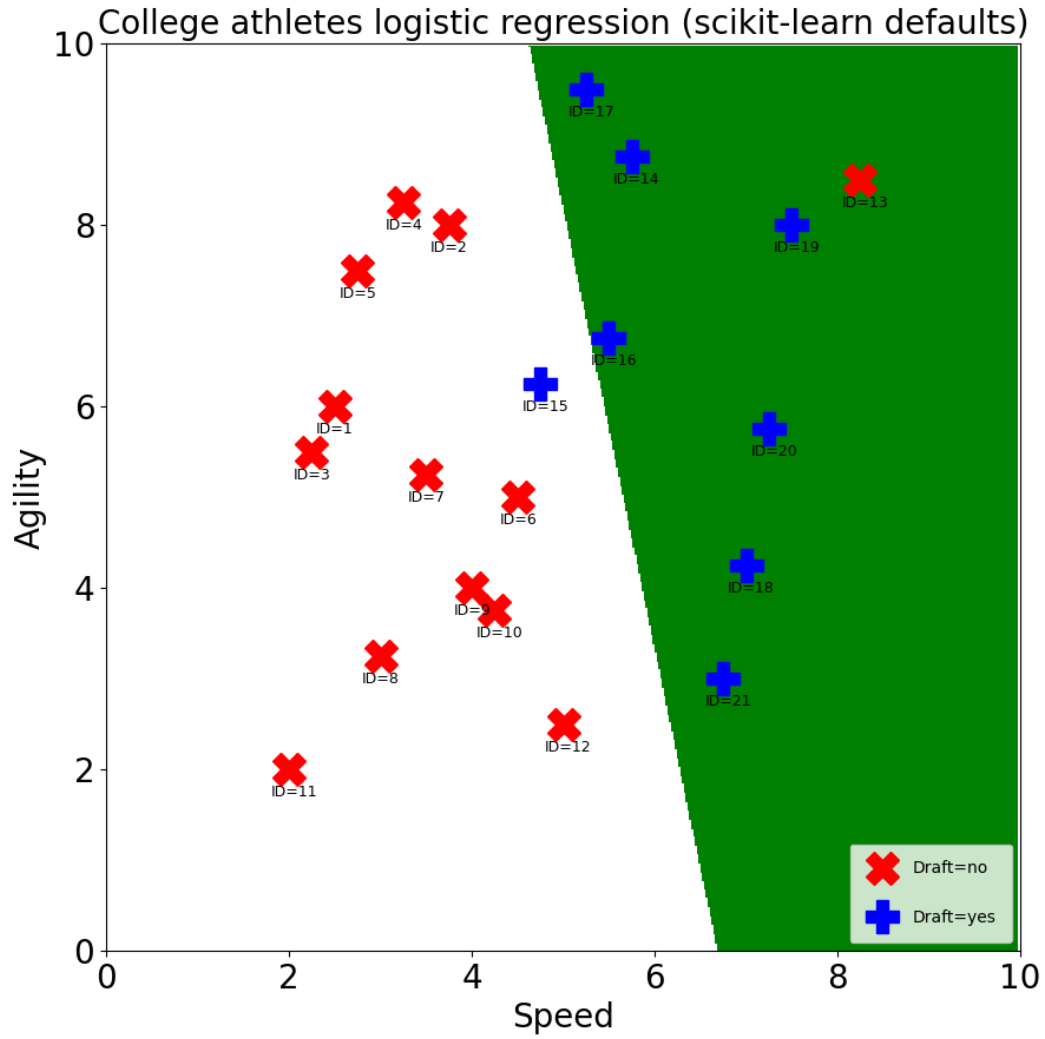
- How to categorise the new unseen example shown with the purple question mark? Need algorithms that will generate a hypothesis / model consistent with the training data
- Is the suggested **decision boundary** shown in thin black lines a good one? It is consistent with all of the training data, but it was drawn manually. In general, it won't be possible to draw such decision boundaries manually when dealing with higher dimensional data (e.g., more than 3 features).

Example classification algorithms

- There are many machine learning algorithms available to learn a classification hypothesis / model

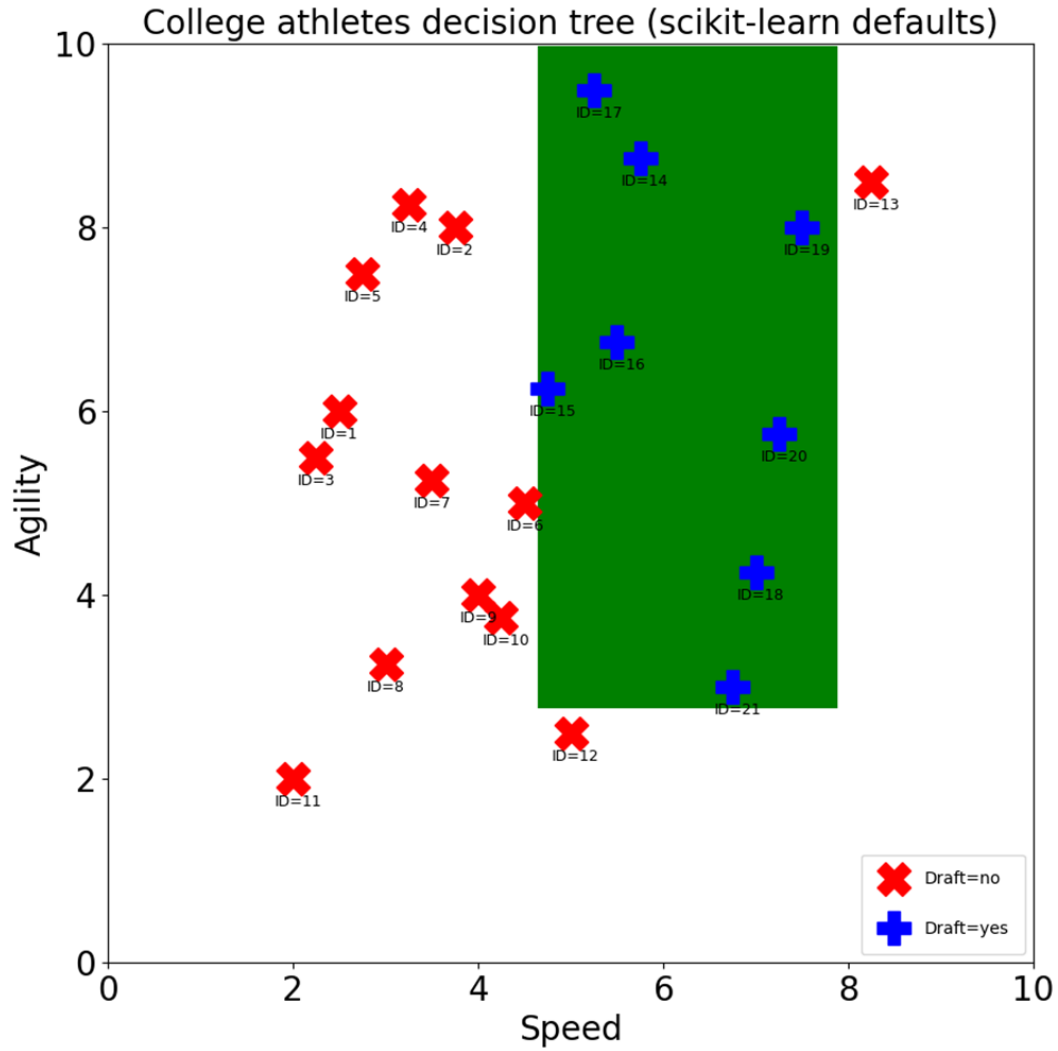
- Some examples (with corresponding scikit-learn classes)
  - *k*-nearest neighbours (e.g., scikit-learn `KNeighborsClassifier`)
  - Decision trees (e.g., scikit-learn `DecisionTreeClassifier`)
  - Gaussian processes (e.g., scikit-learn `GaussianProcessClassifier`)
  - Neural networks / multi-layer perceptron (e.g., scikit-learn `MLPClassifier`)
  - Logistic regression (e.g., scikit-learn `LogisticRegression`)
    - N.B. Logistic regression, despite its name, is a linear model for classification rather than regression.
  - …

Logistic regression on the college athletes dataset



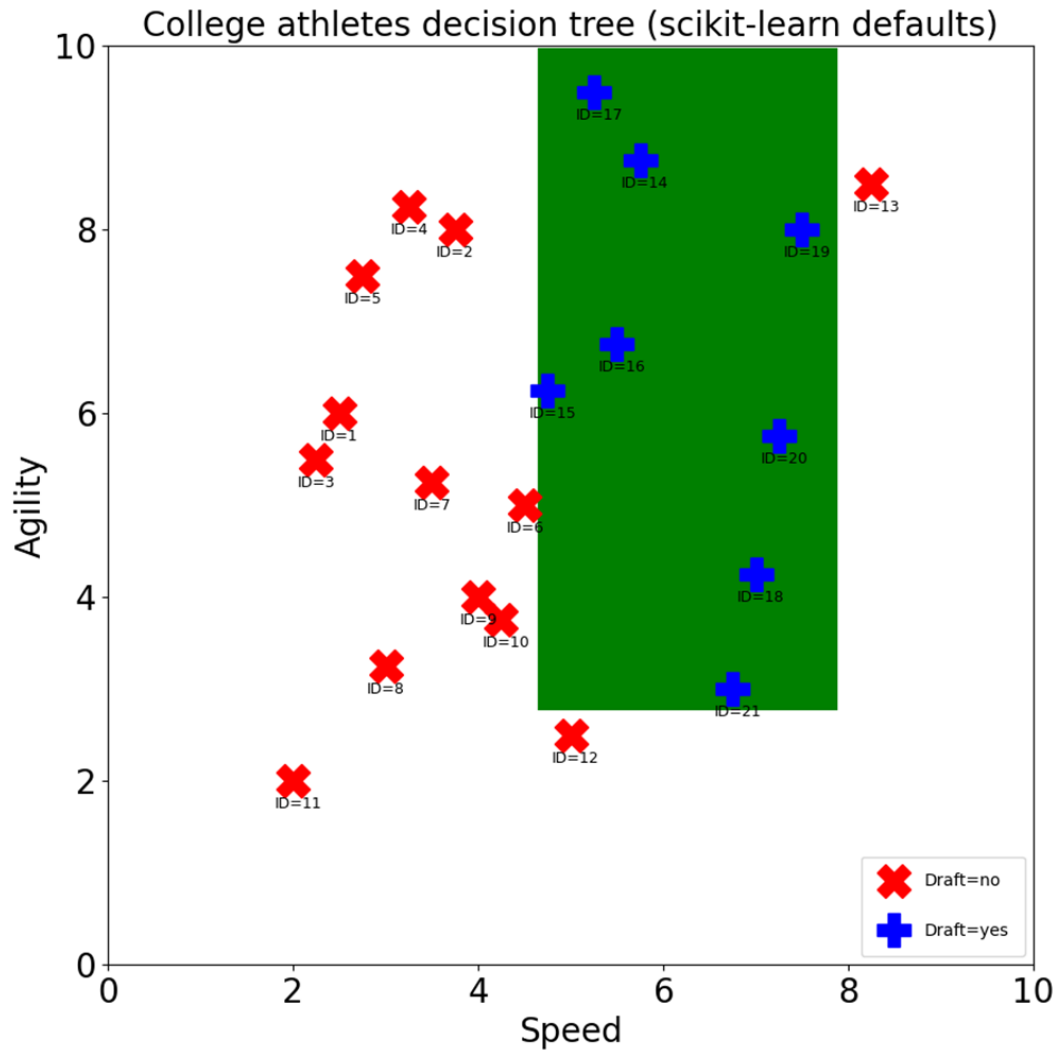
- Here is an example of a very simple hypothesis generated using an ML model – a linear classifier created using the scikit-learn `LogisticRegression` with the default settings
- Is this a good decision boundary? 19/21 training examples correct = 90.4% accuracy
- Note how the decision boundary is a straight line (in 2D). N.B. using logistic regression makes a strong underlying assumption, that the data is **linearly separable** (this may be reasonable for some datasets, but not for others!)

Decision tree on the college athletes dataset



- Here is an example of a more complex hypothesis - generated using the scikit-learn `DecisionTreeClassifier` with the default settings
- Note the two linear decision boundaries – very different form of hypothesis compared to logistic regression
- Is this a good decision boundary? 21/21 training examples correct = 100% accuracy

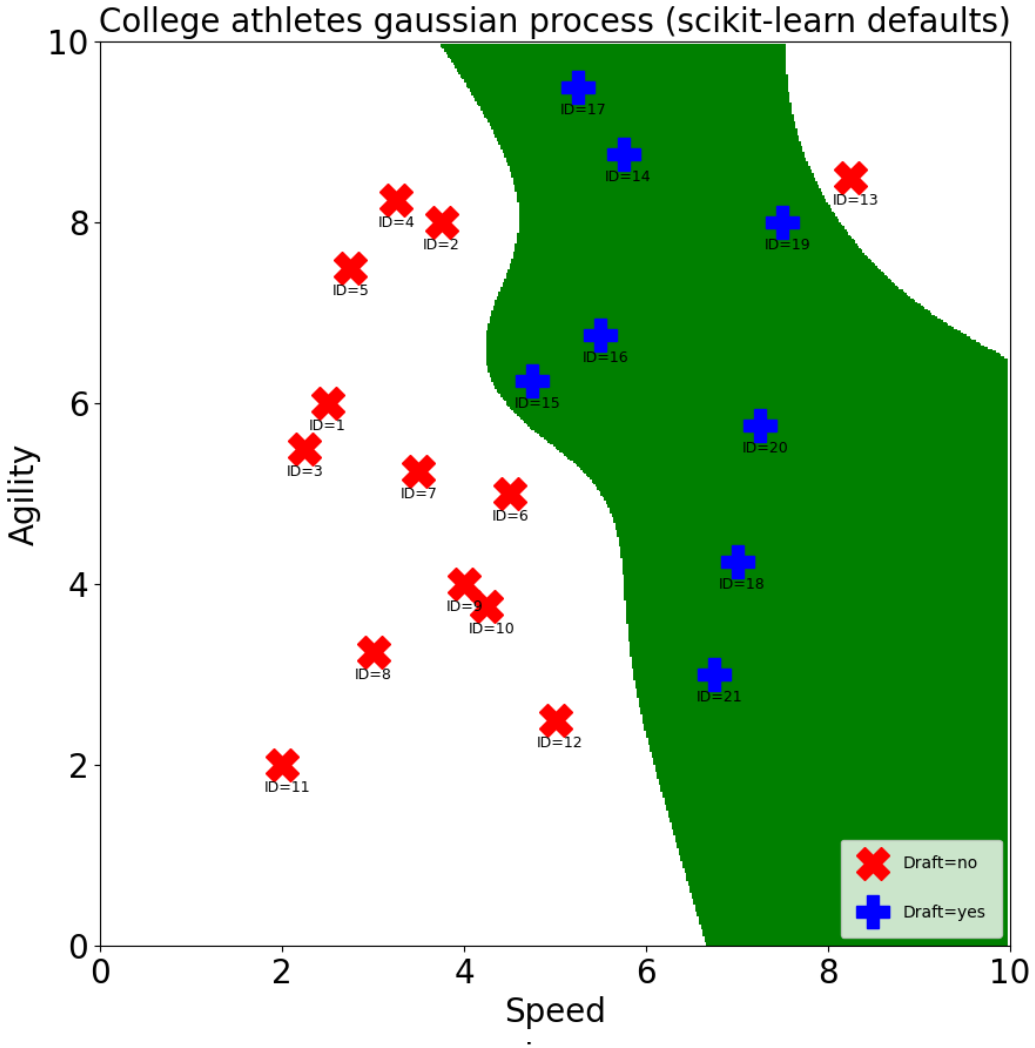
Decision tree on the college athletes dataset




```
|--- feature_0 <= 4.62
|   |--- class: 0
|--- feature_0 >  4.62
|   |--- feature_1 <= 2.75
|   |   |--- class: 0
|   |--- feature_1 >  2.75
|   |   |--- feature_0 <= 7.88
|   |   |   |--- class: 1
|   |   |--- feature_0 >  7.88
|   |   |   |--- class: 0
```

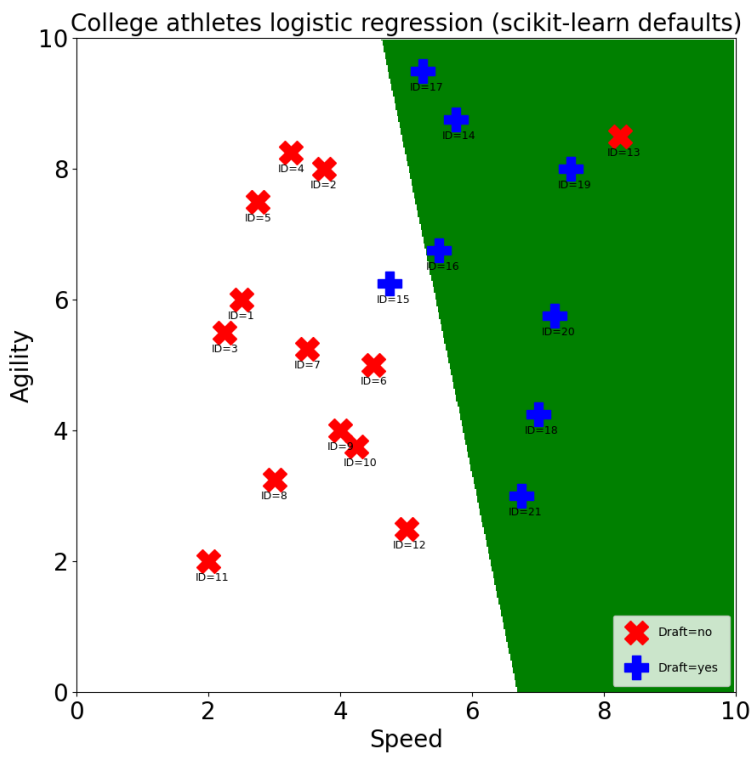
Hypothesis - learned decision rules

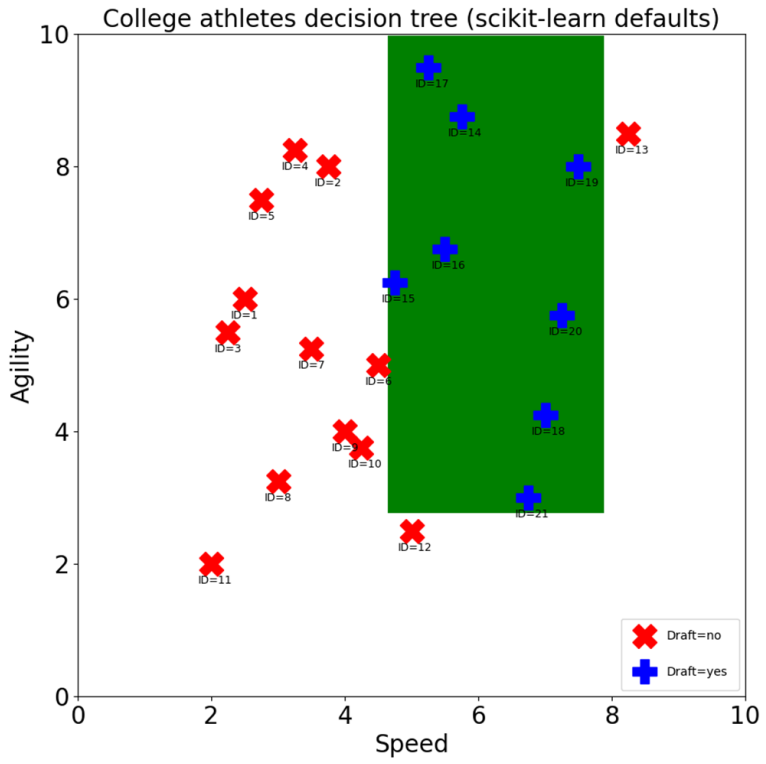
Gaussian process on the college athletes dataset

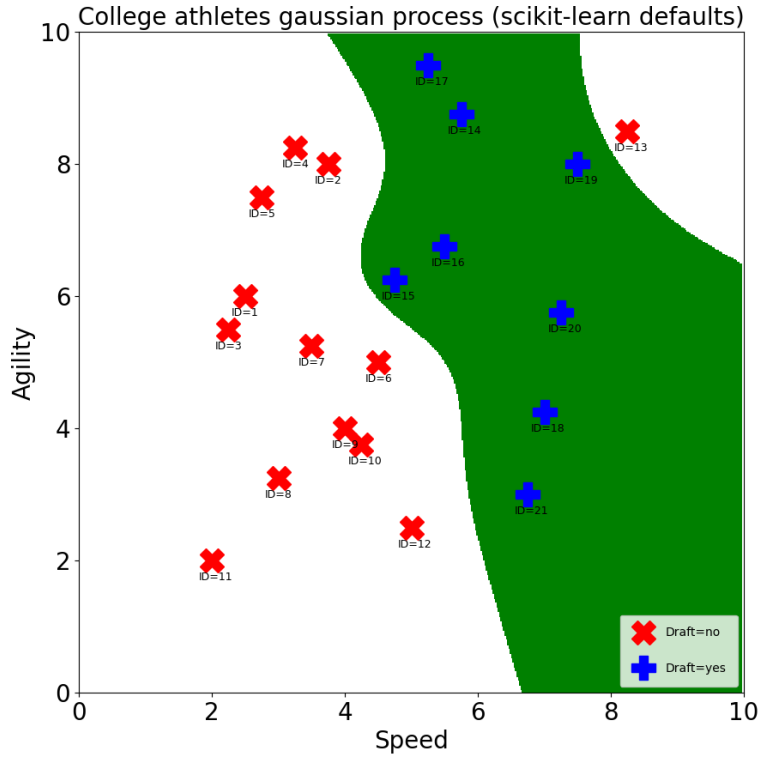


- Here is an example of a much more complex hypothesis - generated using the scikit-learn `GaussianProcessClassifier` with the default settings
- Note the smoothness of the decision boundary compared to the other methods
- Is this a good decision boundary? 21/21 training examples correct = 100% accuracy

Which model should we choose?

