Introduction to Machine Learning

John Quinn

Department of Computer Science
Makerere University, Kampala, Uganda

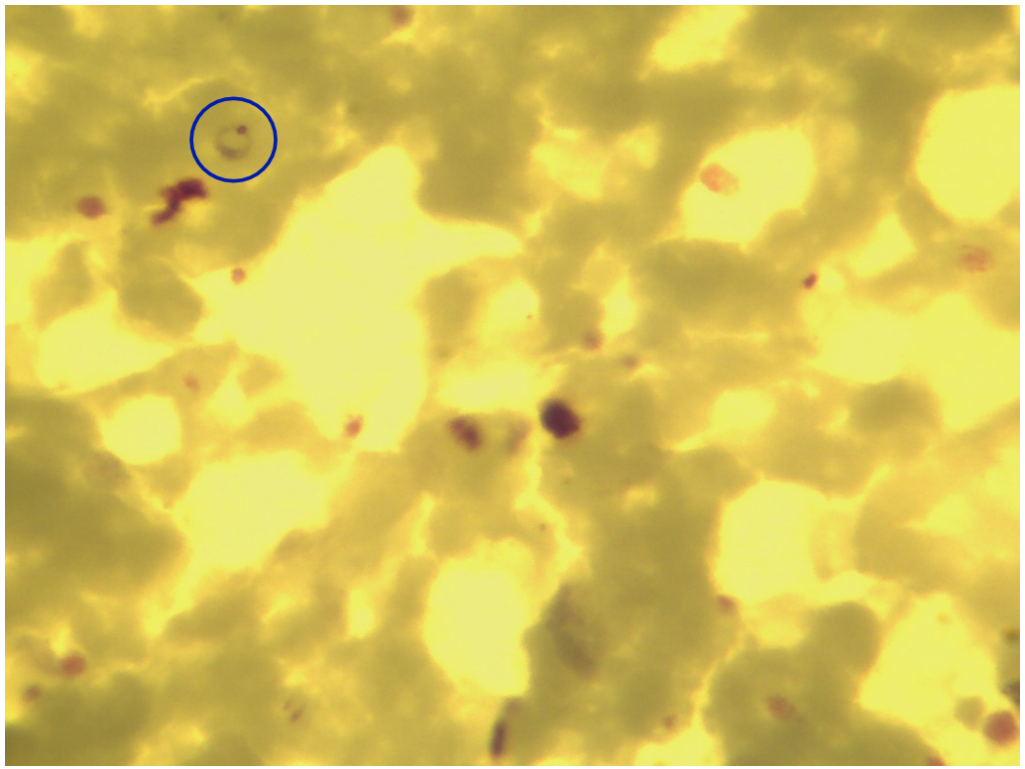
PASCAL Bootcamp 2011

Programming with uncertainty

- In order for machines to carry out high level tasks, they need to be able to deal with uncertainty.
- Uncertainty might come from different sources:
  - Problem might be defined incompletely in terms of a limited number of examples: e.g. object recognition in images.
  - Scale of the problem is too big: e.g. shortest tour of 100 locations – more permutations ($\approx 10^{158}$) than there are atoms in the universe ($\approx 10^{80}$)!
  - Inherent uncertainty in predicting a chaotic system: e.g. weather forecasting.
- We are used to writing software with inputs and outputs, and a clear relationship between the two.
- Solution: get the machine to learn what to do.
- This session will be a ‘cookbook’ of techniques for getting ML systems to work.

Examples of uncertainty

We might want to detect malaria parasites in images of blood cells:



This image could have been any one of $255^{1024*768*3}$ possible manifestations: obviously can't just memorise.

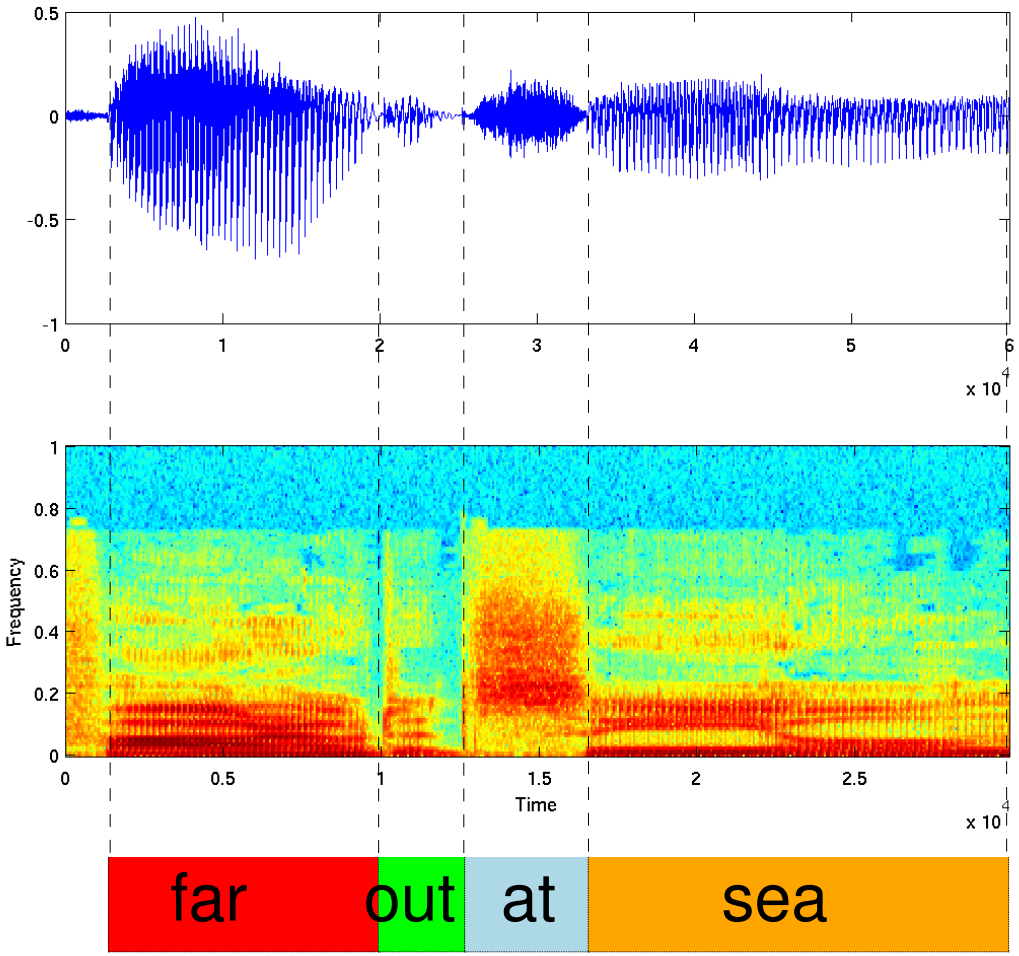
Examples of uncertainty

Given a CCTV video stream of traffic, can we work out whether there is normal traffic flow, congestion or an accident?

If so, is it then possible to predict traffic conditions in the future, based on patterns of traffic density in the past?

Examples of uncertainty

In this type of problem (speech-to-text), we take a recorded speech signal and try to infer the words it contains.



Inductive bias

- Without making some assumptions, we can't learn anything and couldn't deal with these problems.
- However, in most of the domains we deal with practically there are patterns/regularities in the data we observe.
- Humans are good at seeing patterns, even when they don't really exist.

We can usually make two kinds of assumptions:

- **Smoothness:** the output of the functions we try to approximate change little if the input changes little.
- **Simplicity:** we favour simple explanations/models over complicated ones.

Thus we are biased towards these kinds of hypotheses.

Handwritten character recognition

- Imagine you have a database with images of handwritten characters, and want to be able to take a new handwritten character and decide automatically which one it is (a **classification** task).

http://yann.lecun.com/exdb/mnist

- How would you go about this? How about just distinguishing between 1's and 2's?

Printed character recognition

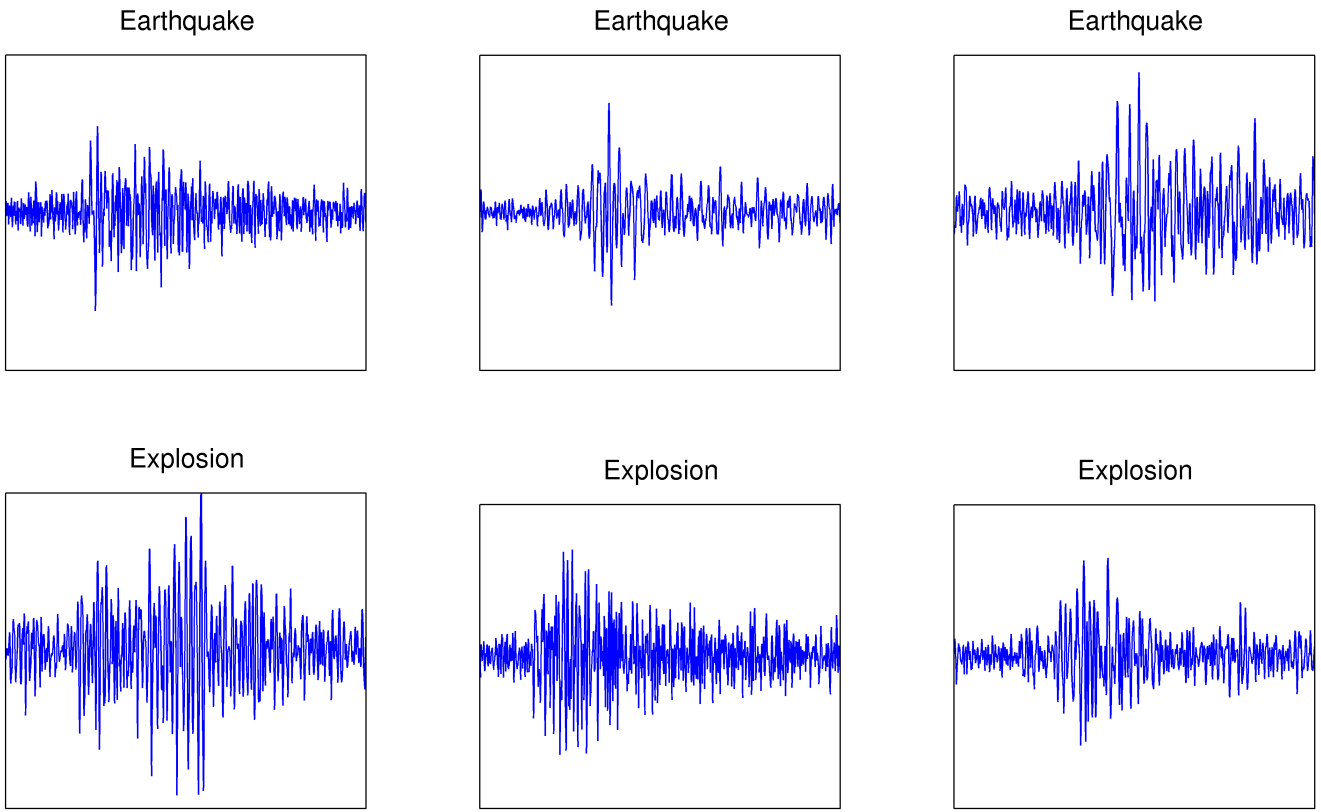
Recognising non-handwritten characters can also be tricky...

What is the essential property that characterises a letter ‘A’? Is there anything invariant in all these images, and can we write it down mathematically?

[Hofstader, On Seeing ‘A’s and Seeing As]

Earthquakes and explosions

Given measurements of vibrations in the ground, is it an earthquake or a mining explosion?



