Foundational Machine Learning 1.02: Decision Trees

Rohit Babbar
rb2608@bath.ac.uk

9th October 2025

UNIVERSITY OF BATH

This lecture

- Decision tree – supervised classification
- A good starter algorithm – easy to understand
- Next week is random forests, which builds on decision trees
  (random forests were the best approach before deep learning)

UNIVERSITY OF BATH

What is the goal? I

- Learn $y = f(x)$ from data
  - $x \in \mathrm{R}^n$ is a n dimensional feature vector
  - $y \in \{0, \ldots, K-1\}$ is the output class when there are K possible output classes.

UNIVERSITY OF BATH

What is the goal? I

- Learn $y = f(x)$ from data
  - $x \in \mathrm{R}^n$ is a n dimensional feature vector
  - $y \in \{0, \ldots, K-1\}$ is the output class when there are K possible output classes.
- We can't consider every possible $f(x)$
  - there are infinitely many that fit any data set!
- Instead $f_\theta(x)$ will belong to a space of functions parametrised by θ

What is the goal? II

- Learn $y = f_\theta(x)$ from data, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- But how do we know which θ is best?

UNIVERSITY OF BATH

What is the goal? II

- Learn $y = f_\theta(x)$ from data, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- But how do we know which θ is best?

- A loss function: $L(y_i, f_\theta(x_i))$
- Total loss: $\sum_{i=1}^{N} L(y_i, f_\theta(x_i))$

UNIVERSITY OF BATH

What is the goal? II

- Learn $y = f_\theta(x)$ from data, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- But how do we know which θ is best?

- A loss function: $L(y_i, f_\theta(x_i))$
- Total loss: $\sum_{i=1}^{N} L(y_i, f_\theta(x_i))$

- Goal: Find θ that minimises $\sum_{i=1}^{N} L(y_i, f_\theta(x_i))$

UNIVERSITY OF BATH

Decision stump I

- Algorithm from last lecture!

```
# Parameters...
feature = 'teeth'
match = False

# Function...
def evaluate(fv):
  return fv[feature] == match

# Fit to data...
best = 0.0
for f in features:
  for m in [False, True]:
    accuracy = performance(f, m, train)
    if accuracy > best:
      feature = f
      match = m
```

UNIVERSITY OF BATH

Decision stump II

- Converted to current notation:
- Parameters:

$$\theta = \{\texttt{feature}, \texttt{match}\}$$

- Function: (δ = Kronecker delta function)

$$f_{\theta(x)} = \delta(x_{\texttt{feature}}, \texttt{match})$$

- Loss function:

$$L(y_i, f_\theta(x_i)) = \begin{cases} 0 & \texttt{if } y_i = f_\theta(x_i) \\ 1 & \texttt{otherwise} \end{cases}$$

(this is called **0–1 loss**)

UNIVERSITY OF BATH

Decision stump III

- What is the space of functions it learns?

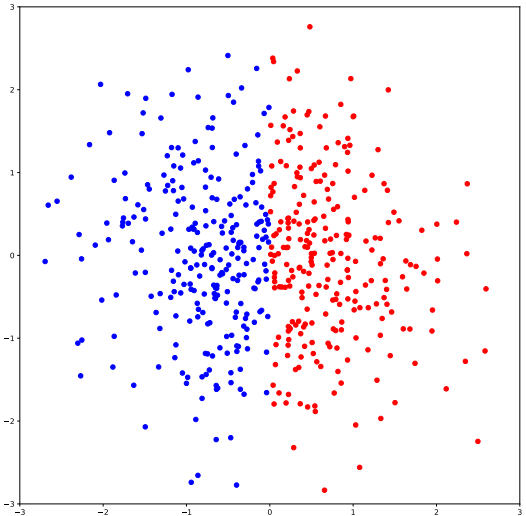
UNIVERSITY OF BATH

Decision stump III

- What is the space of functions it learns?
- Identify function or it's inverse (not)
  – selects input feature most similar to output!
- This is not very useful...