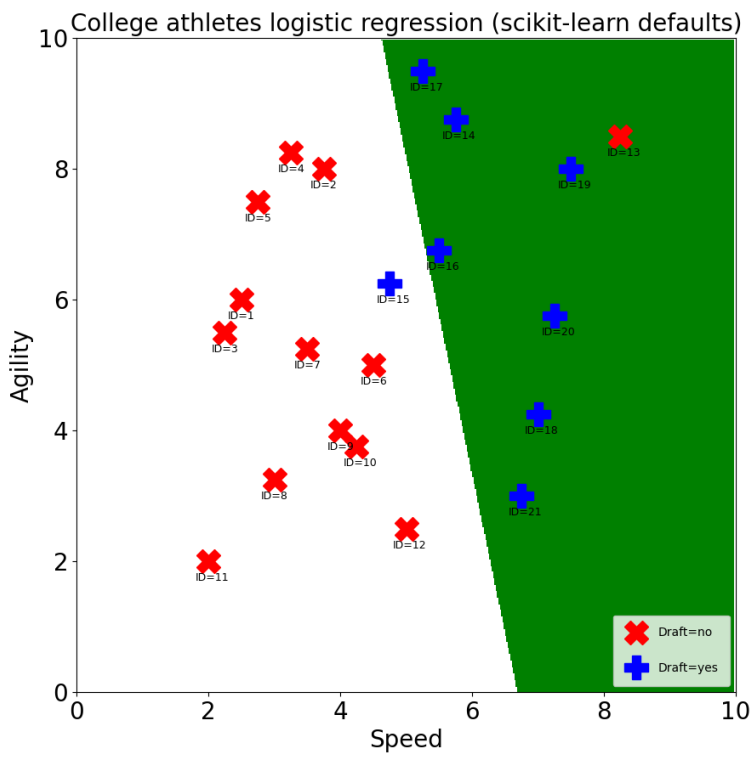


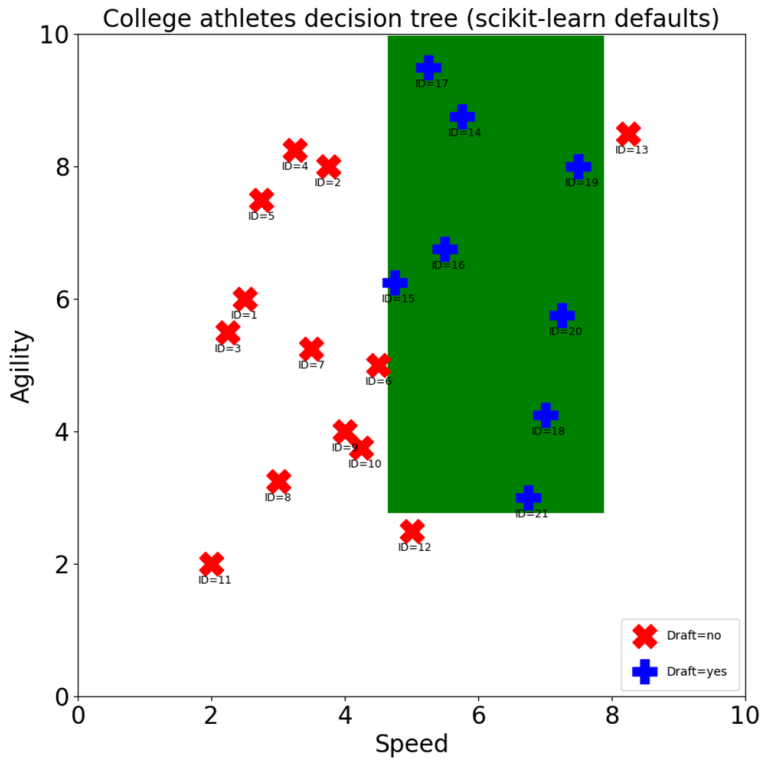


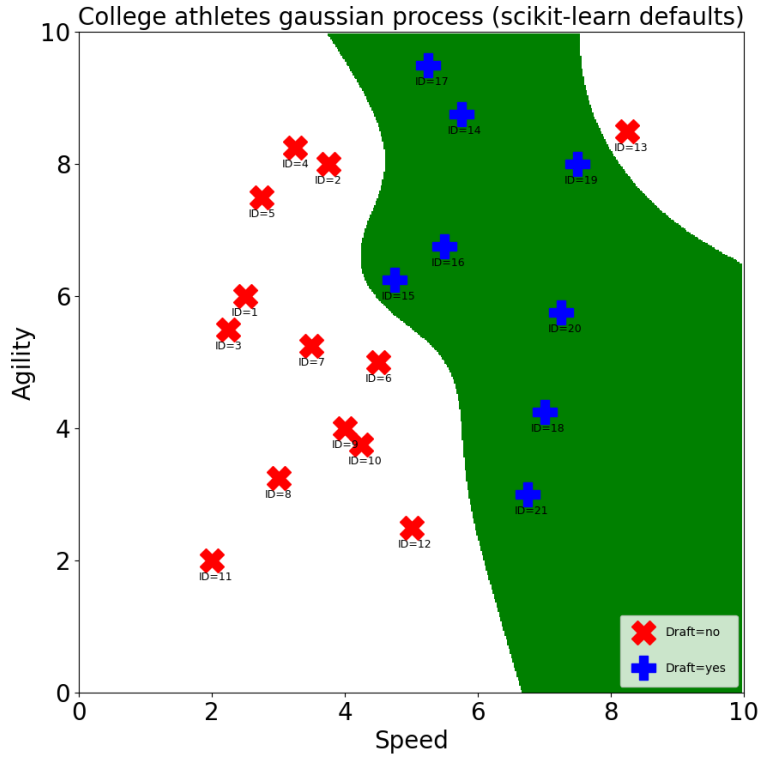


- Q: Which of these three models should we choose?

Which model should we choose?





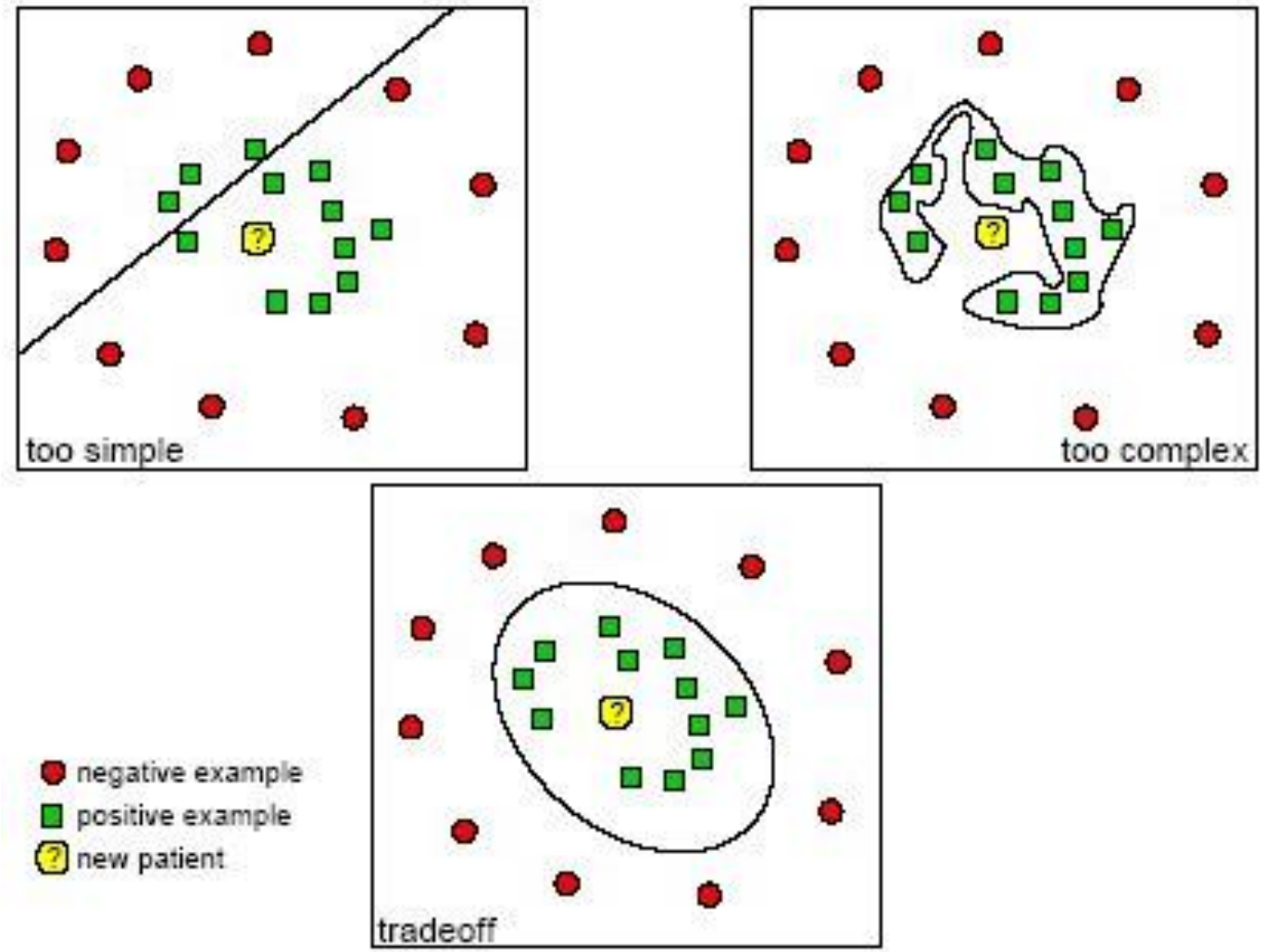


- Q: Which of these three models should we choose?
- A: It's complicated! We need to consider factors such as accuracy on the training data and independent test data, complexity of the hypothesis, per-class accuracy, …
- Will discuss in more detail later. For now, just note the differences between models!

Use of independent test data

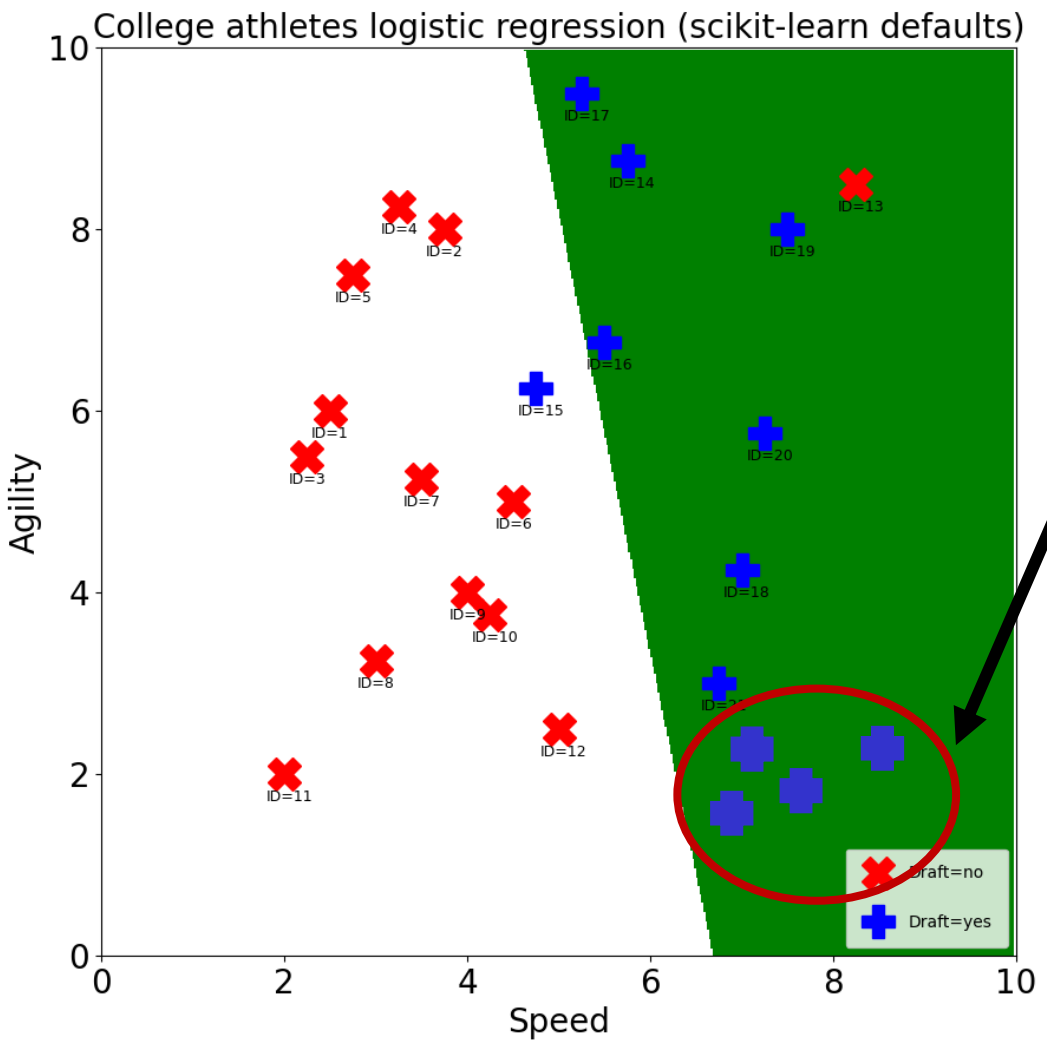
- Use of separate training and test datasets is very important when developing an ML model
- If you use all of your data for training, your model could potentially have good performance on the training data, but poor performance on new independent test data
- Good practice to ‘hold out’ some % of your data for testing only, as this can be used for tests to give an indication of performance in the real world on new, unseen data
- Using separate training and test datasets also help when identifying some common issues with ML models (next slides)

Illustration of underfitting & overfitting



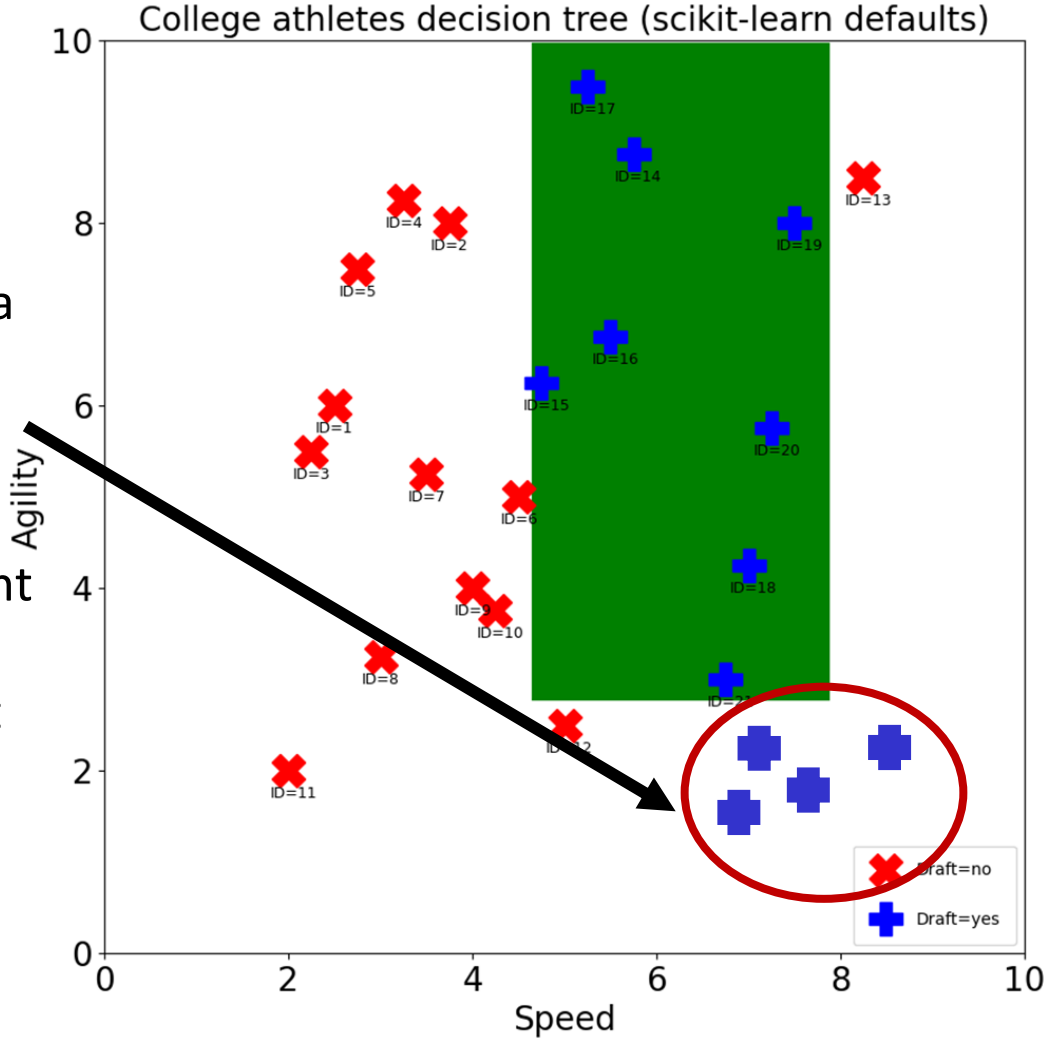
- **Underfitting** – hypothesis is too simple
- **Overfitting** – hypothesis is too complex

Separate test data helps to identify issues…



Example test data not used during training.

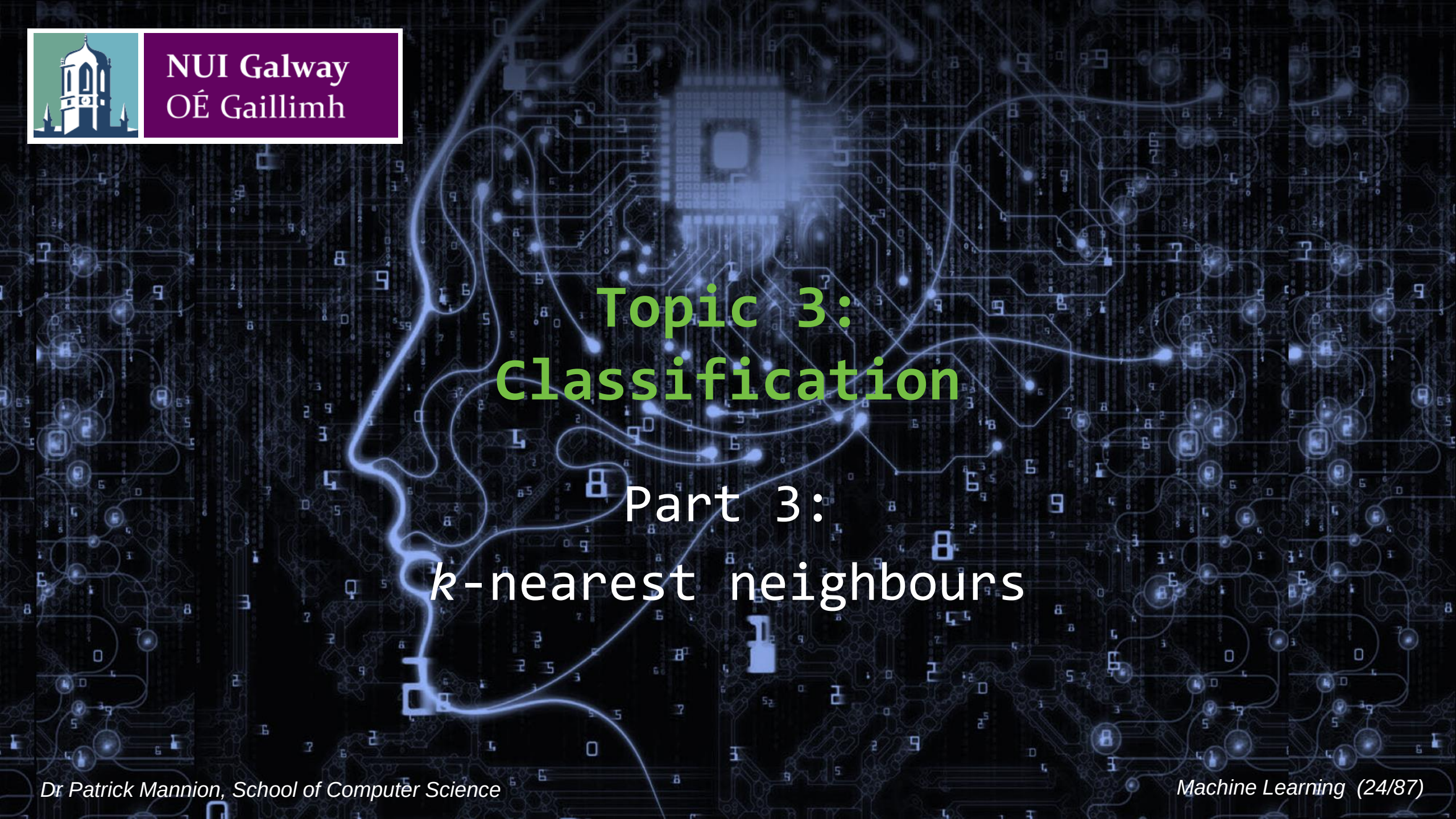
Model on the right performs well on training data, but less well on test data



Detecting underfitting & overfitting

- Previous slides have illustrated concepts only
  - In general, cannot visualise very high dimensional data: -=> can't directly observe overfitting/underfitting

- Main symptom of underfitting:
  - Poor performance even on the training data
- Main symptom of overfitting:
  - *Much* better performance on the training data than on independent test data
  - (Slightly better performance is to be expected)



Topic 3: Classification

Part 3: *k*-nearest neighbours

K-nearest neighbours algorithm

- One of the simplest machine learning algorithms is ***k*-nearest neighbours** (or *k*-NN for short)
- *k*-NN generates a hypothesis / model using a very simple principle:
  - predictions for the label or value assigned to a **query instance** should be made based on the most *similar* instances in the training dataset.
  - Hence this is also known as *similarity-based learning*
- *k*-NN can be used for both classification and regression tasks, in this lecture we will focus on its application to classification tasks only
- Implementation in scikit-learn: `KNeighborsClassifier`
  - https://scikit-learn.org/stable/modules/neighbors.html
  - https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

k Nearest Neighbours Algorithm

- Operation is relatively easy to appreciate
- Key insight:
  - Each example is a point in the feature space
  - If samples are close to each other in feature space, they should be close in their target values
- Related to *Case-based Reasoning*
- How it works:
  - When you want to classify a new **query case**:

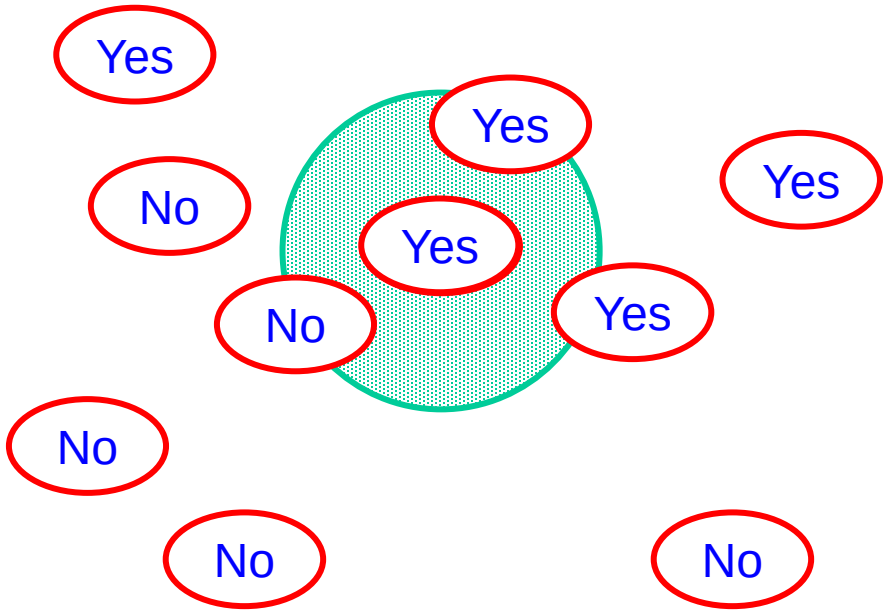
    Compare it to the stored set and retrieve the *k* most similar one(s)

