



# Introduction to Machine Learning

Part 1:  
Learning objectives and overview



## Learning objectives for this section

Having completed this section successfully, you will be able to ...

1. Discuss definitions of Machine Learning
2. Describe what major categories of ML task entail: classification, regression, clustering, relationship discovery and reinforcement learning
3. Discuss the relationship with Data Mining
4. Explain the Data Mining process
5. Consider current and future applications of Machine Learning and Data Mining



## Prerequisites

- This is for students who already have a degree or substantial experience in computer science, software development or a closely related subject area
- You need to understand:
  - How to program (any language)
  - Algorithm analysis
  - Basic statistics and probability
  - Knowledge of standard mathematical notation (i.e. how to read an equation)



## Resources

- Course slides:
  - Necessary but insufficient!
- Recommended books:
  - List available on Blackboard
  - Will also provide references in individual sections
- Others:
  - Andrew Ng's Coursera Machine Learning Course
  - Sebastian Thrun's Udacity AI Course
  - Contributions welcome!  
**If you find useful links, email them either to me ([patrick.mannion@nuigalway.ie](mailto:patrick.mannion@nuigalway.ie)) or to Prof Michael Madden ([michael.madden@nuigalway.ie](mailto:michael.madden@nuigalway.ie)).**



## Overview of topic

1. Learning objectives and overview
2. What is Machine Learning?
3. Types of Machine Learning task
4. Overview of Data Mining
5. Applications of Machine Learning



# Introduction to Machine Learning

Part 2:

## What is Machine Learning?



# What is Machine Learning? [1]

- Samuel, 1959:
  - "Field of study that gives computers the ability to learn without being explicitly programmed"
- Witten & Frank, 1999:
  - Learning is changing behaviour in a way that makes **performance** better in the future

Arthur Samuel, 1901-1990

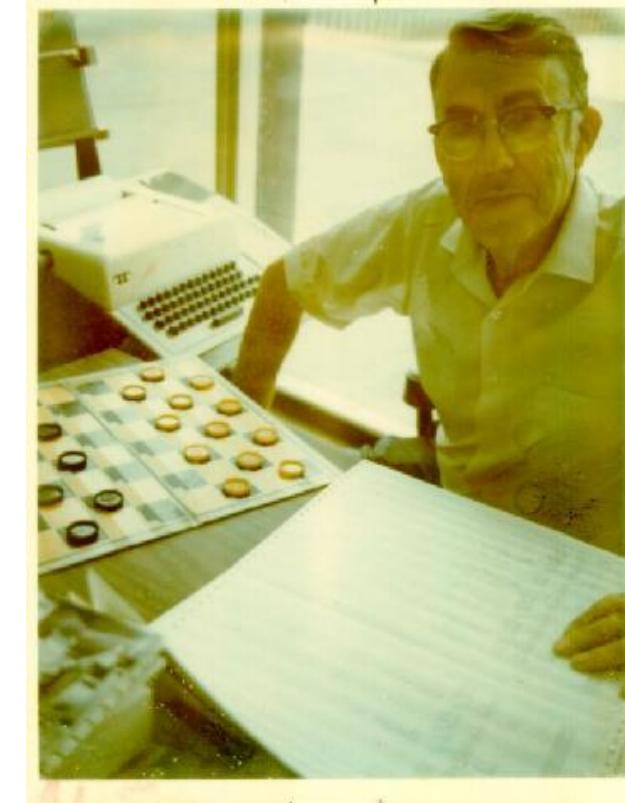
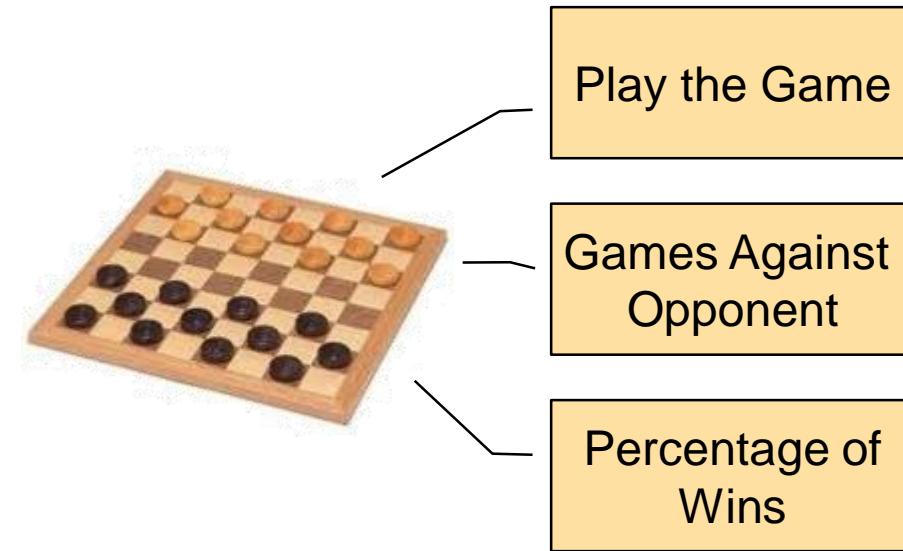


Image source: <http://www.computer.org/portal/web/awards/cp-samuel>



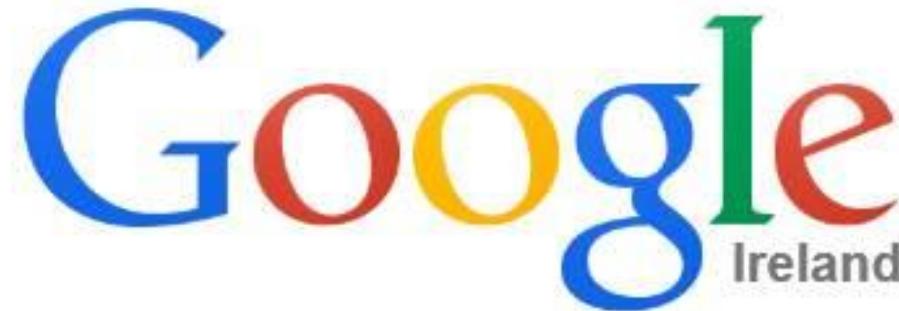
## What is Machine Learning? [2]

- Mitchell, 1997:
  - Improvement with experience at some task
  - A well-defined ML problem:
    - Improve over task  $T$
    - wrt **performance** measure  $P$
    - based on experience  $E$
  - For draughts/checkers example, what are  $T$ ,  $P$ ,  $E$ ?
- Other possible definitions
  - Philosophical and psychological considerations ...
  - Relationship to Artificial Intelligence generally ...
  - Artificial Intelligence  $\neq$  Machine Learning  $\neq$  Deep Learning
  - **Artificial Intelligence  $\supsetneq$  Machine Learning  $\supsetneq$  Deep Learning**





## What is Machine Learning? [3]



machine learning is |

machine learning is **bullshit**

machine learning is **hard**

machine learning is **fun**

machine learning is **the future**

machine learning is **not as cool as it sounds**

machine learning is

machine learning is **math**



# Introduction to Machine Learning

Part 3:

## Types of Machine Learning task



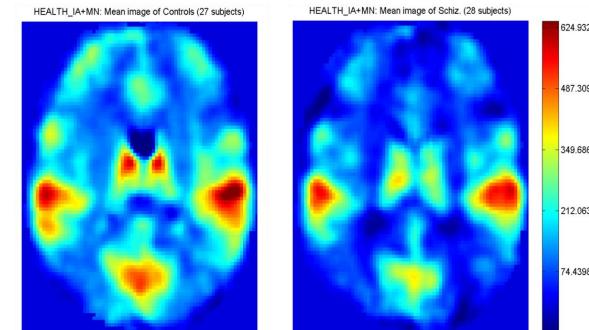
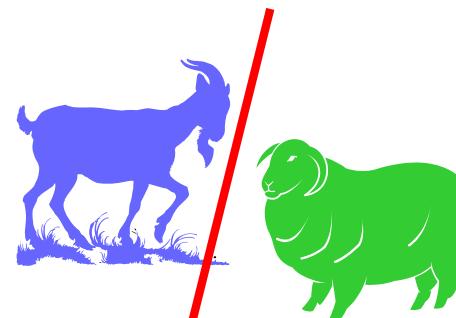
## Machine Learning techniques

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning



# Major Types of Task [1]

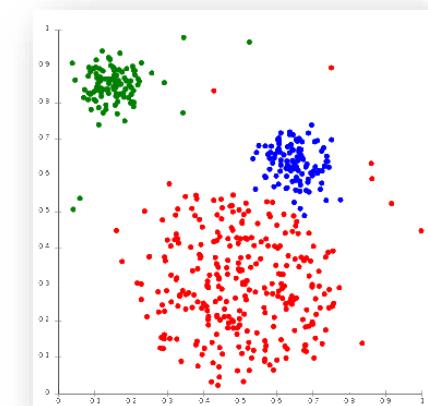
## 1. Classification



## 2. Regression



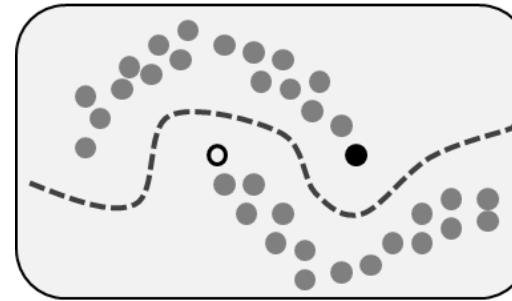
## 3. Clustering





## Major Types of Task [2]

### 4. Co-Training



### 5. Relationship Discovery

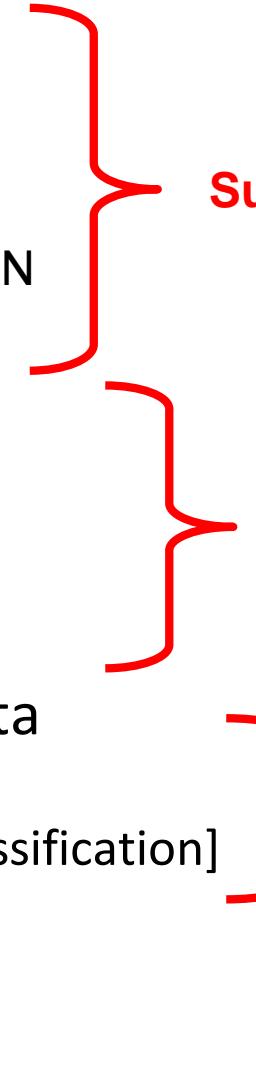
beer  $\Leftrightarrow$  diapers

### 6. Reinforcement Learning





## Techniques for these Tasks

- Classification  
Decision trees, SVMs
  - Regression  
Linear Regression, Neural nets; k-NN  
(good for Classification too)
  - Clustering  
k-Means, EM-clustering
  - Relationship Discovery  
Association Rules; Bayesian nets
  - Learning From Part-Labelled Data  
Co-Training; Transductive Learning  
[Combines ideas from clustering & classification]
  - Reinforcement Learning  
Q-Learning, SARSA
- 
- The techniques listed are grouped into four categories using red curly braces:
- Supervised**: Includes Classification and Regression.
  - Unsupervised**: Includes Clustering and Relationship Discovery.
  - Semi-supervised**: Includes Learning From Part-Labelled Data.
  - Reward-based**: Includes Reinforcement Learning.



## What do these have in common?

- In all cases, machine searches for a **hypothesis** that best describes the data presented to it
- Choices to be made:
  - How is hypothesis expressed?  
*mathematical equation, logic rules, diagrammatic form, table, parameters of a model (e.g. weights of an ANN), ...*
  - How is search carried out?  
*systematic (breadth-first or depth-first), heuristic (most promising first), ...*
  - How do we measure quality of hypothesis?
  - What is appropriate format for data?
  - How much data is required?



# What else do we need to know about?

- To apply ML:
  - How to formulate a problem
  - How to prepare the data
  - How to select an appropriate algorithm
  - How to interpret the results
- To evaluate results and compare methods:
  - Separation between training, testing & validation
  - Performance measures:  
simple metrics, statistical tests, graphical methods
  - To improve performance
  - Ensemble methods
  - Theoretical bounds on performance



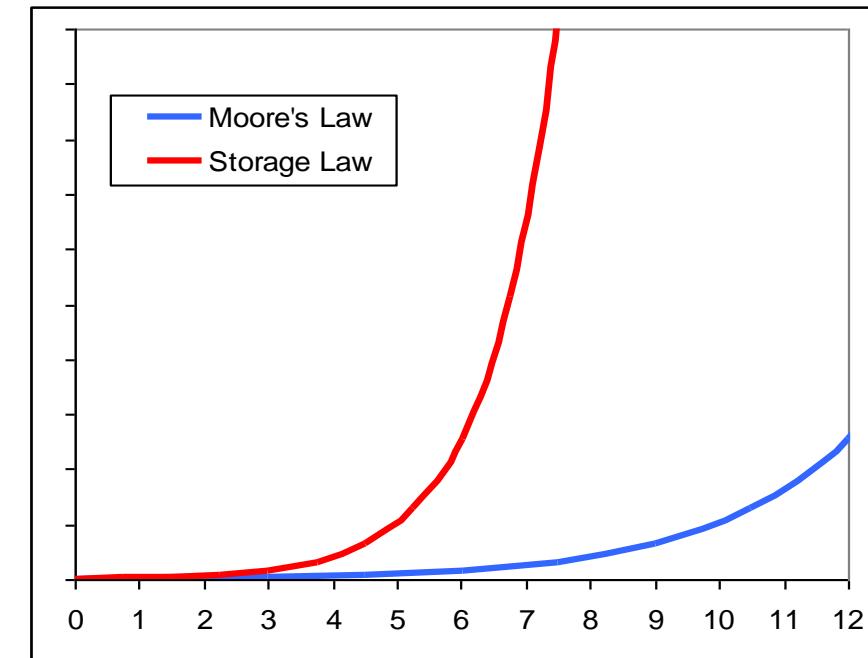
# Introduction to Machine Learning

## Part 4: Overview of Data Mining



# Data Mining: What's the Link?

- Data Mining:
  - Extract **interesting** knowledge from **large unstructured** datasets
  - **non-obvious / comprehensible / meaningful / useful**
- Storage Law (Fayyad & Uthurusamy, Comms.ACM 2002)
  - Storage capacity **doubling** every year
  - Faster than Moore's law
  - Result: write-only “data tombs”
- Developments in ML essential to be able to process and exploit this lost data





# Big Data

---

Data sets of scale and complexity such that they can be difficult to process using current standard methods

- Standard DB tools & data management apps
- Moving target





# Big Data

- Data scale dimensions (One or more of “3 Vs”):
  - **Volume**: terabytes and up
  - **Velocity**: from batch to streaming data
  - **Variety**: numeric, video, sensor, unstructured text ...
- Fashionable to add others that are not key ...
  - Veracity: quality & uncertainty associated with items
  - Variability: change / inconsistency over time
  - Value: for the organisation
- Key techniques:
  - Sampling; inductive learning; clustering; associations
  - Distributed programming methods



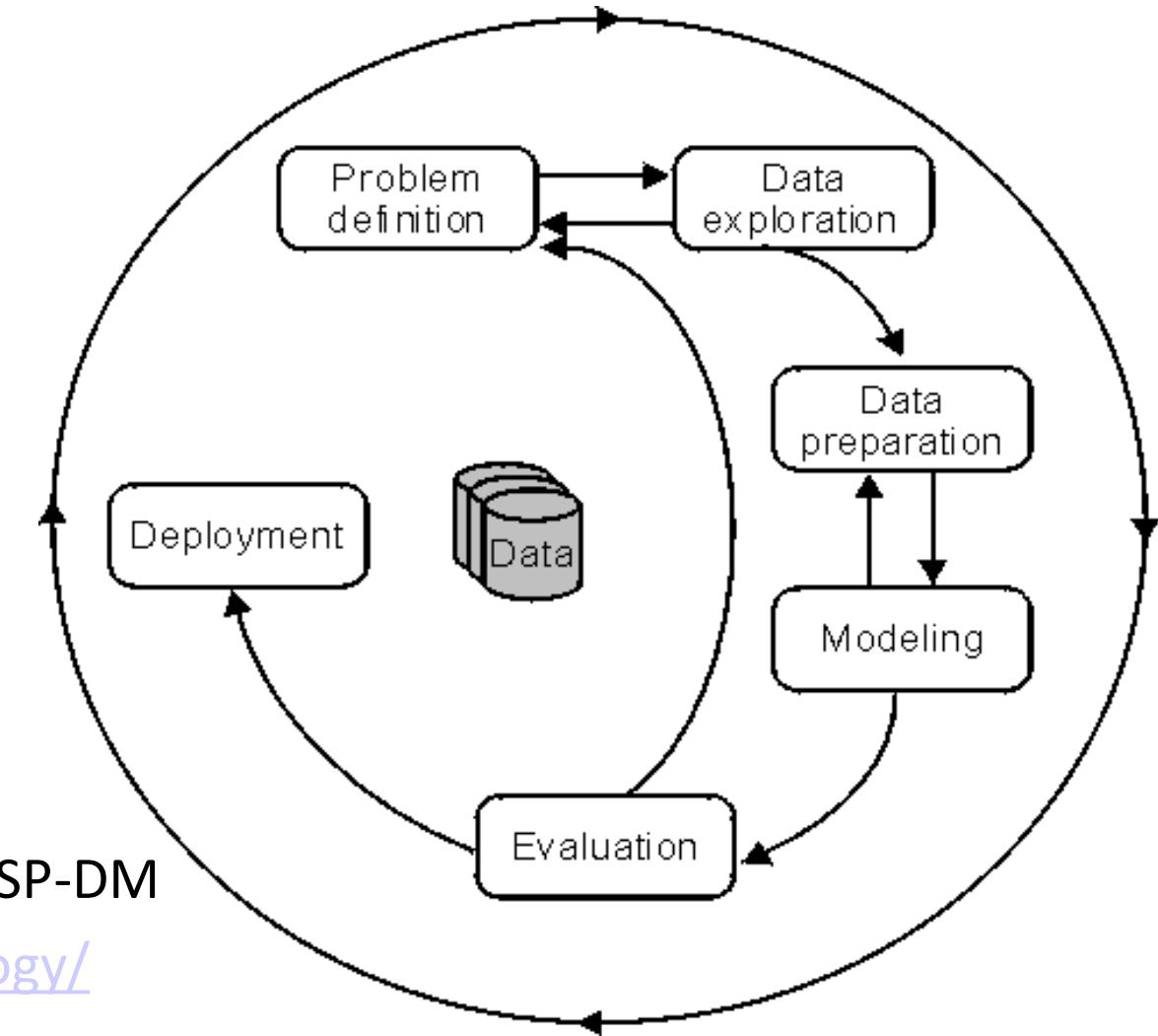
# CRISP-DM Data Mining Process

- Problem Definition
- Data Exploration
- Data Preparation
- Modelling
- Evaluation
- Deployment

Cross Industry Standard Process for  
Data Mining (CRISP-DM) process model

This link gives a summary of the main steps in CRISP-DM

<https://www.sv-europe.com/crisp-dm-methodology/>





# Introduction to Machine Learning

## Part 5: Applications of Machine Learning



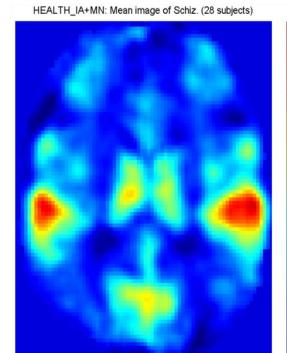
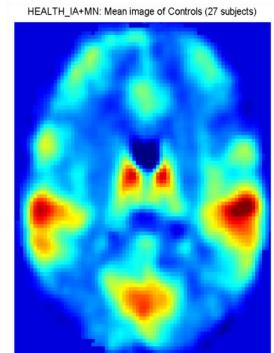
## Current & Emerging Applications

Any ideas?

What companies use ML & DM?



# Users of ML & DM





## High-Profile Examples ...

# Forbes

New Posts  
+4 posts this hour

Most Popular  
Most Disliked Athletes



**Kashmir Hill**, Forbes Staff  
Welcome to The Not-So Private Parts where technology & privacy collide  
[+ Follow](#) (1,178)

TECH | 2/16/2012 @ 11:02AM | 1,930,513 views

## How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did

Forbes, 16 Feb 2012

<http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/>



# How Netflix is turning viewers into puppets

"House of Cards" gives viewers exactly what Big Data says we want. This won't end well

BY ANDREW LEONARD



House of Cards (BBC, 1990)

↔ ★ ★ ★ ★ ★

↔ Kevin Spacey (Actor)

↔ David Fincher (Dir.)

Salon, 1 Feb 2013

<https://www.salon.com/2013/02/01/how.netflix.is.turning.viewers.into.puppets/>



# Deep Learning for Object Recognition: Hinton & colleagues, NIPS 2012



mite

container ship

motor scooter

leopard



grille

mushroom

cherry

Madagascar cat





# AI/ML for Autonomous Vehicles



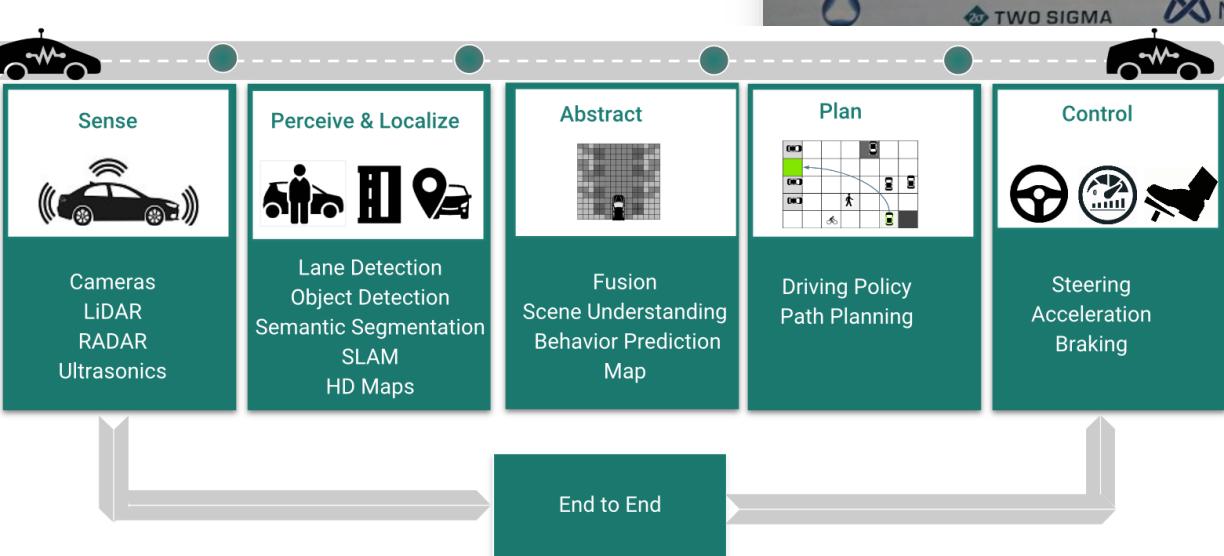
FiveAI: delivering autonomous vehicles  
to London in 2019

KEY COMPETENCIES WE'RE HIRING

Structure Form Motion (SFM) Depth and Pose Estimation  
Stereo Reconstruction Optical Flow  
Pixel-Wise Segmentation SLAM Multi-task Learning  
Recurrent Neural Networks Unsupervised Learning POMDP  
Interpretability / XAI Agent Intention

FiveAI/Careers info@five.ai @ FiveAI

Work on  
self-driving cars





# Learning from Experience Without a Teacher



Learns to play the game Go, just by playing games against itself

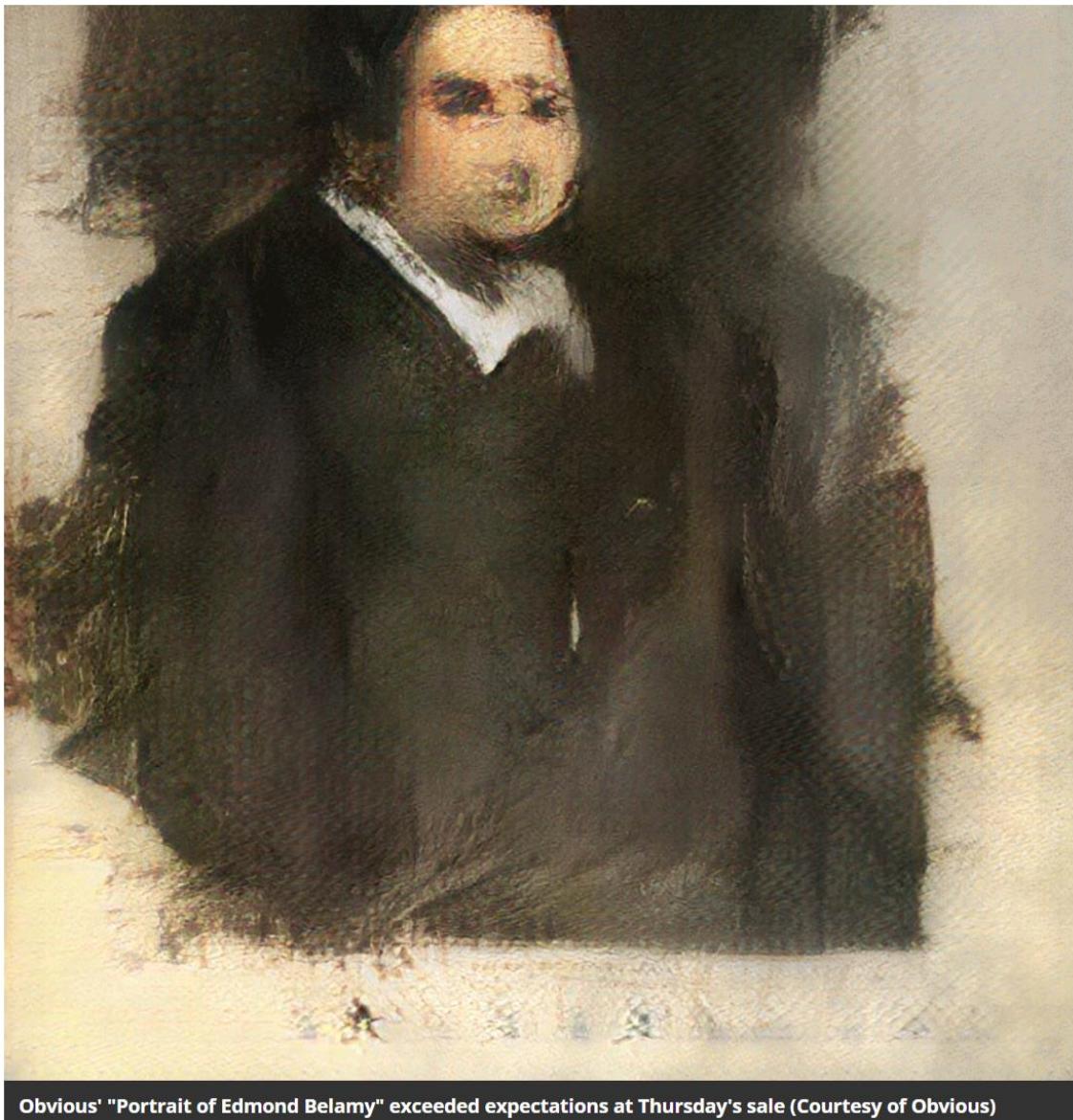
Starting from completely random play

<https://deepmind.com/blog/alphago-zero-learning-scratch/>





## Generative adversarial networks



Obvious' "Portrait of Edmond Belamy" exceeded expectations at Thursday's sale (Courtesy of Obvious)

**SMARTNEWS** *Keeping you current*

### Christie's Is First to Sell Art Made by Artificial Intelligence, But What Does That Mean?

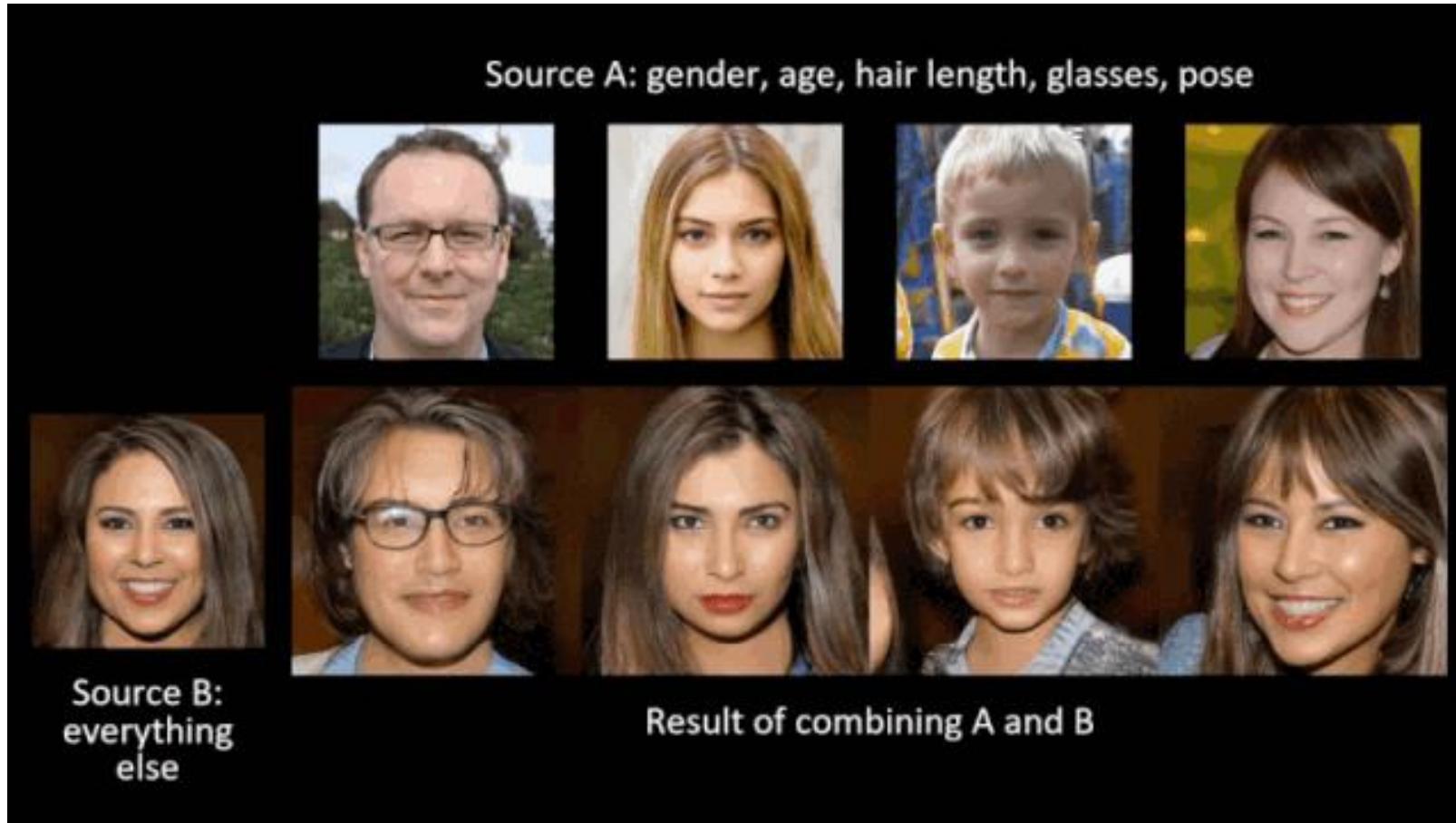
Paris-based art collective Obvious' 'Portrait of Edmond Belamy' sold for \$432,500, nearly 45 times its initial estimate

October 2018

<https://www.smithsonianmag.com/smарт-news/christies-first-sell-art-made-artificial-intelligence-what-does-mean-180970642/>



# Generative adversarial networks



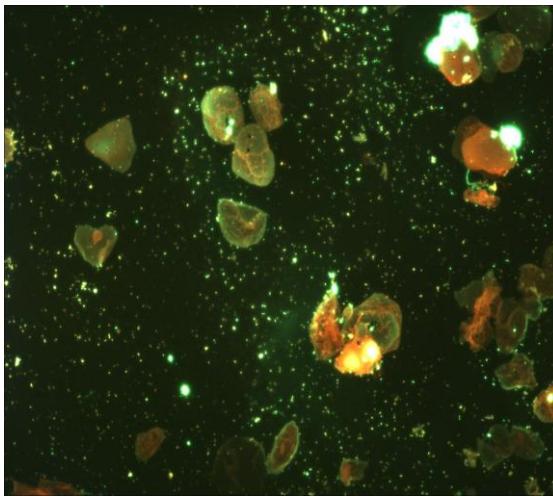
<https://medium.com/syncedreview/nvidia-open-sources-hyper-realistic-face-generator-stylegan-f346e1a73826>

<https://www.theverge.com/2019/3/19/18272602/ai-art-generation-gan-nvidia-doodle-landscapes>

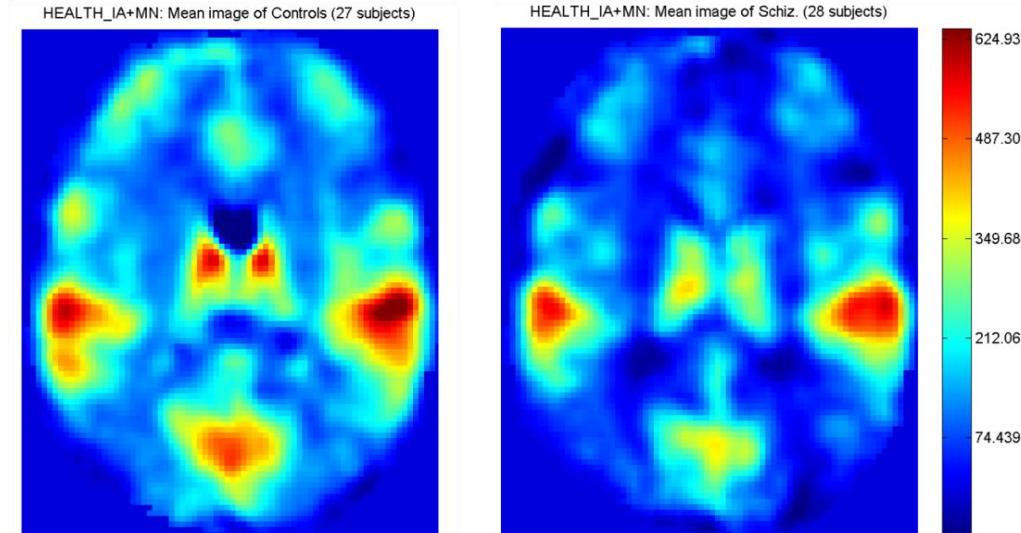


# Some local examples at NUI Galway: Image & Sensor Data Mining

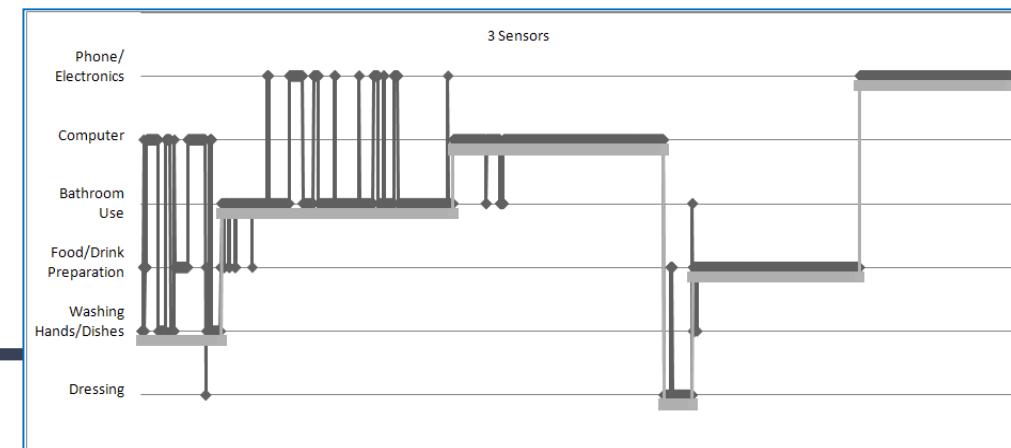
UC Irvine / NIH BIRN collaboration:  
Using fMRI to distinguish subjects  
with Schizophrenia from controls



Analysing microscope images  
of sputum to screen for TB:  
Image processing, ML,  
Sequential statistics



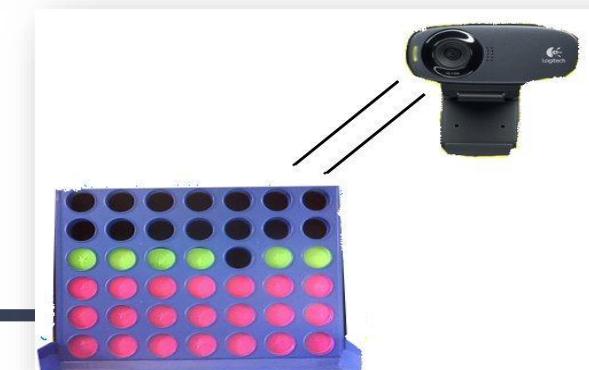
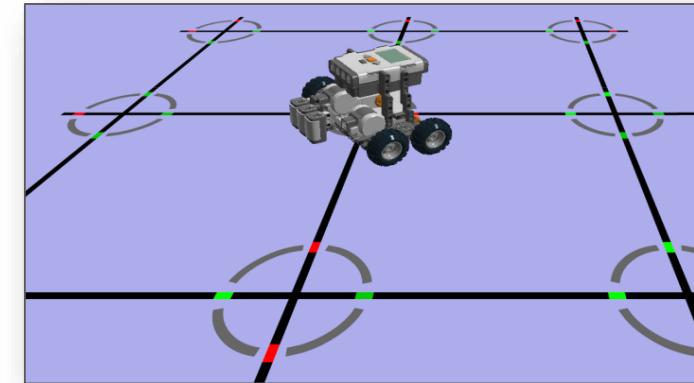
Identify Activities of Daily Living from  
sensors: Ensemble DTW Classifier





## Some local examples at NUI Galway: Reinforcement Learning

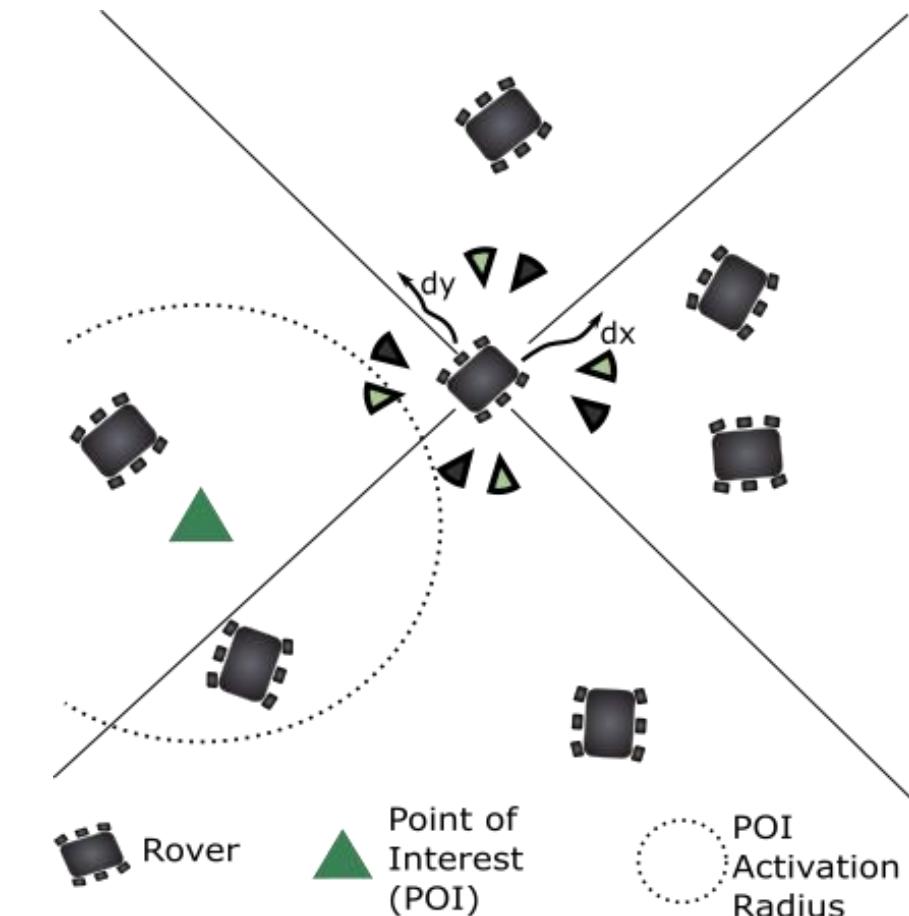
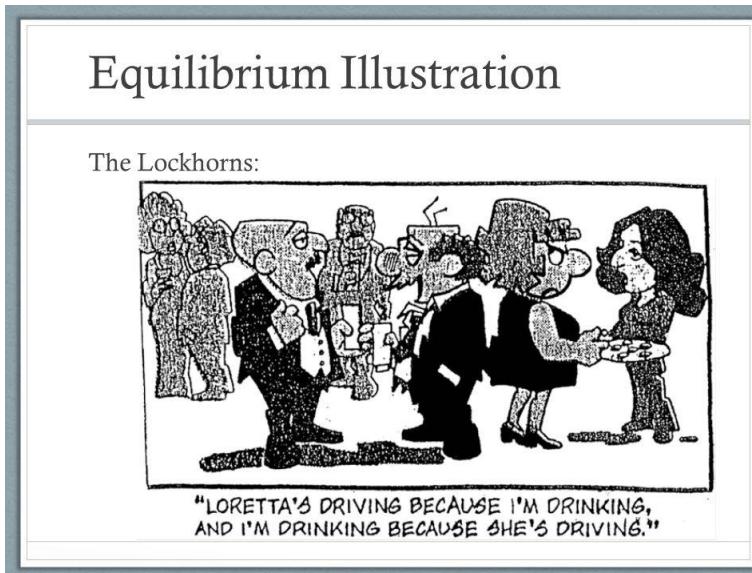
- RL Agent that learns to play UT2004 through trial & error
  - Goal: human-like performance
- Robots that learn to navigate mazes & solve puzzles





# Some local examples at NUI Galway: Multi-agent learning and decision making

- Learning equilibrium concepts such as Nash equilibria
- Multi-agent / multi-robot coordination
- Multi-objective decision making
- Applications to many different areas e.g. transportation, smart grid, conflict negotiation, auctions, etc.





# ROCSAFE: Remotely Operated CBRNE Scene Assessment & Forensic Examination

RAVs with automatic navigation and routes optimised for finding zones of interest and scene overview



Video, images, relayed to Central Decision Management.  
Command Centre with map-based GUI showing threat colour maps, etc.  
Video & maps augmented with analysed sensor results.

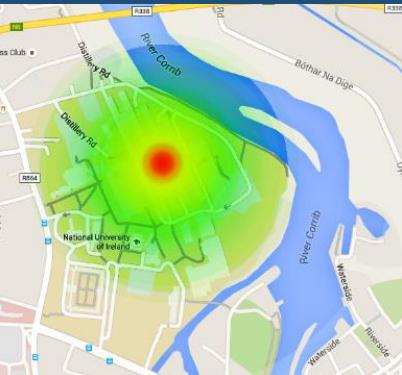


All data streamed to Central Decision Management

Forensic results, when available, transmitted to Central Decision Management



RGVs deployed directly to zones of interest. Equipped with tools for forensic evidence collection



Forensic samples delivered to mobile lab [which is outside the scope of ROCSAFE]



## The Future ...

- Algorithms for learning from mixed media
- Systems that can automatically adapt to changing circumstances (adaptive, 'self-healing')
  - Software and embedded in hardware
- Search engines capable of resolving ambiguity and synthesising results from multiple sources
  - What will the weather be like in Finland next week?
  - Wolfram|Alpha, IBM's Watson
- Cumulative learning and transfer of skills
- Active experimentation
- Databases and programming languages with built-in learning; Cloud APIs
- Sensors everywhere; small & wearable computing



## Learning Objectives: Review

If you have been paying close attention, you will now be able to ...

1. Discuss definitions of Machine Learning
2. Describe what major categories of ML task entail: classification, regression, clustering, relationship discovery and reinforcement learning
3. Discuss the relationship with Data Mining
4. Explain the Data Mining process
5. Consider current and future applications of Machine Learning and Data Mining



# Topic 2: Information-based learning

## Part 1: Introduction



# Learning objectives

After completing this topic successfully, you will be able to ...

1. Explain what supervised learning is
2. Distinguish it from unsupervised learning and reinforcement learning
3. Describe in detail an algorithm for decision tree induction
4. Apply decision tree induction to a data set
5. List related algorithms
6. Discuss high-level concepts such as choice of hypothesis language, overfitting, underfitting and noise

Reading: Russell & Norvig 3<sup>rd</sup> Ed, Chapter 18.18.4; Kelleher et al. Chapter 4



# Overview of topic

## This week:

1. Introduction, learning objectives and overview
2. Supervised learning principles
3. Decision trees
4. Entropy
5. Information gain

## Next week:

6. The ID3 algorithm
7. Issues in decision tree learning
8. ID3 extensions and related algorithms
9. Supervised learning considerations
10. Review of topic



# Topic 2: Information-based learning

## Part 2: Supervised learning principles



# 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)

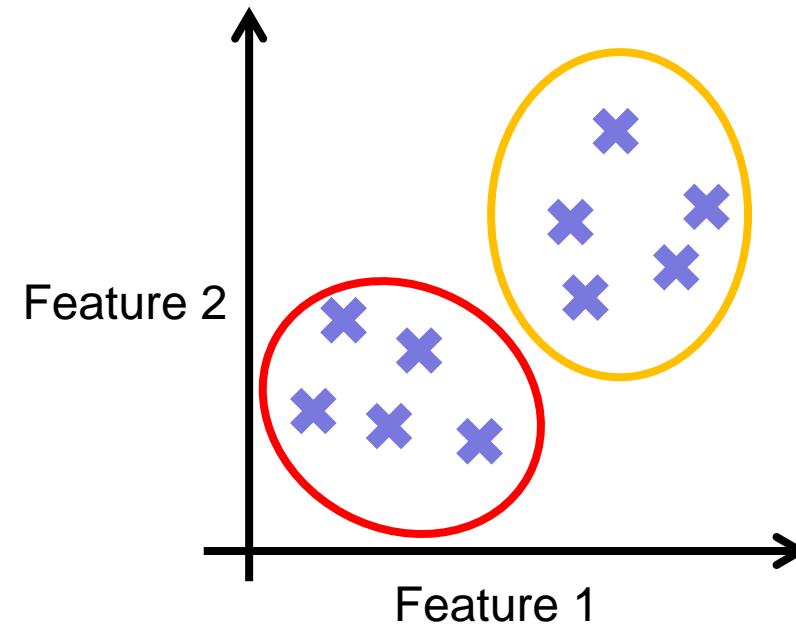
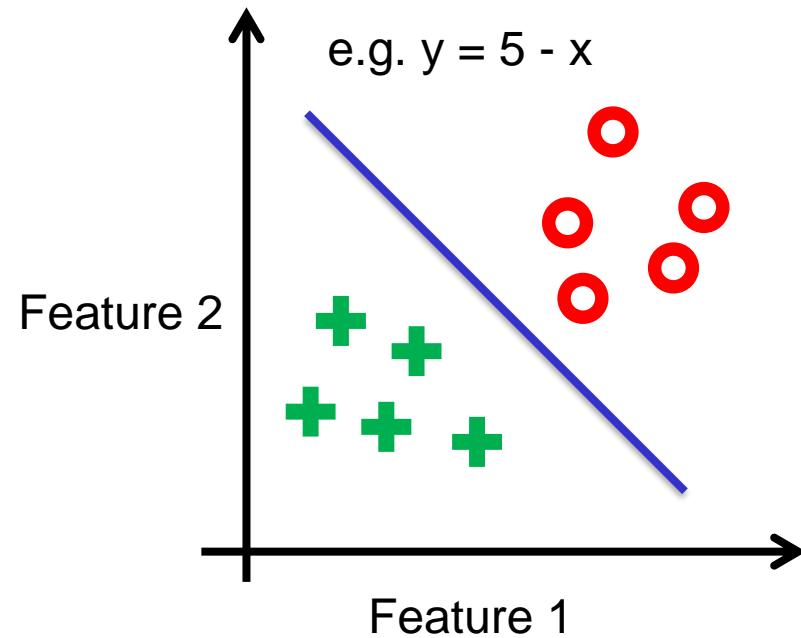
## Key feature:

given "right answers" / "ground truth" as start point.

Which tasks are classification, and which are regression?



# Supervised vs. unsupervised learning





# 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?





# Training data example

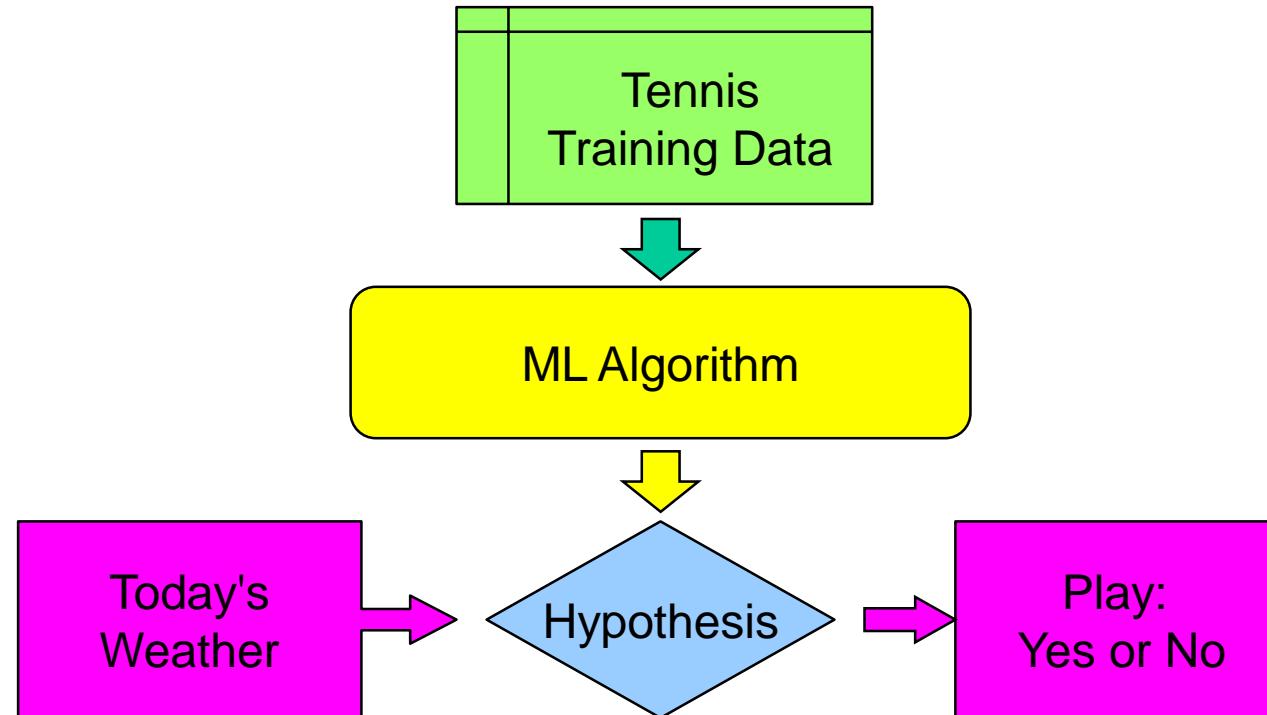
Anyone for Tennis?						Task Description
ID	Outlook	Temp	Humidity	Windy	Play?	
A	sunny	hot	high	false	no	
B	sunny	hot	high	true	no	
C	overcast	hot	high	false	yes	
D	rainy	mild	high	false	yes	
E	rainy	cool	normal	false	yes	
F	rainy	cool	normal	true	no	

Annotations pointing to specific parts of the table:

- Attributes/Dimensions**: Points to the column headers.
- One Training Case**: Points to row C.
- Identifier (not used in learning)**: Points to the ID column.
- Attribute values = Independent variables**: Points to the Outlook, Temp, Humidity, and Windy columns.
- Labels/Classes = Dependent variables**: Points to the Play? column.



# Overview of the supervised learning process





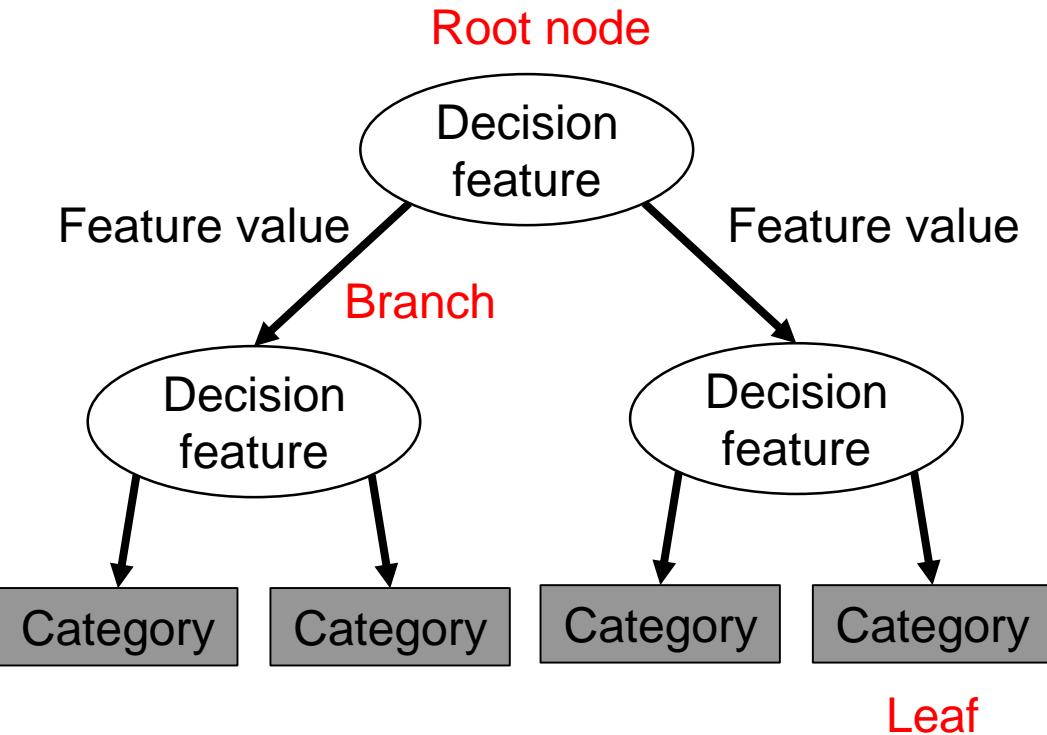
# Topic 2: Information-based learning

## Part 3: Decision trees



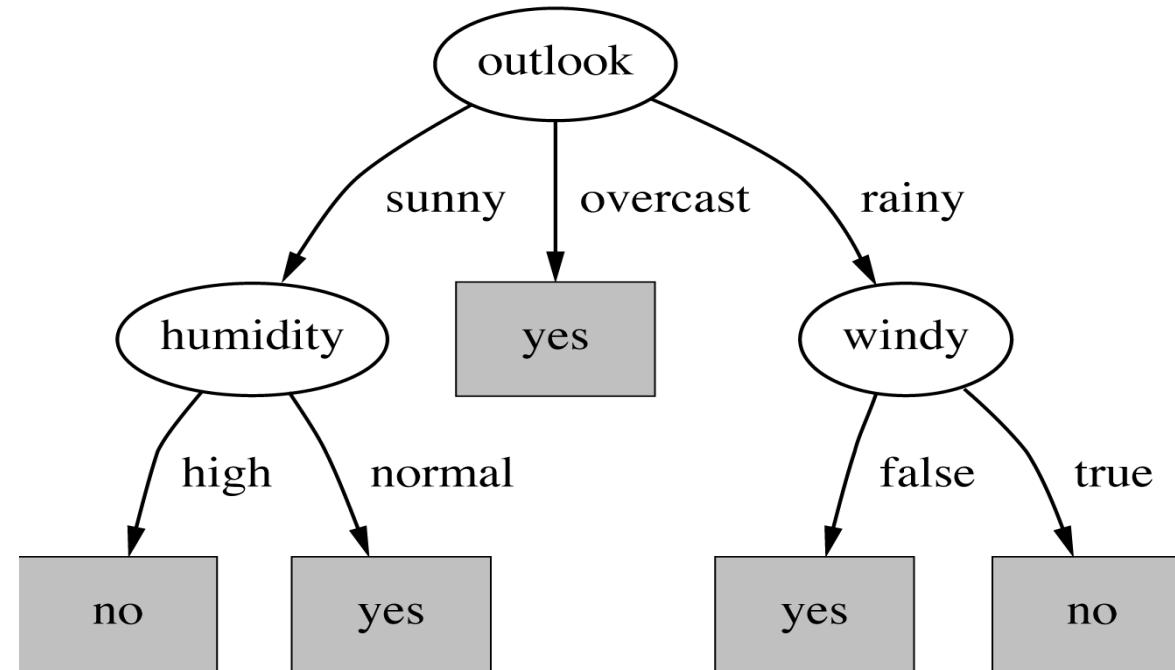
# Decision trees

- 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 **weather.xls** (spreadsheet) or **weathertxt.csv** (comma separated values format)
- 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.



