

# Introduction to Machine Learning

Rob Lyon

[robert.lyon@cs.man.ac.uk](mailto:robert.lyon@cs.man.ac.uk)

Machine Learning & Optimisation Group, School of Computer Science  
&  
Pulsar Group

# Overview

- **A whirlwind introduction.**
- **No hard sell, just facts.**
- **Hopefully make terminology clear.**
- **Highlight gotchas, share wisdom.**
- **Demonstrate example code.**

# What is ML?

Human Ability	A.I. Research Field
Seeing	Computer Vision
Talking	Speech Synthesis
Listening	Speech Recognition
Understanding Language	Natural Language Processing
Reasoning	Automated Reasoning
Consciousness	Philosophy / Cognitive Science
Moving	Robotics
Learning	Machine Learning

- **Learning to approximate complex functions.**
- **The illusion of intelligence.**

Many people hear the term machine learning and immediately think of killer movie cyborgs and computer systems. But ML is just one of the many fields of A.I. that people are currently working on.

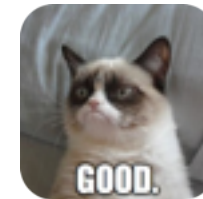
It's important to note that machine learning isn't focused on generating intelligence like yours or mine, but rather on approximating the positive results we obtain with our intelligence.

Think of it this way. If something responds in the way you would given the same information/input, you would probably deem it intelligent. But for most of the mundane decisions we need to make day to day, real conscious intelligence isn't really required. This is where machine learning comes in. It is focused on building mathematical models that can approximate our decision making processes given some data, and the correct decision to be made with that data.

So you could say our aim is to build systems that are able to mimic human decisions when given exactly the same information.

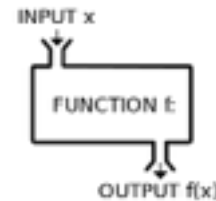
# Overview

Machine learning is essentially statistics. But we rebranded it and renamed everything in the process. Sorry.



# Learning from Data

- Some underlying process we want to study.
- We observe this process and record data.
- So there's some true function  $f(x)$ .
- We record data describing  $f(x)'$ .
- We construct a model of the process  $f(x)''$ .



There are many natural and artificial processes we wish to study as scientists. To do so we observe them in as much detail as possible, and record the data describing them. The data we collect is usually incomplete, since we can't yet record in infinite detail all the characteristics of the process we are trying to study. For instance its not possible to record the position, state etc of every atom in a blood sample, and in any case such detail is not always needed. But there is always some underlying function generating the data we observe which I call  $f(x)$  – the true function.

We want to understand this function, but our observations only describe an incomplete record of the function's behaviour so we only see an approximation of its behaviour,  $f(x)'$ .

However we can use machine learning to try and reconstruct the true function  $f(x)$ , from the data we have collected describing  $f(x)'$ . This is a useful thing to do in many domains. For instance imagine we want to identify automated spam email. If we had access to the exact code used to generate it, we'd be able to build a spam filter that could always get rid of it. But we don't have access to the code (it's hidden in the depths of some hacker's hard-drive!), all we have are examples of the spam it generates. If we can model the the function using machine learning we can build a filter capable of identifying spam most of the time. I call this model  $f(x)''$ . It won't be perfect. It will make mistakes. But until we have better data and a more complete view of the underlying function, it's the best we can do. Though very often these machine learning built models are extremely accurate.

# Unlabelled Data

- Consists only of data  $X$ .

$$x_i \in X$$

- Data set made up of examples = instances.
- Each example usually defined  $x_i = \{x_i^1, \dots, x_i^m\}$  .

$$x_i^j \in \mathbb{R}$$

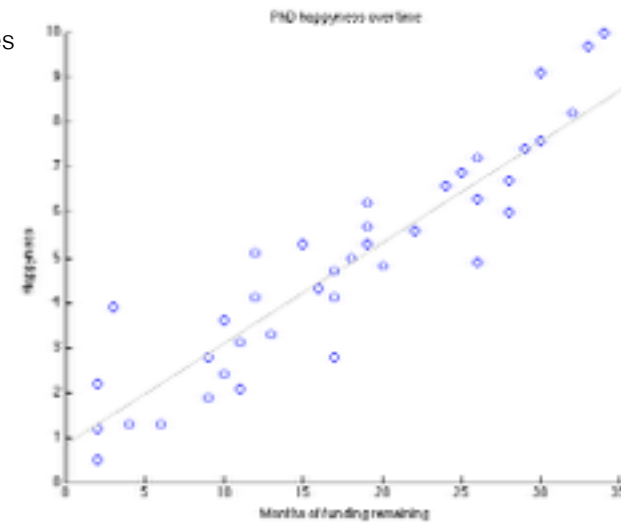
- Each example is unlabelled.

# Example

Features / Variables / Attributes

Examples  
Instances  
Patterns

Example	x	x
1	7.4	29
2	6.6	24
3	3.1	11
...	...	...
n	0.5	2



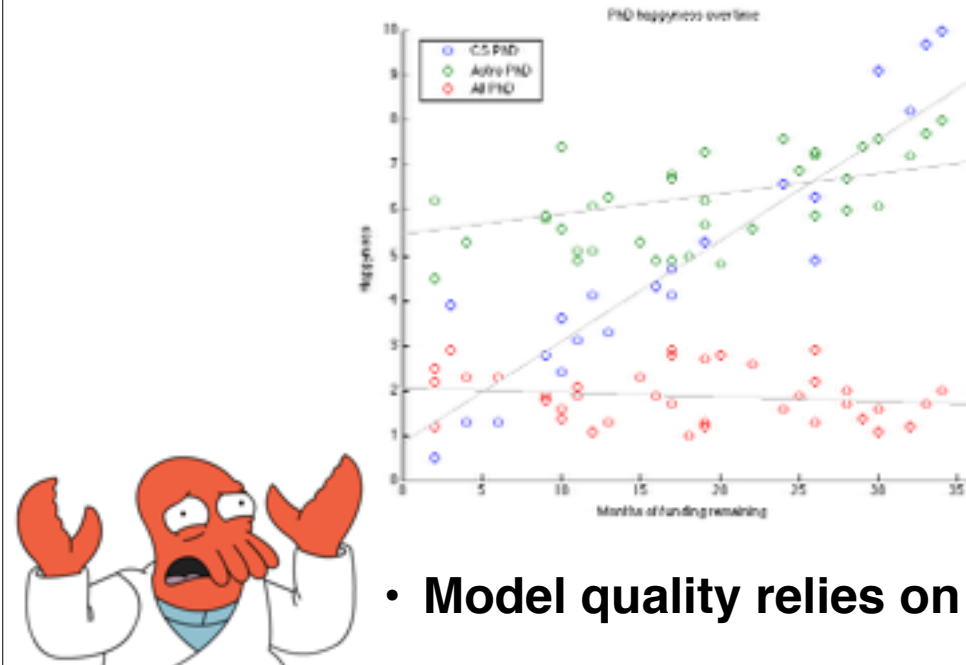
- **Often describes regression problems.**

The graph on the right depicts PhD student happiness as a function of remaining months of funding. This is most definitely a fake data set I generated to illustrate the concept of unlabelled data.

We may want to cluster unlabelled data to find distinct groups, i.e. clusters of happy students.

We may want to predict the happiness of an arbitrary student given the number of months of funding they have left.

# Example



- **Model quality relies on data collected.**

This graph extends the previous example. It shows happiness data points for CS PhD students (become really unhappy over time as funding runs out), astronomy PhD students (minor drop in happiness over time), and all remaining PhD students (who are just plain miserable!). This plot highlights the following: the quality of the learning models we build, is entirely dependent on the data we give them.



# Labelled: Binary Data

- Consists of data  $X$  and labels  $Y$ .

$$x_i \in X \qquad y_i \in Y$$

- Data set made up of examples = instances.
- Each example usually defined  $x_i = \{x_i^1, \dots, x_i^m\}$ .

$$x_i^j \in \mathbb{R}$$

- Each example is either labelled or unlabelled.

$$Y = \{-1, 0, 1\}$$

Labelled binary aka two-class data is also everywhere. Candidate pulsar data is binary since we would like to split signal detections into pulsar and non-pulsar classes. The spam classification problem mentioned earlier is also binary, since the aim is to separate spam from non-spam. The crucial difference here is that labelled data contains class information. In other words, unlabelled data consists only of data  $X$ , and labelled data consists of  $X$  and  $Y$ . The  $Y$  indicates the class membership of each example in  $X$ .

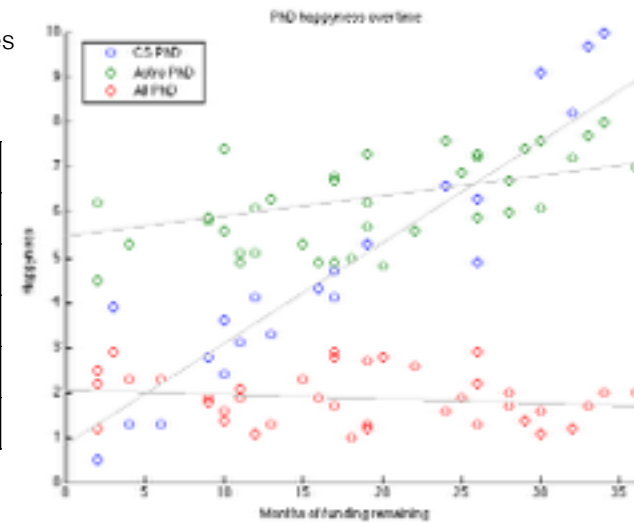
# Example

Features / Variables / Attributes

Examples  
Instances  
Patterns

Example	x	x	y
1	7.4	29	1
2	6.6	24	1
3	3.1	11	0
...	...	...	...
n	8.5	2	0

Class label / Target



- **Now different questions can be asked.**

Again the graph on the right depicts PhD student happiness as a function of remaining months of funding. It shows happiness data points for CS PhD students (become really unhappy over time as funding runs out), astronomy PhD students (minor drop in happiness over time), and all remaining PhD students (who are just plain miserable!).

This time however the plot has been generated from labelled data. This allows us to ask questions we couldn't ask before when dealing with only unlabelled data. For example we can now ask,

If given a happiness level and the months remaining for a student x, is that student a CS student or not a CS student.