Data: http://anson.ucdavis.edu/~shumway/

Crop disease diagnosis

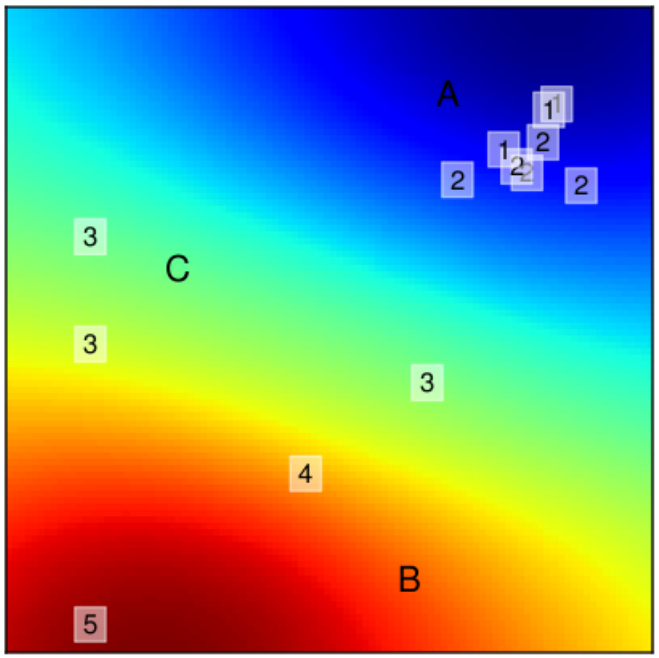
Given images of crops taken from a cameraphone, is it possible to diagnose disease?

Healthy cassava

Cassava mosaic disease

Crop disease density

Given observations of diseased crops, we might want to predict what is happening across the map:

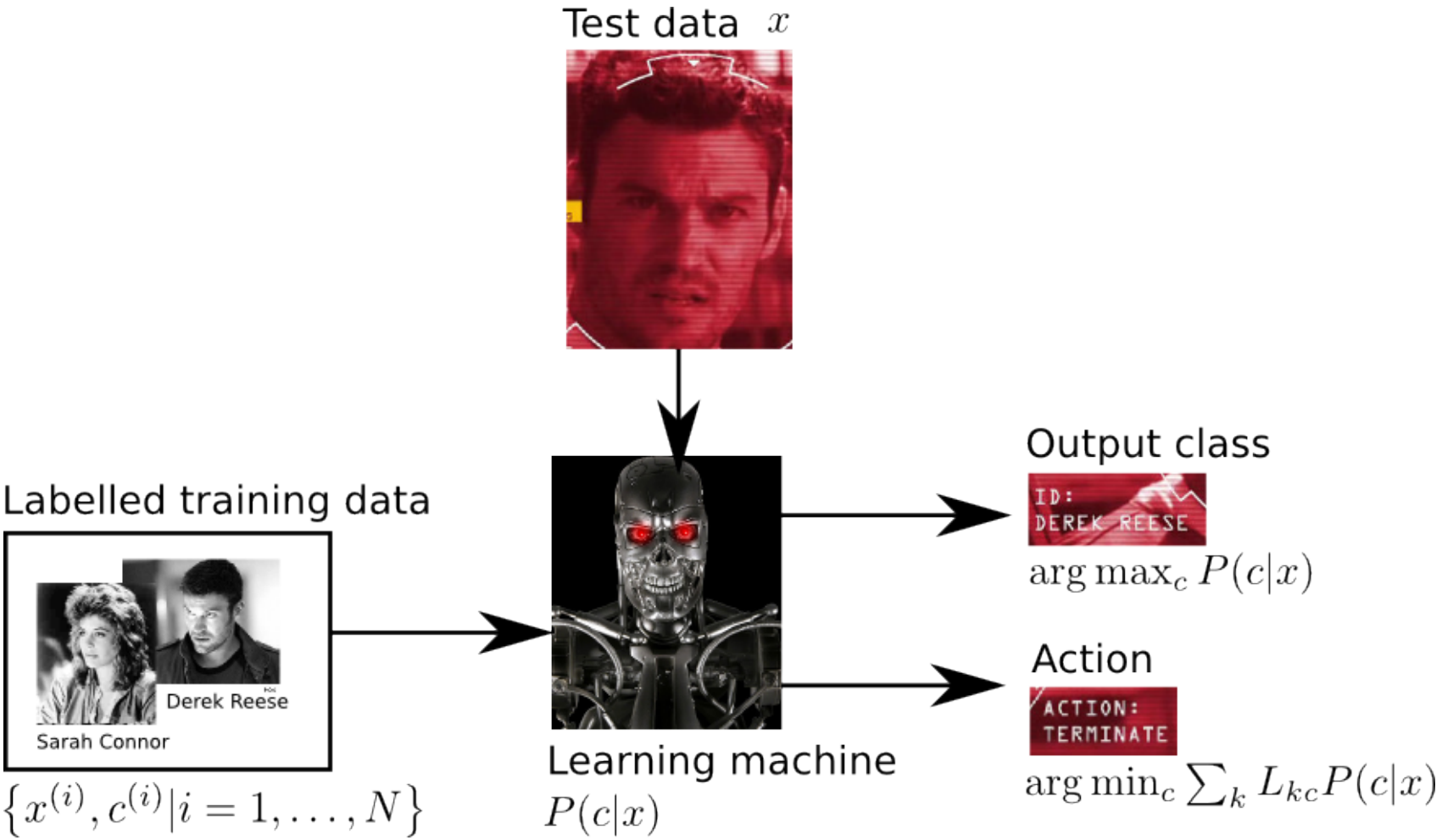


For example, if the light boxes represent disease levels of crops surveyed, what might the predicted disease level be at locations A, B, and C?

Face recognition

Many of these preceeding examples are basically learning a mapping between inputs and outputs. Where there is training data (examples of inputs and outputs), approximating the function is known as **supervised learning**.

A common supervised learning setup

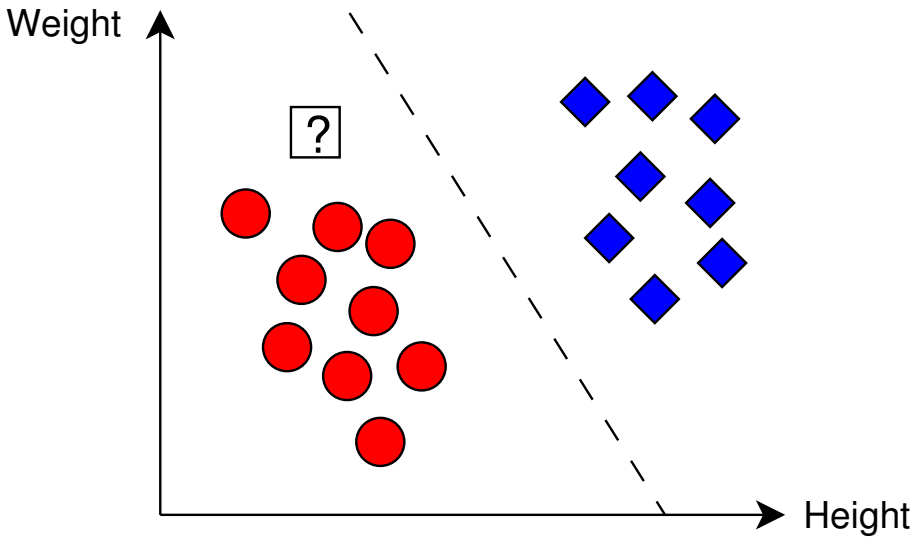


How can $P(c|x)$ be learnt, or even represented?

Linear decision boundaries

Imagine you have two dimensional data $\{x^{(i)} \in \mathbb{R}^2_+\}$, the heights and weights of different animals.

The red circles are all your examples of $c^{(i)} = 1$ (snarks). The blue diamonds are all your examples of $c^{(i)} = 2$ (boojums). Given a new datapoint, which type of animal is it?

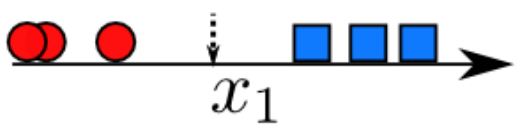


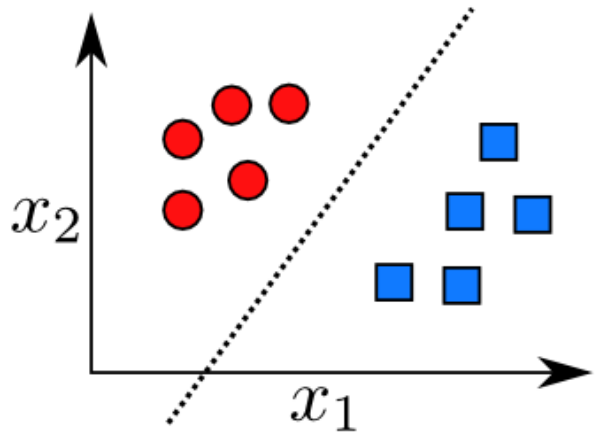
We might answer this by proposing a linear decision boundary (denoted here by the dashed line). The set of points on the line are the inputs for which both classes are equally likely.

Linear decision boundaries

In 1D, a decision boundary can be described with the equation $x_1 w_1 + b = 0$. If for some x_1 this quantity $(x_1 w_1 + b)$ is positive, we assign to "blue squares", if negative we assign to "red circles".

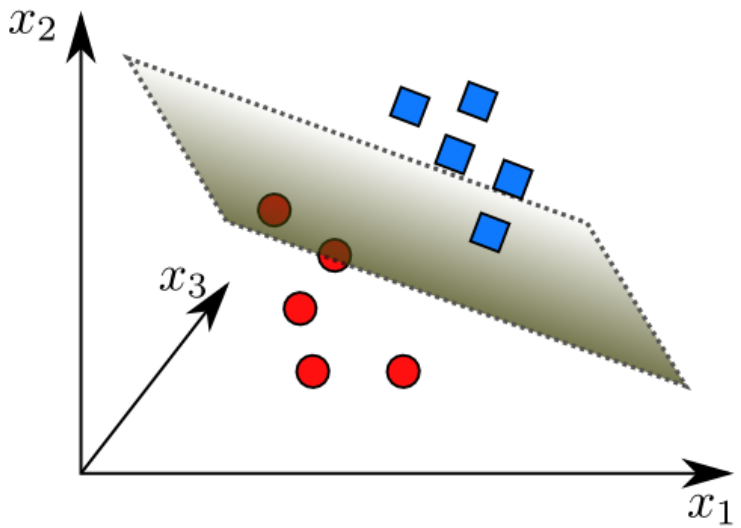
In 2D, the boundary is at $w_1 x_1 + w_2 x_2 + b = 0$.





Linear decision boundaries

With three dimensions the decision boundary is a plane (in higher dimensions a hyperplane).