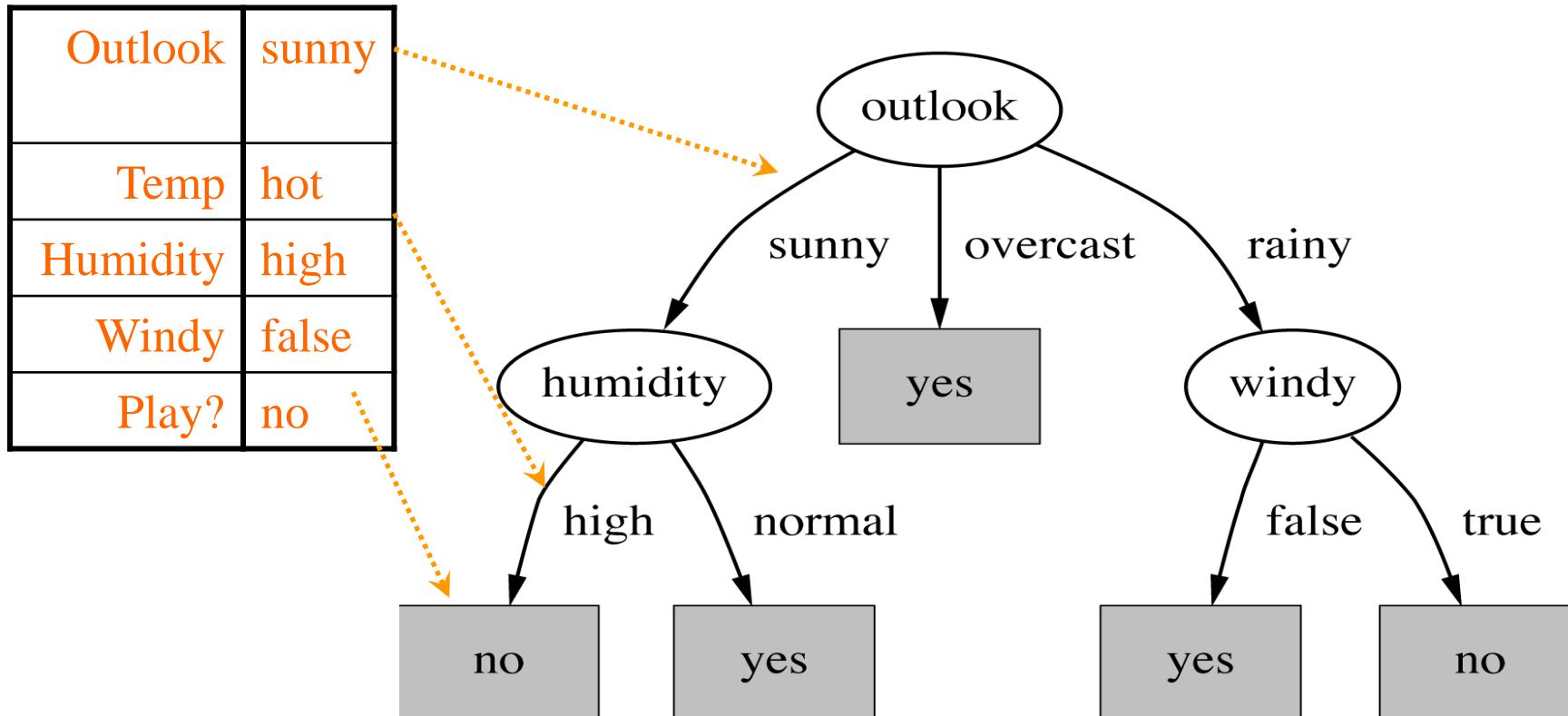
## Example dataset for induction (2)

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)

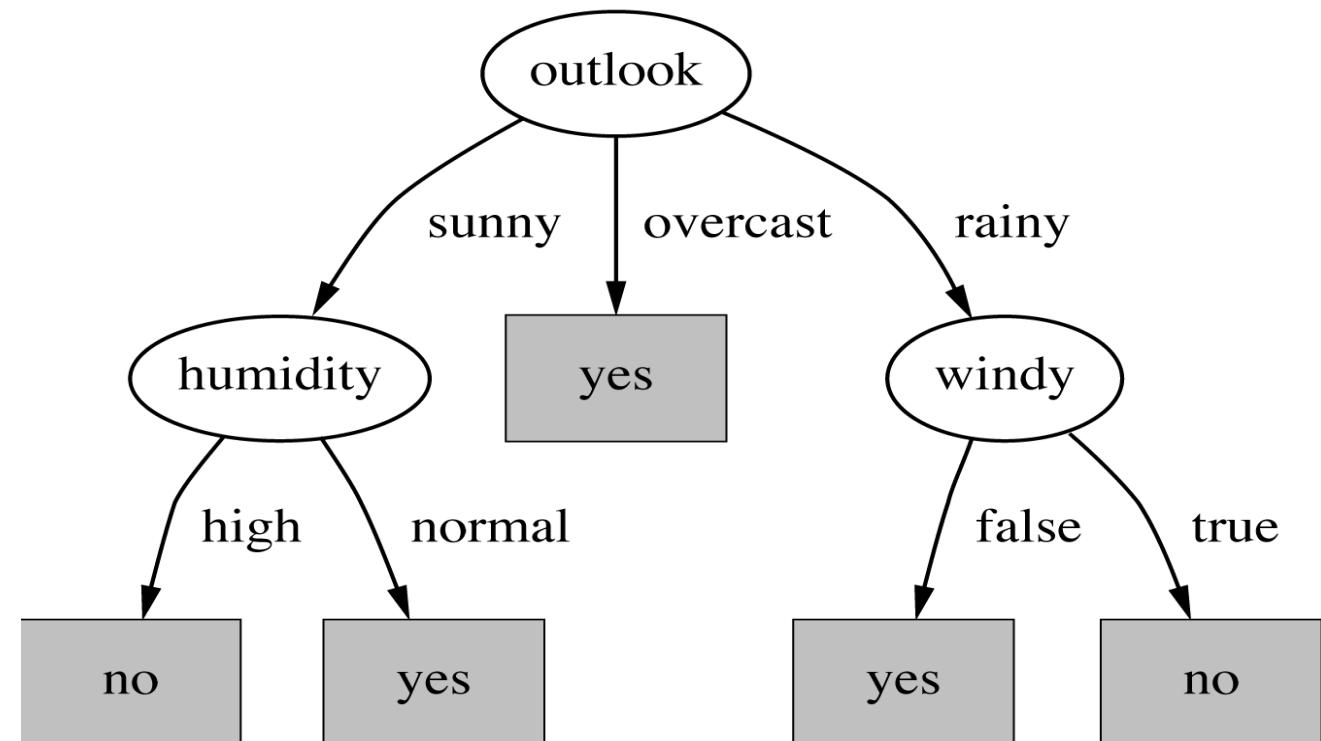




## Decision tree for this data (2)

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 **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



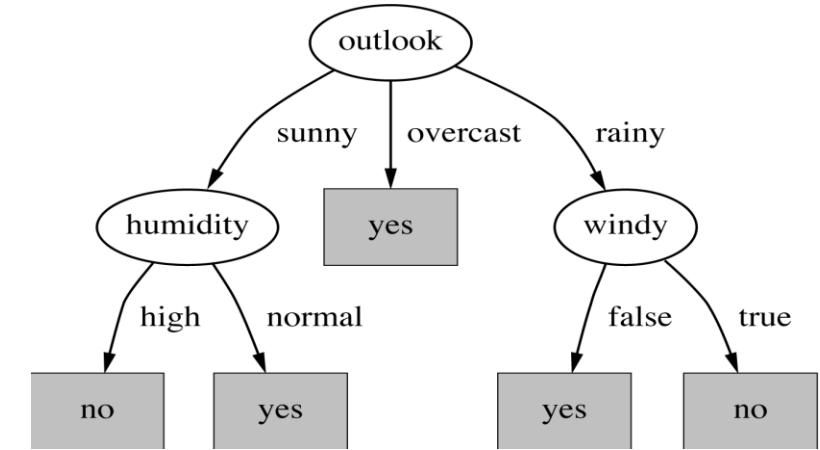
# Topic 2: Information-based learning

## Part 4: 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**

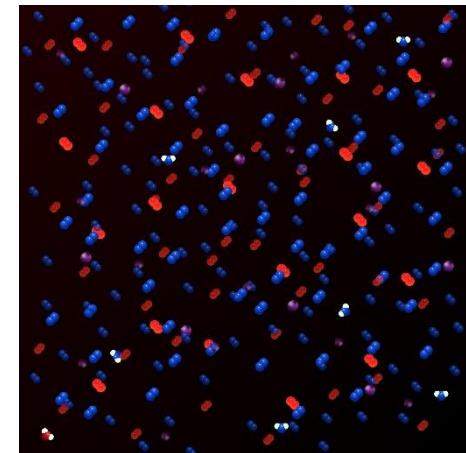


Anyone for Tennis?					
ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	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:

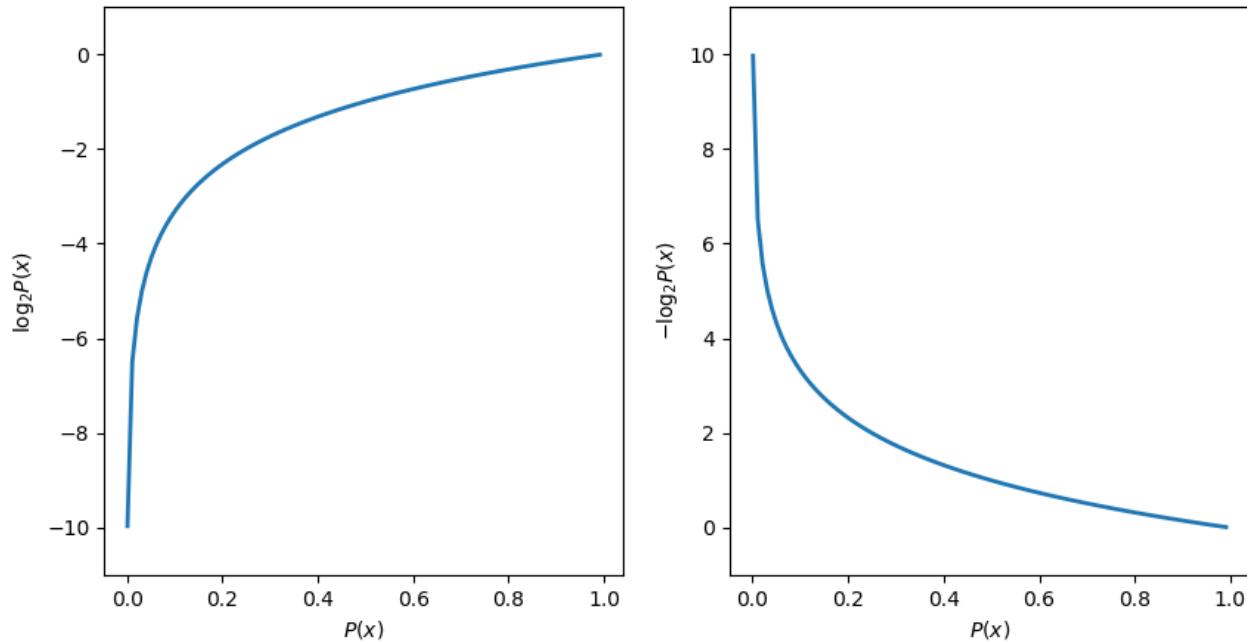
$$\text{Ent}(S) = \sum_{i=1}^n -p_i \log_2 p_i$$

- 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?

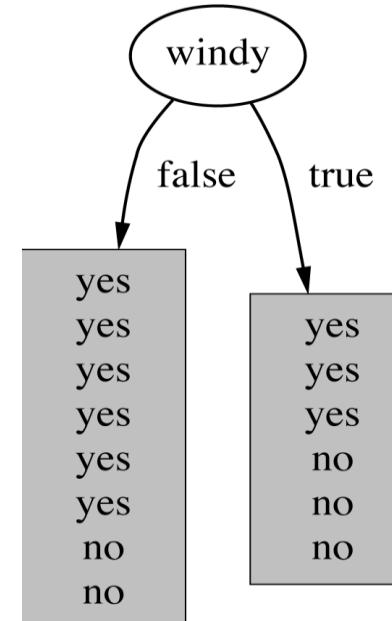
ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy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

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

$$\begin{aligned}\text{Ent}(S_{\text{windy}=\text{false}}) &= \text{Ent}([6+, 2-]) \\ &= -6/8 \log_2(6/8) - 2/8 \log_2(2/8) \\ &= 0.3112 + 0.5 = 0.8112\end{aligned}$$



$$\begin{aligned}\text{Ent}(S_{\text{windy}=\text{true}}) &= \text{Ent}([3+, 3-]) \\ &= -3/6 \log_2(3/6) - 3/6 \log_2(3/6) \\ &= 0.5 + 0.5 = 1.0\end{aligned}$$

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 2: Information-based learning

## Part 5: 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

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

$$|S|=14$$

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

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

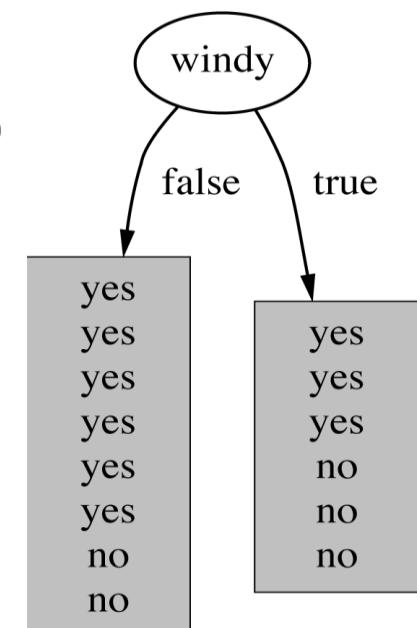
Gain(S, Windy)

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

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

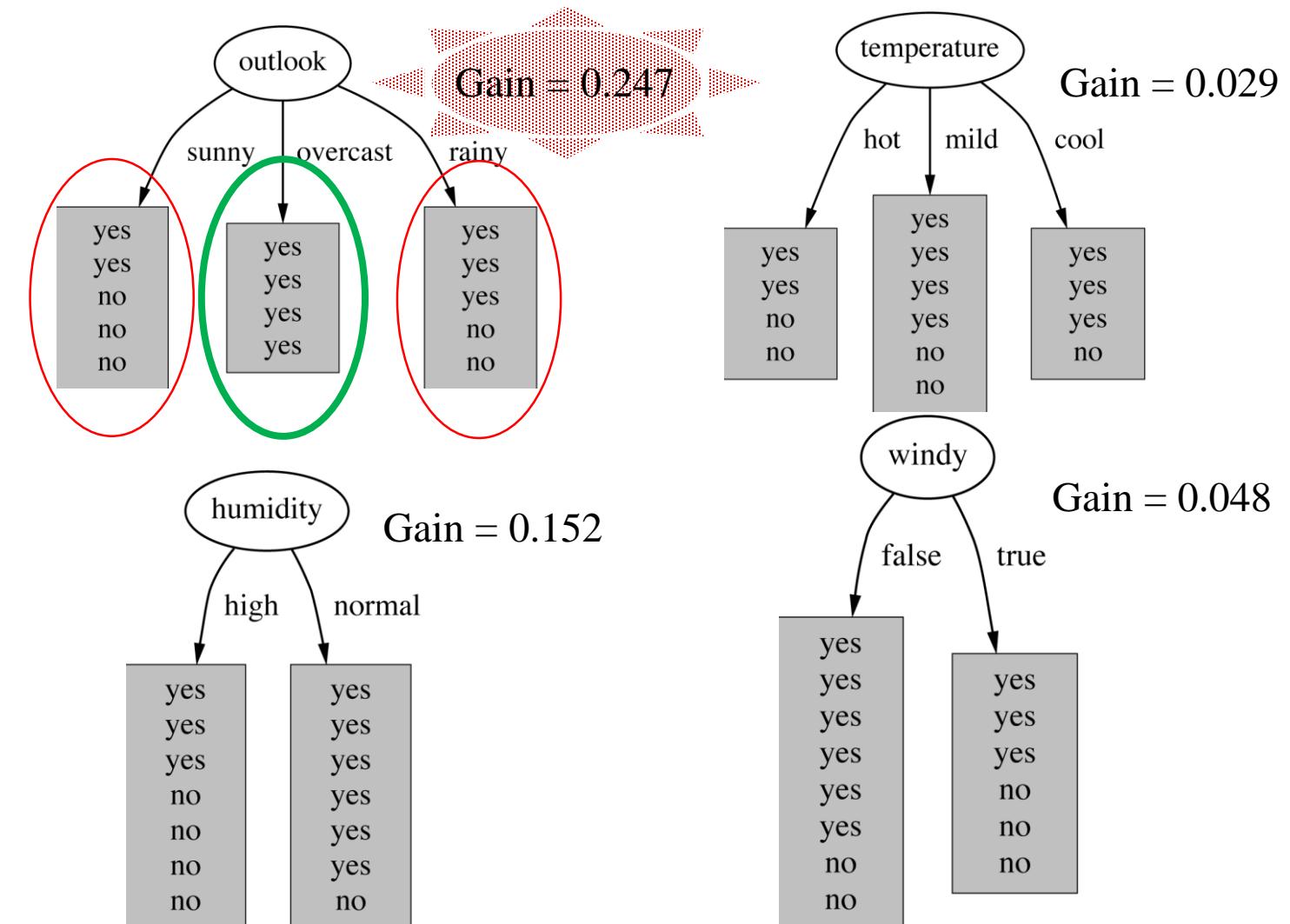
$$= 0.940 - (6/14) 1.00 - (8/14) 0.811$$

Gain(S, Windy) = **0.048**





# Best partitioning = highest information gain



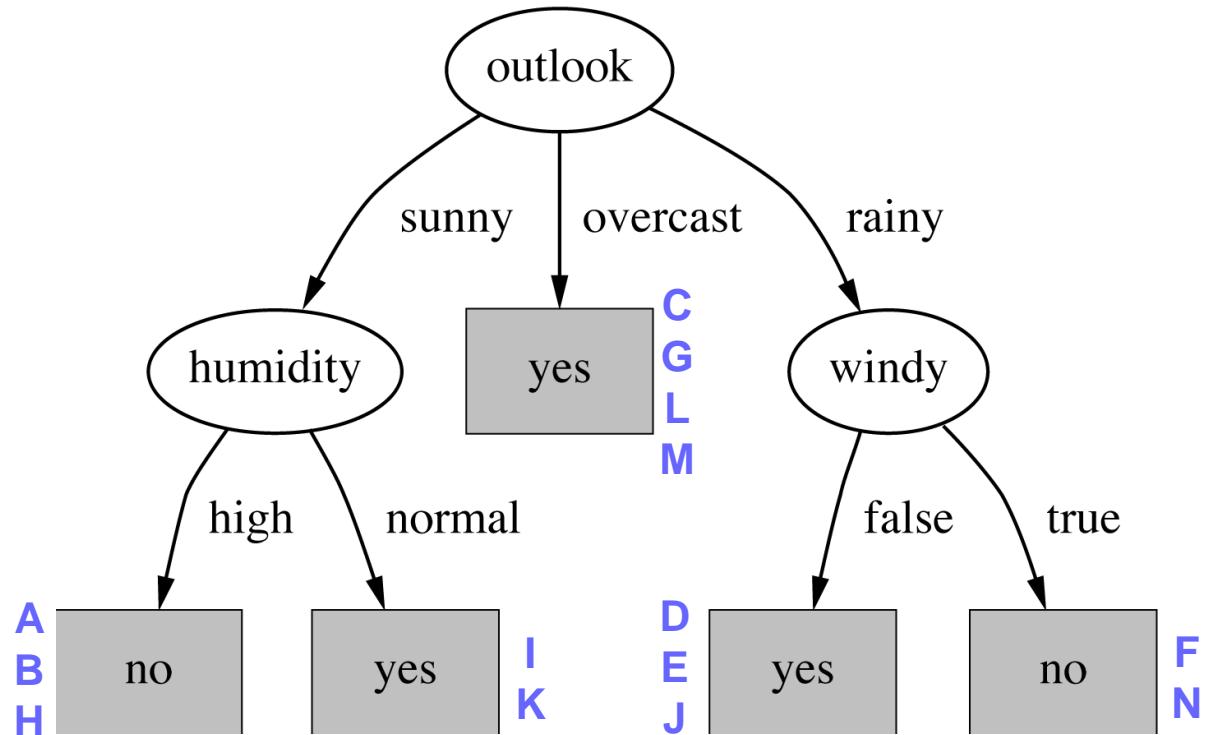
Anyone for Tennis?					
ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	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}?

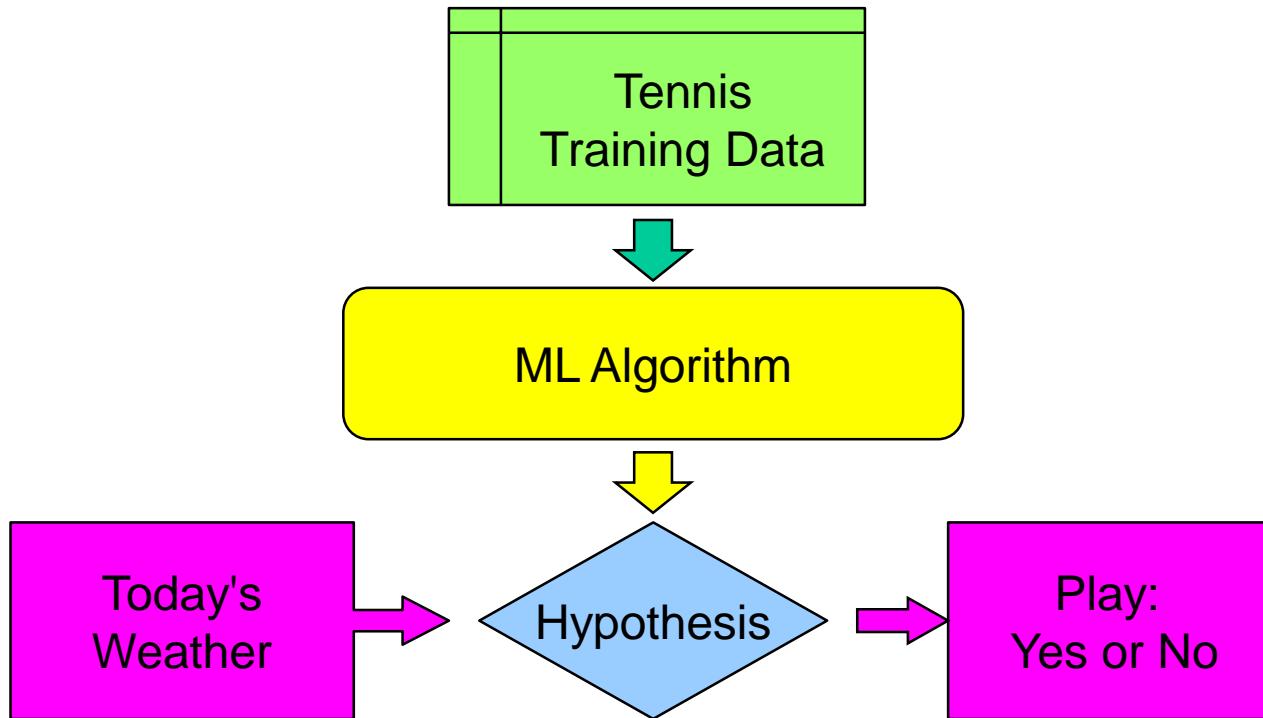


# Topic 2: Information-based learning

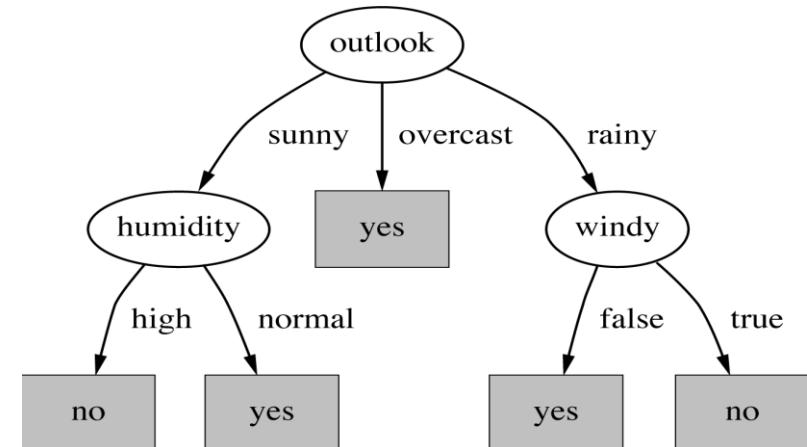
## Part 6: The ID3 algorithm



# 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

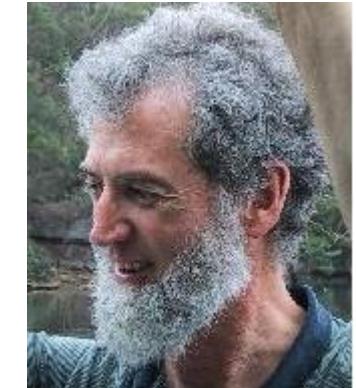
- This node **partitions** the data:  
apply the algorithm **recursively** to each partition