# Labelled: Multi-class Data

- Consists of data  $X$  and labels  $Y$ .

$$x_i \in X \qquad y_i \in Y$$

- Data set made up of examples = instances.
- Each example usually defined  $x_i = \{x_i^1, \dots, x_i^m\}$ .

$$x_i^j \in \mathbb{R}$$

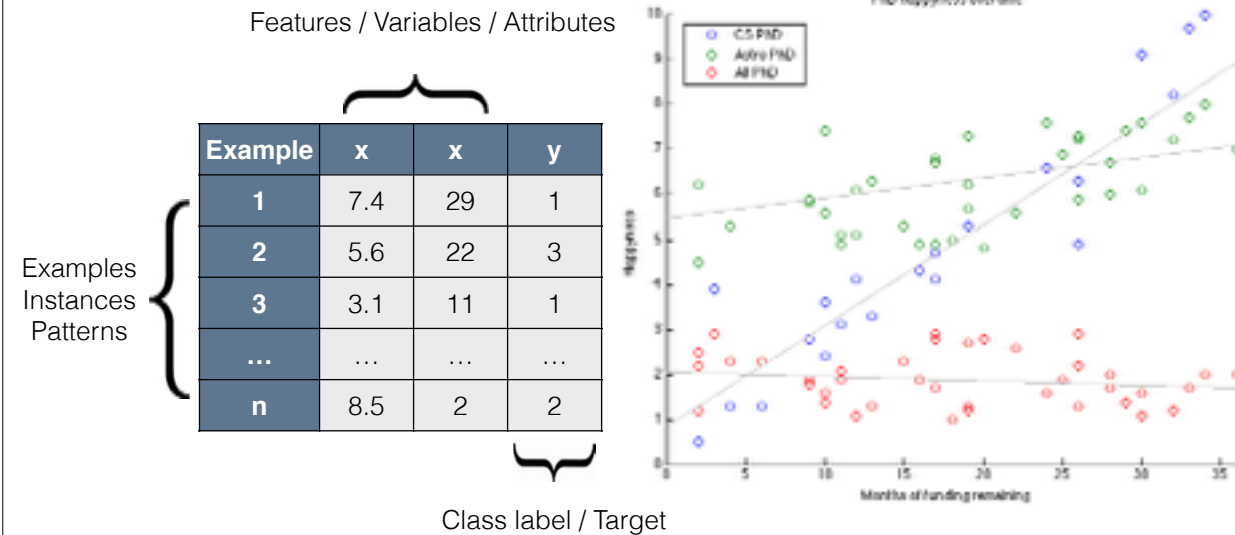
- Each example is either labelled or unlabelled.

$$Y = \{\text{class } 1, \dots, \text{class } n\}$$

Labelled multi-class data is also everywhere. Candidate pulsar data can also be considered multi-class, since we can split signal detections into RFI, noise, slow-pulsar, ms-pulsar classes. Often problems are defined as binary simply for convenience and simplicity, but in reality they are multi-class.

The crucial difference here is that there are more than 2 labels possible for each example in  $X$ .

# Example



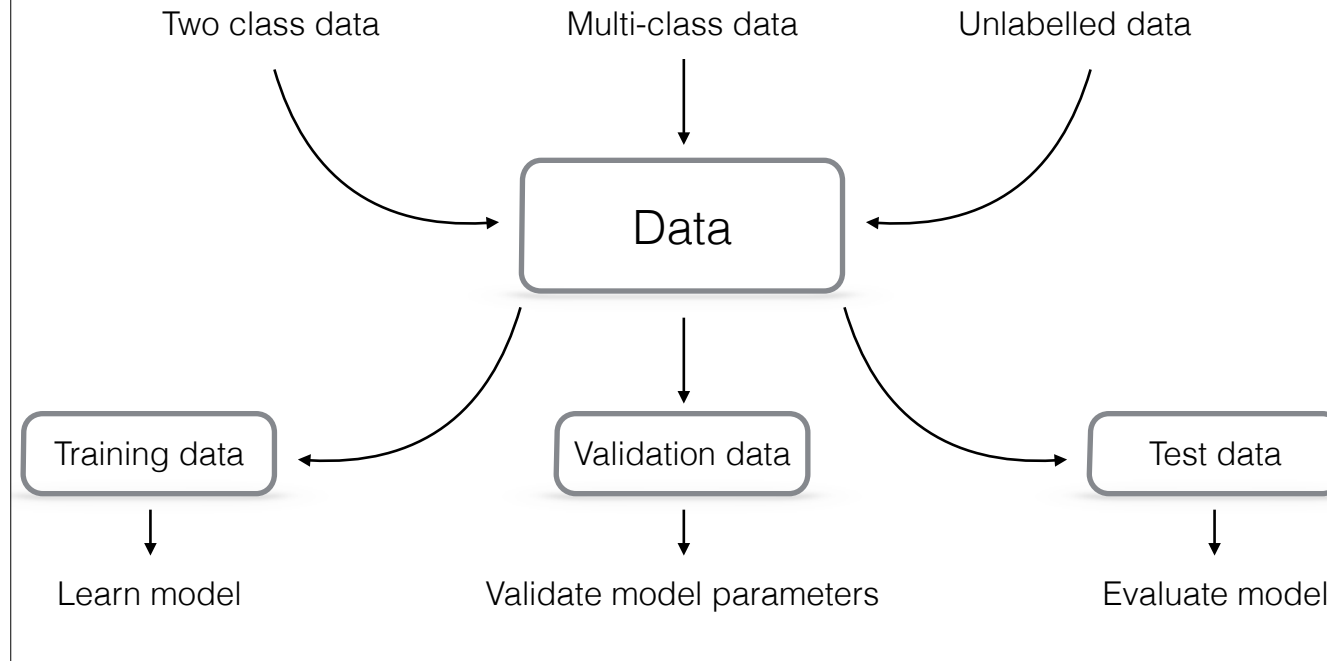
- **Again, different questions can be asked.**

Again the graph on the right depicts PhD student happiness as a function of remaining months of funding. It shows happiness data points for CS PhD students (become really unhappy over time as funding runs out), astronomy PhD students (minor drop in happiness over time), and all remaining PhD students (who are just plain miserable!).

This time however the plot has been generated from labelled multi-class data. This allows us to ask questions we couldn't ask before when dealing with only two-class data. For example we can now ask,

If given a happiness level and the months remaining for student x, is that student a CS or Astro, or any other student?  
Are CS and Astro students similarly happy (via clustering)?  
Given a happiness level, is a student likely working in the astronomy domain (regression)?

# Training & Test Data



No matter what data you have, this data must be prepared for use by machine learning algorithms.

Crucially ML algorithms require at least two (sometimes three) types of data set. They require:

**Training data:**

Used by our algorithms to learn the function we would like approximated.

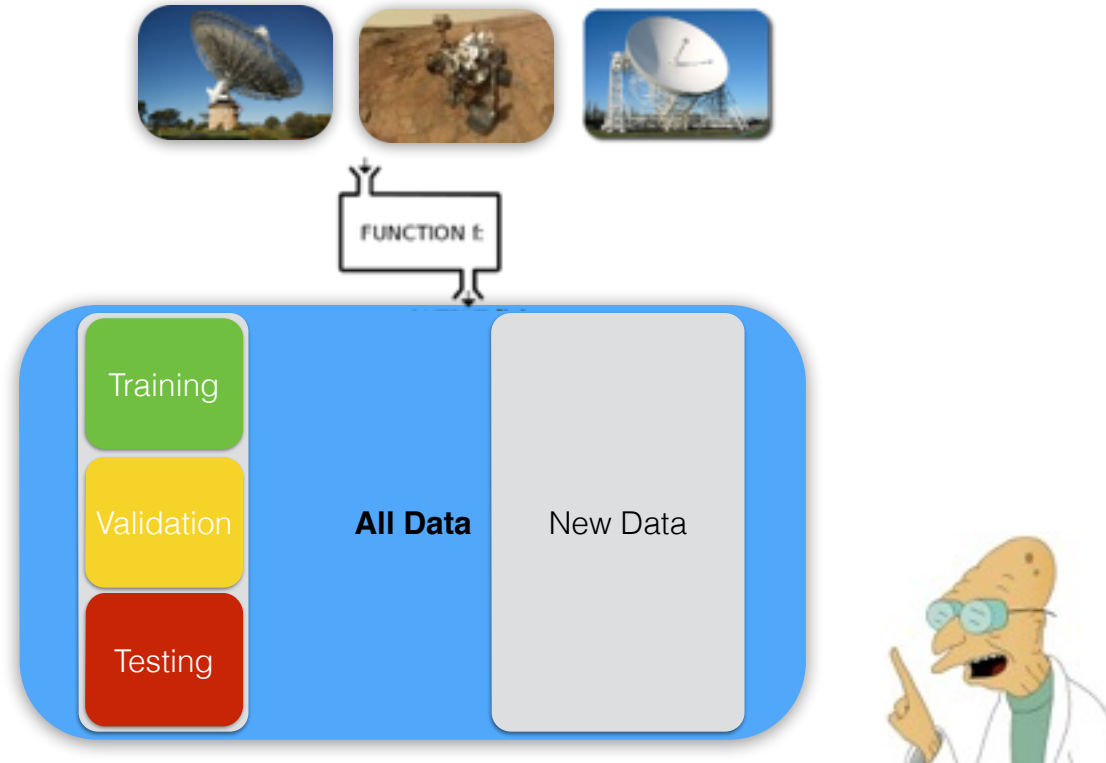
**Testing data:**

Used to evaluate the performance/correctness of the function we generated during training.

**Validation data:**

Used to tune parameters in some learning algorithms to improve the quality of the function being generated.