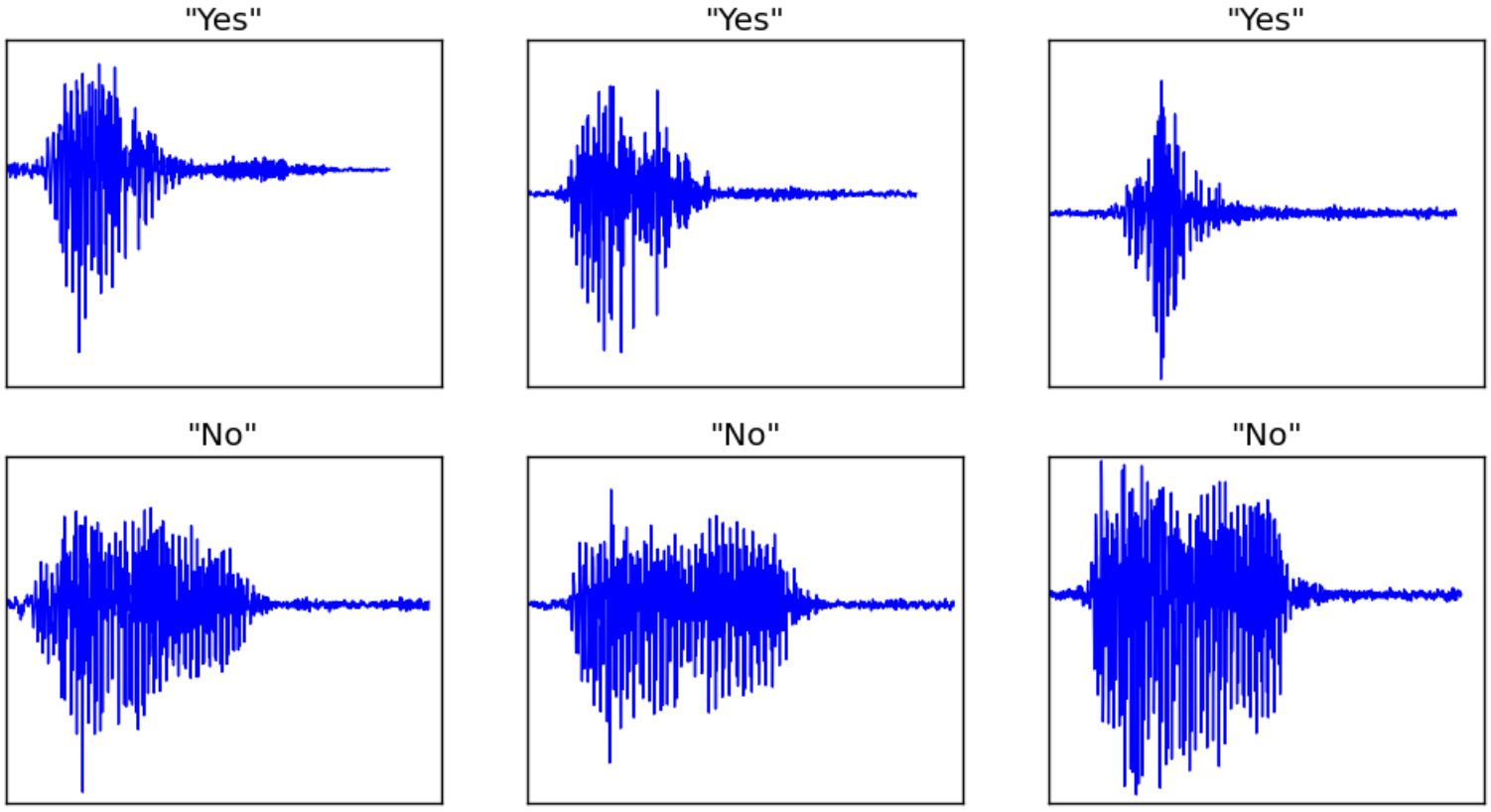
This can be expressed in vector notation as $w^\top x + b = 0$.

Moving x perpendicularly away from the decision boundary, the quantity $w^\top x + b$ becomes increasingly positive in one direction, and increasingly negative in the other direction.

Speech recognition example

Take a non-trivial example, distinguishing between speech recordings of “yes” and “no”.

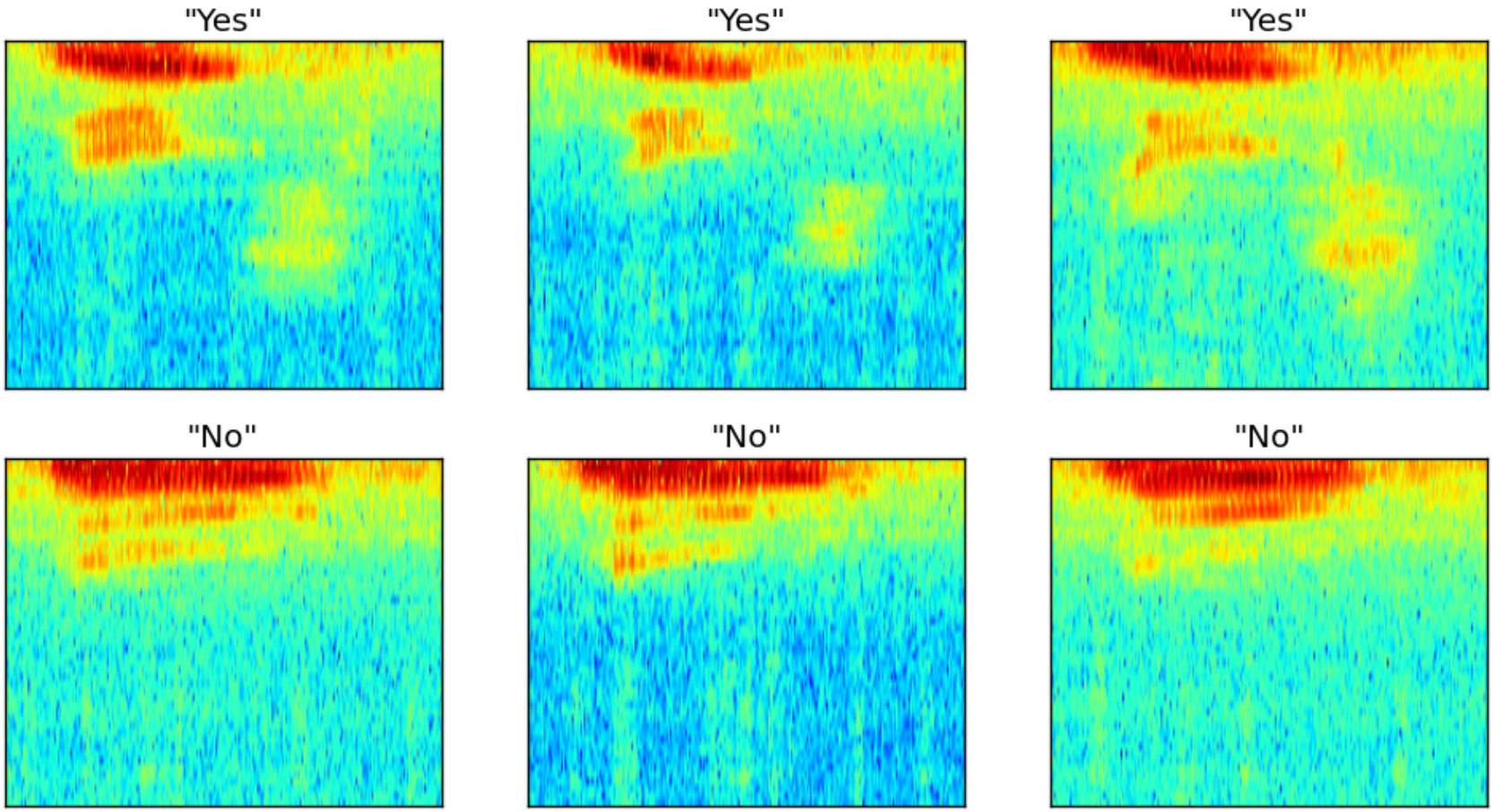
The raw data for a few training examples looks like this:



Speech recognition example

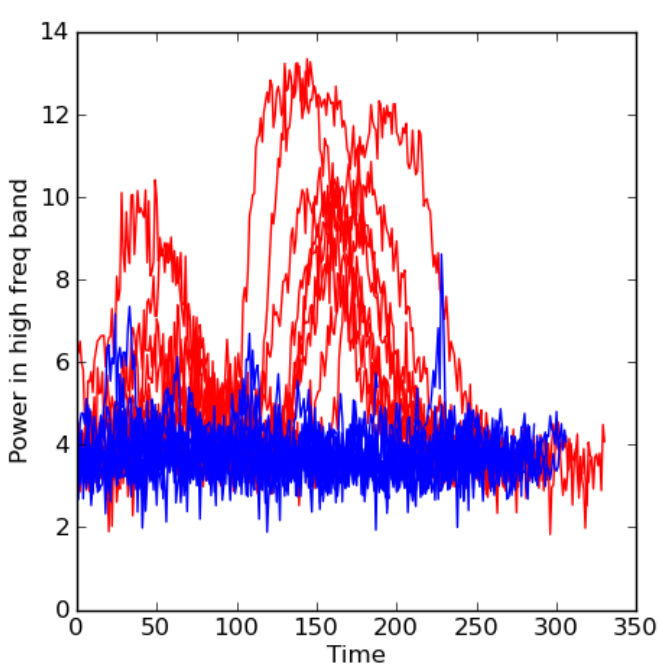
We can try to represent this data in a more meaningful way.

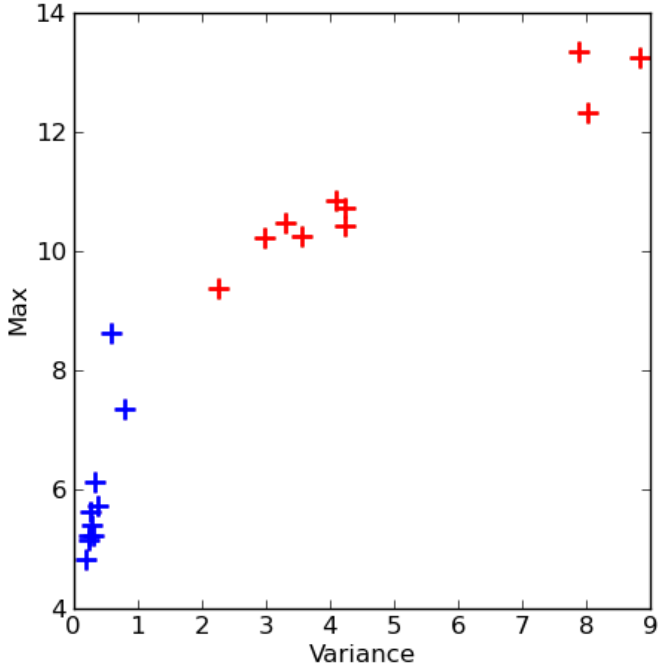
For example, with a spectrogram (time against frequency):



Speech recognition example

Averages along high frequency bands are plotted on the left. (red denotes “yes”, blue denotes “no”, 10 examples of each)





If we calculate two features, the mean and variance, we can show examples on a 2D plot (right).

Back to linear classification...

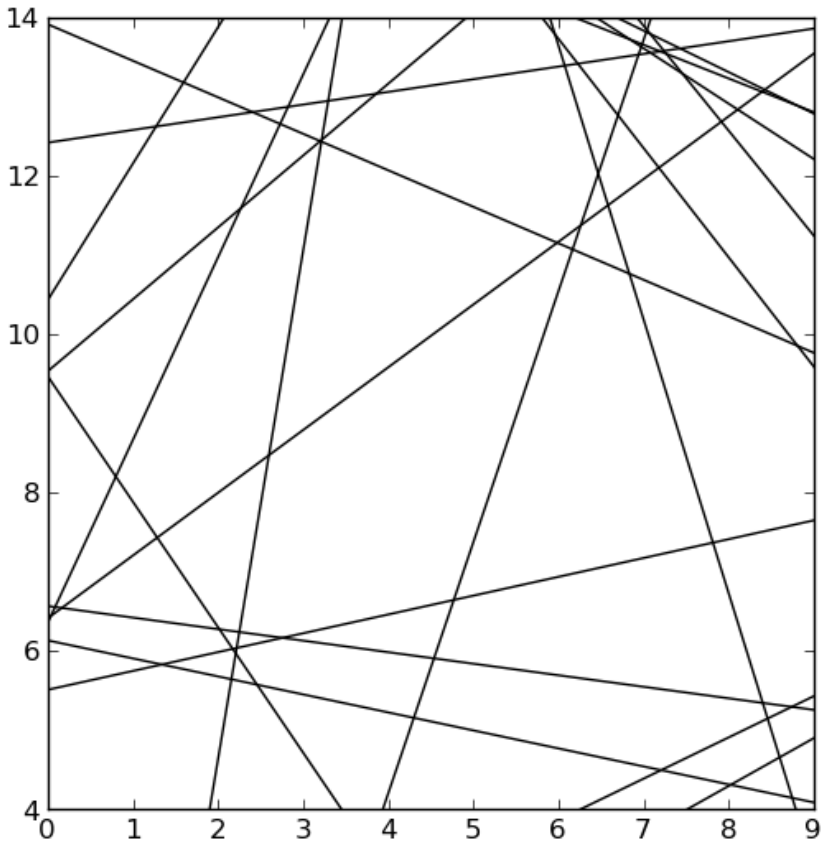
Speech recognition example

How could we choose a linear decision boundary? How about choosing lots of random lines, then seeing which ones match the data.