# 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 vi of Best:  
12.    let Examplesi = subset of Examples that have Best=vi      RECURSIVE  
13.    let Subtree = ID3(Examplesi, Attributes-Best, Target) ← CALL  
14.    add branch from Tree to Subtree with label vi  
15.  return Tree
```

Ross Quinlan, 1986



} BASE CASES

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



# Topic 2: Information-based learning

## Part 7: Issues in 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

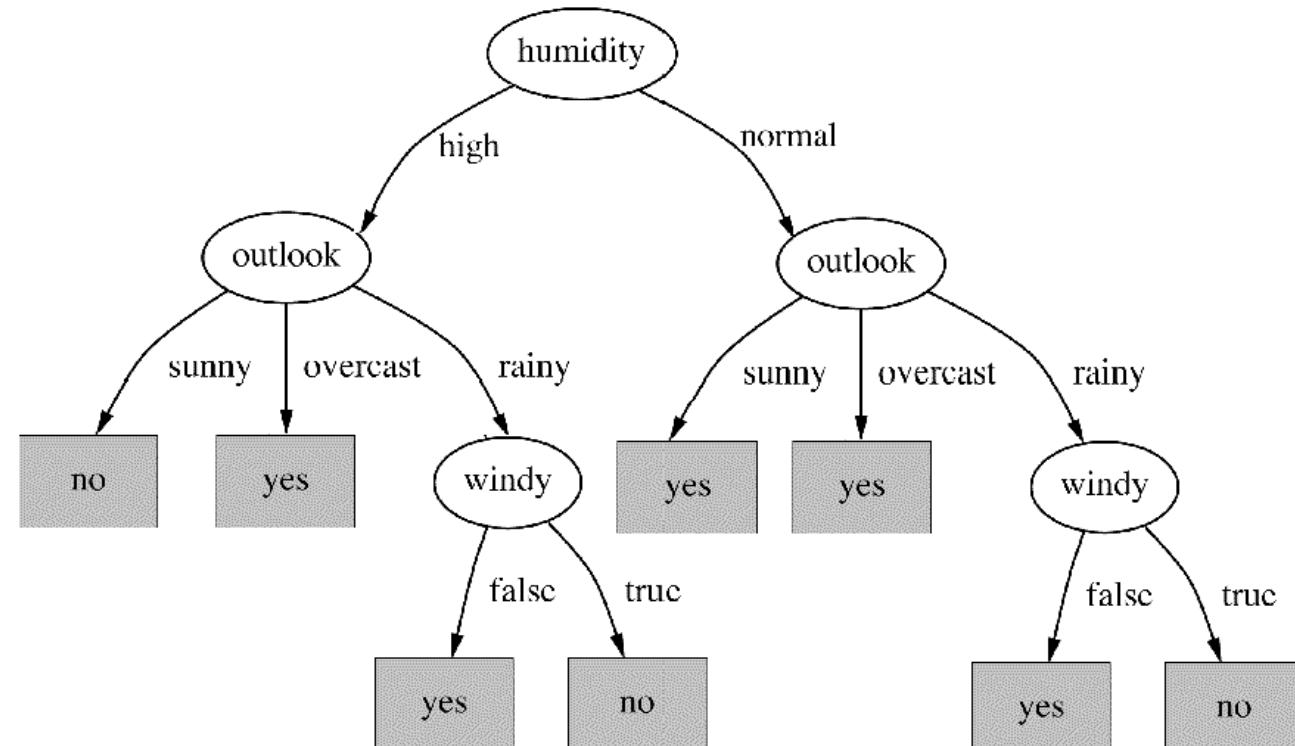
- What about inconsistent ("noisy") data?
  - Use majority class (line 7 of ID3 alg.)  
7. else if all **Attributes** are tested then return majority class
  - or interpret as probabilities
  - or return “average” target feature value
- What about missing data?
  - Given a complete decision tree, how 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?
E	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.  $\leq$  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  $\chi^2$ )
  - 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  $\chi^2$ , reduced error pruning (Quinlan, 1987) )



# Topic 2: Information-based learning

## Part 7: ID3 extensions and related algorithms



# Continuous-valued attributes

- What about continuous-valued attributes?
  - Pick threshold value  $T$  for attribute A, and test whether  $A > T$
  - Information Gain can be used to decide which  $T$  is best
  - Could select  $T$  at a **midpoint** where classification changes

Temp	40	48	54	60	72	80	85	90
Play?	No	No		Yes	Yes	Yes		No

The table illustrates a dataset for a continuous attribute 'Temp' (Temperature) and a binary classification attribute 'Play?'. The 'Temp' column shows values 40, 48, 54, 60, 72, 80, 85, and 90. The 'Play?' column shows 'No' for temperatures 40, 48, 72, 80, and 90, and 'Yes' for 60 and 85. Two red vertical lines are drawn at Temp = 54 and Temp = 85, marking these as potential threshold values where the classification status changes from 'No' to 'Yes'.



## Selecting the best attribute: alternative metrics (1)

- Earlier, we introduced the concept of information gain, which we can use as a metric for the discriminatory power of an attribute
- Information gain does have some drawbacks; it tends to favour attributes that can take on a large number of different values
- One alternative is to use the **information gain ratio**

$$\text{GainRatio}(S, A) = \frac{\text{Gain}(S, A)}{\sum_{v \in \text{Values}(A)} -p_v \log_2 p_v}$$

- The divisor of this fraction measures the amount of information used to compute the gain value, and is the entropy of S with respect to A



## Selecting the best attribute: alternative metrics (2)

- Another alternative is to use the **Gini index** instead of entropy as a measure of the impurity of a set

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

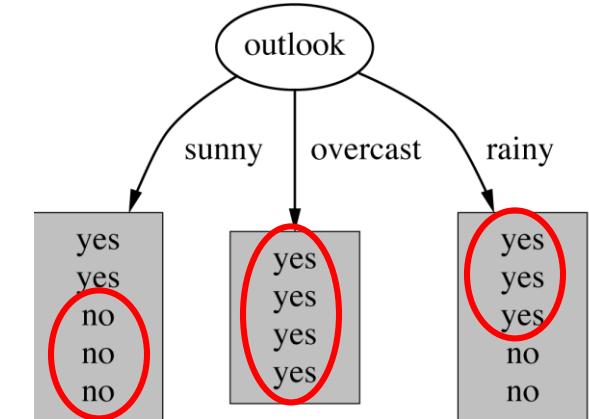
- Then the gain for a feature may be calculated based on the reduction in the Gini index (rather than a reduction in entropy):

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



# Related algorithms [1]

- 1R
  - Decision tree with just one rule
  - Introduced in a paper by Robert C. Holte (1993).  
*“Very Simple Classification Rules Perform Well on Most Commonly Used Datasets”*, Computer Science Department, University of Ottawa



- Decision Stump
  - 1 rule with 1 test

Outlook = Overcast => YES

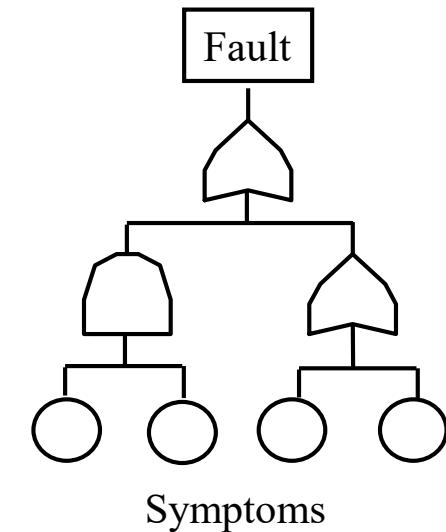
Outlook != Overcast => YES

*These are deliberately simple variants that are used within other algorithms (meta-learning; ensembles). Often referred to as “weak learners”.*



## Related algorithms [2]

- Decision Lists
  - A set of rules (predicate logic), describing the hypothesis, that are followed in the given order
- C4.5 Rules
  - Alternative representation of C4.5 decision trees
- PART: Rules constructed with *partial* DTs
  - Can be more readable than standard DTs
- IFT: Induction of Fault Trees





# Decision tree software

- C4.5
  - Original implementation (C language, command line), available on Ross Quinlan's website (<https://www.rulequest.com/Personal/>)
  - Deals with missing values in the data, high-branching attributes (e.g. ID in the weather data), pruning to avoid overfitting, converting a decision tree to a list of rules
- C5.0
  - Commercial version from RuleQuest Research, with improvements over C4.5
- WEKA software
  - Accompanies book by Witten & Frank, Data Mining: Practical Machine Learning Tools and Techniques
  - Java implementations of many ML algorithms, including C4.5 ([mysteriously called J48](#))
  - Easy-to-use front end and utilities
- Many other implementations in Python and R...



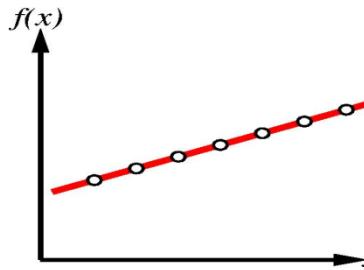
# Topic 2: Information-based learning

## Part 8: Supervised learning considerations

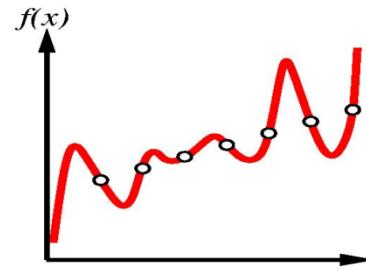


# Supervised Learning Considerations [1]

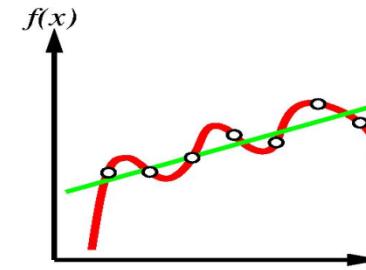
- Various hypotheses can be consistent with observations but inconsistent with each other:  
Which one should we choose?



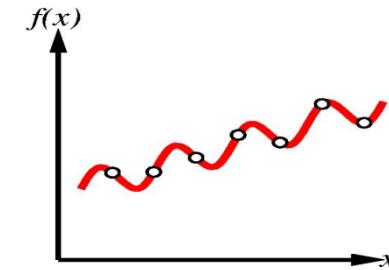
(1) Data with exact linear hypothesis



(2) Same data:  
Exact 7<sup>th</sup>-Order  
polynomial hyp.



(3) Different data:  
Exact 5<sup>th</sup>-order poly  
and approx. linear hyp.

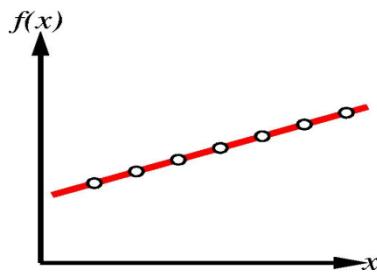


(4) Same data:  
Exact sinusoidal  
hypothesis

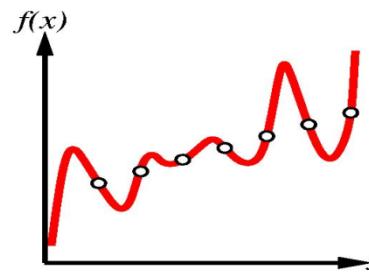


# Supervised Learning Considerations [2]

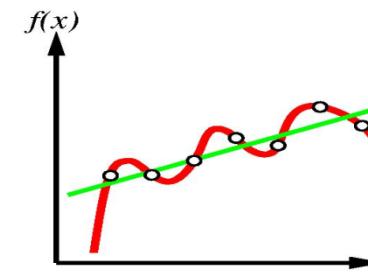
- Various hypotheses can be consistent with observations but inconsistent with each other:  
Which one should we choose?



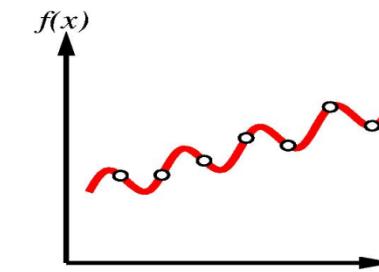
(1) Data with exact linear hypothesis



(2) Same data:  
Exact 7<sup>th</sup>-Order  
polynomial hyp.



(3) Different data:  
Exact 5<sup>th</sup>-order poly  
and approx. linear hyp.



(4) Same data:  
Exact sinusoidal  
hypothesis

- One solution: Ockham's Razor:
  - Prefer *simplest* hypothesis consistent with data
  - Definitions of simplicity (& consistency) may subject to debate
  - Depends strongly on how hypotheses are expressed



## Supervised Learning Considerations [3]

- Hypothesis language is too limited?
  - Might be **unable** to find hypothesis that exactly matches 'true' function
  - If true function is more complex than what hypothesis can express, it will **underfit** the data
  - Saw this in previous slide, 3<sup>rd</sup> and 4<sup>th</sup> figures
- Hypothesis language cannot exactly match true function?
  - there will be a trade-off between **complexity** of hypothesis and how well it **fits the data**



# Supervised Learning Considerations [4]

- Hypothesis language is very **expressive**?
  - Its search space is very large and the **computational complexity** of finding a good hypothesis will be high
  - Also need a large amount of data to avoid **overfitting**
- What can decision trees express?
  - Will learn about other algorithms that express hypotheses differently
  - In general, would like to use an algorithm for a problem that can express the true underlying function



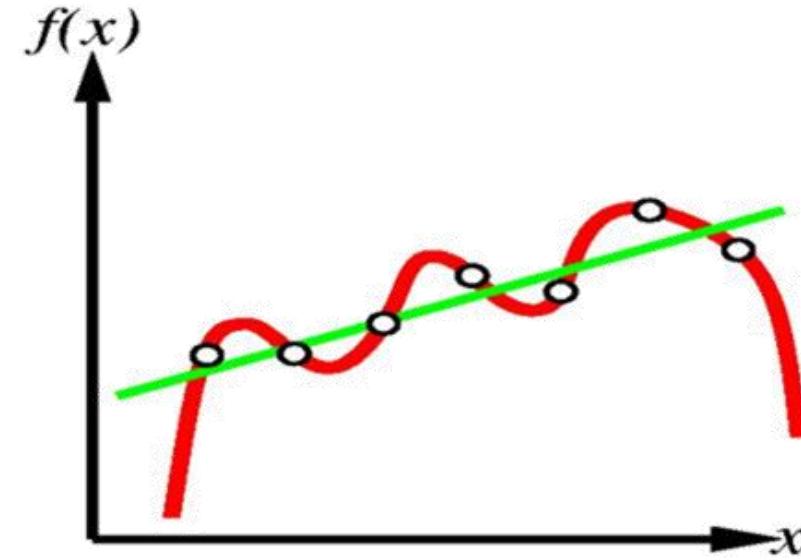
# Supervised Learning Considerations [5]

- But don't forget:  
*we never know the true underlying function*
- E.g. To avoid problem with poorly fitting data from a previous slide
  - Could change algorithm so that, as well as searching for coefficients of polynomials, it tries combinations of trig. functions (sin, cos, tan)
  - Learning problem will become enormously more complex, **but will it solve our problems?**
  - Probably not: you could easily think up some different kind of mathematical function, to generate a new dataset that the algorithm still cannot represent perfectly.
- For this reason, often use relatively simple hypothesis languages, in the absence of special knowledge about domain
  - more complex languages don't come with any real guarantees
  - more simple languages correspond to easier searching.



# Noise, Overfitting and Underfitting [1]

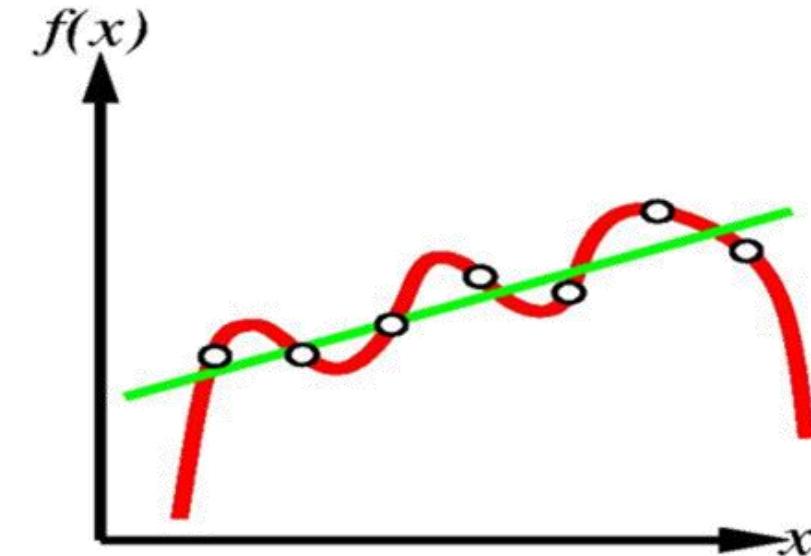
- NOISE:  
imprecise or incorrect attribute values or labels
  - Can't always quantify it, but should know from situation if it is present
  - E.g. labels may require subjective judgement or values may come from imprecise measurements





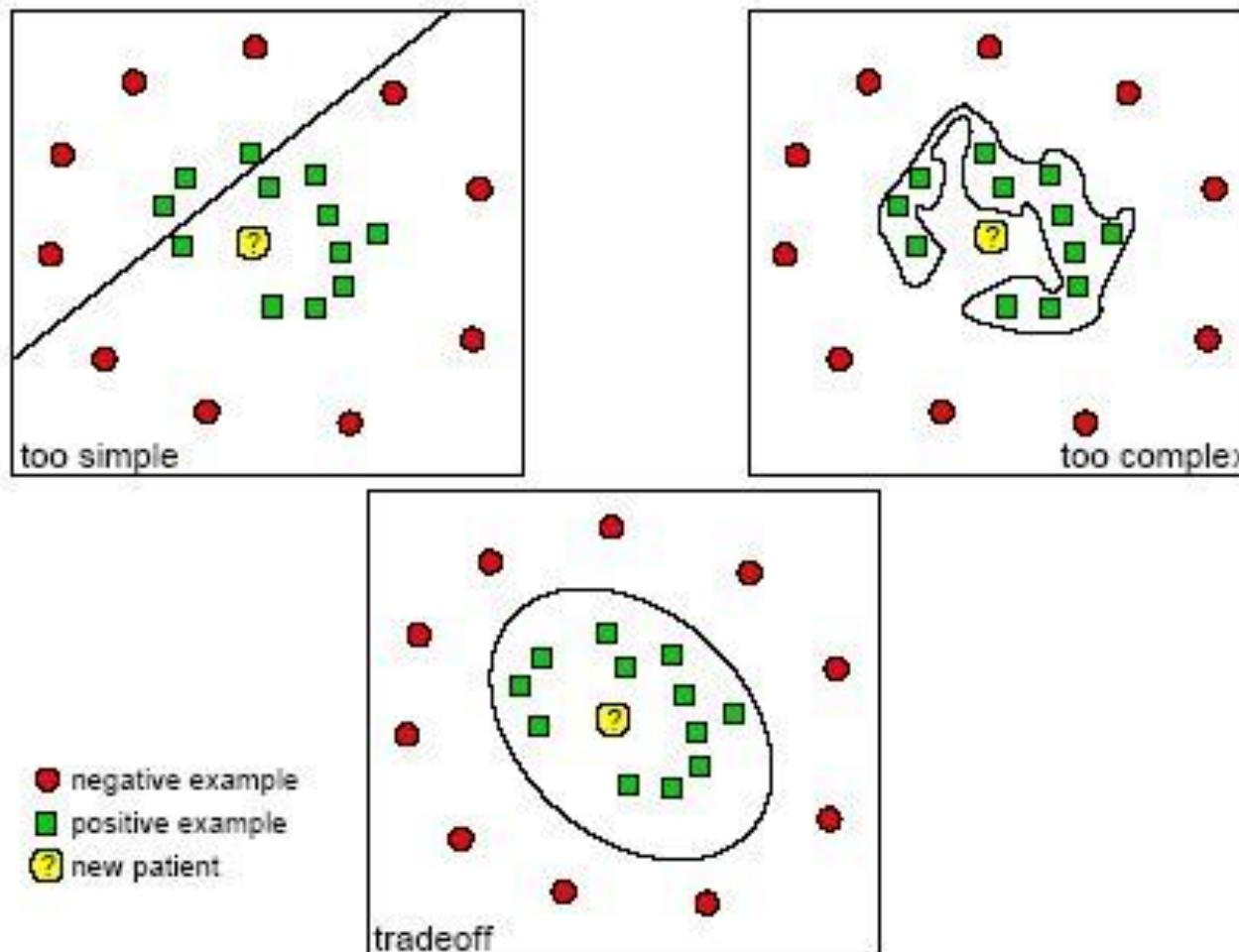
## Noise, Overfitting and Underfitting [2]

- If the data might have noise, harder to decide which hypothesis is best:
  - Linear hypothesis could not fit to it, but polynomial could
  - But which would really be the better choice?
- Complex classification methods prone to **overfitting**; simple ones prone to **underfitting**
  - If you increase complexity of hypothesis, you increase ability to fit to the data, but might also increase risk of overfitting





# Illustration of Underfitting & Overfitting





# 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 2: Information-based learning

## Part 10: Review of topic



# Learning Objectives Review

After completing this topic successfully, you will be able to ...

1. Explain what supervised learning is
2. Distinguish it from unsupervised learning and reinforcement learning
3. Describe in detail an algorithm for decision tree induction
4. Demonstrate the application of decision tree induction to a data set
5. List related algorithms
6. Discuss high-level concepts such as choice of hypothesis language, overfitting, underfitting and noise



# Similarity-based learning

## Part 1: Introduction



# Learning Objectives

After completing this successfully, you will be able to ...

- Explain what instance-based learning is
- Distinguish between *lazy* and *eager* learning
- Describe operation of k-Nearest Neighbours for classification and regression
- Discuss implications of the *curse of dimensionality*
- Discuss implications of selecting different distance metrics
- Identify suitable applications for kNN and explain how it could be applied



# Overview of topic

## This week:

1. Learning objectives and overview
2. Problem description
3. Distance-based similarity
4. The nearest neighbour algorithm
5. The k nearest neighbours algorithm

## Next week:

6. Alternate similarity measures
7. Predicting continuous targets
8. Feature selection
9. Similarity-based learning considerations
10. Review of topic



# Similarity-based learning

## Part 2: Problem description



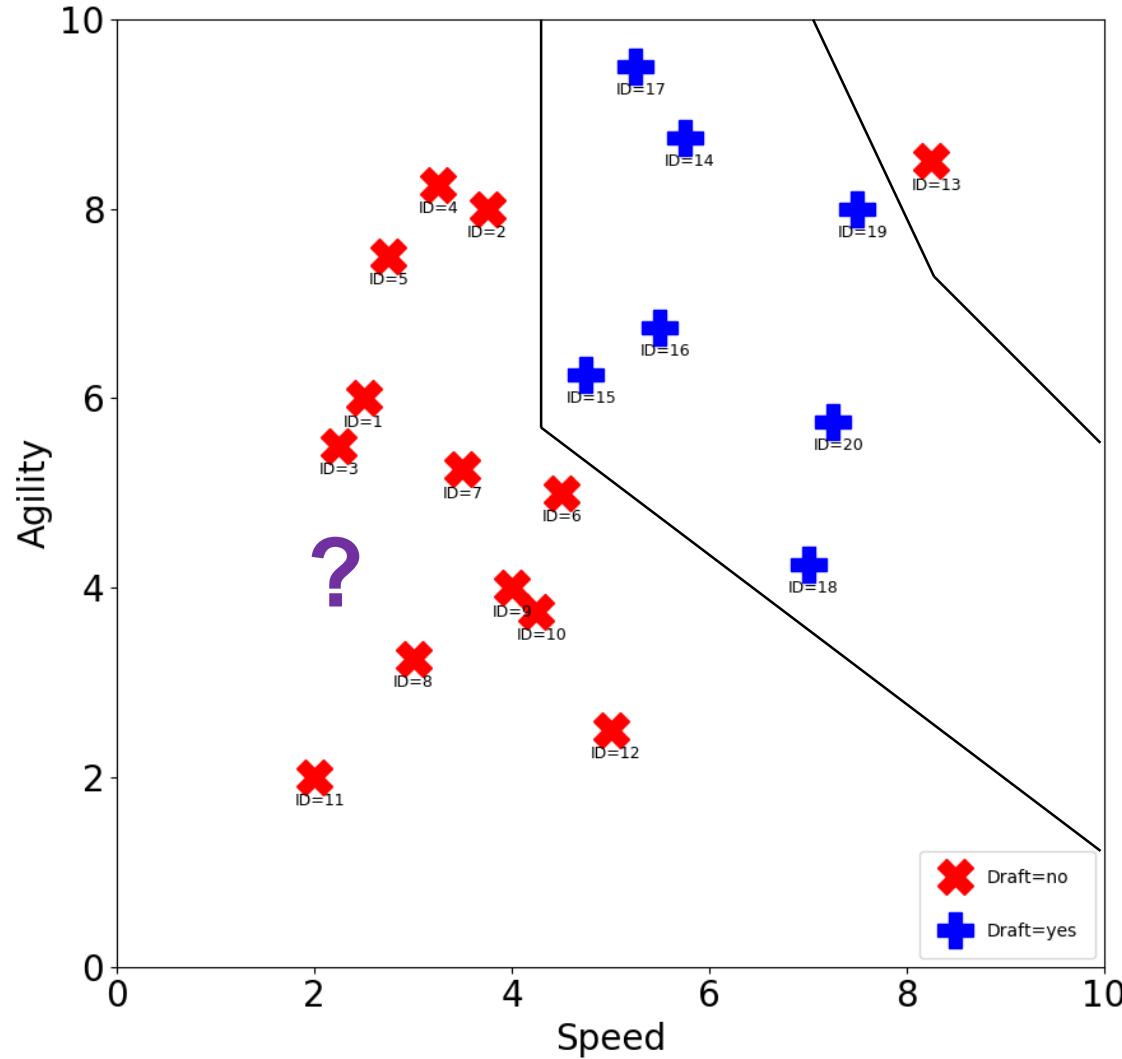
# Example dataset for similarity-based learning

- 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.xlsx** or **college\_athletes.csv**
- Objective:
  - Apply similarity-based learning methods to predict whether an 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



# Similarity-based learning

## Part 3: Distance-based 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.
- $\text{metric}(\mathbf{a}, \mathbf{b})$  is a function that returns the distance between two instances  $\mathbf{a}$  and  $\mathbf{b}$  in a set
- $\mathbf{a}$  and  $\mathbf{b}$  are vectors containing the values of the attributes we are interested in for the data points we wish to measure between



## Properties of a metric

The function  $\text{metric}(a, b)$  should satisfy the following four conditions:

1. **Non-negativity:**  $\text{metric}(a, b) \geq 0$
2. **Identity:**  $\text{metric}(a, b) = 0 \Leftrightarrow a = b$
3. **Symmetry:**  $\text{metric}(a, b) = \text{metric}(b, a)$
4. **Triangular Inequality:**  $\text{metric}(a, b) \leq \text{metric}(a, c) + \text{metric}(b, c)$



# Euclidean distance

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

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

- Here  $m$  is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors  $\mathbf{a}$  and  $\mathbf{b}$ )
- 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

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

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



## Example: calculating distance

Calculate distance between

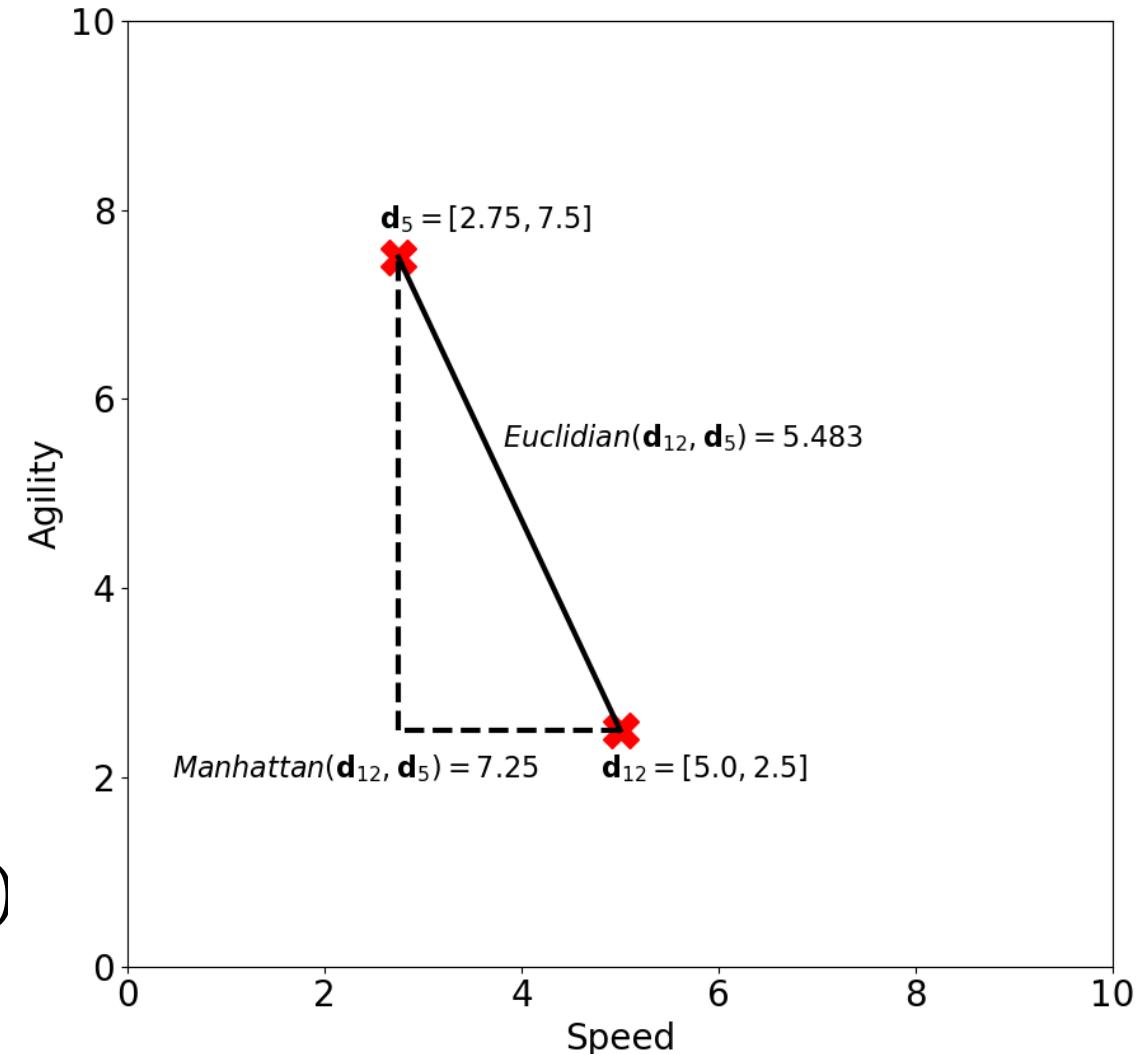
$$\mathbf{d}_{12} = [5.00, 2.50] \text{ and } \mathbf{d}_5 = [2.75, 7.50]$$

*Euclidean*( $\mathbf{d}_{12}, \mathbf{d}_5$ )

$$\begin{aligned} &= \sqrt{(5.00 - 2.75)^2 + (2.50 - 7.50)^2} \\ &= 5.483 \end{aligned}$$

*Manhattan*( $\mathbf{d}_{12}, \mathbf{d}_5$ )

$$\begin{aligned} &= \text{abs}(5.00 - 2.75) + \text{abs}(2.50 - 7.50) \\ &= 7.25 \end{aligned}$$





# Minkowski distance

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

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

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



## 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



# Similarity-based learning

## Part 4: The Nearest Neighbour algorithm



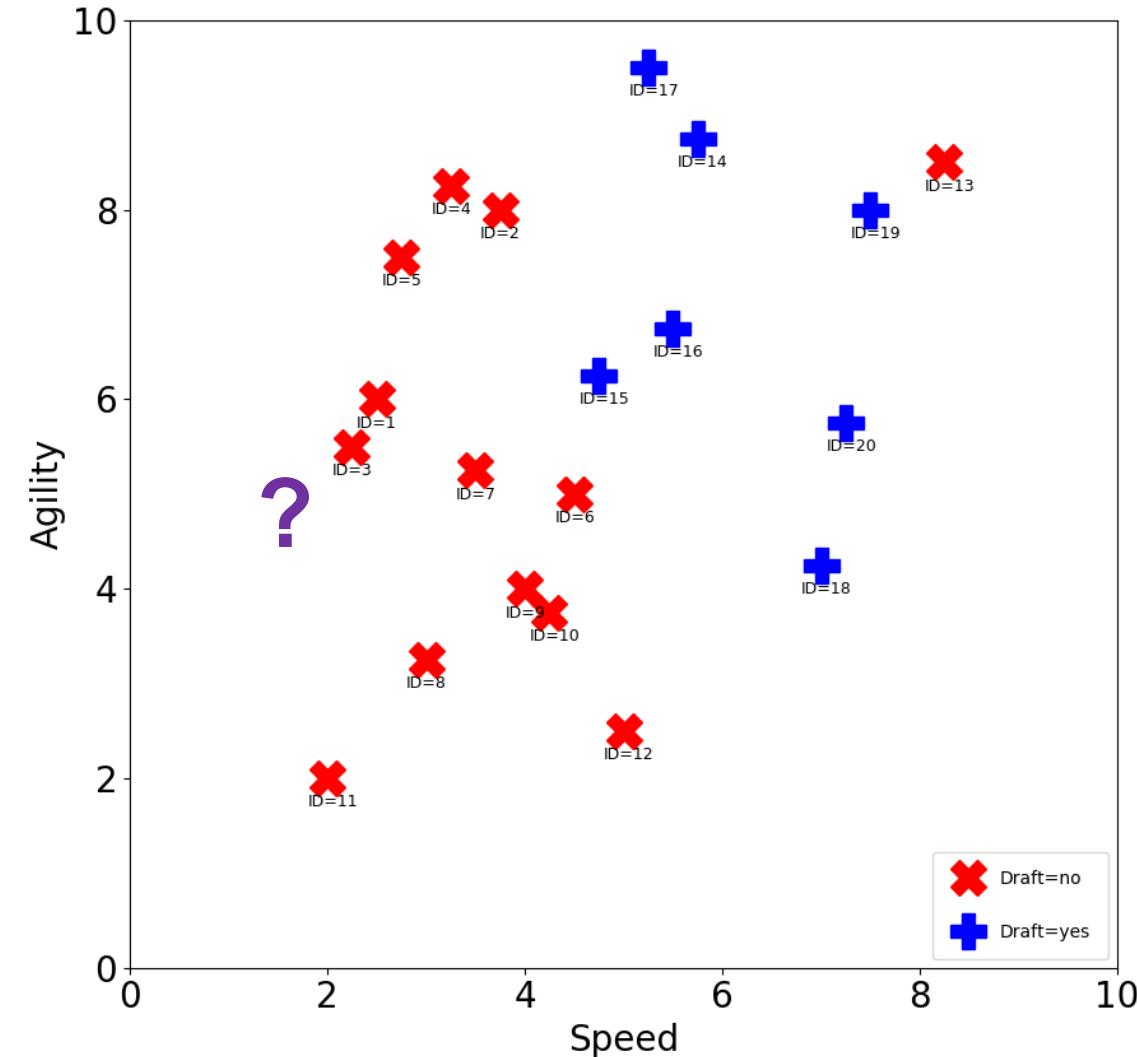
# The Nearest Neighbour algorithm

- 1-Nearst 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 (more on this later)



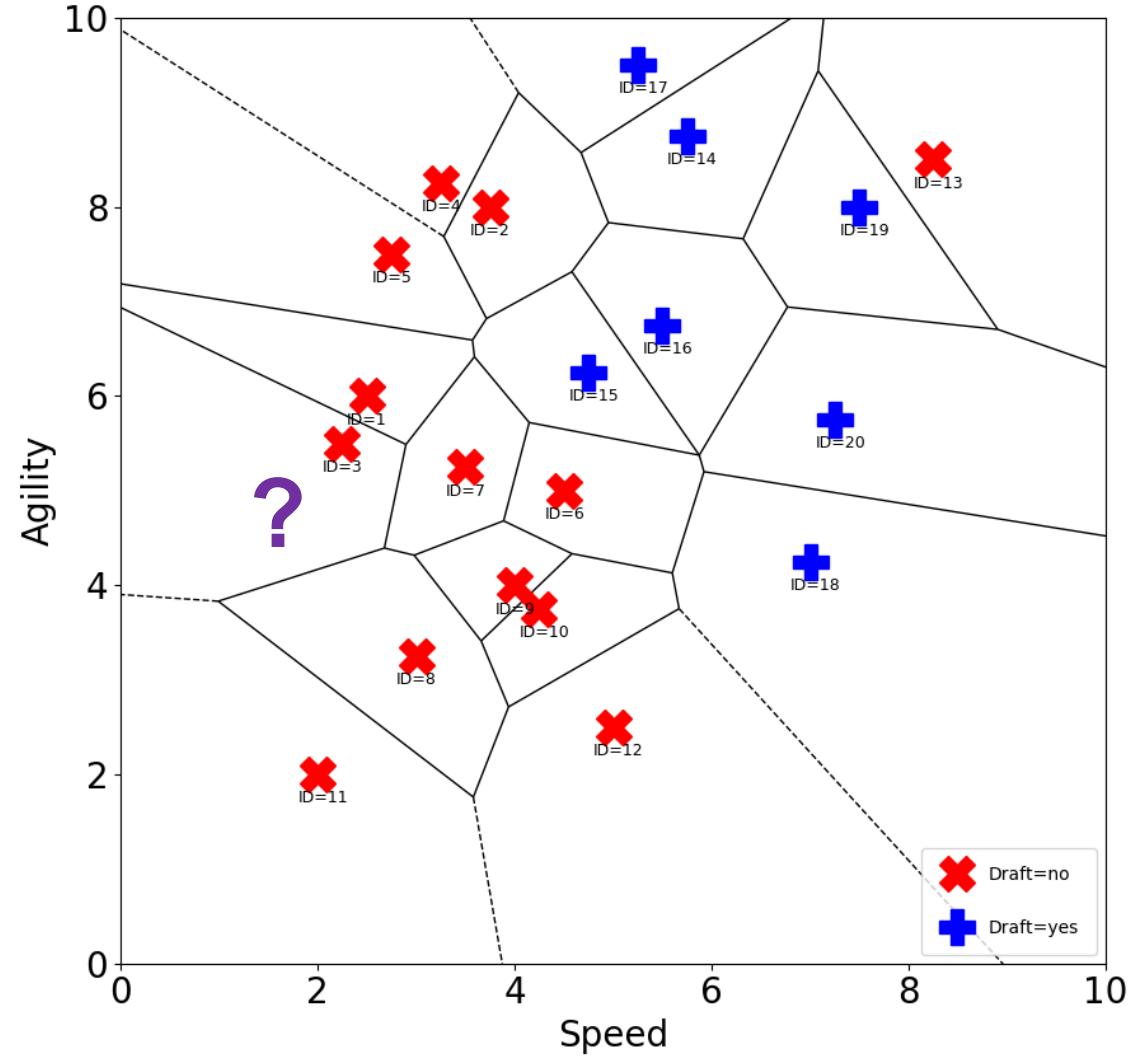
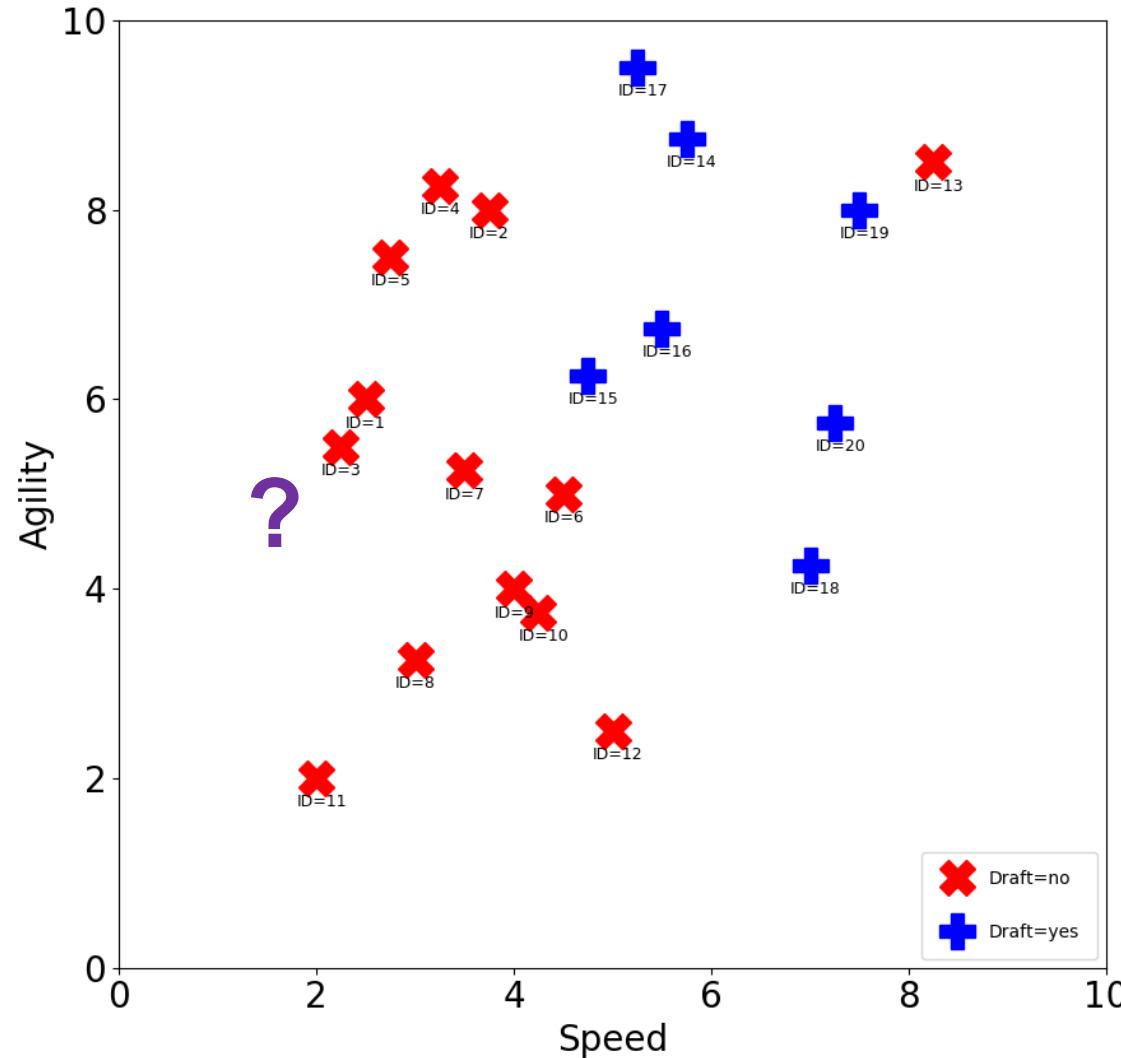
# Visualising decision boundaries

- 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



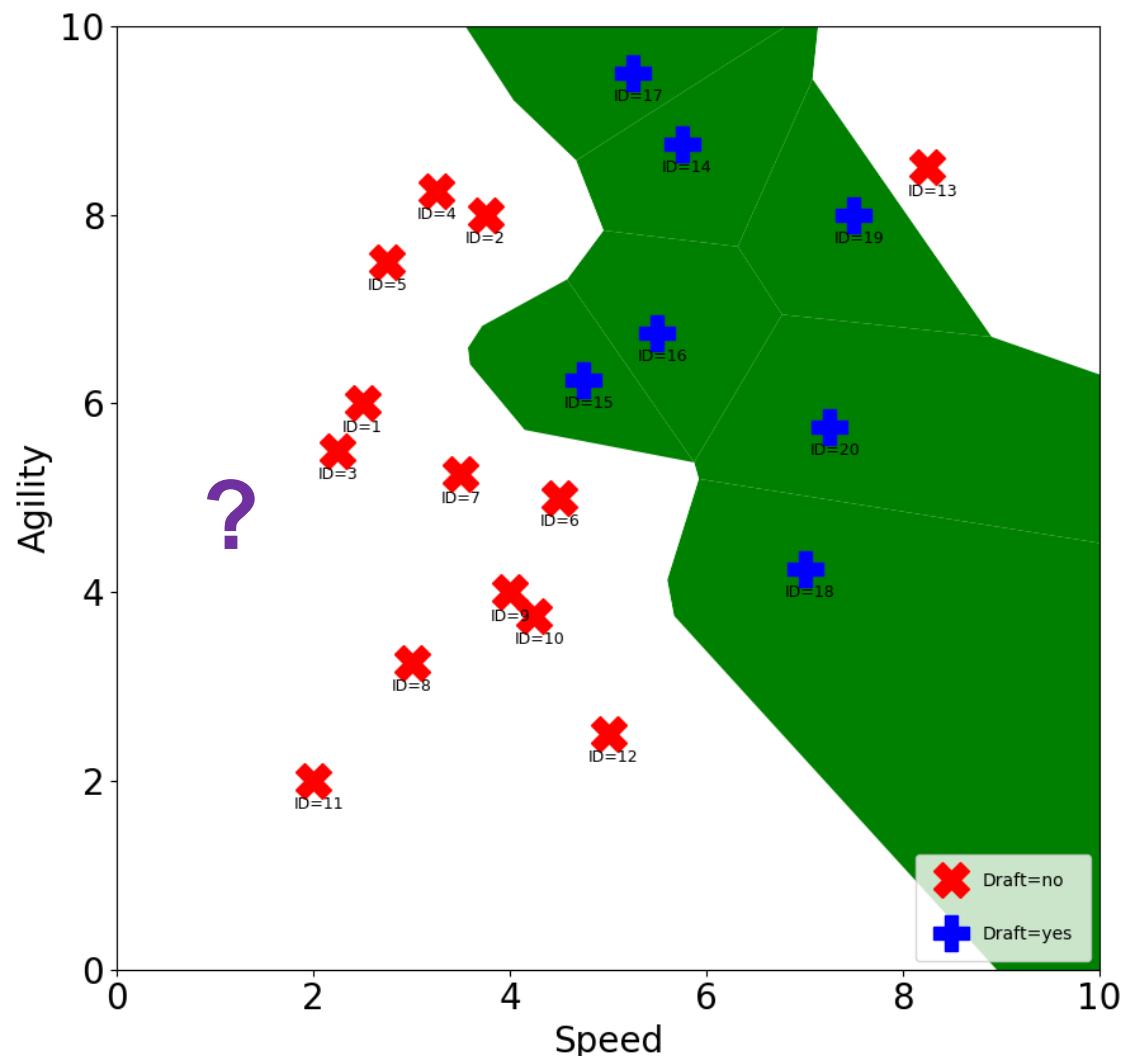
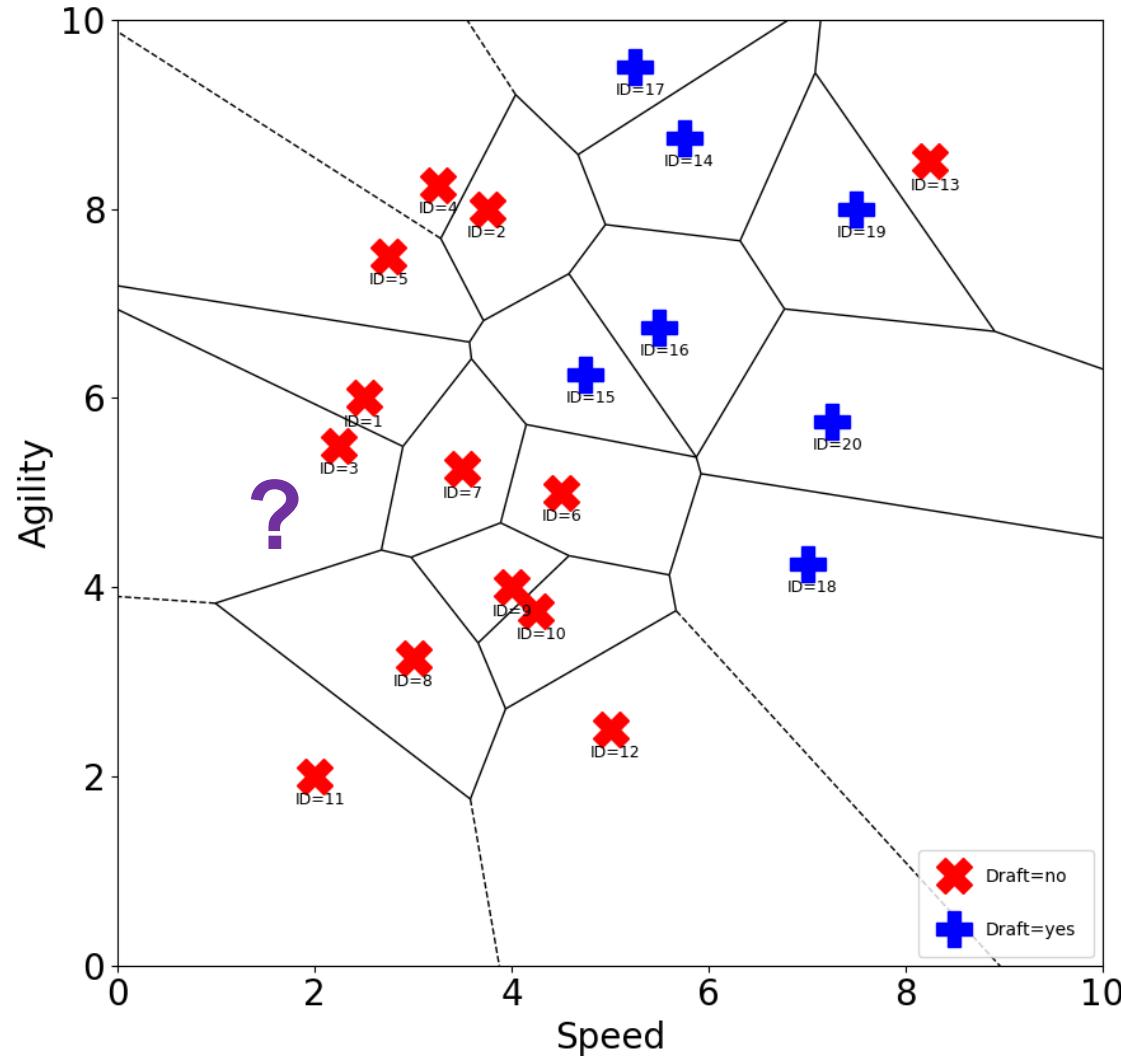


# Voronoi tessellation





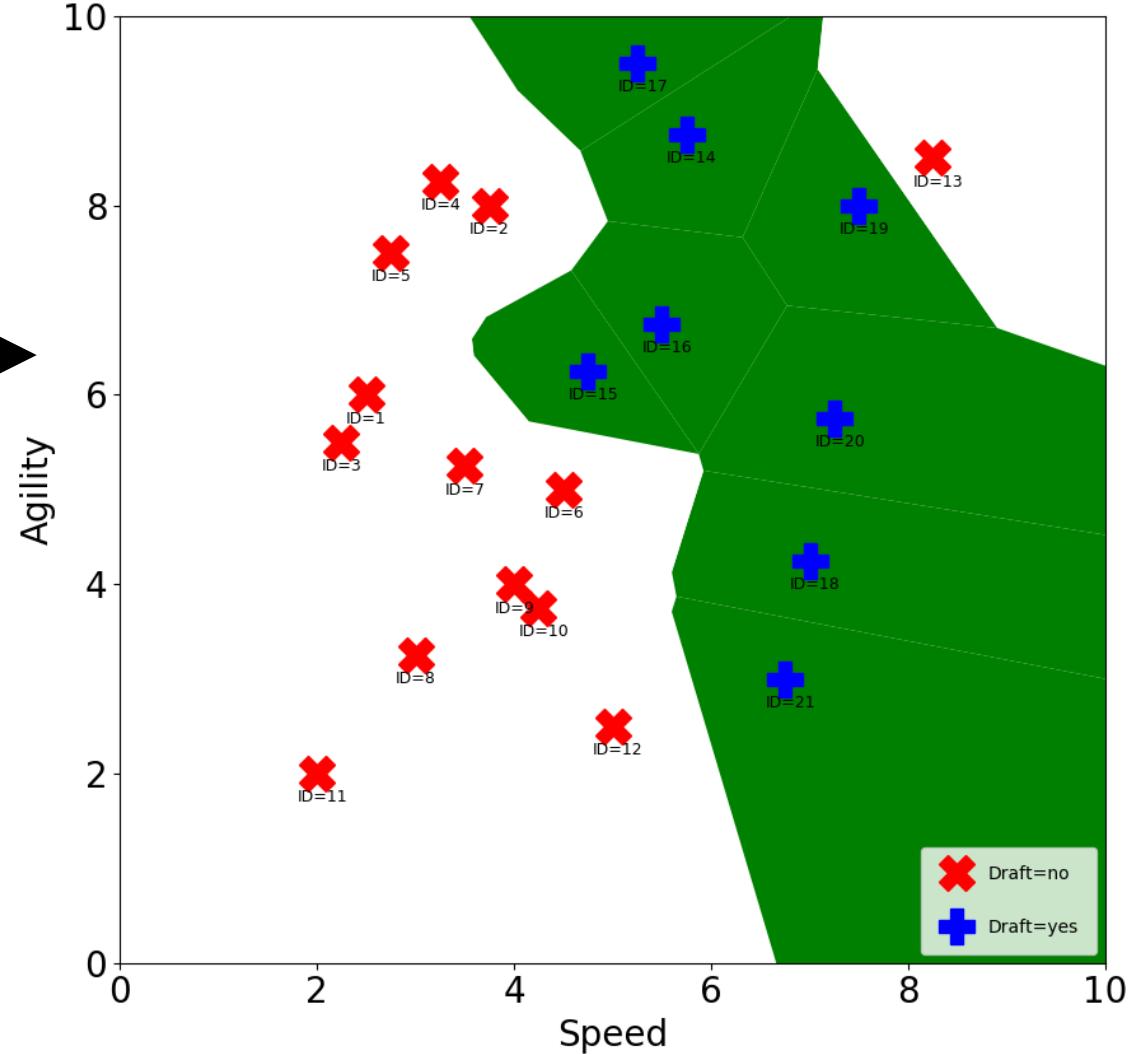
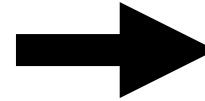
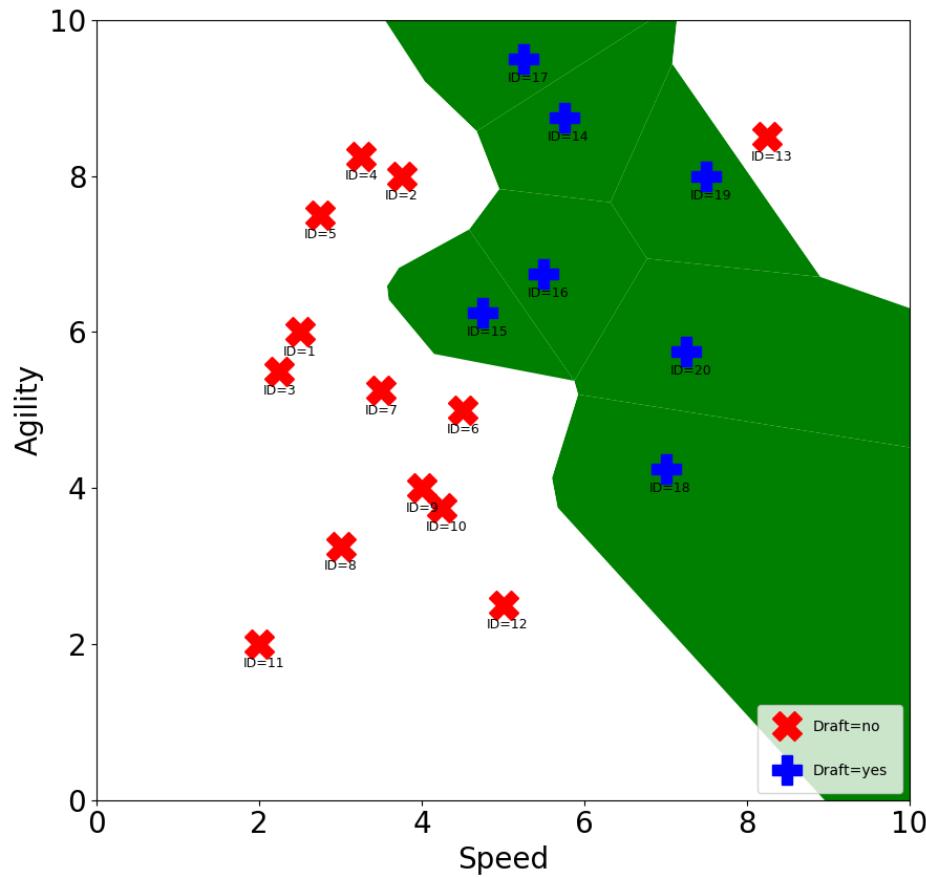
# 1-NN Decision boundary from Voronoi tessellation





# Effect of adding more training data

ID	Speed	Agility	Draft?
21	6.75	3.00	yes





# Simple Example of 1-NN

- ***HousePrices-1NN.xlsx***
  - Very simple & inflexible 1-NN implementation
  - Cannot handle other datasets directly
- How it works:
  - Training data scaled using z-normalisation (more on normalisation later)
  - Enter a query: it is scaled using same scale factors
  - Euclidean/Manhattan distances between query and each instance in the training set are computed
  - Minimum distance identified
  - Look up corresponding row to get 1-NN price estimate

	Size (m <sup>2</sup> )	# Beds	# Floors	Age (yrs)	Price (k€)
A	195	5	1	40	450
B	130	3	2	35	220
C	140	3	2	26	310
D	80	2	1	30	170
E	180	5	2	38	400
Query	130	4	2	30	?



# Similarity-based learning

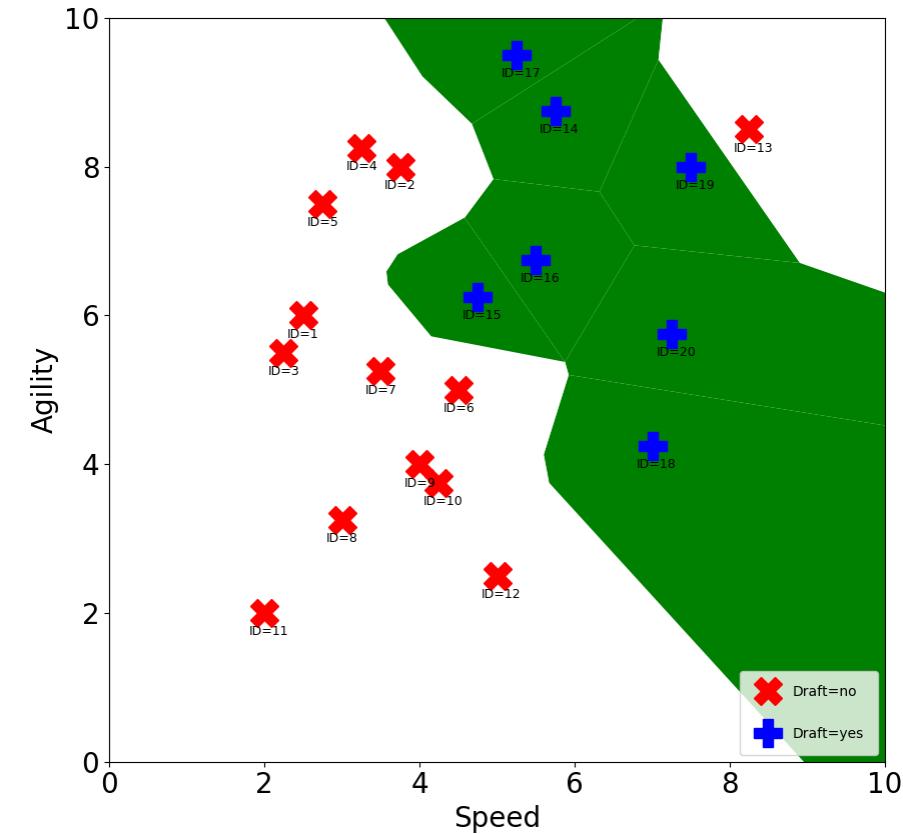
Part 5:

## The $k$ Nearest Neighbours algorithm



# $k$ 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)



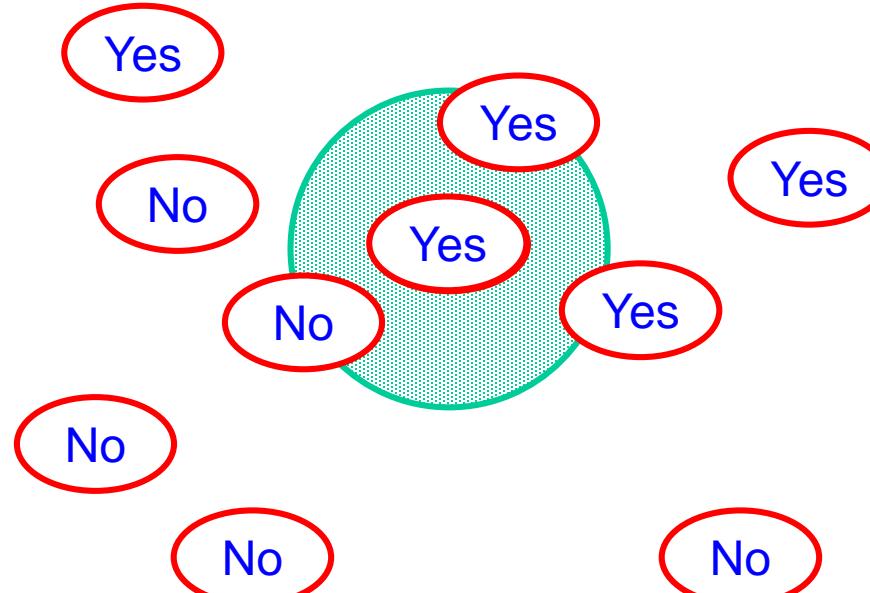


# ***k Nearest Neighbours Algorithm***

- ***k Nearest Neighbours algorithm:***
  - Base prediction on several ( $k$ ) nearest neighbours
  - Compute distance from 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

**3 nearest neighbours vote:  
Decision is Yes**

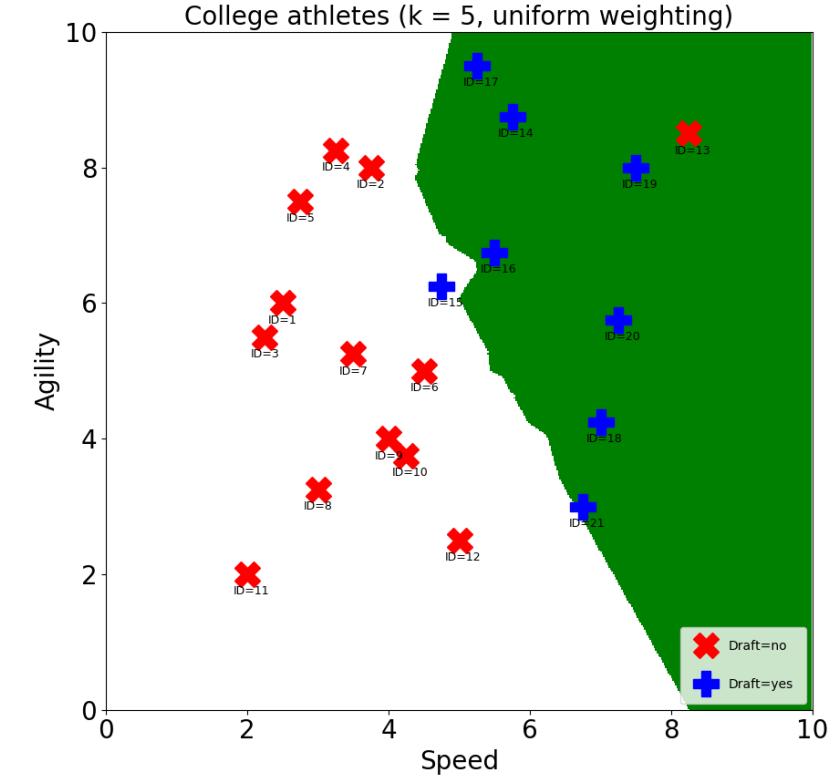
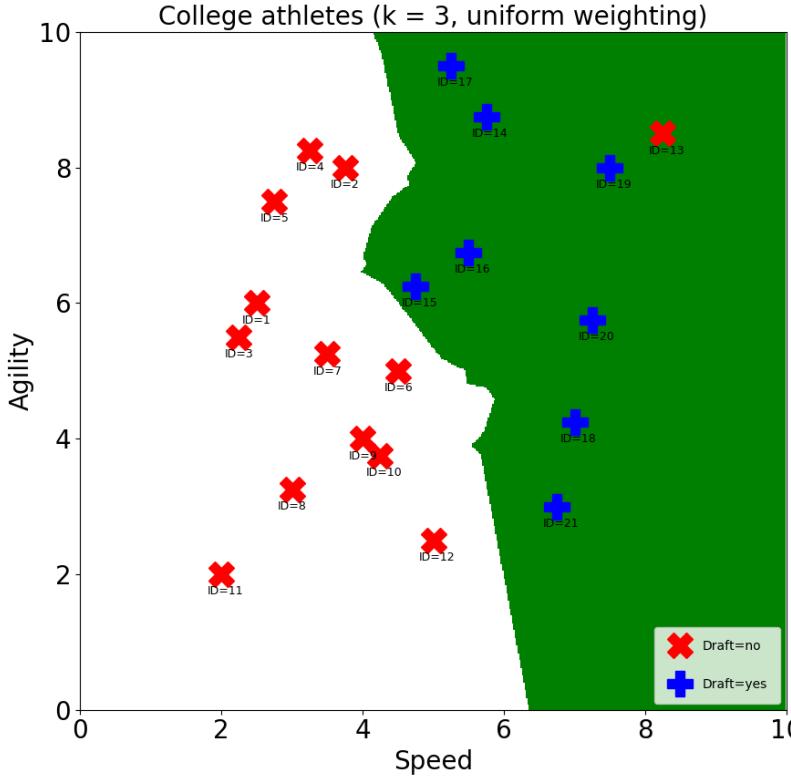
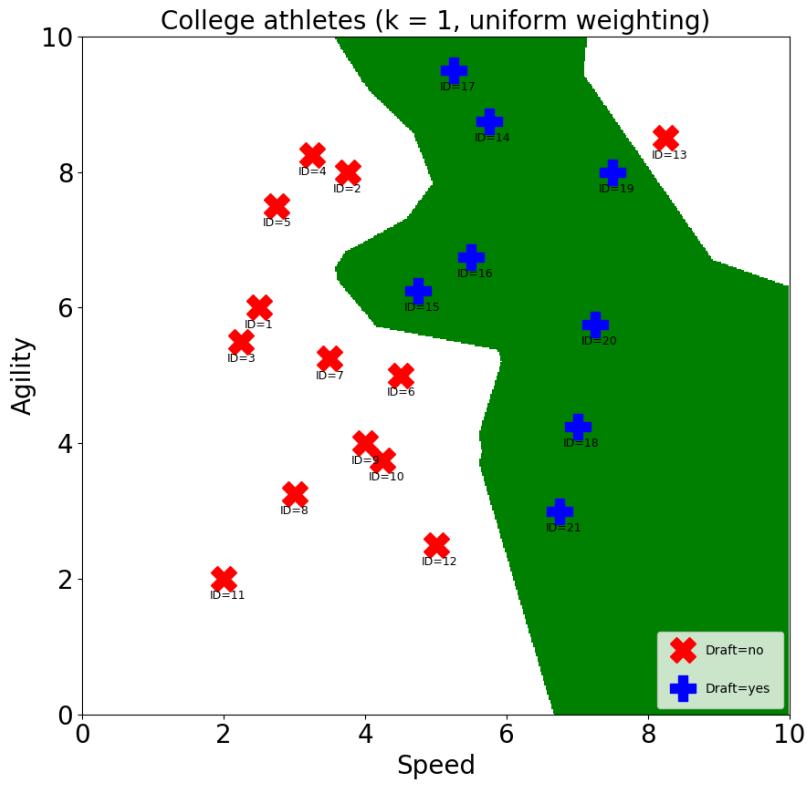


# Choosing a value for $k$

- 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)

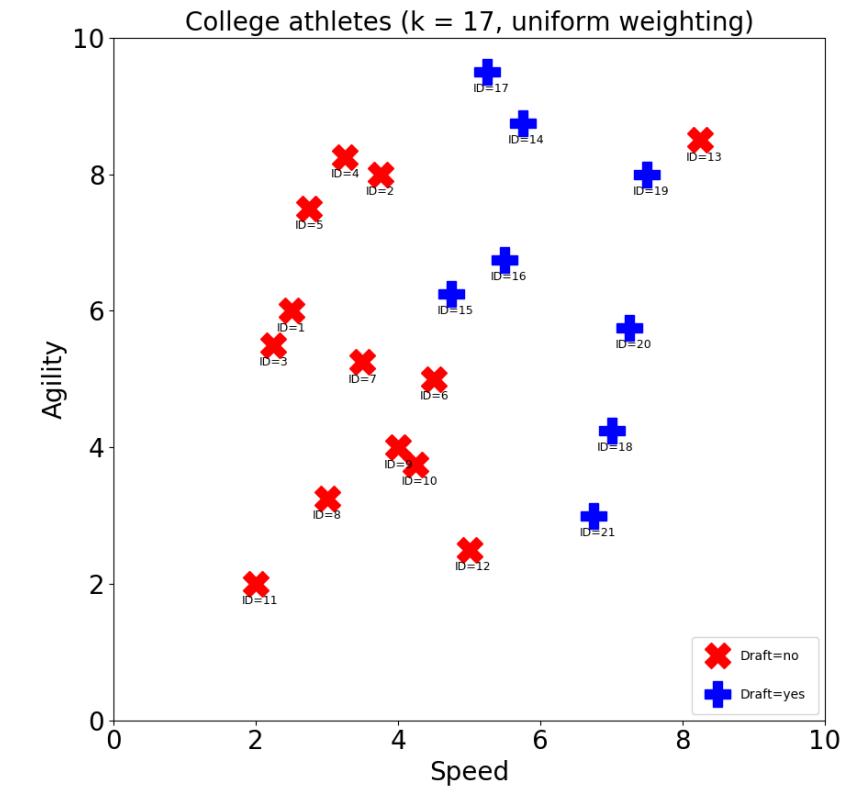
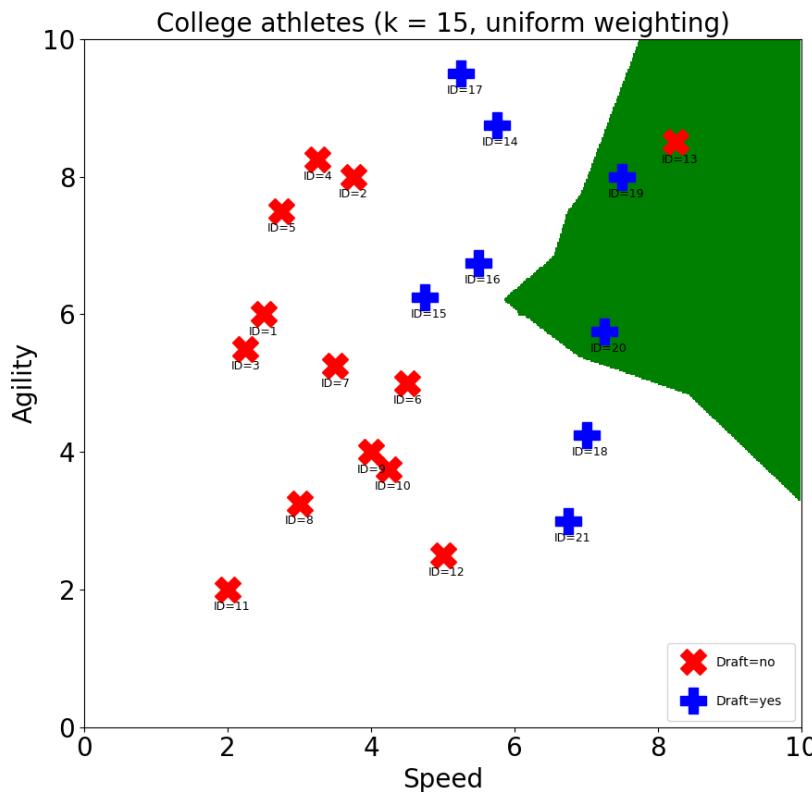
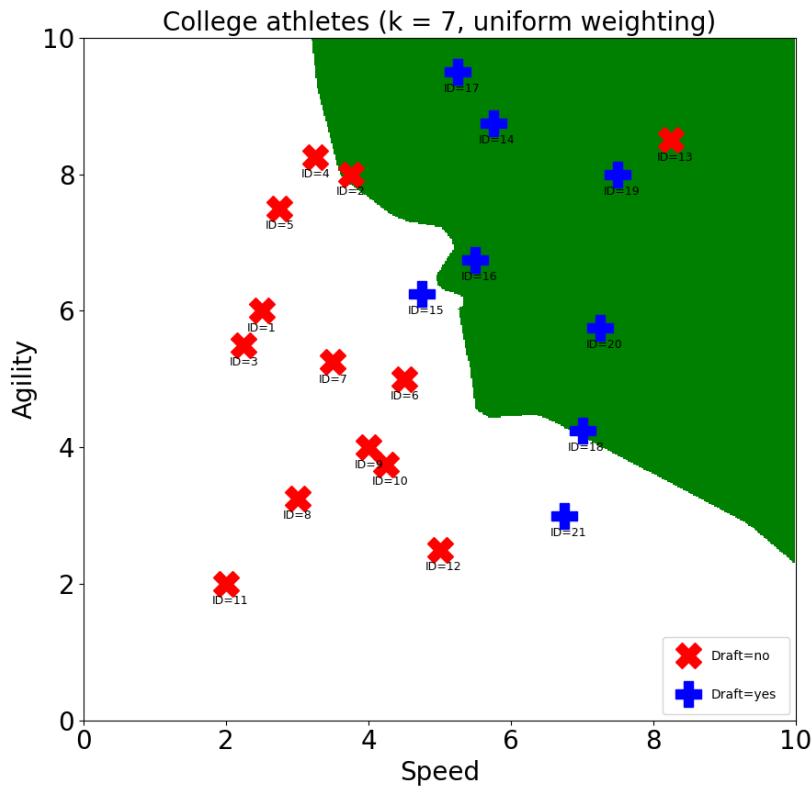


# Effect of increasing $k$





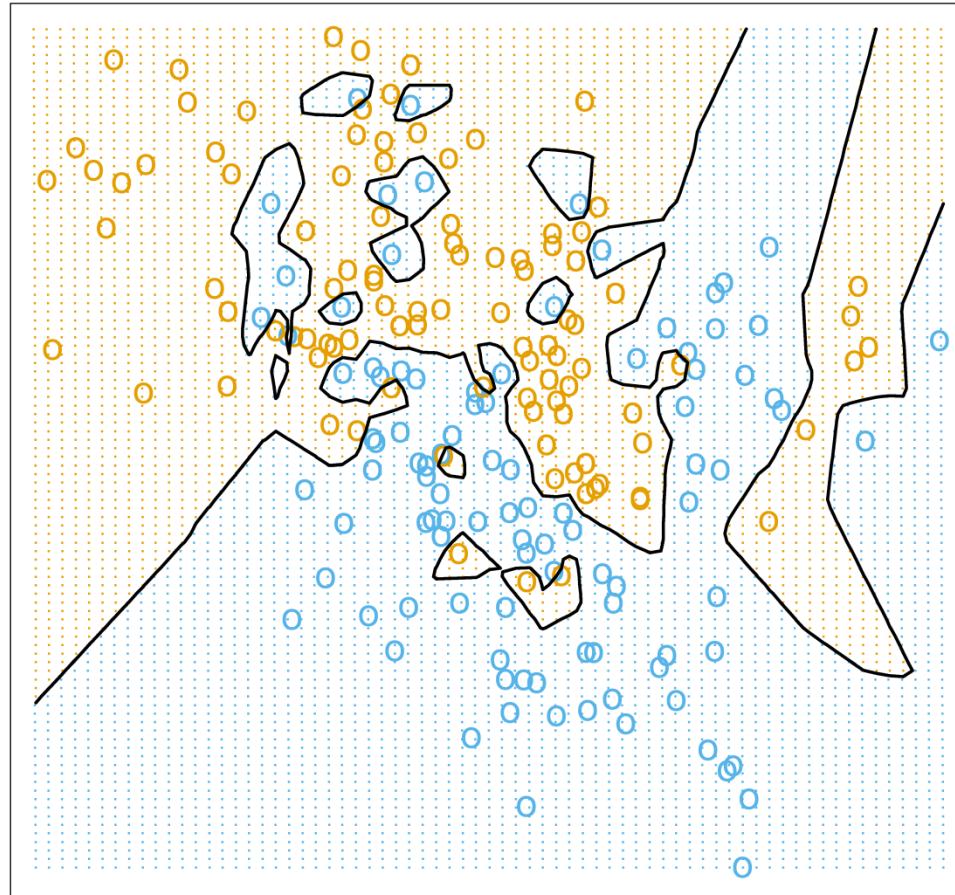
# Effect of increasing $k$



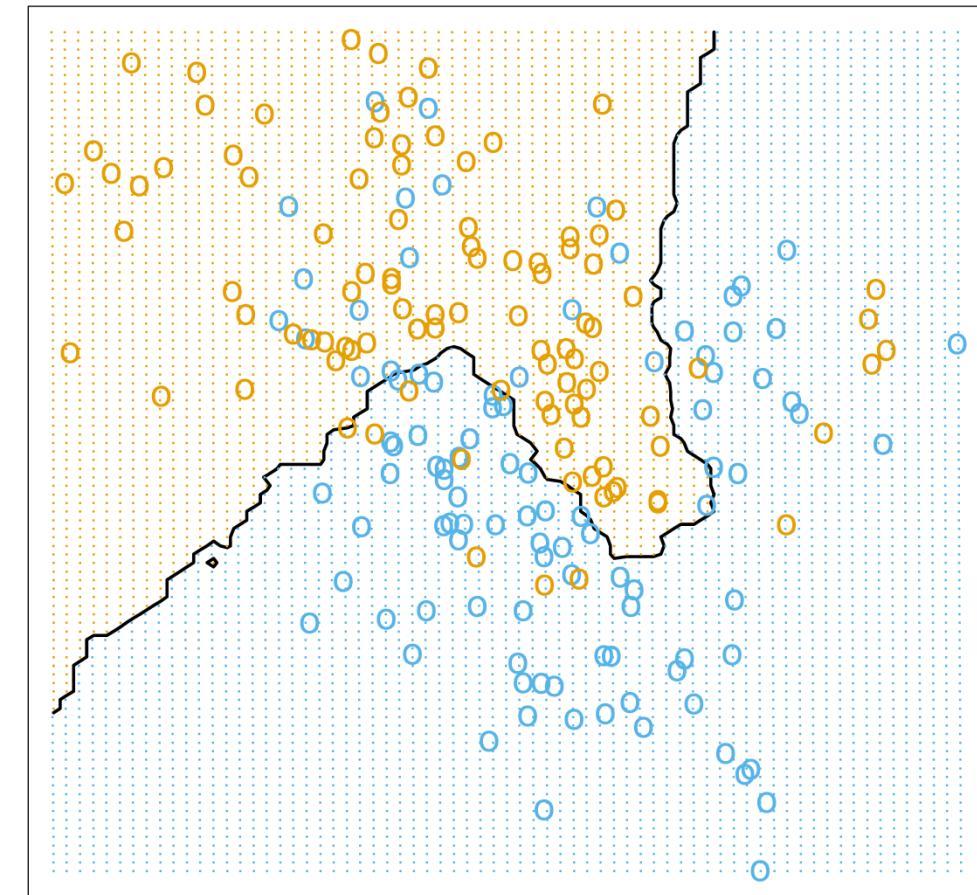


# Smoothing Effect of $k$

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier

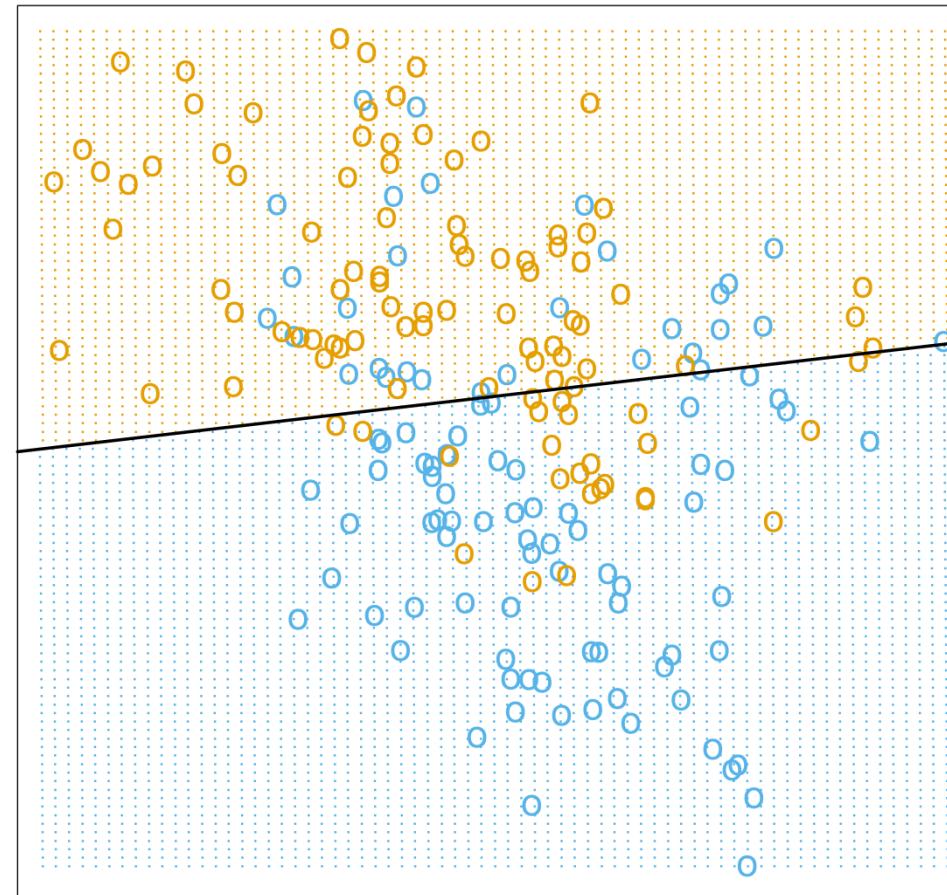


Hastie, Tibshirani & Friedman, Figs 2.3 and 2.2. Recall discussion of overfitting in Topic 2.



## Aside: Underfitting on Same Data

Linear Regression of 0/1 Response



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

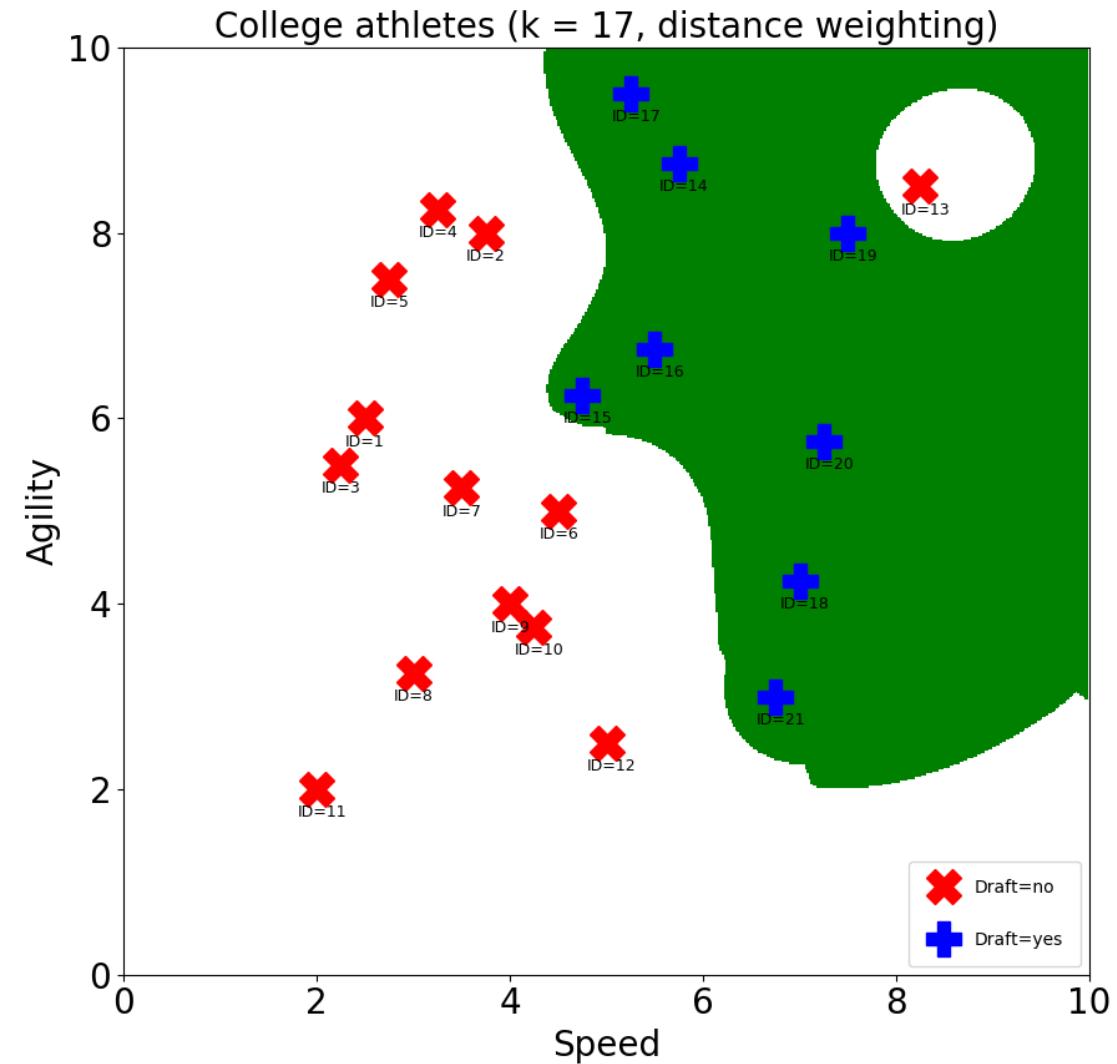
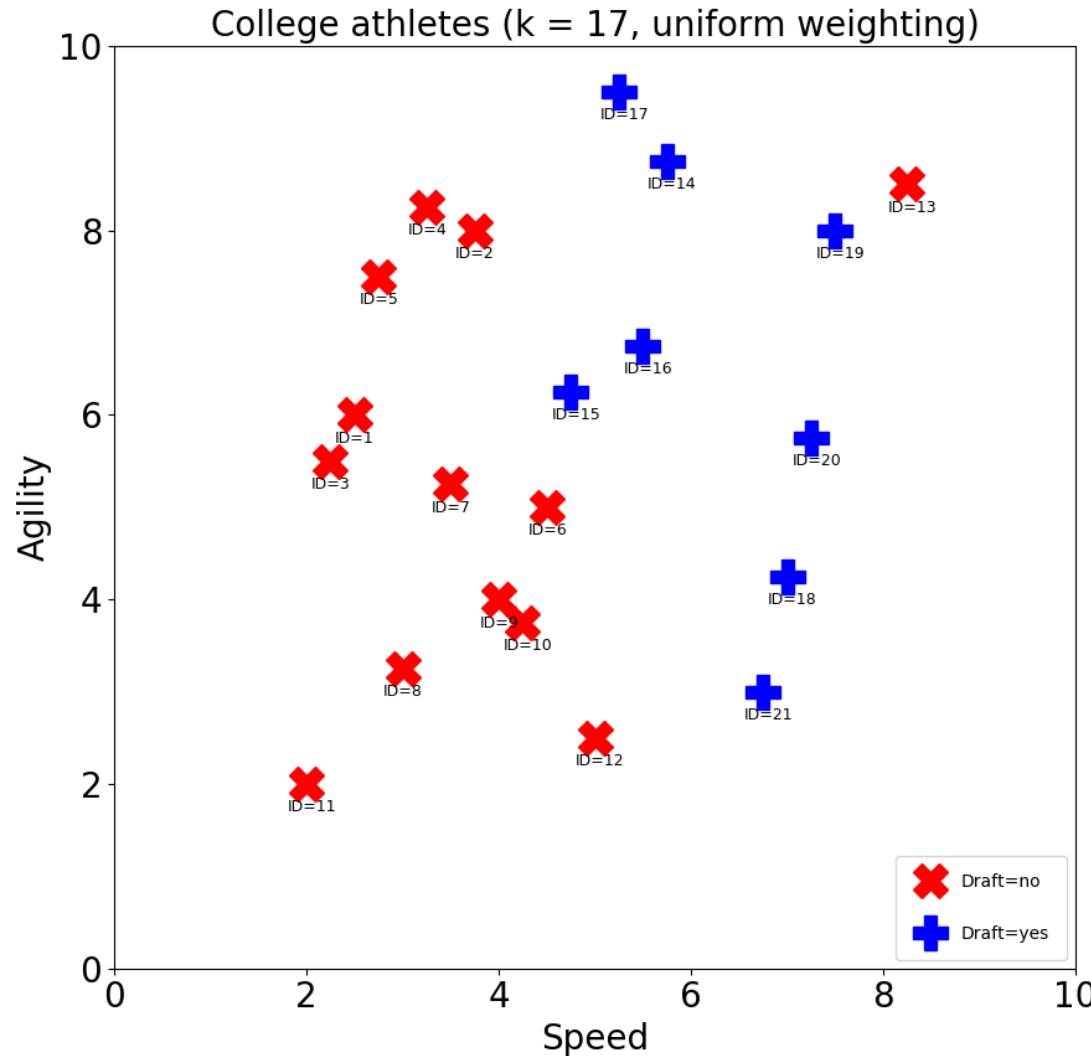


# Distance-weighted kNN

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



# Effect of distance weighting





# Similarity-based learning

## Part 6: Alternate similarity measures



## Recap: measuring similarity

- So far, we have measured similarity using distance metrics only
- Each independent variable/attribute is treated as a dimension in hyperspace
- We discussed the well-known Euclidean and Manhattan distance metrics
- We also considered the Minkowski distance, which generalises both Euclidean and Manhattan distances
- In this section we will discuss some issues which can be encountered when measuring similarity and introduce some alternate ways to measure similarity (e.g. similarity indices)
- Note: difference between a similarity index and a distance metric



# Data normalisation

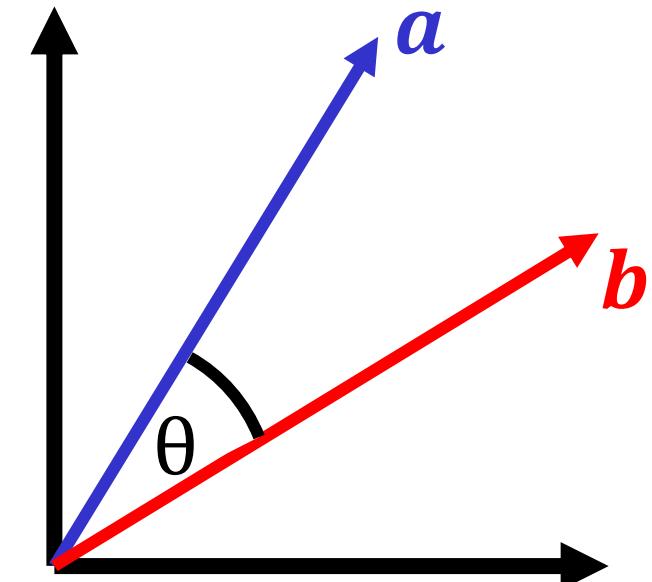
- Problem — Scaling:
  - Attribute 1 has range 0-10,  
Attribute 2 has range 0-1000
  - Attribute 2 will dominate calculations
- Solution:
  - Rescale all dimensions independently
    - Mean=0, Std deviation=1 [Z-Normalisation] ([HousePrices-1NN.xlsx](#) has a worked example)  
$$D \leftarrow (D - \text{Mean}) / \text{StDev}$$
    - Min=0, Max=1 [0-1 Normalisation] (also referred to as “range normalisation”)  
$$D \leftarrow (D - \text{Min}) / (\text{Max} - \text{Min})$$
 ([also utopia = max, nadir=min](#))

Normalisation is important in many other areas of machine learning and optimisation



# Cosine similarity

- Cosine similarity is an **index** that may be used to measure the similarity of two instances with continuous attributes. It is the **cosine** of the inner angle between the two vectors that extend from the origin to the points of interest in the feature space
- As before  $m$  is the number of features/attributes (i.e. the dimension of the vectors  $\mathbf{a}$  and  $\mathbf{b}$ )
- $\mathbf{a} \cdot \mathbf{b}$  is the vector dot product, and  $|\mathbf{a}|$  is the magnitude of the vector  $\mathbf{a}$

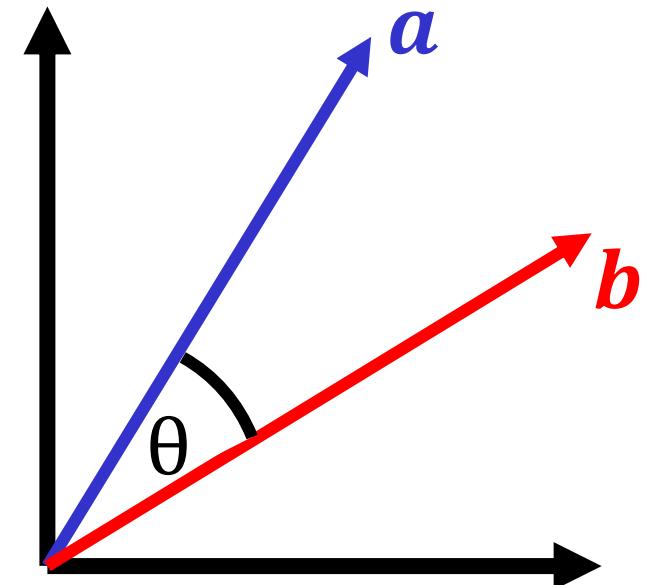


$$\text{Cosine}(\mathbf{a}, \mathbf{b}) = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} = \frac{\sum_{i=1}^m (\mathbf{a}[i] \times \mathbf{b}[i])}{\sqrt{\sum_{i=1}^m (\mathbf{a}[i])^2} \times \sqrt{\sum_{i=1}^m (\mathbf{b}[i])^2}}$$



# Cosine similarity

- The values of  $\text{Cosine}(a,b)$  will be in the range 0 to 1 if all attribute values are non-negative, or -1 to +1 if negative values are present. Can use 0-1 range normalisation to ensure  $\text{Cosine}(a,b)$  in 0-1 range.
- $\text{Cosine}(a,b) = 1 \rightarrow$  most similar (vectors are parallel)
- $\text{Cosine}(a,b) = 0 \rightarrow$  least similar (vectors at 90 deg)
- **Note:** for this index higher values indicate greater similarity (for distance metrics the opposite is true)
- $\text{Cosine}(a,b)$  allows **scale-invariant** comparisons between instances



$$\cos(0) = 1.0$$

$$\cos(90) = 0.0$$



# 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)
- One 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.



# Similarity for discrete attributes

Similarity measures for binary (e.g. yes/no) attributes include:

- **The Russel-Rao similarity index**
  - Focuses on measuring “co-presences”, e.g. when comparing users of a web store, compare which pages/products they have clicked on (the attributes are for each product/page, whether that product/page was visited before)
  - The similarity score of two users is based on how many co-presences they share
- **The Sokal-Michener similarity index**
  - Measures similarity using co-presences as well as co-absences
  - Co-absences of attributes may be just as important as co-presences in some domains, e.g. in medical applications, it may be important to compare which symptoms each patient has, as well as which symptoms each patient does not have



# Choosing a Distance Metric (1)

- Choices will be limited by attribute data type, e.g. continuous or discrete
- In general, want a metric that:
  - Reflects differences between cases that are important
  - Downplays differences that are irrelevant
  - Application-dependent: work needed here
- E.g. Cosine similarity based on angle, not absolute value
  - May be good to compare objects if we care that they are the same shape but not the same scale
  - Less good if the scale is important



## Choosing a Distance Metric (2)

- Possible to design distance metrics of our own
  - Given our understanding of what is important in a domain, can formulate a metric that emphasises what is important and de-emphasises what is not
- Remember that distance metrics must obey **Metric Axioms**:
  1. **Non-negativity**:  $\text{metric}(a, b) \geq 0$
  2. **Identity**:  $\text{metric}(a, b) = 0 \Leftrightarrow a = b$
  3. **Symmetry**:  $\text{metric}(a, b) = \text{metric}(b, a)$
  4. **Triangular Inequality**:  $\text{metric}(a, b) \leq \text{metric}(a, c) + \text{metric}(b, c)$
- These axioms may occasionally be violated in similarity measures, but only with good reason!
  - E.g. “A contains B” is non-symmetric



# Similarity-based learning

## Part 7: Predicting continuous targets



# kNN - recap

- **$k$ -Nearest Neighbour algorithm:**
  - Base prediction on several ( $k$ ) nearest neighbours
  - Compute distance from query case to all stored cases, and pick the nearest  $k$  neighbours
- Classification with kNN:
  - Neighbours vote on classification of test case
- **Regression:**
  - **Average the value of the neighbours**



## kNN regression – uniform weighting

$$\text{prediction}(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^k t_i$$

- $\mathbf{q}$  is a vector containing the attribute values for the query instance
- $k$  is the number of neighbours as before
- $t_i$  is the target value for neighbour  $i$
- This assumes that each neighbour is given equal weighting



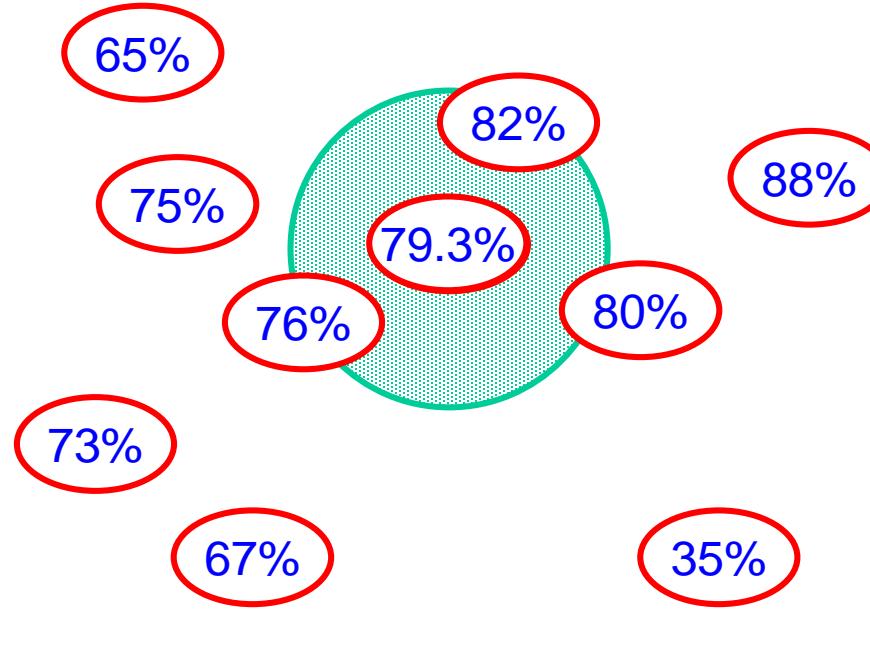
## kNN regression – distance weighting

$$\text{prediction}(\mathbf{q}) = \frac{\sum_{i=1}^k \left( \frac{1}{\text{dist}(\mathbf{q}, \mathbf{d}_i)^2} \times t_i \right)}{\sum_{i=1}^k \left( \frac{1}{\text{dist}(\mathbf{q}, \mathbf{d}_i)^2} \right)}$$

- $\mathbf{q}$  is a vector containing the attribute values for the query instance
- $\text{dist}(\mathbf{q}, \mathbf{d}_i)$  returns the distance between the query and neighbour  $i$
- This assumes that each neighbour is given a weighting based on the inverse square of its distance from the query



# $k$ Nearest Neighbours: Visualisation of Regression Example



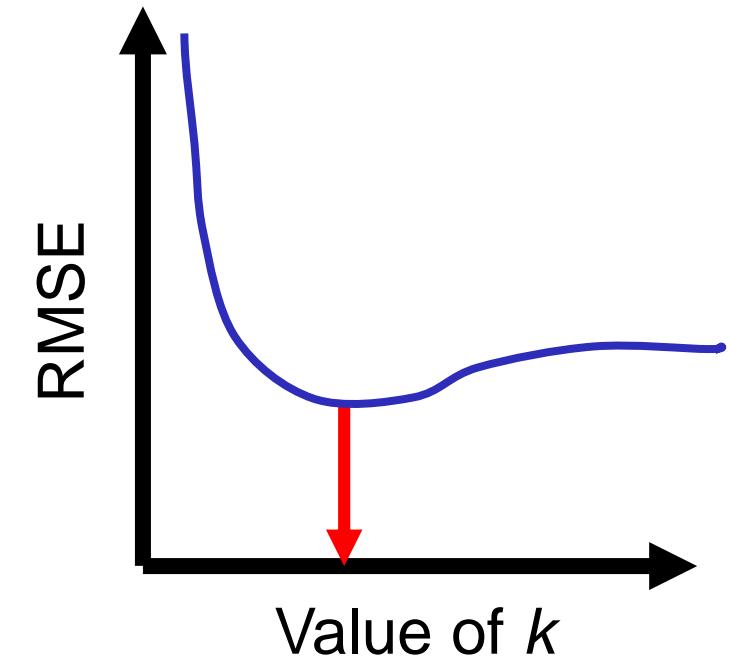
**Estimate: 79.3%**

This visualisation assumes Euclidean distance and uniform weighting



# Selecting $k$ for regression tasks

- As with  $k$ NN for classification, selecting an appropriate value of  $k$  is important for regression tasks. How should we set it?
- One solution: test a range of values of  $k$  and select the one that gives the best score on your chosen evaluation metric, e.g. Root Mean Square Error (RMSE). Should incorporate cross validation when computing the RMSE for each value of  $k$
- Note: even values of  $k$  aren't a problem for regression tasks as majority voting isn't used



Pick the value of  $k$  that  
Gives the lowest RMSE

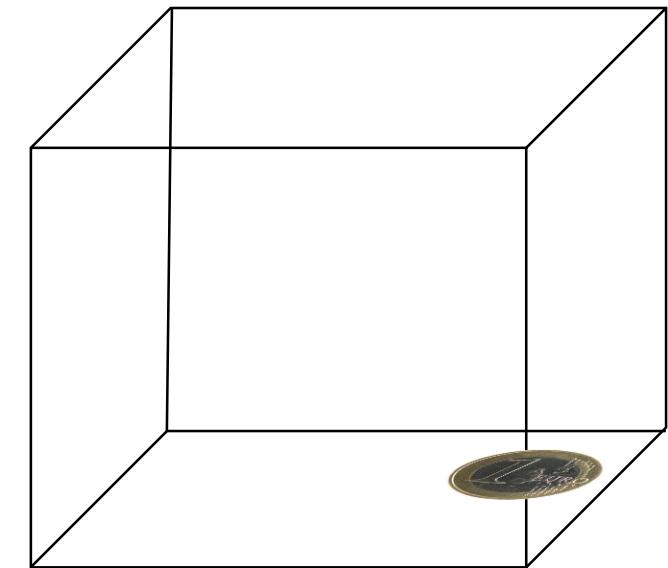
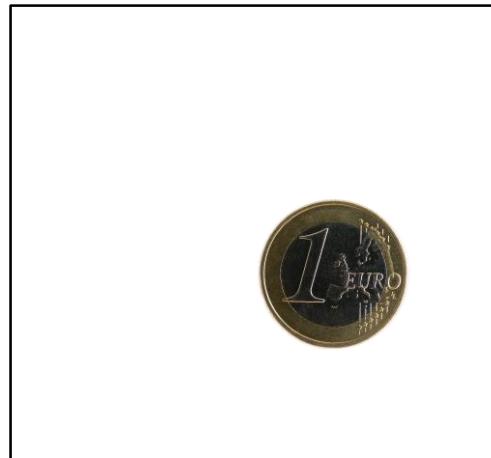


# Similarity-based learning

## Part 8: Feature selection



# The Curse of Dimensionality [1]





# The Curse of Dimensionality [2]

- Problem — Curse of Dimensionality:
  - Some attributes are much more significant than others
  - All considered equally in distance metric => bad predictions
  - With many attributes, everything becomes 'distant' [see next]
- Solution a:
  - Assign weighting to each dimension  
**(not** same as distance-weighted kNN!)
  - Optimise weighting to minimise error
- Solution b:
  - Give some dimensions 0 weight:  
**Feature Subset Selection**

Any algorithm that considers all attributes in a high-dimensional space equally has this problem, not just kNN + Euclidean Distance!



# The Curse of Dimensionality [3]

- Russell & Norvig:

Consider  $N$  cases with  $d$  dimensions, in hypercube of **unit** volume

Assume neighbourhoods are hypercubes, length  $b$ : volume is  $b^d$

To contain  $k$  points, average neighbourhood must occupy  $k/N$  of entire volume

$$\Rightarrow b^d = k/N$$

$$\Rightarrow b = (k/N)^{1/d}$$

High dimensions:

$$k = 10; N = 1,000,000; d = 100 \Rightarrow b = 0.89$$

i.e. neighbourhood spans nearly 90% of each dimension of space!

Low dimensions:

$$k \text{ and } N \text{ unchanged; } d = 2 \Rightarrow b = 0.003 \text{ [OK]}$$

- High-D spaces are generally very sparse: all neighbours far away



# Feature Selection

- Fortunately, some algorithms partially mitigate the effects of the curse of dimensionality (e.g. decision tree learning). This is not true for all algorithms however, and heuristics for search can sometimes be misleading!
- kNN and many other algorithms use all attributes when making a prediction
- Acquiring more data is not (always) a realistic option
- The best way to avoid the curse is to use only the most useful features during learning, this process is known as feature selection



## Types of features

We may wish to distinguish between different types of descriptive features:

- **Predictive:** provides information that is useful when estimating the correct target value
- **Interacting:** provides useful information only when considered in conjunction with other features
- **Redundant:** features that have a strong correlation with another feature
- **Irrelevant:** doesn't provide any useful information for estimating the target value

Ideally, a good feature selection approach should identify the smallest subset of features that maintain prediction performance



# Feature Selection Approaches

- **Rank and prune:**
  - rank features according to their predictive power and keep only the top X%
  - A **filter** is a measure of predictive power used during ranking, e.g. information gain
  - Drawback: features evaluated in isolation, so we will miss useful interacting features
- **Search for useful feature subsets:**
  - We can pick out useful interacting features by evaluating feature subsets
  - Could generate, evaluate and rank all possible feature subsets then pick best (essentially a brute force approach, computationally expensive/infeasible?)
  - Better approach: **greedy local search**, build feature subset iteratively by starting out with an empty selection, then trying to add additional features incrementally. Requires evaluation experiments along the way. Stop trying to add more features to the selection once termination conditions are met.



# Similarity-based learning

## Part 9: Similarity-based learning considerations



# Lazy vs Eager Learning [1]

- Eager Learning
  - When given training data, construct model for future use in prediction that summarises the data
  - **Analogy: compilation in programming language**
  - Slow in model construction,  
quicker in subsequent use
  - Model itself may be useful/informative
- Eager Learning Algorithms:
  - Decision Tree, Artificial Neural Network, ...
- On the other hand ...



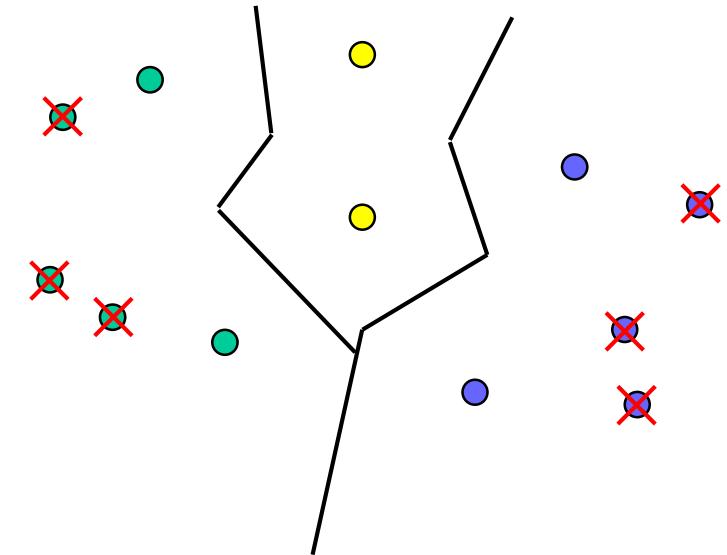
## Lazy vs Eager Learning [2]

- Lazy Learning
  - No explicit global model constructed
  - Calculations **deferred** until new case to be classified
- Creates many local approximations
  - Whereas eager learner must create a global approximation
  - For same form of hypothesis, lazy learner can represent more complex functions
  - E.g. fitting a line to neighbours rather than fitting a line to all of the data
- Lazy Learning Algorithms:
  - 1NN, k Nearest Neighbours, CBR, ...
- Concept drift



# Other Nearest Neighbour variants [1]

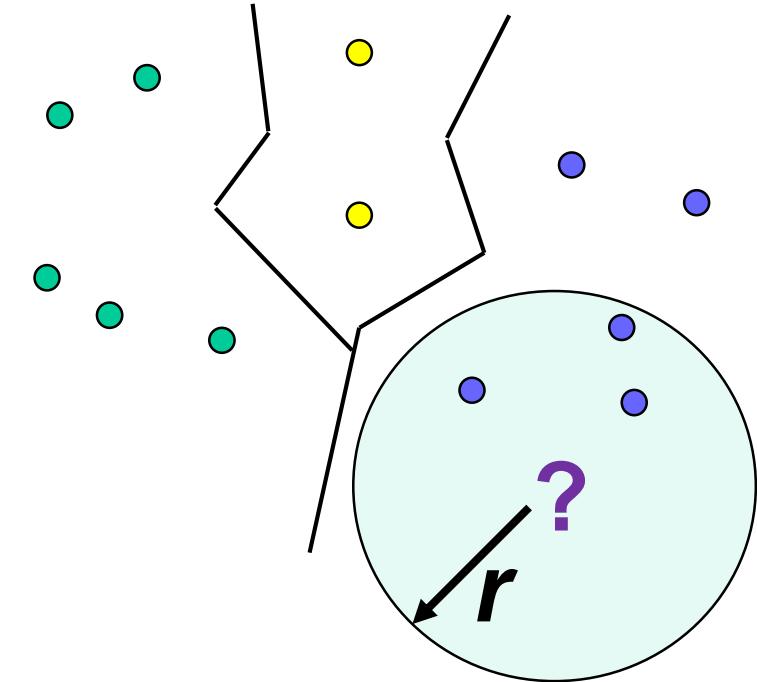
- Condensed kNN
  - Classification time proportional to:
    - number of cases
    - number of attributes per case
  - To speed up calculation,  
eliminate unnecessary ones:  
Away from **decision boundary**





## Other Nearest Neighbour variants [2]

- Radius-based Nearest Neighbours
  - Instead of selecting  $k$  nearest neighbours, could also select all neighbours within a certain radius  $r$
  - e.g. RadiusNeighborsClassifier in scikit-learn
  - $r$  should be chosen carefully
  - Normalisation is very important!
  - Can also specify a default class for instances that have no neighbours within  $r$  – potentially useful for flagging outliers





# When to use kNN

- Consider using when:
  - Plenty of training cases
  - Moderate number of attributes per case (e.g. < 20)
- Benefits:
  - Easy to implement, few parameters to tune ( $k$  + similarity measure)
  - Comprehensibility: easy to justify decisions
  - Complex target functions => good for **non-coherent concepts** (e.g. “toxic/non-toxic”). Classes don’t have to be linearly separable.
  - No summarisation => information not lost
- Drawbacks:
  - Problems with irrelevant attributes, many attributes
  - May be slow in classification/regression (no summarisation)



# Similarity-based learning

Part 10:  
Review of topic



## Learning Objectives: Review

After completing this successfully, you are now able to ...

- Explain what instance-based learning is
- Distinguish between *lazy* and *eager* learning
- Describe operation of k-Nearest Neighbours for classification and regression
- Discuss implications of the *curse of dimensionality*
- Discuss implications of selecting different distance metrics
- Identify suitable applications for kNN and explain how it could be applied



# Reinforcement Learning

## Part 1: Introduction



# Learning objectives

Having completed this topic, you should be able to ...

- Explain concepts of reward-based learning & reinforcement learning
- Define terms such as partially/fully observable; stochastic/deterministic; benign/adversarial; discrete/continuous
- Define the Markov property, Markov chains & MDPs
- Describe & implement an algorithm to find optimal policy for MDPs
- Describe & implement reinforcement learning algorithms
- Analyse problems to determine how they can be framed in terms of reinforcement learning
- Discuss some classic applications of reinforcement learning



# Overview of topic

1. Introduction and learning objectives
2. Motivation
3. General principles
4. Markov decision processes
5. The Q-learning algorithm
6. Q-learning worked example

*N.B. this topic is a very brief introduction to the field of RL. A decent textbook such as “Reinforcement Learning: An Introduction” by Sutton and Barto is a good starting point if you would like to learn more. A .pdf copy of the Sutton and Barto book is available for free online.*



# Reinforcement Learning

## Part 2: Motivation



# What is an agent?

“Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators”  
(Russell & Norvig, 2009)

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while(true){
            input = Console.ReadLine();
            if (input == "exit")break;
            newchild.Properties["co"].Add(
                ("Auditing Department");
                ("newchild.CommitChanges()");
                ("newchild.Close()");
                ("newchild"));
        }
    }
}
```

