

Department of Biochemistry & Biophysics – Structural Biochemistry and Bioinformatics Course

Introduction to Machine Learning Techniques for Bioinformatics

Bob MacCallum

`maccallr@sbc.su.se`

Outline for today's lecture

What's it all about?

Classical/statistical methods

Nearest neighbour methods

Neural networks

Support vector machines

Genetic algorithms

Genetic programming

Why do we need machine learning?

In **classification**, **prediction** and **pattern recognition** tasks.

- What will be the value of Ericsson shares tomorrow?
- What is a high-risk insurance/loan customer?
- Is a tumour aggressive?
- Is a person employed at this company?
- Who is that doing 230 km/h on the motorway?

Some overlap between these three tasks.

Why do we need machine learning?

Imagine the following:

Stockholm city planners need to know how many **trucks, buses** and **cars** use certain roads/bridges/tunnels etc.

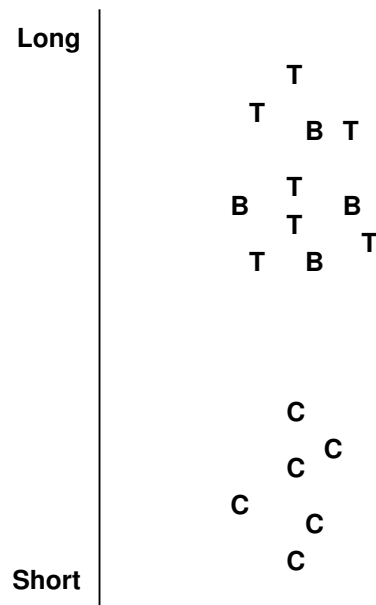
It is not practical to pay people to do this job.

Instead, using cameras and road sensors they will collect data on **length, weight** and **number of wheels**.

How can we count the different vehicle types using this information?

Classification from one measurement

We start by considering vehicle length.

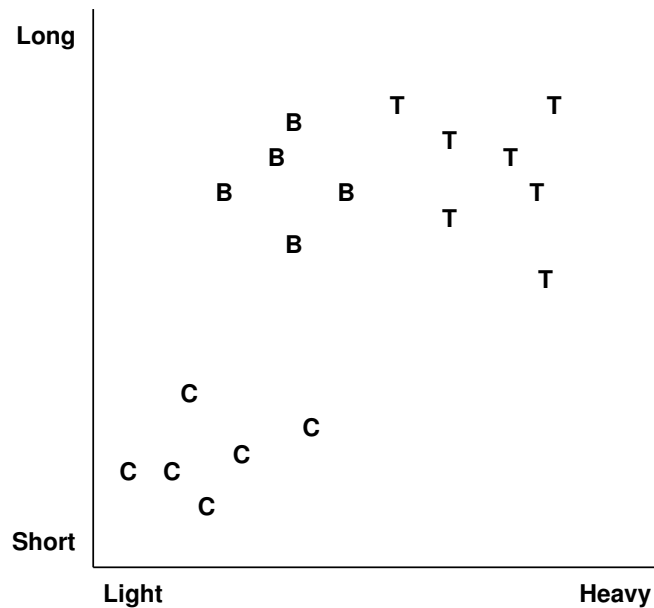


In an ideal world this is sufficient to separate cars from trucks and buses (we ignore limousines for now).

Can use a simple threshold or probability distributions.

Classification from two measurements

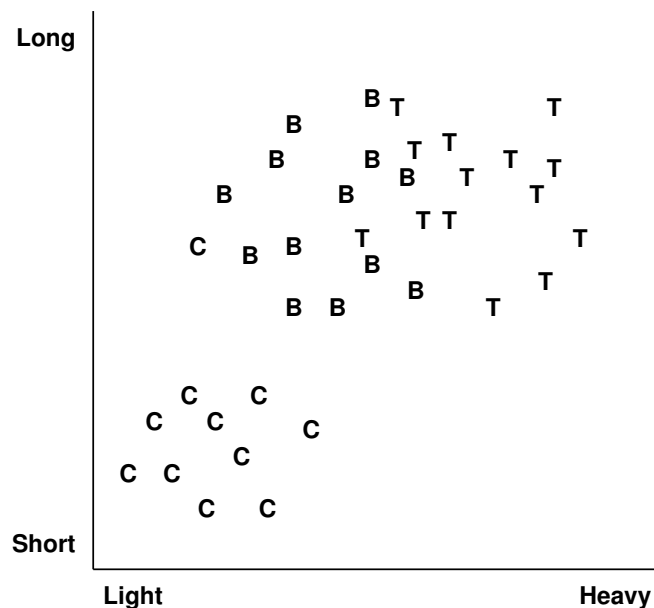
Now we have vehicle length and weight



Two thresholds will separate all three groups, but...