    Give the query case a label based on the similar one(s)

k Nearest Neighbours Algorithm

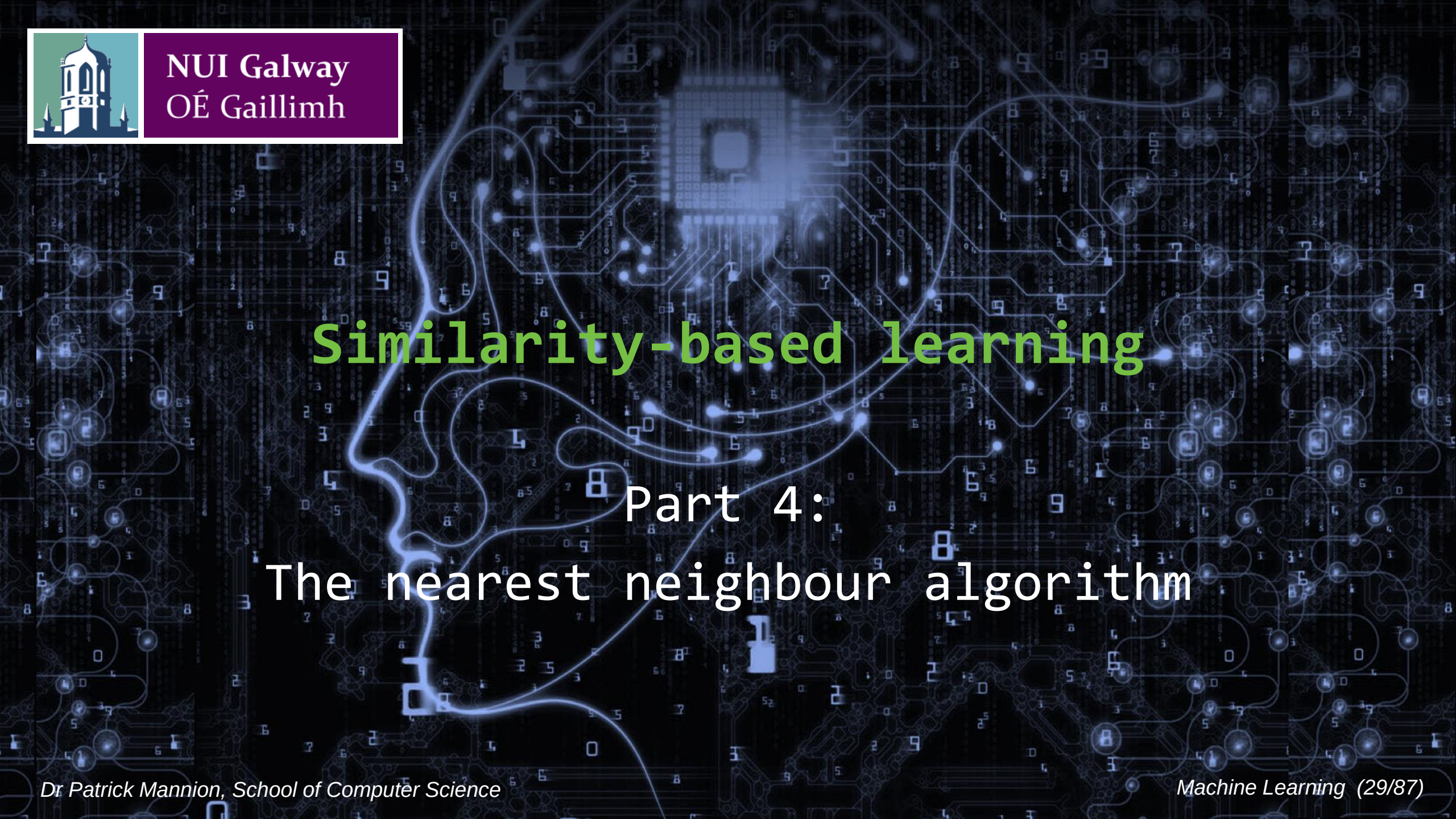
- *k* Nearest Neighbours algorithm:
  - Base prediction on several (*k*) nearest neighbours
  - Compute similarity of query case to all stored cases, and pick the nearest *k* neighbours
  - Simplest way to do this: sort them by distance, pick lowest *k*
  - More efficient: can identify k nearest in a single pass through the list of distances
- Classification with kNN:
  - Neighbours vote on classification of test case
  - Prediction is the majority class

k Nearest Neighbours: Visualisation of Classification Example



This visualisation assumes Euclidean distance

When *k*=3, the 3 nearest neighbours vote:
Decision is Yes



NUI Galway
OÉ Gaillimh

Similarity-based learning

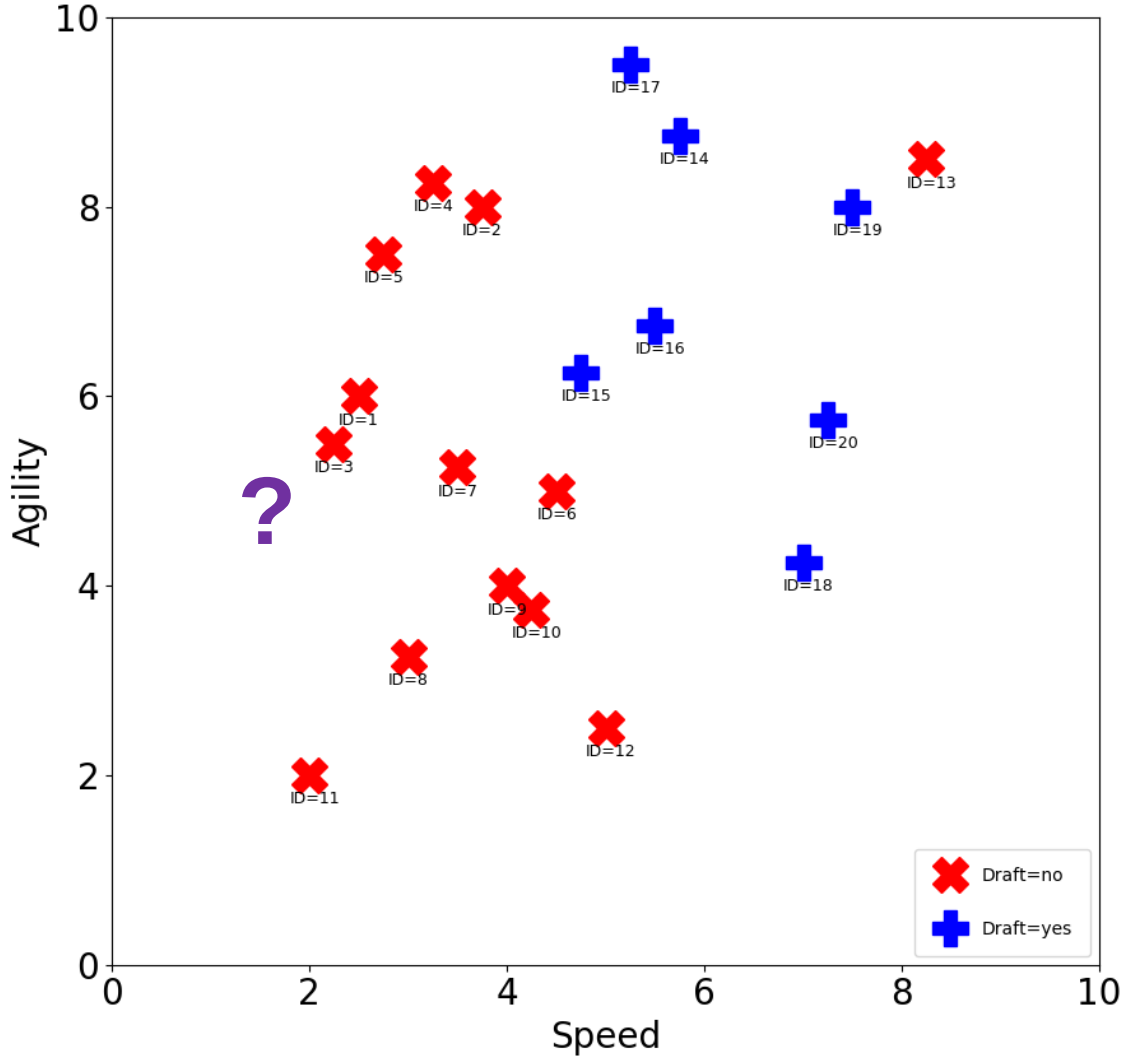
Part 4:
The nearest neighbour algorithm

The Nearest Neighbour algorithm

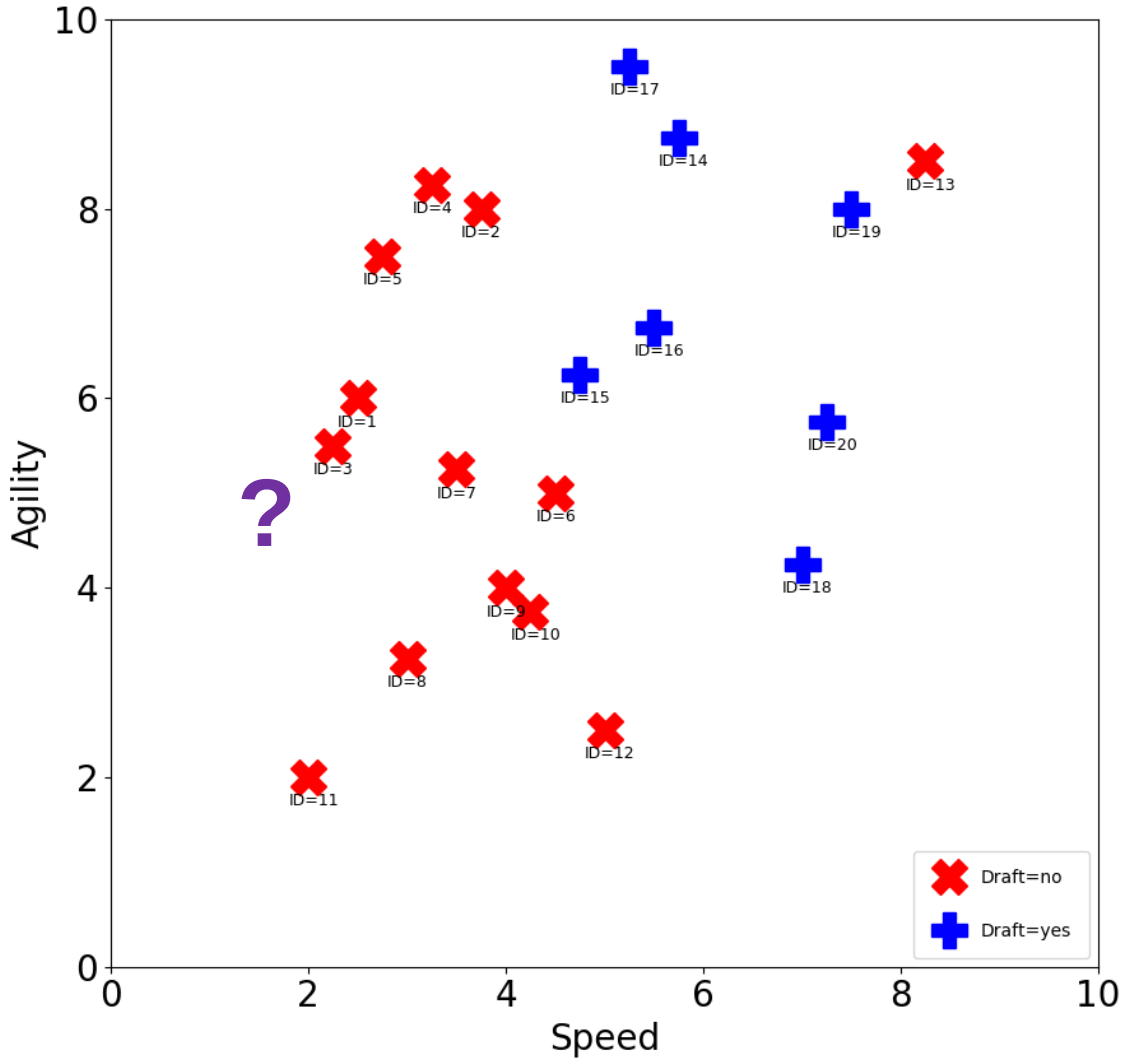
- 1-Nearest Neighbour algorithm:
  - Simplest similarity-based/instance-based method
  - No real training phase: just store the training cases
  - Given a query case with value to be predicted, compute its distance from all stored instances
  - Choose the nearest one; assign the test case to have the same label (class or regression value) as this one
  - Requires a **distance metric**
  - Main problem: susceptibility to noise
- To reduce susceptibility to noise, use more than 1 neighbour (the *k*-NN algorithm)

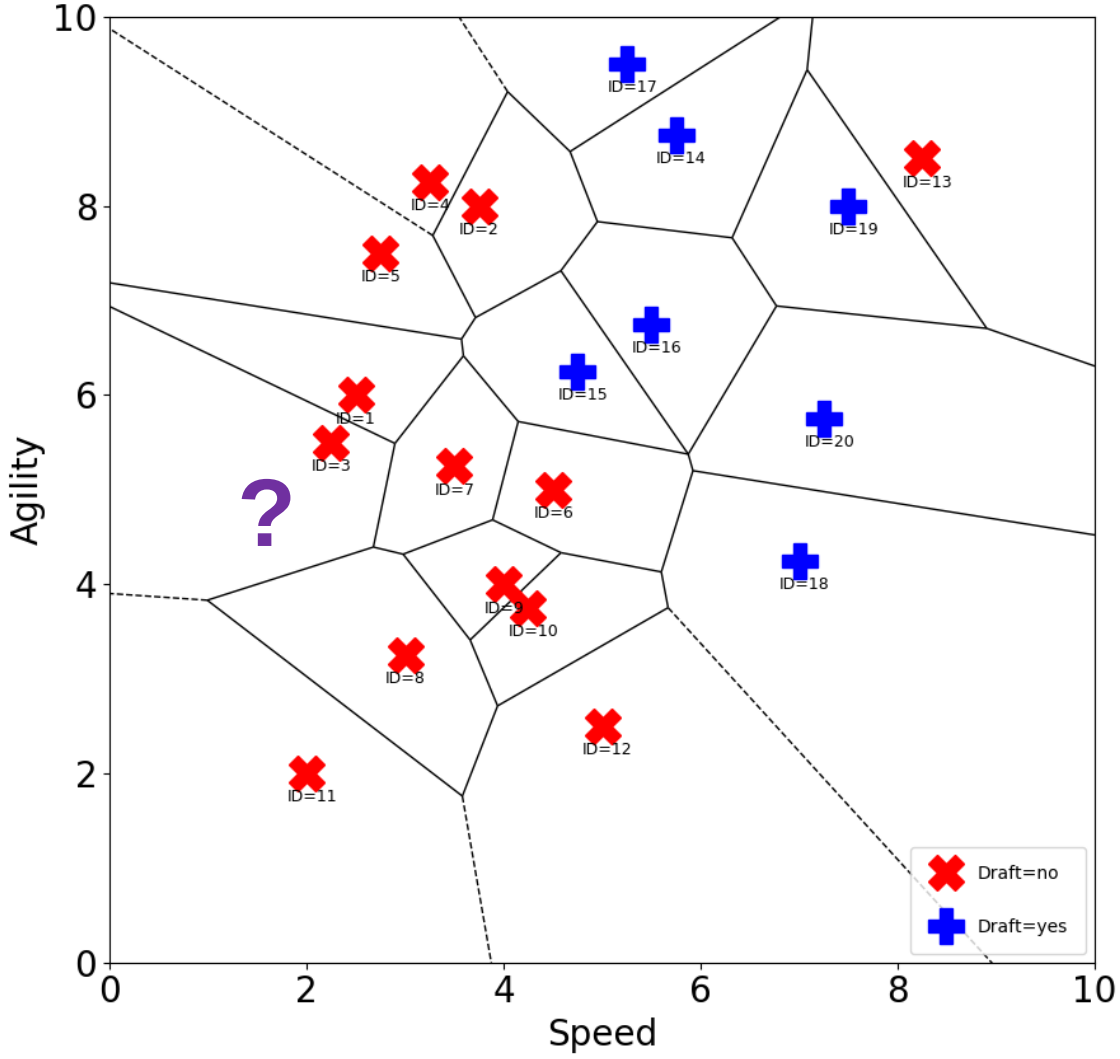
Visualising decision boundaries

- By visually inspecting the feature space plot, we can see that 1 NN will predict the target class as “no”
- 1 NN with Euclidean distance is equivalent to partitioning the feature space into a **Voronoi tessellation**
- Finding the predicted target class is equivalent to finding which **Voronoi region** it occupies
- Note: all visualisations from here on use Euclidean distance