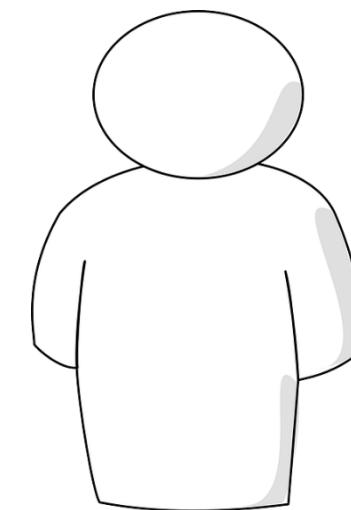
A Computer Program



A Computer



A Robot



A Person



# What is an agent?

- Agents are typically designed to solve decision making problems. These can be “single shot” or sequential – we will focus on sequential problems.
- How does an agent interact with the world?
  - Sensors: to sense the state of the world
  - Actuators: to interact with the world
  - Policy: specifies how an agent should act, based on the perceived state of the world



# What is a state?

- The sensory information available to an agent at a particular timestep
- Agent's state information may be limited (e.g. cannot sense items hidden by a “fog of war” in RTS games)
- Examples of state information:
  - Position (x,y,z coordinates), fuel remaining (robotics, autonomous vehicles)
  - Positions of pieces on a game board (e.g. draughts, backgammon, chess, go)
  - Positions of allies/enemies/powerups (FPS games)
  - Current amounts of resources available (RTS games)
  - Distributions of vehicles queueing at an intersection (traffic signal control)
  - Current demand level/power output (electricity generation scenarios)



# What is an action?

- Actions carried out by an agent change the environmental conditions
- Actions may not always be successful and may not always have the same outcome!
- Examples of actions:
  - Change an agent's/vehicle's/robot's position (e.g. move north/south, up/down)
  - Pick up an object (e.g. ammo, weapons, powerups)
  - Shoot an enemy (FPS games)
  - Build a new unit or structure, research a technology (RTS games)
  - Change the lights at an intersection (traffic signal control)
  - Change the power output of a generator (electricity generation scenarios)



# What is a policy?

- A policy is a mapping from environmental states to actions
- An agent's policy determines how it will act for a given sensory input
- Policies can easily be hand-coded for agents in simple environments (e.g. finite state machines have been widely used in video games)
- More difficult to hand-code policies for complex environments; becomes time consuming. We may not always know the correct action, but we usually know whether a task is being performed well (e.g. high scores in a video game are good).
- Rather than explicitly programming all possible agent behaviour, could instead create agents that can learn how to act (near) optimally
  - **This is the motivation for using reinforcement learning (RL)!**
  - Policies can be generated using many different techniques, e.g. RL, planning, evolution, metaheuristic algorithms, imitation learning etc. We will focus on RL.



# Reinforcement Learning

## Part 3: General principles



# What is Reinforcement Learning (RL)?

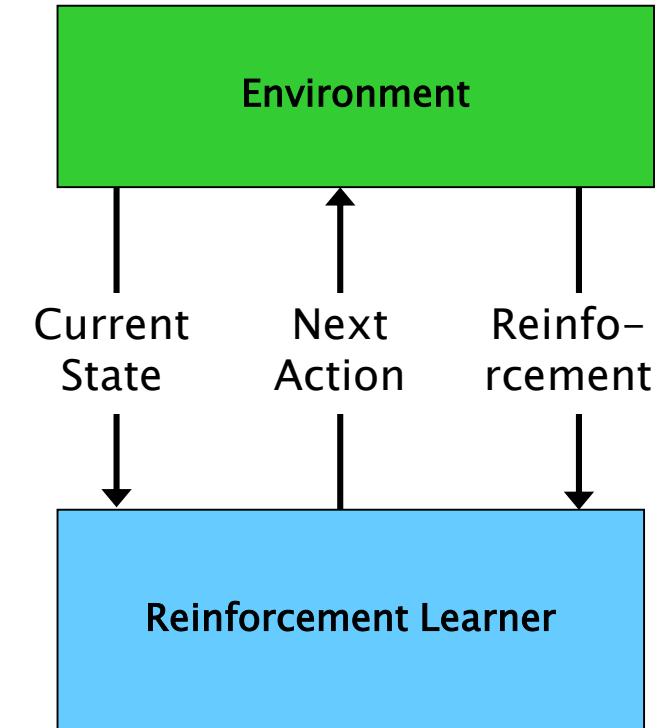
- RL agents learn by interacting with their environment
- Simple analogy – training a pet
- Desirable behaviour is rewarded and undesirable behaviour is punished when training a pet – similar principles apply when training an RL agent
- RL is inspired by concepts in the behaviourism literature (reinforcement)





# RL overview (1)

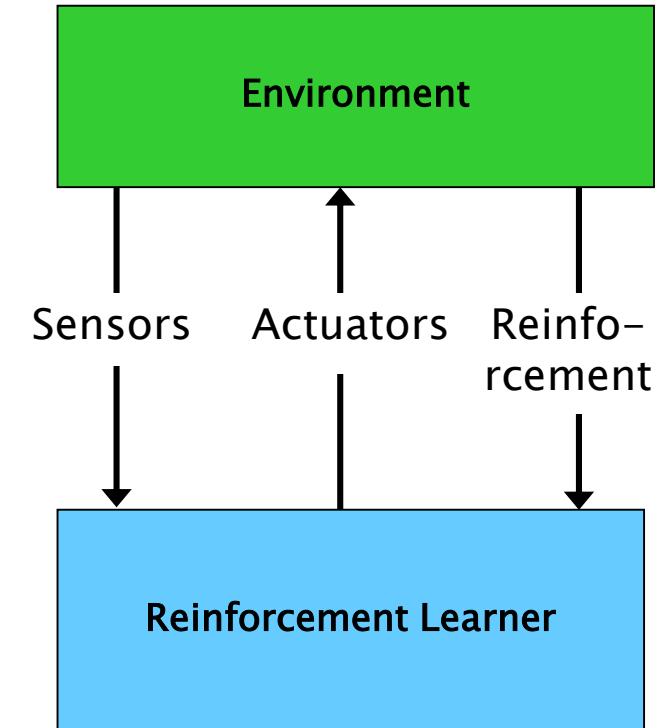
- Learning through trial and error
  - Agent **explores** environment
  - **Rewards** for successful outcomes
  - Punishments for unsuccessful ones
- Perception and action
  - State corresponds to what is perceived
- May Be:
  - Fully or Partially Observable [camera points forward]
  - Discrete or Continuous
  - Benign or Adversarial
  - Deterministic or Stochastic [discussed later]





## RL overview (2)

- Sensors and Actuators
- Seeks to maximise long-term reward
  - Acts randomly at first
  - Builds map of states+actions -> rewards
  - Gradually develops optimal strategy
  - Rewards may be **delayed** (not immediate)





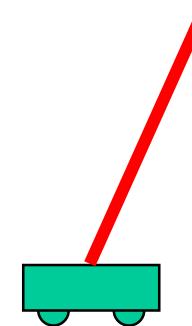
## RL is appropriate when ...

- Tasks are poorly defined
- Domain is not fully known
- Don't know immediate effect of all actions
- Don't know **how** task is done well,  
just **whether** it is done well



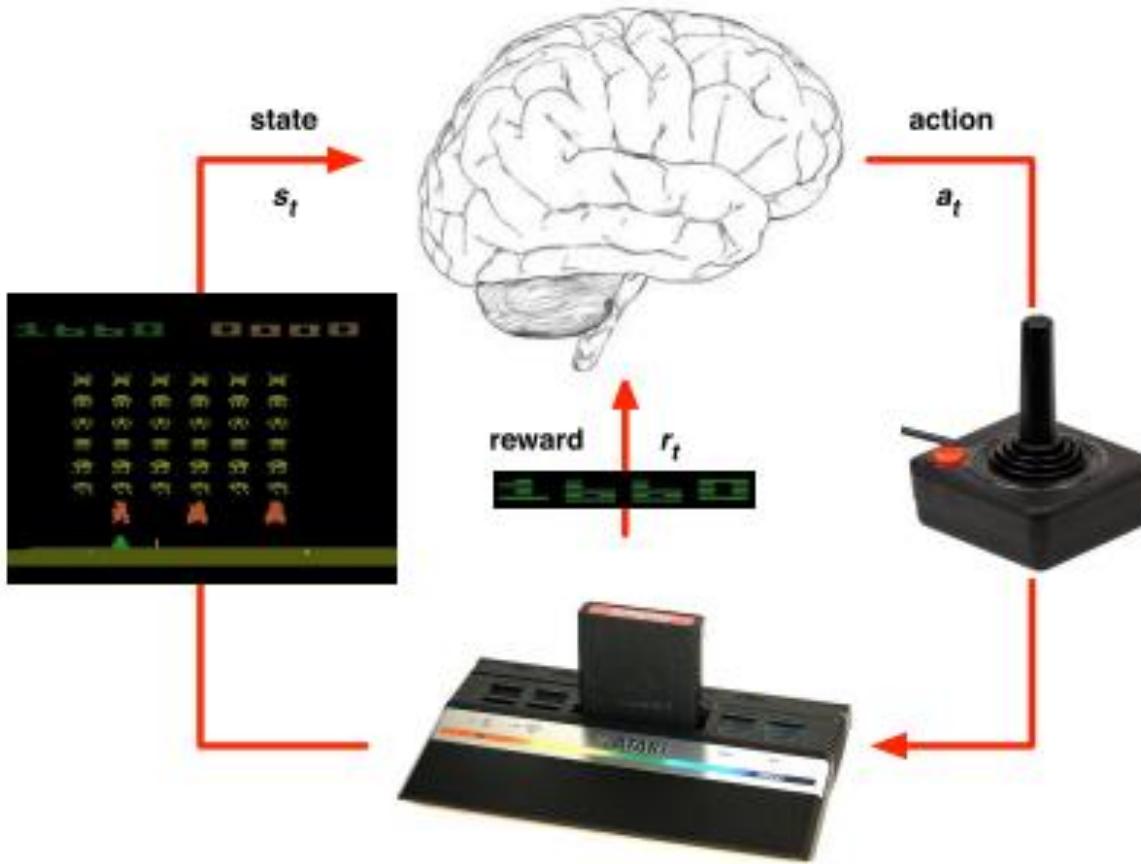
## Classic applications

- Pole-Balancing Robot [Michie, 1968 & others]
  - Inverted pendulum
  - Actions: left, right, pause [fixed vel.]
- TD-Gammon [Tesauro, 1995]
  - Learned to play backgammon from self-play
  - States: Simple & hi-level descriptions of positions
  - Rewards: Win=+100, Lose=-100, Others=0
  - Over 1 million games against itself [small % of state space]
  - Plays at human expert level





# Recent applications of RL (Deepmind)



Mnih et al. (2015)

The National logo and navigation bar (UAE, WORLD, BUSINESS, OPINION, ARTS&CULTURE, LIFESTYLE, SPORT) are visible at the top. The main headline reads: "Self-learning computer eclipses human ability at complex game Go". Below the headline, a subtext states: "Scientists announce that game-playing Artificial Intelligence computer mastered the ancient game of Go in three days, and say it could teach itself more than just playing games". The author is Robert Matthews, dated October 28, 2017. The article has 318 shares. On the right, there is an image of a Go board with stones and a person's hand, and a sidebar titled "EDITOR'S PICKS" with three news items: MONEY, SCIENCE, and THE AMERICAS.

Silver et al. (2017)



# Reinforcement Learning

## Part 4: Markov decision processes



# Andrey Markov

## Andrey Markov

- 14 June 1856 – 20 July 1922
- Russian mathematician
- Best known for work on theory of stochastic processes
- Subject of his research later became known as Markov chains and Markov decision processes





# Terminology: Deterministic/Stochastic

- Deterministic:
  - Next state of environment is uniquely determined by the current state and the executed action
- Stochastic
  - Inherent level of uncertainty
  - Executed action may result in different states
  - E.g. wheel slip, external forces, opponent, dice
  - Or uncertain due to laziness/ignorance



# Terminology: Markov Property

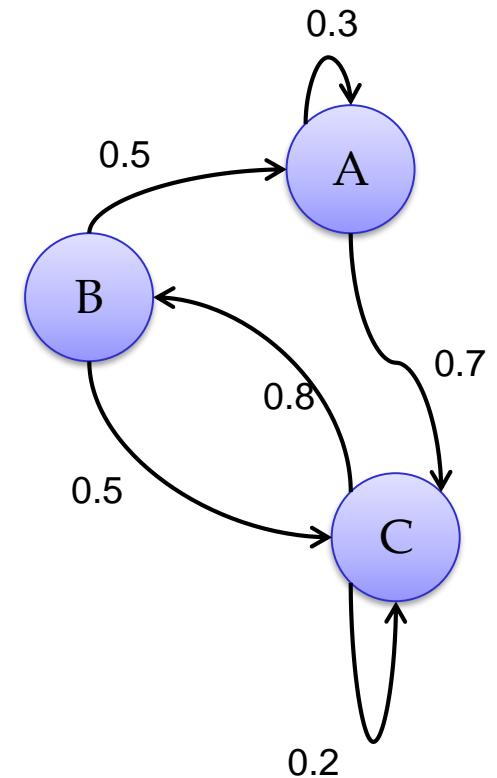
- Stochastic process has the **Markov Property** if its next state depends only upon its current state, not any events that preceded current state
  - "Memoryless"
  - i.e. conditional probability distribution of next state of the process depends only upon the present state
- Draughts example:
  - Current State given by current board layout
  - Next state unaffected by past states
  - ∴ all board states have the Markov property





# Terminology: Markov Chain

- A Markov chain is a stochastic process with the Markov property.
- A Markov chain can undergo transitions from one state to another, where the transition between states is determined by a particular probability distribution.





# Markov Decision Process

- MDP is an extension of Markov chains with 2 additions:
  - (1) choosing actions; (2) rewards
- Choice:
  - at every step, agent can choose what action they want to take
  - The result of actions is still stochastic
  - In state  $s$ , any available action  $a$  can be chosen
  - Result is a new random state  $s'$ , with probability depending on  $s$  and  $a$
- Reward:
  - The agent receives a reward based on the old state, action taken, and new state
  - The goal of the MDP is to maximise the expected discounted sum of rewards



# MDP Formal Definition

A Markov decision process is a tuple  $\langle S, A, T, R \rangle$  consisting of:

- A set of states  $S$
- A set of actions  $A$
- A transition function  $T(s, a, s')$ :
  - model of the agent's environment.  $T$  gives the probability of reaching state  $s'$  when selecting action  $a$  in state  $s$
- A reward function  $R(s, a, s')$ :
  - specifies the scalar reward received when action  $a$  is selected in state  $s$ , and the environment transitions to  $s'$
  - Rewards may also be based on reaching a state, e.g.  $R(s)$



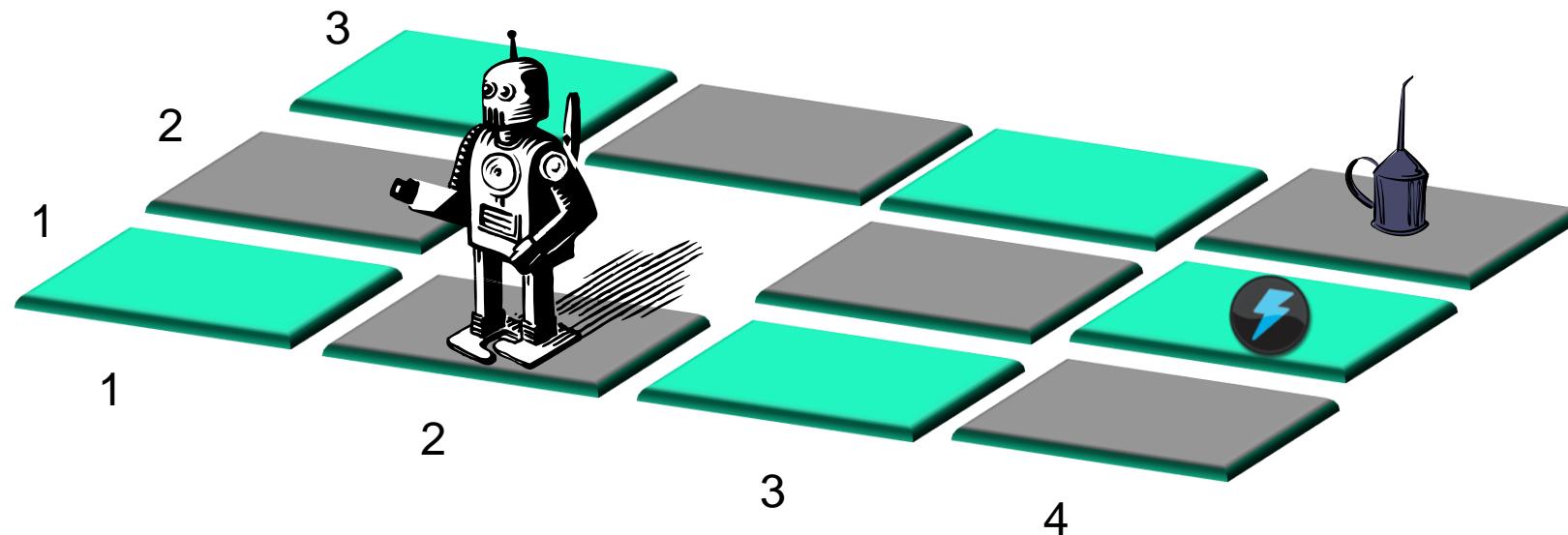
# Simple MDP example

4x3 environment with 11 states [and (2,2) unavailable]  $\rightarrow |S| = 11$

Reward +1 in (4,3); negative reward -1 in (4,2). These are terminal states.

Could include cost for each move e.g. -0.04; encourages agent to solve task in fewer timesteps.

4 actions: North, South, East, West  $\rightarrow |A| = 4$



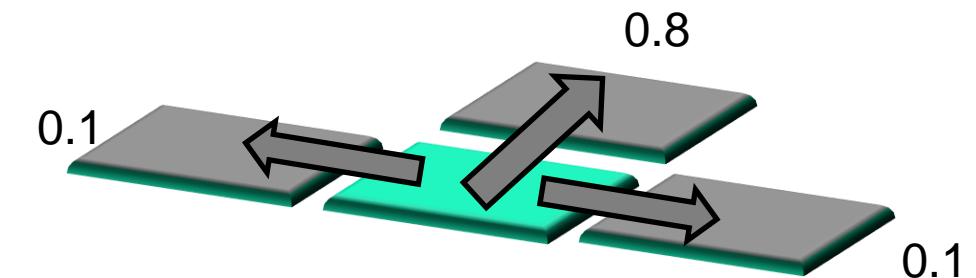
Example based on Russell & Norvig, Ch. 17  
Diagram by Dr Ted Scully.



# Simple MDP example

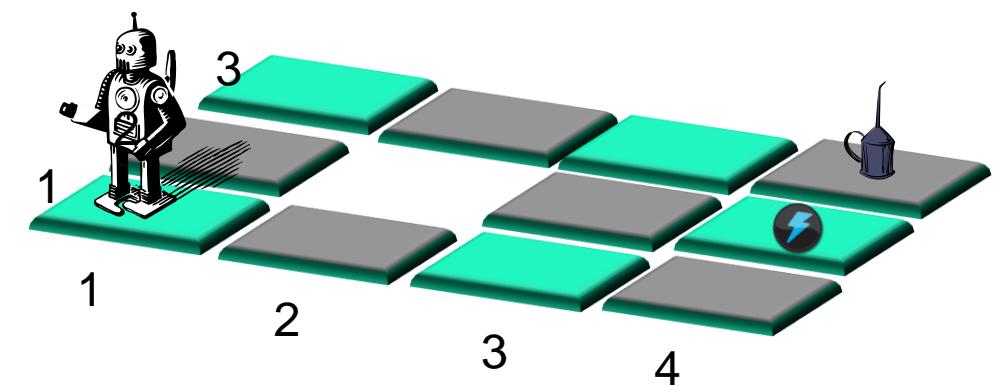
Actions are stochastic:

Take desired action with  $p=0.8$ ; move perpendicular with  $p=0.1$  in each direction (may result in no net movement)



If actions **were not** stochastic, sequence [Up, Up, Right, Right, Right] would lead from (1,1) to (4,3).

Because they **are**, this will work with probability  $(0.8)^5 = 0.32768$ , and probability  $(0.1)^4 \times 0.8$  that these actions end in alternate route to goal  
 $\Rightarrow$ total is 0.32776.

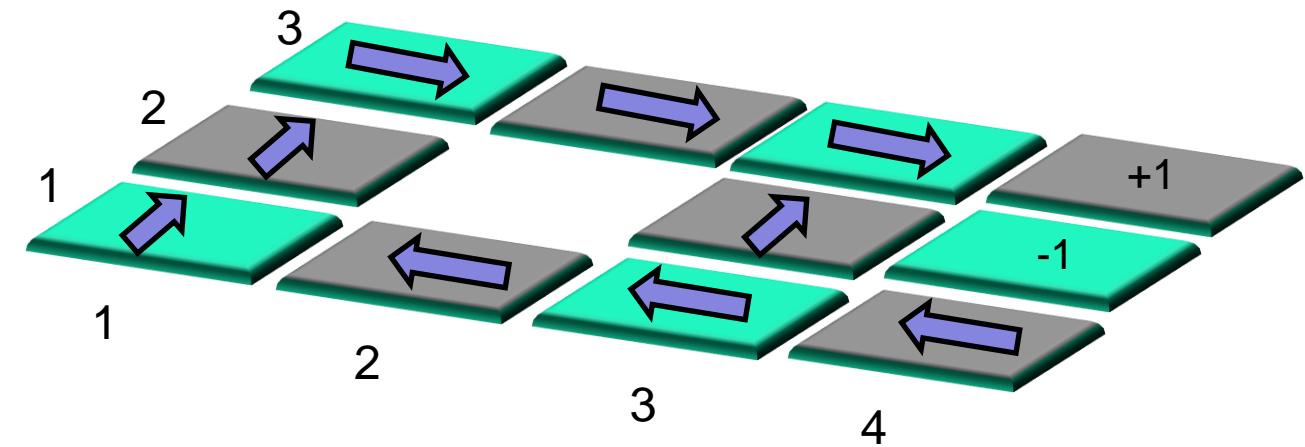




## Policy: Solution to the MDP

- ▶ A policy  $\pi$  represents a solution to an MDP problem; example shown below
- ▶ Defines what action to take in every state
- ▶ Optimal policy  $\pi^*$  is one that yields the highest expected discounted sum of rewards (the highest value to the agent)
- ▶  $V^\pi \rightarrow$  Value of a policy  $\pi$  over a time horizon  $h$  (can have  $h = \infty$ )
  - ▶ Add up all discounted rewards expected during policy execution

$$V^\pi = E \left[ \sum_{t=0}^{t=h} \gamma^t r_t \right]$$





# Bellman Equation

- The value of a state:
  - Immediate reward for that state plus the expected **discounted** value of the next state, assuming that the agent chooses the optimal action
  - Discount factor  $\gamma$  in range 0-1:  
if less than 1, values of future states carry less importance
- Calculated with the Bellman Equation
  - Note that the expected future value is calculated using  $T$ . As well as the value of each possible future state  $s'$ , we also need to consider the likelihood of reaching each  $s'$

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s, a, s') V(s')$$



# Value Iteration Algorithm

- Iterative algorithm that uses Bellman Equation to calculate the optimal policy for an MDP.
  - Simple implementation: see `GridWorld.java`

- Initialise reward matrix,  $R$ , for all states

- Initialise  $V'$  for all states to 0

- Repeat until convergence:

$$V = V'$$

Loop over all states:

- In each state  $s$ :

$V'(s) := R(s)$  if a terminal state

$V'(s) := \text{Bellman eqn}$  otherwise (computed using  $V(s)$ , not  $V'(s)$ )



Sample output:

0.8	0.9	0.9	1.0
0.8	0.0	0.7	-1.0
0.7	0.7	0.6	0.4

Best policy:

E	E	E	+
N	#	N	-
N	W	W	W



# Limitations of Value Iteration

- Need to know all rewards
- Need to know all transition probabilities
- What if we don't?

**Use Reinforcement Learning!**

- Explore an unknown environment
- Try different actions, see what rewards you get, keep going if you get knocked back to the start
- "Playing a new game whose rules you don't know..."



# MDPs in Reinforcement Learning

- Transition model
  - Model of probability of reaching state  $s'$  if you perform action  $a$  in state  $s$
- Markov Decision Process
  - Decision on what action to perform depends on current state only
  - Previous states irrelevant
- In RL context
  - Agent observes state  $s$ , takes action  $a$ : Gets from environment reward  $r$ , new state  $s'$
  - MDP => Transition model and  $r$  are functions of  $s, a$  only
- RL tasks are almost always assumed to be **Finite MDP**
  - Finite number of states and actions
  - State description sufficient that the context of previous states is not required





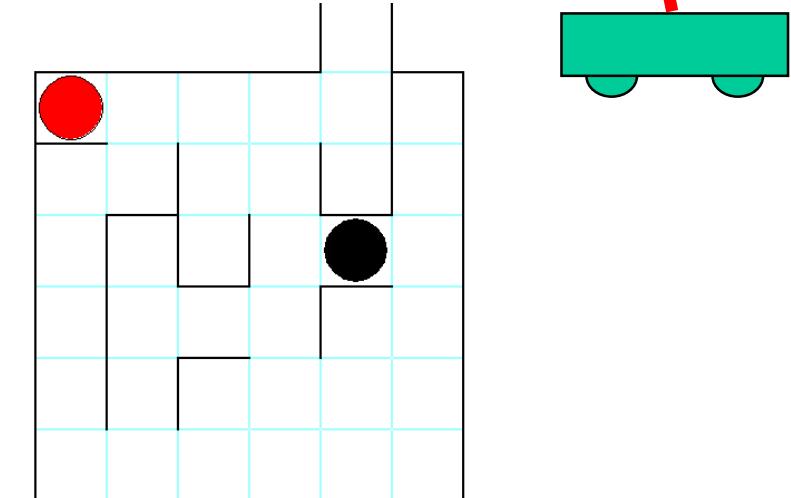
# Defining States, Actions and Rewards

- Defining Actions: usually easy
- Defining Rewards: Must reflect learning goals
  - E.g. shortest path through maze: penalise no. of steps
- Try changing rewards in GridWorld MDP solution:
  - $r = 0$ : take as many steps as possible to avoid -1 pit
  - $r = -0.04$ : minor penalty for each step – prefer shorter paths
    - $r = -0.2$ : life is bad!    $r = -1.7$ : life is horrible!
  - $r = 1$ : life is wonderful!



# State Descriptions

- Describing State:
  - Remember MDP assumption
  - Sufficient info so that previous states irrelevant
  - Note: In some applications can only approximate MDP
- What state information is needed for these environments to preserve MDP assumption?
  - Simple maze
  - Theseus & Minotaur maze
  - Pole-balancing robot





# RL Learning Task (1)



- Basic framework:
  - Agent observes state  $s_t$ , takes action  $a_t$ :  
Gets from environment reward  $r_t$ , new state  $s_{t+1}$
  - Transition function  $T(s_t, a_t) \rightarrow s_{t+1}$  and  
reward function  $R(s_t, a_t, s_{t+1}) \rightarrow r_t$  known by env., not agent
- Need to learn policy function
  - Selects next action given current state:  $p(s_t) \rightarrow a_t$



## RL Learning Task (2)



- Objective:
  - Maximise total **discounted** future reward
  - Total Value of a policy  $p$  starting from state  $s_t$ :  
$$V^p(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$
- $\gamma$  is discount factor:
  - More distant rewards are worth less now
  - Range [0,1]: should be close to 1 (e.g. 0.9)



# Reinforcement Learning

## Part 5: The Q-learning algorithm



# Q-Learning Algorithm (1)

- Q-Learning: popular RL approach
  - Learns an action-value function:  
 $Q(s,a)$  is value of performing action  $a$  in state  $s$
  - $Q(s,a)$  values estimated from experience
  - Does **not** need to learn transition function or reward fn => “**model-free**” as not attempting to model the environment
- Basic Idea:
  - Construct array, indexed by  $s$  and  $a$ , to hold Q-values
  - In any state, select action with highest Q-value  
(subject to exploration: later)
  - Tie-breaker: select one at random
  - Q-values initially 0: estimate  $Q(s_t, a_t)$  from immediate reward  $r_t$  and discounted estimated future reward of best action in next state,  $\gamma Q_{\max}(s_{t+1}, a_{t+1})$



## Q-Learning Algorithm (2)

- Q-learning update: (temporal difference update rule)

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- max: selects action with highest expected value in  $s_{t+1}$
- $\gamma$  = discount factor as before
- $\alpha$  = learning rate: range [0,1]

- Learning Rate:

- instead of completely discarding old value of  $Q(s_t, a_t)$ , adjust it proportional to  $\alpha$ :
  - Reward/transition might be stochastic
  - $Q(s_{t+1}, a_{t+1})$  is **probably** inaccurate
- Higher  $\alpha \Rightarrow$  old  $Q(s_t, a_t)$  value less important
- Often start with high  $\alpha$  and reduce with experience of specific state/action pair:  
$$\alpha = 1/(1 + \text{visits}(s, a))$$
- Deterministic case: can use  $\alpha = 1$



# Q-Learning Algorithm (3)

- Initial stages:
  - All Q-values equal (zero): random actions tried
  - Q-values only updated when first reward (+/-) found
  - Next time adjacent to that state, Q-value affected
- Exploration:
  - Shouldn't always select action with highest Q-value
    - Might have found a sub-optimal solution, giving low reward  $> 0$
  - As with much of AI/ML (and real life!), there is a trade-off between exploiting knowledge, and exploring to find new (possibly better) options
  - Need mechanism to try new actions. One solution:  $\epsilon$ -greedy exploration
    - Choose  $\epsilon$  in range  $[0,1]$ : want low value, e.g. 0.01
    - Each time, pick number  $n$  in range  $[0,1]$
    - If  $n \leq \epsilon$ , choose random action instead of one with highest Q-value



## Q-Learning Algorithm (4)

- The full Q-Learning procedure:
    - For repeated episodes [games] in an environment
- 1  $\forall s, \forall a: Q(s,a) = 0$
  - 2 Repeat (for each episode):
    - 3 Initialise  $s$  to starting state
    - 4 Repeat (for each step of episode):
      - 5  $a = \arg \max_a Q(s, a)$
      - 6 With probability  $\varepsilon$ :  $a = \text{random action}$
      - 7 Take action  $a$ , observe reward  $r$  and next state  $s'$
      - 8 
$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$$
      - 9  $s = s'$
    - 10 Until  $s$  is terminal
    - 11 Until stable solution found



# Generalisation in RL

- Q-values table can be very large!
  - Large number of states [e.g. board game] and actions
  - Need to visit all state/actions multiple times to get accurate estimates of Q-values
  - Would be nice to generalise from experience of similar states
- Solution:
  - Replace table with a lookup function implemented using a function approximator
  - Most commonly, feed-forward neural network
  - Used in TD-Gammon and other applications to good effect.
  - Recent Deep RL works use multi-layer ANNs for generalisation



# RL and the Temporal Credit Assignment Problem

- When an agent acts in an environment, and rewards are few and far between, it must take a lot of steps before gaining reward (e.g. at maze exit)
- E.g. just before maze exit, does the final step deserve all the reward?
- RL in general seeks to solve the credit assignment problem by iteratively propagating the influence of delayed reinforcements to all states and actions that led to that reinforcement



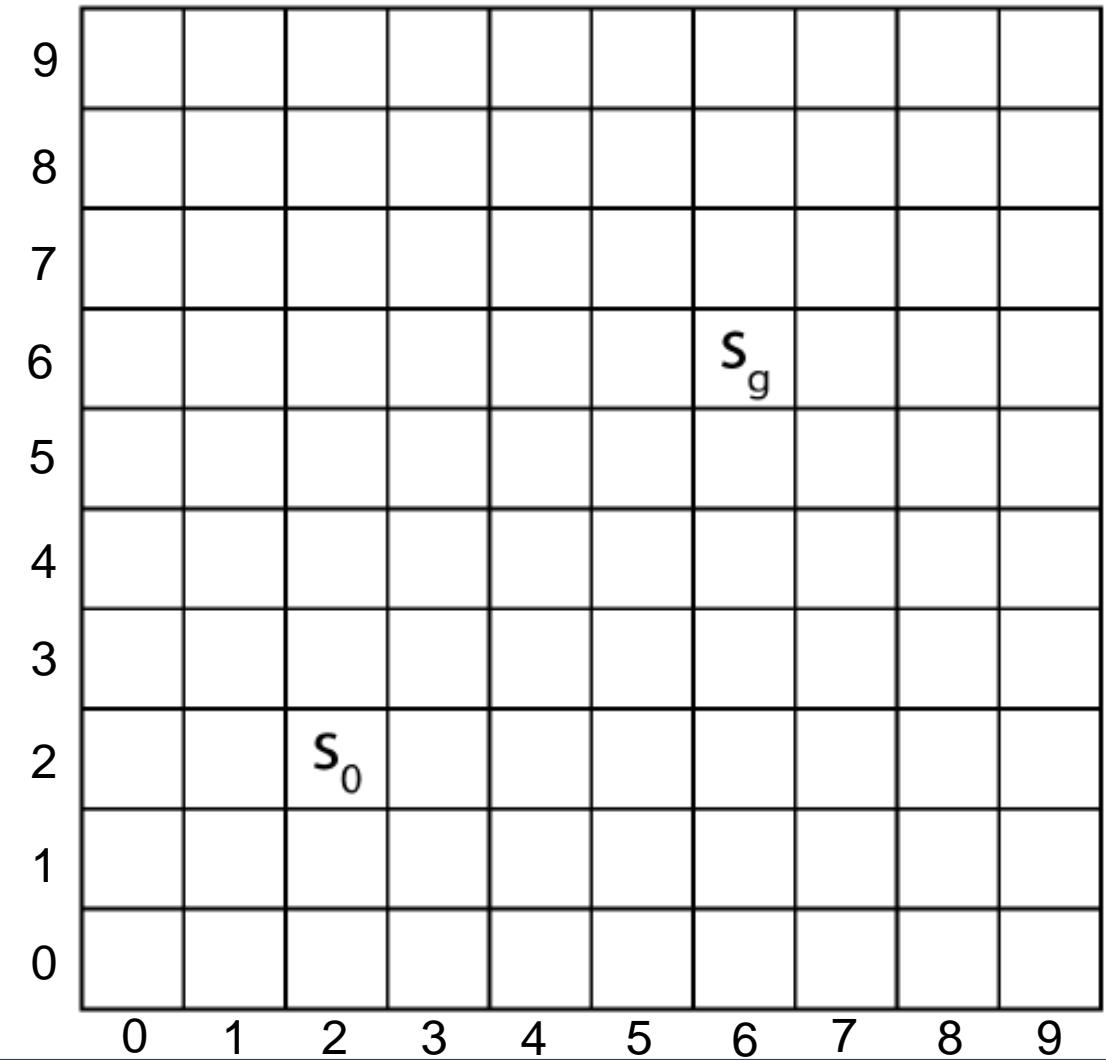
# Reinforcement Learning

## Part 6: Q-learning worked example



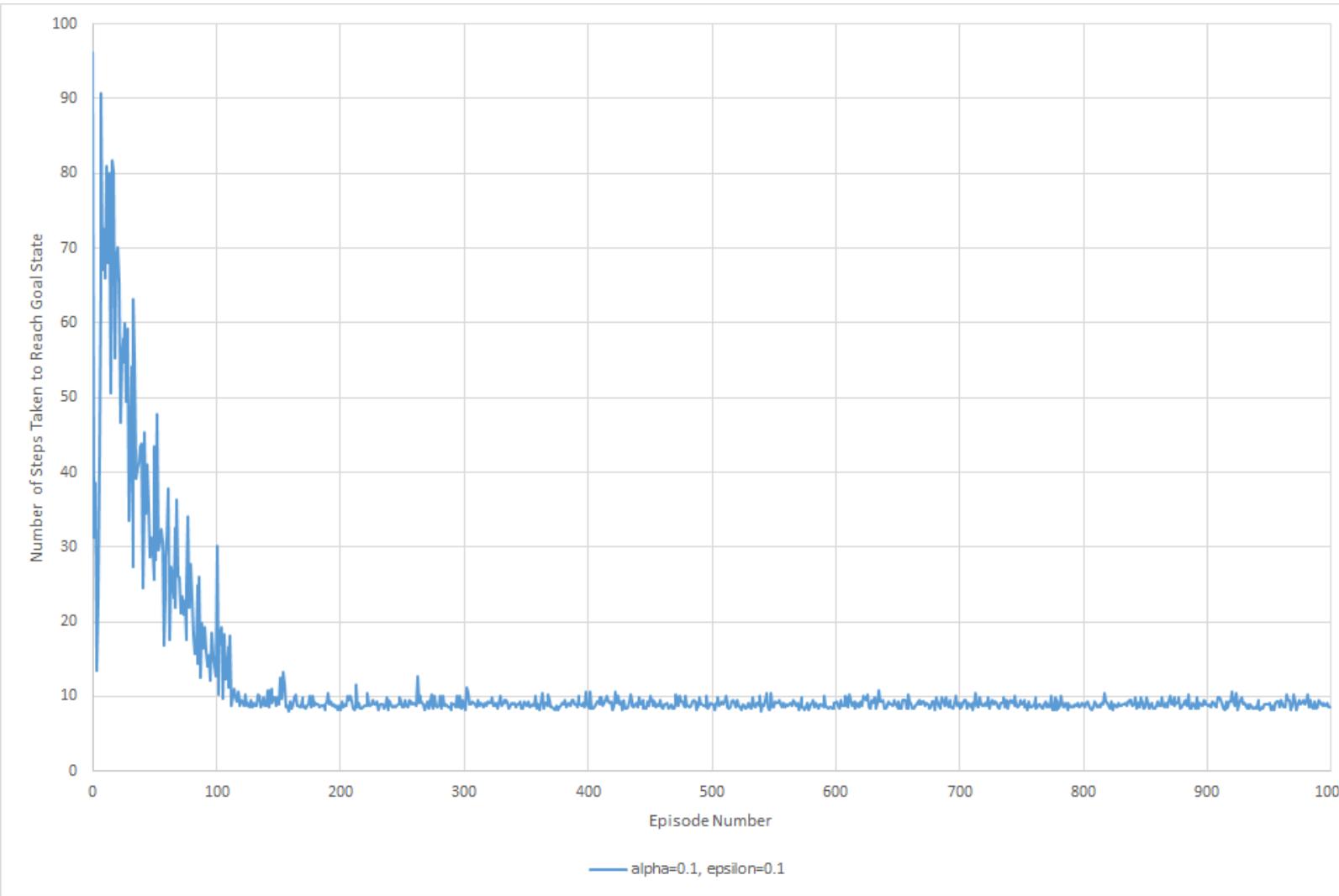
# Worked example: Q-learning in Gridworld

- Gridworld: a single agent starts in an initial position (state)  $s_0$ , and must reach a goal state  $s_g$
- Actions available are to move North, East, South or West
- State-action value estimates (Q values) updated at each timestep
- Episodic – agent learns by trial and error over several episodes
- Sparse reward function – agent is only rewarded when it reaches the goal state
- Rewards: +10 for reaching goal, -1 for all other turns





# Sample learning curve for this problem



Sample code:  
[Gridworld\\_Qlearning.zip](#)

Worksheet:  
[Gridworld\\_Qlearning.pdf](#)

This sample graph shows the steps to goal averaged over 10 runs, with:

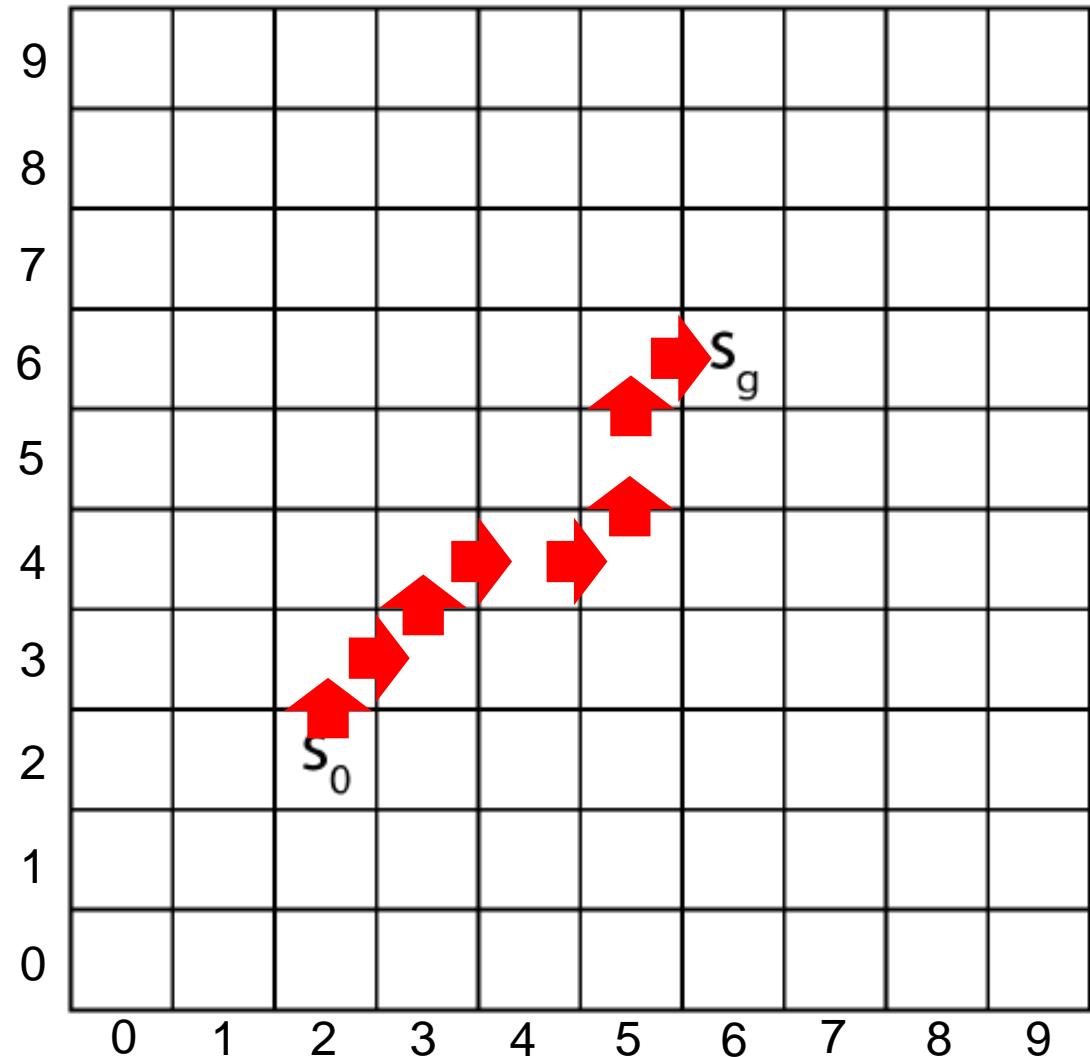
- alpha = 0.1
- epsilon = 0.1
- gamma = 1.0

Learning curve flattens out  
=> Convergence!