Continuous decision stump I

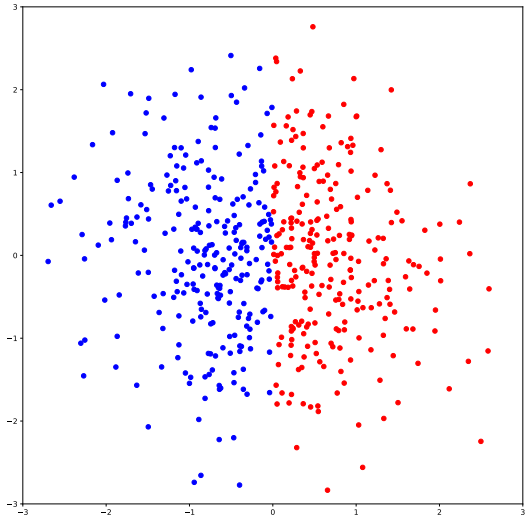
- What if the input was continuous?
  (colours denote classes)



Continuous decision stump I

- What if the input was continuous?
  (colours denote classes)
- We could instead **split** (partition) the space:

$$f_\theta(x) = \begin{cases} 0 & \text{if } x_{\texttt{feature}} < \texttt{split} \\ 1 & x_{\texttt{feature}} \geq \texttt{split} \end{cases}$$

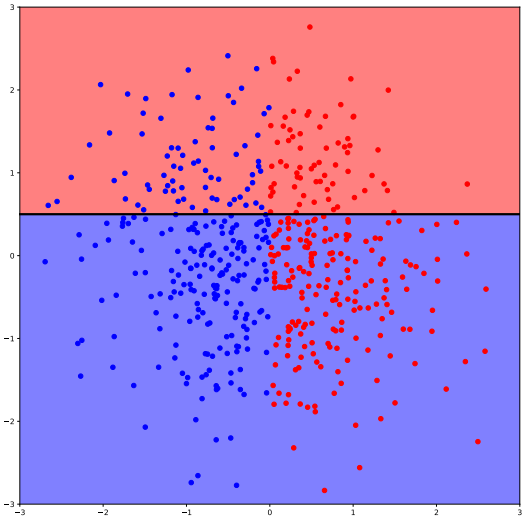


Continuous decision stump II

- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.5

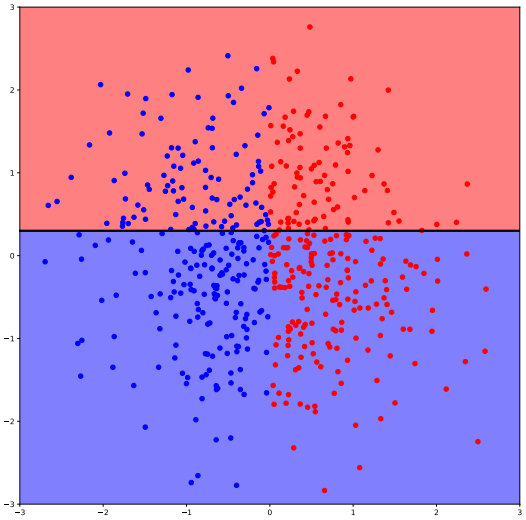
  accuracy = 50.8%

  (accuracy = one minus average 0–1 loss)



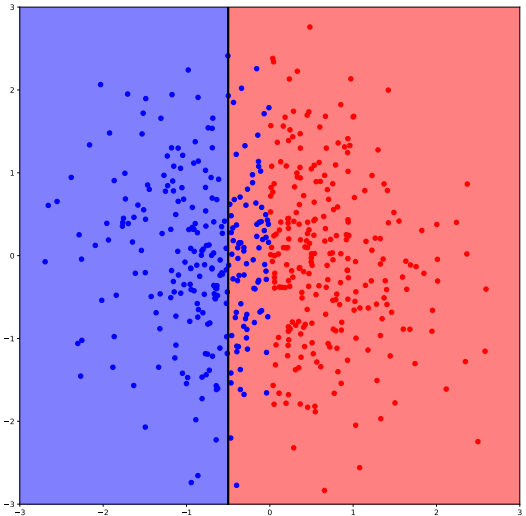
Continuous decision stump II

- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.5
  accuracy = 50.8%
  (accuracy = one minus average 0–1 loss)
- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.3
  accuracy = 52.1%