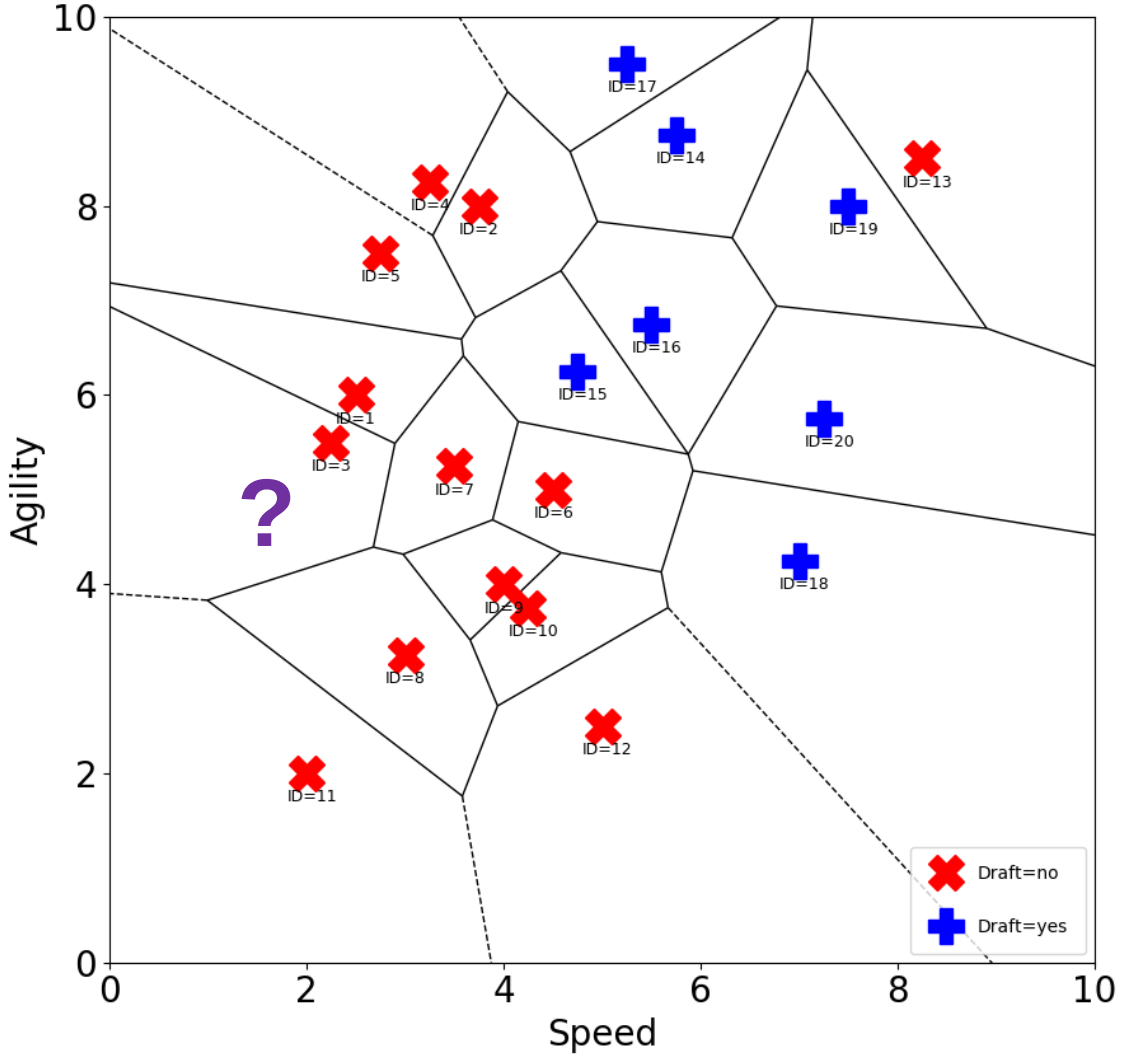


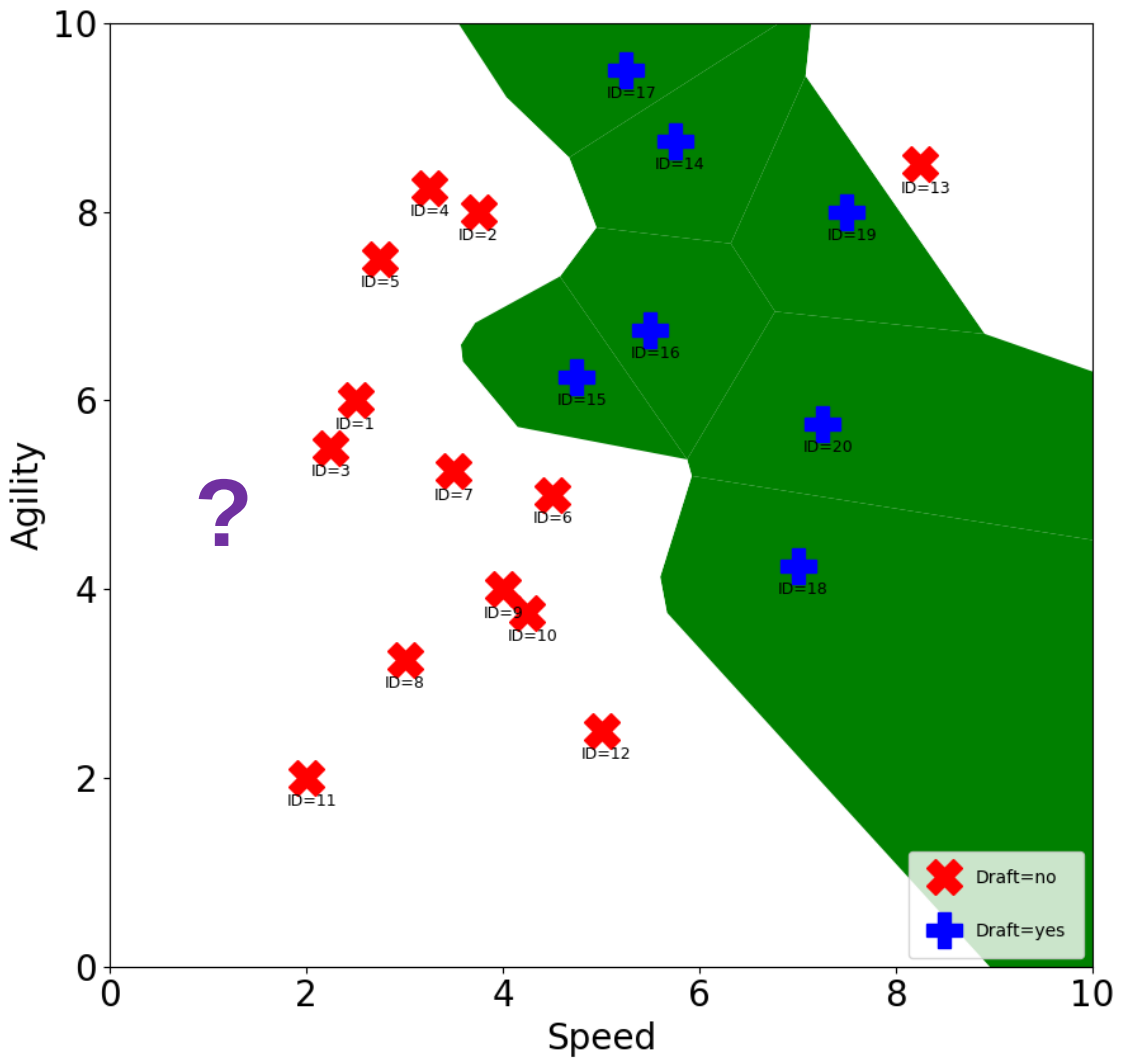
Voronoi tessellation





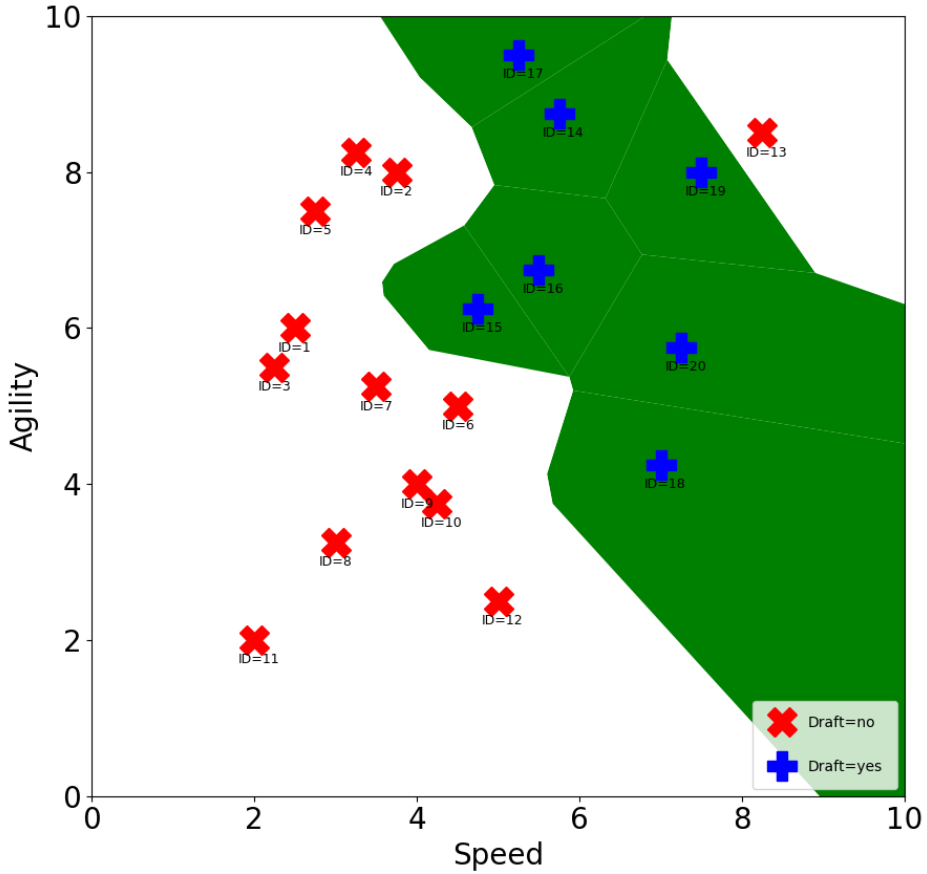
1-NN Decision boundary from Voronoi tessellation

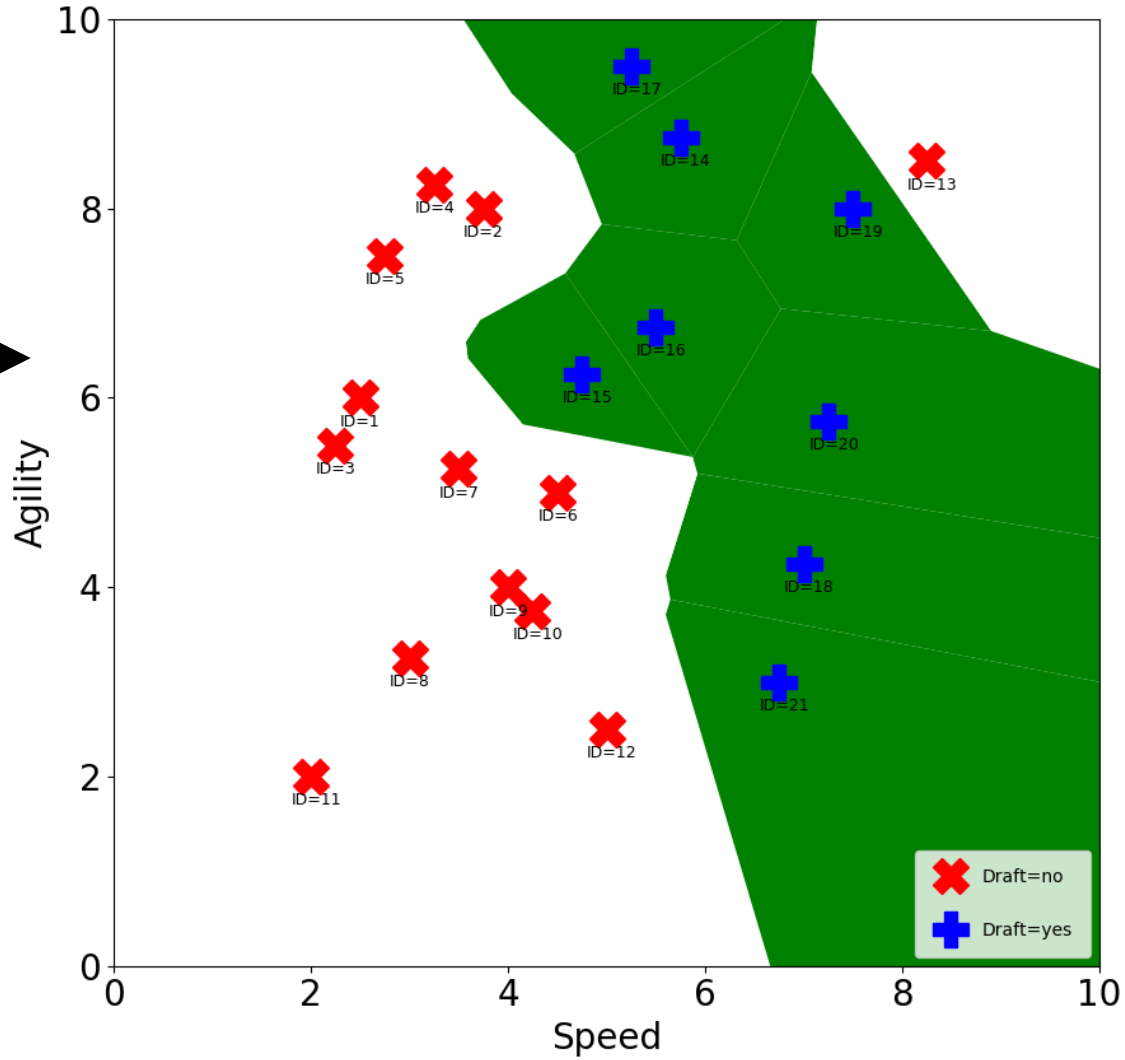


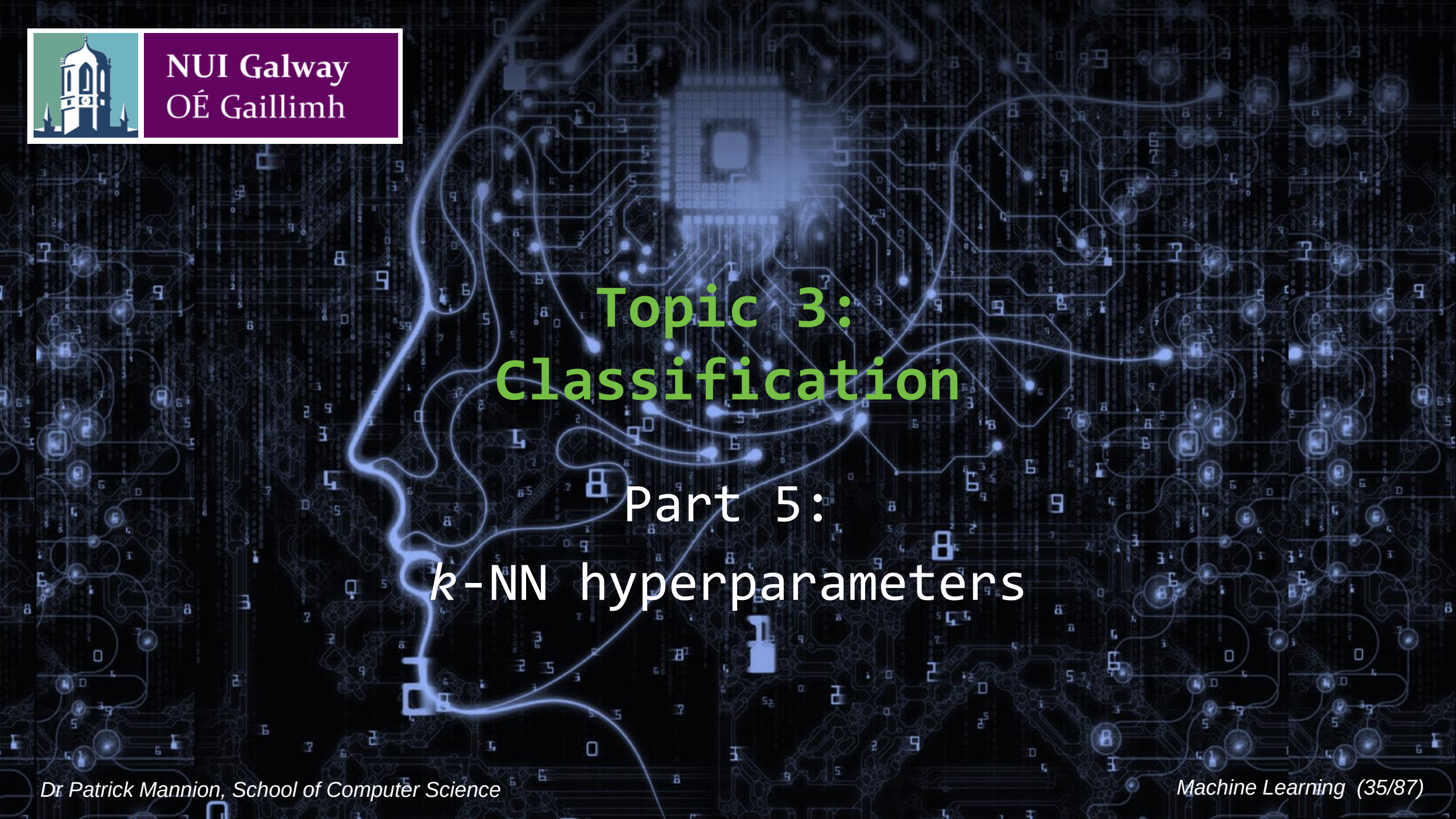


Effect of adding more training data

ID	Speed	Agility	Draft?
21	6.75	3.00	yes







Topic 3: Classification

Part 5: *k*-NN hyperparameters

Hyperparameters (1/2)

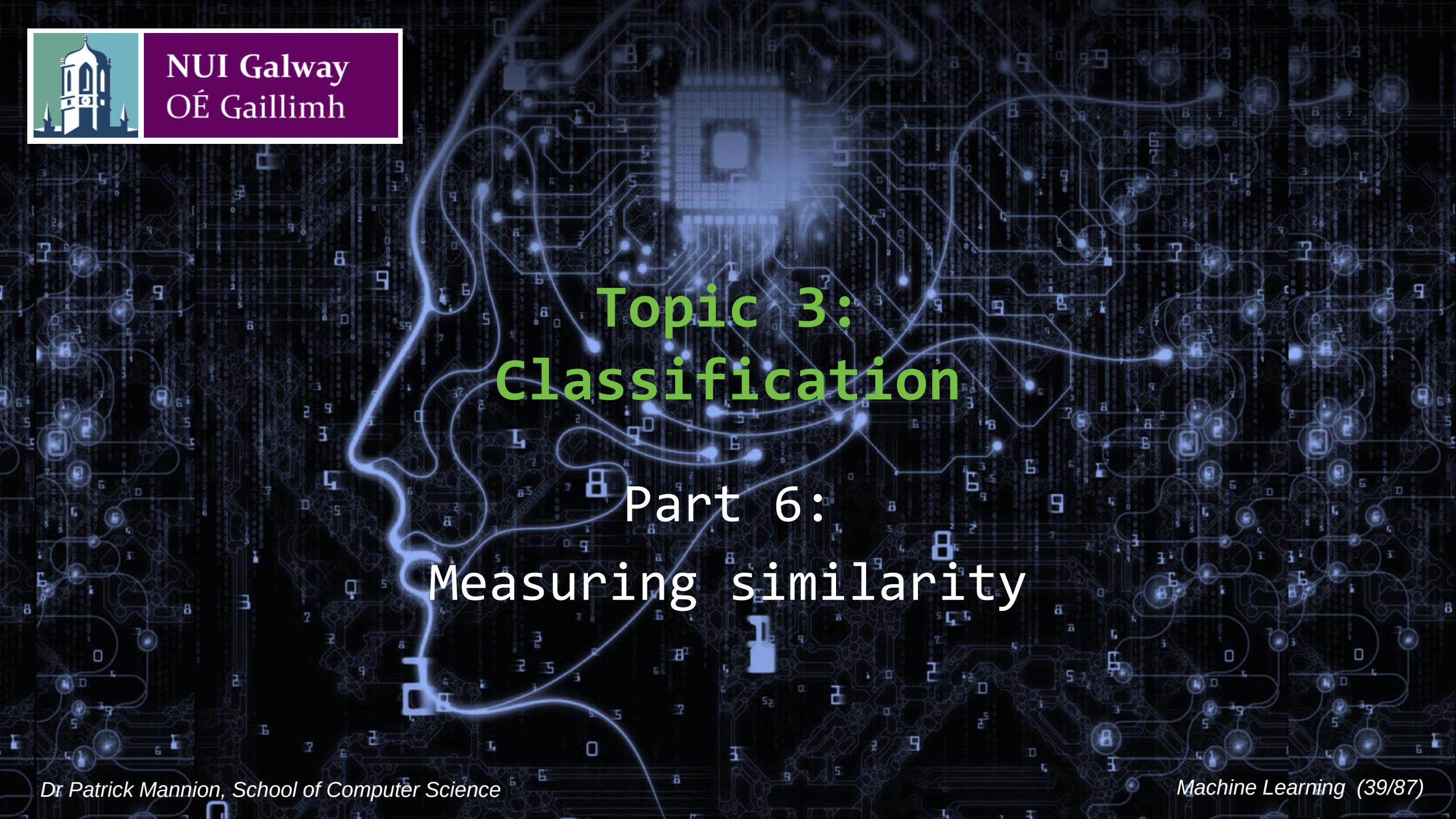
- The *k*-NN algorithm also introduces a new concept to us that is very important for ML algorithms in general: **hyperparameters**
- In ML algorithms, a hyperparameter is a parameter set by the user that is used to control the behaviour of the learning process
- Many ML algorithms also have other parameters that are set by the algorithm during its learning process (e.g., the weights assigned to connections between neurons in an artificial neural network)
- Examples of hyperparameters include:
  - Learning rate (typically denoted using the Greek lowercase letter α 'alpha')
  - Topology of a neural network (the number and layout of neurons)
  - The choice of optimiser when updating the weights of a neural network

Hyperparameters (2/2)

- Many ML algorithms are very sensitive to the choice of hyperparameters -> poor choices of values give poor performance!
- Therefore, hyperparameter tuning (determining the values that give best performance) is an important topic in ML
- Some simple ML algorithms do not have any hyperparameters (e.g., ordinary least squares linear regression that we used in the Jupyter Lab in topic 2)
- We will discuss hyperparameters in more detail in later lectures as we encounter other more complex algorithms

k-NN hyperparameters

- *k*-NN has several key hyperparameters that we must choose before applying it to a dataset:
  - the number of neighbours *k* to take into account when making a prediction (called `n_neighbours` in the scikit-learn `KNeighborsClassifier` implementation)
  - the method used to measure how similar instances are to one another (called `metric` in scikit-learn)
  - the weighting scheme to be used (called `weights` in scikit-learn)
- We will discuss each hyperparameter in detail in later slides
- See example file **k-NN_hyperparameters.ipynb** on Blackboard



Topic 3: Classification

Part 6: Measuring similarity

Measuring similarity using distance

- Consider the college athletes dataset from earlier
- How should we measure the similarity between instances in this case? E.g. how similar are datapoints 5 and 12?
- Distance is one option: plot the points in 2D space and draw a straight line between them
- This approach can scale to arbitrarily high dimensions as we will see. We can think of each feature of interest as a dimension in hyperspace

College Athletes			
ID	Speed	Agility	Draft
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

Measuring similarity using distance

- A **metric** or distance function may be used to define the distance between any pair of elements in a set.
- $metric(\boldsymbol{a}, \boldsymbol{b})$ is a function that returns the distance between two instances $\boldsymbol{a}$ and $\boldsymbol{b}$ in a set
- $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors containing the values of the attributes we are interested in for the data points we wish to measure between

Euclidean distance

- Euclidean distance is one of the best-known distance metrics
- Computes the length of a straight line between two points

$$Euclidean(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{m} (\boldsymbol{a}[i] - \boldsymbol{b}[i])^2}$$

- Here m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- Square root of the sum of squared differences for each feature

Manhattan distance

- Manhattan distance (also known as “taxicab distance”)
- Computes the length of a straight line between two points

$$Manhattan(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{m} abs(\boldsymbol{a}[i] - \boldsymbol{b}[i])$$

- As before m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- $abs()$ returns the absolute value
- Sum of the absolute differences for each feature

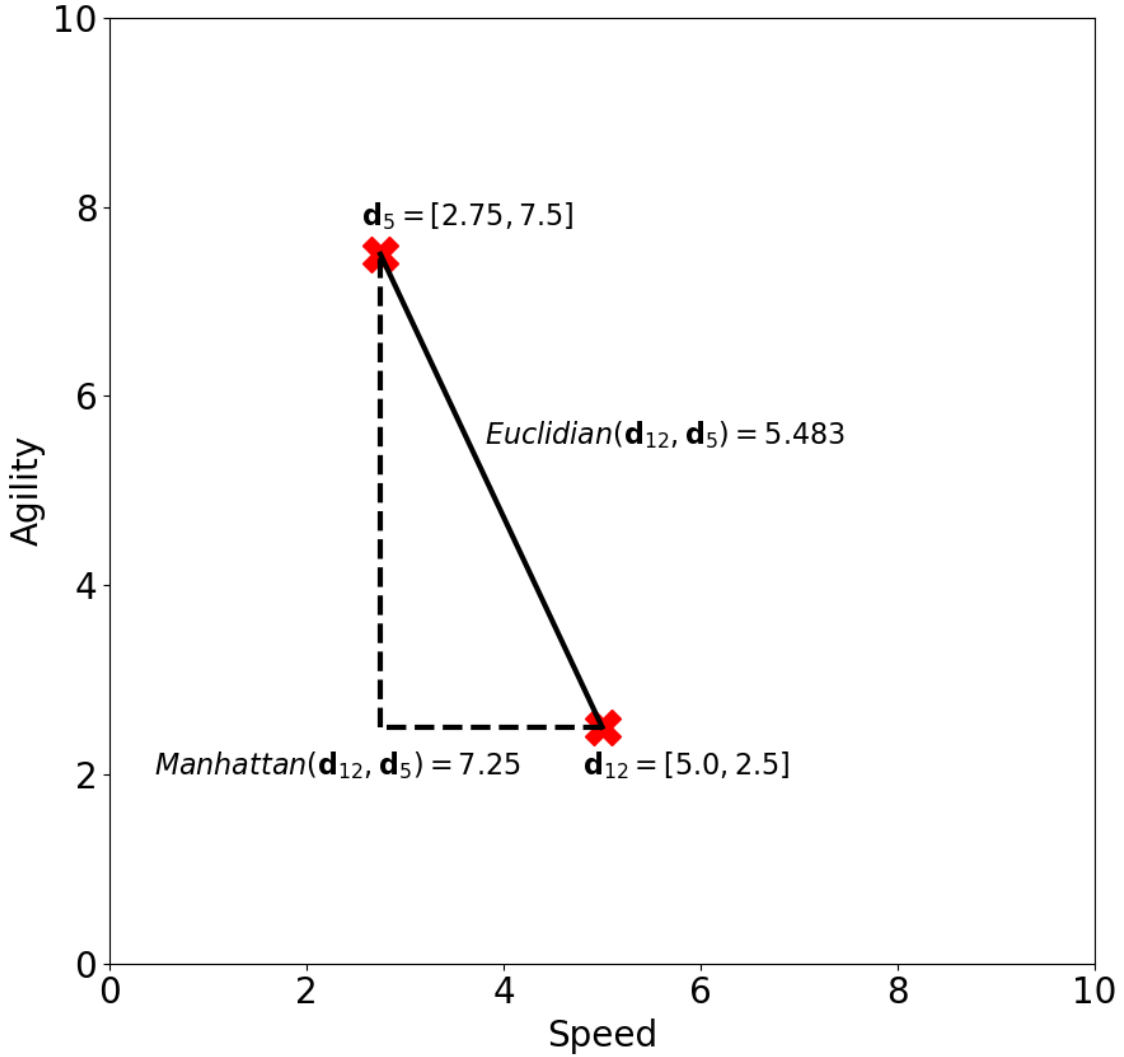
Example: calculating distance

Calculate distance between

$\boldsymbol{d}_{12} = [5.00, 2.50]$ and $\boldsymbol{d}_{5} = [2.75, 7.50]$

$$Euclidean(\boldsymbol{d}_{12}, \boldsymbol{d}_{5})$$
$$= \sqrt{(5.00 - 2.75)^2 + (2.50 - 7.50)^2}$$
$$= 5.483$$

$$Manhattan(\boldsymbol{d}_{12}, \boldsymbol{d}_{5})$$
$$= abs(5.00 - 2.75) + abs(2.50 - 7.50)$$
$$= 7.25$$



Minkowski distance

- The Minkowski distance metric generalises both the Manhattan distance and the Euclidean distance metrics

$$Minkowski(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{m} abs(\boldsymbol{a}[i] - \boldsymbol{b}[i])^p \right)^{\frac{1}{p}}$$