# Q values for a sample optimal path

State (x,y)	North	East	South	West
(0,0)	-2.373	-2.403	-2.397	-2.392
...	...	...	...	...
(2,2)	3.000	2.087	-0.913	-0.600
(2,3)	2.592	4.000	1.524	0.928
...				
(3,3)	5.000	4.323	1.762	2.603
(3,4)	2.701	6.000	3.561	2.851
...				
(4,4)	6.263	7.000	4.506	4.096
...				
(5,4)	8.000	4.219	3.180	5.801
(5,5)	9.000	7.008	6.704	5.840
(5,6)	2.318	10.000	7.511	7.370
...				
(9,9)	-0.200	-0.200	-0.200	-0.200

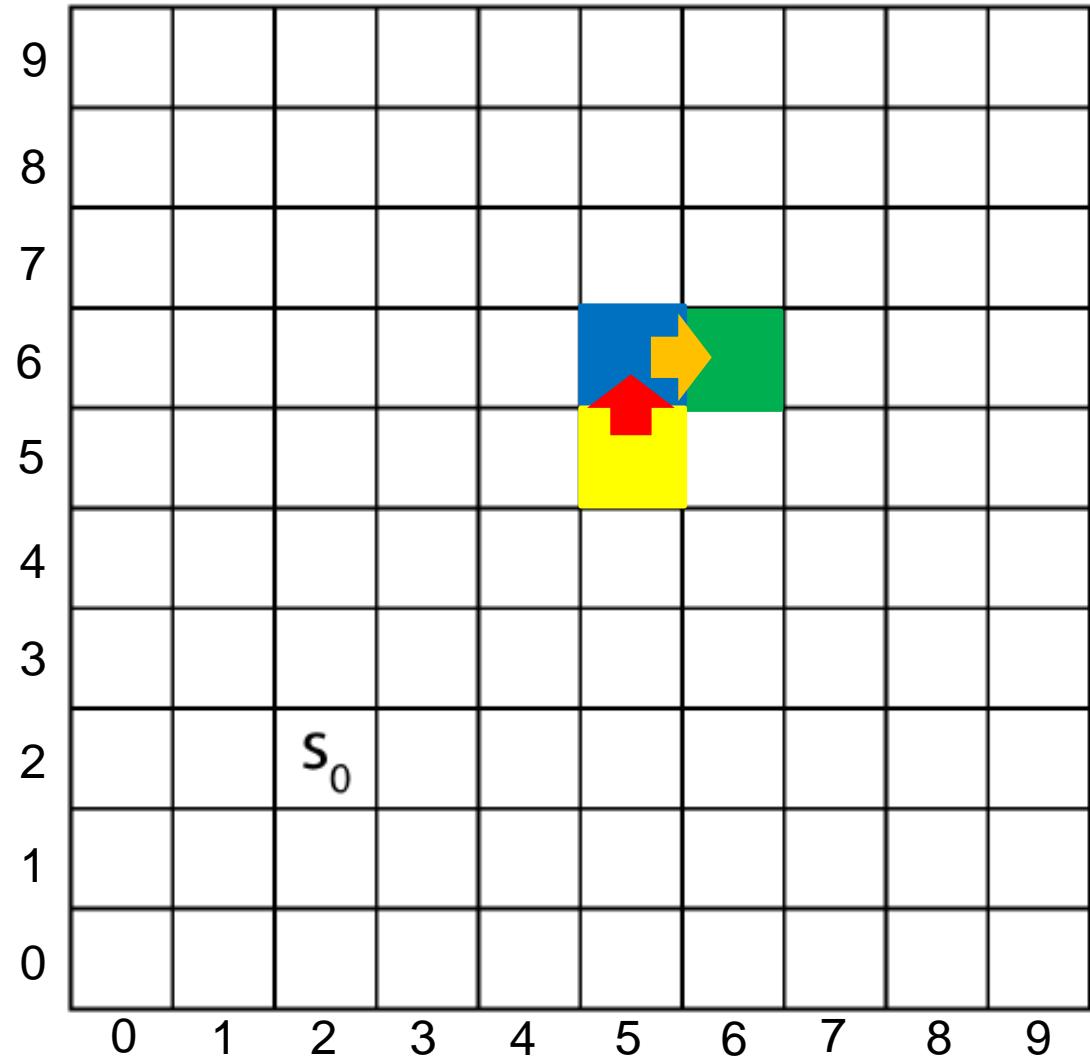




# Example Q-learning process (single timestep)

State (x,y)	North	East	South	West
(5,5)	8.250	7.008	6.704	5.840
(5,6)	2.318	10.000	7.511	7.370

- Initial state  $s: (5,5)$
- Max valued action  $a: \text{North}$
- Next state  $s': (5,6)$
- Reward received  $r: -1$
- Calculate new Q value  $Q(s,a)$  for  $Q( (5,5) , \text{North} )$  using max valued action in next state (next slide)



$$\text{NewQ} = 8.325$$

$$\text{OldQ} = 8.25$$

Reward Received = -1

Max Q value in next state = 10

$$\text{OldQ} = 8.25$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate = 0.1      Discount factor = 1.0

State (x,y)	North	East	South	West
(5,5)	8.250	7.008	6.704	5.840
(5,6)	2.318	10.000	7.511	7.370

Q table at beginning of timestep

State (x,y)	North	East	South	West
(5,5)	8.325	7.008	6.704	5.840
(5,6)	2.318	10.000	7.511	7.370

Q table at end of timestep



## Final remarks

- RL is an area of active research; not as mature as other ML paradigms
- Problems with using RL in the real world:
  - e.g. trial and error learning using expensive equipment/robots can be expensive and dangerous). Could train agent in simulation first, then refine in real hardware
  - Ethical considerations, validating behaviour/compliance with laws and standards
- Current research topics include:
  - Dealing with sparse reward functions, curse of dimensionality, sample complexity
  - Multi-Agent Reinforcement Learning (MARL)
  - Integrating domain knowledge (e.g. advice from a human expert)
  - Safe and explainable decision making



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 7: Evaluating Classifier Performance; Practical Advice; Some Machine Learning Tools

Prof. Michael Madden  
Chair of Computer Science  
Head of Machine Learning & Data Mining Group  
National University of Ireland Galway



# Learning Objectives

After successfully completing this, you will be able to ...

- Discuss distinctions between training, testing & validation data sets
- Perform cross-validation
- Describe and compute performance measures, learning curves and ROC graphs
- Select and perform appropriate statistical tests
- Give guidance on compiling a data set and selecting & comparing methods for a task
- Discuss related issues, including being able to decide how many test data sets are enough
- Discuss and select machine learning tools



# Why Measure Performance?

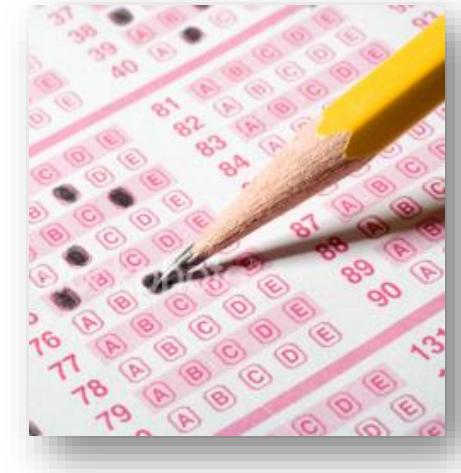
How do we determine if hypothesis is sufficiently good?

- Need ways to measure performance
- Need to define acceptable performance for task

We always want to know:

How well will this model work on future data?

- Since we can't look into the future, we have to evaluate the model on data already available to us
- **IMPORTANT:**  
All forms of evaluation assume that data available to us is representative of future data on which we will apply models



No one ML algorithm is always best

- Need to be able to compare algorithms/variations/tweaks
- Wolpert's "No Free Lunch" theorem



# Basic Performance Measures - Classification

For Classification:

- Basic measure is **Classification Accuracy** ( $1 - \text{Error Rate}$ )
- Proportion of test cases classified correctly
- Multiple choice test: some may be right by chance
- Can easily estimate lower bound on performance:  
**How? (See next slide)**



# Basic Performance Measures - Classification

Test Cases				
Outlook	Temp	Humidity	Windy	
sunny	hot	high	false	
overcast	hot	high	false	
rainy	mild	high	false	
overcast	cool	normal	true	
sunny	mild	high	false	
rainy	mild	normal	false	
rainy	mild	normal	false	

4/7 predictions correct = accuracy of 57.14% = error rate of 42.86%

In training dataset, 9 of 14 are “yes” class

=> guessing “yes” every time would give accuracy of 64.3%



# Basic Performance Measures – Regression

---

For Regression:

- Root Mean Squared Error (error = Actual - Prediction)
- Square each error, average over test cases, get square root
- Also Maximum Absolute Error, Mean Absolute Error, ...



# Basic Performance Measures - Regression

Test Cases				
Size (m <sup>2</sup> )	# Beds	# Floors	Age (yrs)	
195	5	1	40	
130	3	2	35	
140	3	2	26	
80	2	1	30	
180	5	2	38	



# Confusion Matrix

More informative than Accuracy for multi-class problems

- If some errors are more serious than others, can multiply it by a **mis-classification cost matrix**
- If Category A = Positive ,B = Negative, can see **true positives, false negatives**, etc.

		Predicted	
		Category A	Category B
Actual	Category A	$w$	$x$
	Category B	$y$	$z$



# Confusion Matrix - Example

		Actual		Actual Class	Classifier Predicted	Classifier Performance
		Positive				
Predicted	Positive	3	1	no	no	✓
	Negative	2	1	yes	yes	✓
				yes	no	✗
				no	yes	✓
				yes	no	✗
				yes	yes	✓



# Confusion Matrix: Other Examples

		Predicted		
		Goat	Horse	Sheep
Actual	Goat	10	0	6
	Horse	0	15	1
	Sheep	5	1	9

Predicted:				Nursery
NOT_RECOM	RECOMMEND	VERY_RECOM	PRIORITY	Actual:
1464	0	0	0	NOT_RECOM
0	16	0	0	RECOMMEND
0	91	1264	50	VERY_RECOM
0	0	143	1292	PRIORITY



# Training, Testing and Validation (1)

What data do we use for testing?

- Performance on training data is not sufficient:  
**Why not?**

If we have **unlimited** data ...

- Randomly sample enough to construct hypothesis
  - too little: bad hypothesis
  - too much: slow!
  - how much? use a **Learning Curve [coming up soon]**
- Randomly sample more to test hypothesis
- If you change hypothesis, re-sample more data

In practice, have **limited** amount of labelled data...

- Need to divide into statistically identical subsets



# Training, Testing and Validation (2)

If we have large amount of labelled data, but not unlimited:

- Randomly sample a **testing set** (a.k.a. holdout set) and **hide it**
- Divide remainder into **training sets** and **validation sets**

Using these datasets:

- **Training data:** used to construct hypotheses
- **Validation data:** used to evaluate/compare hypotheses,  
while selecting algorithms and parameter settings
- When you're happy you've found the best algorithm and parameter  
settings you can, construct final hypothesis using **training & validation  
data together**
- **Finally,** unlock the drawer with the **testing data:**  
evaluate objectively using it; publish/communicate these results



# Training, Testing and Validation (3)

To be confident train/validation sets are representative

- Often need to re-divide multiple times and average results
- Check standard deviation of results as well

A pitfall of random sub-sampling

- All the train/validation/test sets come from the same source and are not truly independent of each other
- Implications for statistical tests: see later



# Learning Curves

Repeat for various values of X from 0% to 100%:

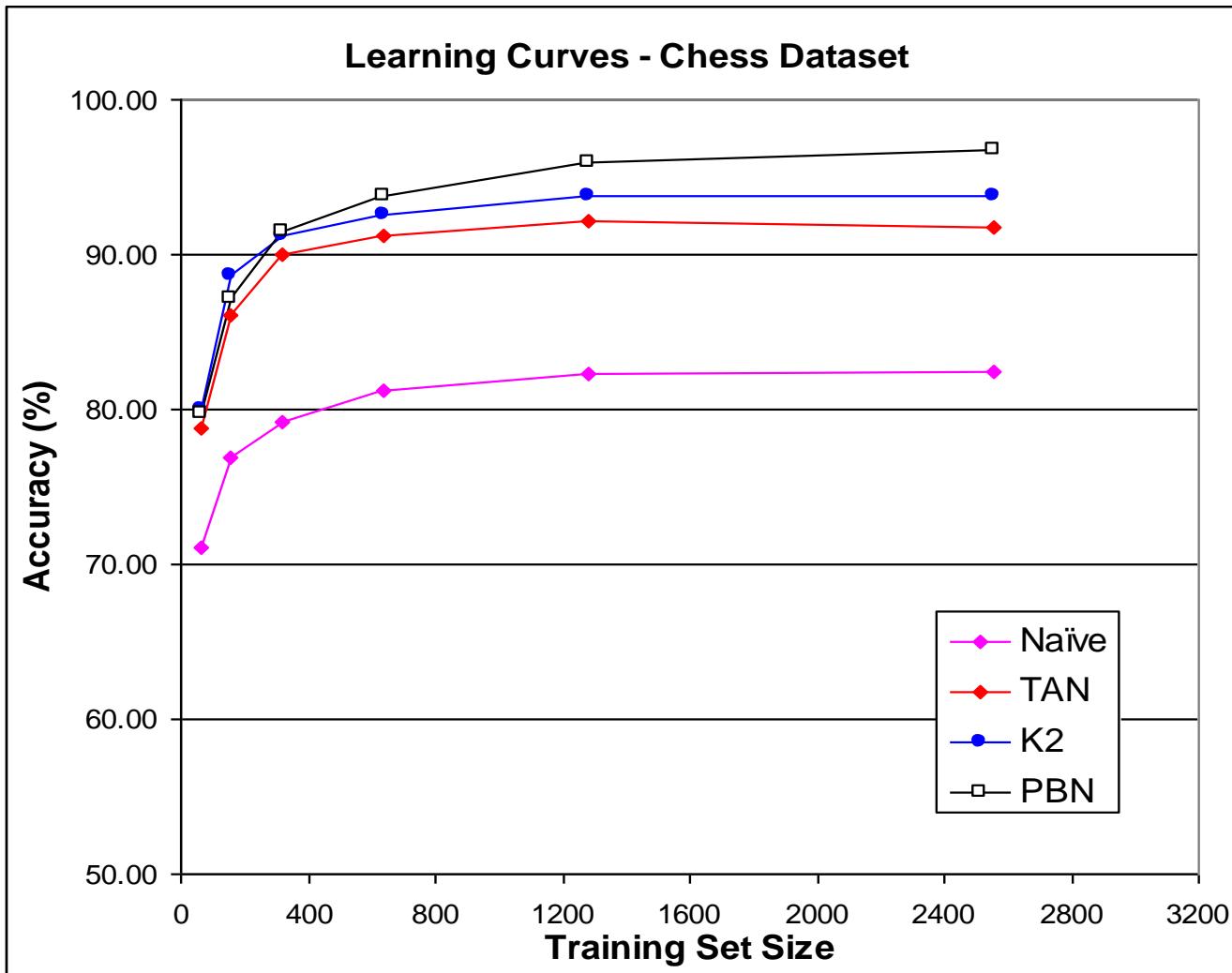
- Repeat several times:
  - Train on X% of data (randomly sampled)
  - Test on the rest
- Average the results at X%

Popular because...

- Useful for comparing techniques
- Insight into how much data is sufficient for a technique
- Indicative of error in the limit



# Learning Curves





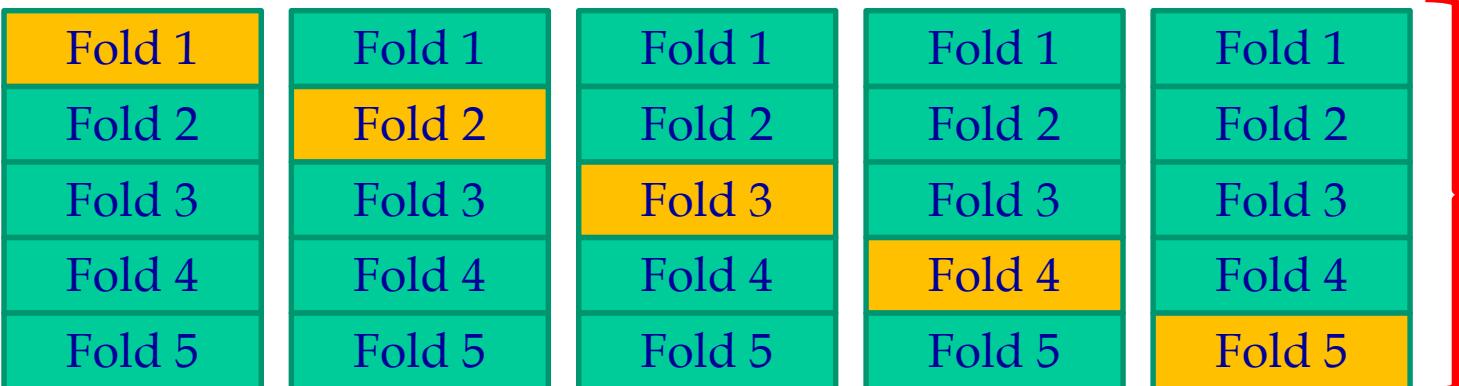
# Cross-Validation (1)

Useful for training/validation if data is somewhat scarce ...

- Divide into  $N$  'folds': subsets of near-equal size
- Often use  $N = 10$ : want fair no. of folds, with  $>30$  examples each
- **Stratified**: similar distribution of data in each fold

Repeat for each fold  $i$  from 1 to  $N$ :

- keep fold  $i$  for testing
- construct hypothesis using the rest for training
- test on fold  $i$



Test

Train

All data used  
 $N-1$  times for  
training

All data used  
one time for  
testing



# Cross-Validation (2)

## Result:

- Have constructed (and discarded)  $N$  hypotheses, using all data  $N - 1$  times for training
- Have used **all data for testing** once, even though we have never trained and tested on the same data!
- Trade-off: computation vs. data

NOTE: **all** training data used to build a **final** hypothesis

- The cross-validation result constitutes an estimate of its likely future performance
- For 10-CV, 90% overlap between final hypothesis and those in folds, and all training cases have operated as independent test cases
- Still should test this final hypothesis on a **held-out test set**



# Cross-Validation: Variations

## Repeated N-fold Cross-Validation:

- Shuffle the data, do cross-validation, shuffle again
- Ensure you have sufficient data that shuffles are meaningful

## Leave-One-Out Cross-Validation

- No. of folds = no. of samples
- Not as good statistically, but used with datasets of limited size



## Cross-Validation: Some Caveats

Repeated cross-validation can take a lot of CPU time:

- E.g. do 5 runs of 10-fold cross-validation:
  - Each fold involves building a full new model on nearly all data (90%)
  - Repeated 5 times with the data shuffled each time
- Result: takes **50 times as long** as if we had done a single train/validation split
- **Of course, even if training times are slow, performing analyses with the new model will still be fast**

Testing on a validation set is quicker, but ...

- Must ensure that validation set is sampled from same population as the training data
- If you don't have very much data, and you use some for a validation set, you will have less for use in model building



# ROC Curves

Particularly useful for comparing probabilistic classifiers

- Classifier outputs relative probability of each class
- Pick the one with highest probability?
- **What about if classification is whether/not to do cancer biopsy?**

ROC Curve: perform comparisons independent of misclassification costs

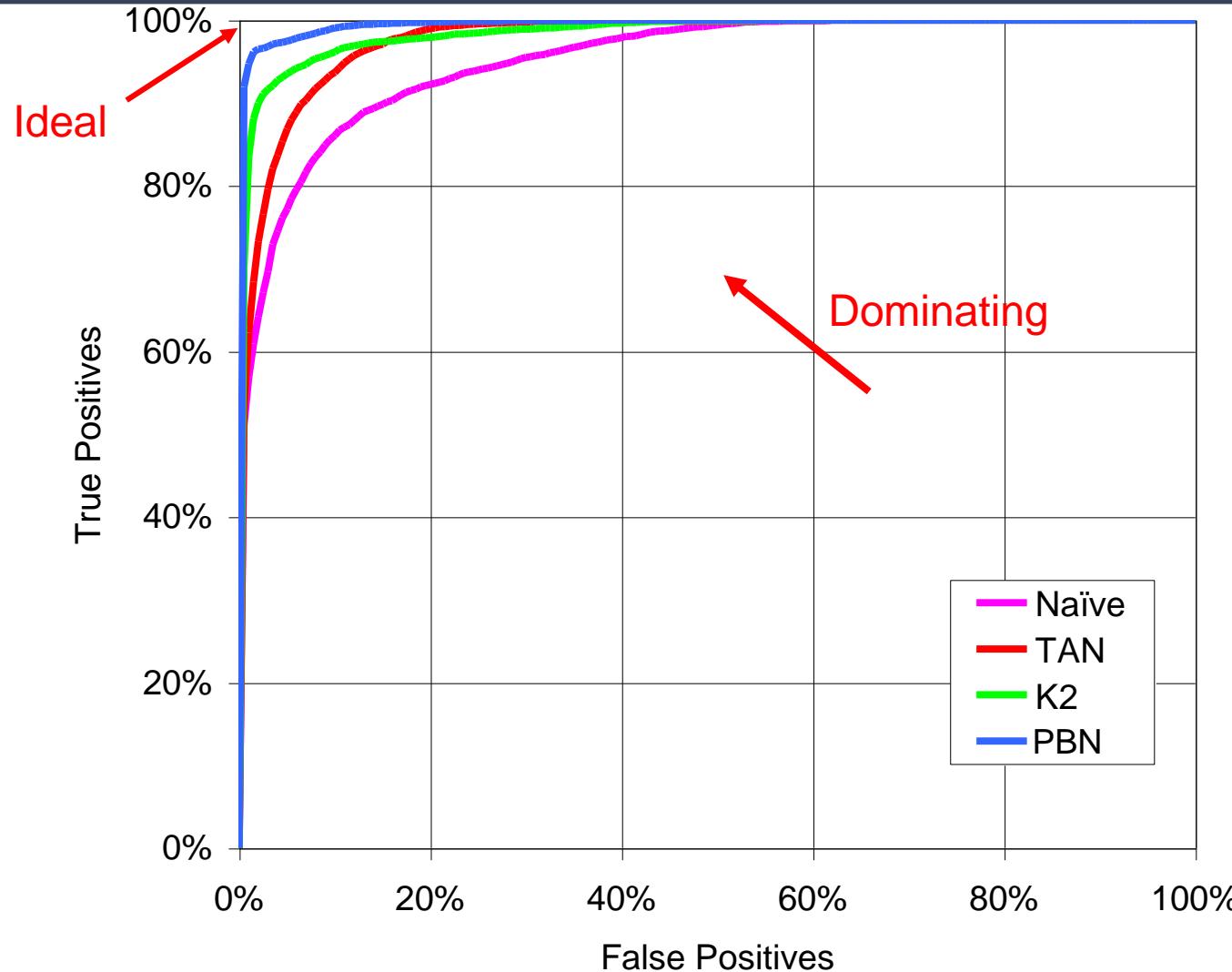
- Family of classifiers: one for every possible probability threshold
- Suitable for two-class problems

Can also calculate **Area Under ROC** (AUROC)

- Single-number measure of performance under all possible decision thresholds (multi-class definitions exist)
- Alternative to RMSE



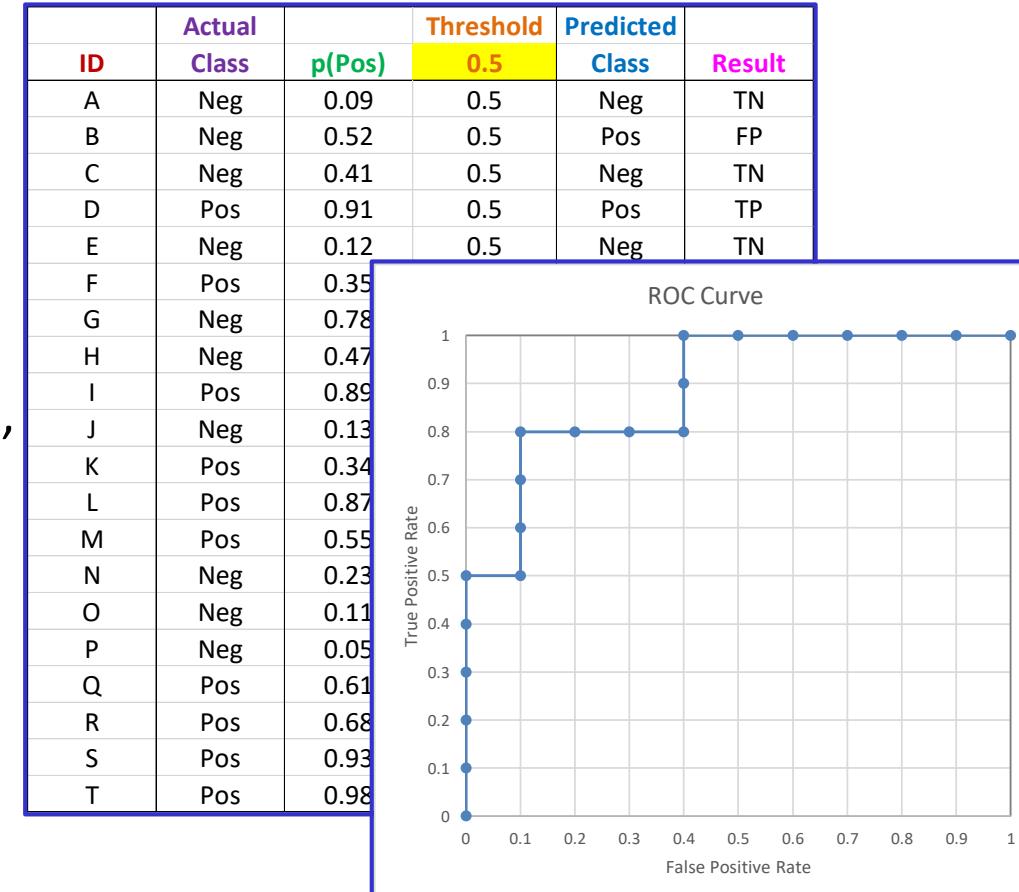
# ROC Curve Example





# Tutorial on Computing Confusion Matrices & ROC Curves

- See [ROC-Curve-Tutorial-MMadden.xlsx](#)
- A single ROC curve can have a stepped shape
  - With large amounts of data, steps become small
  - If we average over multiple ROC curves from random subsets of data, steps tend to go away
  - We can also join the curves peak-to-peak: equivalent to replacing points that are dominated by weighted averages of points that dominate them





# Comparing two Classifiers/Algorithms (1)

If A has lower error (RMSE, AUROC, ...) than B

- it might be better
- it might have happened by chance
- How can we tell if difference is statistically significant?

Paired t-test

- To decide whether average results of A and B are same
- Null Hypothesis: no statistical difference between results
- Paired: use same training/testing data for A and B  
[random seed]
- Calculate t-statistic:
- Pick confidence level
- Look up t-critical (DOF=k-1)
- See if t-statistic > t-critical

$$t = \frac{\bar{d}}{\sqrt{\sigma_d^2/k}}$$

mean of difference between pairs of measurements  
variance  
no. of measurements



# Comparing two Classifiers/Algorithms (2)

A tutorial on how to do this in Excel: see **T-Test.xls**.

Caveat:

- Paired t-test assumes each individual pair of results is independent of each other pair
- If we did random sub-sampling, this is not true
- High risk of identifying a difference where none exists:  
**Type I Error** [Dietterich 1997]

Modified version of t-statistic to account for this:

- **Corrected Resampled T-Test** [Nadio & Bengio, 2003]
- Also applicable to cross-validation runs



# Comparing two Hypotheses/Algorithms (3)

Should include significance tests when presenting results of accuracy comparisons ...

- Here, figures are accuracy +/- standard deviation
- Statistically best results (in each row) highlighted

	Naïve	TAN	K2
<b>Chess</b>	87.63 $\pm$ 1.61	91.68 $\pm$ 1.09	94.03 $\pm$ 0.87
<b>Breast Cancer</b>	<b>97.81 <math>\pm</math> 0.51</b>	<b>97.47 <math>\pm</math> 0.68</b>	<b>97.17 <math>\pm</math> 1.05</b>
<b>LED-24</b>	<b>73.28 <math>\pm</math> 0.70</b>	<b>73.18 <math>\pm</math> 0.63</b>	<b>73.14 <math>\pm</math> 0.73</b>
<b>DNA Splice</b>	94.80 $\pm$ 0.44	94.75 $\pm$ 0.42	<b>96.22 <math>\pm</math> 0.64</b>
<b>Lymphography</b>	83.60 $\pm$ 9.82	<b>85.47 <math>\pm</math> 9.49</b>	81.47 $\pm$ 10.37
<b>Nursery</b>	90.48 $\pm$ 0.41	<b>94.16 <math>\pm</math> 0.33</b>	92.63 $\pm$ 0.67
<b>SPECT</b>	71.70 $\pm$ 6.56	<b>81.25 <math>\pm</math> 4.78</b>	<b>80.19 <math>\pm</math> 4.66</b>
<b>TicTacToe</b>	70.69 $\pm$ 1.94	75.08 $\pm$ 1.86	74.04 $\pm$ 3.51



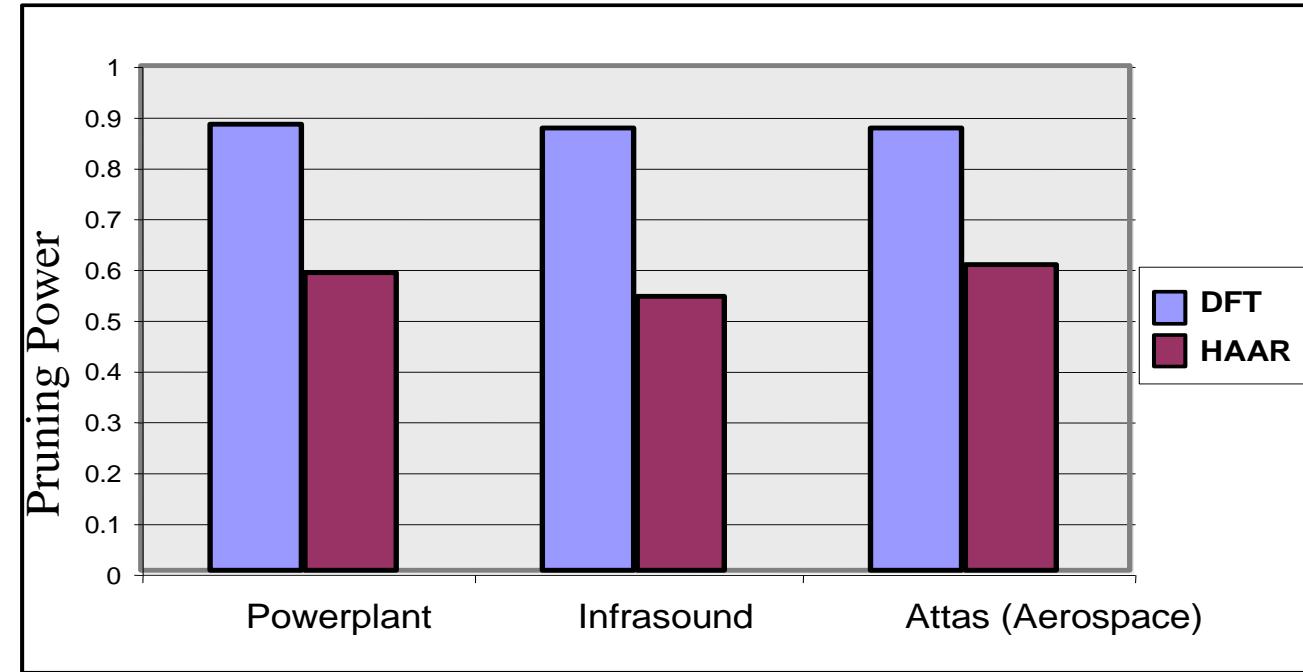
# Comparing Algorithms: How many Test Data Sets are Enough?

Essential point:

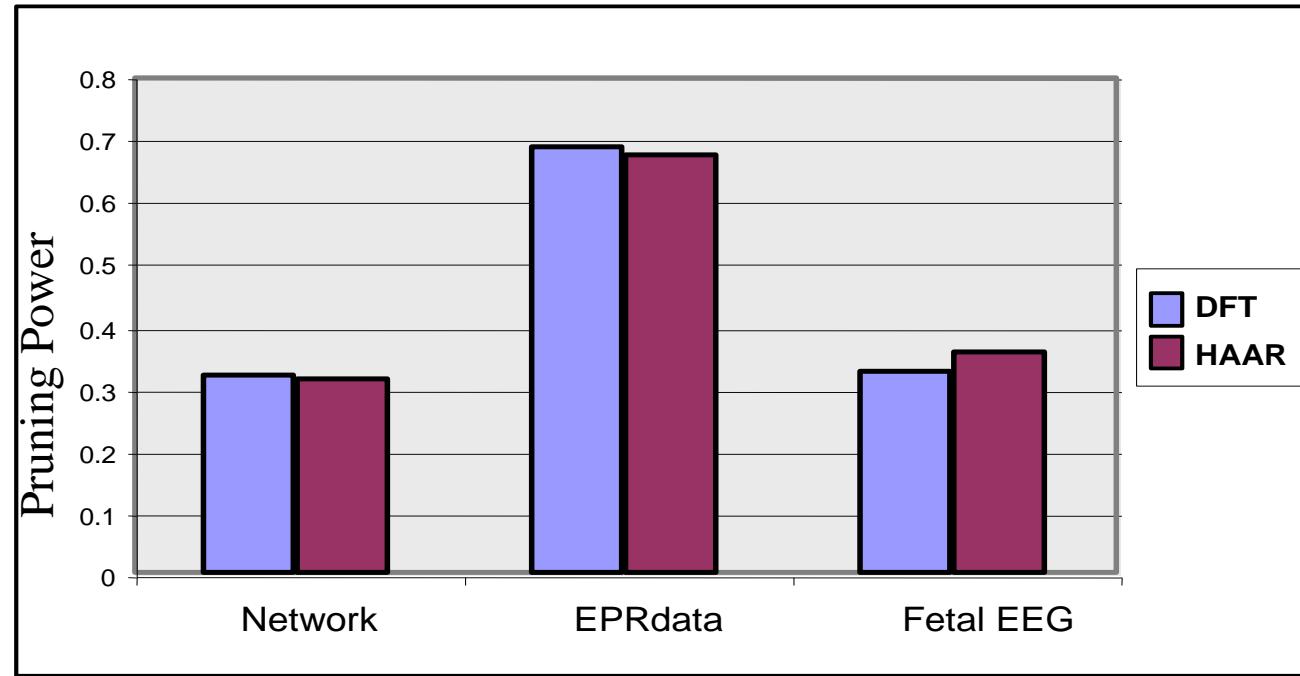
- If we want to determine whether one technique is on average better than another, we typically use several datasets to test them
- If you only use a few different datasets, you can ‘show’ whatever you want!

Reference:

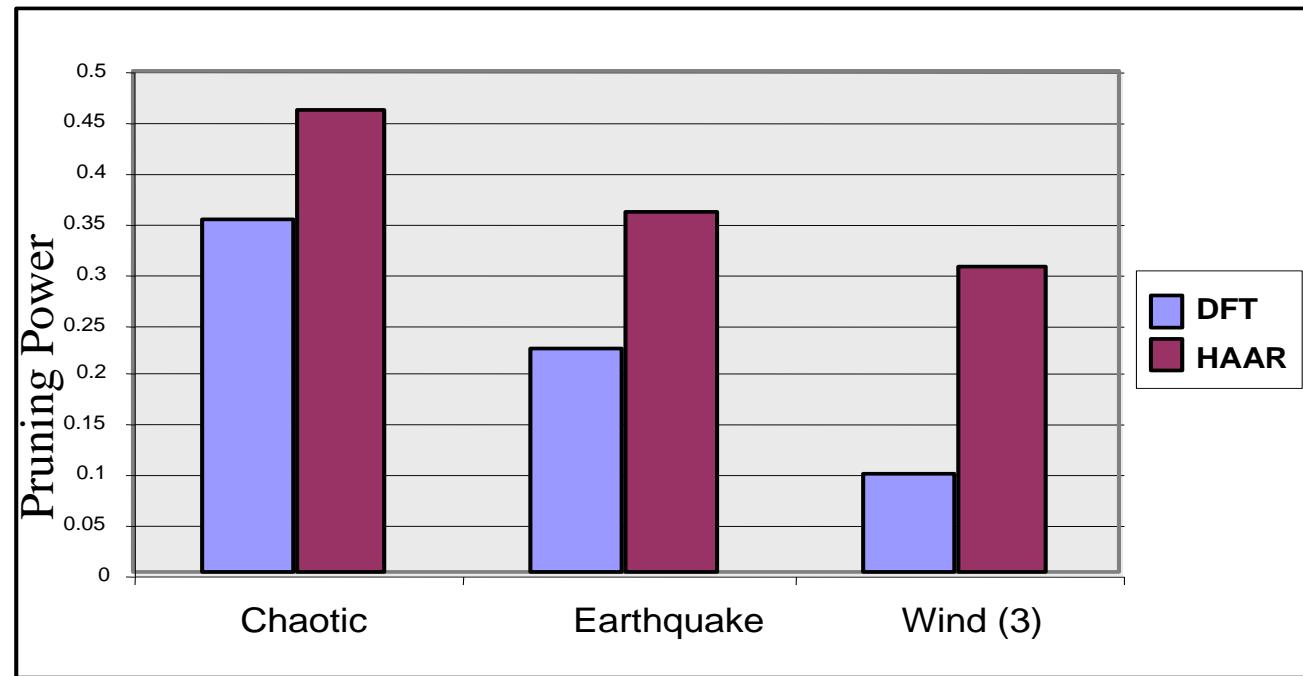
- “On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration”, by Keogh and Kasetty, *Data Mining and Knowledge Discovery*, Oct. 2003.
- Paper concerned with time series techniques, but point is generally applicable
- More details: see tutorial from 14<sup>th</sup> ECML conference, at  
<http://www.cs.ucr.edu/~eamonn/>



These tests show that DFT is much better than HAAR



These tests show that DFT and HAAR perform similarly



These tests show that DFT is much worse than HAAR



# How to Improve Performance of ML Models

There are several strategies for generating improved ML models with lower error rates

In order of increasing difficulty ...

1. Adjust the parameters of the chosen ML algorithm
2. Select a different ML algorithm to build a new model
3. Use appropriate pre-processing on the data set
4. Improve the training data set; begin by examining any cases that are persistently misclassified



# A Systematic Model Development Procedure (1)

From your dataset, select some as a hold-out set

- Ideally, these should be samples that do not occur in exactly the same form elsewhere in the dataset
- Lock this away!

Using the rest of your data, select a pre-processing method, a model-building method, and settings for it

- Test it on the training data to see if it's promising
- If it is, test it using repeated cross-validation

Repeat the previous step as necessary, until you have found a combination that works for your data

Only then, unlock your hold-out set and test on it



# A Systematic Model Development Procedure (2)

As you follow this procedure, always good practice to keep notes of what you have tried and what results you got

- Learn from these about what methods are good for your domain

Bear in mind that you may need to adjust your dataset

- There may be gaps, inadequacies or errors in it
- Watch out for systematic errors, independent of analysis method

Other issues to watch out for:

- As you adjust a setting of a model-building method, you should find that it ultimately plateaus or starts to get worse: then you know you've gone far enough
- If there is a big divergence between performance on training set and on independent data, it's a sign of **overfitting**



# A Systematic Model Development Procedure (3)

There are no hard-and-fast rules for which algorithm will work well for your data

- Different datasets have different properties
- Different algorithms make different assumptions, that are either well suited or poorly suited to the particular dataset
- Wolpert's "No Free Lunch Theorem", mentioned earlier

Algorithms that reflect general properties of your application domain tend to work well

- Choose or design algorithms based on experience of the domain to make them "close to the problem", so that they work better than generic statistical methods



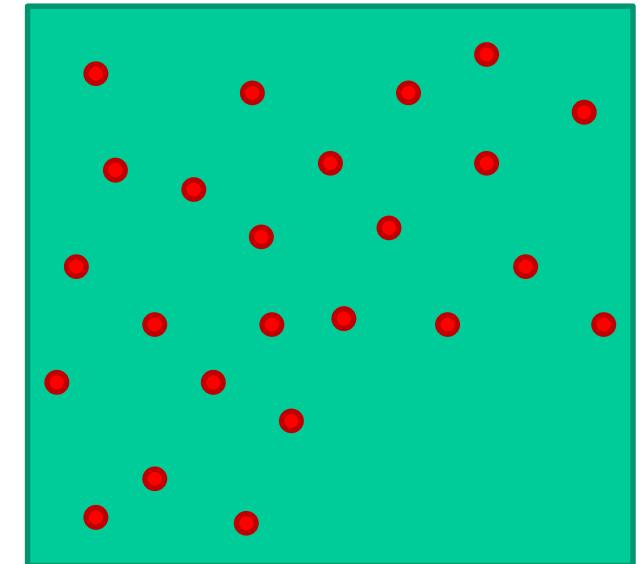
# Training Dataset Considerations (1)

- A training dataset consists of:
  - A set of cases, with the same quantities measured for each
  - For each one, the correct answer to the question to be answered: this is **ground truth**: essential for both constructing and evaluating the model
- This data will be used to build a model that generalizes and interpolates from the data
  - For new, unknown cases, make best estimate of correct answer



## Training Dataset Considerations (2)

- Training data must cover an appropriate range of samples & situations
  - Model-building methods assume that the training data is drawn from the same space as future data on which it will be applied
  - Model evaluation methods are only meaningful with this assumption
- Consider how the model will be used
  - What data collection conditions will apply?
  - Will there be variations, noise, etc?
  - What kinds of data will be presented to the model?
- Aim to get good **coverage** of the space of data on which the model will subsequently be used





# Training Dataset: Coverage for Classification

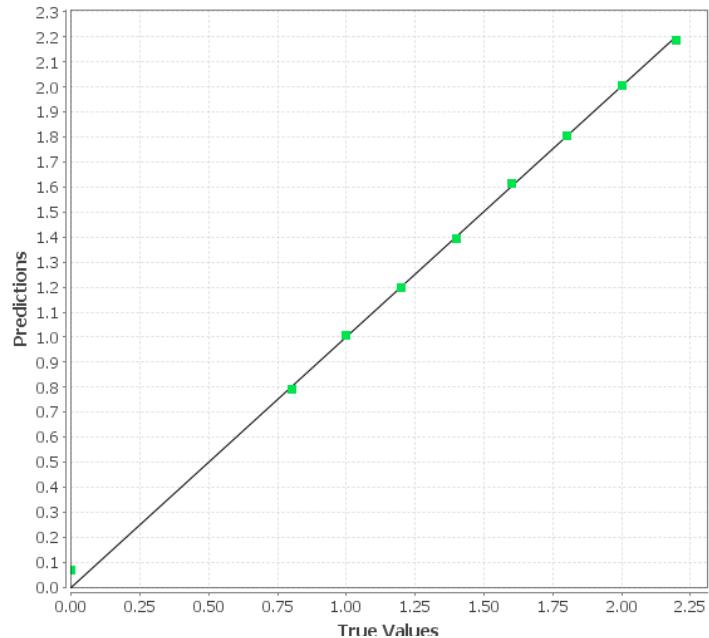
- Example: building a model to detect cocaine in wide variety of cutting agents
- Need a representative sample of cutting agents
- Need **positive and negative** samples
- **Positive:** Pure cocaine; cocaine in a variety of cutting agents
- **Negative:** Cutting agents without cocaine; other drugs
- For classification model, do **not** need exact concentration in each sample, as long as you are certain it is present/absent in each
- Do **not** need every possible combination of cocaine plus all possible cutting agents, but **do** need enough for the model to be able reliably **generalize** from the data





# Training Dataset: Coverage for Regression

- E.g: building a model for concentration of a solvent, for a process monitoring
  - The solution is tightly controlled
  - Solvent should be between 10% and 15%
  - Want to estimate it to within 0.1%
- Need samples covering a good range of concentrations, focusing between 10% and 15%
  - Not much point in having lots of samples in the range 20%-90%
- Some pure samples and some 0% are useful to include
- To estimate concentration to 0.1%, must have at least that accuracy in ground truth measurements
  - Variations of that amount must manifest themselves in the data!





# Guidelines for Choosing Algorithms

In what order should you try different algorithms?

- Try simpler algorithms before more complex ones
- Occam's Razor: Reduce risk of **overfitting**

Try simpler settings before more complex ones

- E.g. Linear SVM before Quadratic or Cubic

Be aware of the effects of pre-processing:

- Can help or harm
- E.g. normalisation can destroy quantitative information

No amount of algorithmics can fix bad data

- Wrongly labelled, instrument errors, dominated by noise, containing unknown constituents, etc.
- Not representative of the problem
- Not a good sampling of the problem space



# Machine Learning is an Iterative Experimental Process

- Try different algorithms that might be relevant to the application; modify existing algorithms if needed
- Experiment with different settings
- Figure out whether pre-processing might help
- Understand the effects and assumptions of methods
- Be familiar with your data
- If you see systematic errors in models, check whether they arise from the data
- Be patient, be thorough, question everything!

*Practical  
machine  
learning  
advice*



# Some (FOSS) ML Tools

## Weka: Command line or GUI; API in Java

The image shows three windows of the Weka software:

- Weka GUI Chooser:** A window titled "Weka GUI Chooser" with tabs for Program, Visualization, Tools, and Help. It features the WEKA logo and a bird icon. The "Program" tab is selected.
- Weka Explorer:** A window titled "Weka Explorer" with tabs for Preprocess, Classify, Cluster, Associate, Select attributes, and Visualize. The "Classify" tab is selected, showing a classifier named "J48 -C 0.25 -M 2".
  - Test options:** Radio buttons for "Use training set" (selected), "Supplied test set", "Cross-validation", and "Percentage split".
  - Classifier output:** A list of metrics including Correctly Classified, Incorrectly Classified, Kappa statistic, Mean absolute error, Root mean square error, Relative absolute error, Root relative error, and Total Number.
  - Result list:** A list of recent runs: 10:06:04 - trees.Id3, 10:06:19 - trees.Id3, and 10:06:35 - trees.J48 (the last one is highlighted).
- Weka Classifier Tree Visualizer:** A window titled "Weka Classifier Tree Visualizer: 10:06:35 - trees.J48 (weather.symb...)" showing a decision tree for weather prediction.

```
graph TD; outlook{outlook} -- "= sunny" --> humidity{humidity}; outlook -- "= overcast" --> yes40["yes (4.0)"]; outlook -- "= rainy" --> windy{windy}; humidity -- "= high" --> no30["no (3.0)"]; humidity -- "= normal" --> yes20["yes (2.0)"]; windy -- "= TRUE" --> no20["no (2.0)"]; windy -- "= FALSE" --> yes30["yes (3.0)"];
```

**Weighted Avg.**: 1 0 1 1 1 1

**Confusion Matrix**:

	a	b	-- classified as
a	9	0	a = yes
b	0	5	b = no



# Some (FOSS) ML Tools

## Apache Mahout:

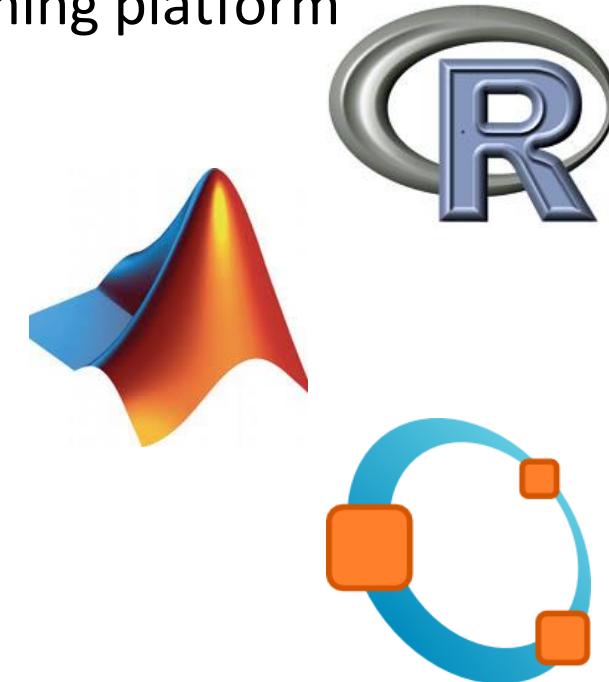
- Scala algorithms implementations built on Spark
- Java implementations of algorithms that are distributed (built on Hadoop) or otherwise scalable:  
older ones built with MapReduce
- Collaborative filtering, clustering and classification





## Other Tools Often Used

- R
  - General-purpose mathematical programming platform
  - CRAN has good collection ML packages
- Matlab & Mathematica
  - Neither are FOSS! Quite expensive
- Octave
  - Open-source partial clone of Matlab
- Your favourite programming language
  - Lisp
  - Java
  - C++
  - Python

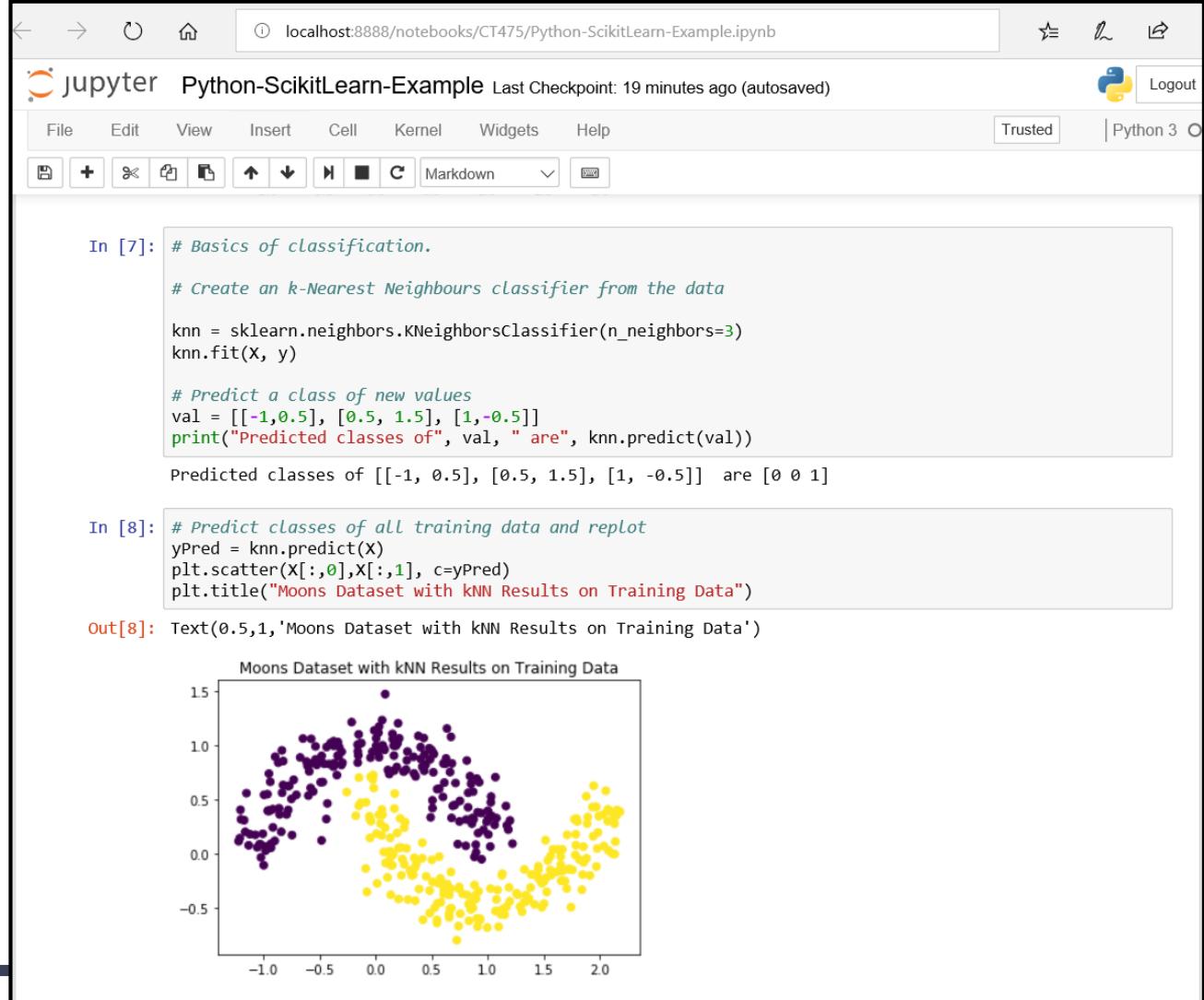




# Other Tools Often Used

## Jupyter Notebook with scikit-learn

- Interactive Python with rich-text web interface
- Other languages also supported
- Keep code, notes & outputs together
- Useful Packages:  
**scikit-learn** (ML);  
uses NumPy, SciPy & Matplotlib



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** localhost:8888/notebooks/CT475/Python-SkicitLearn-Example.ipynb
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- In [7]:**

```
# Basics of classification.  
  
# Create an k-Nearest Neighbours classifier from the data  
  
knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=3)  
knn.fit(X, y)  
  
# Predict a class of new values  
val = [[-1,0.5], [0.5, 1.5], [1,-0.5]]  
print("Predicted classes of", val, "are", knn.predict(val))
```

Predicted classes of [[-1, 0.5], [0.5, 1.5], [1, -0.5]] are [0 0 1]
- In [8]:**

```
# Predict classes of all training data and replot  
yPred = knn.predict(X)  
plt.scatter(X[:,0],X[:,1], c=yPred)  
plt.title("Moons Dataset with kNN Results on Training Data")
```
- Out[8]:** Text(0.5,1,'Moons Dataset with kNN Results on Training Data')
- Figure:** A scatter plot titled "Moons Dataset with kNN Results on Training Data". The plot shows two classes of data points (purple and yellow) separated by a decision boundary, with the kNN results overlaid.



# Learning Objectives - Review

Now that you completed this topic successfully, you can ...

- Discuss distinctions between training, testing & validation data sets
- Perform cross-validation
- Describe and compute performance measures, learning curves and ROC graphs
- Select and perform appropriate statistical tests
- Give guidance on compiling a data set and selecting & comparing methods for a task
- Discuss related issues, including being able to decide how many test data sets are enough
- Discuss and select machine learning tools



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 8: Linear Regression in One and Multiple Variables (Part 1)

Prof. Michael Madden  
Chair of Computer Science  
Head of Machine Learning & Data Mining Group  
National University of Ireland Galway



# Learning Objectives

After successfully completing this topic, you will be able to ...