# Training & Test Data

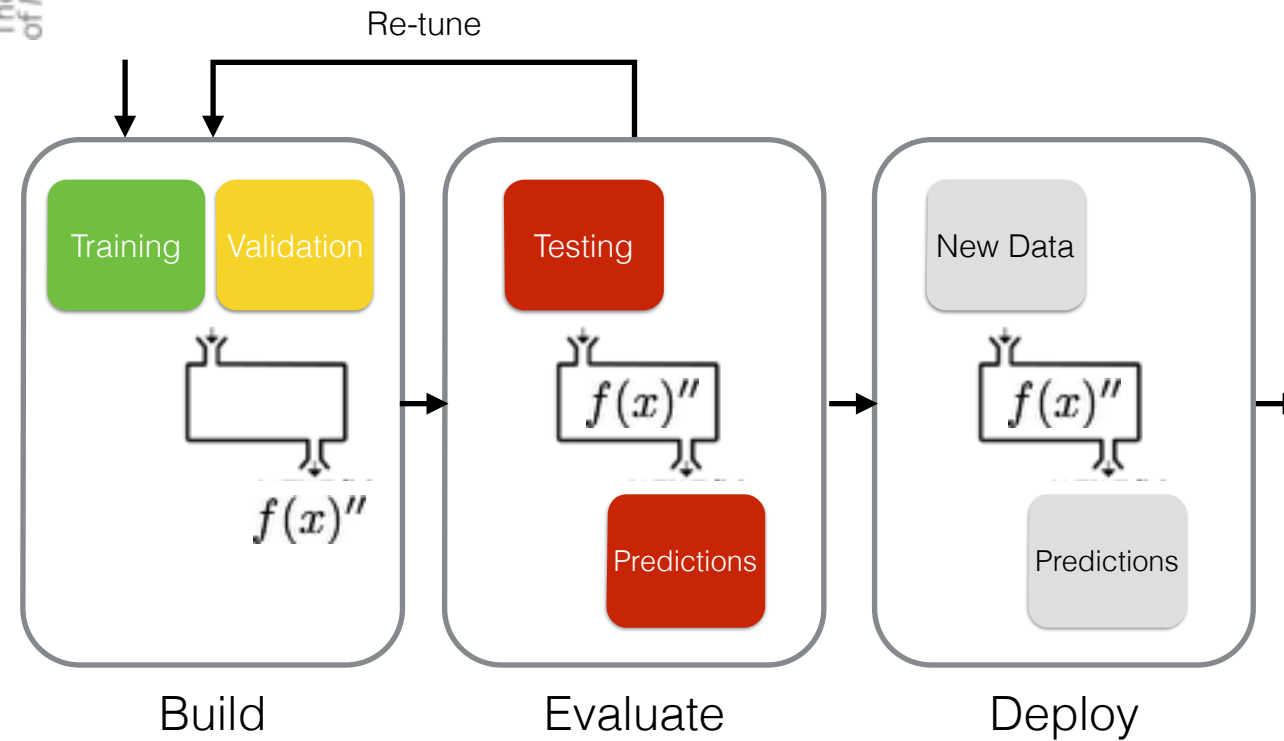


Training, validation and test data sets must be disjoint. Examples should not be repeated. Each data set must be representative of the true data distribution, otherwise learning algorithms will likely become biased in some intrinsic way.

If the true distribution is heavily imbalanced, then it is acceptable to rebalance the testing and training data sets. There are no precise rules of thumb here. However I can guide you by my own experience. In my own work, pulsar data has an imbalance of  $+1:-10,000$  – that's ten thousand negative examples for every positive. When creating testing and training data sets for my algorithms, my training distribution is never worse than  $+1:-10$ . This is because there has been some CS research done in the past which suggests that imbalances greater than this in the training set, cause hurtful biases. Note though that my test distribution remains  $+1:-10,000$ , since my algorithms have to operate on this kind of distribution in reality.

Note that if faced with small sample sizes – all bets are off. Machine learning algorithms do not perform well on these sorts of data sets. If you don't have much data, or only have small training/test sets (fewer than 50 examples per class), all I can say is this: get more data. Somehow, just get it. Generate it artificially if you have to.

# Learning Process



Overview of how we build learning algorithms:

- 1) First we create training and test data sets (stratified sampling very useful here).
- 2) Build the algorithm.
- 3) Evaluate the algorithm's performance on test data.
- 4) Re-tune the classifier or improve the training set and repeat steps 1–3.
- 5) Deploy the algorithm on real data.

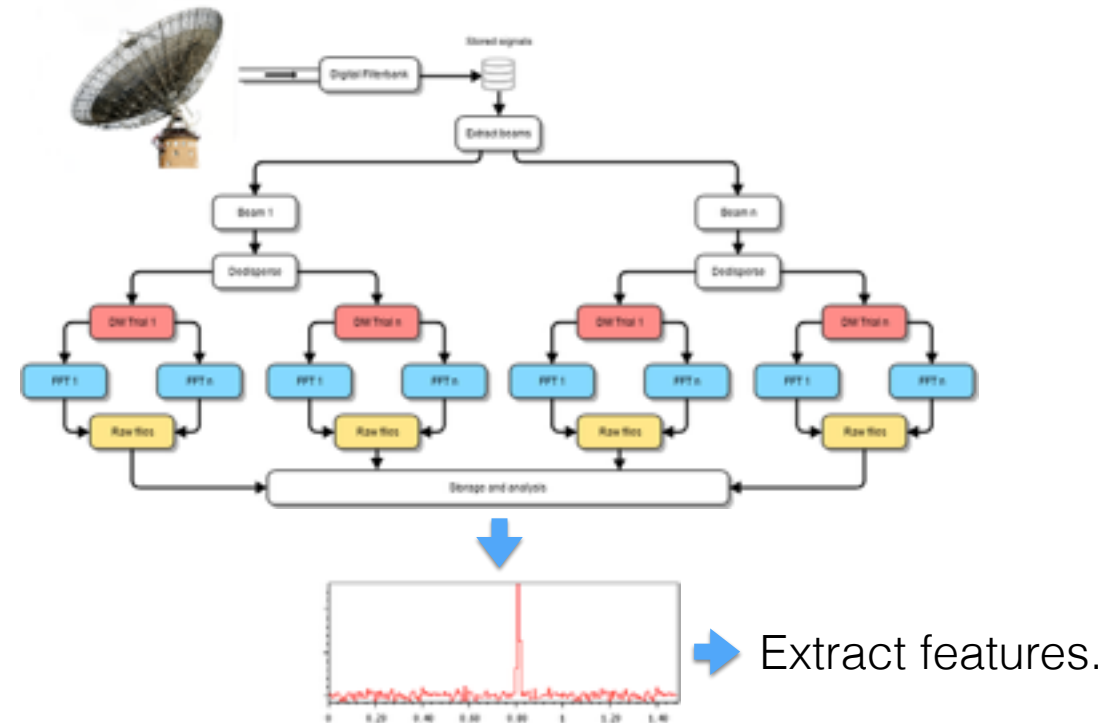
# Data Collection & Pre-processing

You're data producers, not just consumers.

*With great power comes great responsibility.*



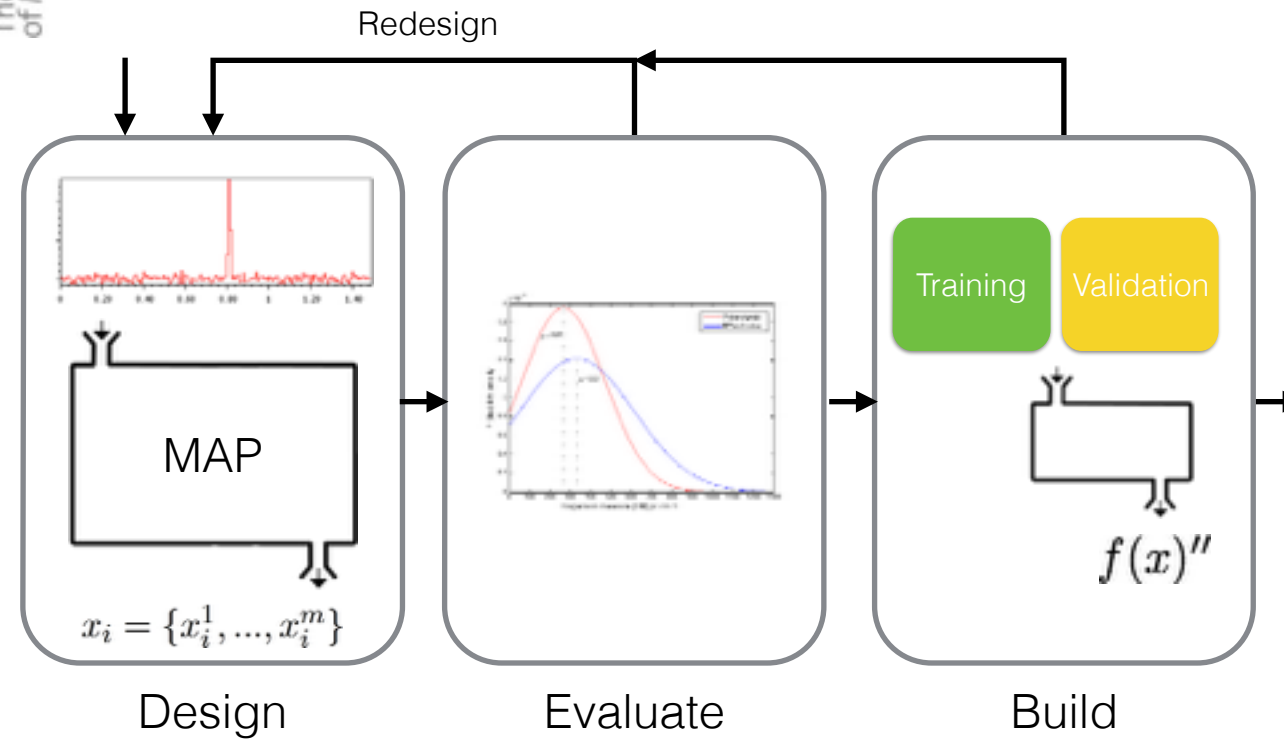
# Collection



The data we collect is often not very useful in its raw form. Consider a candidate pulsar 'pulse profile' as an example (basically shows signal intensity over time). This data is comprised of 128 continuous values, describing the pulse intensity in each time bin.

On its own this data doesn't say very much, especially if we want to classify the signals. However we can design new features that summarise the key characteristics of the signal, which we think will be useful for separating pulsar and non-pulsar examples. For instance how well does a sine curve fit to the signal? If it fits very well, then the signal is probably RFI.

# Design Features



Feature design is a process which you need to undertake in stages.