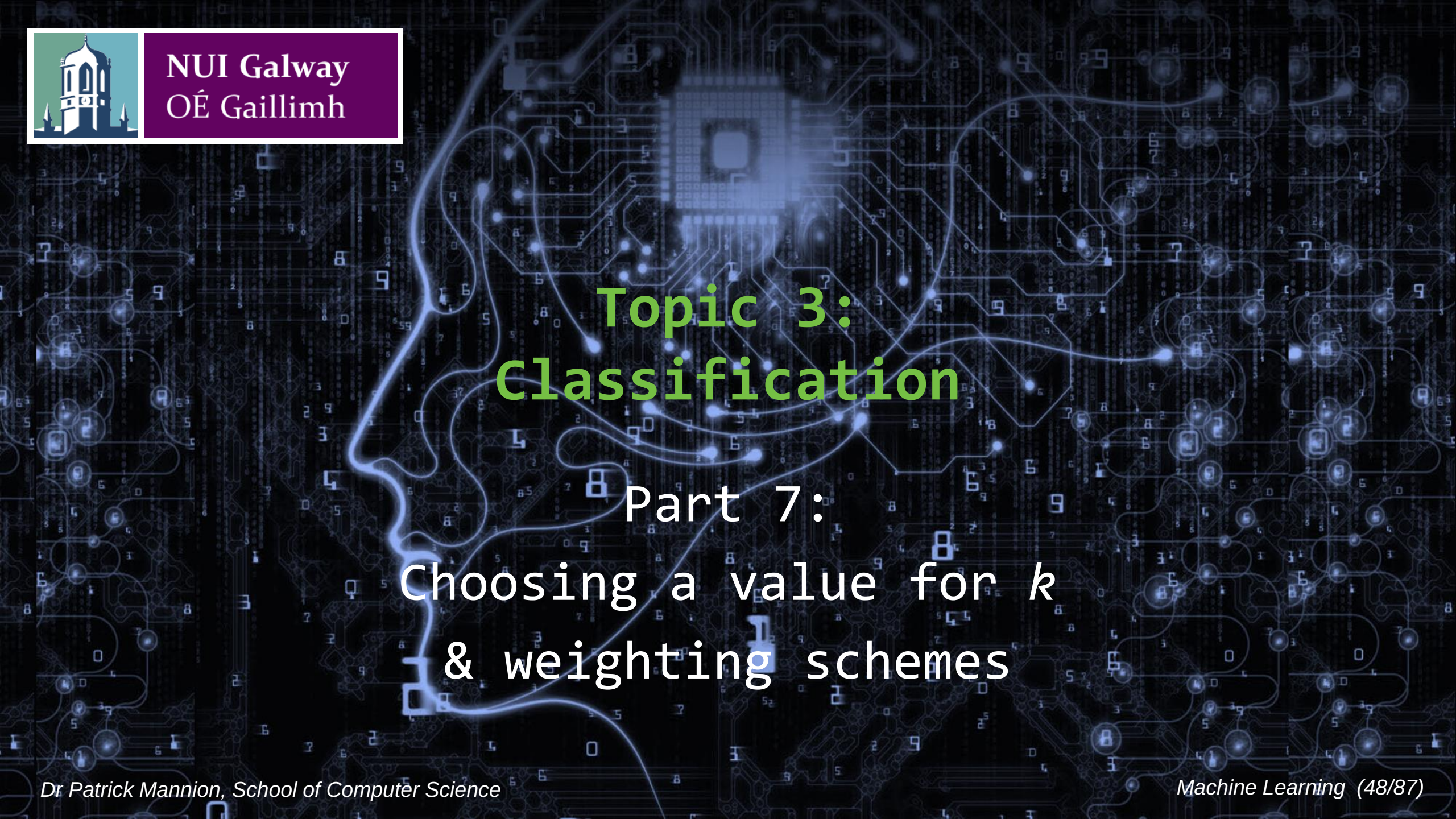
- As before m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- $abs()$ returns the absolute value
- Sum of the absolute differences for each feature

Similarity for discrete attributes

- So far, we have considered similarity measures that only apply to continuous attributes
- Do not confuse discrete/continuous attributes with classification/regression!
- Many datasets have attributes that have a finite number of discrete values (e.g. Yes/No or True/False, survey responses, ratings)
- Once approach to handling discrete attributes is the **Hamming distance**
- Hamming distance is calculated 0 for each attribute where both cases have the same value, 1 for each where they are different
- E.g. Hamming distance between “Stephen" and “Stefann" is 3.

Comparison of distance metrics

- Euclidian and Manhattan distance are most commonly used, although it is possible to define infinitely many distance metrics using the Minkowski distance
- Manhattan is cheaper to compute than Euclidean as it is not necessary to compute the squares of differences and a square root, so may be a good choice for very large datasets if computational resources are limited
- It's worthwhile to try out several different distance metrics to see which is most suitable for the dataset at hand
- Many other methods to measure similarity exist, e.g. cosine similarity, Russel-Rao, Sokal-Michener



NUI Galway
OÉ Gaillimh

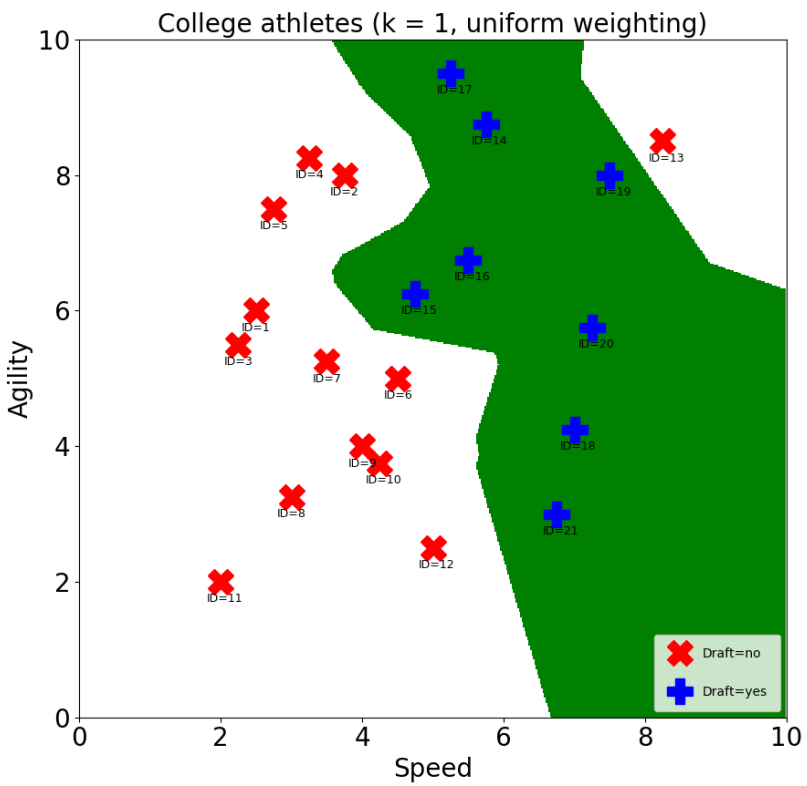
Topic 3: Classification

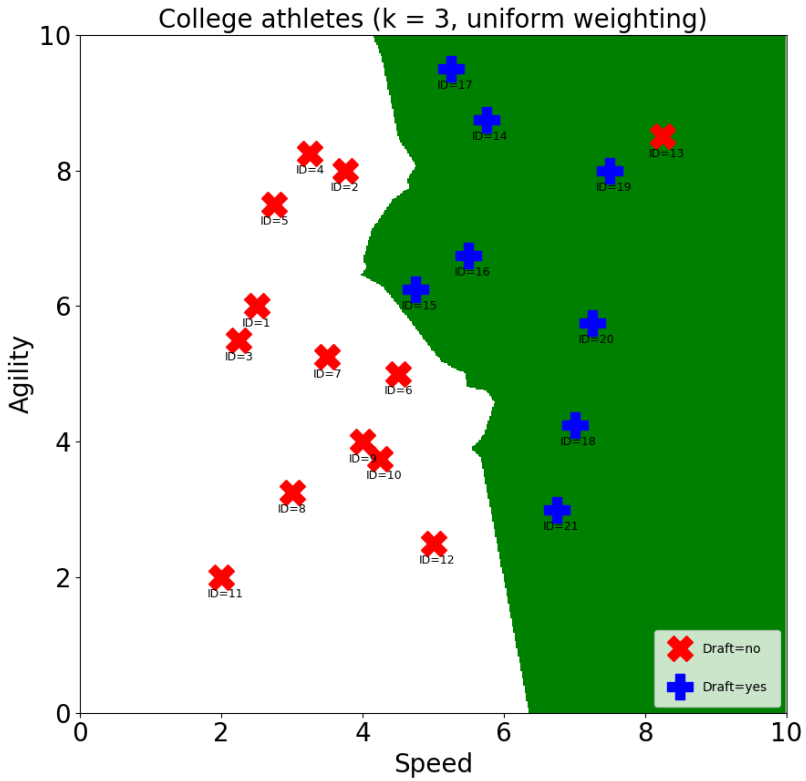
Part 7: Choosing a value for *k* & weighting schemes

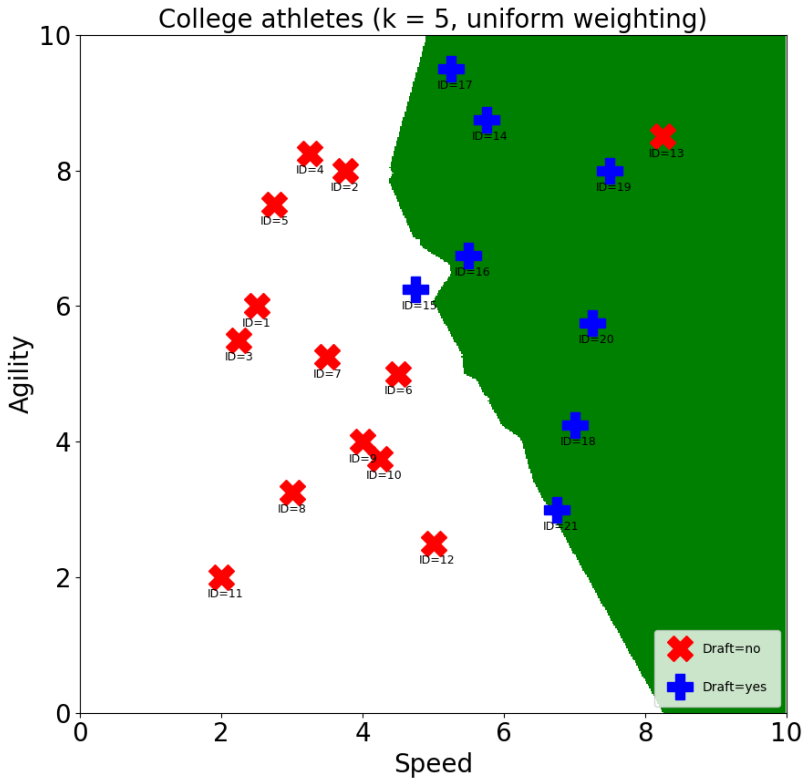
Choosing a value for *k*

- What value for *k*?
  - At least 3 (if not using 1-NN); often in range 5-21
  - Application dependent: need to experiment to find optimum
  - Can use all cases with **distance-weighted kNN (later)**
- Increasing *k* has a smoothing effect
  - Too-low: tends to overfit if data is noisy
  - Too-high: tends to underfit
  - In imbalanced datasets, the majority target class tends to dominate for larger *k*
- Note: *k* does not affect computational cost much
  - Most cost of computation is in calculating distances from the query to all stored instances (linear in #cases and #attributes)

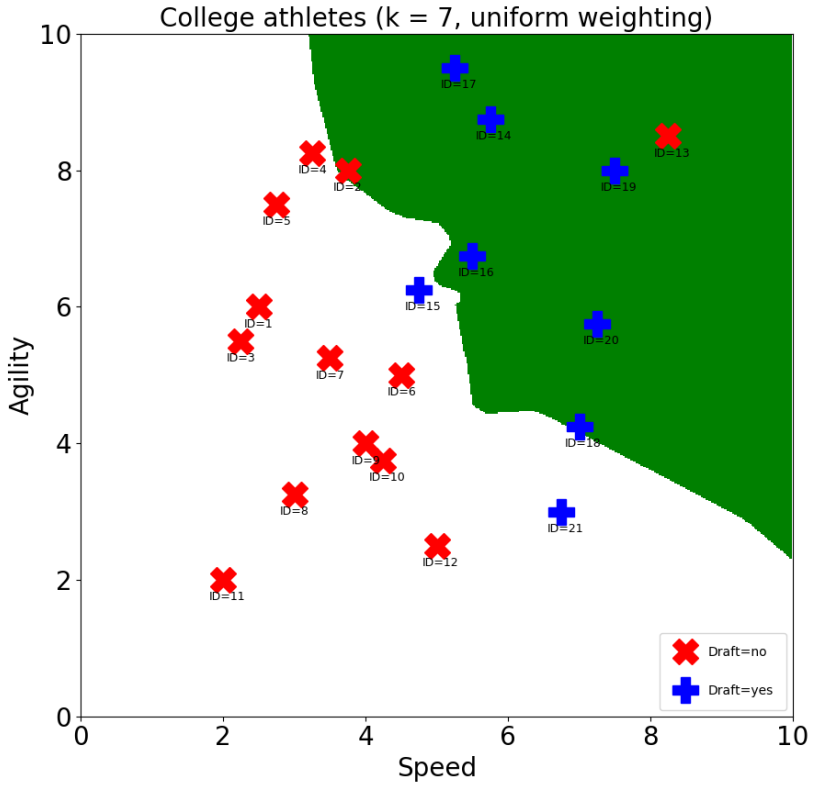
Effect of increasing *k*

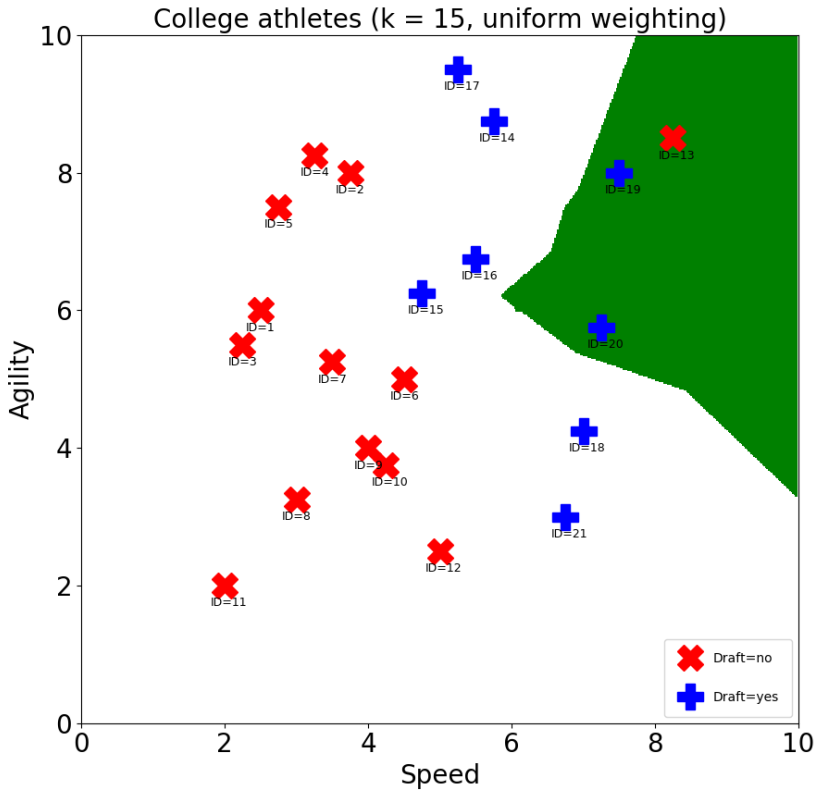


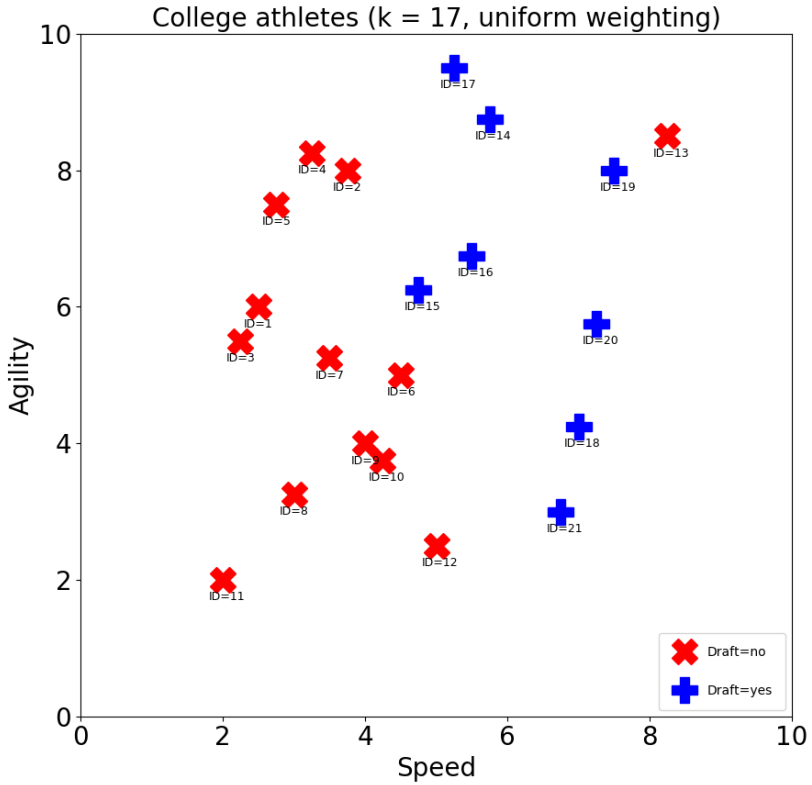




Effect of increasing *k*

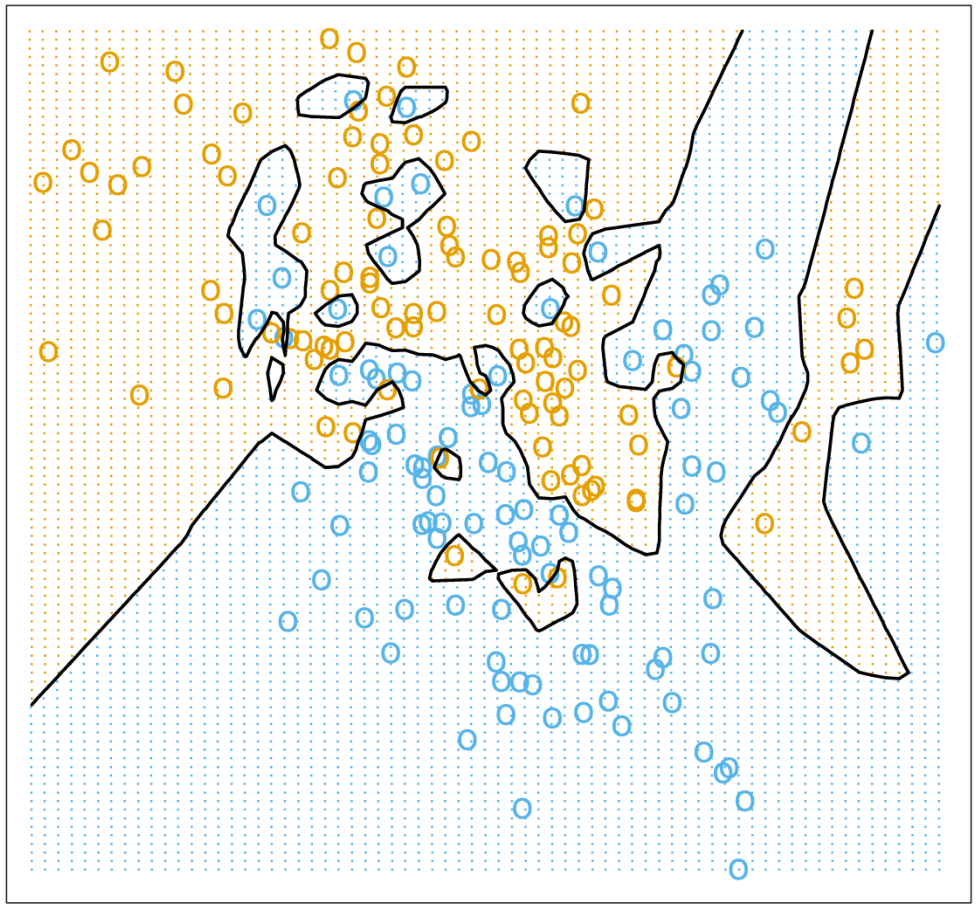




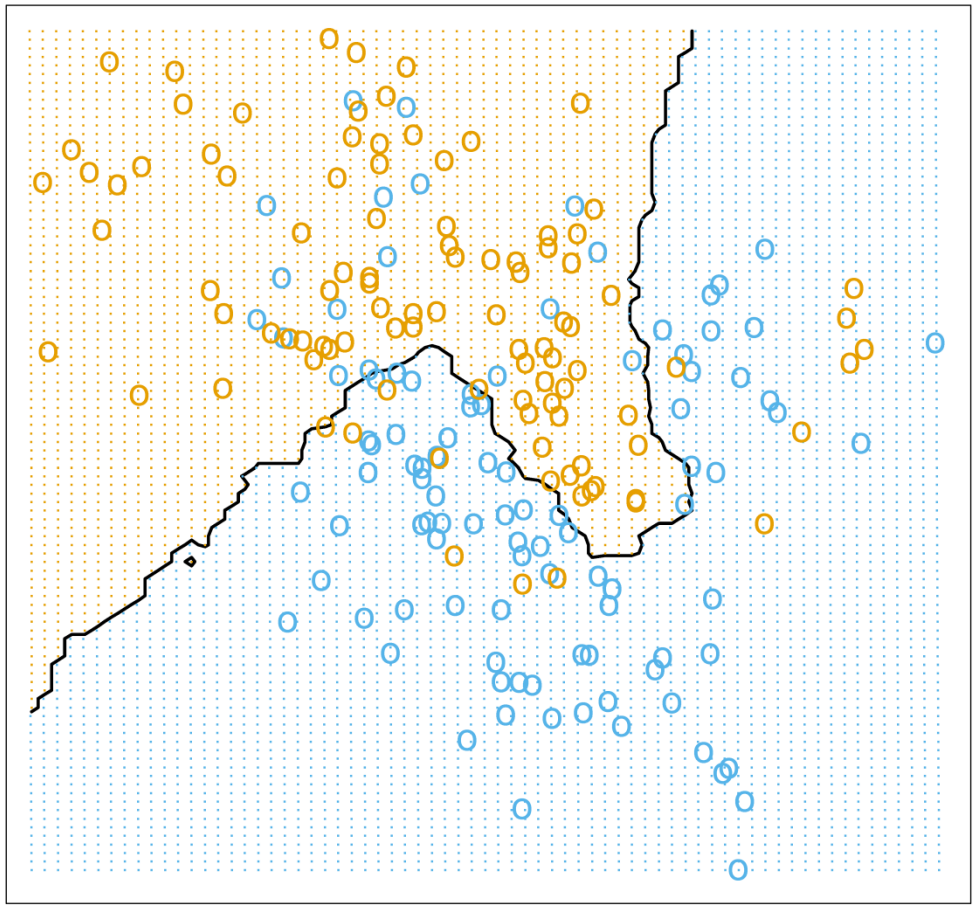


Smoothing Effect of k

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier



Hastie, Tibshirani & Friedman, Figs 2.3 and 2.2. Recall discussion of overfitting in Topic 2.