Classification from two measurements

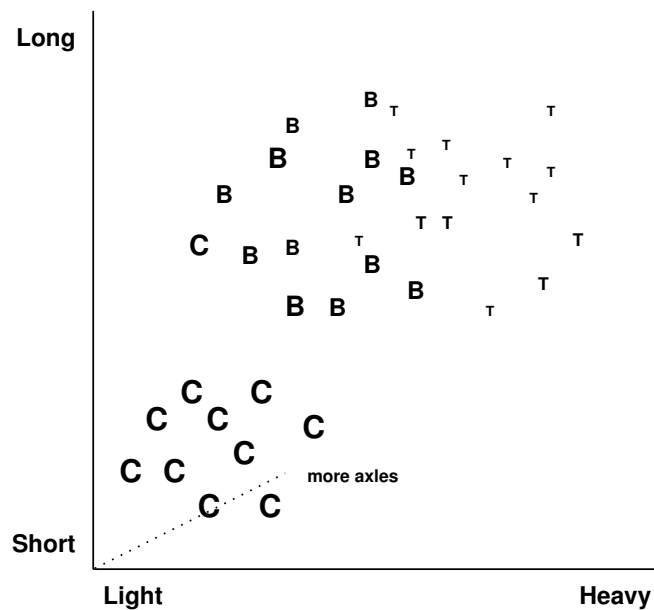
... when take more measurements the borders get blurry



so maybe it will help to take another measurement: the number of axles (wheels) which pass over the sensors...

Classification from 3+ measurements

With 3 dimensions we need to find a plane or surface which separates the classes



Classification problems often use more than 3 measurements!

Key concept: supervised learning

There are two phases in the development of classifiers:

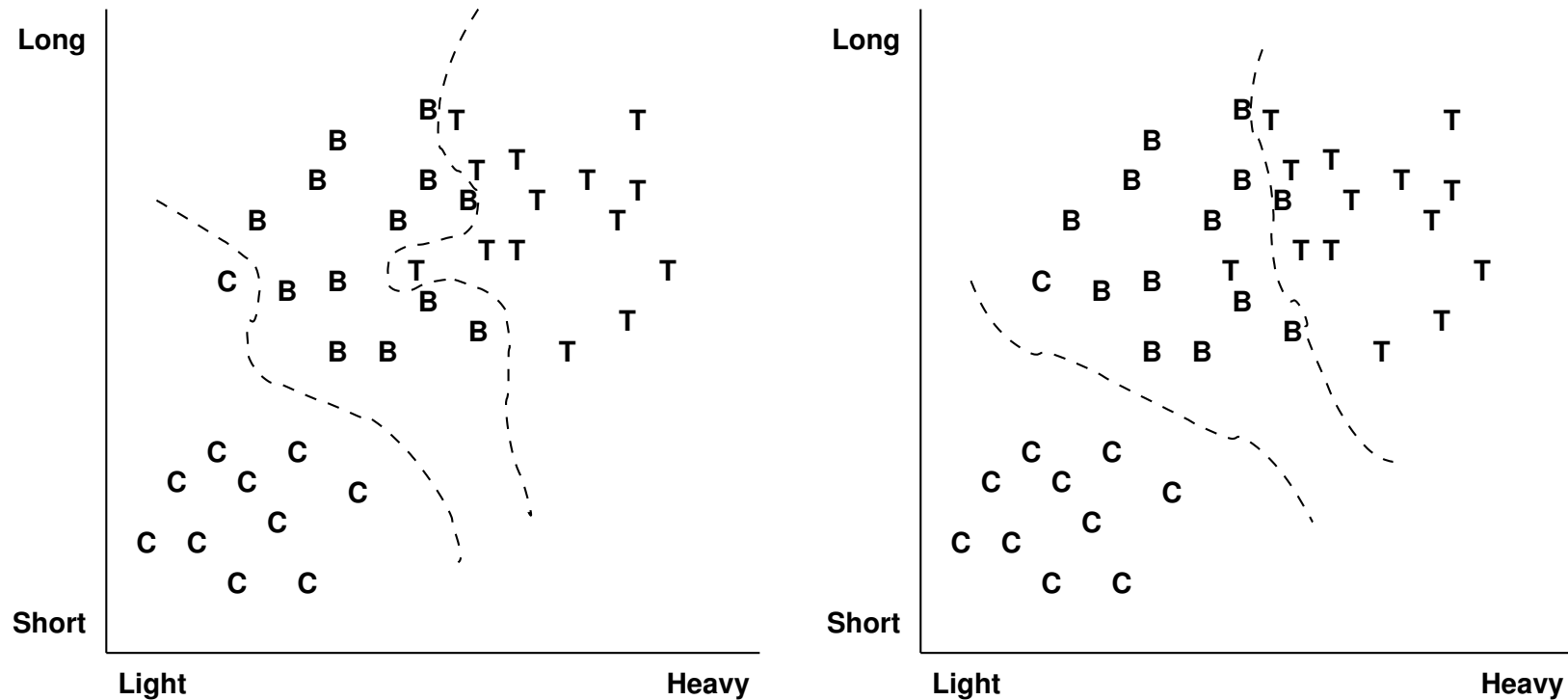
training - collect data from vehicles of *known* types (i.e. with a person recording them); then decide thresholds or separation surfaces

testing - apply classifier to data that was not used for training, to see how well you do (see next slide)