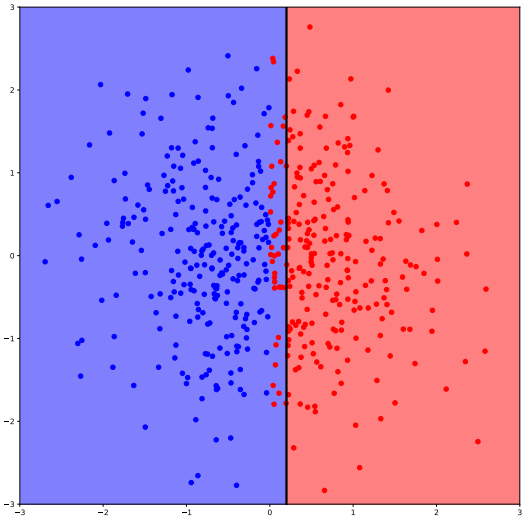
Continuous decision stump II

- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.5
  accuracy = 50.8%
  (accuracy = one minus average 0–1 loss)
- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.3
  accuracy = 52.1%
- `feature` = 1st dim (i.e., along X-axis), `split` = −0.5
  accuracy = 83.2%



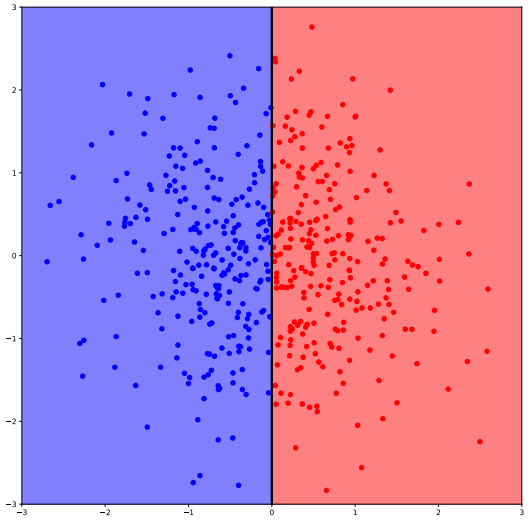
Continuous decision stump II

- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.5
  accuracy = 50.8%
  (accuracy = one minus average 0–1 loss)
- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.3
  accuracy = 52.1%
- `feature` = 1st dim (i.e., along X-axis), `split` = −0.5
  accuracy = 83.2%
- `feature` = 1st dim (i.e., along X-axis), `split` = 0.2
  accuracy = 92.4%



Continuous decision stump II

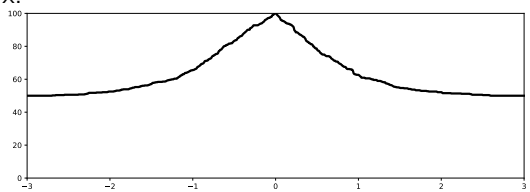
- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.5
  accuracy = 50.8%
  (accuracy = one minus average 0–1 loss)
- `feature` = 2nd dim (i.e., along Y-axis), `split` = 0.3
  accuracy = 52.1%
- `feature` = 1st dim (i.e., along X-axis), `split` = −0.5
  accuracy = 83.2%
- `feature` = 1st dim (i.e., along X-axis), `split` = 0.2
  accuracy = 92.4%
- `feature` = 1st dim (i.e., along X-axis), `split` = 0.0
  accuracy = 100%



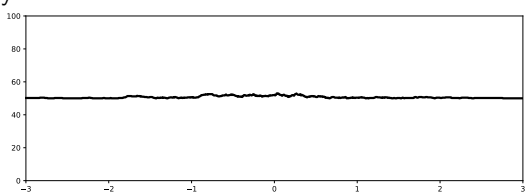
Continuous decision stump III

- How to find best parameters?
- Brute force:
  1. Sweep each dimension and consider every split
     (half way between each pair of exemplars when sorted along that axis)
  2. Evaluate loss function for every split
  3. Choose best

x:

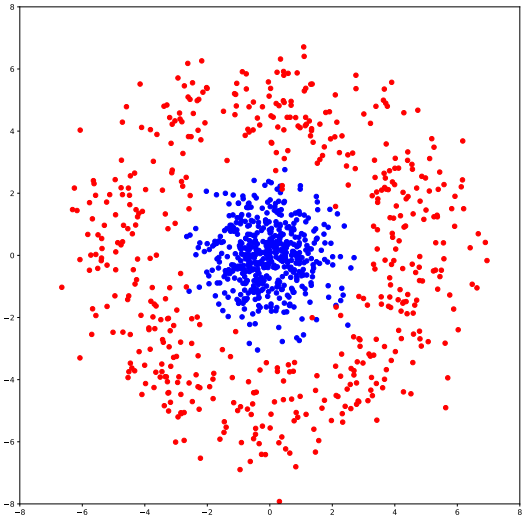


y:



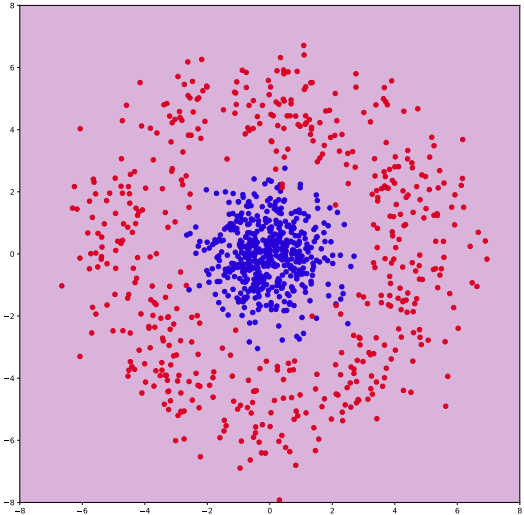
Continuous decision stump IV

- What about this?
- Axis-aligned separation is rare in real data!



Decision trees I

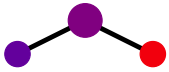
- What if we did this recursively?

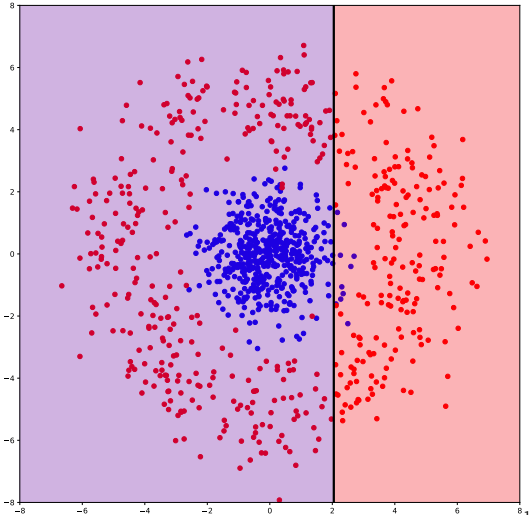


Decision trees I

- What if we did this recursively?

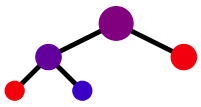
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree

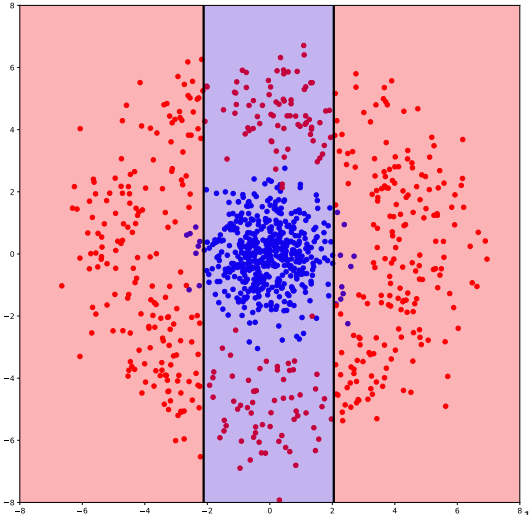




Decision trees I

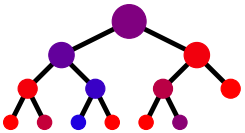
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again