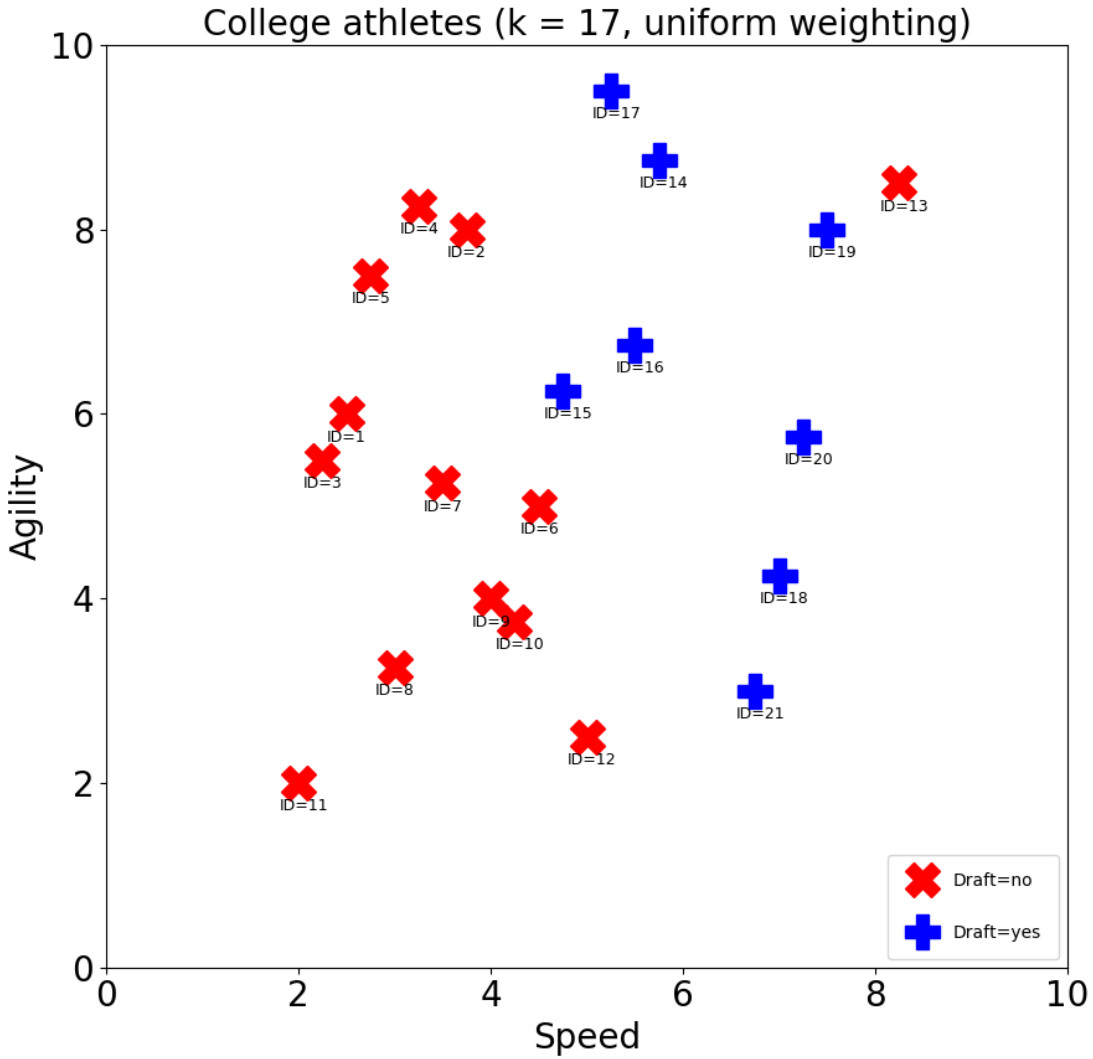
Aside: Underfitting on Same Data

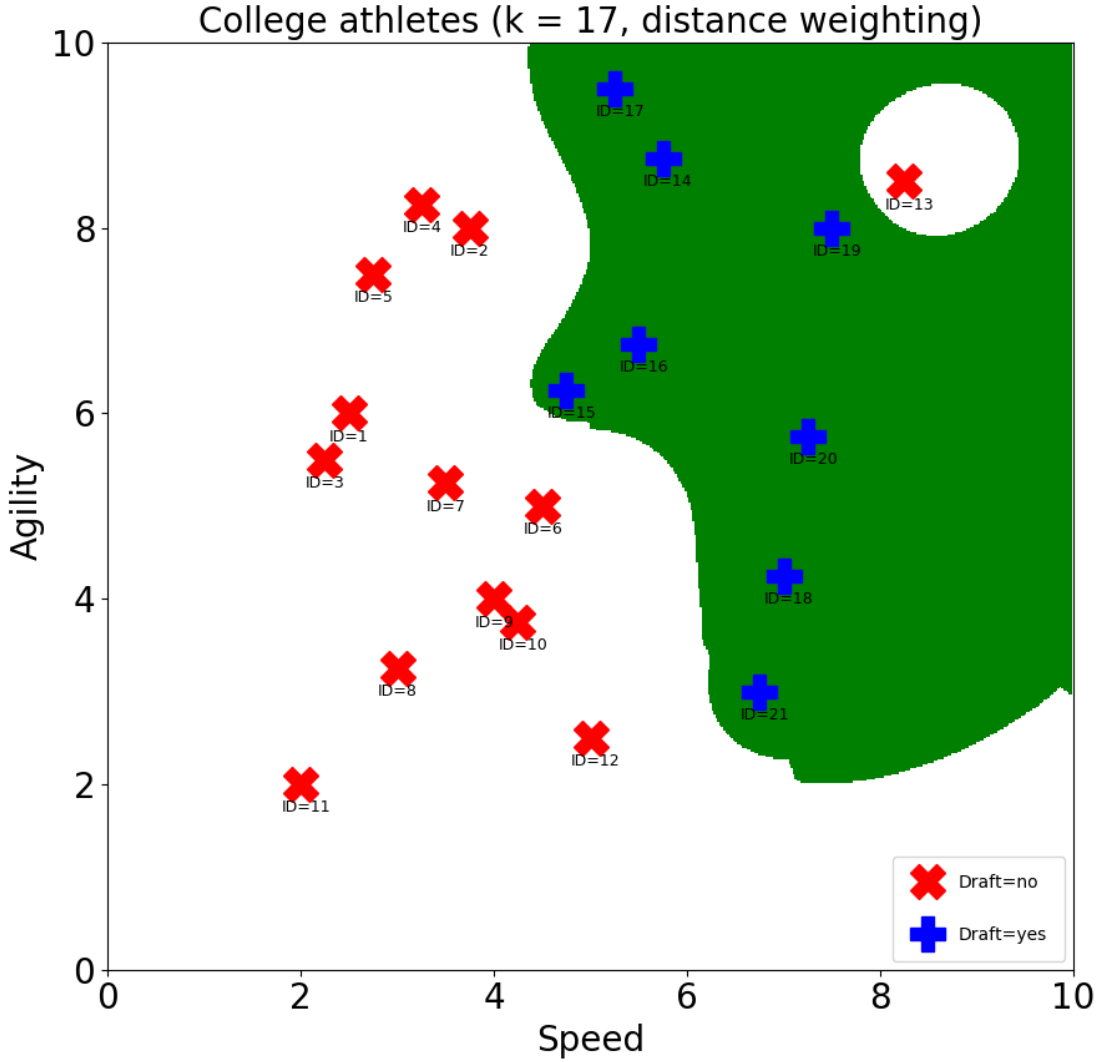
Hastie, Tibshirani & Friedman, Figs 2.1. Will cover linear regression in a future topic.

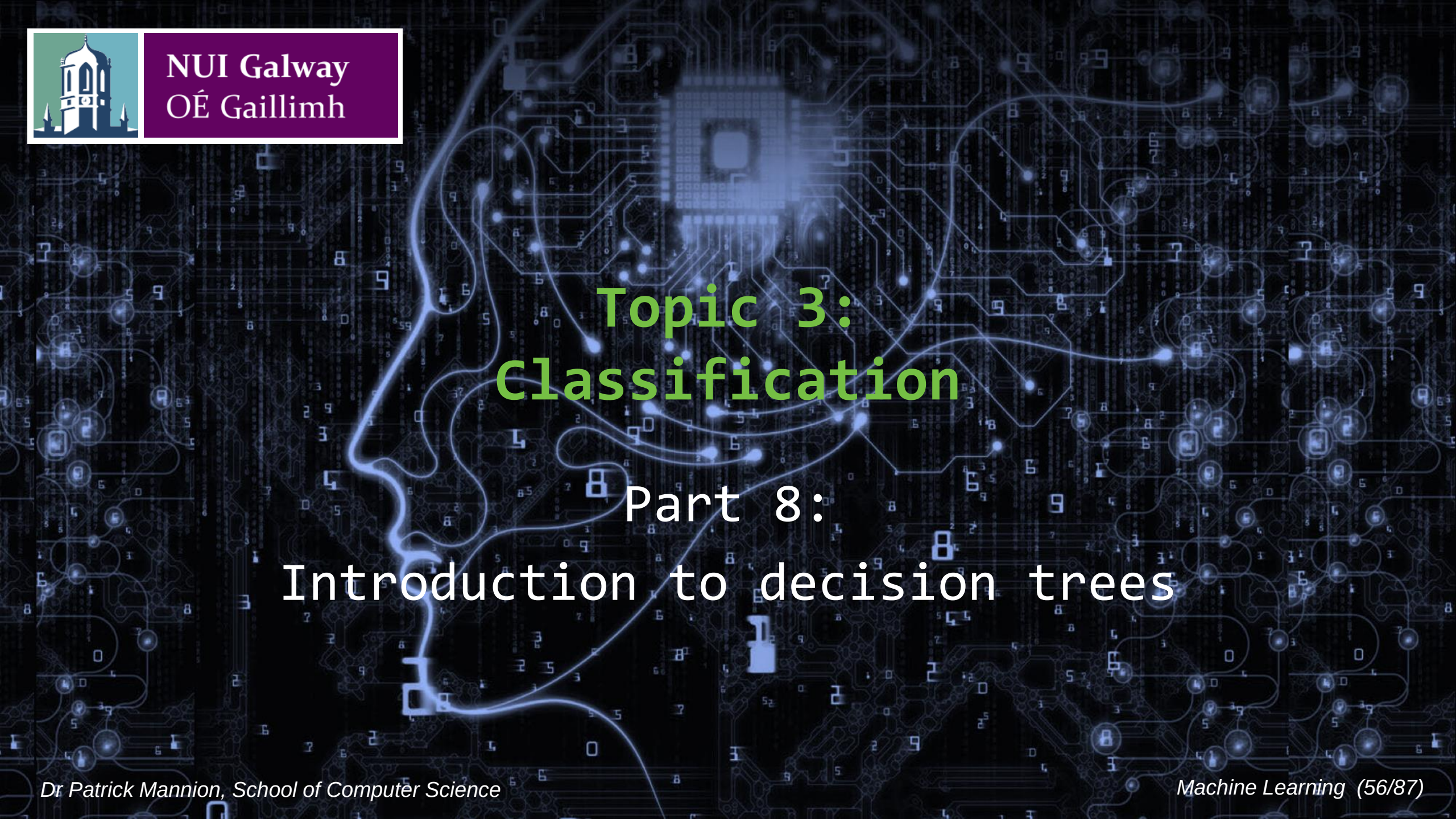
Distance-weighted kNN

- Distance-Weighted kNN
  - Give each neighbour weight: inverse of distance from target
  - Use weighted vote or weighted average
  - Reasonable to use $k = |all\ training\ cases|$

Effect of distance weighting





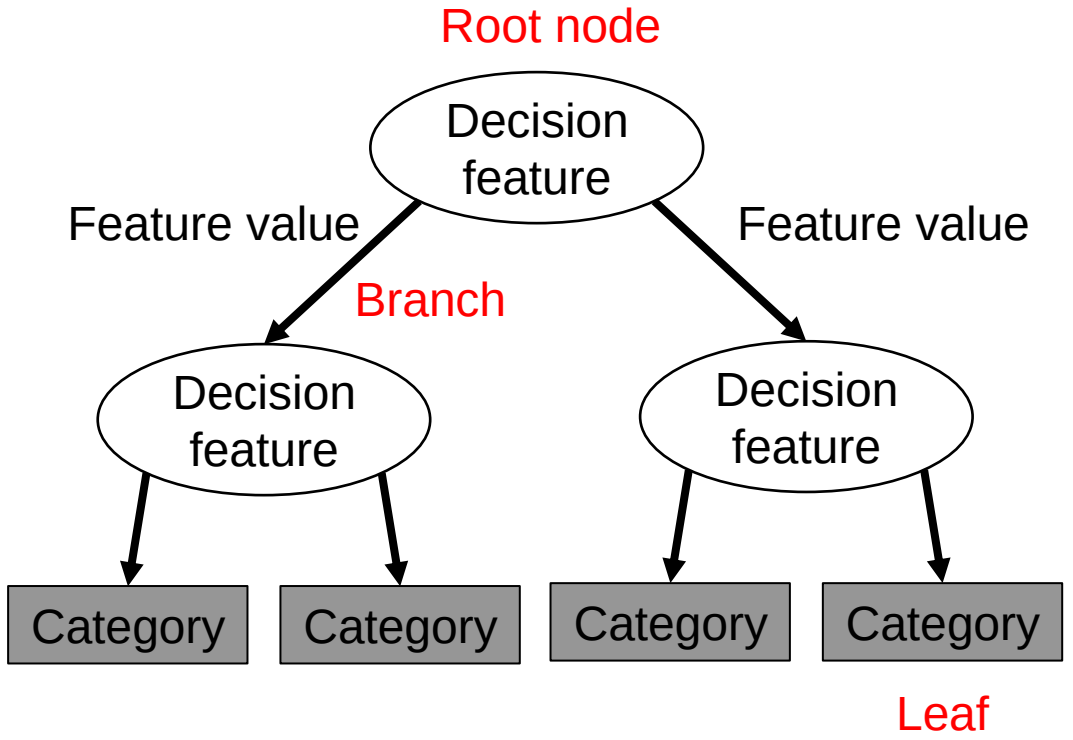


Topic 3: Classification

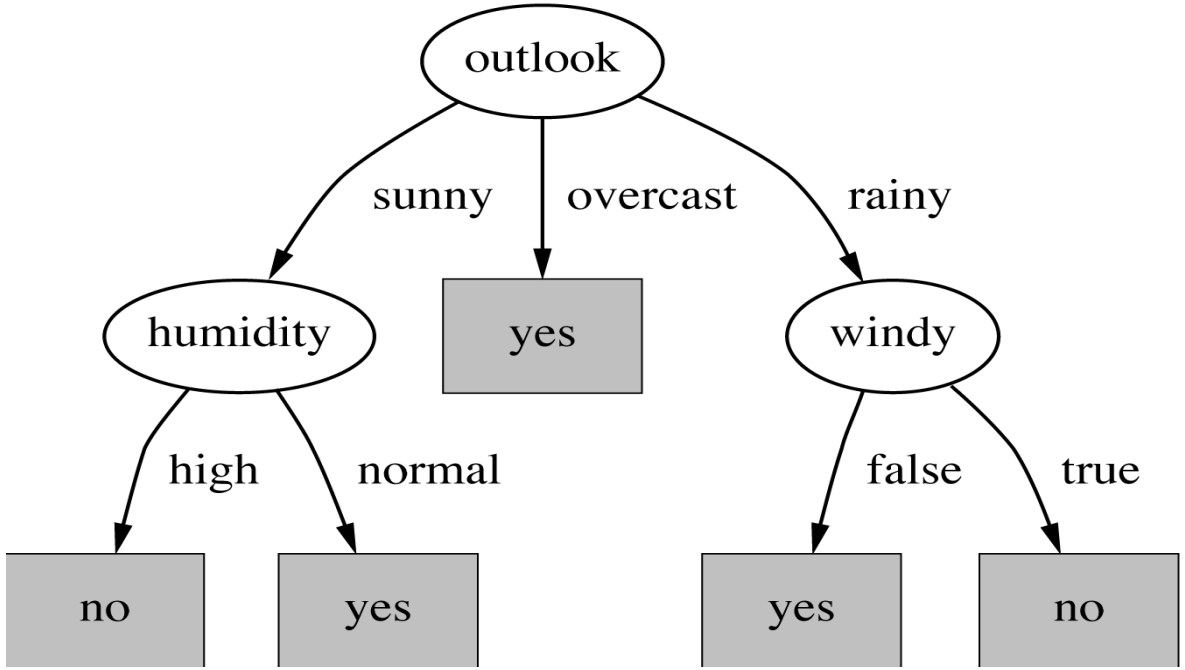
Part 8: Introduction to decision trees

Decision trees

- Decision trees are a fundamental structure used in information-based machine learning
- Main idea: use a decision tree as a predictive model, to decide what category/label/class an item belongs to based on the values of its features
- So-called due to their tree-like structure:
  - A node (where two branches intersect) is a decision point. Nodes partition the data.
  - observations about an item (values of features) are represented using branches
  - The terminal nodes are called leaves; these specify the target label for an item



Decision tree for a sample dataset



Example dataset for induction (1)

- Weather dataset
  - Four attributes:
    outlook: sunny / overcast / rainy
    temperature: hot / mild / cool
    humidity: high / normal
    windy: true / false
  - Used to decide whether or not to *play tennis*
  - 14 examples in dataset
  - See file **weather.csv** on Blackboard

- Objective:
  - Find hypothesis that *describes the cases* given and can be used to *make decisions* in other cases
  - Express the hypothesis as a decision tree.

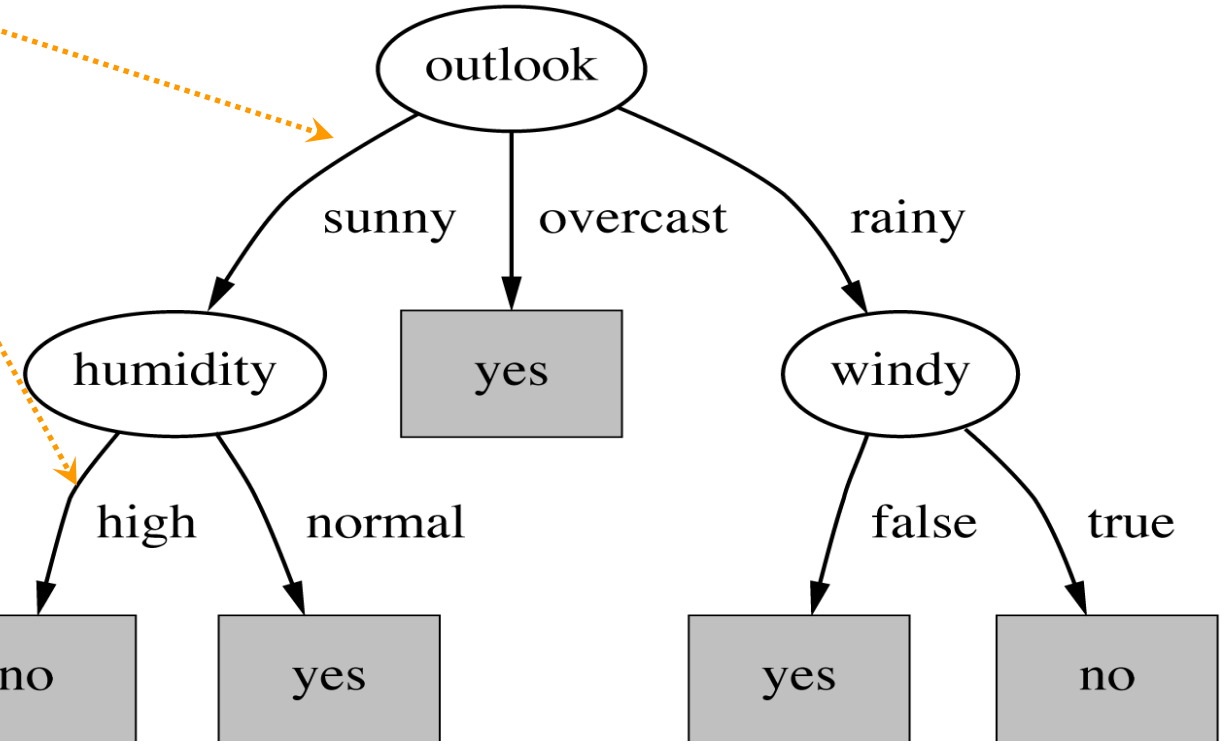
Example dataset for induction (2)

Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

Decision tree for this data (1)

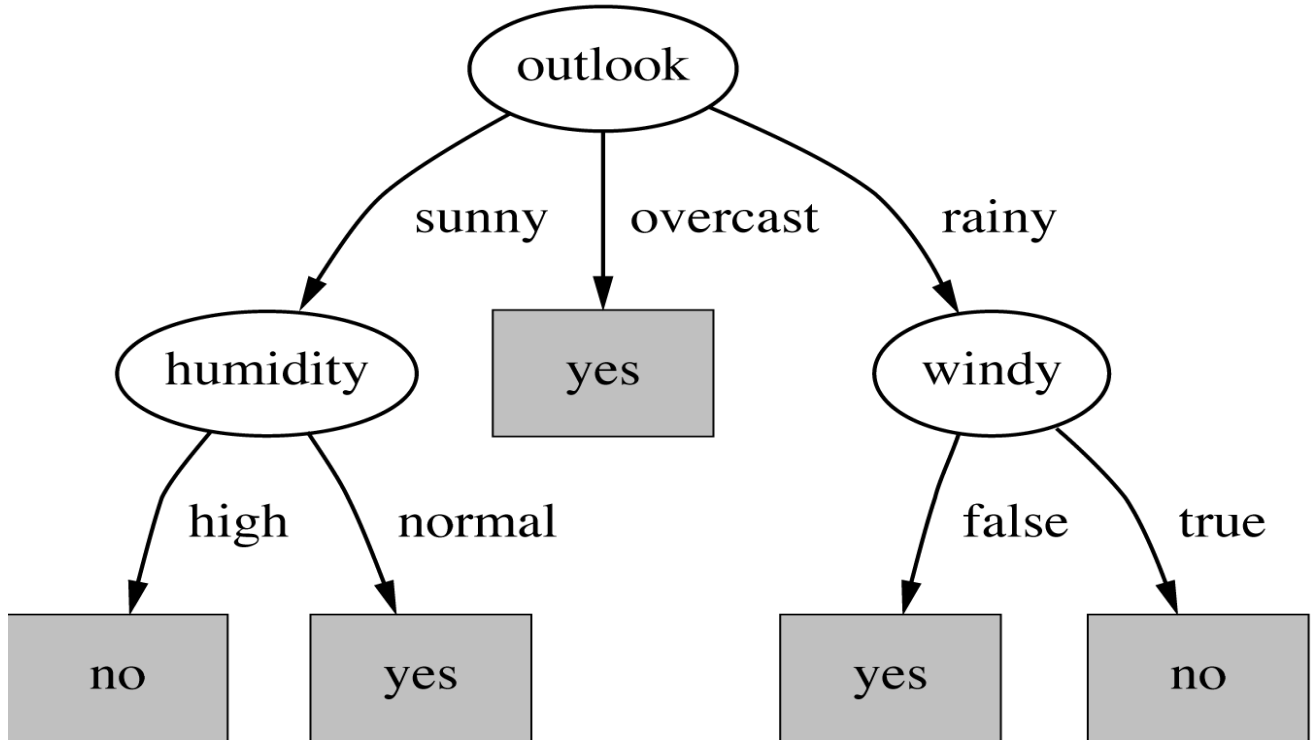
Outlook	sunny
Temp	hot
Humidity	high
Windy	false
Play?	no