A third phase is “deployment” of course

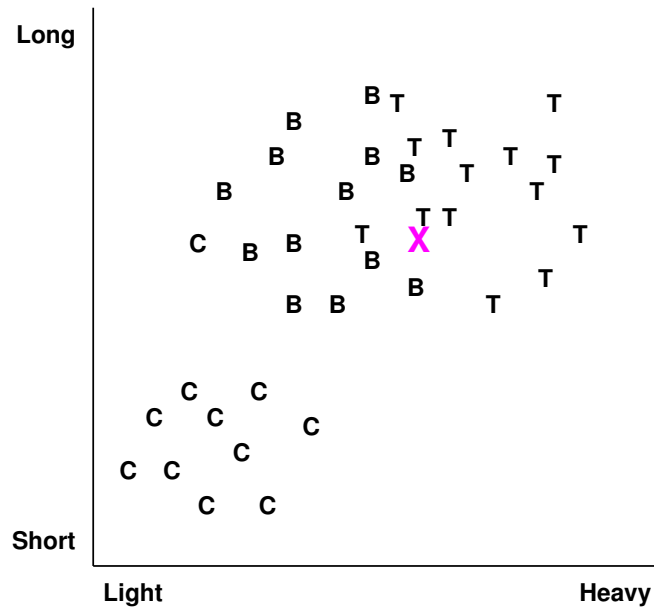
The testing data should not have a relationship with the training data (as we will see later)

Key concept: overfitting



On unseen data, the classifier on the left may actually perform worse – also known as **overtraining** – if you avoid it your predictor/classifier can **generalise**

Nearest neighbour methods

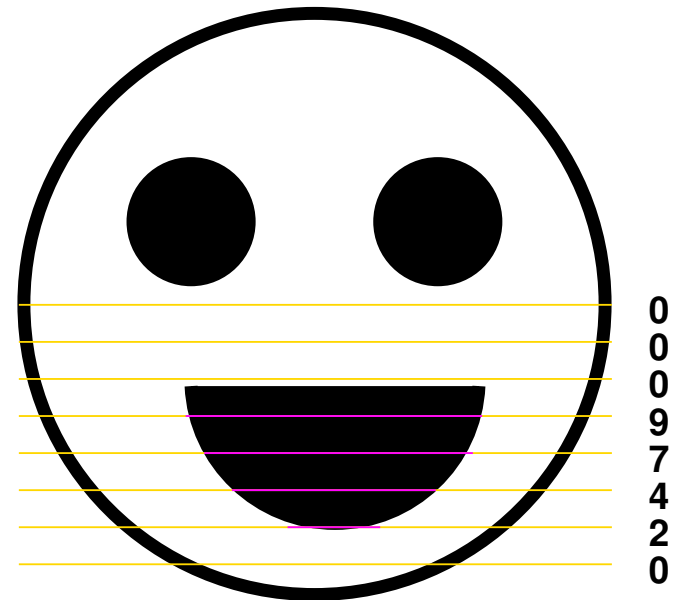
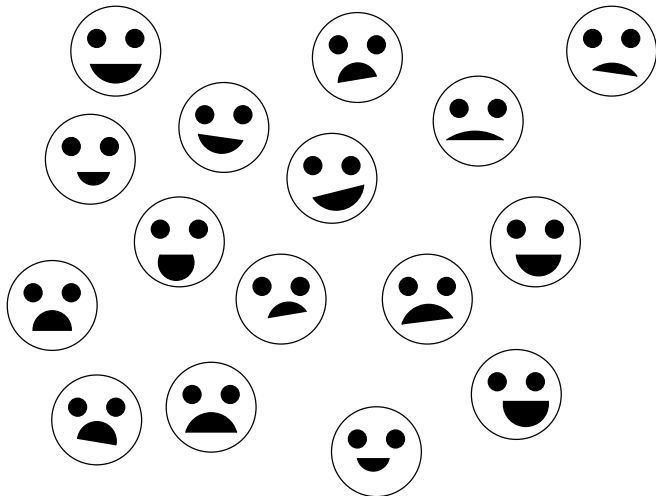


Given some existing classified data, and a new point “X”, you can classify it according to the class of the nearest point, or the majority of the nearest k neighbours.

It's as simple as that!

Key concept: feature extraction

In the vehicle example we had easily measurable quantities. The faces below are easy for us to classify, but are difficult for machines. They usually require a pre-processing stage called feature extraction, where **domain-specific knowledge** is used.