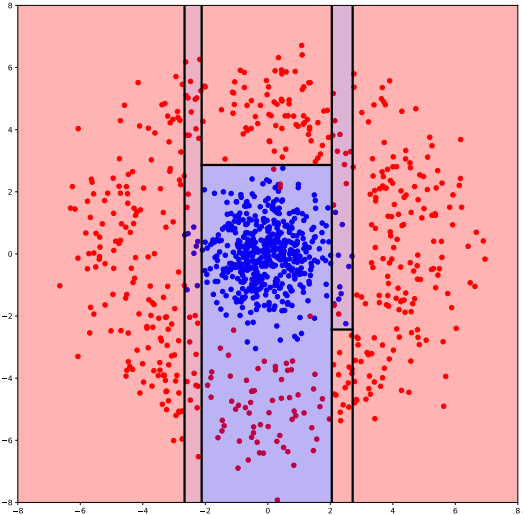




Decision trees I

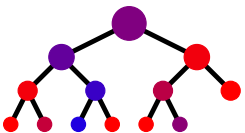
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again
- Jumping ahead...

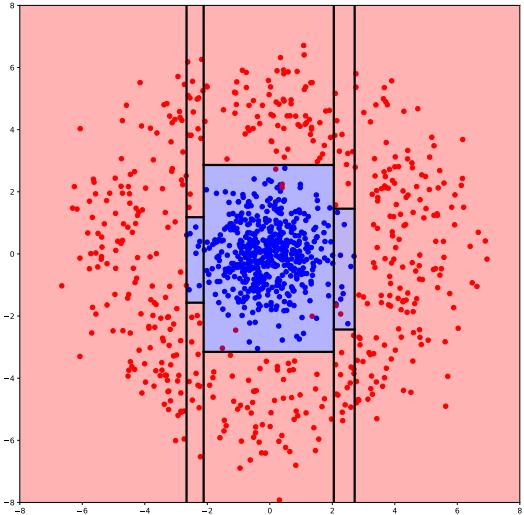




Decision trees I

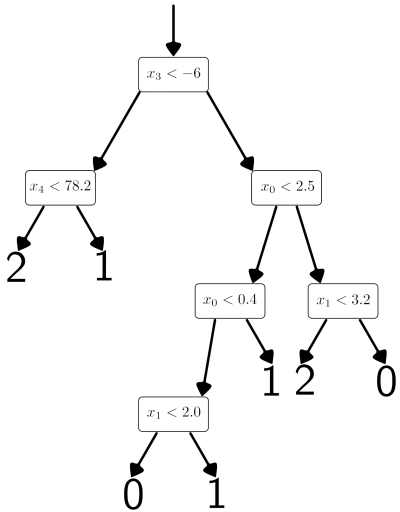
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again
- Jumping ahead...
- **Decision tree = recursive splitting**





Decision trees II

- The function parameter, θ, is a binary tree
- There are three classes (0,1 and 2), and four features (x_0, x_1, x_3, and x_4)
- Example - deciding a loan application [a], three classes could be **approve** (**0**), **reject** (**1**), and **hold** (**2**); Four features could be `age`, `salary/income`, `marital status`, `profession`

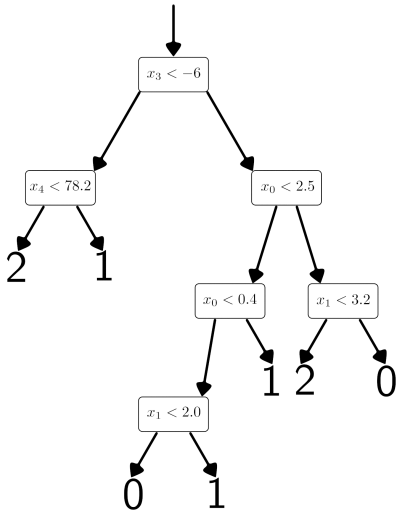


[a] without going into its merits

Decision trees II

- The function parameter, θ, is a binary tree
- There are three classes (0,1 and 2), and four features (x_0, x_1, x_3, and x_4)
- Example - deciding a loan application [a], three classes could be **approve** (**0**), **reject** (**1**), and **hold** (**2**); Four features could be `age`, `salary/income`, `marital status`, `profession`