- 1) Design some features.
- 2) Write some code to convert your raw data into the new features and output them in to some usable format.
- 3) Evaluate class separability using these features i.e. do they separate pulsar/non-pulsar or spam/non-spam well.
- 4) Build and evaluate a classifier using data made up of ONLY the new features. Evaluate it's performance. If the classifier performs well, then great! If not you may need to go back to stage 1.

- **Statistical hypothesis testing.**
- **Automated tools exist to select features using:**
  - **Wrapper method.**
  - **Filtering (Ranking via correlation coefficient).**
  - **Information theory approaches.**

The screenshot shows the 'Run' configuration dialog in the NetBeans IDE. The 'Run' configuration is selected. The 'Main Class' field is set to 'com.muhimbi.Yazlari'. The 'Project' field is set to 'Yazlari'. The 'Working Directory' field is set to '\$PROJECT\_HOME'. The 'Run' button is highlighted.

Here I show how to use the feature selection tools built it to WEKA, a Java based machine learning tool.

# Goal of Feature Design

Maximise class separability.

I've written some matlab scripts that do basic K-NN classification, and provided some data to test with. Here I run the script as part of a demonstration. Run the script: Matlab\_FeatureBoxplots.m

# Advice (1)

- Among competing hypotheses, the one with the fewest assumptions should be selected - **Occam's Razor.**



Sometimes the simplest features prove to be the best. Don't discount the simple.

Also if a feature appears to be working well, then try its inverse to test if things degrade. If they don't, perhaps the feature which you thought useful, is not responsible for good performance.

# Curse of Dimensionality

- Too many features cause problems.
- Increase runtime and memory overheads.
- Predictive power reduces as dimensionality

increases:

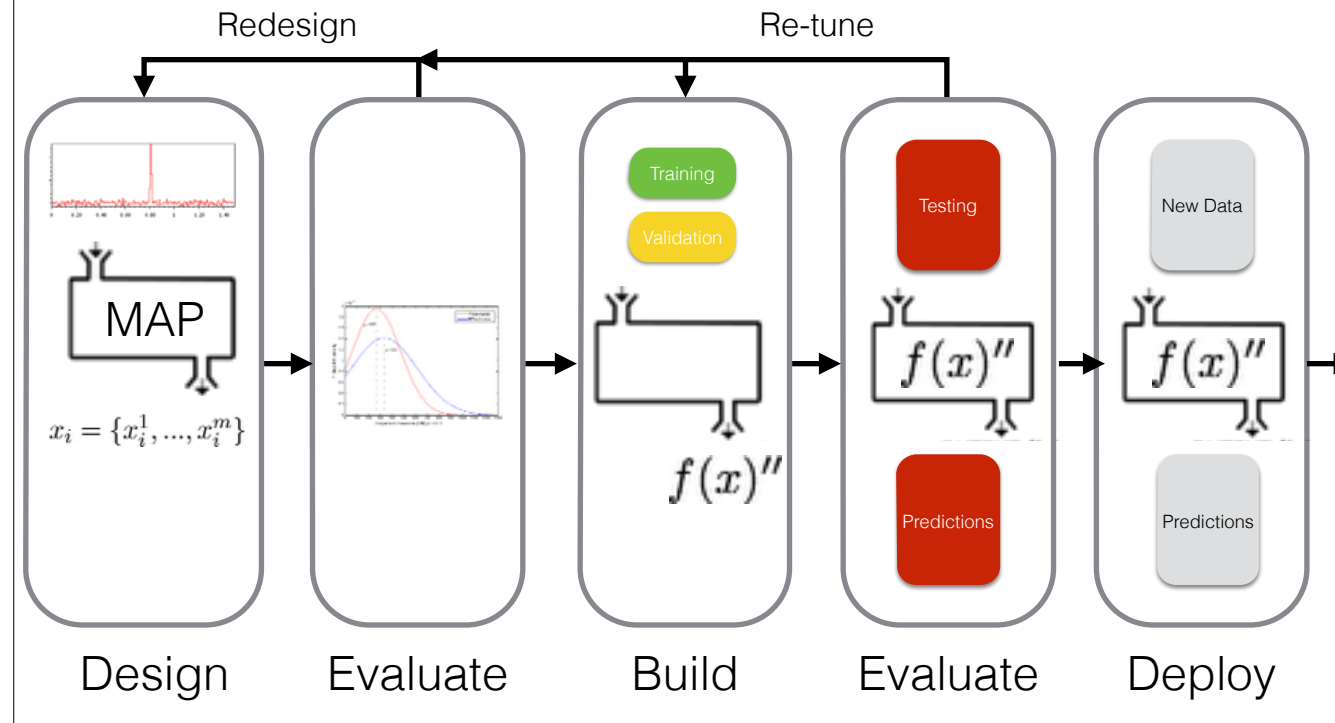
- *Hughes effect*.
- More training samples required.
- Caused by sparsity in high dimensional space.



Generally its better to have as few features has possible. Classifiers will be faster to train upon data with fewer features, require less memory overheads, and classify data faster.

Also high dimensional data is very difficult to visualise.

# Complete Steps



An overview of the whole process.

# Advice (2)

- **Takes time to assign true labels.**
- **When you analyse data, label it if you can.**
- **Saves time in the long run.**
- **Integer labels usually best.**
- **Store data sensibly for future use.**
- **Normalise data where possible.**



It's hard to know what data will be useful in the future. Also as experts your analysis of data is worth time and crucially money – so if you study data then it's probably worth your time labelling it. In the bioinformatics domain doctors are paid by the hour to manually label genomic and cancer data. Unfortunately I can't pay you to label astronomical data, but I can implore you to do it!

Data normalisation is also very important, since it helps prevent classifiers being skewed by large outliers. So normalise your data if you can, but record the process you used to do it.



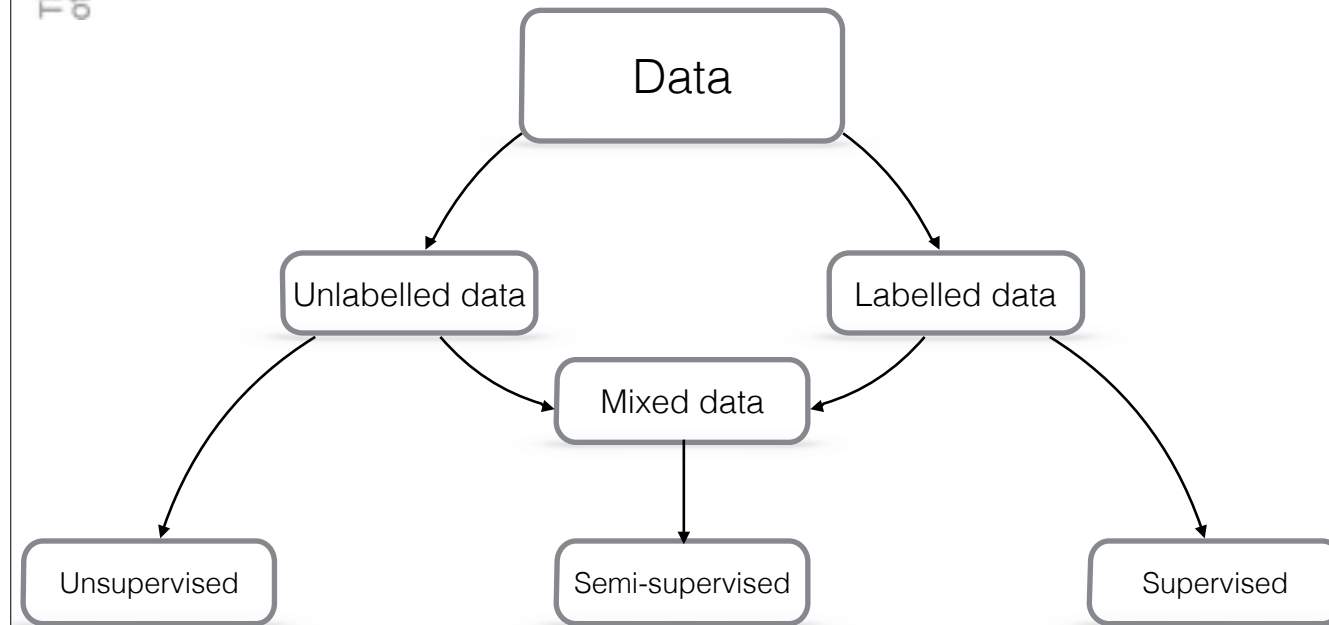
# I have data, now what?

Depends. What do you need / want to know?

# What can ML do?



# Learning Approaches



Machine learning algorithms come in three varieties.

Unsupervised, semi-supervised, and supervised learning approaches.

Unsupervised learning requires only unlabelled data.

Supervised learning requires only labelled data.

Semi-supervised learning uses both labelled and unlabelled data. The rationale here is that unlabelled data is useful for learning.

# Supervised Learning

- Requires fully labelled training data.
- Most standard classifiers are supervised learners.
- Student teacher-model.
  - Decision trees.
  - Neural networks
  - SVM
- Used to make class predictions.



# Unsupervised

- **Doesn't require labels (though they're helpful).**
- **Most standard classifiers are supervised learners.**
- **Learn data structure.**
  - **Self-organizing Map (SOM) .**
  - **Clustering**
  - **Dimensionality reduction.**
- **Often used to label unlabelled data.**

# Semi-supervised

- **Requires only some labelled training data.**
- **Attempts to learn from unlabelled examples..**
- **Student teacher-model.**
  - **Generative models.**
  - **Low density separation.**
- **Often used to label new data.**

These approaches are often used where there is little labelled data.

For example you can train a clustering algorithm using the little labelled data you have, and then label the clusters. The simplest way to label the clusters, would be to assign each cluster the label of the most prevalent class it contains. You then cluster each unlabelled data point one-by-one, and find which cluster its closest to.

You then assign the unlabelled data point the label of the cluster it's closest to.

# Classification

- Predicting labels for unseen examples.
- Need to build a classifier to separate the classes.