Neural Computation

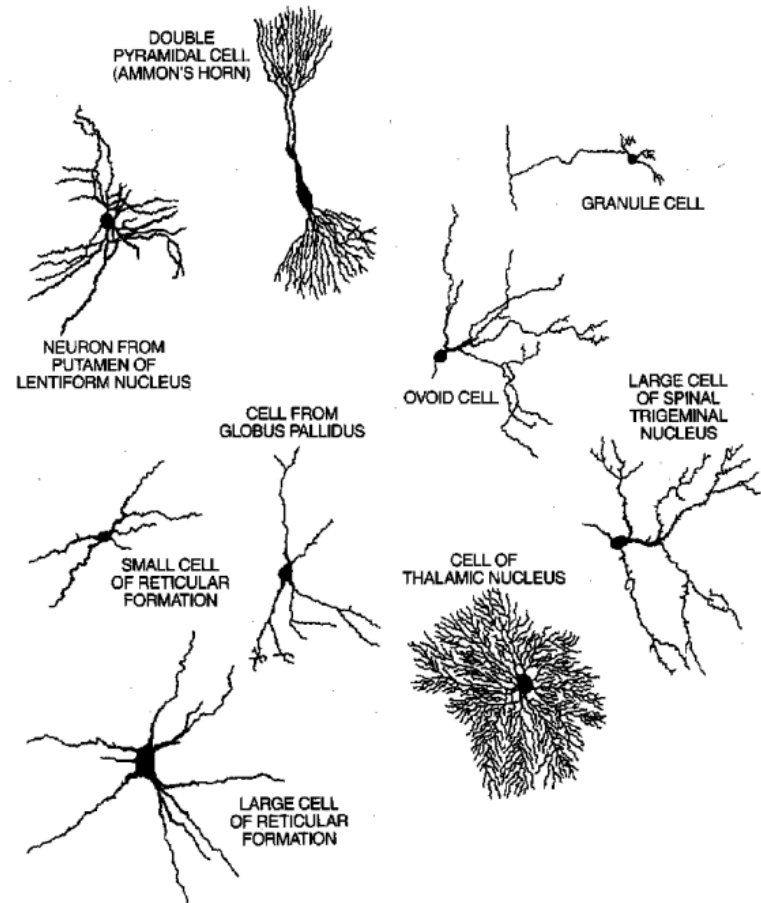
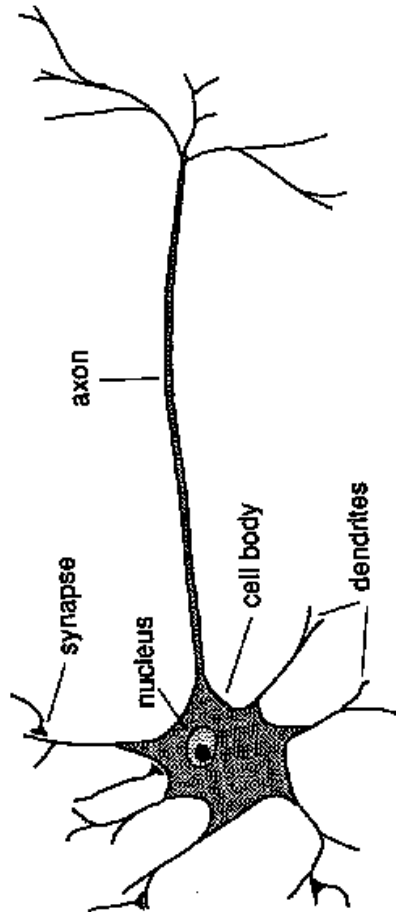
Nervous systems operate on a number of levels:

- stimulus → response
- regulation (e.g. heart rate/breathing by brain stem)
- coordination, balance & movement
- sight and sound
- language and thought (on a good day!)

Cope well with **noisy data** and **missing information**.