- Explain what Linear Regression is and its use in ML
- Describe and implement a Gradient Descent algorithm for learning LR parameters with one or multiple variables
- Describe and implement extensions such as Stochastic GD, regularisation, and polynomial regression
- Considering the characteristics of linear regression, recommend when it would be appropriate to an application
- Discuss and apply feature engineering methods including feature scaling, feature reduction, and transformation methods (e.g. PCA)



# Structure of Videos for This Topic

---

## Week 7

- Linear regression; closed-form solution for linear regression
- Gradient descent for linear regression with one input variable
- Multiple Linear Regression: solving in closed form and with gradient descent

## Week 8

- Feature scaling; polynomial regression
- Bias, variance, underfitting and overfitting in regression
- Gradient descent with regularisation; dimension reduction
- Feature engineering approaches including transformations and subset selection



# Linear Regression Overview

- Given target value ( $y$ ) and set of attribute values ( $x_m$ ), goal of Linear Regression is to find an equation that describes the target in terms of the attributes

- Equation is of the form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

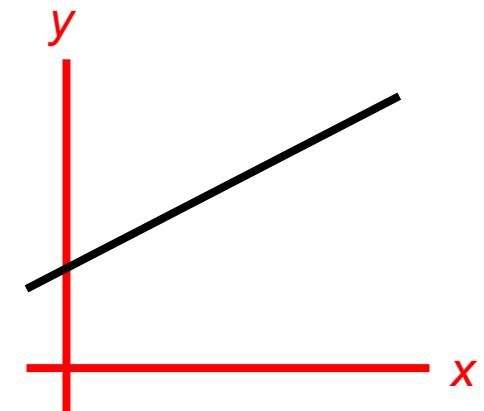
where  $\theta_m$  is the weight associated with attribute  $x_m$

- This is the equation of a line

- Compare to a 2D line, where  $x$  and  $y$  are dimensions:  $y = a + b x$

- Now have higher dimensions, instead of  $x$  we have  $x_1, x_2$ , etc

- Use  $\theta_0$  as intercept rather than  $a$ , and  $\theta_1$  relates to slope in dimension  $x_1$





# Linear Regression Overview

- Have to find equation of this form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

where  $\theta_m$  is the weight associated with attribute  $x_m$

- In other words, have to find values of  $\theta_0 \ \theta_1 \ \theta_2$  etc
- Weights found using least squares fit
  - For a given training set, the weights are found that minimize the squared error between predicted and actual target values
  - If a weight is 0, that attribute has no effect on outcome
  - Assumes that there is little/no correlation between dimensions



# Linear Regression Overview

- Training data consists of sets of values for  $x_1 \ x_2 \dots y$

Size (m <sup>2</sup> )	# Beds	# Floors	Age (yrs)	Price (k€)
195	5	1	40	450
130	3	2	35	220
140	3	2	26	310
80	2	1	30	170
180	5	2	38	400

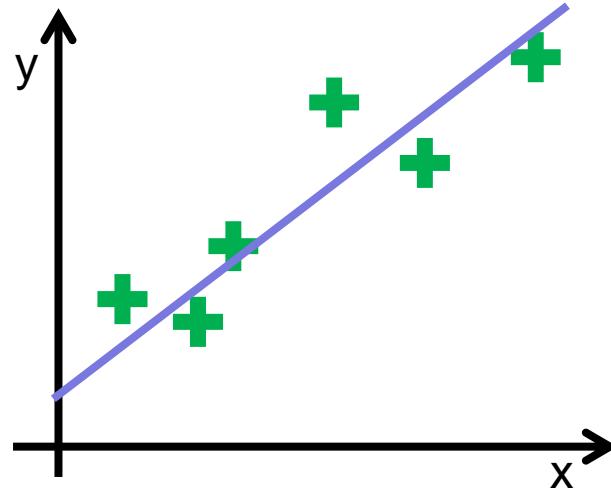


$x_1 \quad x_2 \quad x_3 \quad x_4 \quad y$

- We formulate a hypothesis of the form  
$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$
where  $x$  is vector  $[x_1 \ x_2 \ \dots]$  and  $\theta$  is vector  $[\theta_0 \ \theta_1 \ \theta_2 \ \dots]$
- **Learning objective:** find values for  $\theta$  such that the value output by  $h_{\theta}(x)$  for a given vector  $x$  is as close to  $y$  as possible



# Linear Regression in 1 Variable



Training data: green +

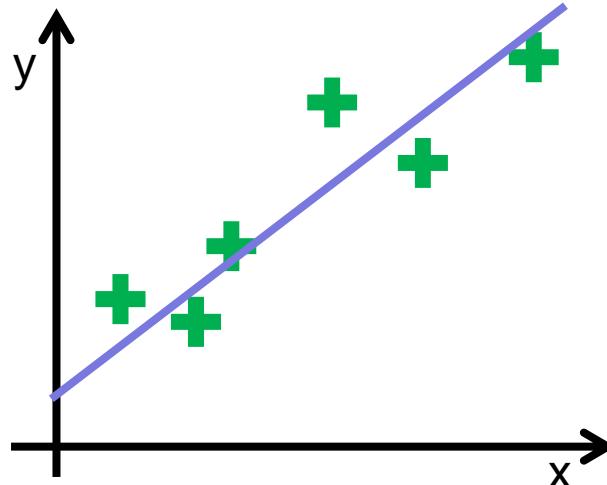
Hypothesis: blue line.

Hypothesis  $h_{\theta}(x)$   
has the form  $h_{\theta}(x) = \theta_0 + \theta_1 x$

$\theta_0, \theta_1$  are the **model  
parameters/weights**.



# Linear Regression in 1 Variable



Training data: green +

Hypothesis: blue line.

Hypothesis  $h_\theta(x)$   
has the form  $h_\theta(x) = \theta_0 + \theta_1 x$   
 $\theta_0, \theta_1$  are the **model  
parameters/weights**.

**Learning objective:** choose  $\theta_0, \theta_1$  such that for each training example  $(x^{(i)}, y^{(i)})$ ,  $h_\theta(x^{(i)})$  is as **close as possible** to  $y^{(i)}$  on average.

One way to formalise this is mean squared error:

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

This defines a **squared error cost function**,  $J(\theta_0, \theta_1)$ :

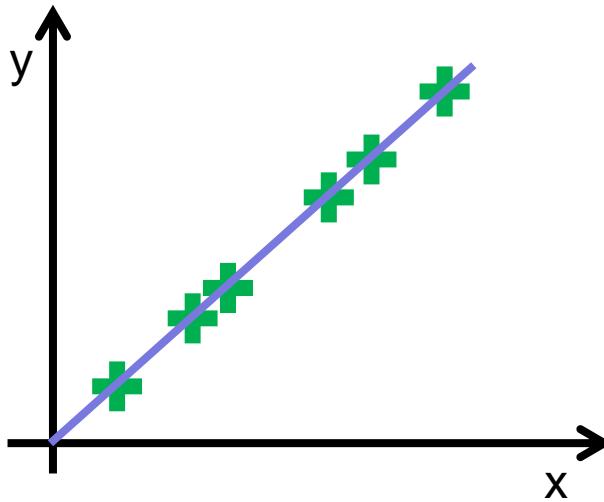
$$J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^N (h_\theta(x^{(i)}) - y^{(i)})^2$$

Minimize  $J$  to find the optimal hypothesis.

Note: could use a different cost function; may yield slightly different results. Squared loss dates to Gauss.

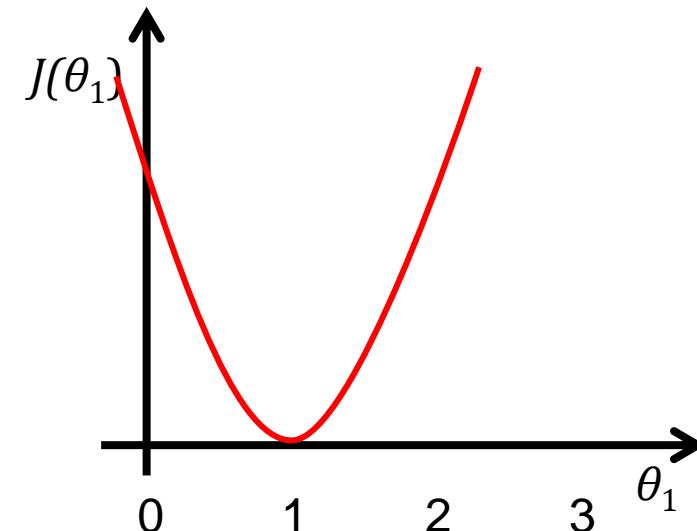


# Cost Function Behaviour [1]



Simplified case:  
data values are  $y=x$

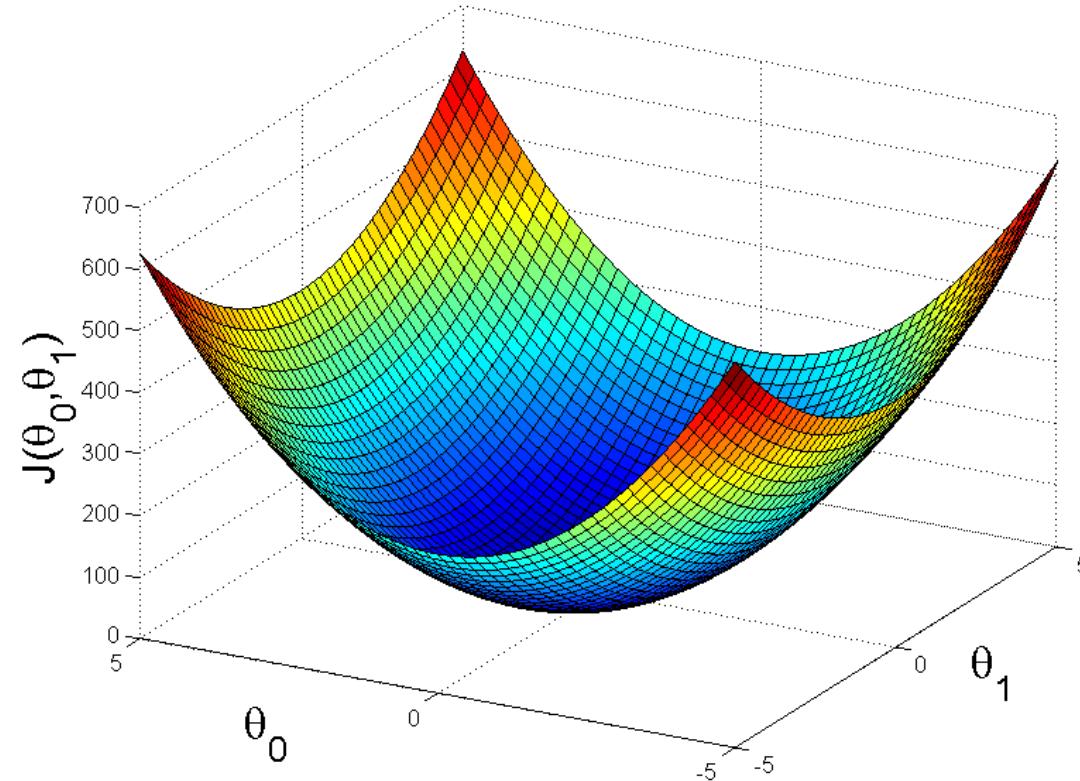
Hypothesis form is  $h_{\theta}(x) = \theta_1 x$   
 $\theta_0 = 0$  (intercept is 0)  
Only vary  $\theta_1$



If there is a straight line that goes through the data, it is defined by the parameters of  $\theta$  where  $J(\theta) = 0$ , irrespective of form of  $J$ .  
If not, min value of  $J$  will be positive and will depend on specific cost fn used.  
Either way,  $J$  is convex, has single minimum.



## Cost Function Behaviour [2]



If we need to optimize  $\theta_0$  and  $\theta_1$ :  
still have single minimum, still a convex optimization problem.  
This also holds when we move to linear regression in multiple variables, with more  $\theta_i$ .



# How Do We Optimize the Parameters?

There are two main ways to find  $\theta_0$  and  $\theta_1$ :

1. Closed form solution:

- Because it is a convex optimization problem, the solution has a unique form

2. Gradient Descent:

- General-purpose algorithm for finding a local minimum of any continuous differentiable function
- For convex hull, it can always find the correct answer if its parameters are reasonable
- Iterative; not as efficient as closed form
- Applicable to many optimisation tasks

Either way, need partial derivatives of  $J(\theta_0, \theta_1)$  ...



# Cost Function Derivatives

Given a cost function  $J(\theta_0, \theta_1 \dots \theta_m)$ ,  
its minimum value w.r.t. all  $\theta_i$  is found when all of its partial  
derivatives are zero.

In this case:

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

Partial derivatives:

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

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



# Cost Function: Closed Form Solution

The partial derivatives have a unique solution:

$$\theta_0 = \frac{1}{N} \sum y^{(i)} - \frac{\theta_1}{N} \sum x^{(i)}$$

$$\theta_1 = \frac{N \sum x^{(i)} y^{(i)} - (\sum x^{(i)}) (\sum y^{(i)})}{N \sum (x^{(i)})^2 - (\sum x^{(i)})^2}$$

This can be computed with a small amount of code or even in a spreadsheet.

LinRegClosedForm.xlsx

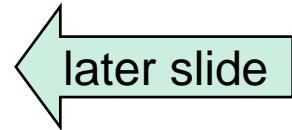


# Gradient Descent

- General-purpose method that works **beyond linear regression**: does not require closed-form solution
- Make initial guess; take incremental steps 'downhill' with step size controlled by **learning rate  $\alpha$**  until little/no change

Batch Gradient Descent **Algorithm**:

**initialise  $\theta$**  to any set of valid initial values

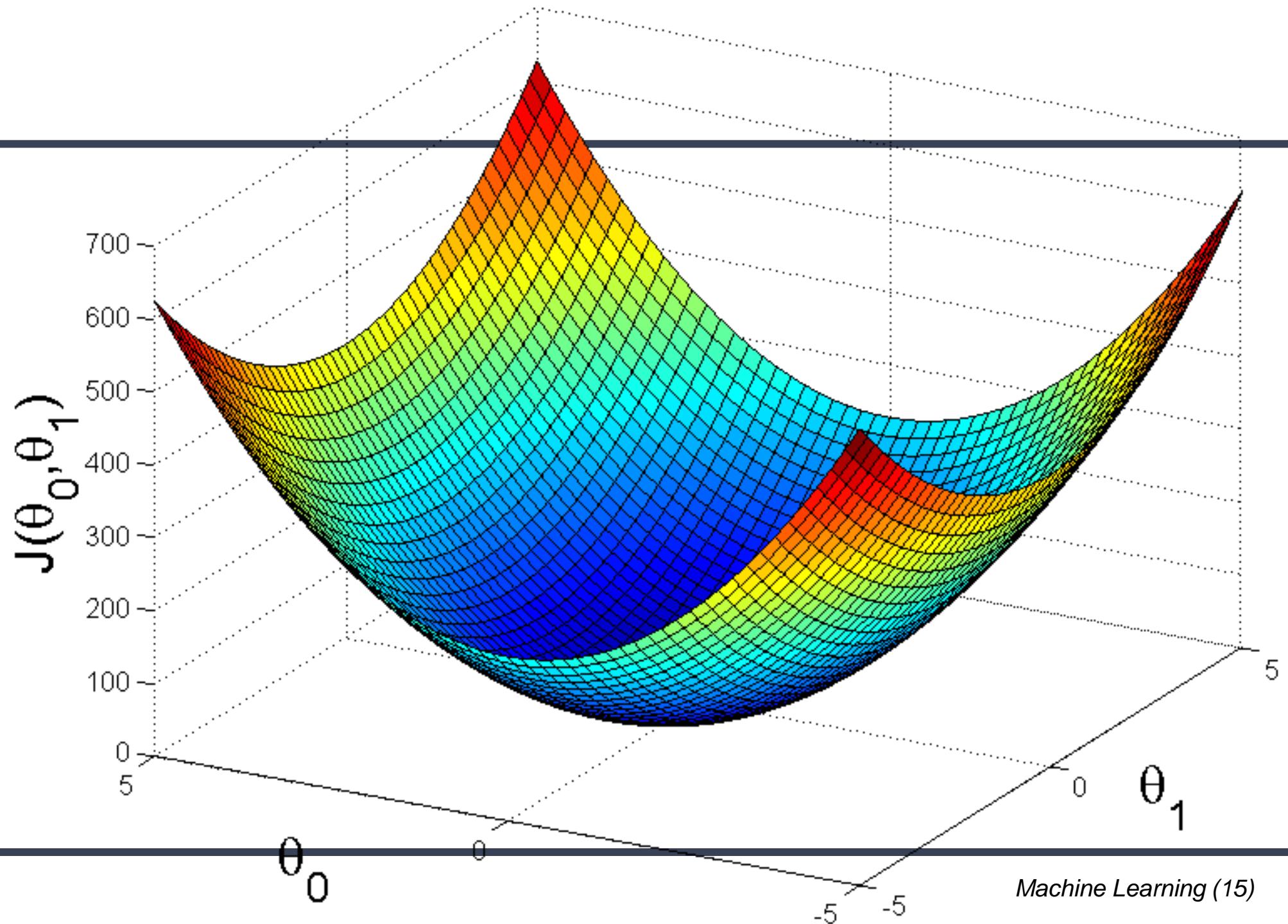
**repeat** until convergence (or time limit reached):  later slide

**simultaneously foreach**  $\theta_j$  in  $\theta$  do:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

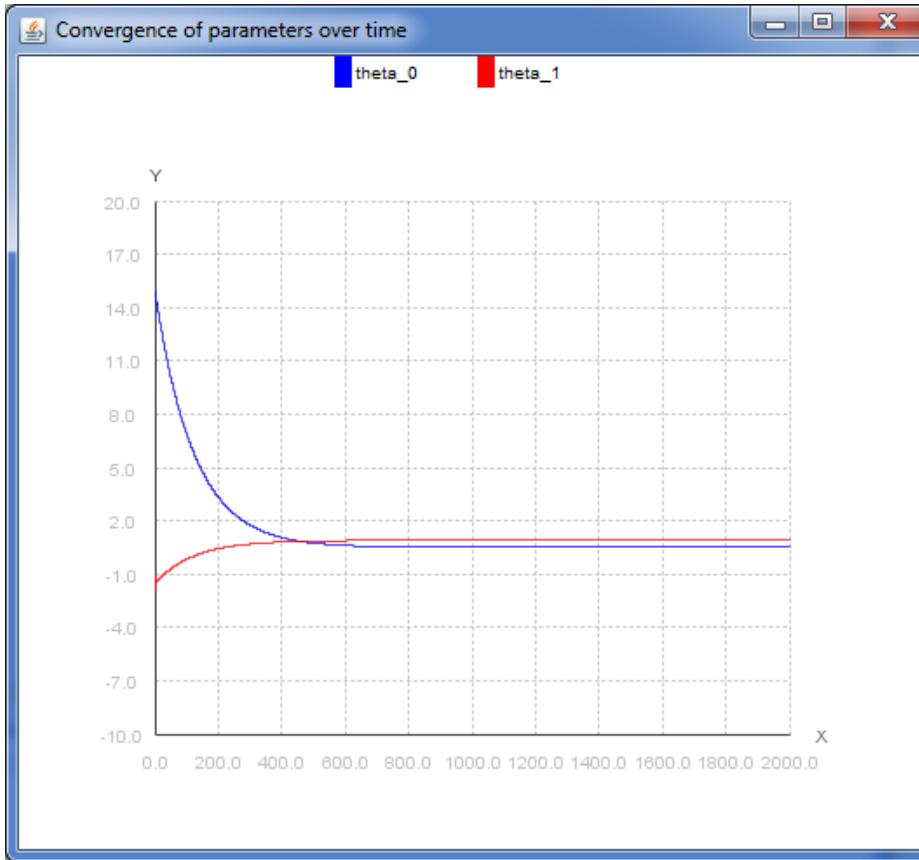


# Gradient Descent Illustration

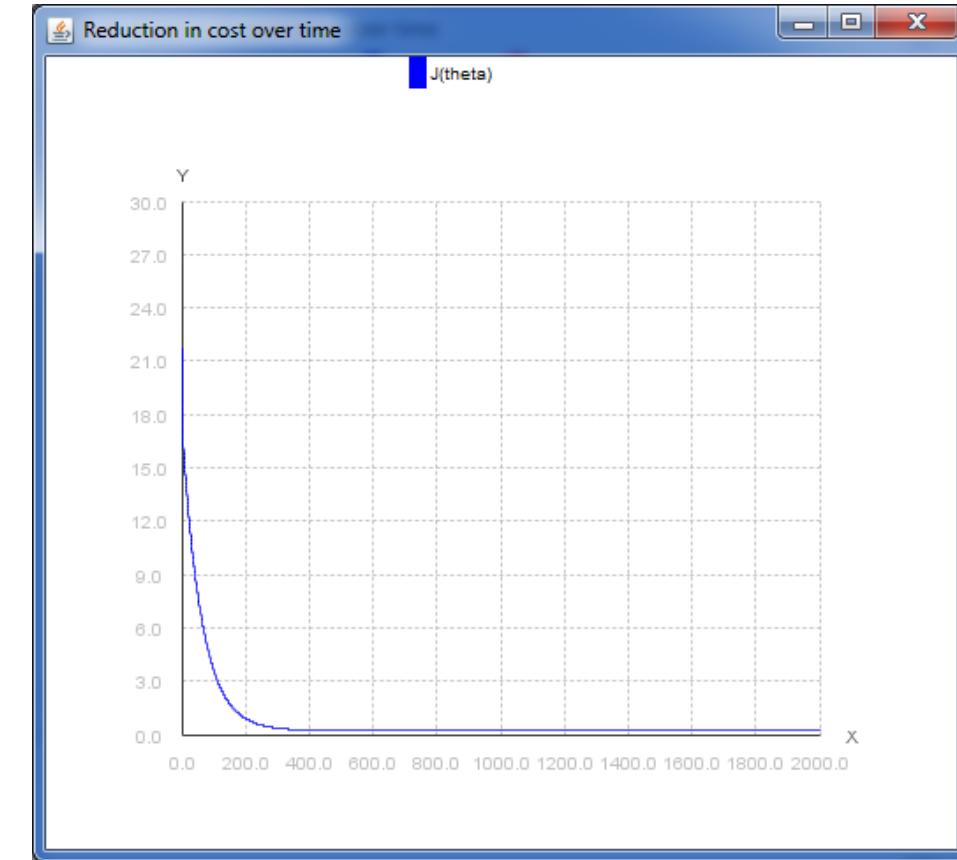




# Checking for Convergence



Changes to  $\theta_i$  will be initially large, reducing as iterations proceed.  
If progress is slow, increase learning rate.



Cost **must decrease** in every iteration, otherwise  $\alpha$  is too large



# Gradient Descent Illustration

- I have written two implementations of Gradient Descent for Linear Regression in one variable, both of which are on Blackboard
- Java version:  
`GradientDescent.java`
- Python (Jupyter Notebook) version:  
`SimpleLinearRegression.ipynb`
- We will examine the second one ...



# Stochastic Gradient Descent

- Rather than using all data together ("batch") in the update function, randomly select a single example for updating  $\theta$ 
  - Keep repeating until convergence
  - Partial derivatives are simplified: only 1 example so  $N=1$ , no  $\Sigma$

$$\frac{\partial}{\partial \theta_1} J(\theta) = \cancel{\frac{1}{N} \sum_{i=1}^N} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$



# Stochastic Gradient Descent

- Can be faster than batch gradient descent
  - Convergence is not guaranteed
- Works well in **online learning**
  - New data arriving all of the time
  - May be ***high velocity*** data
  - As long as all data are identically drawn from same distribution, just sample the data stream as fast as possible for updates



# Multiple Linear Regression [1]

- Usually, have multiple input variables that relate to a quantity of interest

Size (m <sup>2</sup> )	# Beds	# Floors	Age (yrs)	Price (k€)
195	5	1	40	450
130	3	2	35	220
$x^{(3)}$	140	3	2	310
$y^{(3)}$	80	2	1	170
	180	5	2	400

$x_2^{(3)}$

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad y$

N  
no. of cases

No. of attributes:  $m$

Now,  $\mathbf{x}^{(i)}$  is a feature vector of length  $m$ , values from one row

Individual features denoted  $x_j^{(i)}$



# Multiple Linear Regression [2]

Hypothesis is now of the form:

$$h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

To simplify notation, add a dummy input variable  $x_0 = 1$ .

Then:  $h_{\theta}(\mathbf{x}^{(i)}) = \boldsymbol{\theta} \cdot \mathbf{x}^{(i)} = \sum \theta_j x_j^{(i)}$

Cost function essentially same as before, now using vector notation:

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

Reminder of vector dot product:

$$\mathbf{a} = \langle a_1 \ a_2 \ a_3 \ a_4 \rangle$$

$$\mathbf{b} = \langle b_1 \ b_2 \ b_3 \ b_4 \rangle$$

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \dots$$



## Multiple LR: Closed Form Solution

As before, can be solved in closed form.

Let  $\mathbf{X}$  be full data matrix of inputs: row  $i$  corresponds to  $\mathbf{x}^{(i)}$

Let  $\mathbf{y}$  be vector of all outputs

Then the optimal vector of values for  $\boldsymbol{\theta}$  that minimises squared error is found from:

$$\boldsymbol{\theta}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Non-iterative, but requires inverting a matrix of size  $m \times m$   
( $m$  = no. of attributes):  $O(m^3)$

Gradient descent faster for large  $m$  (e.g.  $> 1000$ ).



## Multiple LR: Gradient Descent

The algorithm is **the same** for Multiple LR as before:

**initialise  $\theta$**  to any set of valid initial values

**repeat** until convergence or until limit is reached:

**simultaneously foreach**  $\theta_j$  in  $\theta$  do:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

The partial derivatives are **different**: can be verified to be direct generalisation of the single-variable case:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{N} \sum_{i=1}^N (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$



# Multiple LR: Gradient Descent

- Let's compare the partial derivatives ...

- Single variable case:

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

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

- Multiple variable case:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$



---

*End of*

# Linear Regression in One and Multiple Variables (Part 1)



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 9: Linear Regression in One and Multiple Variables (Part 2)

Prof. Michael Madden  
Chair of Computer Science  
Head of Machine Learning & Data Mining Group  
National University of Ireland Galway



# Review - Multiple Linear Regression [1]

Given multiple input variables ( $x_i$ ) that relate to a quantity of interest ( $y$ ) ...

Size (m <sup>2</sup> )	# Beds	# Floors	Age (yrs)	Price (k€)
195	5	1	40	450
130	3	2	35	220
$x^{(3)}$ 140	3	2	26	310
$y^{(3)}$ 80	2	1	30	170
180	5	2	38	400

$x_2^{(3)}$

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad y$

N  
no. of cases

Goal is to find parameters  $\theta$  for hypothesis  $h_{\theta}(x)$  of the form

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

such that for a given input vector  $x^{(i)}$ ,  $h_{\theta}(x^{(i)})$  best approximates  $y^{(i)}$



# Review- Solving Multiple LR with Gradient Descent

The algorithm for Gradient Descent for Multiple LR is:

**initialise  $\theta$**  to any set of valid initial values

**repeat** until convergence or until limit is reached:

**simultaneously foreach**  $\theta_j$  in  $\theta$  do:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

The partial derivatives are given by:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{N} \sum_{i=1}^N (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$



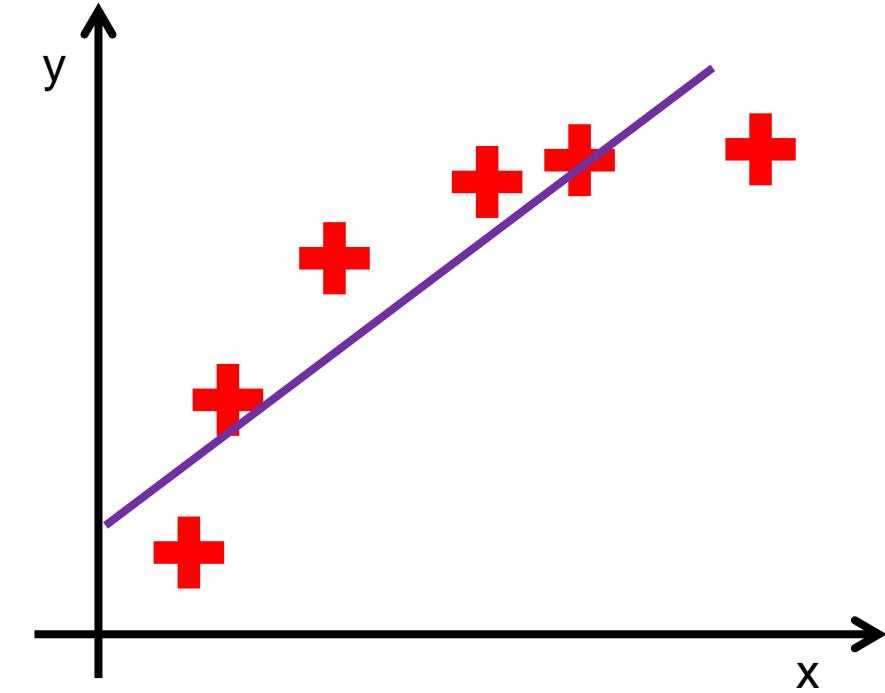
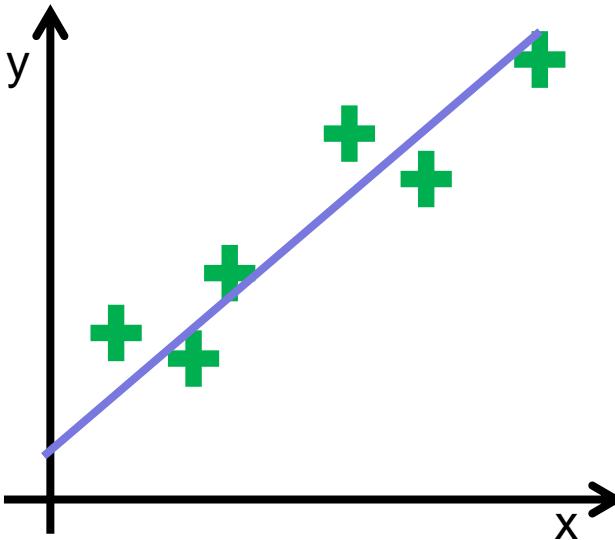
# Feature Scaling

- Feature scaling can help improve G.D. convergence
  - G.D. uses a single tolerance and learning rate:  
helps if all features cover approximately similar ranges
  - Mean-centring the data can help if initial guesses are 0
- Techniques we discussed for kNN work well to rescale all dimensions independently
  - Mean=0, Std deviation=1 [Z-Normalisation]  
$$D \leftarrow (D - \text{Mean}) / \text{StDev}$$
  - Min=-1, Max=1 [-1/1 Normalisation]  
$$D \leftarrow 2(D - \text{Min}) / (\text{Max} - \text{Min}) - 1$$



# Linearity Assumption

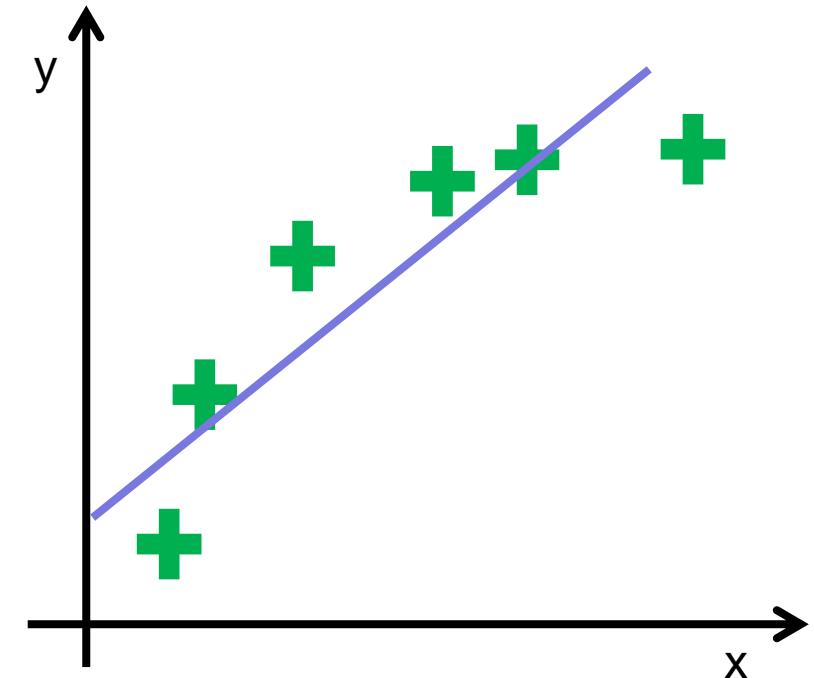
- Linear Regression has a very strong **bias**:
  - Assumes that there is a linear relationship between inputs and outputs
  - Useful if the relationship really is approximately linear, apart from some noise or small variations we don't need to capture
  - Less useful if not ...





# Polynomial Regression [1]

- Key insight: inputs to Linear Regression do not have to be the original raw data
  - Can generate new input variables based on **transformations** of the original ones
- That is basis for **Polynomial Regression**:
  - For some/all of the input variables  $x_n$ , add additional new variables: e.g.  $x_1^2, x_1^3, x_1x_2 \dots$
  - Then perform linear regression as normal





## Polynomial Regression [2]

- Example: suppose initial dataset has attributes  $x_1, x_2, x_3$ 
  - Second-Order Polynomial Regression: create new version of dataset where attributes are:  $x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2$
  - Third-Order Polynomial Regression: create new version of dataset where attributes are **Second-Order ones and**:  $x_1x_2x_3, x_1^3, x_2^3, x_3^3, x_1x_2^2, x_1^2x_2, \dots$
- Using this new version of the dataset, run Multiple Linear Regression
  - Using standard Linear Regression algorithm to discover hypotheses that are non-linear relative to original attributes, since the new inputs to LR are non-linear transformations of original attributes
  - Warning: number of features increases dramatically
- Won't go into full algorithm details, but here is a simple illustrative example with just 1 input variable: `SimplePolynomialRegression.ipynb`



# Polynomial Regression - Final Comments

---

- While polynomial regression is a reasonably common variant of linear regression, it is also possible to use other transformations
- There are other ways of extending linear regression to deal with non-linear data, that are outside of the scope of this module
  - Support Vector Machine with non-linear kernel
  - Multi-layer perceptron: feed-forward neural network with at least one hidden layer



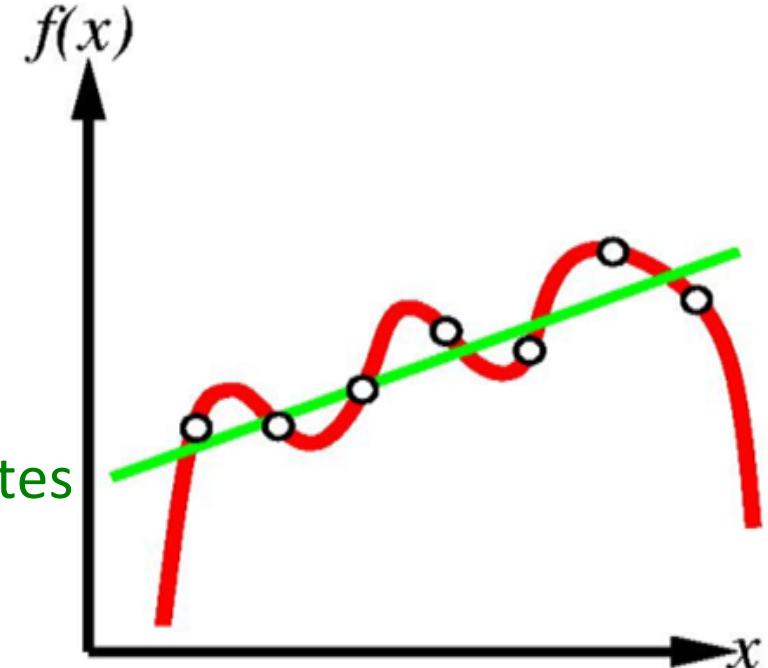
# Regression: Bias, Variance, Underfitting and Overfitting [1]

- ML algorithms may be characterised in terms of **bias** and **variance**:
- Variance errors: over-sensitivity to small changes in training data.
  - High variance => algorithm may fit to random noise in training data, rather than real relationships in data => **over-fitting**
- Bias errors: due to bad assumptions in the learning algorithm.
  - High bias => algorithm cannot capture all relevant relationships between attributes and output => **underfitting**
- Important: this is an entirely separate concept from bias in the sense of errors caused by humans who have biased judgements
  - Select ML models that support their biases, or supply biased training data
  - Many data sets may have historical biases; e.g. recruitment; sentencing



# Regression : Bias, Variance, Underfitting and Overfitting [2]

- ML Algorithms with **complex hypothesis language** are prone to **overfitting**
- Those with **simpler hypothesis language** are prone to **underfitting**
  - If you increase complexity of hypothesis, you increase ability to fit to the data, but might also increase risk of overfitting
- Linear Regression:
  - Simple hypotheses: straight lines
  - High bias and low variance; risk of underfitting
- Polynomial Regression:
  - More complex hypotheses: many combinations of attributes
  - Higher variance and weaker bias; risk of overfitting
  - Need more data





# Gradient Descent with Regularization

- High number of variables (whether from Polynomial Regression or otherwise) increases risk of overfitting
- Potential solutions:
  - Reduce number of variables (see later slides)
  - **Regularize**: encourage low/zero values for weights  $\theta$ :  $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$
- **Regularization**:
  - Add a term to cost function to reflect some measure of **complexity** of the hypothesis:  
$$\text{Cost}(\theta) = \text{EmpiricalError}(\theta) + \lambda \text{Complexity}(\theta)$$
  - Then, Linear Regression will find parameters for  $\theta$  that make a trade-off of error vs. complexity
  - Can also be seen as trading off **bias vs. variance**



# Gradient Descent with Regularization

- New cost function:

$$Cost(\theta) = EmpiricalError(\theta) + \lambda Complexity(\theta)$$

Empirical Error: old cost function  $J()$

Complexity: 
$$L_q(\theta) = \sum_{i=1}^m |\theta_i|^q$$

- Note:

$\lambda$  controls tradeoff between error & complexity.

We don't regularize  $\theta_0$  as we don't want to eliminate it.

Depending on  $q$ , get different regularisation effects:

$q=1$  is popular because it encourages sparse models (often sets many weights to 0), but  $q=2$  simplifies the maths



# Gradient Descent with Regularization

The cost function therefore becomes:

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

And its partial derivative for Gradient Descent is:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{N} \theta_j \quad (\text{where } j > 1)$$

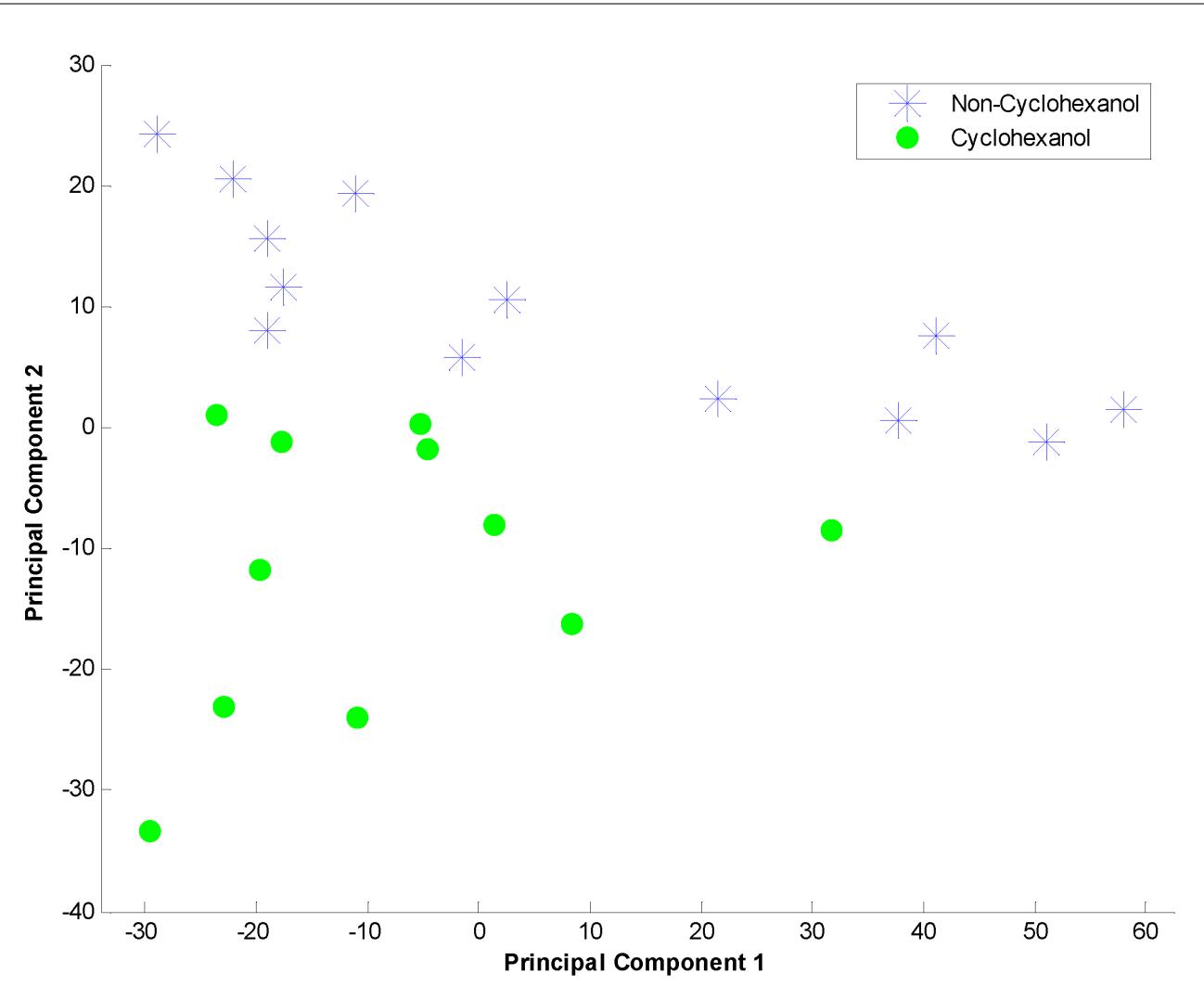


# Linear Regression & Dimension Reduction

- Linear Regression can fail (and give very bad predictions) if the number of samples is **smaller than** the number of dimensions per sample, *unless* you use regularisation
  - Closed form is ill-conditioned; lose convergence guarantees of GD
  - In some application domains, this happens routinely
  - Chemometrics, DNA analysis, medical image analysis, ...
- Solution: Feature Selection or Transformation+Selection
  - Reduce dimensionality
  - Manual or algorithmic
  - Also improves speed of computations
- Principal Component Analysis is popular:
  - Transforms data to new axes where dimensions are ordered in terms of how much of the data variance they explain
  - Discard all but top  $N$  dimensions ( $N$  is application-dependent)
  - Good fit with Linear Regression assumptions: linear & non-correlated

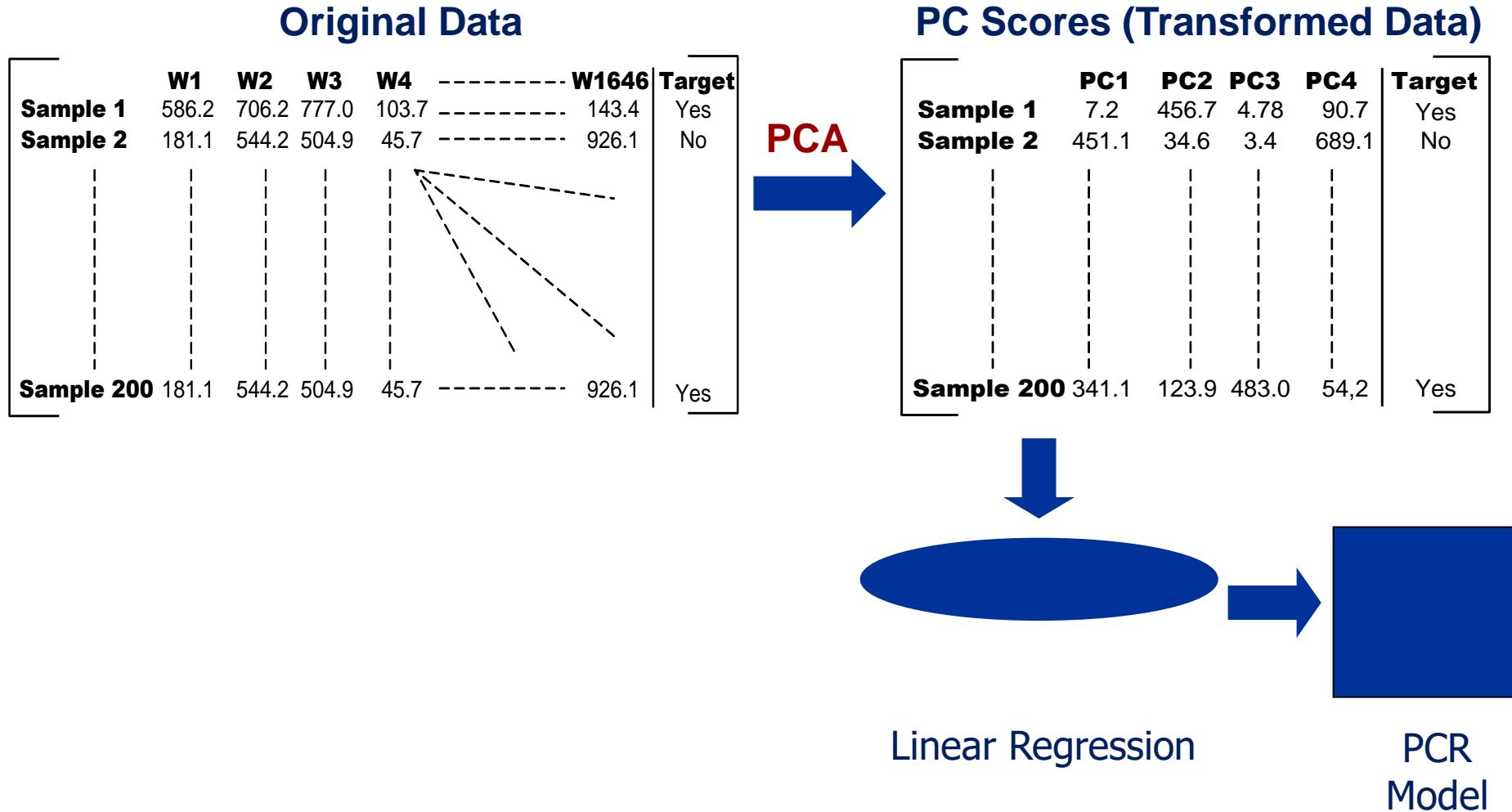


# PCA Example





# Principal Component Regression





# PCA Features

- When to consider using PCA:
  - Useful pre-treatment for many ML methods
  - Can capture most of variance in data, while greatly reducing dimensionality
- Particularly useful with methods that do not work well with high-dimensional data, e.g.
  - Linear Regression
  - $k$  Nearest Neighbours
- Limitations:
  - ‘Blind’ to the data labels
  - In some situations major variance is not associated with your question of interest
  - E.g. want to measure small variations in an active ingredient; anomaly detection



# When to Use Linear Regression

- Consider using when:
  - More training cases than attributes per case
- Benefits:
  - Reasonably fast
  - Simple hypothesis representation
    - => Good comprehensibility
    - => low risk of overfitting, at least in low dimensions
- Drawbacks:
  - Simple hypothesis representation
    - => not good for complex decision planes



# Feature Engineering

- Feature Engineering - broad term encompassing:
  - **Feature extraction:**  
e.g. extracting some high-level features from an image
  - **Feature transformation:**  
e.g. principal component analysis, FFTs, etc
  - **Feature subset selection:**  
keeping only a subset of the original features,  
discarding the rest



# Feature Subset Selection: Overview (1)

- Feature selection implicit in all data pre-processing
  - Normally use domain knowledge to decide what features are relevant
  - PlayTennis example: do we need to consider whether the stock market rose or fell today?
- Aside:
  - Sometimes we are mistaken in thinking a feature is irrelevant!
  - Sometimes we'd like to include a feature but we can't measure it
- Automatic Feature Subset Selection:
  - Starting from list of features that **might be** relevant, identify those that are **actually** relevant
  - Hence, Feature **Subset** Selection

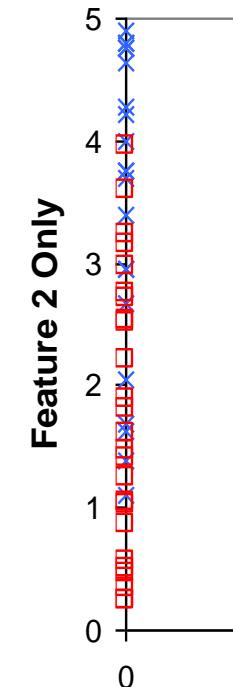
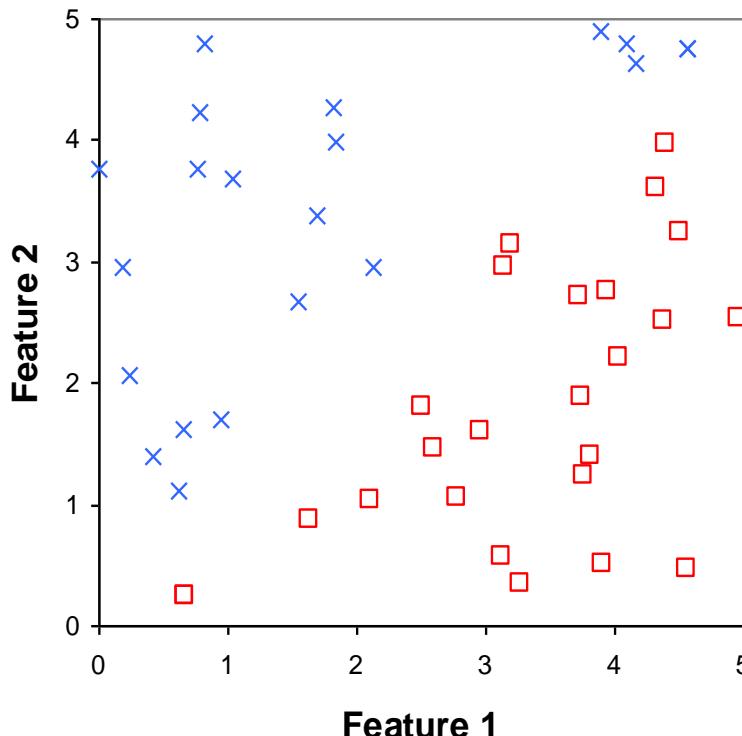


## Feature Subset Selection: Overview (2)

- A general technique ...
  - Particularly useful for kNN, Naïve Bayes, ...
  - Any method that assumes all attributes significant and/or independent
- Can also improve performance of algs that don't!
  - E.g. Decision trees, neural nets
- Why?
  - The less junk they have to deal with, the less likely they are to be confused
- Can also improve economy of representation



# Features Can't be Considered Independently



Here, we must look at both Feature 1 and Feature 2

Either one alone is insufficient to distinguish between the two different classes



# Feature Subset Selection is Hard ...

- Suppose we have 100 features, but 98 are irrelevant
- If features could be considered independently:
  - Just test each in turn to find the important ones: 100 tests
  - **But they can't!**
- Need to consider **feature combinations**
  - $2^{100}$  tests!
  - **So when I say hard, I mean NP**
- Two approaches: Filter and Wrapper
  - Both require search and evaluation: differ in evaluation
  - $2^N$  combinations of N features => heuristic search



# Feature Selection: Filter

- Evaluates features independent of learning alg.
  - Requires measurement of relevance of features to class:
    - Statistical tests
    - Features selected by decision tree
    - kNN: same/different values for nearby instances with different/same class
    - other
  - Filter bias different to induction alg bias





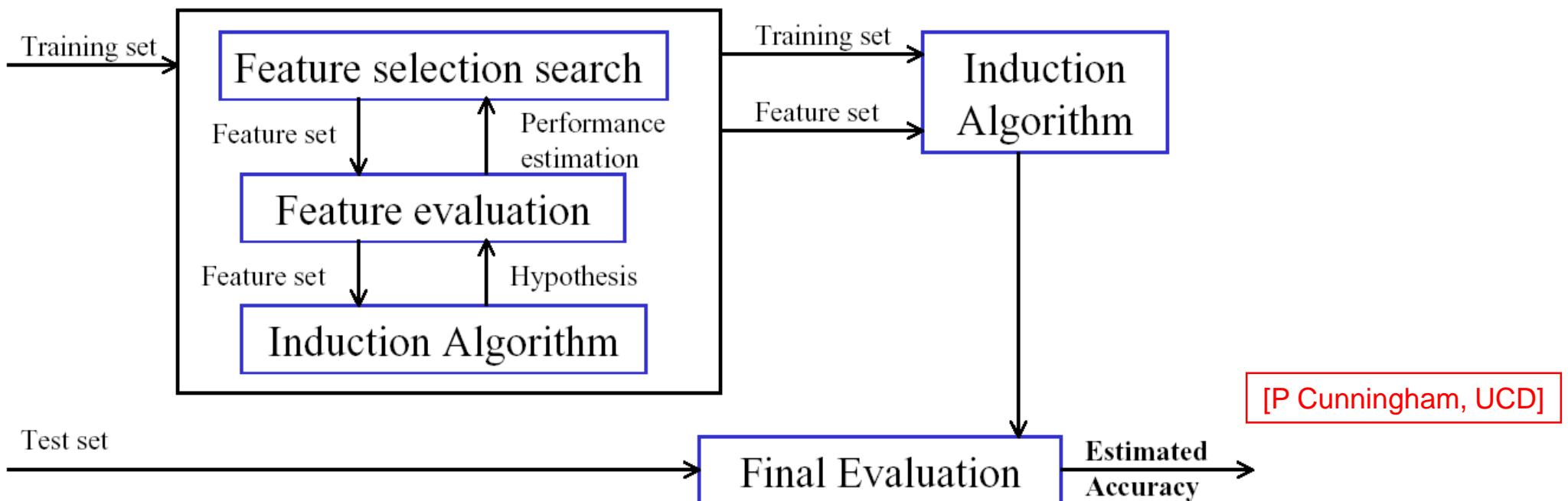
## Feature Selection: Filter

- Filters are usually less computationally intensive than wrappers. Often used in the pre-process phase.
- But they produce a feature set which is not tuned to the specific algorithm more general than a wrapper.
- Can be useful for exposing the relationships between features.



# Feature Selection: Wrapper

- Selection process wrapped around learning alg
  - Features selected are suited to learning alg
  - More computationally intensive





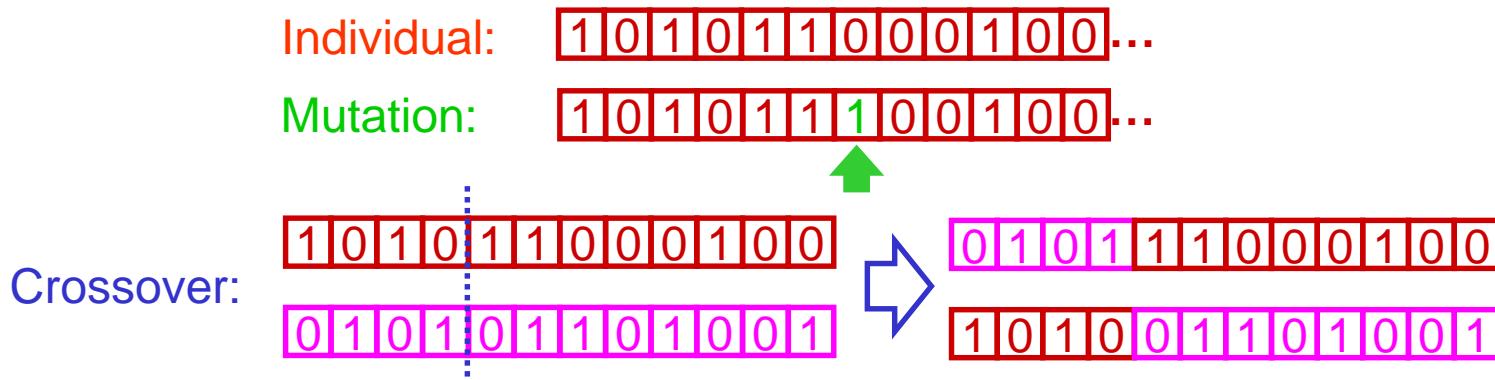
# Feature Selection: Wrapper Example (1)

- Prediction of concentration of chemicals from spectra
  - 510 points/spectrum: most correlated or irrelevant
  - Search for a subset of attributes that **minimises prediction error** on known cases
- Search using Genetic Algorithm:
  - Optimisation technique inspired by evolution
- Procedure:
  - Create random population of potential solutions
  - Assess fitness of each (**1/error**)
  - Delete unfit individuals; create new population by **crossover** and **mutation**
  - Repeat for a large number of generations until a stable solution is found



# Feature Selection: Wrapper Example (2)

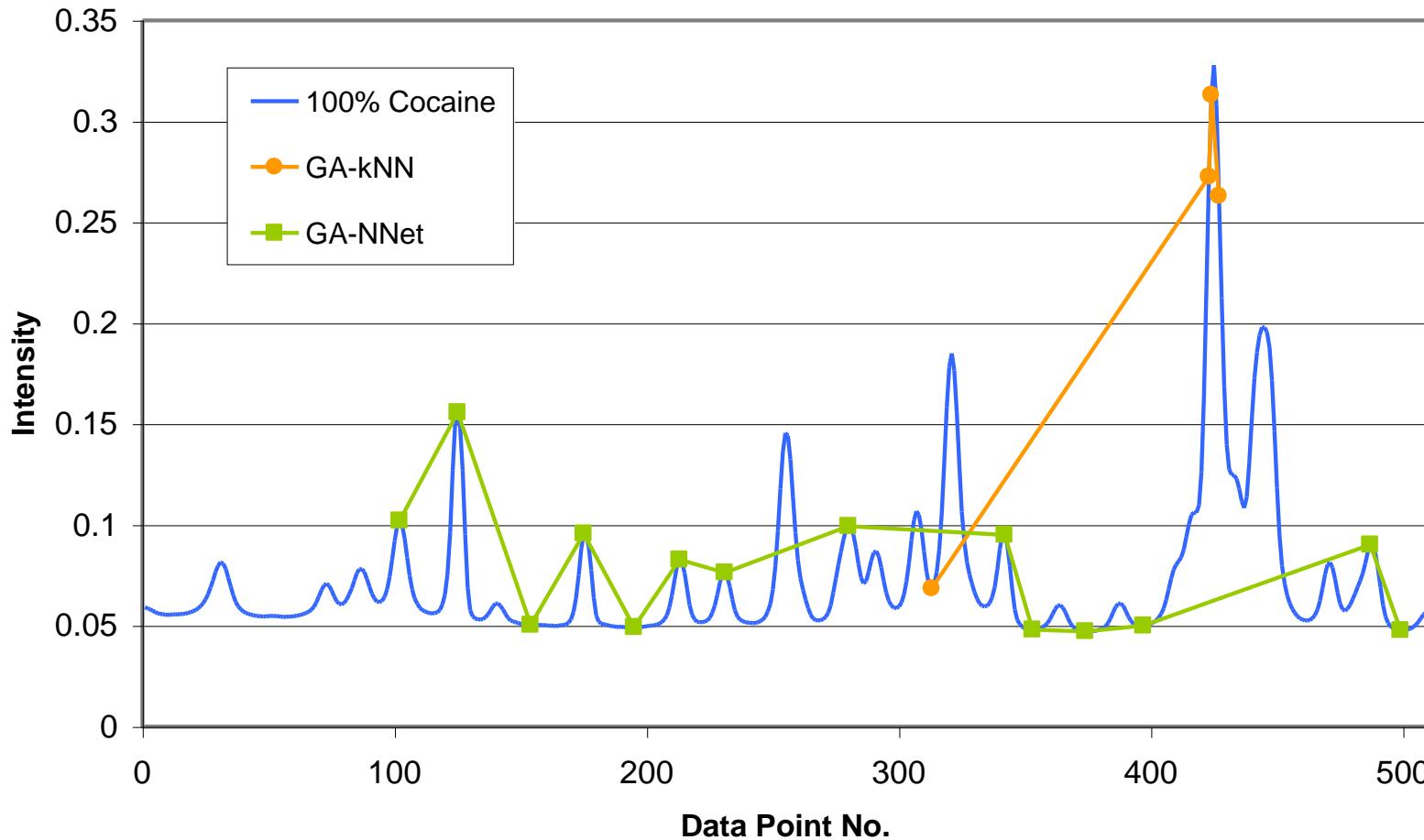
- In this application:
  - Each **individual** is string of 510 binary digits:  
1=select this point in all spectra, 0=ignore it
  - Fitness assessed by applying ML algorithm (kNN or NNet) and calculating Root Mean Square Error
  - **Mutation**: randomly toggle a bit in an individual
  - **Crossover**: select a random point in two individuals, cut and swap 'heads' and 'tails'





# Feature Selection: Wrapper Example (3)

- Hennessy, Madden & Ryder, 2004
- NNet selected 14 points; kNN selected just 4





# Feature Engineering: Transforming & Extracting Features

- “Feature Engineering is the Key”
  - Domingos, CASM 2012
  - “Easily the most important factor is the features used” in determining if a ML project will succeed/fail
- Some estimate that they spend 70% of their time on feature engineering activities
- Such work is **mainly** manual in nature, based on understanding the domain and the algorithms you are working on



# Feature Transformation/Extraction: Motivating Example

**City 1 Lat. City 1 Lng. City 2 Lat. City 2 Lng. Drivable?**

123.24	46.71	121.33	47.34	Yes
123.24	56.91	121.33	55.23	Yes
123.24	46.71	121.33	55.34	No
123.24	46.71	130.99	47.34	No

ML algorithms would likely fail at this task

But if we use our knowledge of trigonometry to compute distances ...