Training data

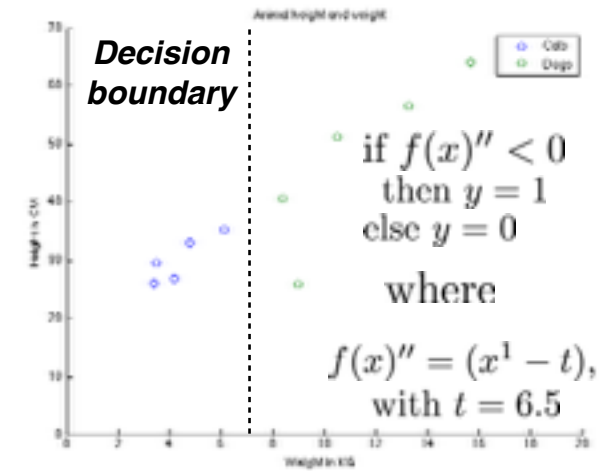
	x	x	y
1	29.5	3.5	1
2	26.1	3.4	1
3	33.1	4.8	1
...	...	...	...
n	51.2	10.5	0



$f(x)''$

Decision Stump

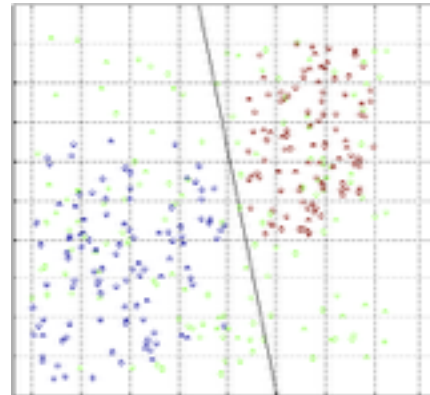
if  $x^1 < 6.5$   
then  $y = 1$   
else  $y = 0$



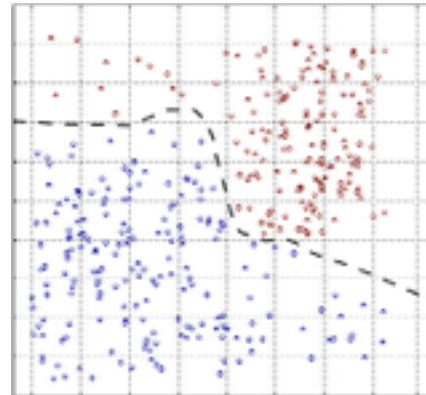
This slide shows a simple example of data classification. It also shows the simplest classifier around being constructed: the decision stump. Since the training examples describing cats and dogs can be easily separated according to weight, the stump simply chooses an appropriate split point. Here that is chosen to be 6.5. This produces a decision boundary that splits the training examples.

# Linearly Separable?

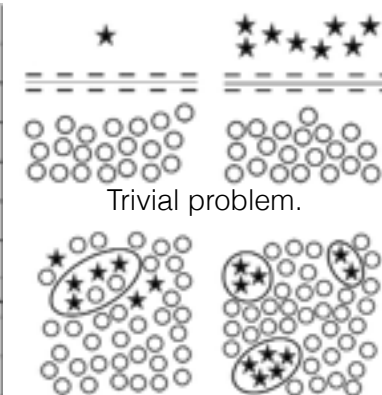
- Predicting labels for unseen examples.
- Need to build a classifier to separate the classes.



Linearly separable



Non-linearly separable



Tougher problems.

The previous example was linearly separable with a single contiguous boundary. However in reality most classification problems are much more complex. Here we see some examples. The example in the bottom right shows the imbalanced learning problem, which often occurs in experimental/observational science. This is a problem occurring when we are trying to identify rare occurrences in a sea of common examples. For example pulsar signal detection, credit card fraud detection, higgs boson detection!



# Classifiers

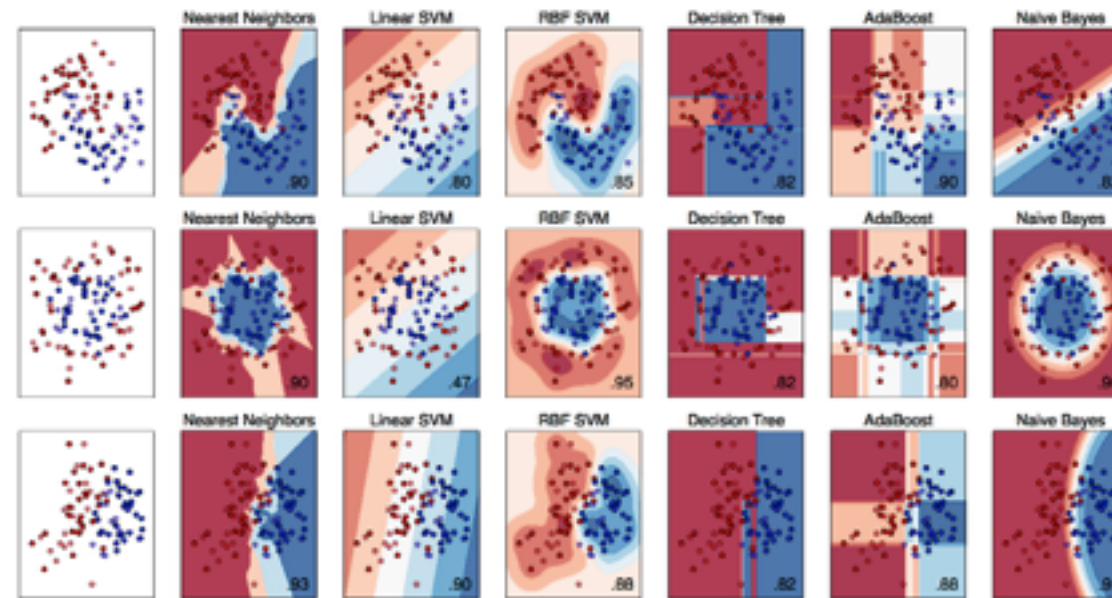
- Literally countless to choose from.
- All trade-off complexity and performance.
- So which to use?

Algorithm	Fitting Speed	Prediction Speed	Memory Usage	Easy to Interpret	Handles Categorical Predictors
Trees	Fast	Fast	Low	Yes	Yes
SVM	Medium	Medium	High	Yes / No	No
Naive Bayes	Fast	Fast	Low	Yes	Yes
Neural Networks	Medium	Fast	High / Low	No	No
Nearest Neighbor	Fast	Medium	High	No	Yes

# No free lunch

- The “no free lunch theory” indicates that there will never be a single best algorithm, better than all others in terms of predictive power. However, apart from their predictive performance, each classifier has its own attractive properties which are important to different groups of people (Wolpert & Macready).

# No free lunch

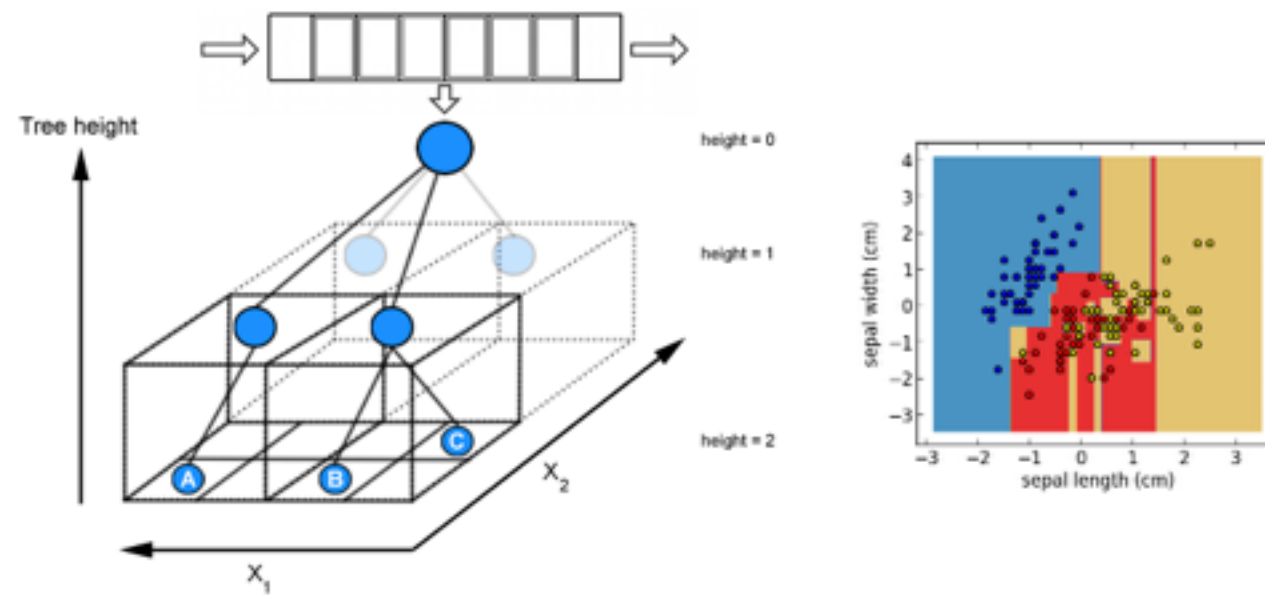


This slide shows three common data sets in the left most column. The remaining columns show how some common classifiers perform on these data sets. It's clear that no one algorithm works best across all data sets.



# Standard Classifiers

A very, very brief overview.

# Tree Learning



This slide shows firstly on the left, how a decision tree classifier (a supervised learner) partitions the training data into regions. The image on the right shows how these decision regions can be used to produce classifications.

# Tree Building

Training data, **S**

	Height	Weight	Length	Legs	Eyes	y
x	34.1	4.8	41.5	4	2	1
x	26.9	4.2	46.7	4	2	1
x	35.3	8.5	45.5	4	2	1
x	77.3	53.2	104.3	4	2	1
x	51.2	10.5	69.9	4	2	0
x	40.6	8.4	57.1	4	2	0
x	64.1	15.7	80.3	4	2	0
x	25.9	9	55.2	4	2	0
x	56.6	13.3	66.7	4	2	0
x	17.6	2.5	20.7	4	2	0

Which attribute best to split upon?

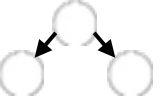
- Choose the attribute that maximises information gain.

Entropy =  $-(4/10) \log_2 (4/10) + -(6/10) \log_2 (6/10)$   
= .971to 3.d.p.

Values(Weight) = 4.8,4.2,...,2.5

S=[4+,6-]      S<sub>weight <5</sub>=[2+, 1-]  
                         S<sub>weight >5</sub>=[2+,5-]

Continuous data would be discretised into bins, then the best threshold chosen. Here 5 is a test split.