Two things to sort out: how to generate lines, how to score each line to see if it fits the data.

Speech recognition example

Generation: a decision boundary is represented by a vector and an offset. A single point can represent a plane. Take the vector from the origin, then plane is perpendicular. So generate many points, and we get the corresponding line.



Speech recognition example

We want to find a 'likelihood' of the decision boundary, given the data.

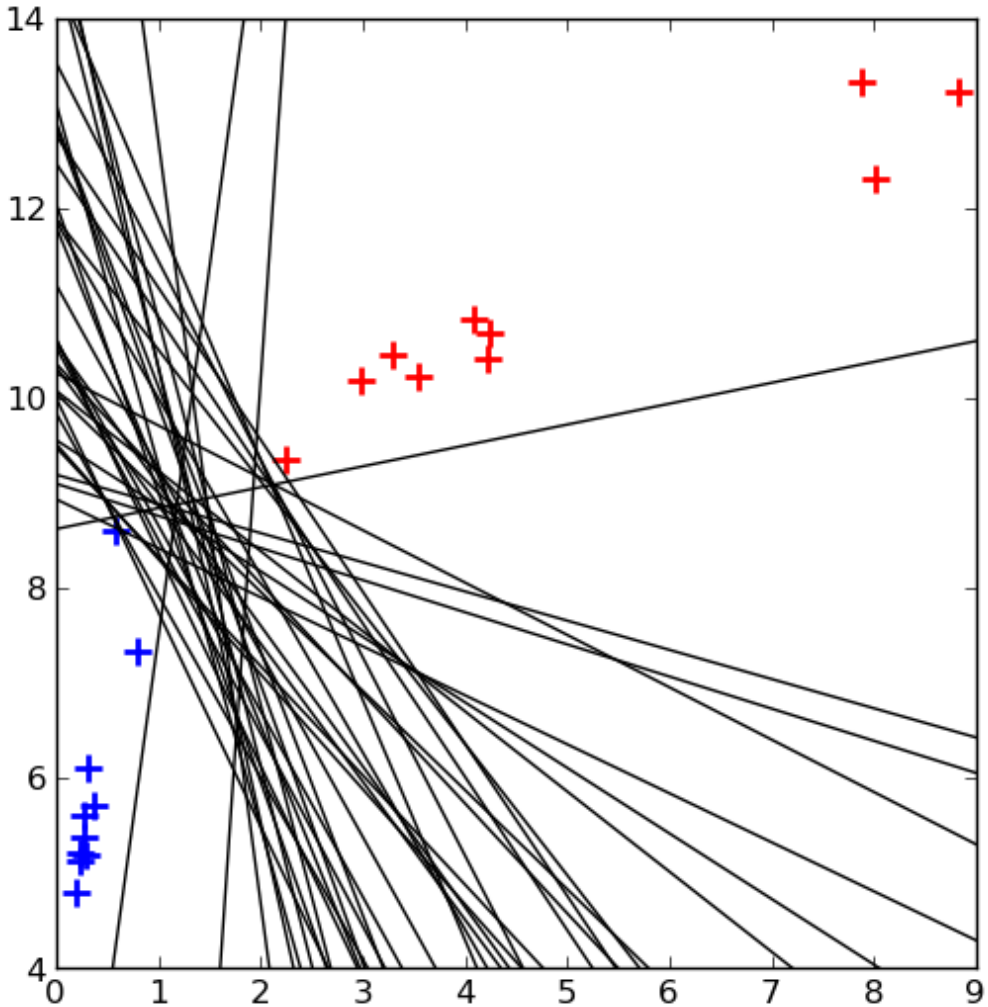
We can use a simple measure: if all the "yes"s are on the positive side of the boundary, and all the "no"s are on the negative side, then the boundary has a likelihood of one given the observed data. Otherwise it has a likelihood of zero.

$$P(c^{(i)} = c | x^{(i)}, w, b) = \begin{cases} 1 & c(w^\top x^{(i)} + b) > 0 \\ 0 & c(w^\top x^{(i)} + b) < 0 \end{cases}$$

where $c \in \{-1, +1\}$ and $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$.

Speech recognition example

If we generate many lines, we find that several of them have likelihood 1:



Bayes rule

Bayes rule gives us a way to reason about which parameters are most likely. We use it to obtain a posterior distribution over all possible parameter values:

$$\begin{aligned} P(\text{parameters}|\text{data}) &= \frac{P(\text{parameters})P(\text{data}|\text{parameters})}{P(\text{data})} \\ &\propto P(\text{parameters})P(\text{data}|\text{parameters}) \end{aligned}$$

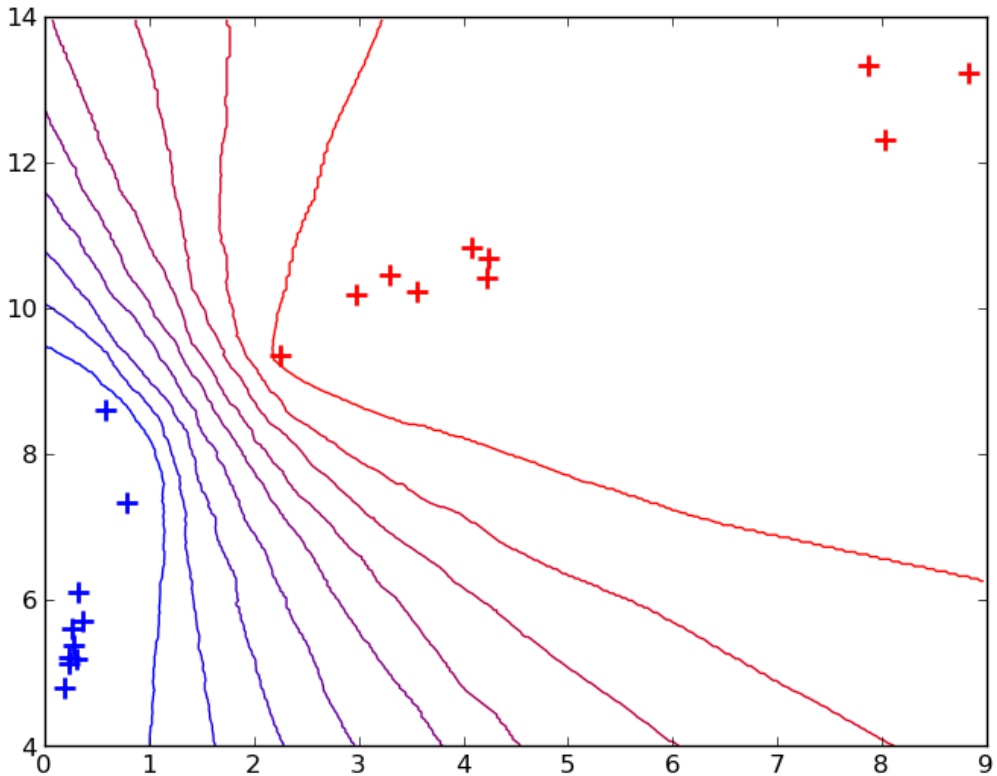
We can then evaluate the probability of seeing any new data values by integrating with respect to the posterior:

$$P(\text{testdata}|\text{data}) = \int_{\text{parameters}} P(\text{testdata}|\text{parameters})P(\text{parameters}|\text{data})$$

Our example: parameters=$\{w, b\}$, data=$\left\{x^{(i)}, c^{(i)} | i = 1, \ldots, N\right\}$.

Bayesian linear classification

With many examples drawn from the prior in the previous example, we can find a posterior (Monte Carlo inference). This contour plot shows the posterior probability of an input being a "yes" (blue represents zero probability, red represents probability 1).



Soft likelihood

Can modify the likelihood so that it is around 0.5 when points are close to the boundary, and getting closer to 0 or 1 as they are further away. There are a class of sigmoid functions which can be used for this.

There are ways of finding parameters which maximise this type of likelihood, called logistic regression.

More than one linear boundary: decision tree learning

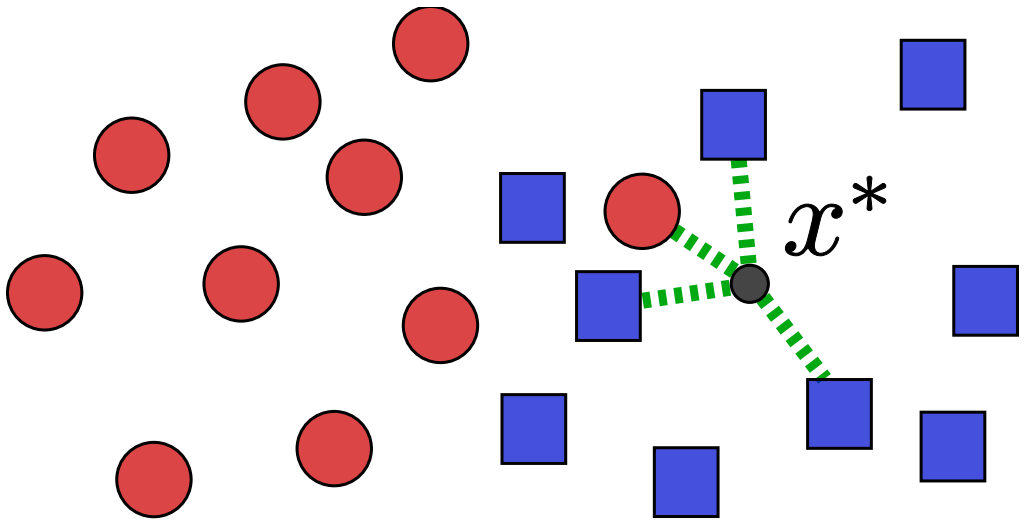
There may be several classes: we can partition the space with a hyperplane more than once.

We can also think about doing this when w is limited to point masses: we then have rectangular/cuboid regions, aligned to the axes.

The position of each boundary can be optimised with mutual information. If we greedily select the best split at each step, we are using a technique known as decision tree learning.

K-nearest neighbour

Simple idea for a classifier: assign whichever class the nearest training data point has (or in general, the majority class in the nearest k neighbours).

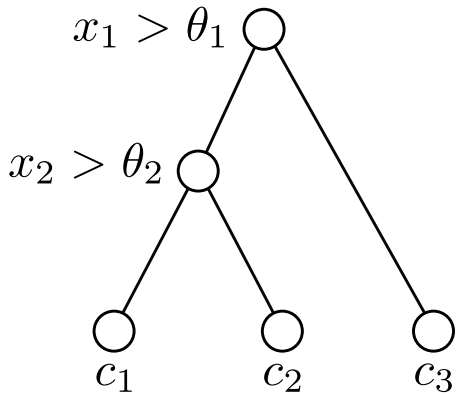


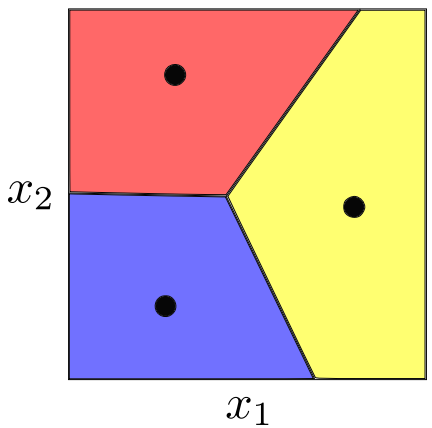
For real-valued data, Euclidean distance is a common choice: $d^2(a,b) = (b-a)^\top(b-a)$

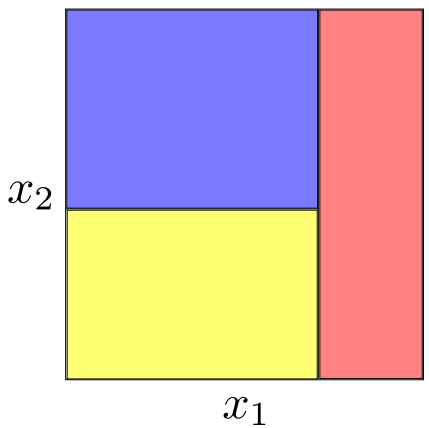
Decision boundaries with neighbours and trees

Decision boundaries for nearest neighbour are defined by distance (Voronoi in 2D), whereas for trees boundary is axis-aligned cuboid regions.

$$\mu_1 = \{x_1^{(1)}, x_2^{(1)}\}$$
$$\mu_2 = \{x_1^{(2)}, x_2^{(2)}\}$$
$$\mu_3 = \{x_1^{(3)}, x_2^{(3)}\}$$







Regression

In the previous examples, the output of the function we've been trying to learn is a class. Sometimes, the output variable is a number, in which case we have a regression problem.