**Distance (mi.) Drivable?**

14	Yes
28	Yes
705	No
2432	No



# Feature Engineering: Key Steps

- Understand your input data
  - Where does it come from?
  - What assumptions come from how it was collected?
  - Any sources of noise and error?
- Understand your algorithms
  - Can they work with the raw data?
  - Would they be helped if the data was adapted?
- Apply an iterative approach
  - Don't just do 10 things at once:  
understand the effect of each step you perform
  - Keep testing to see if your engineering steps help



# Feature Extraction: Cars Example (1)

- Munroe & Madden, 2005
- Objective: Identify the make and model of a vehicle from images
- Five different vehicle types to recognise
  - Apart from these 5, other types included in test set



Opel Corsa



Ford Focus



Ford Fiesta



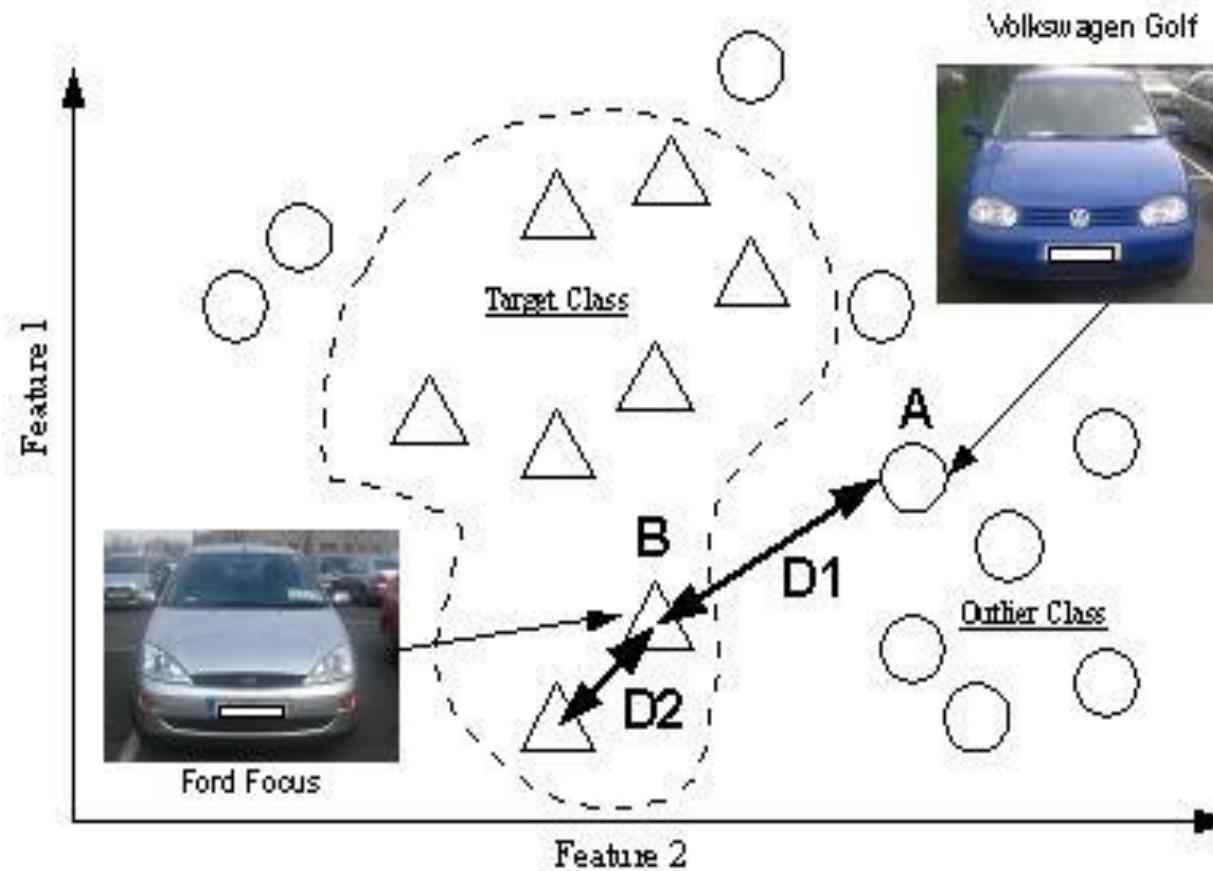
VW Golf



VW Polo



## Feature Extraction: Cars Example (2)

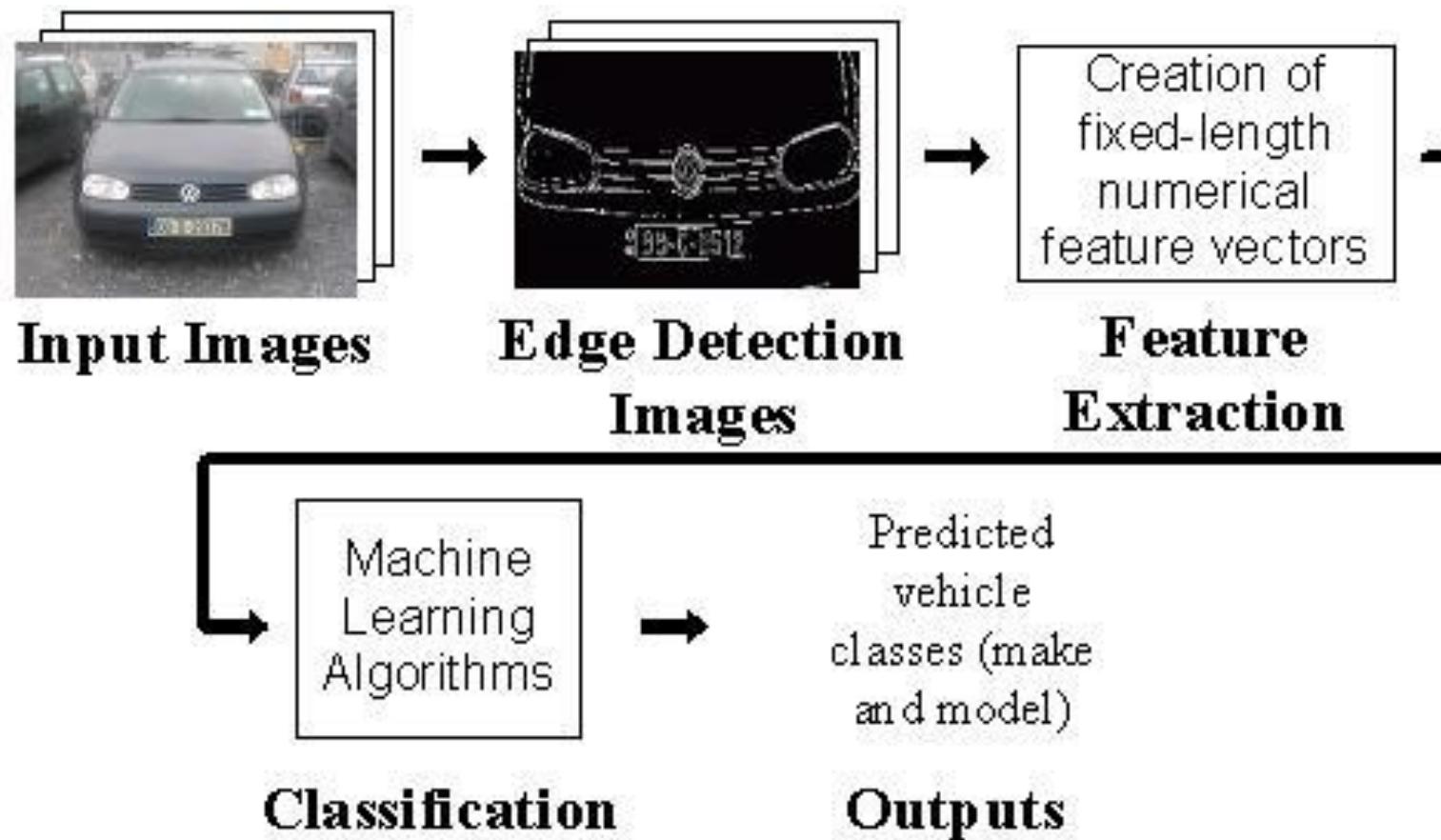


Devised the **One-Class k-Nearest Neighbour** algorithm for this task.  
Also evaluated off-the-shelf **multi-class classification** algorithms.



## Feature Extraction: Cars Example (3)

Overall System for Vehicle Make/Model Recognition from Images

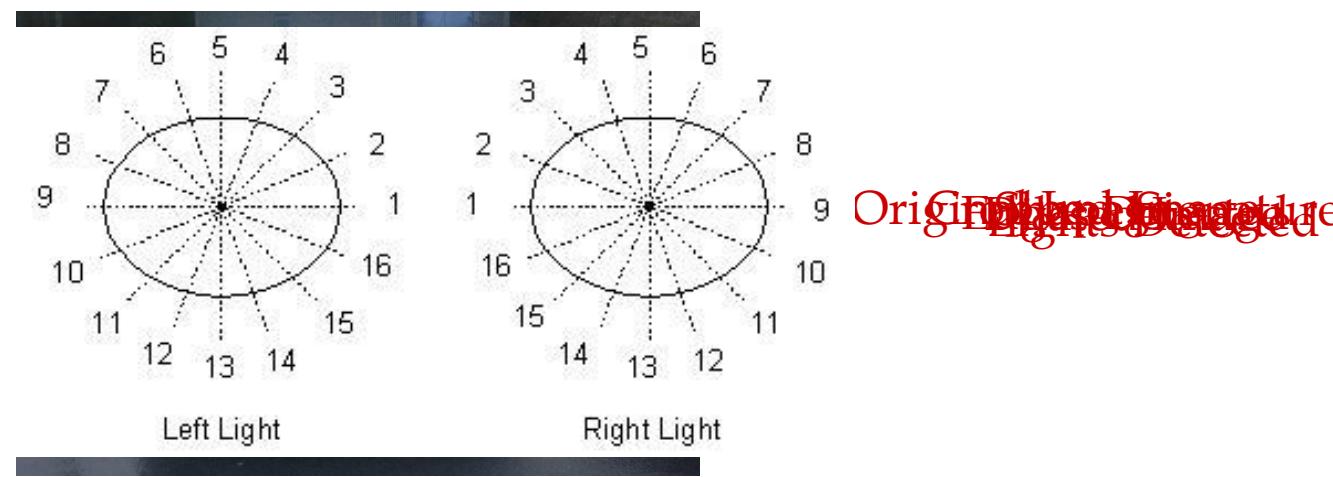




# Feature Extraction: Cars Example (4)

## Feature Extraction Processes

1. Crop the image
2. Canny Edge Detection: Detect the edges
3. Dilation: Grows or thickens the object outlines
4. Detection of vehicle lights
5. Shape Signature: 1-D functional representations of the boundaries of objects





# Feature Engineering: Final Comments

- Manual feature engineering techniques often make classification tasks, but are problem-specific
  - Scaling problem: every new task needs new work
  - Would be preferable if we could use ML to automate this
- If you study deep learning, you will learn that one of its goals is to integrate:
  - Feature engineering
  - Classification/regression
  - Model selectioninto a unified analysis framework.



# Learning Objectives - Review

Having successfully completed the material from last week and this week, you can now ...

- Explain what Linear Regression is and its use in ML
- Describe and implement a Gradient Descent algorithm for learning LR parameters with one or multiple variables
- Describe and implement extensions such as Stochastic GD, regularisation, and polynomial regression
- Considering the characteristics of linear regression, recommend when it would be appropriate to an application
- Discuss and apply feature engineering methods including feature scaling, feature reduction, and transformation methods (e.g. PCA)



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 10: Linear Classifiers with Hard and Soft Thresholds

Prof. Michael Madden

Chair of Computer Science

Head of Machine Learning & Data Mining Group

National University of Ireland Galway



# Learning Objectives

After successfully completing this topic, you will be able to ...

- Explain the drawbacks of using linear regression directly for classification problems
- Describe and implement approaches to improve on this:  
Linear Classifiers and Logistic Regression
- Discuss their characteristics and limitations



# Classification

- Is unknown sample  $\text{X}$  or  $\text{O}$  ?

Goal: Find model that correctly classifies  
a new sample,  $x_i$

- Training Data:  $(x_i : y_i)$

$s_1: (11, 3 : +1) \text{ O}$

$s_2: (3, 10 : +1) \text{ O}$

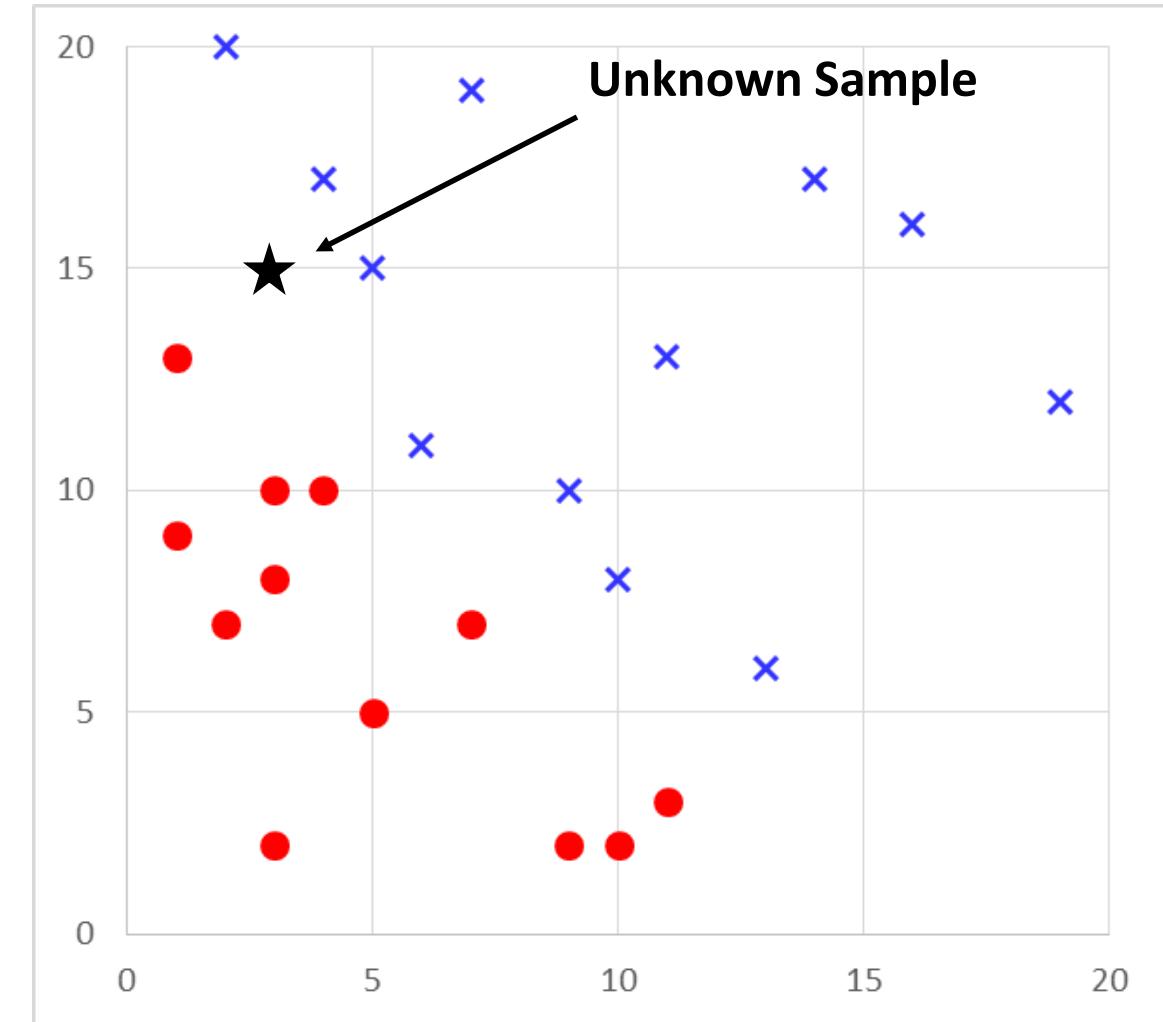
$s_3: (14, 17 : -1) \text{ X}$

$s_4: (16, 16 : -1) \text{ X}$

...

- Unknown Sample  $s = (3, 15 : ?)$

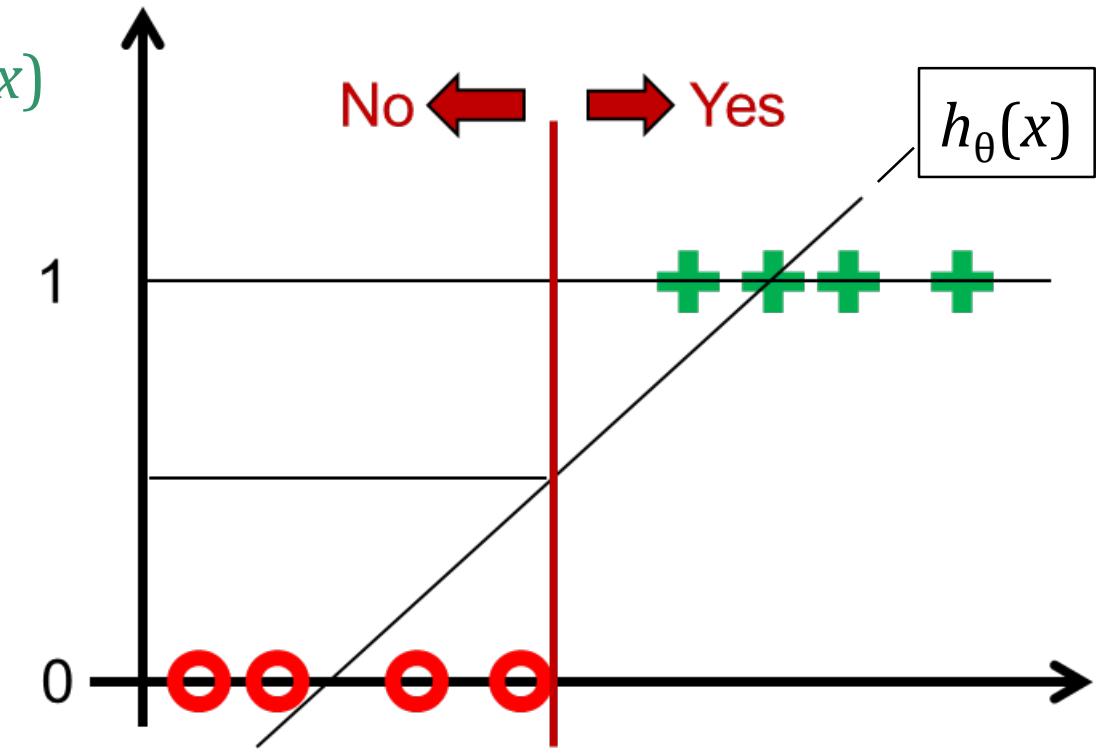
What class does this belong to?





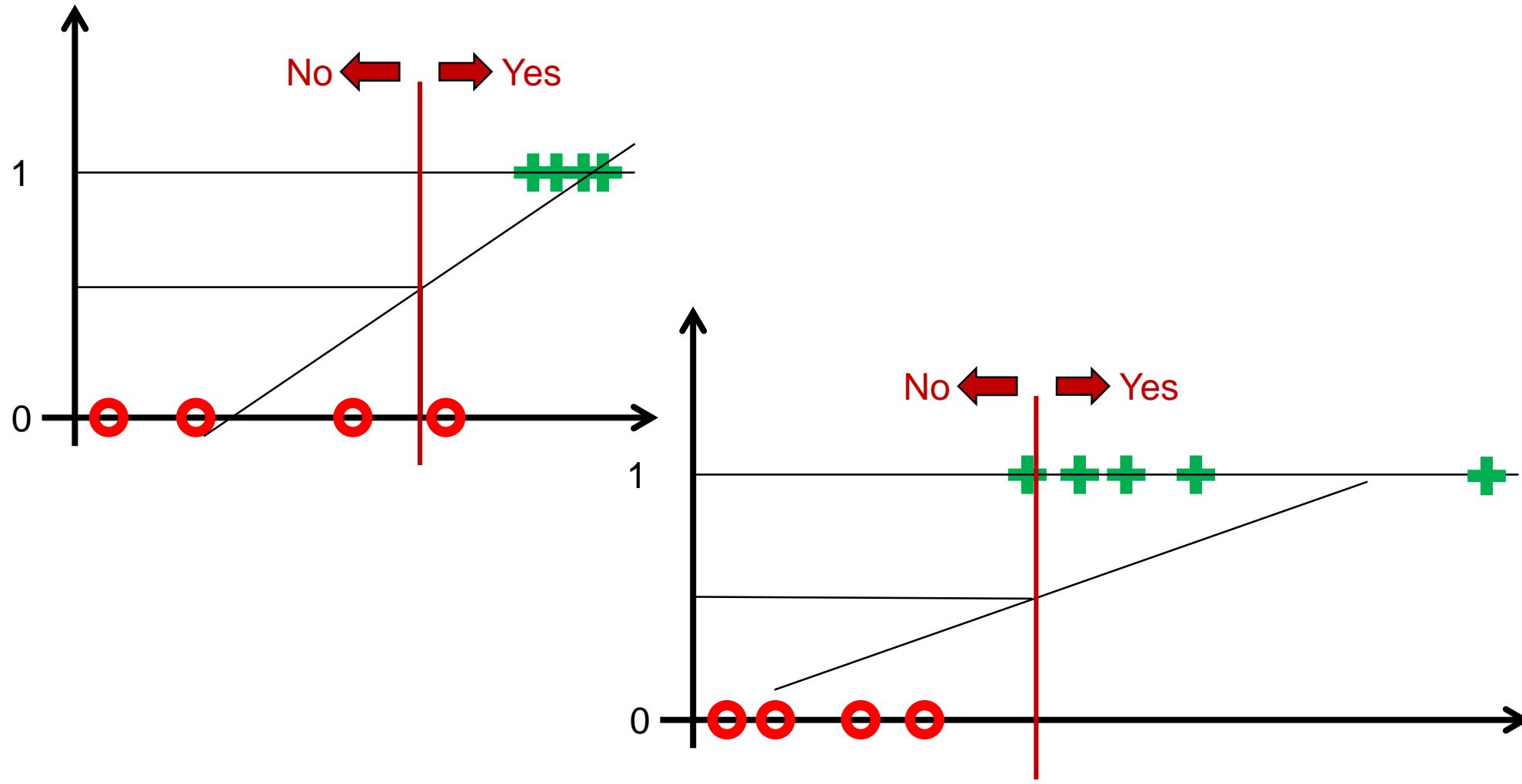
# From Linear Regression to Classification (1)

- Suppose we want to build a Yes/No **classification** model
- We know how to do linear regression:
  - Could encode No as 0 and Yes as 1
  - Perform linear regression to find  $h_{\theta}(x)$
  - Threshold predictions:
    - $h_{\theta}(x) \geq 0.5 \Rightarrow \text{Yes}$
    - $h_{\theta}(x) < 0.5 \Rightarrow \text{No}$
- Unfortunately, using Linear Regression directly doesn't always work very well...





# From Linear Regression to Classification (2)





# From Linear Regression to Classification (3)

- The reason for the problem:

Linear regression parameters are found without taking into account that there are only two 'real' output values, 0/1

A threshold is applied to outputs to convert them to 0/1, but only **after** the regression hyperplane is learned

- The solution:

Incorporate the threshold in the objective function, so that it is taken into account in the cost function and therefore becomes part of the objective to be learned

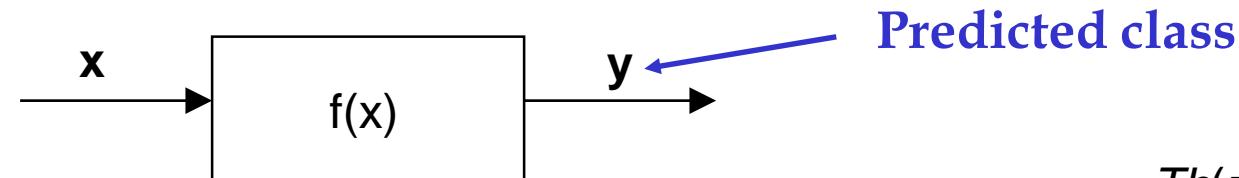
Could use a **hard** or **soft** threshold:

lead to **Linear Classifiers & Logistic Regression**, respectively...



# Linear Classifier

Goal: Given sample  $x$ ,  
find function  $f(x)$  that correctly classifies it



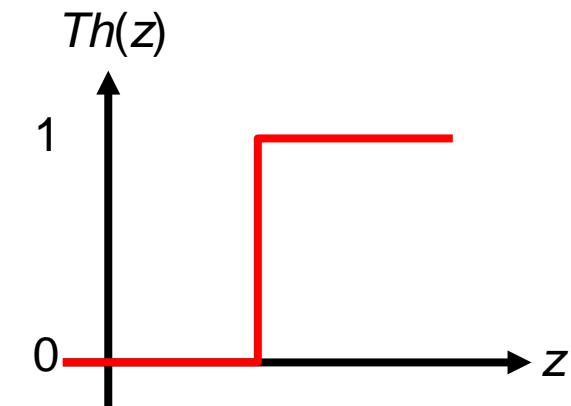
$$f(x) = \theta \cdot x \quad \begin{aligned} f(x) < 0.5: \text{sample is } X \\ f(x) \geq 0.5: \text{sample is } O \end{aligned}$$

As before, add a dummy input variable  $x_0 = 1$

To classify a new sample,  $x$ :

- Calculate the dot product of the weight vector,  $\theta$ , and  $x$
- Apply a **hard threshold**:

$$\text{Threshold}(z) = 1 \text{ if } z \geq 0.5, 0 \text{ otherwise}$$





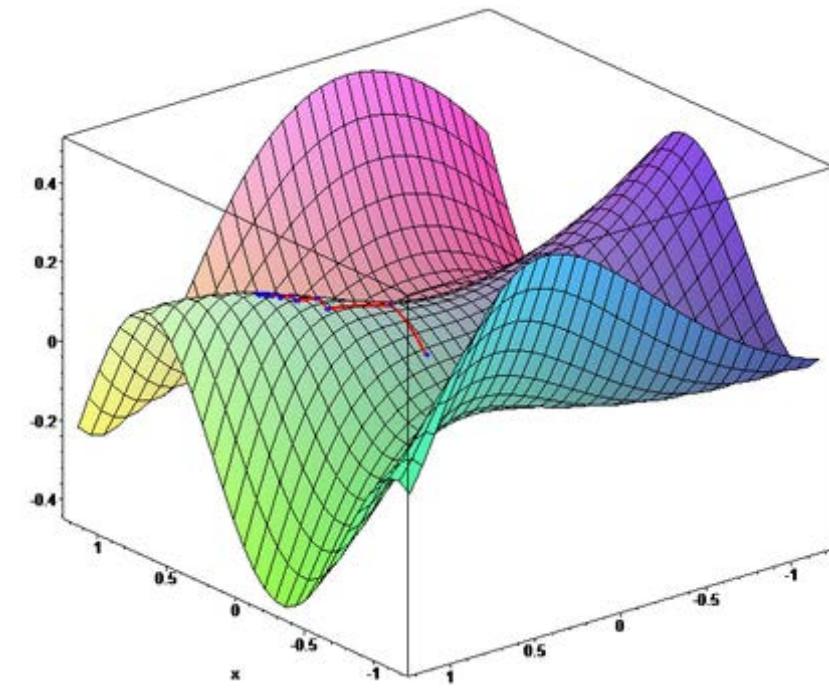
# Building a Linear Classifier

- To build a linear classifier:

$$h_{\theta}(x) = \text{Threshold}(f(x)) = \text{Threshold}(\boldsymbol{\theta} \cdot x)$$

- Find a function  $f(x)$  (i.e. find values for  $\boldsymbol{\theta}$ ) that correctly classifies training data **when put through threshold function**

- Can use Gradient Descent to find values for  $\boldsymbol{\theta}$ 
  - But  $h_{\theta}(x)$  is not differentiable, because of hard threshold function ...





# Building a Linear Classifier: Perceptron Learning Rule

- Because  $h_{\theta}(x)$  is not differentiable, cannot use the exact same approach that we used for Linear Regression

- Instead need **Perceptron Learning Rule** to update  $\theta$  values

- Also called other names

- Usually used with Stochastic Gradient Descent

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\theta}(x)) \cdot x_j \quad \text{for a single training case } (x, y)$$

- Notes:

- For Stochastic GD, picking only one sample, so N=1

- Converges to a solution, provided data linearly separable



# How does Perceptron Learning Rule Compare to Linear Regression Update Rule?

- **Perceptron Learning Rule:**

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\theta}(x)) \cdot x_j \quad \text{for a single example } (x, y)$$

- Linear Regression update rule from last topic:

$$\theta_j \leftarrow \theta_j - \alpha(h_{\theta}(x) - y)x_j$$

Set  $N=1$ , merge equations ...



# Building a Linear Classifier: Perceptron Learning Rule

- Looks just like MLR update rule (previous topic), but behaviour is different as  $h()$  and  $y$  are 0 or 1:
  - Output correct ( $h_{\theta}(x) = y$ ): weights unchanged
  - $y = 1$  but  $h_{\theta}(x) = 0$ : increase  $\theta_i$  if  $x_i$  positive
  - $y = 0$  but  $h_{\theta}(x) = 1$ : decrease  $\theta_i$  if  $x_i$  positive
- To guarantee convergence, need to **decay**  $\alpha$  in proportion to  $1/t$ 
  - For Perceptron, must use smaller values of  $\alpha$  in each successive iteration
  - Unlike Linear Regression update rule: fixed  $\alpha$  because gradients get smaller
  - $t$  is iteration (“time step”)

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\theta}(x)) \cdot x_j$$



# Putting it Together: Linear Classifier Using Stochastic GD with Perceptron Learning Rule

Linear Classifier Learning Algorithm:

**initialise**  $\theta$  to any set of valid initial values

**Initialise**  $\alpha_0$  to some step size

**repeat**  $t=1:T$  times, or until convergence if earlier:

**select** a training example at random

$$\alpha \leftarrow \alpha_0/t$$

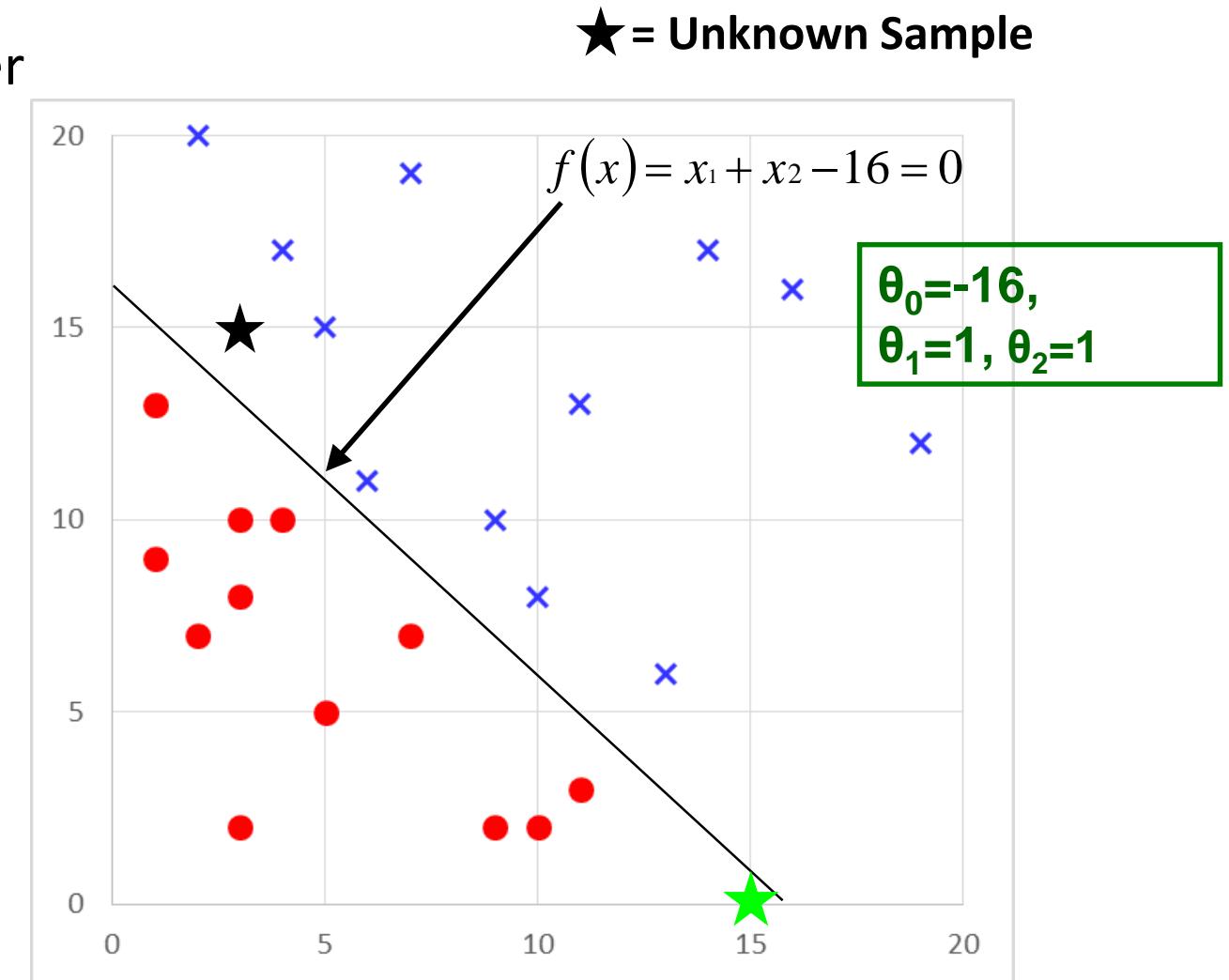
**simultaneously foreach**  $\theta_j$  in  $\theta$  **do**:

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\theta}(x)) \cdot x_j$$



# Linear Classifier: Result

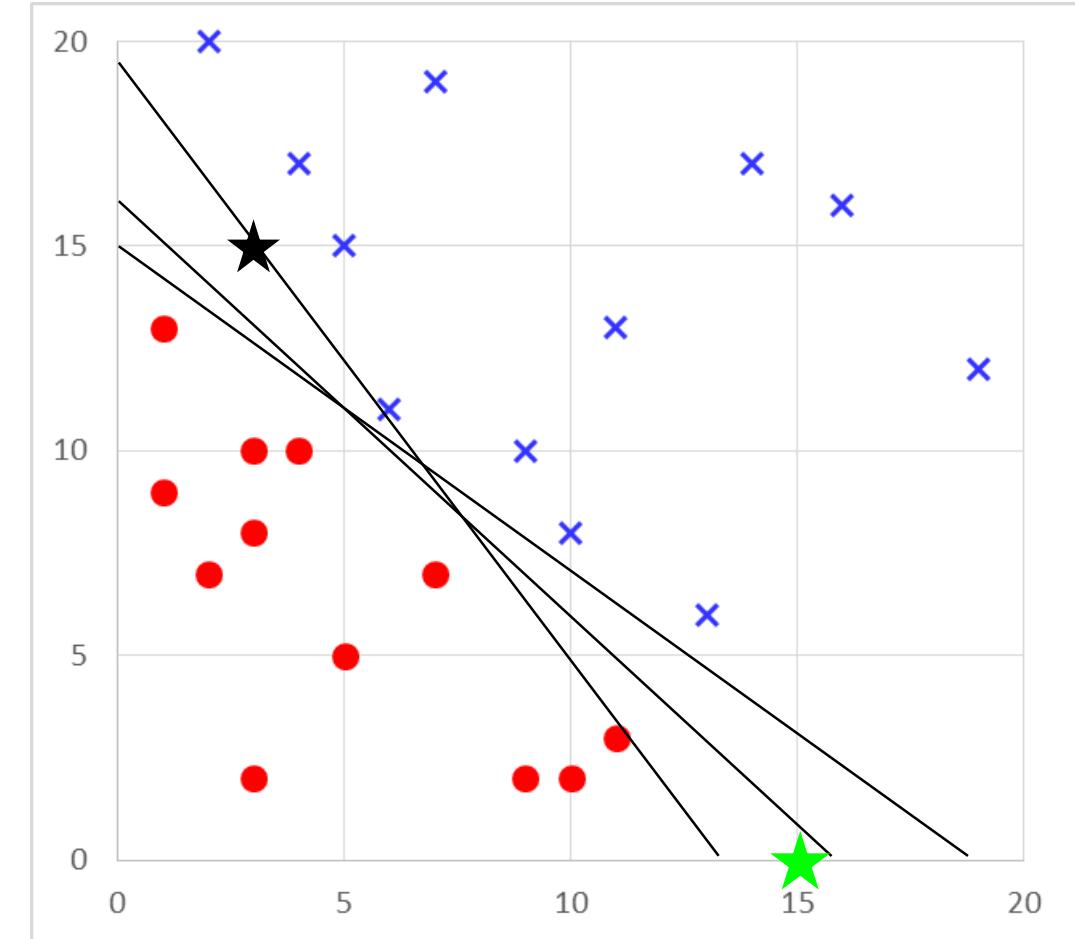
- Example of a successful classifier
- Data are linearly separable:  
can perfectly classify them
- To classify unknown sample  
 $x = (3, 15; ?)$ :  
 $f(x) = 3 + 15 - 16$   
 $f(x) = 2$   
 $f(x) > 0 \Rightarrow \text{sample is } X$
- Another unknown sample  
Green star:  $x = (15, 0)$   
 $f(x) = -1 \Rightarrow \text{sample is } O$
- However, many functions could separate this data....





# Linear Classifier: Which is Best?

- There are many linear classifiers to choose from
  - Which one you find will depend on parameter search settings
- All work equally on linearly separable training data
  - But some will output a different prediction for unknown samples
  - Because Perceptron Rule works with 1/0 values, converges fully as soon boundary line found to fully separate data
  - Where it stops depends on if approaching from “red side” or “blue side”





## Avoiding This Behaviour ...

- Would like to find a boundary line that falls between the two classes, separating them as well as possible
  - To address this, need a different approach:  
**Linear Support Vector Machine**
  - Finds the maximum margin hyperplane separating classes
  - Not within the scope of this module
- The Logistic Classifier (coming up next) uses a “soft margin” that tends to push boundary away from nearest training data
  - However Logistic Classifier is **not** guaranteed to maximize the separating plane, whereas SVM is guaranteed to.



# Soft Thresholds & Logistic Regression

- Instead of hard threshold used in Linear Classifier, could use a *soft* one

Allow  $h_{\theta}(x)$  to take on values in range [0,1]

Have it switch rapidly from 0 to 1 (almost step function)

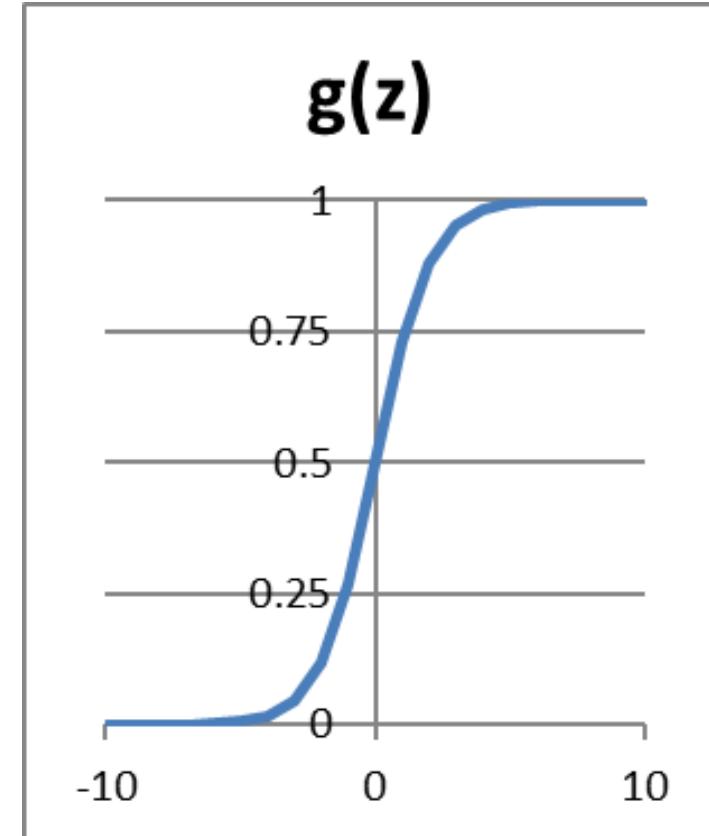
- Go from the linear regression formula:

$$h_{\theta}(x) = \theta \cdot x$$

- To this:

$$h_{\theta}(x) = g(\theta \cdot x) \quad \text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

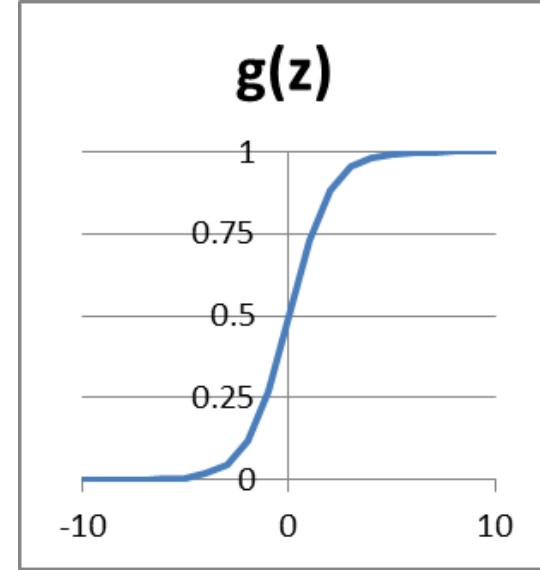
- $g(z)$  is called the sigmoidal or logistic function





# Logistic Regression

- How we interpret the output:  
 $h_{\theta}(x)$  is an estimate of the probability that  $y=1$  for input  $x$ , given the parameters  $\theta$
- As before, can derive a cost function for  $h_{\theta}(x)$  and optimise parameters with gradient descent
  - The cost function is differentiable: can use standard GD as with Linear Regression
- Important: Logistic Regression is used for classification tasks, **not** for regression tasks!





# Logistic Regression Cost Function [1]

- Probability that  $y=1$  (Positive Class) for a case  $x$  is given by  $h_{\theta}(x)$

$$P(y=1 | x) = h_{\theta}(x)$$

- Therefore, probability that  $y=0$  (Negative class) is  $1 - h_{\theta}(x)$

$$P(y=0 | x) = 1 - h_{\theta}(x)$$

- We can combine these equations to cover both  $y=1$  and  $y=0$ :

$$P(y | x) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

- Starting from this, a cost function can be defined, though I won't show its derivation (I include the index  $(i)$  for a training instance):

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^N y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))$$

Extra material



# Logistic Regression Cost Function [2]

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

- Behaviour:
  - As  $h_{\boldsymbol{\theta}}(\mathbf{x})$  tends to the correct value (either  $y=1$  or  $y=0$ ),  $J(\boldsymbol{\theta})$  tends to 0
  - As  $h_{\boldsymbol{\theta}}(\mathbf{x})$  tends to the wrong value,  $J(\boldsymbol{\theta})$  tends towards infinity
- The partial derivative of this cost function is:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

Extra material

- Surprisingly, this looks identical to the linear regression case, though the hypothesis function is different
- Note: There are various definitions and derivations; I am following Stanford ones: <http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/>



# Comparing Linear Classifiers With Hard and Logistic Thresholds

- Both find a hyperplane between classes, with threshold function to convert real number to 0/1
  - Both assume that classes are linearly separable
  - Neither attempt to maximise margin, though sigmoid can push Logistic Regressor boundary out from cases closest to it
- Parameters found with Gradient Descent
  - Logistic: Standard GD approach
  - Hard: Use a different update rule (Perceptron Update Rule) and have to decay  $\alpha$
- Logistic Regression outputs probabilities
  - Reflects uncertainty close to decision boundaries
- Logistic Regression has better convergence and behaves better when data are not linearly separable



# Learning Objectives Review

In this topic you have learned to ...

- Explain the drawbacks of using linear regression directly for classification problems
- Describe and implement approaches to correct this:  
Linear Classifiers and Logistic Regression
- Discuss their characteristics and limitations

Final Note – these are important foundational concepts:  
Limitations of Linear Classifiers addressed with SVMs;  
Logistic Regression leads to ideas in Neural Networks.



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 11: Probabilistic Machine Learning (Part 1)

Prof. Michael Madden  
Chair of Computer Science  
Head of Machine Learning & Data Mining Group  
National University of Ireland Galway



# Learning Objectives

After successfully completing this, you will be able to ...

- Discuss the motivation for handling uncertainty in ML
- Distinguish between prior and conditional probability
- Demonstrate understanding of how to use the axioms of probability and Bayes' rule
- Describe and apply the Naïve Bayes classifier to inductive learning problems
- Show how Bayesian Networks represent influence and independence of variables
- Discuss how BNs can be used for classification & data exploration.



# Structure of Videos for Probabilistic ML Topic

## Week 11

- Topic Part 1A: Review of Probability Basics
- Topic Part 1B: Reasoning with Bayes' Rule

## Week 12

- Topic Part 2A: Probabilistic Classifiers
- Topic Part 2B: Bayesian Networks



# Part 1A: Review of Probability Basics



# Why Consider Uncertainty? (1)

- In a deterministic domain, is there uncertainty?
- What are the sources of uncertainty?
  - **Incomplete knowledge:** lack of relevant facts, partial observations, inaccurate measurements, incomplete domain theory ...
  - **Inability to process:** too complex to use all possible relevant data in computations, or to consider all possible exceptions and qualifications



## Why Consider Uncertainty? (2)

Example: Going to airport – will  $t$  minutes be enough?

Problems:

- incomplete observations (road state, other drivers' plans, etc.)
- noisy sensors (traffic reports)
- uncertainty in action outcomes (flat tyre, etc.)
- immense complexity of modeling traffic

Therefore, purely logical approach either:

- Risks falsehood:  
“90 minutes will get me there on time”, or
- Leads to conclusions that are too weak for decision making:  
“90 minutes will get me there on time, if there's no accident, and it doesn't rain, and my car doesn't break down ...”  
“24 hours will get me there on time” – but requires very long wait





# Techniques for Handling Uncertainty

## Default or Nonmonotonic logic:

- Assume car does not have flat tyre
- Assume 90 minutes is OK unless contradicted by evidence

**Issues:** What assumptions are reasonable?  
How should contradiction be handled?

## Rules with Certainty Factors:

- **90 minutes | $\rightarrow_{0.3}$  get there on time** (ie 30% certainty)
- **Sprinkler | $\rightarrow_{0.99}$  WetGrass**
- **WetGrass | $\rightarrow_{0.7}$  Rain**

**Issues:** Problems with combination: Sprinkler causes rain?

## Probability:

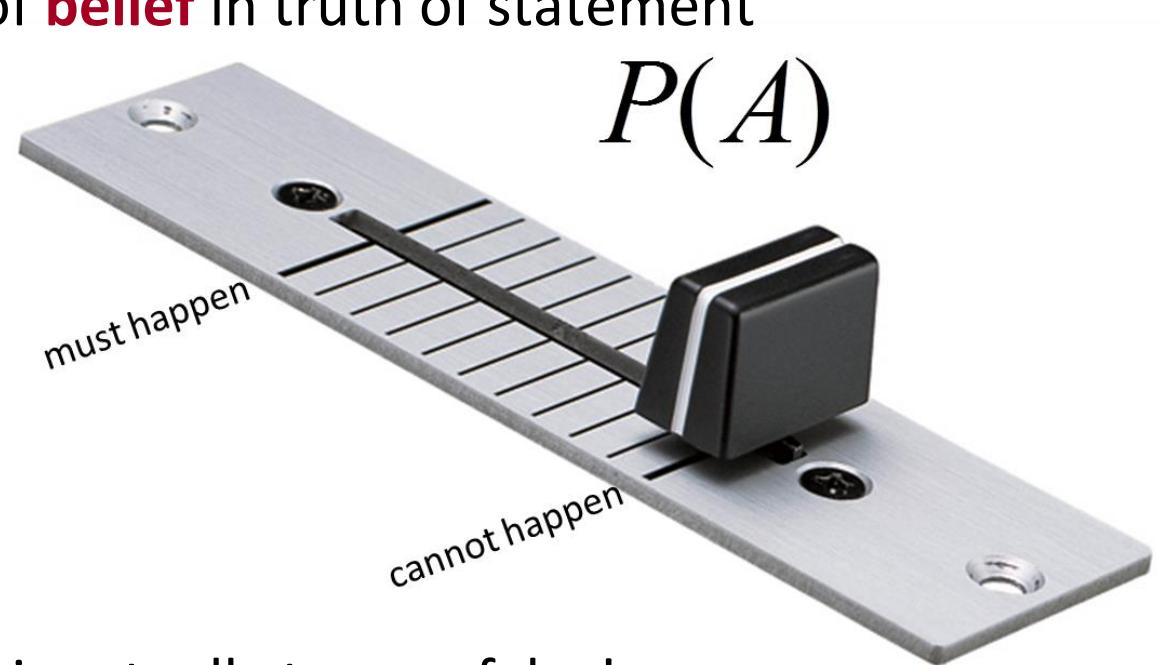
- Model agent's degree of belief
- Given the available evidence,  
90 minutes will get me there on time **with probability 0.4**



# Review of Probability (1)

Probability: a way of **summarising** uncertainties

$0 \leq P(s) \leq 1$ : level of **belief** in truth of statement



**Important:**

- Statement itself is actually true or false!
- Our beliefs may change when we observe new evidence



## Review of Probability (2)

How do we assess probabilities?

- Statistical data, general rules, logical considerations, or combination of evidence
- Although probabilities are personal, they must still be reasonable and **rational**

Example:

- Pick a card ...

Before looking:  $P(\text{Card} = \text{Q}\heartsuit) = 1/52$

After looking:  $P(\text{Card} = \text{Q}\heartsuit) = 0 \text{ or } 1$



# Review: Probability Notation (1)

We will consider discrete random variables:

- **Random variable**: cannot directly control its value, we can just observe it
- **Boolean-valued** random variable: denotes an event, with some degree of uncertainty as to whether it occurs:  
**Cavity: <true, false>**  
 $P(\text{Cavity}=\text{true})$  is also written as  **$P(\text{cavity})$**   
 $P(\text{Cavity}=\text{false})$  is also written as  **$P(\neg\text{cavity})$**
- General case of **discrete random variable**: values in domain are **mutually exclusive and exhaustive**  
**ExamResult: <a, b, c, d, fail>**  
 $P(\text{ExamResult}=\text{a}) = 0.2, \dots$   
 $P(\text{ExamResult}) = <0.2, 0.3, 0.3, 0.15, 0.05>$



## Review: Probability Notation (2)

Probabilities of Values in a Domain Sum to 1

- Provided they are mutually exclusive and exhaustive:  
**P(Elvis=alive) = .01; P(Elvis=dead) = .99**

Elementary proposition:

- Constructed by assigning a value to a random variable:
- Example 1: *Weather = sunny*
- Example 2: *Cavity = false*



## Review: Probability Notation (3)

Complex proposition:

- Formed from **elementary propositions** and logical operators
- Example: *Weather = sunny*  $\vee$  *Cavity = false*

Logical operators and notations used to represent them:

AND:  $a \wedge b$  (mnemonic: similar shape to an A)

OR:  $a \vee b$

NOT:  $\neg a$



## Review: Probability Notation (4)

Atomic event: complete specification of state of the ‘world’

- World: environment/scenario about which we are reasoning
- E.g.: if world consists of just two Boolean variables, *Anna\_Here* and *Bob\_Here*, There are 4 distinct atomic events:
  - $\text{Anna\_Here} = \text{false} \wedge \text{Bob\_Here} = \text{false}$
  - $\text{Anna\_Here} = \text{false} \wedge \text{Bob\_Here} = \text{true}$
  - $\text{Anna\_Here} = \text{true} \wedge \text{Bob\_Here} = \text{false}$
  - $\text{Anna\_Here} = \text{true} \wedge \text{Bob\_Here} = \text{true}$
- Atomic events are **mutually exclusive and exhaustive**: only one is true; their probabilities sum to 1



## Review: Axioms of Probability (1)

1. Probability of any proposition is between 0 and 1:  
 $0 \leq P(a) \leq 1$
2. Necessarily true propositions have probability 1;  
necessarily false propositions have probability 0:  
 $P(\text{false}) = 0, P(\text{true}) = 1$
3. Probability of a disjunction ( $a \vee b$ ) is:  
 $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$   
(Sum Rule)

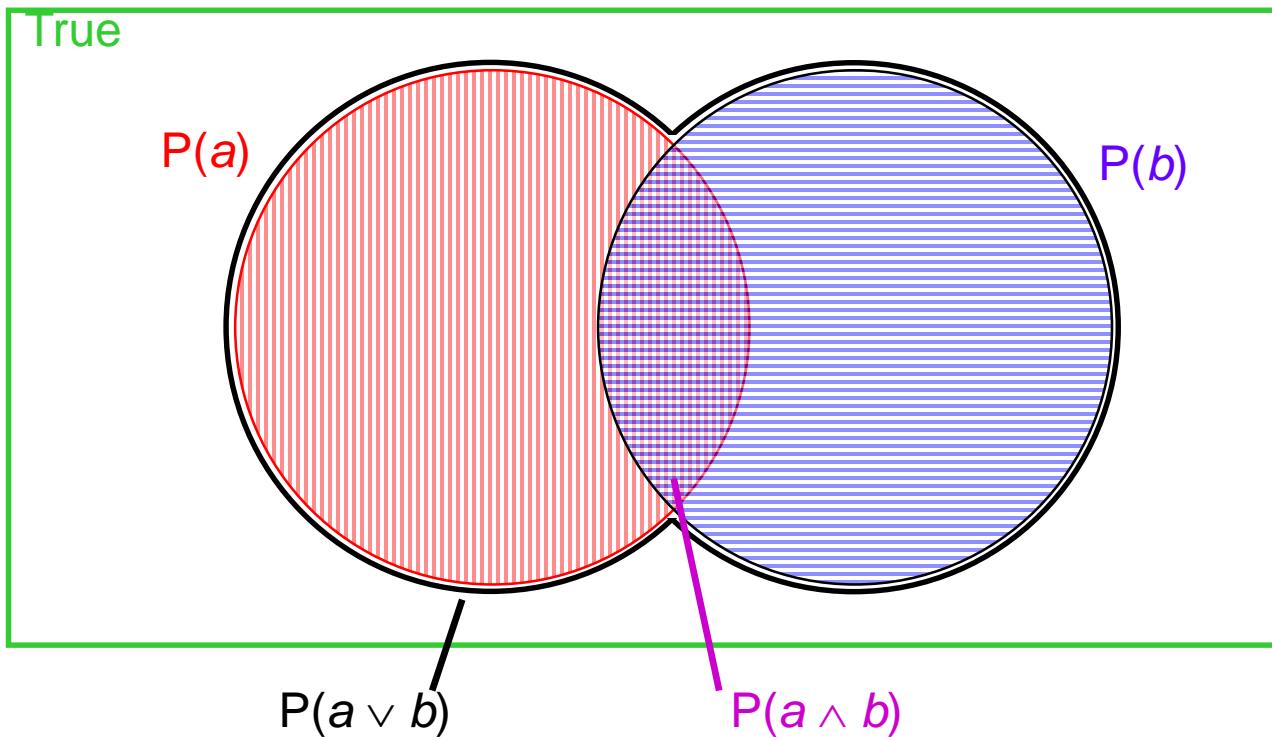
Kolmogorov's  
Axioms: all other  
probability rules  
can be derived  
from these



# Review: Axioms of Probability (2)

Illustration of Sum Rule:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$





# Unconditional and Conditional Probability

## Unconditional Probability:

- $P(a)$  = degree of belief in proposition  $a$  in *absence* of any other information
- Also known as **prior probability**:  
Belief *prior* to arrival of any new information
- Specified as a **probability distribution**:  
 $P(\text{ExamResult}) = <0.2, 0.3, 0.3, 0.15, 0.05>$   
 $P(\text{Anna\_Here}=\text{true}) = .98, P(\text{Anna\_Here}=\text{false}) = .02$

## Conditional Probability:

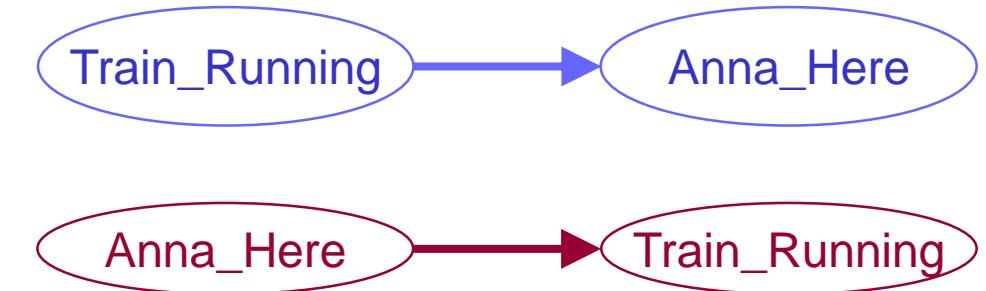
- Also known as **posterior probability**: Belief *post* arrival of new information
- Probability is *conditioned* by other evidence
- $P(\text{Anna\_Here}=\text{false} \mid \text{Train\_Running}=\text{false}) = 0.8$   
“Prob. That Anna is NOT Here is 0.8,  
given that all you know is that the Train is NOT running.”