Decision tree for this data (2)

Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no



Inductive learning of a decision tree

Step 1

- For all attributes that have not yet been used in the tree, calculate their impurity (**entropy or Gini index)** and **information/Gini gain** values for the training samples

Step 2

- Select the attribute that has the highest information gain

Step 3

- Make a tree node containing that attribute

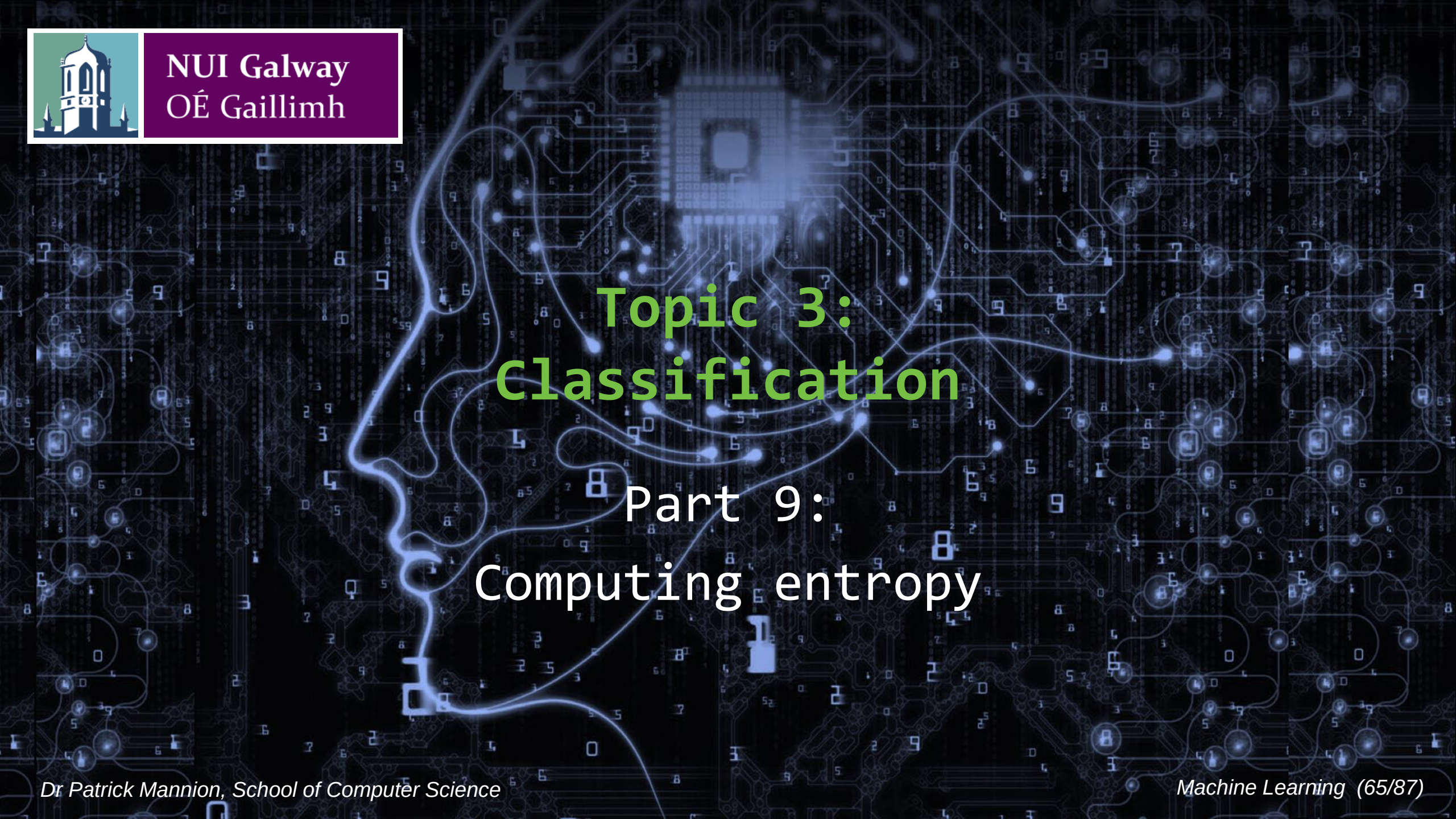
Repeat

- This node **partitions** the data:
  apply the algorithm **recursively** to each partition

Decision trees in scikit-learn

- The main class used in scikit-learn to implement decision tree learning for classification tasks is `DecisionTreeClassifier` https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- An overview of decision tree learning in scikit-learn is available at https://scikit-learn.org/stable/modules/tree.html

- N.B. the default measure of impurity in scikit-learn is the Gini index, but entropy is an option also. In the lecture notes and .ipynb sample we will see examples of both

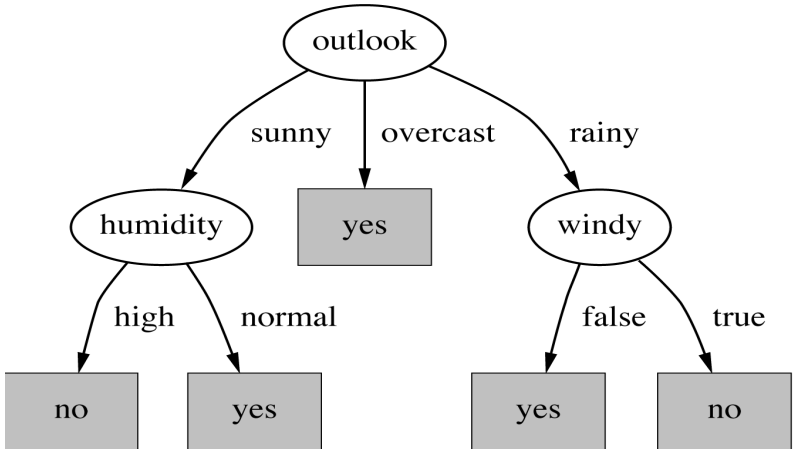


Topic 3: Classification

Part 9: Computing entropy

Motivation

- We already saw how some descriptive features can more effectively discriminate between (or predict) classes which are present in the dataset
- Decision trees partition the data at each node, so it makes sense to use features which have higher discriminatory power "higher up" in a decision tree.
- Therefore, we need to develop a formal measure of the discriminatory power of a given attribute
- Information gain – this can be calculated using entropy (we can also use Gini Gain)

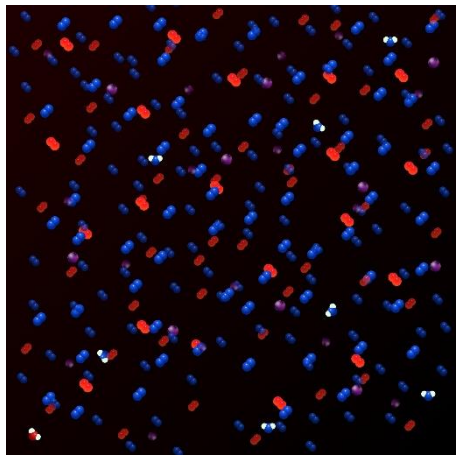


Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

Entropy

- Claude Shannon (often referred to as “the father of information theory”) proposed a measure to of the impurity of the elements in a set, referred to as entropy
- Entropy may be used to measure of the uncertainty of a random variable
- The term entropy generally refers to disorder or uncertainty, so the use of this term in the context of information theory is analogous to the other well-known use of the term in statistical thermodynamics
- Acquisition of information (information gain) corresponds to a reduction in entropy
- “Information is the resolution of uncertainty” (Shannon)
- 1948 article “A Mathematical Theory of Communication”



Calculating entropy

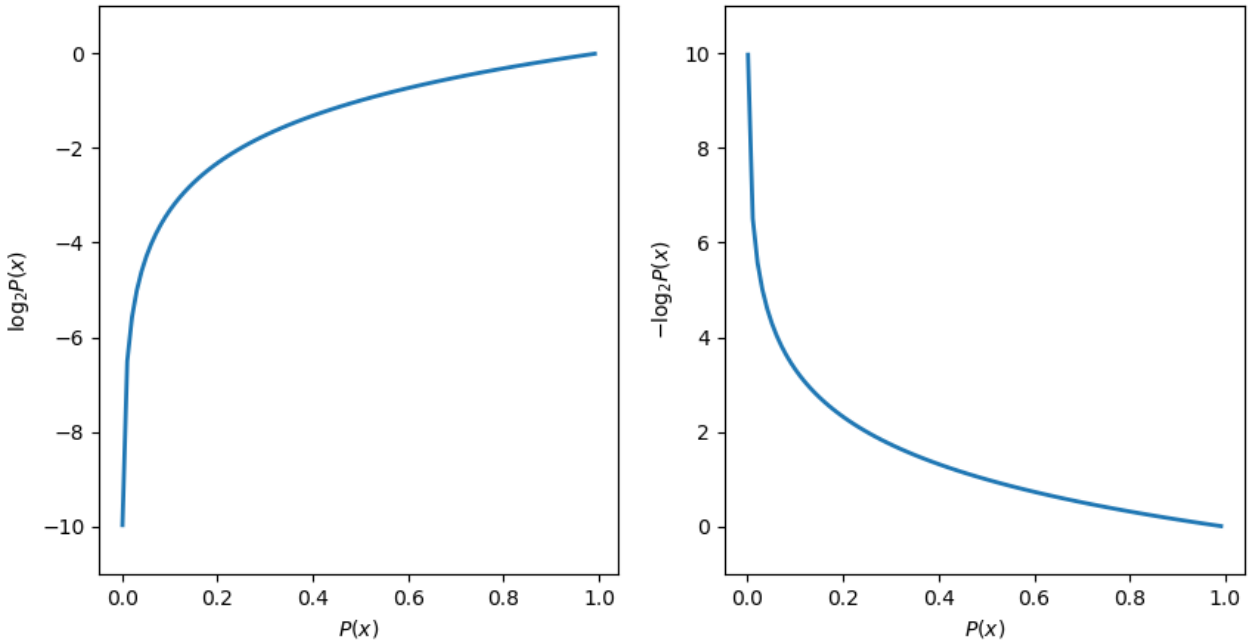
- The entropy of a dataset S with n different classes may be calculated as:

$$\text{Ent}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

- Here p_i is the proportion of class *i* in the dataset.
- This is an example of a probability mass function
- Entropy is typically measured in bits (note $\log_2$ in the equation above)
- The lowest possible entropy output from this function is 0 ($\log_2 1 = 0$)
- The highest possible entropy is $\log_2 n$ (=1 when there are only 2 classes)

Why use the binary logarithm?

- A useful measure of uncertainty should:
  - Assign high uncertainty values to outcomes with a low probability
  - Assign low uncertainty values to outcomes with a high probability
- Consider the plot to the right
  - log_2 returns large negative values when P is close to 0
  - log_2 returns small negative values when P is close to 1
- Using $-log_2$ is more convenient, as this will give positive entropy values, with 0 as the lowest entropy



Entropy worked example 1

$$\text{Ent}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

$$\text{Ent}(S) = \text{Ent}([9+,5-])$$

$$\text{Ent}(S) = -9/14 \log_2(9/14) - 5/14 \log_2(5/14)$$

$$\text{Ent}(S) = 0.9403$$

If calculating this in a spreadsheet application such as Excel, make sure that you are using $\log_2$ (e.g. LOG(9/14, **2**))

Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

Entropy worked example 2

Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

Entropy worked example 2

$$\text{Ent}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

$\text{Ent}(S_{\text{windy=false}}) = \text{Ent}([6+,2-])$

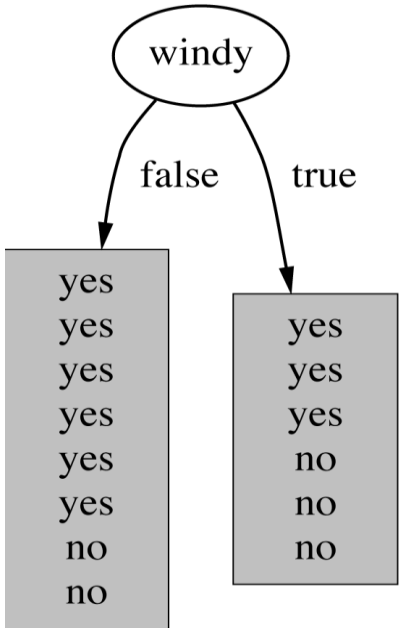
$= -6/8 \log_2(6/8) - 2/8 \log_2(2/8)$

$= 0.3112+0.5 = 0.8112$

$\text{Ent}(S_{\text{windy=true}}) = \text{Ent}([3+,3-] =$

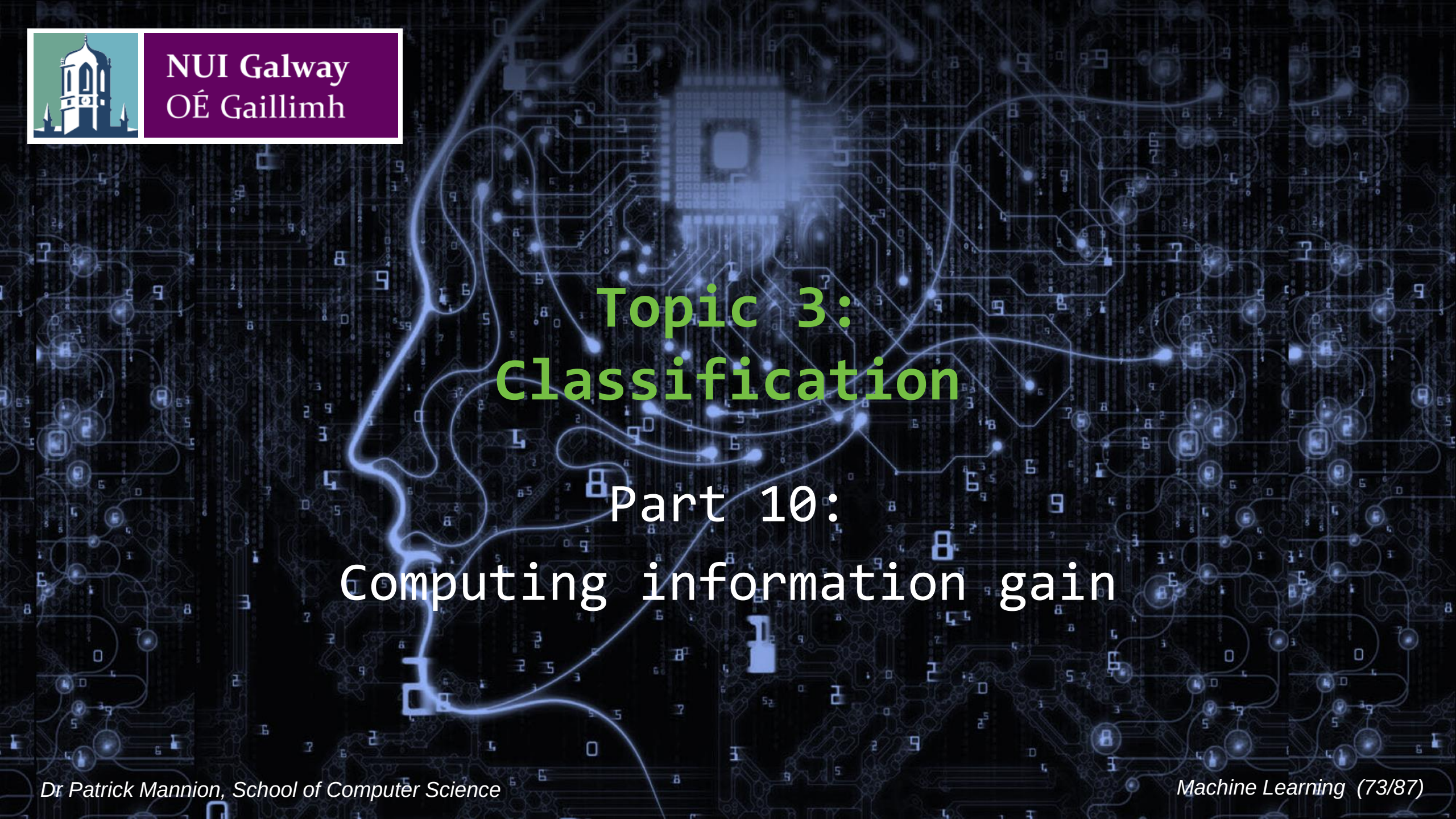
$= -3/6 \log_2(3/6) -3/6 \log_2(3/6)$

$= 0.5+0.5 = \mathbf{1.0}$



Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no



Topic 3: Classification

Part 10: Computing information gain

Information gain

- The **information gain** of an attribute is the reduction in entropy from partitioning the data according to that attribute

$$\text{Gain}(S, A) = \text{Ent}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Ent}(S_v)$$

- Here S is the entire set of data being considered, and S_v refers to each partition of the data according to each possible value v for the attribute
- $|S|$ and $|S_v|$ refer to the cardinality or size of the overall dataset, and the cardinality or size of a partition respectively
- When selecting an attribute for a node in a decision tree, use whichever attribute A gives the greatest information gain

Information gain worked example

$$\text{Gain}(S, A) = \text{Ent}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Ent}(S_v)$$

$|S|=14$

$|S_{\text{windy=true}}|=6$

$|S_{\text{windy=true}}|=8$

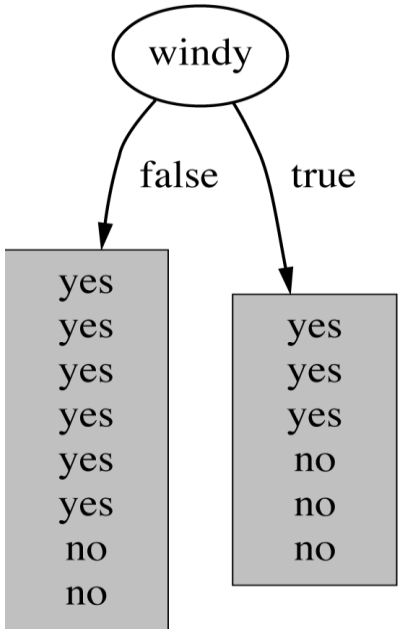
Gain(S, Windy)

$= \text{Ent}(S) - |S_{\text{windy=true}}|/|S| \text{ Ent}(S_{\text{windy=true}}) - |S_{\text{windy=false}}|/|S| \text{ Ent}(S_{\text{windy=false}})$

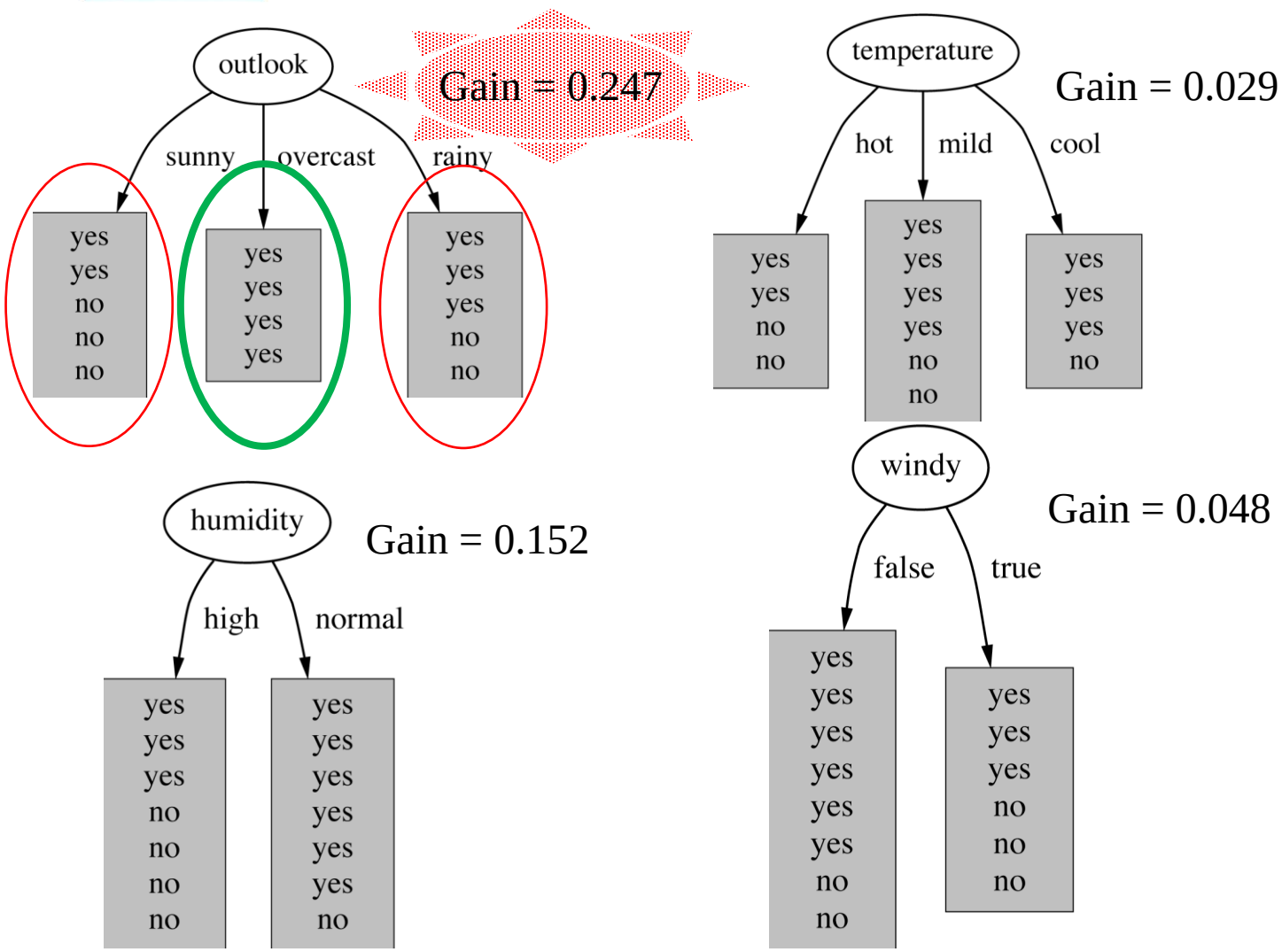
$= \text{Ent}(S) - (6/14) \text{ Ent}([3+,3-]) - (8/14) \text{ Ent}([6+,2-])$

$= 0.940 - (6/14)\ 1.00 - (8/14)\ 0.811$

Gain(S, Windy) = 0.048



Best partitioning = highest information gain

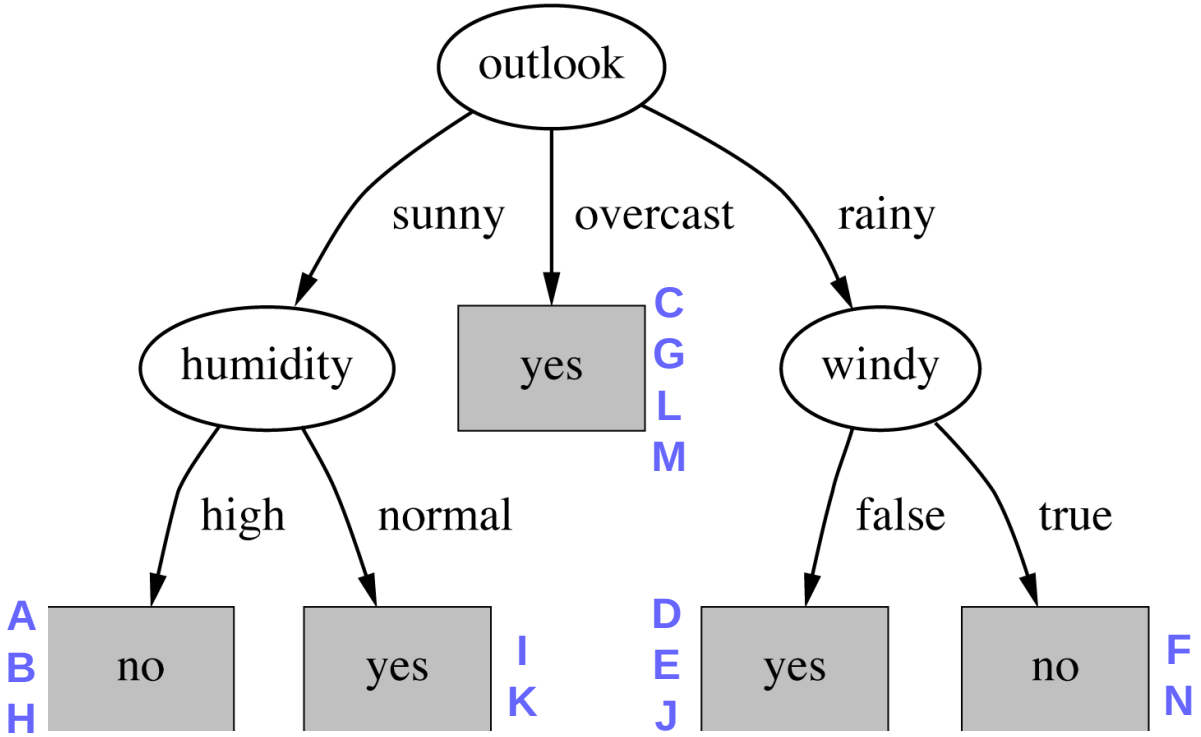


Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

Having found the best split for the root node, repeat the whole procedure with each subset of examples …
S will now refer to the subset in the partition being considered, instead of the entire dataset

Example: complete decision tree



Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no

What about Temp = {Hot, Mild, Cool}?