Least squares is a commonly used method for this. We try to find a set of weights, so that a weighted average of the inputs X gives the output as closely as possible. That is, we want to find w such that for $\hat{y} = Xw$, the residual $\hat{y} - y$ is minimised.

(This is done with the function `linalg.lstsq(X,y)` in numpy.)

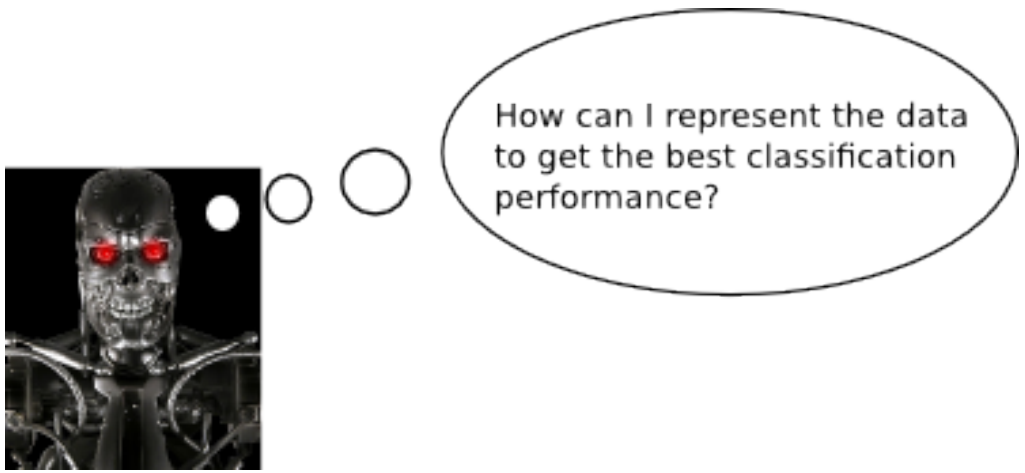
Regression in time series

- Least squares on a series of points gives a method of predicting future points in a time series.
- Patterns exist in many types of sequential data, for example share prices. Here we might want to predict what the next value in the sequence will be.

- More commonly, we try to predict a range of values with different certainties.

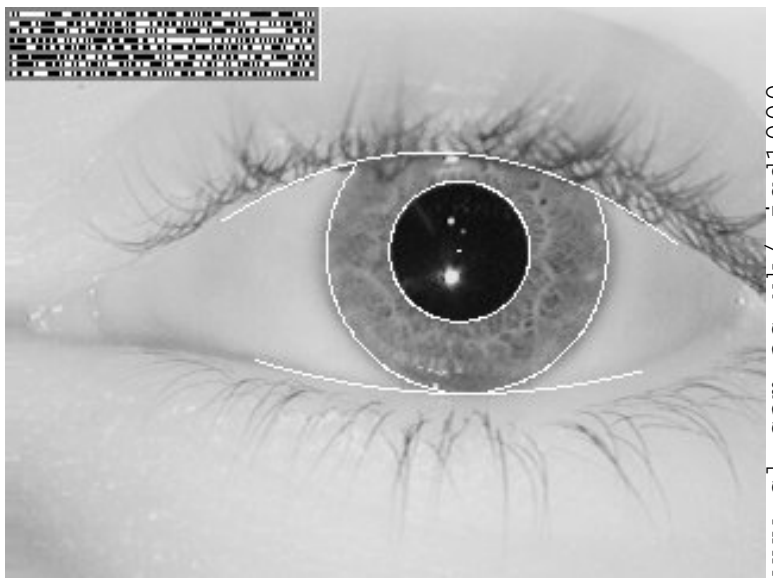
Representation of data

- As we saw in the speech example, the way the data is represented makes a big difference.
- We'll next look at techniques for changing the data to suit particular tasks.



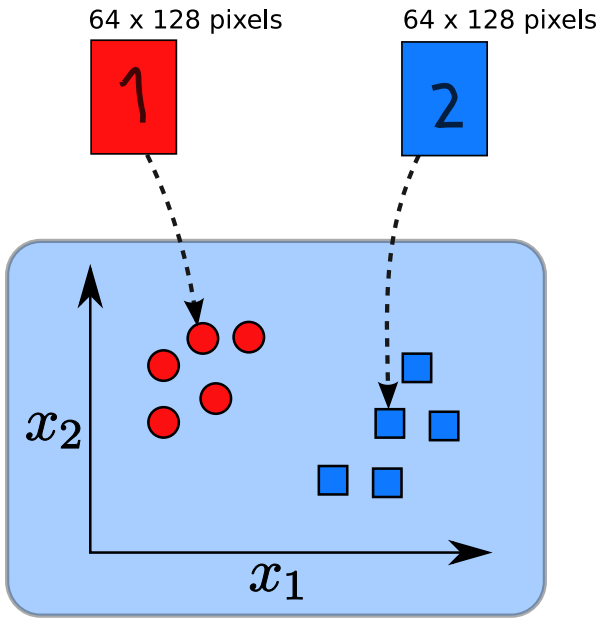
Representation of data

- If you have a hard problem to solve, convert it into an easy one and solve that.
- There might be a simple way to represent a complex dataset, making pattern recognition easy.
- For example, the “iriscode” below contains a compact summary of information about patterns in the iris, which makes it possible to identify individuals:



Dimensionality reduction

- Returning to the handwritten digit identification problem, part of the difficulty is that the raw data is high dimensional (e.g. 64×128 pixels $= 8192$ dimensions).

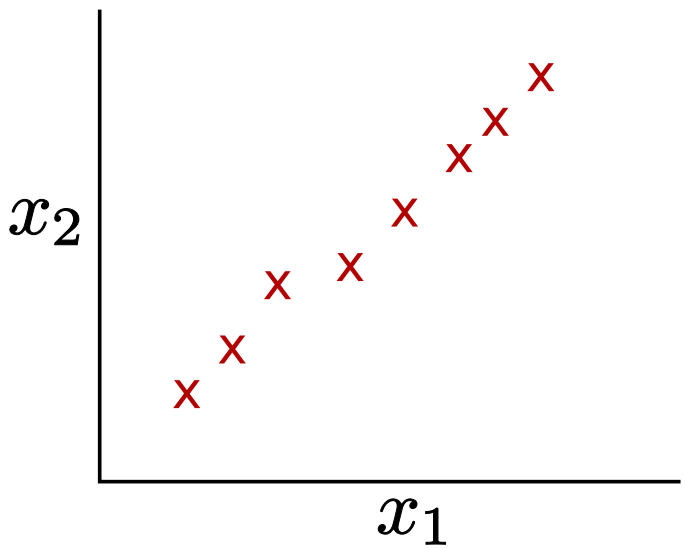


- If there was some way of mapping each image to a point in a lower dimensional space (2D in this example), then the problem might become easier.

Dimensionality reduction

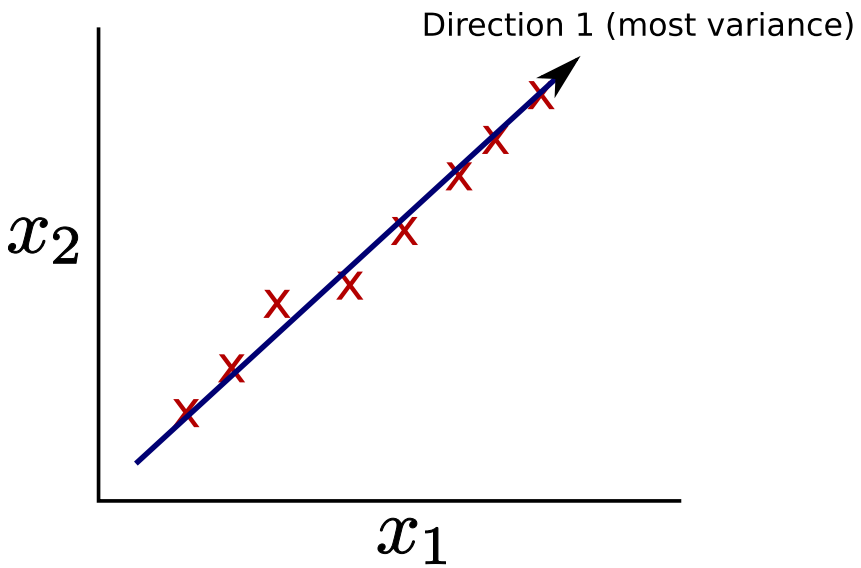
Each of these data points needs two coordinates to specify its position: $\{x_1, x_2\}$.

What system could you use to specify the positions approximately using only one number?



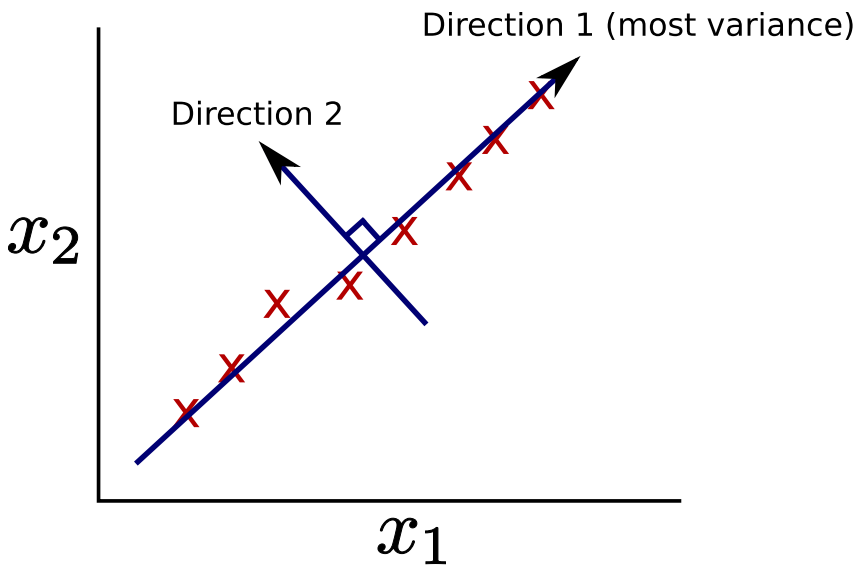
Principal components analysis

One way of doing this is to find the mean of the data, and the first principal component (the direction of maximum variance). We can then specify how far along the line each point is, giving its approximate position.