Gain(S,Weight<5) = Entropy(S) - (3/10)Entropy(S<sub>weight <5</sub>) - (7/10)Entropy(S<sub>weight >5</sub>)  
= 0.971-(3/10)0.918-(7/10)0.863  
= 0.092 to 3.d.p. not great!

$$\text{Entropy}(S) \equiv \sum_{i=1}^c -p_i \log_2 p_i .$$

$$\text{Gain}(S,y_j) \equiv \text{Entropy}(S) - \sum_{v \in \text{Values}(y_j)} \frac{|S_v|}{|S|} \text{Entropy}(S_v).$$

The most difficult aspect of decision tree learning is deciding which is the most discriminating attribute to test at a particular node. An attribute is usually chosen by comparing all the available attributes and deciding which one is optimal according to some measure. Clearly we would like to choose an attribute that will be useful in separating instances belonging to different classes, i.e. the attribute that provides the most meaningful information.

Above I've given an example where the aim is to choose a decision tree split that maximal separates Cats (Class 1) and dogs (Class 2). But which feature is the best to split on?

Early decision tree algorithms such as ID3 used the concept of information gain to decide upon which attribute to use. Information gain measures the expected reduction in entropy, where entropy is a measure that characterises the impurity of an arbitrary collection of instances. Given a set of instances S, where each instance in S may belong to one of c possible classes, the entropy of S is defined as (above in slide),

where p<sub>{i}</sub> is equal to the proportion of instances in S belonging to class i. Thus the information gain, Gain(S,y<sub>{j}</sub>) of a particular attribute y<sub>{j}</sub> is given by (above in slide),

Here Values(y<sub>{j}</sub>) is the set of possible values for the attribute y<sub>{j}</sub>, and S<sub>{v}</sub> is the subset of S for which the attribute y<sub>{j}</sub> has value v. Thus the best attribute at a node, is the one which maximises the gain.

Information gain can then be used to choose the best splits according to information theory. In the example above we have continuous data which can't be directly used by tree algorithms. The data must be discretised in to bins, and then an optimal separation point chosen which bests splits the two classes – in this case cats and dogs.

Lets suppose we trial a split value for weight. If the weight is below 5 kg we consider the animal to be a cat, and above that a dog. That split point can be described as,

S<sub>weight <5</sub>=[2+,1-] -> So at least one dog would be misclassified using this threshold.  
S<sub>weight >5</sub>=[2+,5-] -> At least 2 cats would be misclassified by its complement.

We can proceed to calculate the informationGain (or just the Gain) for this split as shown in the slide.

We can compute the gain for other features and split points, and we choose the one that maximises the gain always.

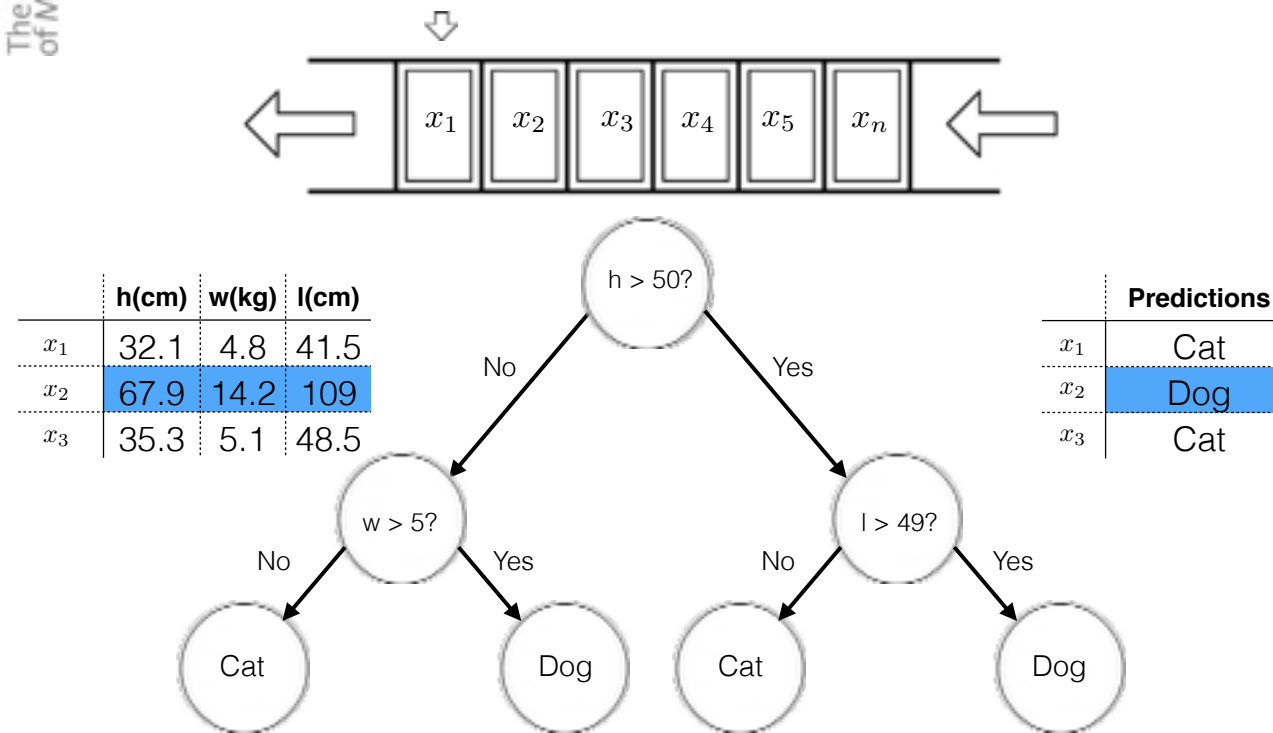
Disclaimer: Information gain is sensitive to skewed class distributions. In my own work I used the Hellinger distance as a skew insensitive replacement for information gain. This chooses tree splits according to those that maximise the class separation.

**Calculations not shown on slides:**

Entropy(S<sub>weight<5</sub>) =  $-(2/3) \log_2 (2/3) + -(1/3) \log_2 (1/3)$   
= .918to 3.d.p.

Entropy(S<sub>weight>5</sub>) =  $-(2/7) \log_2 (2/7) + -(5/7) \log_2 (5/7)$   
= .863to 3.d.p.

# Tree Classification



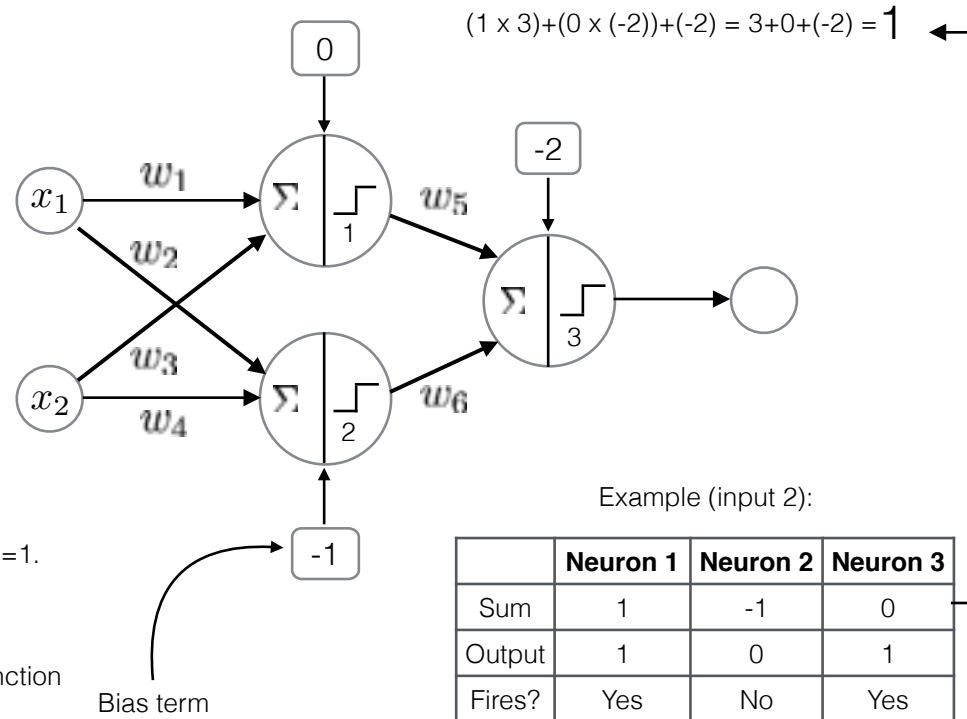
This slide shows how a simple decision tree classifies data in practice. The decision tree is a very simple learner that makes few assumptions, but is effective on many complex datasets. For more details look up C4.5.

# Neural Network

Training data, **S**

	<b>x</b>	<b>x</b>	<b>y</b>
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Set weights one to four =1.  
Set weight five = 3.  
Set weight six = -2.  
Heaviside activation function



Example (input 2):

	<b>Neuron 1</b>	<b>Neuron 2</b>	<b>Neuron 3</b>
Sum	1	-1	0
Output	1	0	1
Fires?	Yes	No	Yes

A supervised learning algorithm. Inspired by the biology of the brain. Involves setting connection weights based on input patterns, in order to cause specific neurons in the network to fire. The observant will notice that this network has been taught logical or (XOR).

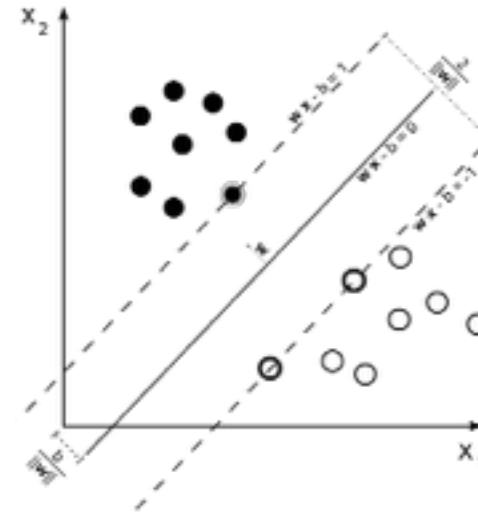
The table on the left shows the training data. The table on the right shows how the network output is computed for the second training pattern. The Neurons use a Heaviside activation function, so fire if the weighted sum they compute is greater than 1, else they do not fire.