# Conditional Probability Graphically ...

**Train\_Running** and **Anna\_Here** interact

- Knowing about **Train\_Running** gives us evidence about **Anna\_Here**
- Or vice versa



Diagrams are easiest to understand if we think about causality in drawing them:

- Which causes which?
- Hidden causes?



# Joint Probability Distribution

For a set of random variables, **joint probability distribution** gives probability of every **atomic event** on those variables

- For  $n$  Boolean variables, size is  $2^n$  (exponential in no. of variables)
- Later: represent more compactly because of independence
- **P(Weather, Tennis)**:  $4 \times 2$  matrix of values:

	Weather = sunny	Weather = rain	Weather = cloudy	Weather = snow
Tennis = true	0.144	0.02	0.016	0.02
Tennis = false	0.576	0.08	0.064	0.08



# A Real-Life Joint Probability Distribution ...



**Nate Silver**  @NateSilver538 · 1h

The joint probabilities are as follows, per our Deluxe model.

D Senate + D House: 18%

D Senate + R House: <1%

R Senate + D House: 68%

R Senate + R House: 14%

So still better than a 30% chance that \*either\* the House or the Senate will result in an upset tonight. Pretty exciting!



# Independence

New evidence may be irrelevant:

$$\begin{aligned} & P(\text{Anna\_Here}=\text{false} \mid \text{Train\_Running}=\text{False}, \text{Exam\_Result}=a) \\ & = P(\text{Anna\_Here}=\text{false} \mid \text{Train\_Running}=\text{False}) \\ & = 0.8 \end{aligned}$$

This indicates independence between variables:

- Exam\_Result independent of Anna\_Here
- Also known as **absolute independence**
- Diagram: no arc

Such simplifications very important

- Can greatly reduce the number of combinations we need to consider

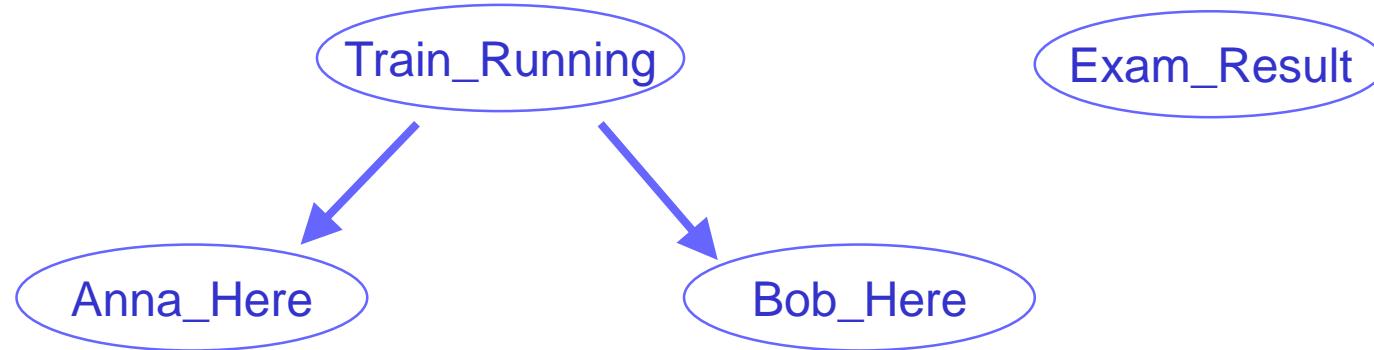




# Conditional Independence (1)

Conditional Independence is different from absolute independence

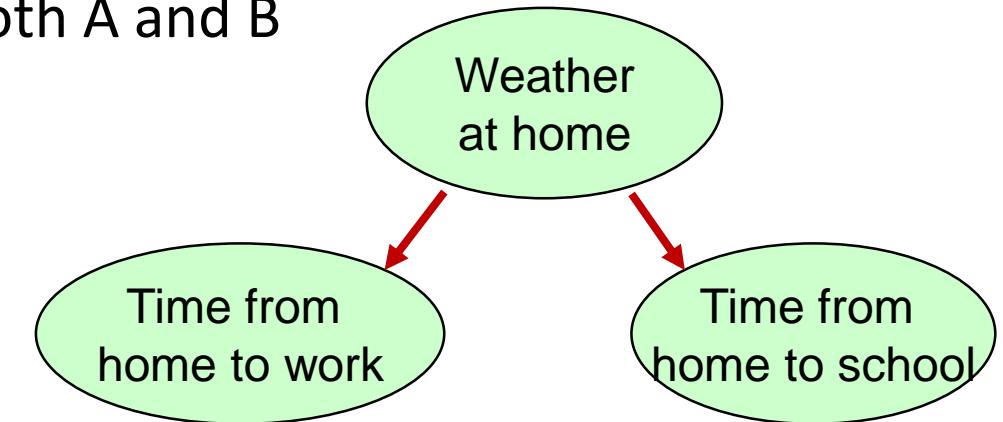
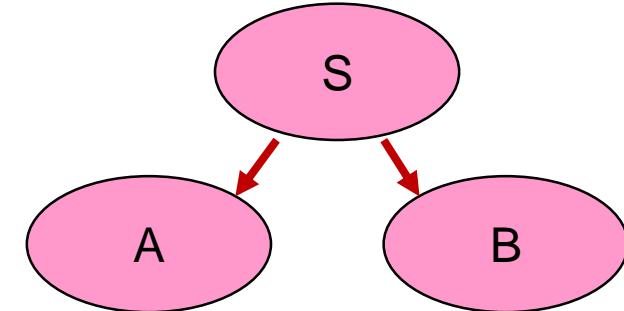
- **Anna** and **Bob** both take the train, so if Anna is not here, it is **more likely** that Bob is not here
- Bob being here is **not completely independent** of Anna being here; they are both dependent on the Train running
- Anna\_Here **conditionally independent** of Bob\_Here **given** Train\_Running
- Graphically:





# Conditional Independence (2)

- Note also:
  - We say that A and B are ***d-separated*** by S (*D*=directional)
  - If we know value of S, then knowing value of A does not give us any additional information about value of B
  - However, in the absence of information about the separating variable S, knowing A **does** give us information about B
  - Relates to S being the root cause of both A and B





# Probability Formulae (1)

## Absolute Independence:

- If two variables are **independent**, the probability of **both** happening is the **product** of their individual probabilities

$$P(A \wedge B) = P(A) P(B) \Leftrightarrow A \text{ independent of } B$$

- Example:

$$P(\text{Suit}=\heartsuit) = 1/4. \quad P(\text{Value}=Q) = 1/13.$$

$$P(\text{Suit}=\heartsuit \wedge \text{Value}=Q) = 1/52.$$

## Conditional Independence:

- If A is **conditionally independent** of B given X:

$$P(A \wedge B | X) = P(A | X) P(B | X) \Leftrightarrow A \text{ cond. indep. of } B \text{ given } X$$

If X is known, knowledge of A's value will not affect opinion of B's value, so can apply same rule as for absolute independence



## Probability Formulae (2)

Product Rule (conjunctions):

$$P(A \wedge B) = P(A | B) P(B) = P(B | A) P(A)$$

Example:

$$\begin{aligned} P(\checkmark | \text{storm}) &= 0.6, P(\text{storm}) = 0.01 \\ \Rightarrow P(\checkmark \wedge \text{storm}) &= 0.006 \end{aligned}$$

If lightning occurs in 60% of storms,  
and a storm occurs 1% of the time,  
then lightning and storm together occur 0.6% of the time

Conditional Probability of two propositions:

$$P(a | b) = P(a \wedge b) / P(b), \text{ for } P(b) > 0$$



## Probability Formulae (3)

Theorem of Total Probability:

- If domain of A is  $\langle a_1, a_2, \dots, a_n \rangle$ , then:

$$P(B) = \sum_n P(B | a_i) P(a_i)$$

$$P(\text{✓}) = P(\text{✓} | \text{storm}) P(\text{storm}) + P(\text{✓} | \neg \text{storm}) P(\neg \text{storm})$$

Total probability of lightning occurring =  
prob. of it *with* a storm + prob. of it *without* a storm

Bayes' Rule:

$$P(B | A) = P(A | B) P(B) / P(A)$$

- Follows from Product Rule
- Allows us to reason about causes when we have observed effects



---

# Part 1B: Reasoning with Bayes' Rule



# Reasoning with Bayes' Rule

Bayes' Rule:

$$P(b | a) = P(a | b) P(b) / P(a)$$

Easily derived from Product Rule:

$$P(a \wedge b) = P(a | b) P(b) = P(b | a) P(a)$$

$$\text{Divide by } P(a): P(a | b) P(b) / P(a) = P(b | a)$$

Applies to variables as well as propositions:

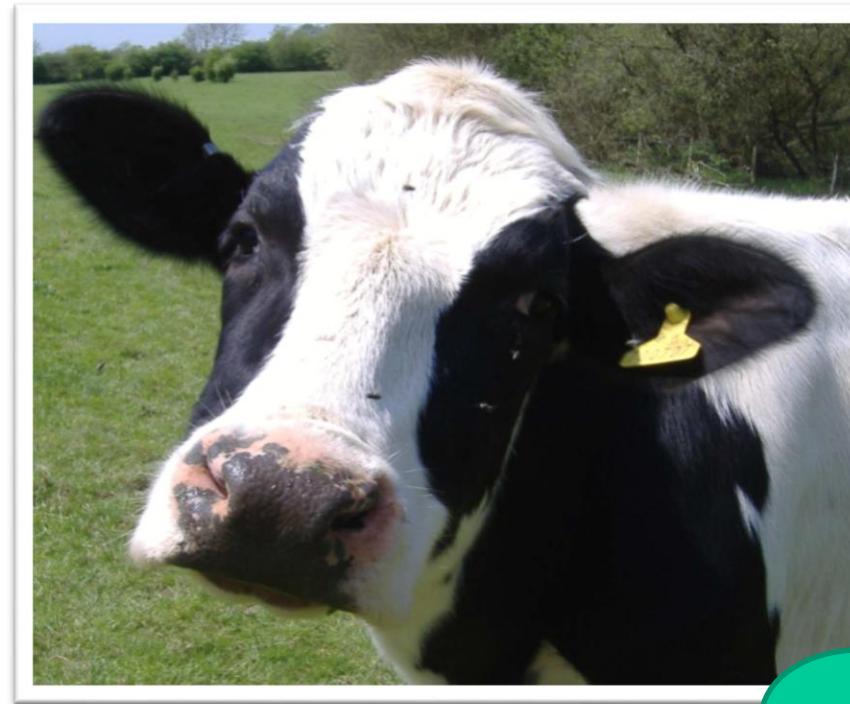
$$P(B | A) = \frac{P(A | B) P(B)}{P(A)}$$

What's the point?

Allows us to reason about an **unobserved cause** (B) when we have observed the **effect** (A)



*Probably not the*  
Rev. Thomas Bayes  
1702-1761



x 20



$$P(\text{cow} \mid \text{caused by cow}) = P(\text{cow causes} \mid \text{caused by cow}) \times \frac{P(\text{cow})}{P(\text{caused by cow})}$$

1



$P(\text{casket} \mid \text{caused by cow})$

$$= P(\text{cow causes casket}) \times P(\text{cow})$$

$P(\text{casket} \mid \text{caused by shark})$

$$= P(\text{shark causes casket}) \times P(\text{shark})$$



---

A disease affects 1 in 1000 people

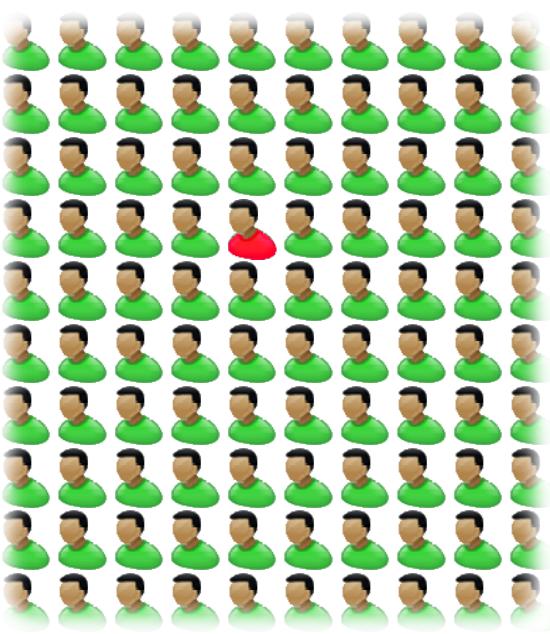
A Person Tests +

Test is 99% accurate

Probably  
have it?

Probably  
don't?



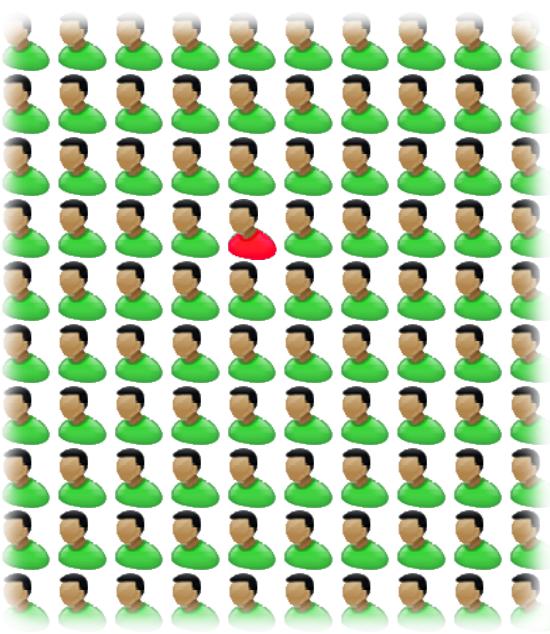


Disease:  
1 in 1000

100

100,000

99,900



100,000

Disease:  
1 in 1000

100

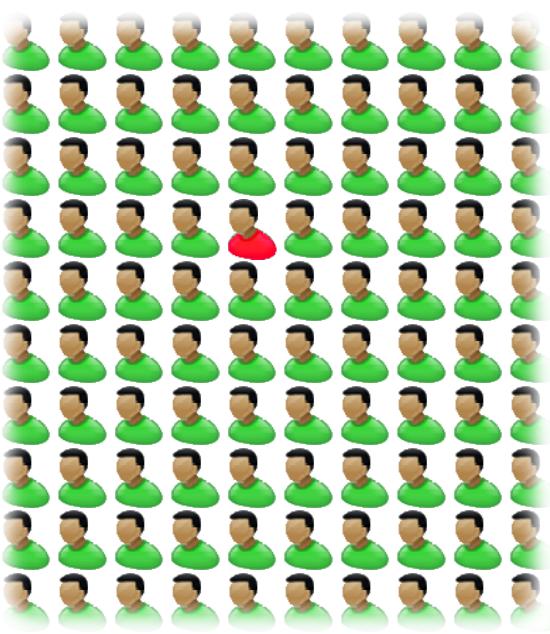
99,900

99%  
1%

99

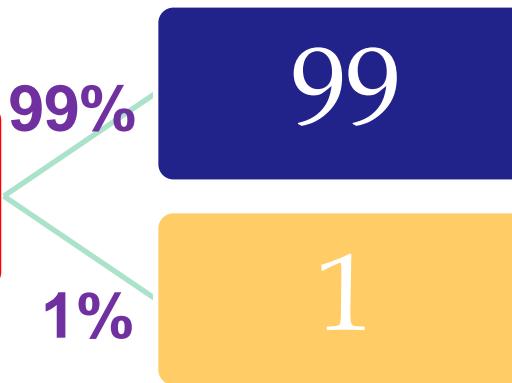
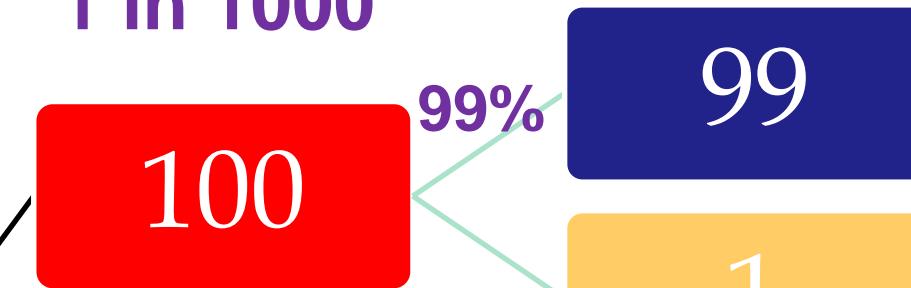
1

Test +



Disease:  
1 in 1000

100,000



Test +

Test +

1098 people test +

99 of them actually +

=>  $P(\text{Disease} \mid \text{Test} +) = 9\%$



# Bayes' Rule: Example

Patient has sore throat: caused by influenza?

- **P(sore\_throat | flu) = 0.75**  
75% probability of a patient with flu having a sore throat,  
based on experience of patients with flu
  - **P(sore\_throat) = 0.1:**  
1 in 10 patients have sore throat, based on observations  
at the surgery
  - **P(flu) = 0.02:**  
1 in 50 patients have flu
- => **P(flu | sore\_throat) =  $.75 \times .02 / .1 = 0.15$**   
15% probability based on this evidence



# Bayes' Rule with Normalisation (1)

Alternative formulation of Bayes' Rule

- Avoid needing to know **prior probability of evidence**:  
In this example, **P(sore\_throat)**
- Instead, compute posterior probability for each value of **query variable**, and normalise:

$$P(\text{flu} \mid \text{sore\_th}) = P(\text{sore\_th} \mid \text{flu}) \times P(\text{flu}) / P(\text{sore\_th})$$

$$P(\neg\text{flu} \mid \text{sore\_th}) = P(\text{sore\_th} \mid \neg\text{flu}) \times P(\neg\text{flu}) / P(\text{sore\_th})$$



# Bayes' Rule with Normalisation (2)

Some observations:

- The probabilities  $P(\text{flu} \mid \dots)$  and  $P(\neg\text{flu} \mid \dots)$  must sum to 1
- Both times we are dividing by same term,  $P(\text{sore\_th})$ , so we can eliminate it by *normalising*  $P(\text{flu} \mid \dots)$  and  $P(\neg\text{flu} \mid \dots)$
- Probability of not having flu is easily found:  
 $P(\neg\text{flu}) = 1 - P(\text{flu})$



# Bayes' Rule with Normalisation (3)

Applying this to the Influenza example:

- Use  $\alpha$  to denote the **normalisation constant**

(this will be equal to  $1/P(\text{sore\_th})$ )

$$P(\text{flu} \mid \text{sore\_th}) = \alpha \times P(\text{sore\_th} \mid \text{flu}) \times P(\text{flu})$$

$$P(\neg \text{flu} \mid \text{sore\_th}) = \alpha \times P(\text{sore\_th} \mid \neg \text{flu}) \times P(\neg \text{flu})$$

- We already have:

$$P(\text{sore\_throat} \mid \text{flu}) = 0.75, \quad P(\text{flu}) = 0.02$$

$$P(\neg \text{flu}) = 1 - P(\text{flu}) = 0.98$$

- In this case, we also need to know  $P(\text{sore\_th} \mid \neg \text{flu})$ , probability of having a sore throat when you don't have flu:  
From observations at the surgery, this is **0.087**



# Bayes' Rule with Normalisation (4)

Can then calculate:

$$P(\text{flu} \mid \text{sore\_th}) = \alpha \times 0.75 \times 0.02 = \alpha 0.015$$

$$P(\neg\text{flu} \mid \text{sore\_th}) = \alpha \times 0.087 \times 0.98 = \alpha 0.08526$$

To eliminate  $\alpha$ , normalise the numbers (sum is 0.10026):

$$P(\text{flu} \mid \text{sore\_th}) = 0.015 / 0.10026 = 0.15 \text{ (as before)}$$

$$P(\neg\text{flu} \mid \text{sore\_th}) = 0.08526 / 0.10026 = 0.85$$

General form of Bayes' Rule with Normalisation:

$$P(Y \mid X) = \alpha P(X \mid Y) P(Y)$$

where  $\alpha$  is constant to make  $P(Y|X)$  entries sum to 1



# Bayes' Rule with Normalisation (5)

General form of Bayes' Rule with Normalisation:

$$P(Y | X) = \alpha P(X | Y) P(Y) \text{ where } \alpha \text{ is normalisation constant}$$

The Product Rule that we saw earlier states that:

$$P(X \wedge Y) = P(X | Y) P(Y)$$

If we combine both equations, we see that:

$$P(Y | X) = \alpha P(X \wedge Y)$$

Therefore, to calculate the probability of **X given Y**,  
we can calculate probability of **X and Y**  
and normalise result to get the final answer.



# Bayes' Rule: Discussion

Why not just measure  $P(\text{flu} \mid \text{sore\_throat})$ ?

- This **diagnostic probability** could be assessed from observations, as the other values are

Rationale: causal probabilities are more robust

- If there's a flu outbreak,  $P(\text{flu})$  will increase
- $P(\text{flu} \mid \text{sore\_throat})$  is difficult to re-assess directly: counts from before outbreak will not be applicable
- However,  $P(\text{flu})$  can be measured again and  $P(\text{sore\_throat} \mid \text{flu})$  is **unchanged**
- From Bayes' Rule, see that  $P(\text{flu} \mid \text{sore\_throat})$  should increase proportionately with  $P(\text{flu})$

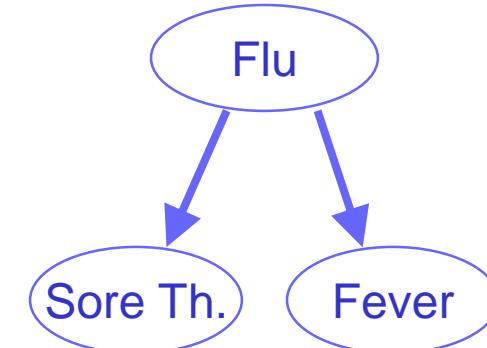


# Bayes' Rule: Combining Evidence

Suppose patient also has fever: effect on reasoning?

$$P(\text{flu} \mid \text{sore\_th} \wedge \text{fever}) = \alpha P(\text{sore\_th} \wedge \text{fever} \mid \text{flu}) P(\text{flu})$$

- Problem: To apply Bayes' Rule directly, need *joint probability*:  $\text{sore\_th} \wedge \text{fever}$



Observation:  $\text{sore\_th}$  **conditionally independent** of  $\text{fever}$  given  $\text{flu}$

- Can use Conditional Independence formula  
$$P(A \wedge B \mid X) = P(A \mid X) P(B \mid X)$$
- Therefore, only need individual conditional probabilities:  
$$P(\text{flu} \mid \text{sore\_th} \wedge \text{fever}) = \alpha P(\text{sore\_th} \mid \text{flu}) P(\text{fever} \mid \text{flu}) P(\text{flu})$$
- Just calculate both using Bayes' Rule and multiply them.



# Bayesian Updating

- Bayes' Rule can be applied iteratively, for sequential testing

- Flu example can operate in this way:

Initially, have prior probability of flu:  $P(\text{flu})$

Then, check whether patient has sore throat and compute

$P(\text{flu} \mid \text{sore\_th})$  by applying Bayes' Rule to update  $P(\text{flu})$

Then, check whether patient has a fever and compute

$P(\text{flu} \mid \text{sore\_th} \wedge \text{fever})$  by applying Bayes' Rule to  $P(\text{flu})$

- It is easy to check that this gives the same result in the case where we assume `sore_th` and `fever` are independent



NUI Galway  
OÉ Gaillimh

# Machine Learning

## Week 12: Probabilistic Machine Learning (Part 2)

Prof. Michael Madden  
Chair of Computer Science  
Head of Machine Learning & Data Mining Group  
National University of Ireland Galway



# Part 2A: Probabilistic Classifiers



# Naïve Bayes Classifier (1)

- Simplest form of Bayesian classifier

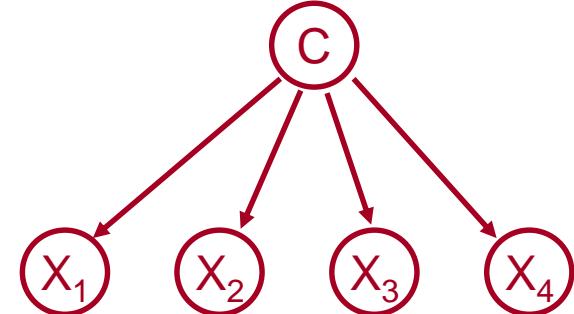
A node for each variable in domain

**C** is **class** node

Other nodes are **evidence** nodes

Alternative view: C is Cause; others are Effects

**Arc** from class node to each evidence node



- “Naïve” assumption:

Evidence nodes assumed to be conditionally independent of each other, given the class node

Simplifies calculations, but may be incorrect



## Naïve Bayes Classifier (2)

- For each arc, need set of probabilities (next slide):

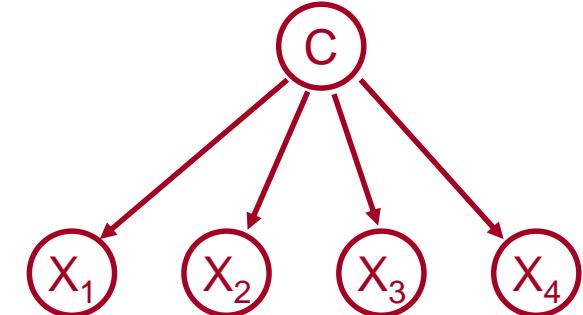
$$P(X_1=\text{true} \mid C=\text{true}) = 1 - P(X_1=\text{false} \mid C=\text{true})$$

$$P(X_1=\text{true} \mid C=\text{false}) = 1 - P(X_1=\text{false} \mid C=\text{false})$$

- To classify a new instance:

- 1: For each possible value of the class node,

Calculate the probability of that class given the values of the other attributes  
(Use Bayes' Rule with Normalisation, assuming Cond. Indep.)



$$P(c_1 \wedge x_1 \wedge x_2 \wedge \dots \wedge x_n) = P(c_1) \prod_i P(x_i|c_1)$$

- 2: Normalise the probabilities of each class; select the most probable.

- Note:  $P(c_j \mid x_1 \wedge \dots \wedge x_n) = P(c_j \wedge x_1 \wedge \dots \wedge x_n) / P(x_1 \wedge \dots \wedge x_n)$   
 $P(x_1 \wedge \dots \wedge x_n)$  is fixed  $\Rightarrow P(c_j \mid \dots) \propto P(c_j \wedge \dots)$



## Naïve Bayes Classifier (3)

- Training a Naïve Bayes Classifier:
  - Arc structure is fixed
  - Just estimate arc probabilities from the training data!
  - Probabilities for one arc: “**Conditional Distribution**”
- To estimate probabilities, can simply use frequencies from training data:

$$P(X=x_1 \mid C=c_1) = N_{x_1 c_1} / N_{c_1}$$

$N_{c_1}$  = count of cases where  $C=c_1$

$N_{x_1 c_1}$  = counts of cases where  $X=x_1$  and  $C=c_1$

- **Laplacian Smoothing** avoids 0 probabilities

Effectively adds  $m$  'virtual observations' with everything seen once

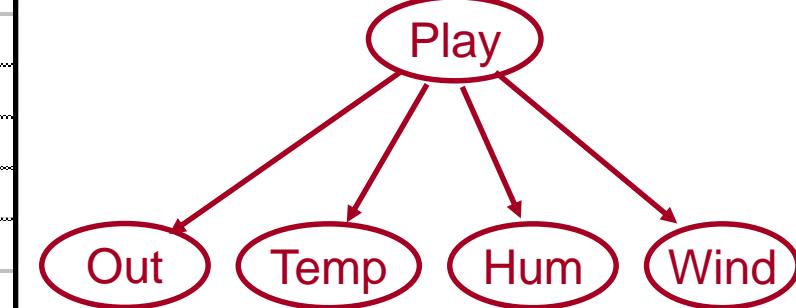
$$(N_{x_1 c_1} + m) / (N_{c_1} + m |X|)$$

Typically use  $m=1$



# Naïve Bayes Example: Tennis (1)

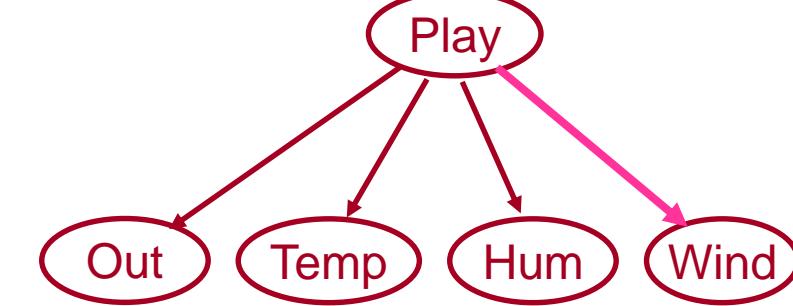
ID	Outlook	Temp	Humidity	Windy	Play?
A	sunny	hot	high	false	no
B	sunny	hot	high	true	no
C	overcast	hot	high	false	yes
D	rainy	mild	high	false	yes
E	rainy	cool	normal	false	yes
F	rainy	cool	normal	true	no
G	overcast	cool	normal	true	yes
H	sunny	mild	high	false	no
I	sunny	cool	normal	false	yes
J	rainy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





# Naïve Bayes Example: Tennis (2)

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



$$P(\text{Wind}=t \mid \text{Play}=y) = 3/9$$

$$P(\text{Wind}=f \mid \text{Play}=y) = 6/9$$

$$P(\text{Wind}=t \mid \text{Play}=n) = 3/5$$

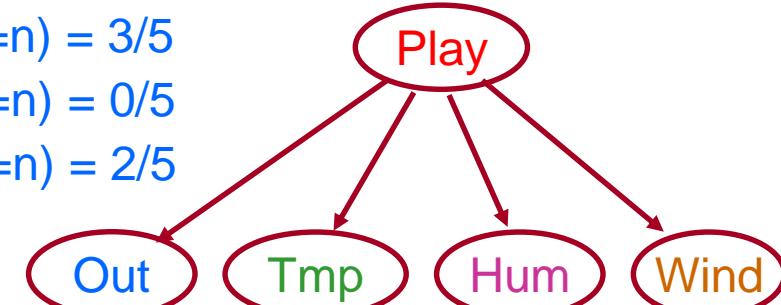
$$P(\text{Wind}=f \mid \text{Play}=n) = 2/5$$

Now do the same for the other attributes ...



# Naïve Bayes Example: Tennis (3)

$$\begin{array}{ll} P(\text{Out}=s \mid \text{Play}=y) = 2/9 & P(\text{Out}=s \mid \text{Play}=n) = 3/5 \\ P(\text{Out}=o \mid \text{Play}=y) = 4/9 & P(\text{Out}=o \mid \text{Play}=n) = 0/5 \\ P(\text{Out}=r \mid \text{Play}=y) = 3/9 & P(\text{Out}=r \mid \text{Play}=n) = 2/5 \end{array}$$



$$\begin{array}{ll} P(\text{Tmp}=h \mid \text{Play}=y) = 2/9 & P(\text{Tmp}=h \mid \text{Play}=n) = 2/5 \\ P(\text{Tmp}=m \mid \text{Play}=y) = 4/9 & P(\text{Tmp}=m \mid \text{Play}=n) = 2/5 \\ P(\text{Tmp}=c \mid \text{Play}=y) = 3/9 & P(\text{Tmp}=c \mid \text{Play}=n) = 1/5 \end{array}$$

$$\begin{array}{ll} P(\text{Hum}=h \mid \text{Play}=y) = 3/9 & P(\text{Hum}=h \mid \text{Play}=n) = 4/5 \\ P(\text{Hum}=n \mid \text{Play}=y) = 6/9 & P(\text{Hum}=n \mid \text{Play}=n) = 1/5 \end{array} \quad \begin{array}{l} P(\text{Play}=y) = 9/14 \\ P(\text{Play}=n) = 5/14 \end{array}$$

$$\begin{array}{ll} P(\text{Wind}=t \mid \text{Play}=y) = 3/9 & P(\text{Wind}=t \mid \text{Play}=n) = 3/5 \\ P(\text{Wind}=f \mid \text{Play}=y) = 6/9 & P(\text{Wind}=f \mid \text{Play}=n) = 2/5 \end{array} \quad \text{(prior probabilities)}$$



## Naïve Bayes Example: Tennis (4)

- Now classify new instance: **sunny**, **cool**, **high**, **true**: Play?

Play is **y** or **n**. Evaluate probability of each given data:

$$P(\text{Play}=y \wedge \text{Out}=s \wedge \text{Tmp}=c \wedge \text{Hum}=h \wedge \text{Wind}=t)$$

$$\begin{aligned} &= P(\text{Play}=y) \times P(\text{Out}=s \mid \text{Play}=y) \times P(\text{Tmp}=c \mid \text{Play}=y) \\ &\quad \times P(\text{Hum}=h \mid \text{Play}=y) \times P(\text{Wind}=t \mid \text{Play}=y) \end{aligned}$$

$$= 9/14 \times 2/9 \times 3/9 \times 3/9 \times 3/9 = 1/189$$

$$P(\text{Play}=n \wedge \text{Out}=s \wedge \text{Tmp}=c \wedge \text{Hum}=h \wedge \text{Wind}=t)$$

$$= 5/14 \times 3/5 \times 1/5 \times 4/5 \times 3/5 = 18/875$$

$$\begin{aligned} &P(c_1 \wedge x_1 \wedge x_2 \wedge \dots \wedge x_n) \\ &= P(c_1) \prod_i P(x_i \mid c_1) \end{aligned}$$

Normalise:

$$P(\text{Play}=y \mid \text{data}) = 125/611 = 20.5\%$$

$$P(\text{Play}=n \mid \text{data}) = 486/611 = 79.5\%$$

- Conclusion: more likely **NOT** to play tennis today.



# Bayesian Spam Filter (1)

Classify messages as spam/ham: Naïve Bayes often used

"Bag of words" representation of messages: Each word in a message is a feature

You are consulting for PhotoGram, a web service for sharing photos with short captions. They wish to automatically identify postings that are spam from the caption text. They have the following small set of captions that are spam and legitimate:

*Spam:*

- “click this link”
- “weight drugs link”
- “drugs news here”

*Legitimate:*

- “puppy sleeping today”
- “good luck puppy”
- “good sleeping”
- “news meeting today”

Using a Naïve Bayes classifier, compute the probability of the following two messages being spam: (1) “weight drugs news”; (2) “puppy news”. Show all steps in your computation and explain any assumptions you make.



## Bayesian Spam Filter (2)

**SPAM**

click this link

weight drugs link

drugs news here

**¬SPAM**

puppy sleeping today

good luck puppy

good sleeping

news meeting today

$$P(\text{SPAM}) = ?$$

$$P(\neg\text{SPAM}) = ?$$

(no smoothing)

$$P(\text{"news"} | \text{SPAM}) = ?$$

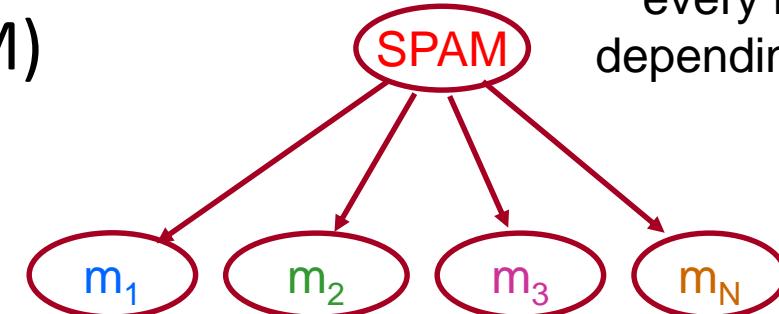
$$P(\text{"news"} | \neg\text{SPAM}) = ?$$



## Bayesian Spam Filter (2)

- For a message  $M = (m_1 m_2 m_3 \dots m_N)$ , want to compute  $P(SPAM | M)$

[New classifier created for every new message, depending on words in it]



- Compute:

$$P'(SPAM | M) = P(SPAM) P(m_1 | SPAM) P(m_2 | SPAM) \dots$$

$$P'(HAM | M) = P(HAM) P(m_1 | HAM) P(m_2 | HAM) \dots$$

And normalise

- Examples:

- $M$  is “puppy news”
- $M$  is “weight drug news”

How do the calculations change if we use Laplacian smoothing?  
See spreadsheet.



## Part 2B: Bayesian Networks



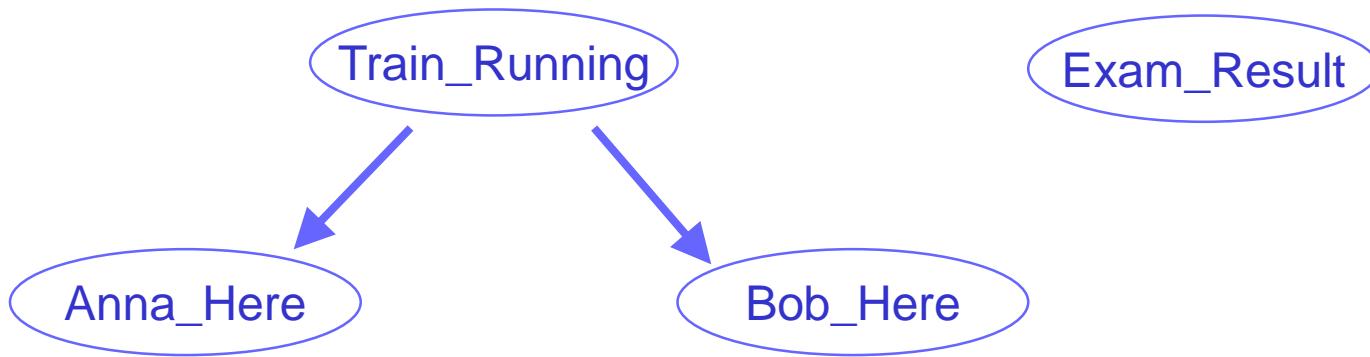
# Bayesian Networks: Syntax (1)

- Graphical notation for conditional independence assertions
  - Allows compact specification of full joint distribution
  - Consists of **Topology + Probabilities**
- Topology (Structure):
  - One node for every variable in domain
  - Arcs between nodes, forming a **directed acyclic graph (DAG)**:
  - Roughly speaking, arc  $X \rightarrow Y$  means “**X directly influences Y**”
- Probabilities:
  - A **local conditional distribution** for each node given its parents:
  - $P(X_i | \text{Parents}(X_i))$
  - Represented as Conditional Probability Table (**CPT**) giving the distribution over  $X_i$  for each combination of its parent values



# Bayesian Networks: Syntax (2)

- Key point:  
Topology of network describes conditional independence assertions
- Example:

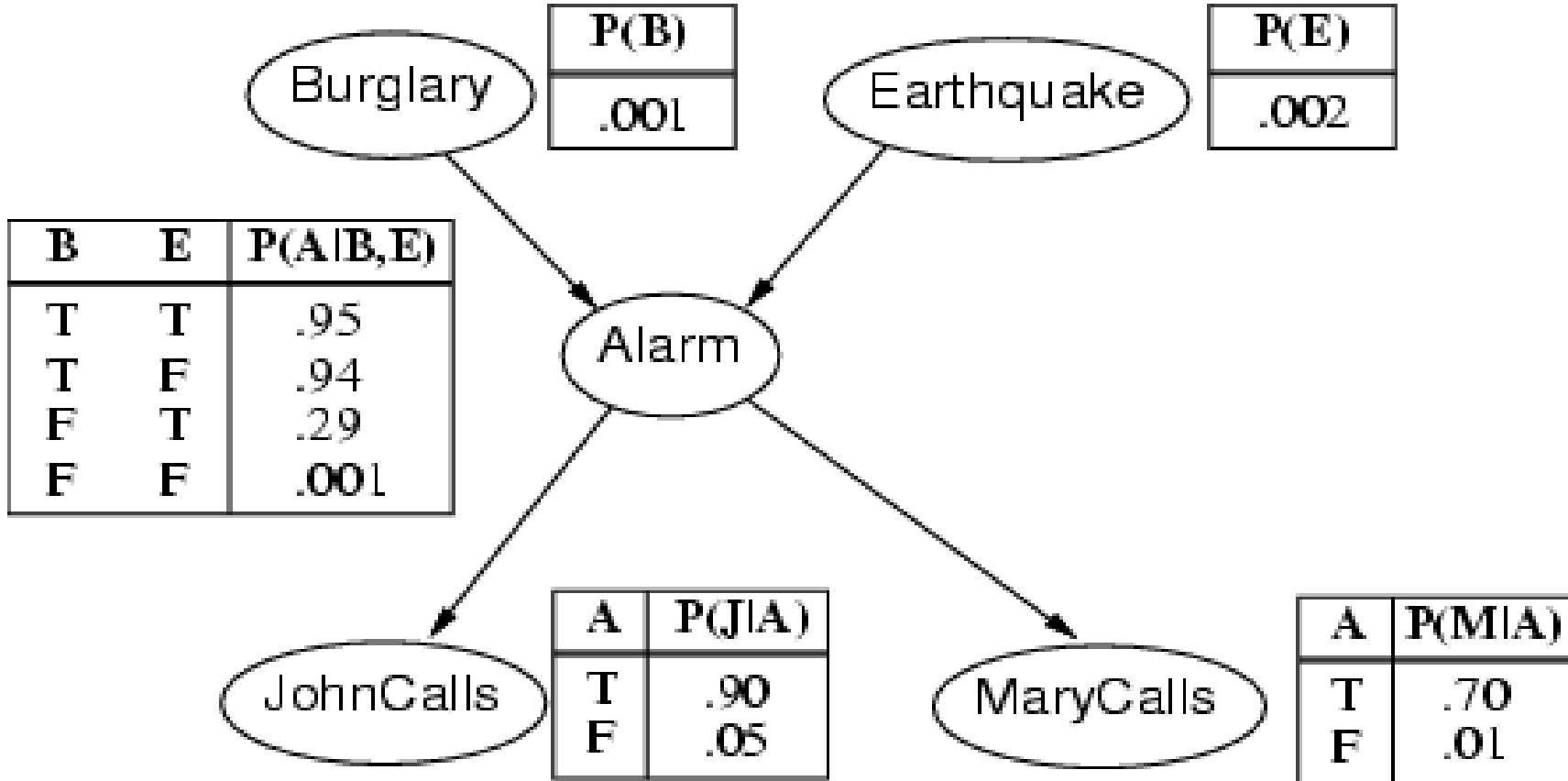


**ExamResult** is independent of the other variables

**Anna\_Here** and **Bob\_Here** are conditionally independent given **Train\_Running**



# Bayesian Networks: Earthquake Example



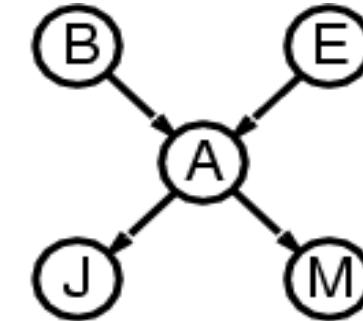
[Russell & Norvig]



# Bayesian Networks: Semantics

- The full joint distribution is the product of the local conditional distributions:

$$P(x_1 \wedge x_2 \wedge \dots \wedge x_n) = \prod_i^n P(x_i \mid \text{Parents}(X_i))$$



- For example:

What is the probability that alarm has activated, but neither burglary nor earthquake has occurred, and both John and Mary call?

$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$$

$$= P(j \mid a) P(m \mid a) P(a \mid \neg b, \neg e) P(\neg b) P(\neg e)$$

$$= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = \mathbf{0.000628}$$



# Bayesian Networks: Compactness

- A CPT for a node  $X_i$  with  $k$  parents has  $2^k$  rows
  - One for each combination of parent values
  - Assuming Binary variables
    - Each row requires one probability value, for  $X_i=\text{true}$   
(probability for  $X_i=\text{false}$  is just  $1 - \text{prob. for } X_i=\text{true}$ )
    - If each node has at most  $k$  parents, the complete network requires the order of  $(n \cdot 2^k)$  numbers
- Much more compact than full joint distribution:
  - Grows **linearly with  $n$**  (assuming a fixed max. number of parents), compared to the full joint distribution which grows **exponentially**
- For burglary network:
  - Have  $1 + 1 + 4 + 2 + 2 = \mathbf{10}$  numbers
  - Full joint distribution would have  $2^5 - 1 = \mathbf{31}$  numbers



# Constructing a Bayesian Network Manually

1. Choose an ordering of variables  $X_1, \dots, X_n$
2. For  $i = 1$  to  $n$ 
  - i. Add  $X_i$  to the network
  - ii. Its potential parents are its **predecessors** in the ordering:  
Only add arc to  $X_i$  if a potential parent **directly influences** it
- Note 1: Using Conditional Independence, rule is:  
Select parents such that  $P(X_i | \text{Parents}(X_i)) = P(X_i | X_1, \dots, X_{i-1})$
- Note 2: the ordering of variables will determine how the network can be structured  
Think about which variables **cause** which  
Add the **root causes** first, then the variables they directly influence,  
and so on  
Last added are those with **no** direct causal influence on any others



# Constructing a BN: Example

Node Ordering: M, J, A, B, E

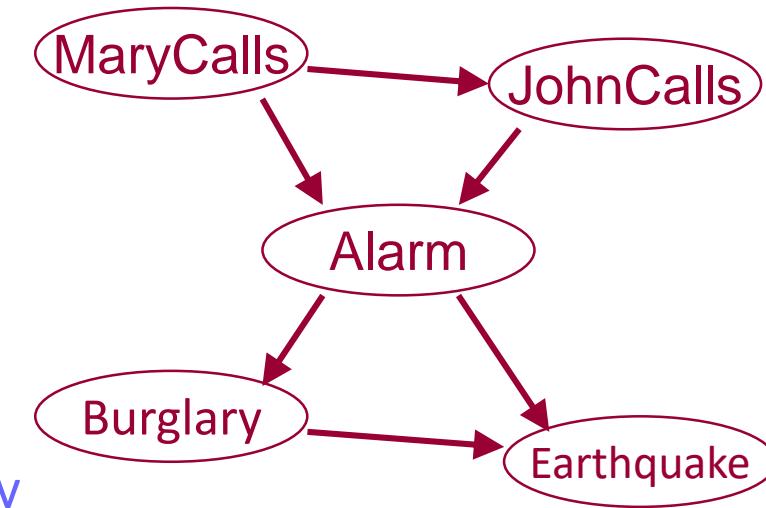
First add MaryCalls: no parents.

Add JohnCalls: If Mary calls, alarm likely to have activated, so more likely that John calls => add arc

Add Alarm: If Mary and John call, more likely that alarm has activated than if one or neither call => add arcs from both

Add Burglary: If we know alarm state, then JohnCalls or MaryCalls adds no more info:  $P(B | A, J, M) = P(B | M)$  => arc from Alarm only

Add Earthquake: If Alarm on, more likely there was an earthquake, unless there was a Burglary to explain the alarm => add arcs from both.





# Learning a BN from Data (1)

- Two sub-tasks in learning a BN:
  - Learn the structure
  - Estimate the probabilities
- Decomposable: can do separately
- Several approaches to structure learning
  - Bad news: finding optimum network is **NP-Hard!**
  - Typically combine quality score with search heuristics
- Examples of scores:
  - Minimum description length
  - Probability of network given data
- Examples of search procedures:
  - Genetic, hill-climbing, conditional independence tests



## Learning a BN from Data (2)

- K2 [Cooper & Herkovits, 1992]

Basis: Which of two **structures** more likely, given **DB**?

i.e. calculate  
equivalent to

$$\begin{aligned} & P(B_{S_i}|D) / P(B_{S_j}|D) \\ & P(B_{s_i}, D) / P(B_{S_j}, D) \end{aligned}$$

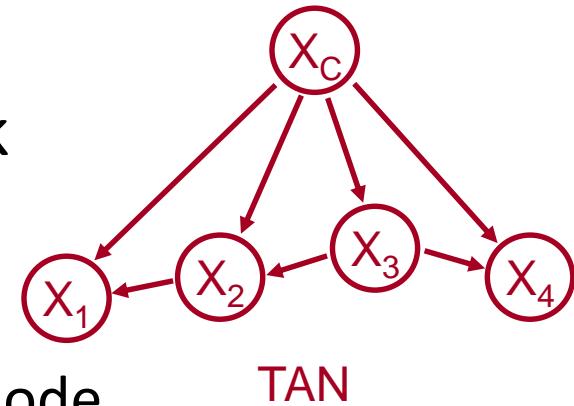
- Others assume a restricted form of network

TAN: Tree Augmented Naïve Bayes  
[Friedman et al '97]

Max. of 1 dependency between children of class node

- After learning structure, learn parameters from data

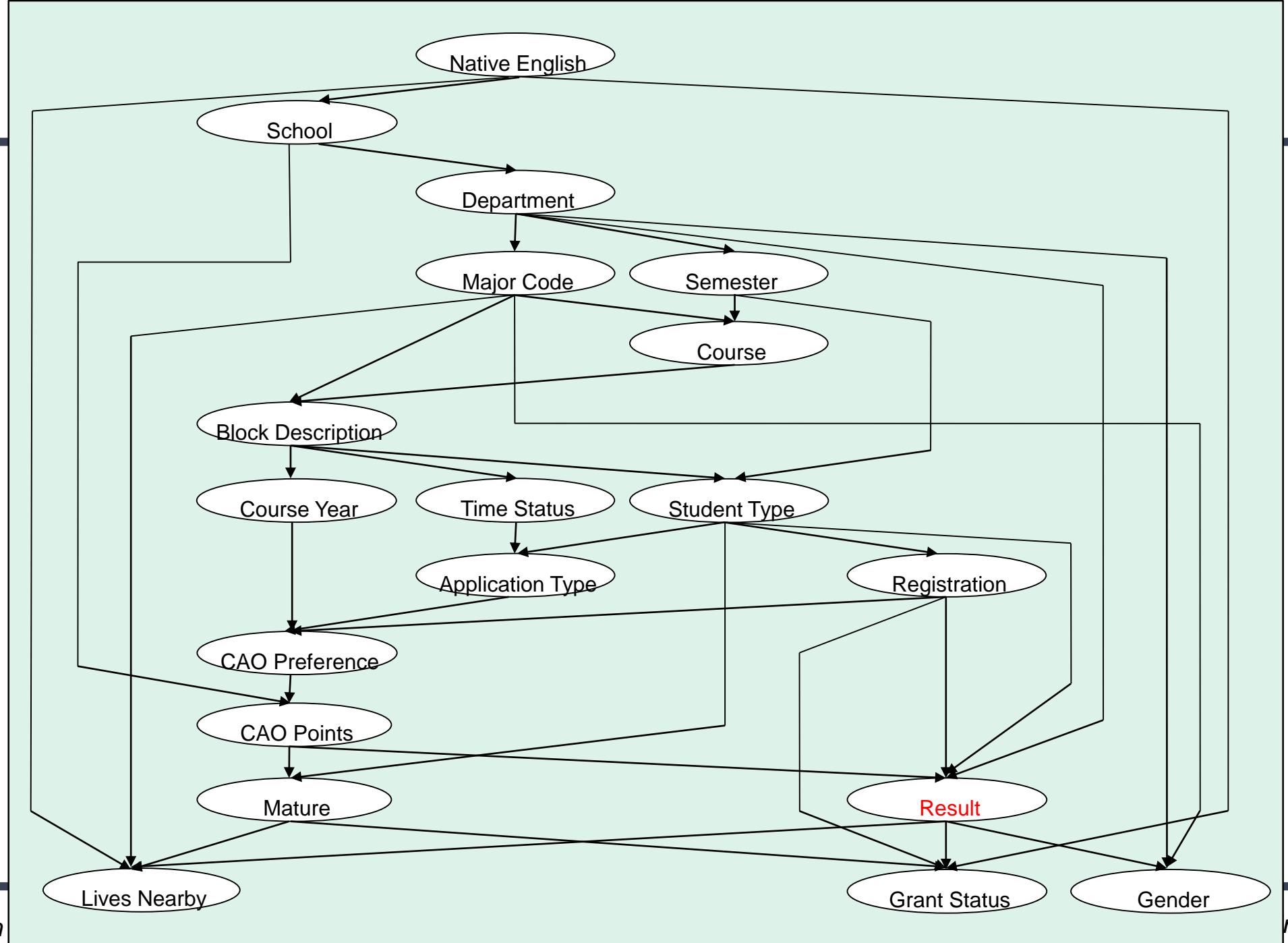
Very similar to learning Naïve Bayes parameters.





# Data Exploration with BNs

- BNs help identify correlations in data (pos. or neg.)  
Rather than just pairwise correlations, multiple ones considered simultaneously  
Absence of arc: No correlation
- Inductively learn BN from data  
Examine it to explore relationships
- Example: Madden, Lyons & Kavanagh, AICS 2008  
Analyse student records with BNs, Decision Trees & Rules  
Evidence-based understanding of how a variety of factors affect students' examination performance  
BN learned with Minimum description length (MDL) score and hill-climbing search  
Note: Data from another college, not NUI Galway





# Data Exploration with BN: Example

- Several 'obvious' relationships:  
E.g. School → Dept → Major Code → Course
- Others less obvious but logical:  
E.g. CAO Preference → CAO Points
- **Markov Blanket of a node:**  
Its parents, its children and its children's parents  
Nodes outside MB do not affect it
- **Markov blanket of Result node:**  
CAO Points, Department, Major Code,  
Grant Status, Lives Nearby, Native English, Registration, Mature, Gender,  
Student Type



# Classification using a BN

- Construct BN by induction from training DB  
Class variable not special
- To classify a new case:  
Generalised version of Naïve Bayes classification  
Assume value of  $X_c$  unknown, all others known  
For each possible value of  $X_c$  calculate joint probability of that instantiation of all variables:

$$P(X_1 = x_1 \wedge \dots \wedge X_n = x_n) = \prod_{i=1}^n P(X_i = x_i \mid \Pi_i = \pi_i)$$

Normalise resulting probabilities

Multiply by cost matrix if required

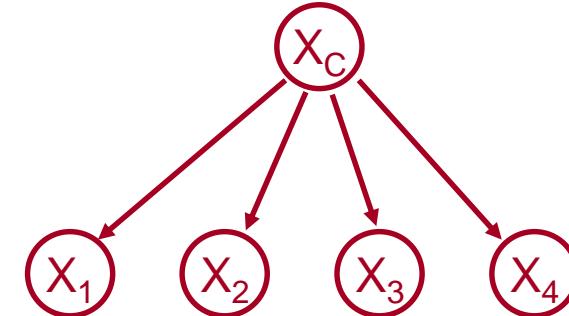
- Note:  
Only need to consider nodes in **Markov Blanket** of  $X_c$



# Restricted Bayesian Classifiers (1)

- **Naïve Bayes**

(saw already)



- Assumptions

1. All variables relevant to classification (**tolerates irrelevant**)
2. Other vars conditionally independent of each other
3. Direction of influence is from class var to others  
(i.e. class var is root cause)

- Relax Assumption 1 (and 2, weakly)

Use subset of variables

Selective Naïve Bayes [Langley & Sage, 1994]



## Restricted Bayesian Classifiers (2)

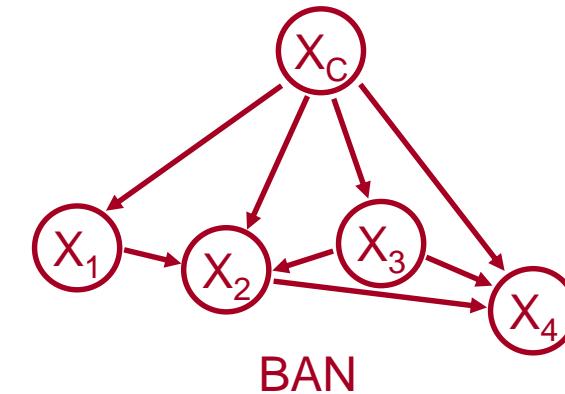
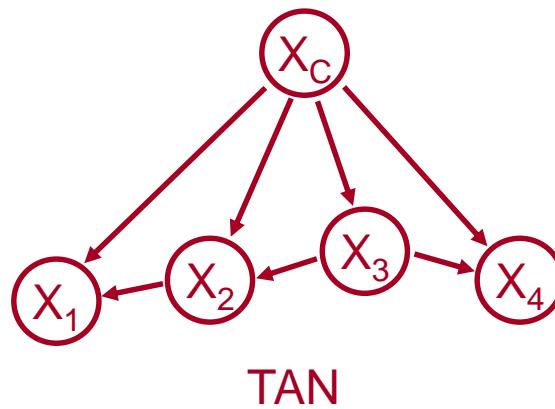
- Relax Assumption 2:  
Additional dependencies between variables

Tree Augmented Naïve Bayes (TAN)

[Friedman et al 1997]

Bayesian Network Augmented Naïve Bayes (BAN)

[Cheng & Greiner 2001]





# Learning Objectives Review

You should now be able to ...

- Discuss the motivation for handling uncertainty in ML
- Distinguish between prior and conditional probability
- Demonstrate understanding of how to use the axioms of probability and Bayes' rule
- Describe and apply the Naïve Bayes classifier to inductive learning problems
- Show how Bayesian Networks represent influence and independence of variables
- Discuss how BNs can be used for classification & data exploration.