Principal components analysis

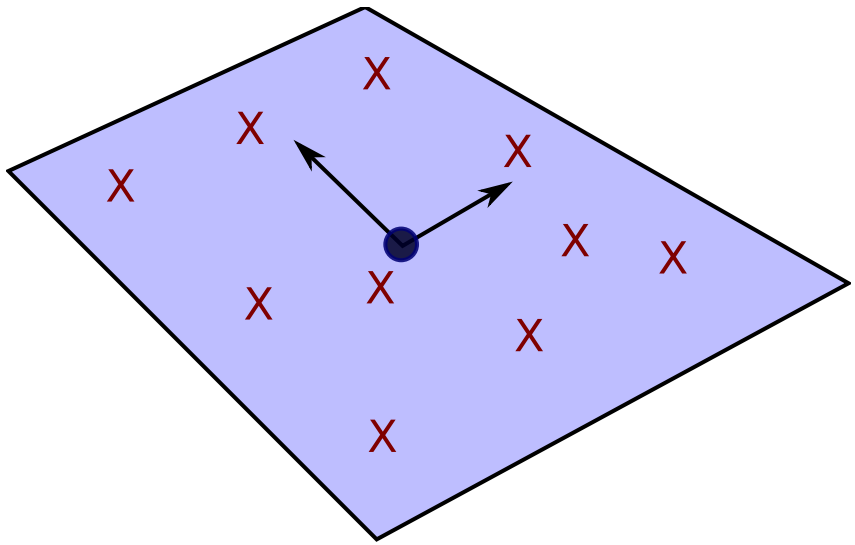
One way of doing this is to find the mean of the data, and the first principal component (the direction of maximum variance). We can then specify how far along the line each point is, giving its approximate position.



If we use another direction at right angles, then we can specify exactly where each point is.

Principal components analysis

From three dimensional data which lies roughly on a 2D plane, we can use two dimensions to get a more compact representation:



Notice that we have effectively replaced the axes with something more “natural” to the data.

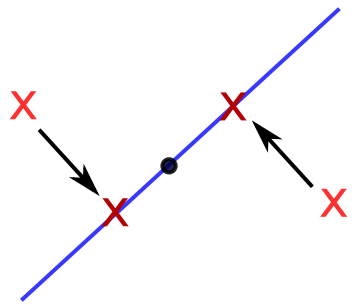
Calculating principal components

1. Calculate the mean of the data
   [`mean()` in numpy]
2. Calculate the covariance of the data
   [`C=cov()` in numpy]
3. Calculate the eigenvalues and eigenvectors of the data
   [`eig(C)` in numpy]
4. Keep the first few eigenvectors, and throw the rest away.

How many principal components should you keep, and how many should you throw away?

Calculating low-D representation

We need to project each point onto the different "axes" (principal components).



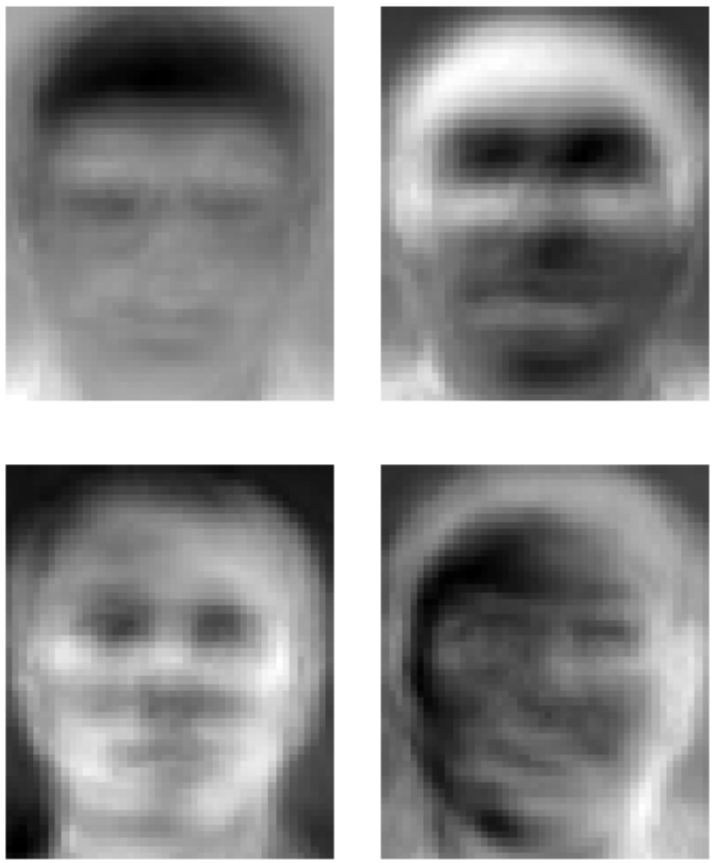
If e_1 is the first eigenvector, x is the datapoint and m is the mean, then the projection is given by:

$$a = e_1^\top (x - m)$$

i.e., just the dot product of two vectors.

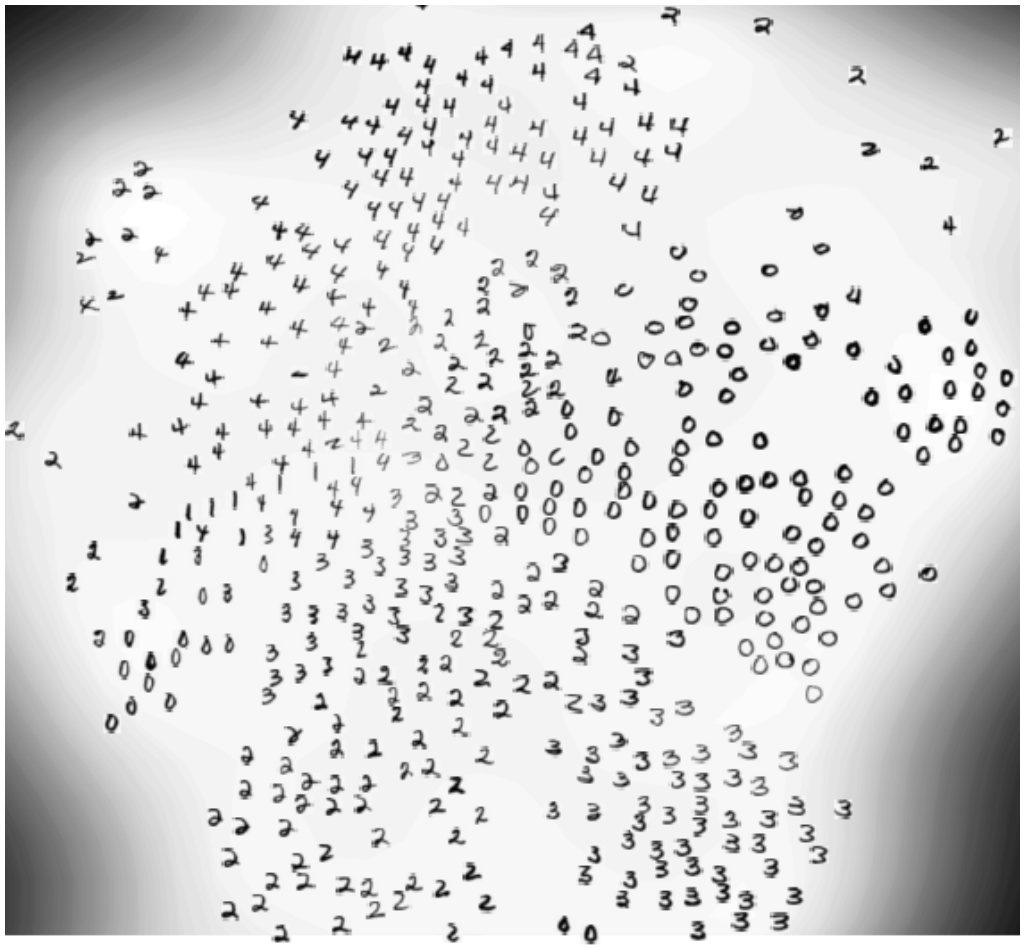
Low-D representation of faces

We can use this technique to reduce the dimensionality of face images. The principal components/eigenvectors are sometimes called “eigenfaces”.



Low-D representation of handwritten digits

Example of non-linear dimensionality reduction with a more sophisticated method:



GP-LVM, Lawrence (NIPS 16)

Unsupervised learning

Unsupervised learning generally tries to find some structure in unlabelled data. Based on some distance measure, for example, we may try to group the data points we have into sets (**clustering**).

This may be useful in order to gain insight into the data; even better, it can serve as a way of finding a good data representation for other tasks. It may be argued that unsupervised learning is only useful as part of some other process. (At least, without an objective of some sort, there is no way to evaluate if one solution is any better than another).

K-means clustering

1. **Intialisation**: set the means $\mu_1, \ldots, \mu_K$ to have the same position as K randomly selected data points.
2. **Assignment**: for each data point, work out which is the nearest mean.

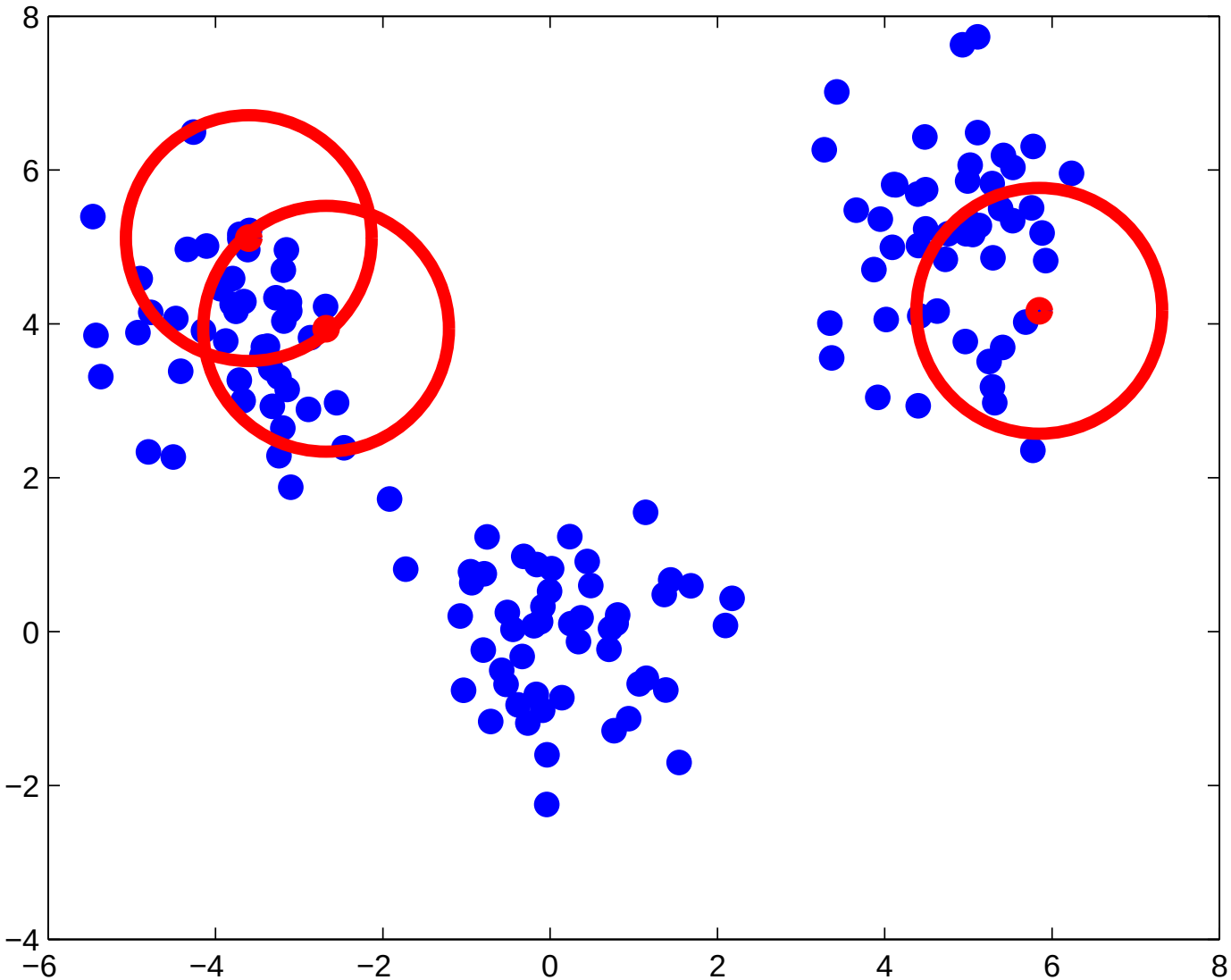
$$k_i = \arg\min_j d(\mu_j, x_i)$$

3. **Update**: re-calculate the K means to be the average of all the data points that "belong" to it.