Real world neural networks consist of thousands on nodes, organised into multilayered structures. Neural networks are excellent pattern recognition systems, and work well in the presence of noisy data or missing values. However the larger the network, the longer it takes to train. Examples of neural network include perceptrons, multi-layered perceptrons, self-organising maps, deep neural networks.



# SVM

- **Support Vector Machine.**
- **Builds a maximal margin hyperplane.**
- **Support vectors are examples on border.**
- **Examples mapped into data space and assigned to one side of the border.**
- **Can perform non-linear classification.**



# Naïve Bayes

Whats the probability of  $y_i$  given an observation  $x_j$ ?  $\longrightarrow P(y_i|x_j)$

Can be calculated from Bayes' theorem: 
$$P(y_i|x_j) = \frac{P(x_j|y_i)P(y_i)}{P(x_j)},$$

where  $P(y_i)$  is the probability of an observation belonging to class  $y_i$ ,  $P(x_j)$  is the probability of  $x_j$  occurring, and  $P(x_j|y_i)$  is the probability of  $x_j$  occurring for class  $y_i$ .

Generalising for observations with multiple variables, the probability of class  $y_i$  given  $x_j = \{(x_j^1), ..., (x_j^n)\}$  is written as,

$$\begin{aligned} P(y_i|x_j) &= P(x_j^1|y_i) \times P(x_j^2|y_i) \times \dots \times P(x_j^n|y_i) \\ &= \prod_{i=1}^n P(x_j^i|y_i). \end{aligned}$$

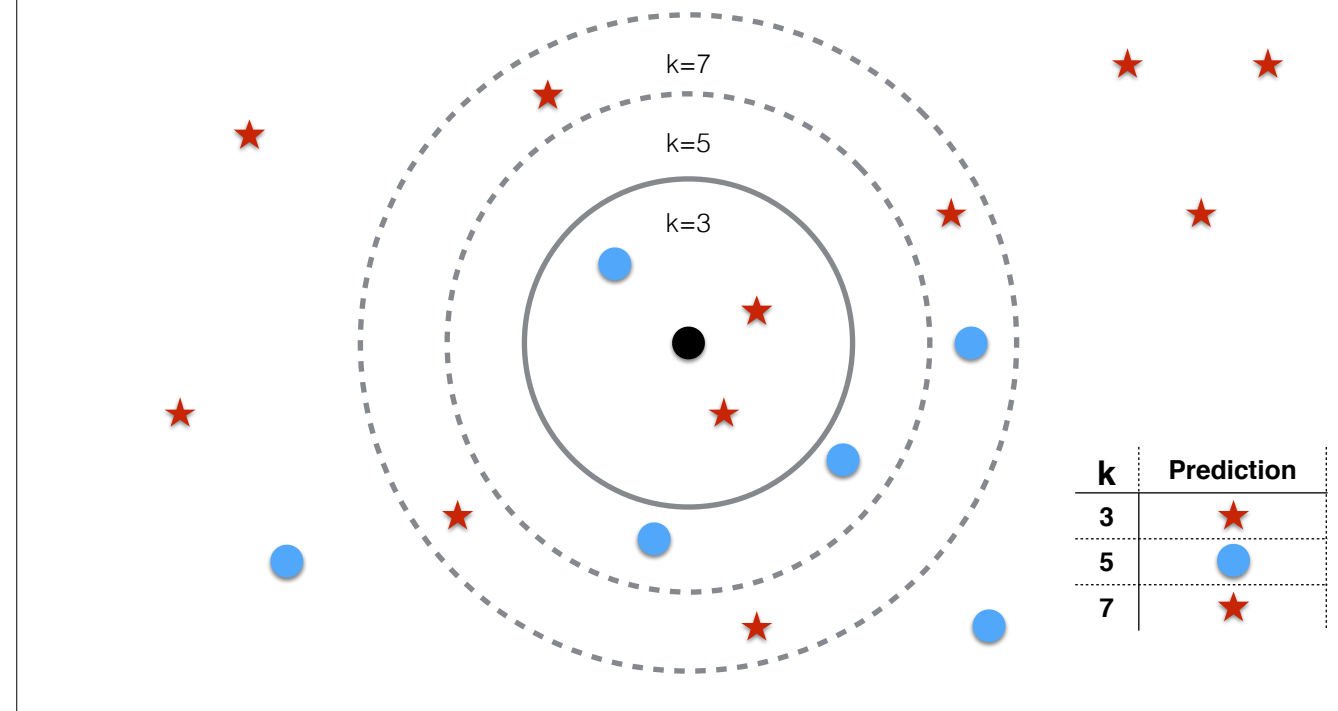
The classifier then picks the maximum a posteriori decision rule:

$$\text{classify}(x_j^1, \dots, x_j^n) = \underset{y_i}{\operatorname{argmax}} p(y_i) \prod_{i=1}^n P(x_j^i|y_i).$$

- **Probabilistic classifier.**
- **Assumes features of independent.**

The most simple probabilistic classifier. If features are not independent, this classifier will perform particularly poorly.

# Nearest Neighbours



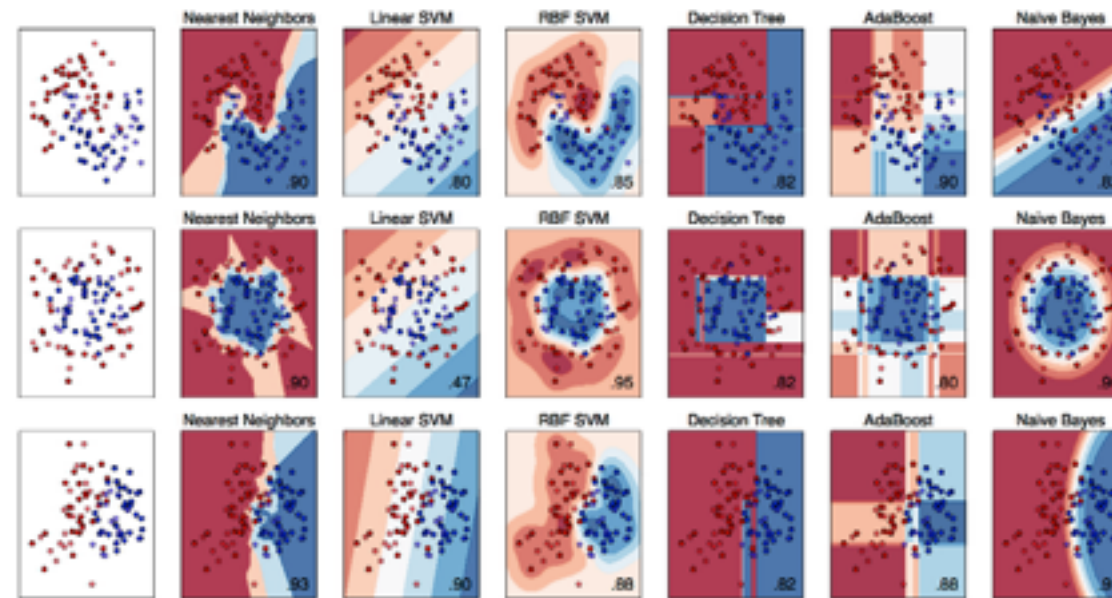
Nearest neighbour models are amongst the most simplest in machine learning. They can be used as unsupervised, semi-supervised and fully supervised learners. Nearest neighbour models such as K-nearest neighbours (k-NN), simply classify data according the known labels of their k-nearest neighbours in data space. Nearest must be computed using a distance metric, usually the euclidean distance (though others such as the Manhattan distance and Mahalanobis distance are also popular).

The main difficulty in using nearest neighbour models, is in deciding the best parameters to use. For instance which distance metric is best to use, and what value of k is optimal. Usually these have to be determined through experimentation (sorry, no quick fix). Nearest neighbour models are not good for all data sets however, and you need to explore their usefulness for your data in detail.

Other types of cluster based classifier worth looking at, include K-means (creates cluster centre around means centroids).

I've written some matlab scripts that do basic k-nn classification, and provided some data to test with. Here I run the script as part of a demonstration. Run the script: Matlab\_KNN\_Example.m

# Another look

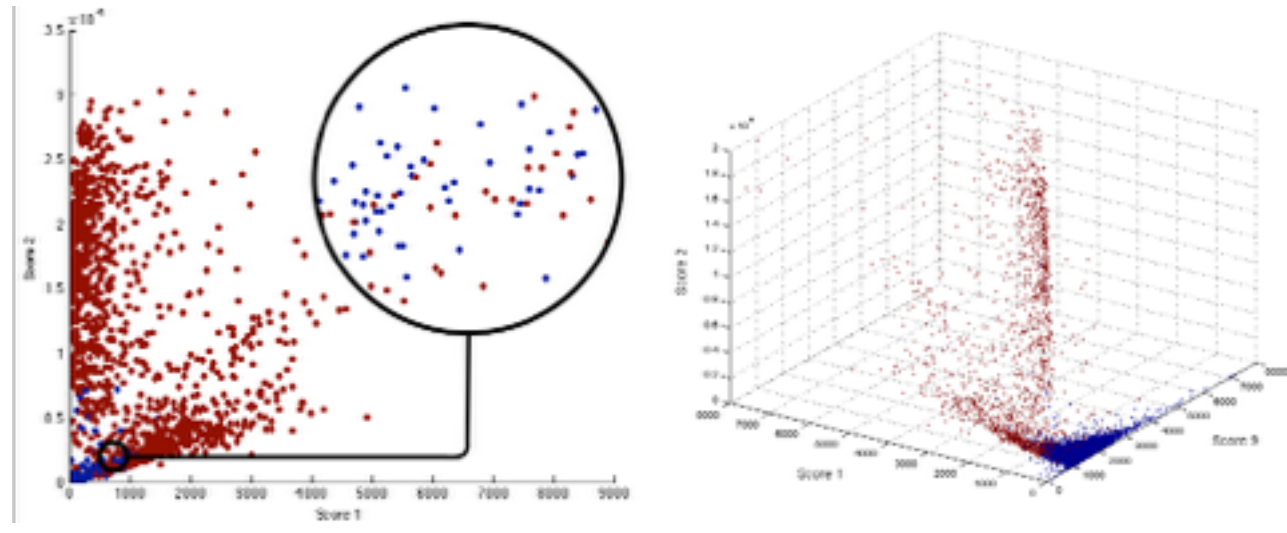


Maybe after talking about a few different algorithms, their performance shown here makes a little more sense?