Biological Neural Networks

10-100 billion neurons in human brain, up to 10,000 connections per neuron

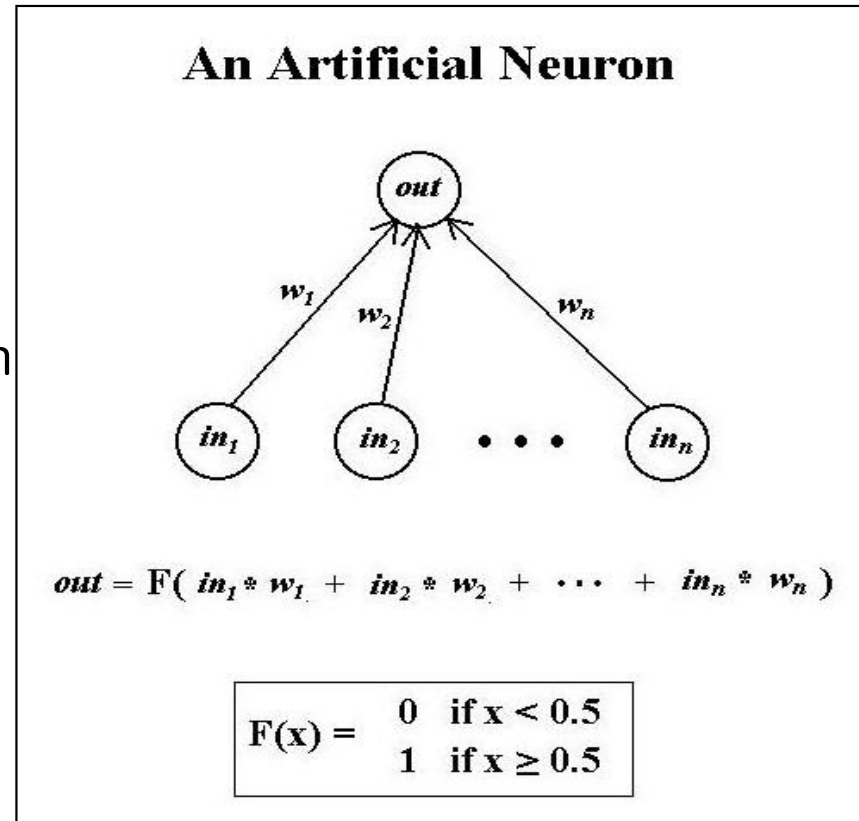


Artificial Neural Networks

Simplified model of a neuron

Weighted sum of inputs

Activation above hard or soft threshold



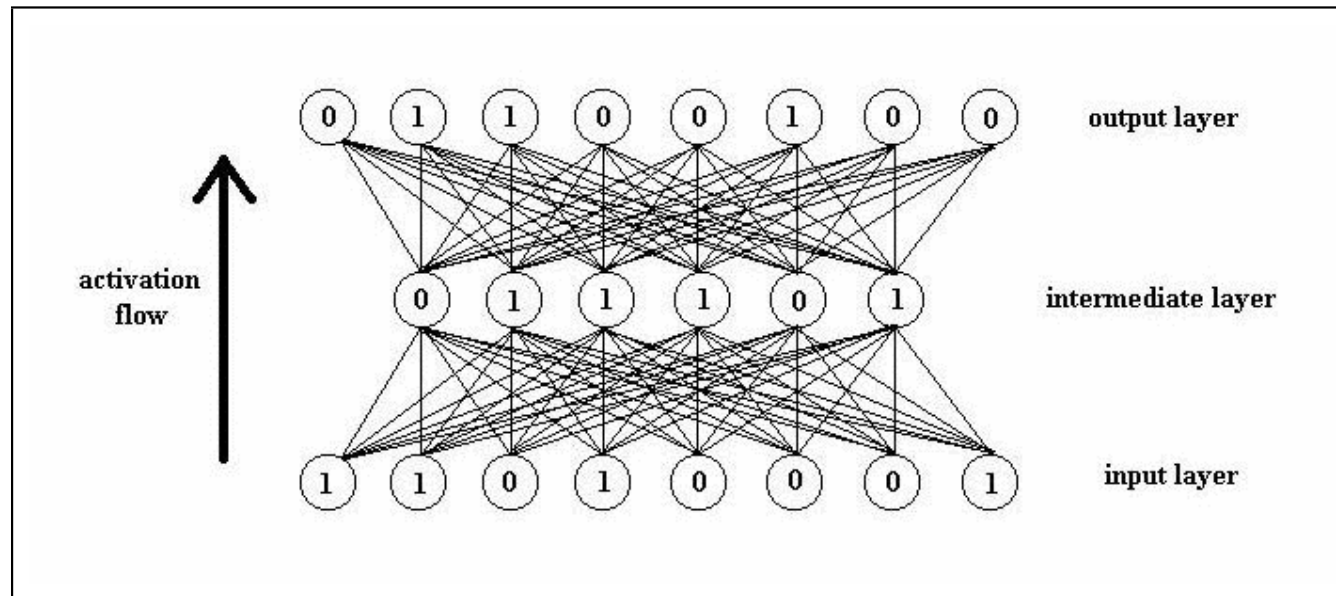
Artificial Neural Networks

Usually but not always:

Relatively simple, regular networks

Layers

Uni-directional flow



Training neural networks

We want the combination of weights that gives the desired output(s) for each set of inputs

In the face example we have 8 inputs for each face. The desired output could be 1 for happy and 0 for sad. You have to choose how many neurons/nodes are in the hidden layer.

How could we find the weights?

- random search
- hill-climbing
- genetic algorithm
- **back-propagation** is most efficient

Back-propagation NN training

1. Present a training sample to the neural network
2. Compare the NN's output to the required output from that sample. Calculate the error in each output neuron.
3. Using the weights on its incoming connections, transfer “blame” to the neurons in the previous layer. Repeat transfer back to input layer.
4. Adjust each weight by a small amount in the direction which minimises the error by an amount proportional to the “blame” assigned to that neuron.
5. Repeat with all other training samples for a number of *epochs*.

Uses of NNs

Very widely used in bioinformatics and engineering

Particularly classification and pattern recognition

Handles large amounts of complex data, including non-linear relationships

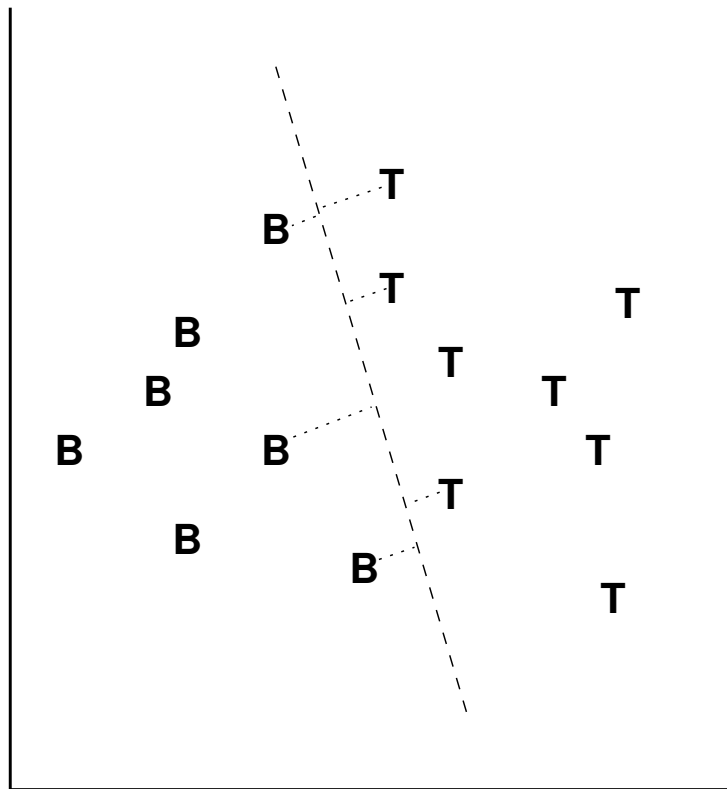
Junk in, junk out...

Reduce number of nodes and training time to prevent overtraining

Black box or analytical tool

Support vector machines

A recently discovered classification tool which performs very well and is difficult to overtrain.