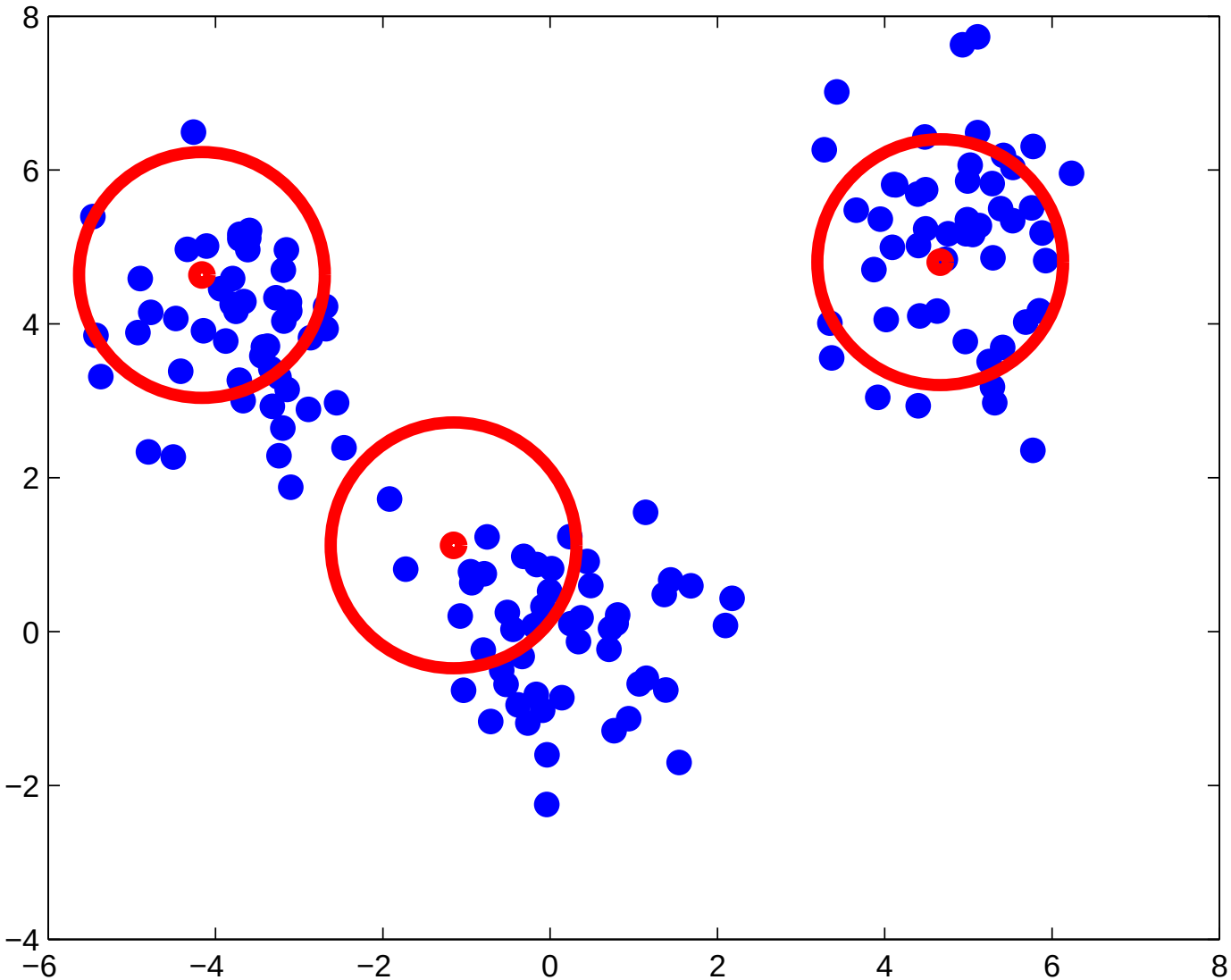
$$\mu_i = \frac{1}{N_i} \sum_{j:k_j=i} x_j$$

where N_i is the number of data points that belong to set i. If the means have converged (no change), then stop. Otherwise go to step 2.

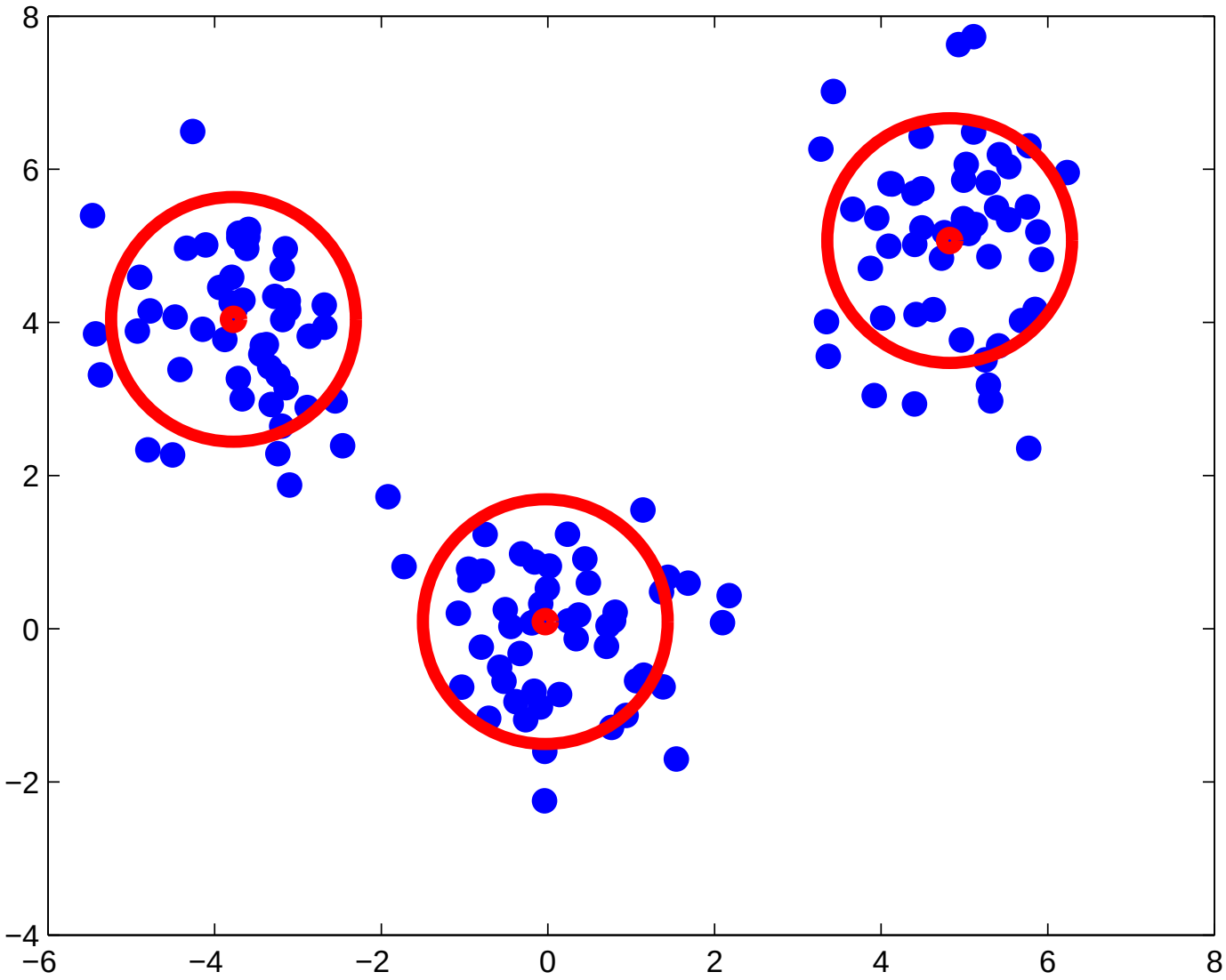
K-means clustering



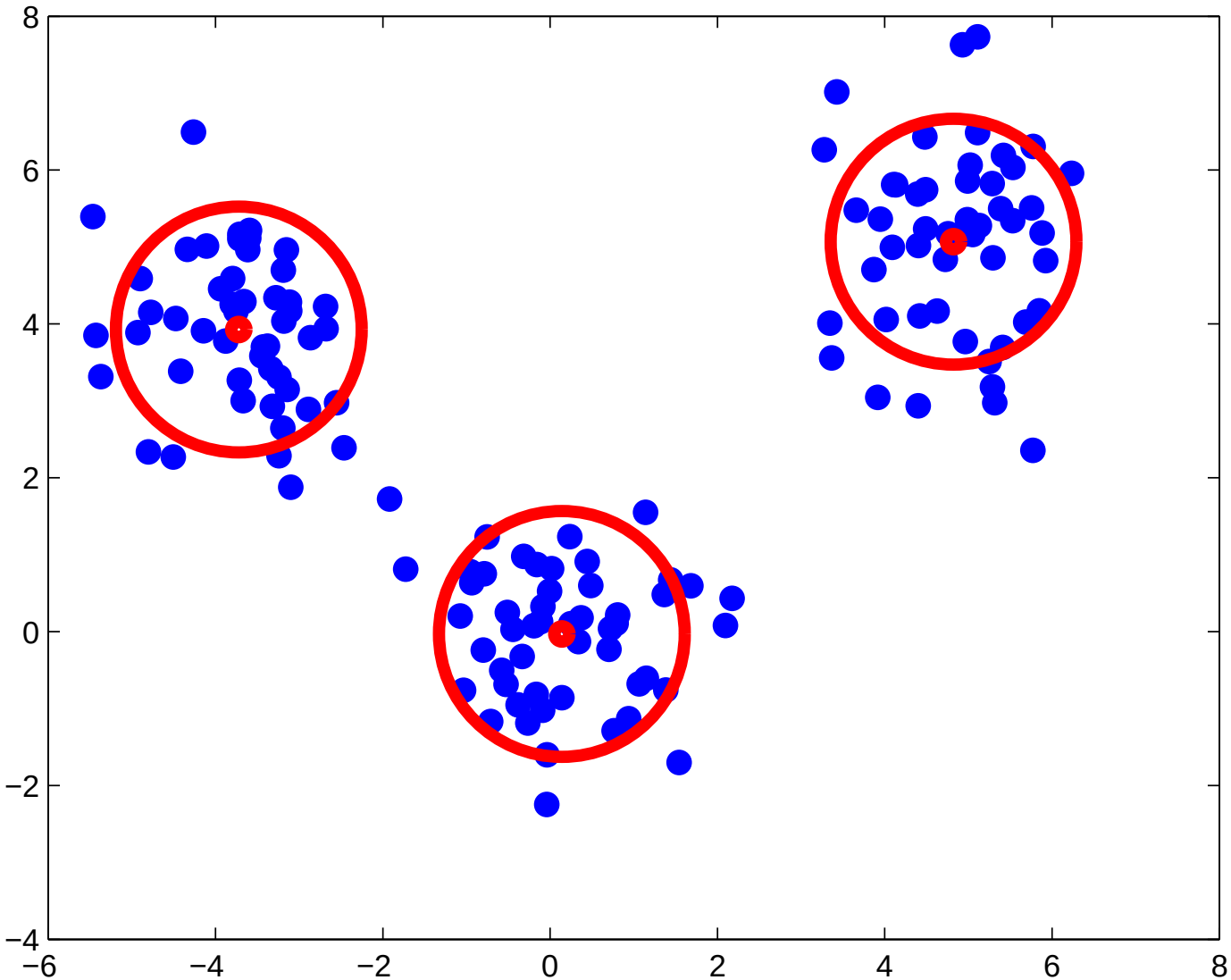
K-means clustering



K-means clustering

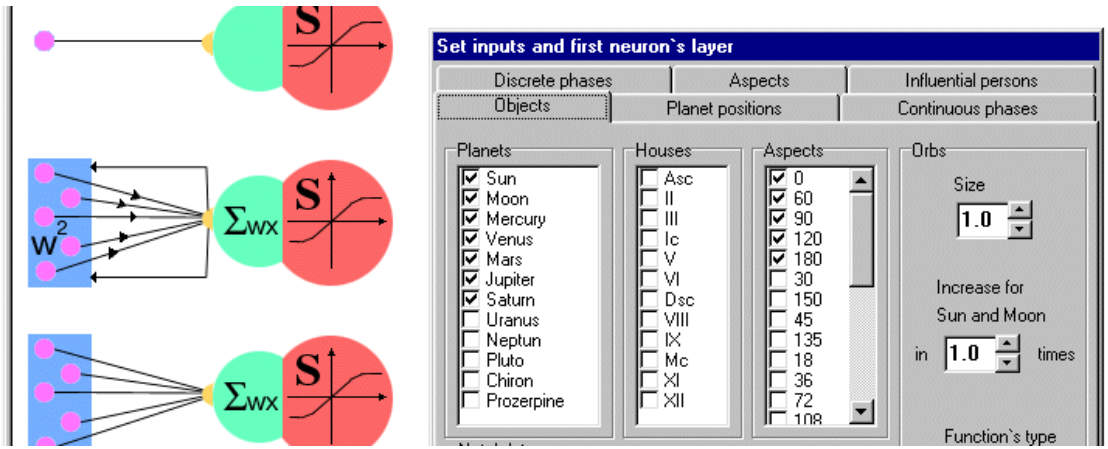


K-means clustering



When to believe a model?