I've written some python scripts that run a number of machine learning classifiers and compare the results. I've also provided some data to test with. Here I run the script as part of a demonstration. Run the script: SciKitLearnTest.py

# Explore your data

- How is the data structured?



It's important to try and explore you data, understand it's geometry if possible. This kind of knowledge can help you choose an appropriate classifier. For instance if your data is relatively spare, a clustering algorithm might work well. If the data is dense, then a clustering approach will likely perform very badly. Maybe try a density separation approach if the following hold true (copied from Wikipedia, cheers wiki!):

## Smoothness assumption

Points which are close to each other are more likely to share a label. This is also generally assumed in supervised learning and yields a preference for geometrically simple decision boundaries. In the case of semi-supervised learning, the smoothness assumption additionally yields a preference for decision boundaries in low-density regions, so that there are fewer points close to each other but in different classes.

## Cluster assumption

The data tend to form discrete clusters, and points in the same cluster are more likely to share a label (although data sharing a label may be spread across multiple clusters). This is a special case of the smoothness assumption and gives rise to feature learning with clustering algorithms.

## Manifold assumption

The data lie approximately on a manifold of much lower dimension than the input space. In this case we can attempt to learn the manifold using both the labeled and unlabelled data to avoid the curse of dimensionality. Then learning can proceed using distances and densities defined on the manifold. The manifold assumption is practical when high-dimensional data are being generated by some process that may be hard to model directly, but which only has a few degrees of freedom. For instance, human voice is controlled by a few vocal folds, and images of various facial expressions are controlled by a few muscles. We would like in these cases to use distances and smoothness in the natural space of the generating problem, rather than in the space of all possible acoustic waves or images respectively.

I've written some matlab scripts that do basic data exploration, and provided some data to explore. Here I run the script as part of a demonstration. Run the script: Matlab\_TwoClass\_3DScatter.m

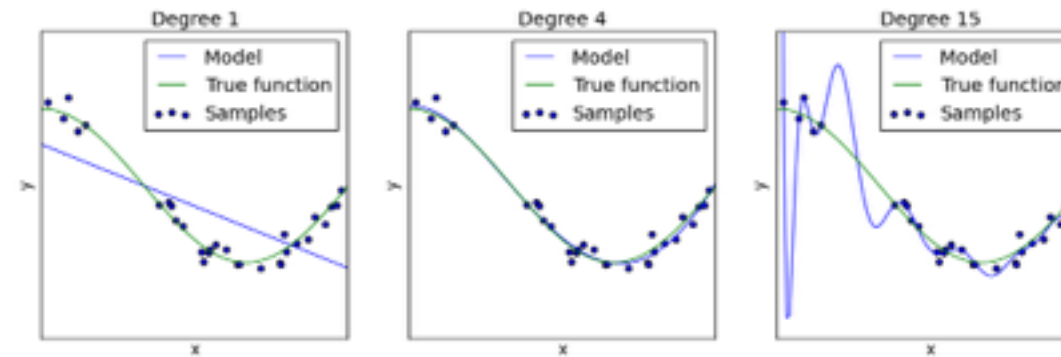


# So you've chosen wisely

Now we build and test the classifier.

I assume you've chosen an algorithm to try. Just don't become too attached to that particular algorithm, it isn't the holy grail after all! Treat like a tool which you hypothesise might be the right one for the job. But be prepared to throw it away if another works better.

# Over-training



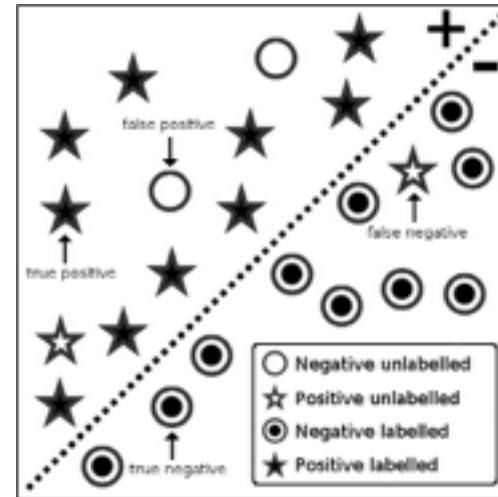
- **Make sure training set is representative.**
- **Make sure test set is representative.**
- **Always evaluate performance on test set!**
- **Your classifier must be able to *generalise*.**

If you keep pushing to obtain 100% accuracy on training data, your classifier will overfit. In other words it will lose its capacity to generalise beyond the examples it's been taught, meaning that in practice its performance will be poor. A simple example is as follows. Imagine you would like to train a system to classify pulsar signals as pulsar/non-pulsar based on plots of signal intensities over time.

You train your classifier until its accuracy is 100%, then present it with a new example that you KNOW to belong to a pulsar. But the classifier gets it wrong. So what happened? In this case the classifier learned to recognise the exact pulse patterns in the training set, as opposed to learning to differentiate pulsar/non-pulsar.

Using separate test data sets helps us to avoid these problems, since we can check if our classifiers have overfit.

# Evaluation



		Actual Class	
		+	-
Predicted Class	+	True Positive	False Positive
	-	False Negative	True Negative

This is the confusion matrix used to evaluate machine learning classifiers for binary classification problems. The matrix can be extended to multi-class problems easily just by adding extra rows and columns as required.

The picture shows how the confusion matrix should be interpreted with respect to a simple linear decision boundary built by a classifier.



# Evaluation Metrics

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Positives} + \text{Negatives}}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

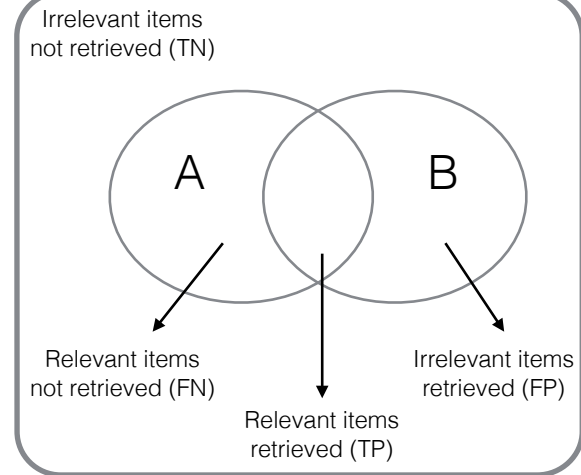
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{G-Mean} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}$$

A = items relevant

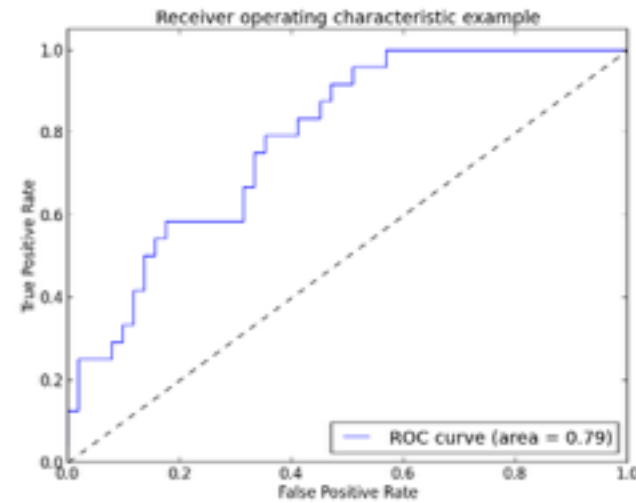
B = items retrieved



There are many more metrics out there. There's are the standard ones that you should be familiar with. These will equip you with the information you need in order to judge how effective you machine learning system is. The set theory interpretation hopefully helps you understand their intuitive meaning

Note that precision and recall are inversely related. As recall goes up, precision goes down, and vice versa.

# Evaluation



- All classifiers trade-off recall and the false positive rate.
- Beware results which claim 100% recall.
  - Over-trained? Unrepresentative training set? Rubbish test set?

Receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The true-positive rate is recall. The false-positive rate is the false alarm rate. The ROC curve is thus the recall as a function of false alarms.

# Better sources of Info

## Books:

- “*Machine Learning*”, Mitchell, T., 1997, <http://www.cs.cmu.edu/~tom/mlbook.html>
- “*Data Mining*”, Witten, I. H., et. al, 2011, <http://www.cs.waikato.ac.nz/ml/weka/book.html>
- “*Artificial Intelligence*”, Norvig, P. & Russel, S., 2009, <http://aima.cs.berkeley.edu>
- “*Advances in Machine Learning and Data Mining for Astronomy*”, Way, M. J., et. al, 2012, <http://pubs.giss.nasa.gov/abs/wa01400w.html>
- “*Pattern Recognition and Machine Learning*”, Bishop, C. M., 2007, Springer.

## Toolkits:

- Matlab: [http://www.mathworks.co.uk/help/stats/index.html#btq\\_uq5](http://www.mathworks.co.uk/help/stats/index.html#btq_uq5)
- SciKit-Learn: <http://scikit-learn.org/stable/> + AstroML <http://www.astroml.org>
- Weka: <http://www.cs.waikato.ac.nz/ml/weka/>
- MOA: <http://moa.cms.waikato.ac.nz>
- Orange: <http://orange.biolab.si>
- R : <http://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/>



# More sources...

**Lectures:**

- Machine Learning, Andrew Ng, Stanford: <https://itunes.apple.com/gb/itunes-u/machine-learning/id384233048?mt=10>
- Coursera course on machine learning: <https://www.coursera.org/course/ml>
- Learning from data: <https://work.caltech.edu/telecourse.html>
- Statistical Learning: <https://class.stanford.edu/courses/HumanitiesScience/StatLearning/Winter2014/about>
- Introduction to Data Science: <https://www.coursera.org/course/datasci>

**Data Sets & Resources:**

- UCI Data Repository: <http://archive.ics.uci.edu/ml/datasets.html>
- KDNuggets: <http://www.kdnuggets.com>
- Kaggle: <https://www.kaggle.com>