SVMs try to put a plane (hyperplane) through your data to separate it into two classes

To maximise generalisation, the plane is placed *as far as possible* from neighbouring data points (so-called support vectors)

If the data is non-linear and cannot be separated by a plane, then various *kernel functions* can be used to transform the data first

Evolutionary Computation

Evolve solutions to computational problems

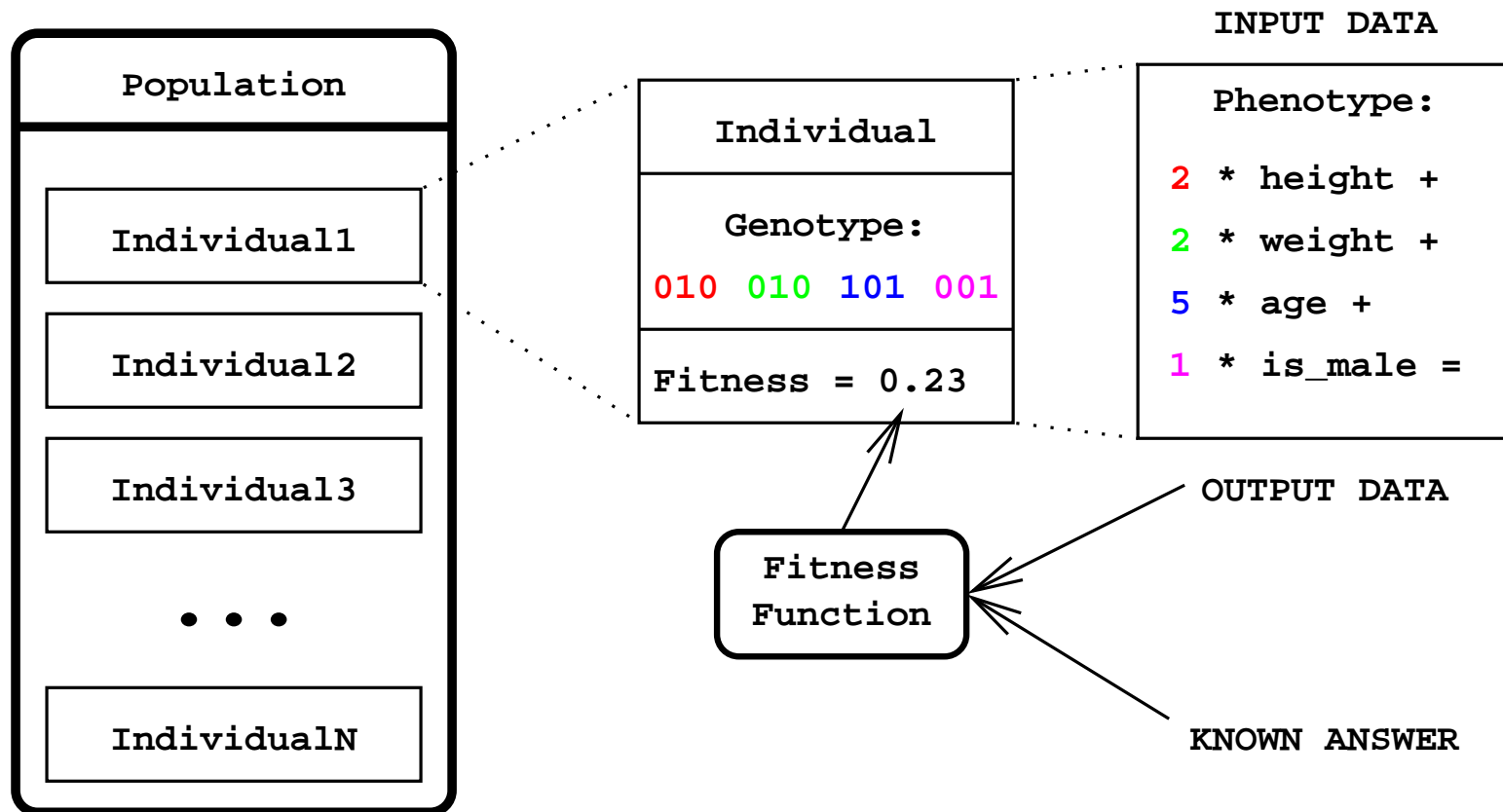
Population of individuals which undergo:

- selection
- reproduction
- recombination and mutation of “genetic material”