Financial Astrology: developers claim near perfect prediction of past share prices based on planetary positions. Will you buy it?

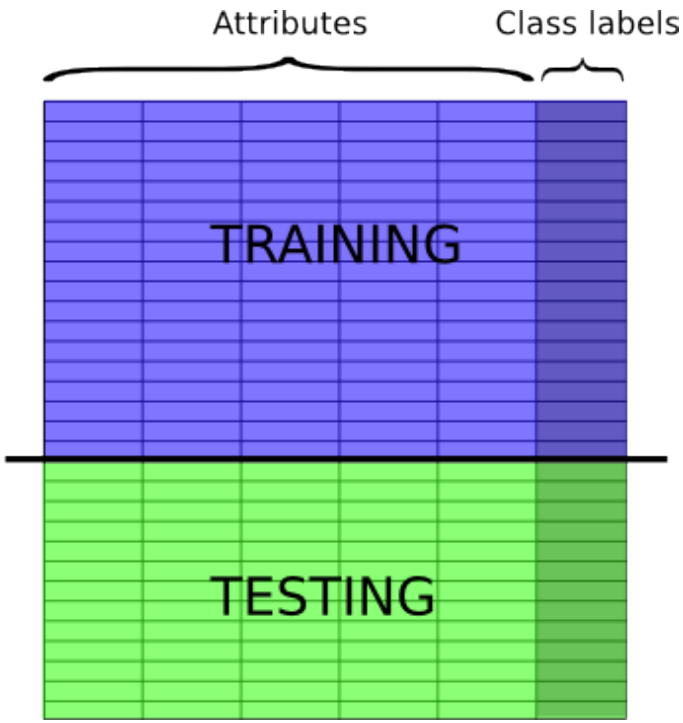


Reasons to believe a model is right:

- It successfully predicts the future
- It provides a compact description of the past
- Model-specific diagnostics suggest it to be plausible
- **Not** because it has high likelihood on training data

Testing and cross validation

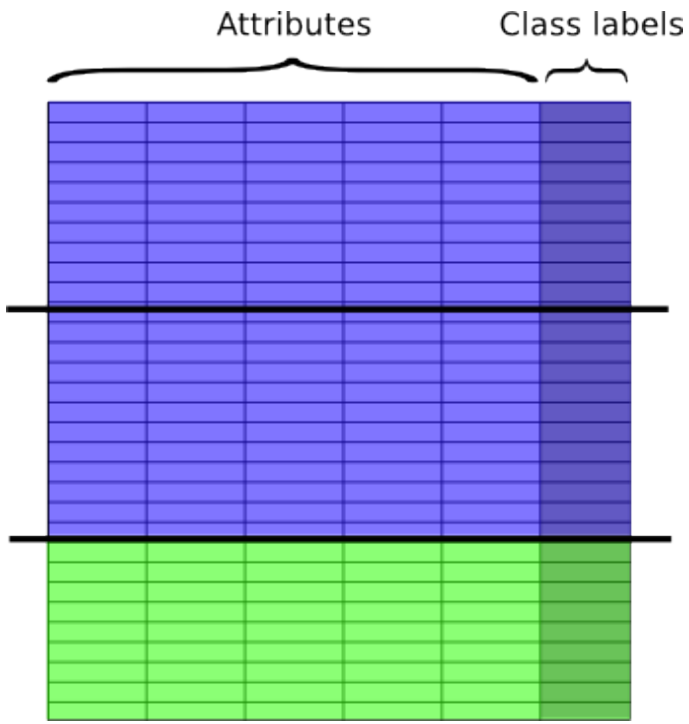
The simpest way to evaluate a learning system is to split the data up into training and test sets:



Simple measures of performance: Accuracy is the proportion of test cases correctly classified, $\frac{TP+TN}{TP+TN+FP+FN}$. Error rate is 1-accuracy.

Testing and cross validation

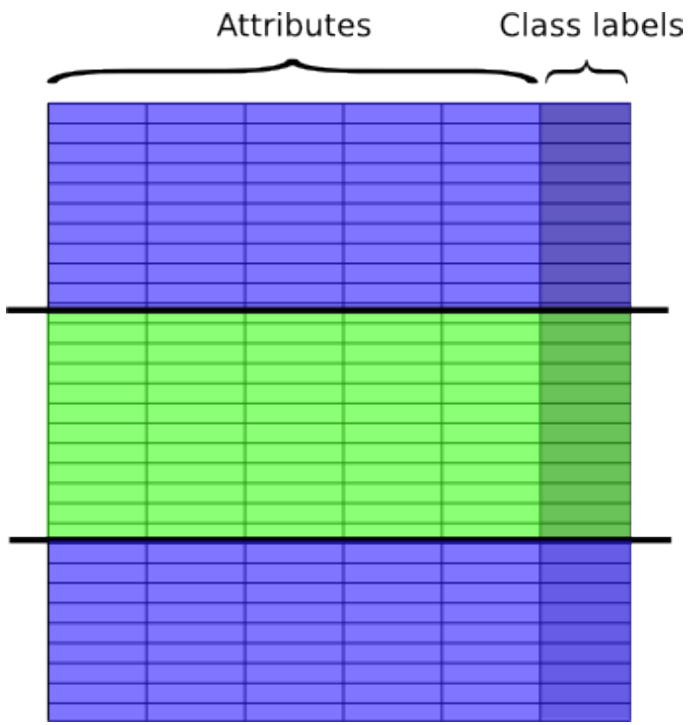
When labelled data is limited (nearly always!), cross validation is useful. The original dataset is partitioned into K sets. For each test set we we train with the remaining $K - 1$. 3-fold cross validation:



Can then find average performance and calculate error bars.

Testing and cross validation

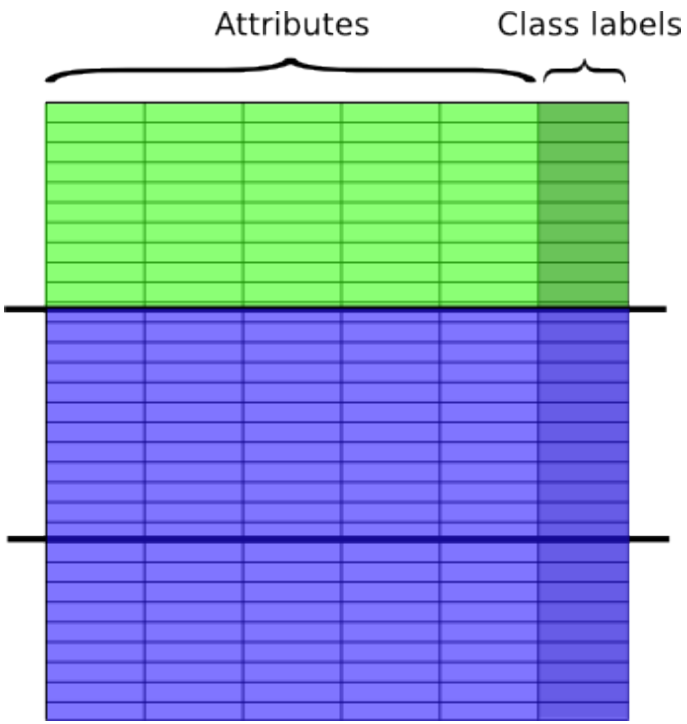
When labelled data is limited (nearly always!), cross validation is useful. The original dataset is partitioned into K sets. For each test set we we train with the remaining $K - 1$. 3-fold cross validation:



Can then find average performance and calculate error bars.

Testing and cross validation

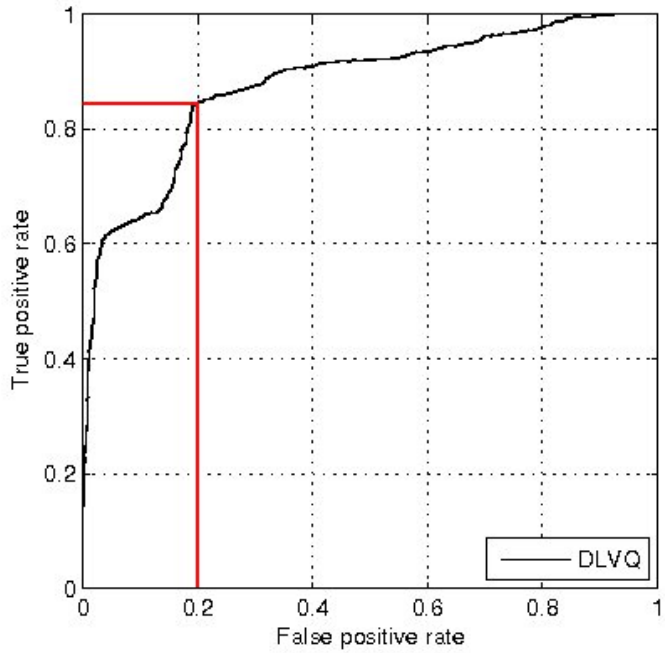
When labelled data is limited (nearly always!), cross validation is useful. The original dataset is partitioned into K sets. For each test set we we train with the remaining $K - 1$. 3-fold cross validation:



Can then find average performance and calculate error bars.

Receiver operating characteristics

For classifiers which give a probability output, we can now read off the tradeoff between false positives and false negatives.



This curve shows that at 20% false alarm rate, we catch about 85% of the true positives. Imagine this was your car alarm – you could decide the tradeoff between sensitivity and specificity.

Generalisation

If we look at the performance of a system on its training data and on 'holdout' evaluation data, we usually see that as the model becomes more complex, there is a point at which increasing complexity makes the performance on the evaluation set worse. In these cases, we say that the model has decreasing generalisation performance.

"Learning starts where memory ends."

Rule of thumb: think about how many 'answers' your system can 'memorise'. If it is comparable to the size of the training set, generalisation performance might not be very good.

Combining learning systems

- Ensemble learning: bagging/boosting
- Cascades of learners
- Mixtures of experts

Other types of learning

- Reinforcement learning
- Transfer learning
- Active learning

General tips for practical machine learning work

- Visualise the data.
- Try to incorporate background knowledge.
- Good features beat fancy models.
- Approximate median proportion of time spent on the different stages of ML projects[1]:
  - data collection: 20%
  - data preparation: 30%
  - changing representation of data: 10%
  - learning model from data: 10%
  - performance evaluation: 10%
- Many interesting research areas in Artificial Intelligence for Development (`http://ai-d.org`)

[1] `http://www.cs.cornell.edu/~mmunson/model-step-survey-results.html`