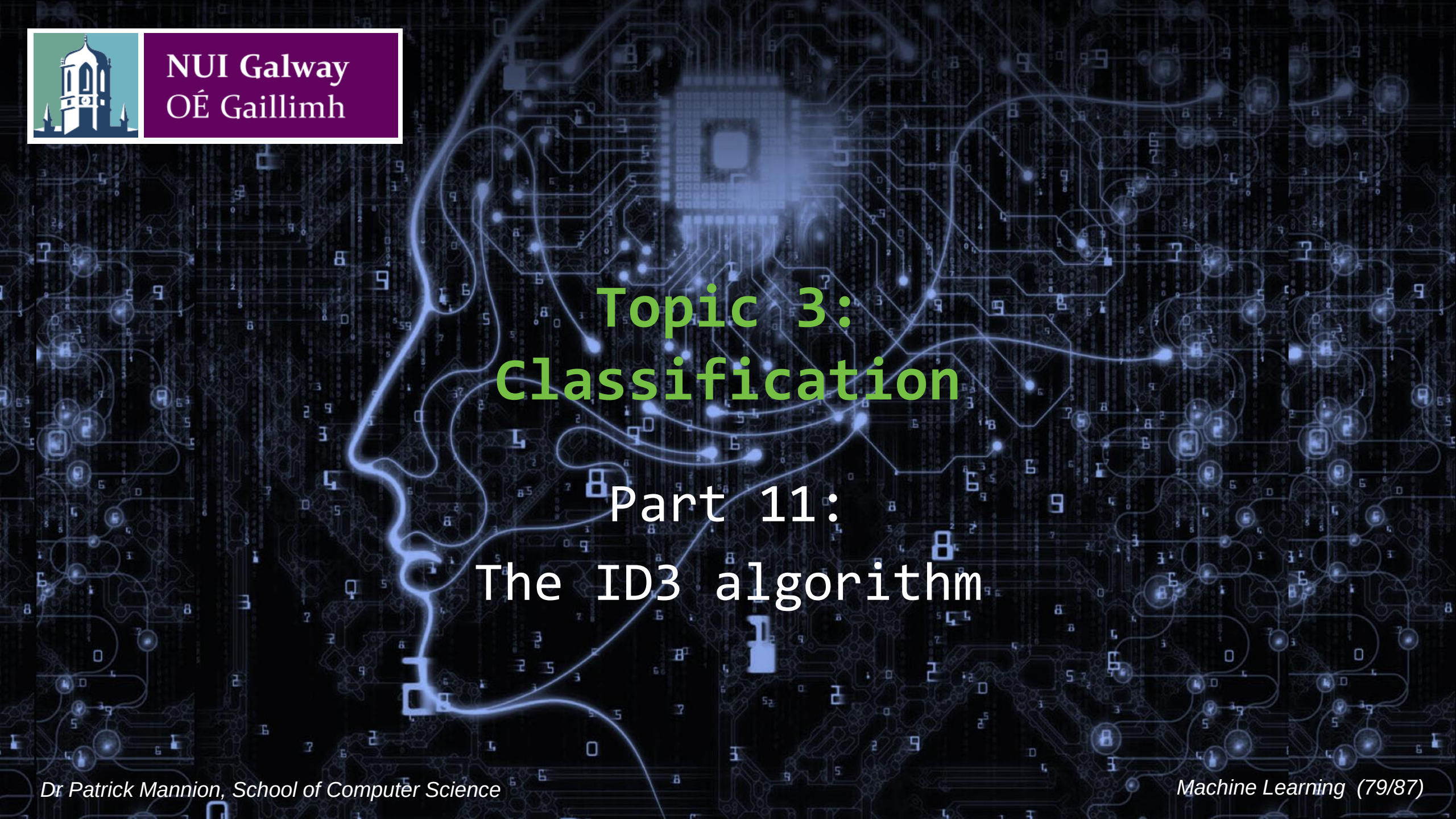
Selecting the best attribute: an alternative metric

- One alternative is to use the **Gini index** instead of entropy as a measure of the impurity of a set (this is the default in scikit-learn)

$$\text{Gini}(S) = 1 - \sum_{i=1}^{n} {p_i}^2$$

- Then the gain for a feature may be calculated based on the reduction in the Gini index (rather than a reduction in entropy):

$$\text{GiniGain}(S, A) = \text{Gini}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Gini}(S_v)$$

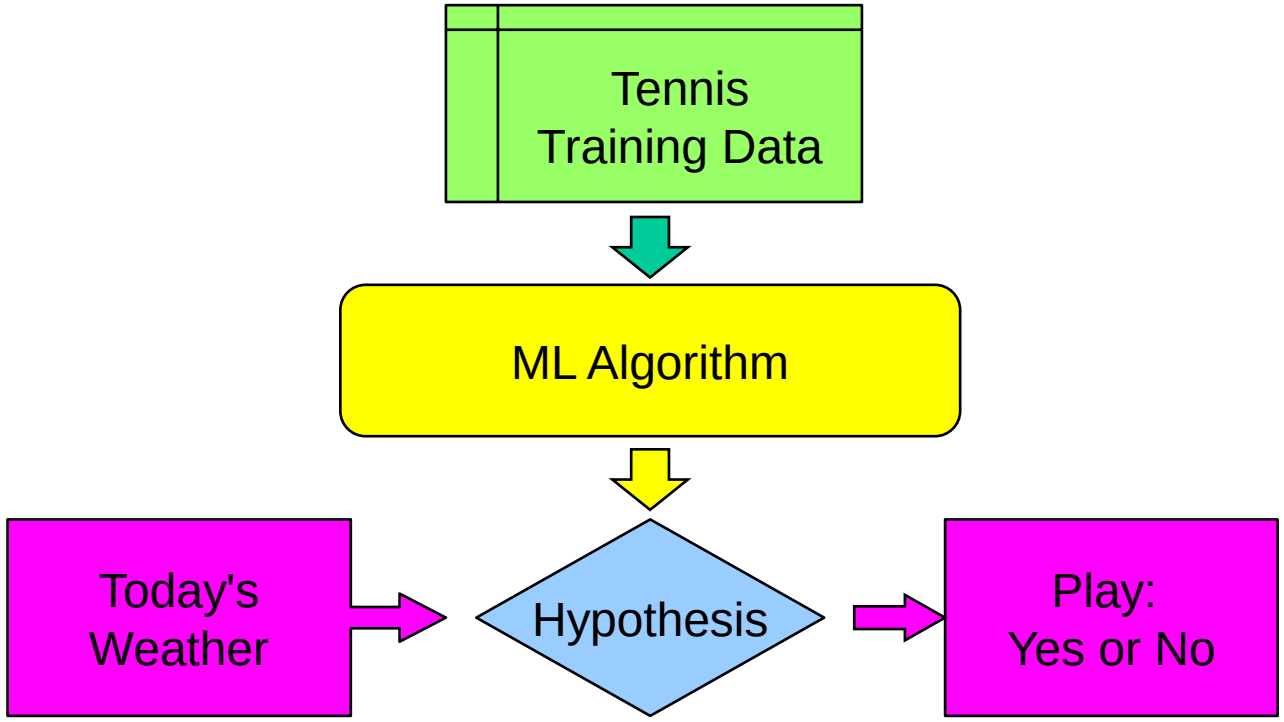


NUI Galway
OÉ Gaillimh

Topic 3: Classification

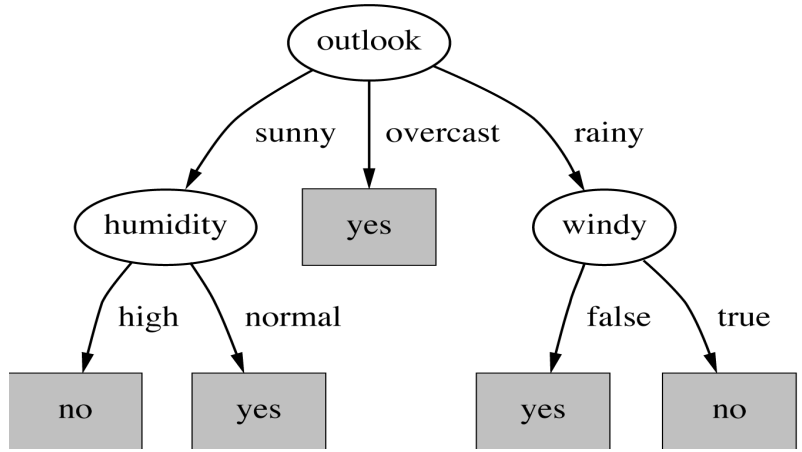
Part 11: The ID3 algorithm

Review: the supervised learning process



Anyone for Tennis?

ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy	mild	normal	false	yes
K	sunny	mild	normal	true	yes
L	overcast	mild	high	true	yes
M	overcast	hot	normal	false	yes
N	rainy	mild	high	true	no



Review: inductive learning of a decision tree

Step 1
- For all attributes that have not yet been used in the tree, calculate their **entropy** and **information gain** values for the training samples

Step 2
- Select the attribute that has the highest information gain

Step 3
- Make a tree node containing that attribute

Repeat
- This node **partitions** the data:
  apply the algorithm **recursively** to each partition

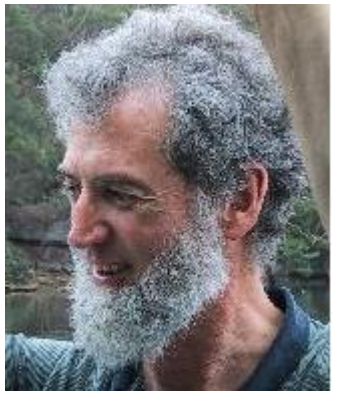
The ID3 algorithm

```
1. ID3(Examples, Attributes, Target):
2. Input: Examples:    set of classified examples
3.        Attributes: set of attributes in the examples
4.        Target:     classification to be predicted
5. if Examples is empty then return a Default class
6. else if all Examples have same class then return this class
7. else if all Attributes are tested then return majority class
8. else:
9.     let Best = attribute that best separates Examples relative to Target
10.    let Tree = new decision tree with Best as root node
11.    foreach value v_i of Best:
12.        let Examples_i = subset of Examples that have Best=v_i
13.        let Subtree = ID3(Examples_i, Attributes-Best, Target)
14.        add branch from Tree to Subtree with label v_i
15.    return Tree
```

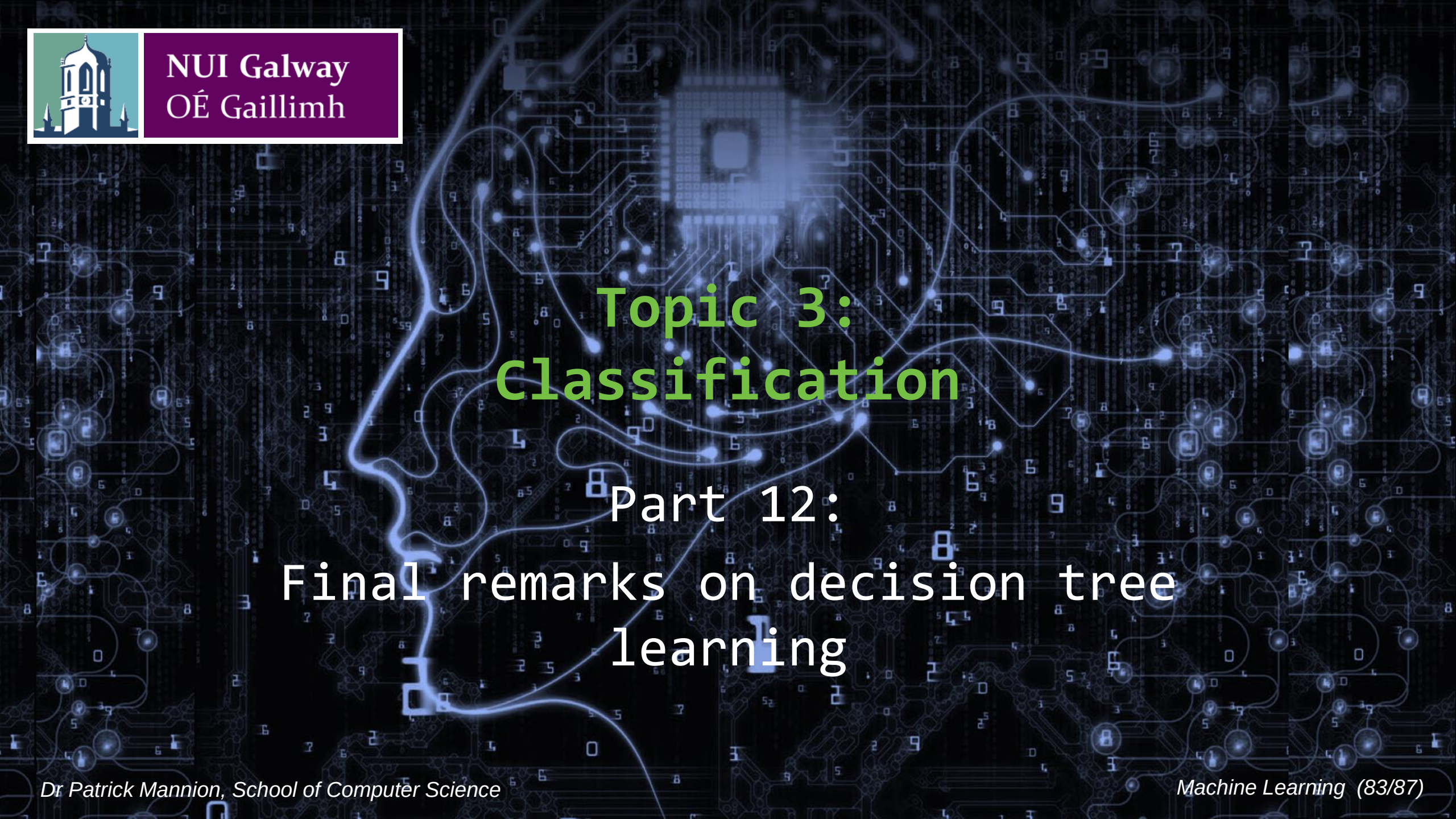
BASE CASES

RECURSIVE CALL

Ross Quinlan, 1986



Based on Algorithm **Decision-Tree-Learning** in Russell & Norvig textbook



NUI Galway
OÉ Gaillimh

Topic 3: Classification

Part 12: Final remarks on decision tree learning

Decision tree characteristics

- Popular because:
  - Relatively **easy** algorithm
  - **Fast**: greedy search without backtracking
  - **Comprehensible** output:
    important in decision-making (medical, financial, …)
  - **Practical**: discrete/numeric, irrelevant attributes, noisy data, …
- Expressiveness: what functions can a DT represent?
  - Technically, any Boolean function (propositional logic)
  - Some functions, however, require exponentially large tree
    (e.g. parity function)
  - Cannot consider relationships between two attributes

Dealing with noisy or missing data

- What about inconsistent ("noisy") data?
  - Use majority class (line 7 of ID3 alg.)

    ```
    7. else if all Attributes are tested then return majority class
    ```

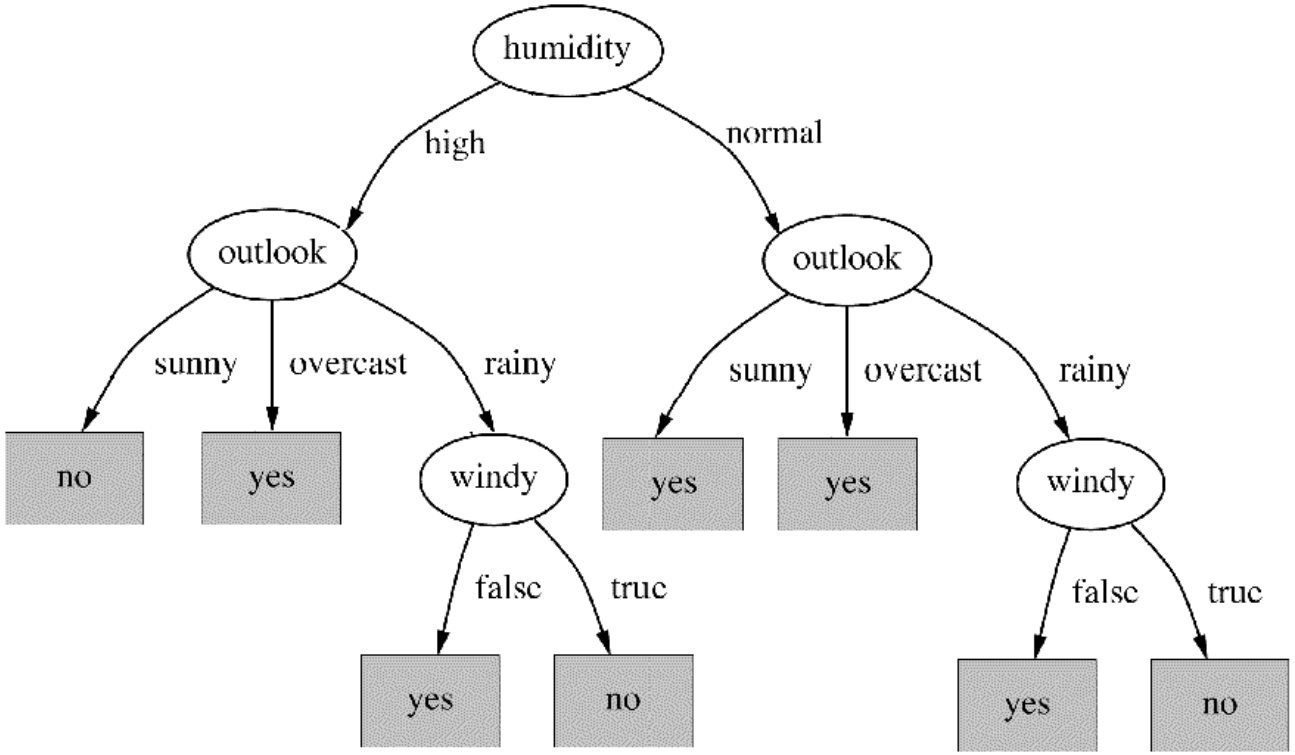
  - or interpret as probabilities
  - or return “average” target feature value

- What about missing data?
  - Given a complete decision tree, how should one classify an example that is missing one of the test attributes?
  - How should one modify the information gain formula when some training examples have unknown values for an attribute?
  - Could assign the most common value among the training examples that reach that node
  - Or could assume the attribute has all possible values, weighting each value according to its frequency among the training examples that reach that node

Instability of decision trees

- Hypothesis found is sensitive to training set used
  - consequence of greedy search
- Replace one example:
  - new one consistent with original tree
- Some algorithmic modifications to reduce the instability of decision tree learning were proposed by Li and Belford in their 2002 paper “Instability of decision tree classification algorithms”.
- Li and Belford’s main idea is to alter the attribute selection procedure, so that the tree learning algorithm is less sensitive to some % of the training dataset being replaced.

ID	Outlook	Temp	Humidity	Windy	Play?
~~C~~	~~overcast~~	~~hot~~	~~high~~	~~false~~	~~yes~~
O	sunny	hot	normal	true	yes



Pruning

- Overfitting occurs in a predictive model when the hypothesis learned makes predictions which are based on spurious patterns in the training data set. Consequence: poor generalisation to new examples.
- Overfitting may happen for a number of reasons, including sampling variance or noise present in the dataset
- **Tree pruning** may be used to combat overfitting in decision trees
- Tree pruning can lead to induced trees which are inconsistent with the training set
- Generally, there are two different approaches to pruning:
  - Pre-pruning (e.g. <= target # of examples in a partition, limiting tree depth, creating a new node only when information gain is above a threshold, statistical tests such as χ^2)
  - Post-pruning (e.g. target # of examples, compare error rate for model on a validation dataset with and without a given subtree; only keep a subtree if it improves the error rate, statistical tests such as χ^2, reduced error pruning (Quinlan, 1987))