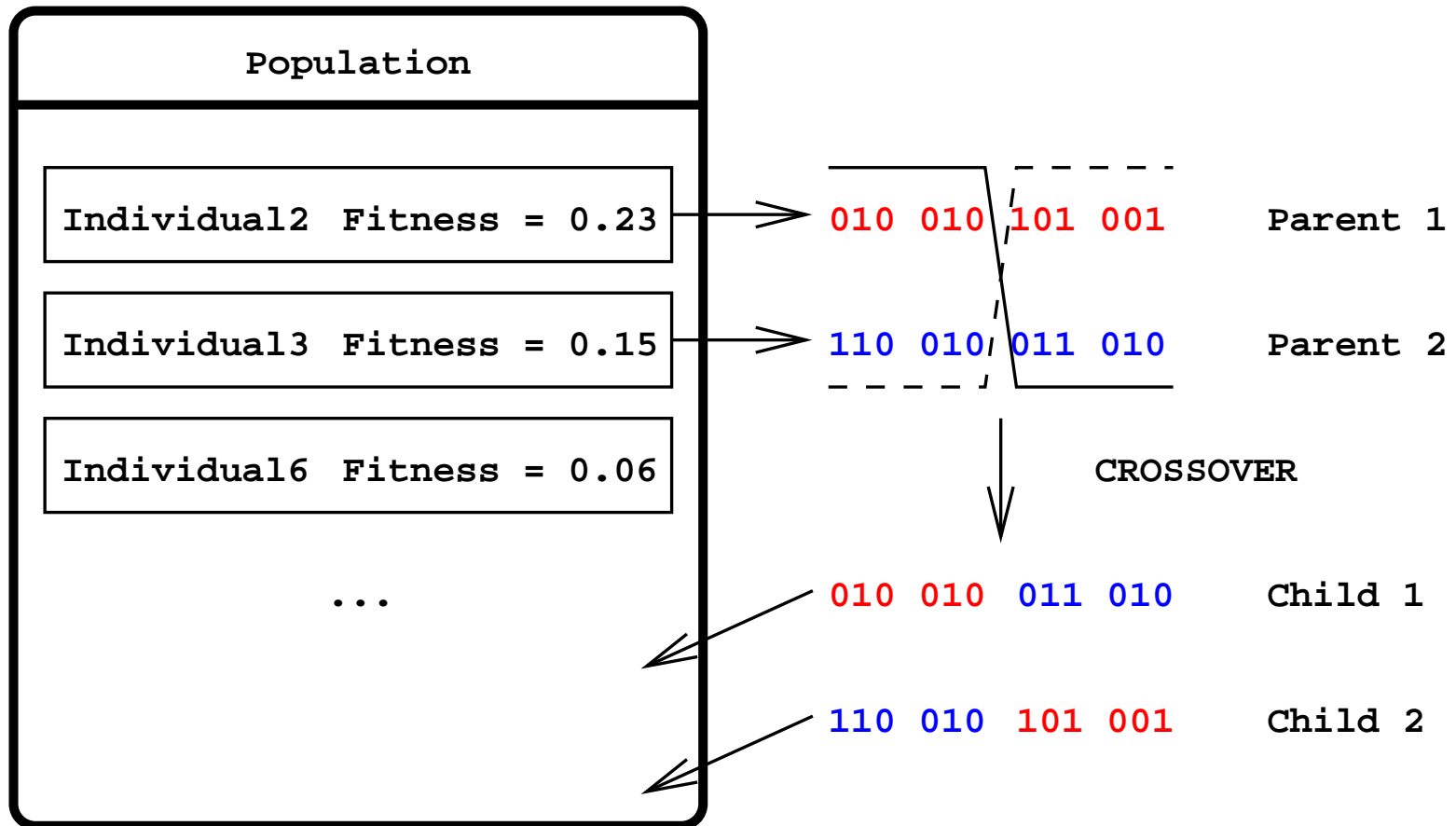
The idea is that you get **exchange of subcomponents**

But it's very crude compared to the real thing

Genetic Algorithms



Genetic Algorithms



Other Evolutionary Computation Approaches

Hybrid algorithms:

- efficient local search
- neural network design

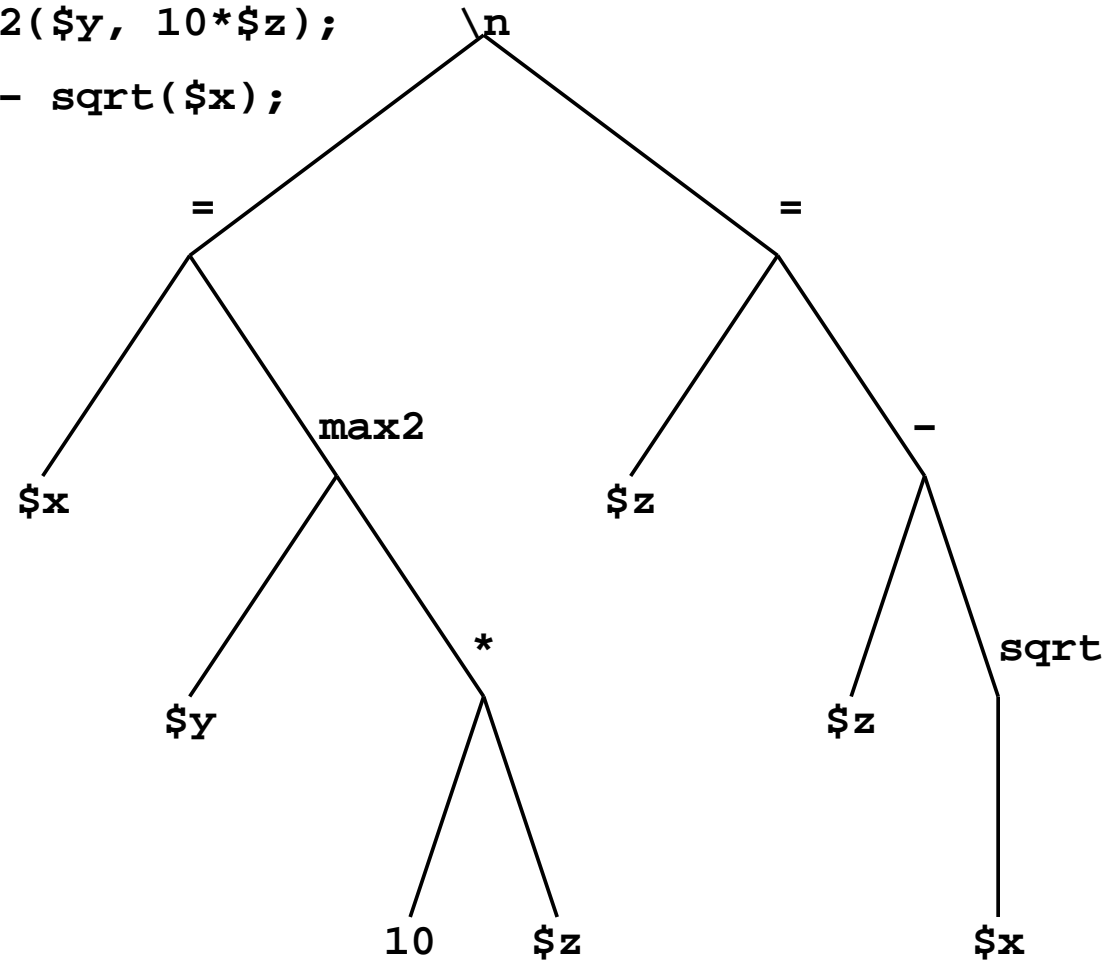
Genetic Programming:

- evolve computer programs
- non-linear tree-like “chromosomes”
- solution can be big or small

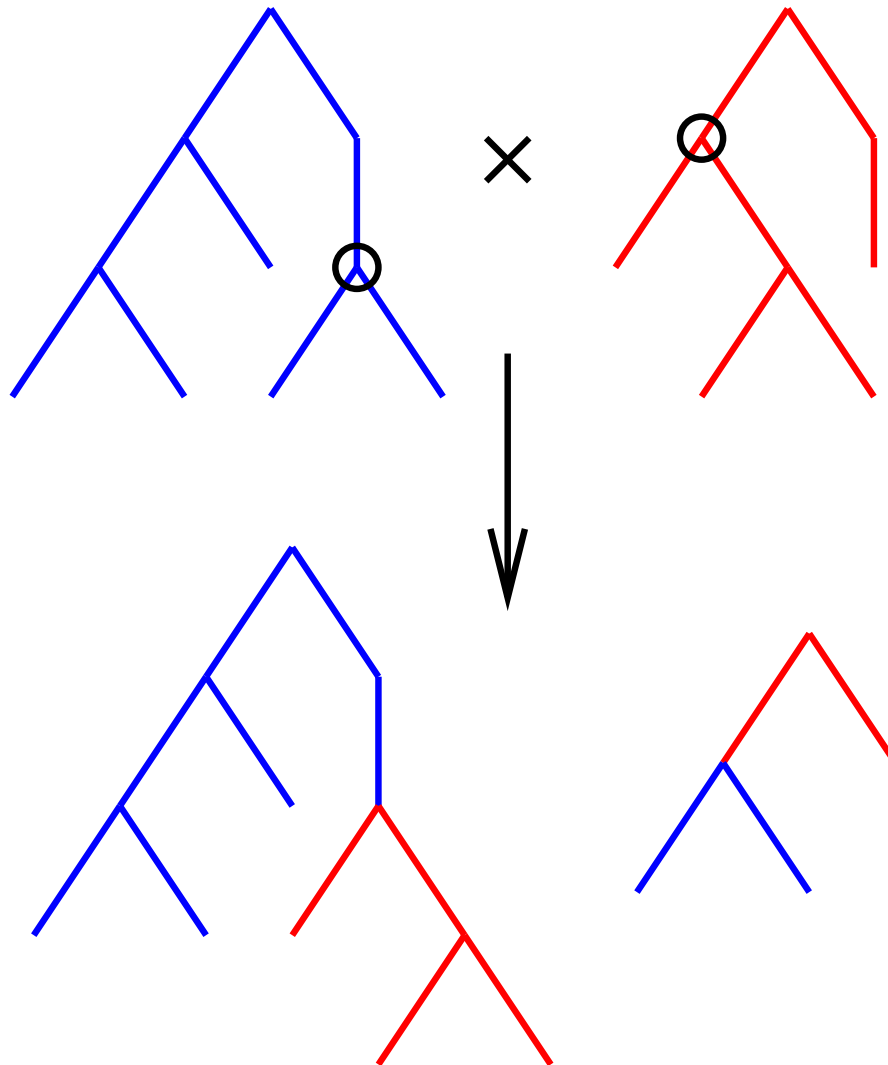
Programs are Trees

```
$x = max2($y, 10*$z);
```

```
$z = $z - sqrt($x);
```



Crossover in GP



can use multiple
crossover points

can do “homologous”
crossover

type aware

Some evolved code from GP

From my work classifying proteins from amino acid sequence into two classes: nuclear and non-nuclear

```
sub nuclear_score {
  my $seq = shift;
  return (((match($seq, 'KR[MRW]') +
            (match($seq, '[NQRT][^T]+[GHKMQRTV]RK') *
             (match($seq, '([S]){1,}') - match($seq, '[V]+'))
           )) * match($seq, '[^AKI][^KGL]KK')) *
          match($seq, '[^GKI][^KGL]KK'));
}

print "$seq is nuclear\n" if (nuclear_score($seq) > 0);
```

Bob's next lecture

We look at a successful application of machine learning in bioinformatics: **secondary structure prediction**