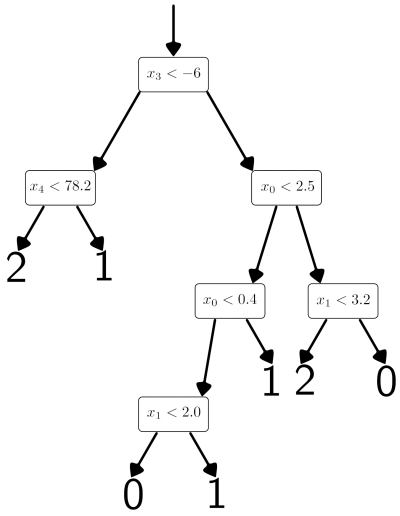
- There are two kinds of node:
  - **internal**, which contain a **split** based on a feature value (rounded rectangles)
  - **leaf**, which contain an **answer/final class** (big numbers)



[a] without going into its merits

Decision trees III

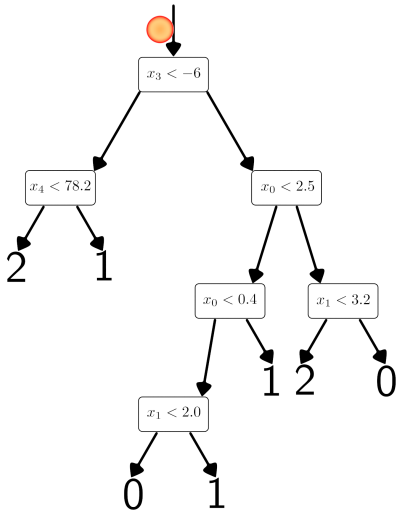
To evaluate the function, $f_\theta(x)$:



Decision trees III

To evaluate the function, $f_\theta(x)$:

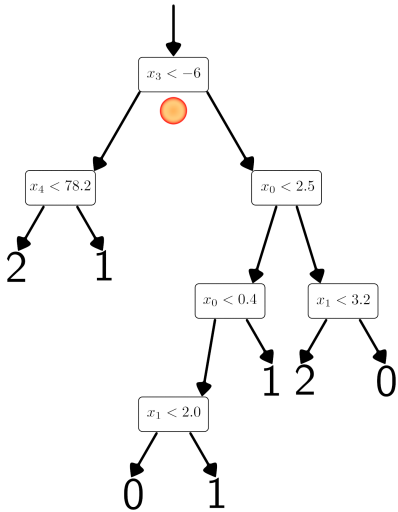
- Start at top



Decision trees III

To evaluate the function, $f_\theta(x)$:

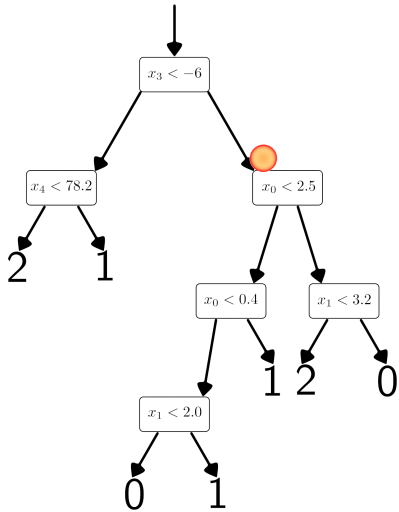
- Start at top
- Move to first split



Decision trees III

To evaluate the function, $f_\theta(x)$:

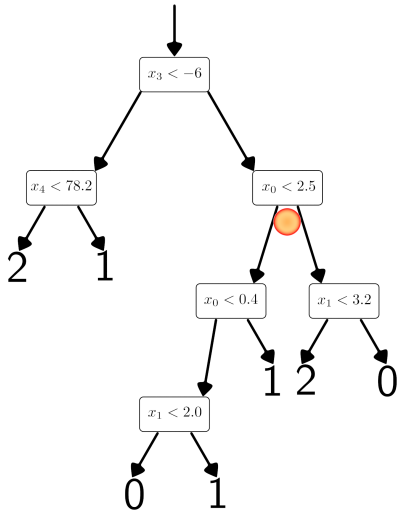
- Start at top
- Move to first split
- Go **left** or **right** based on test



Decision trees III

To evaluate the function, $f_\theta(x)$:

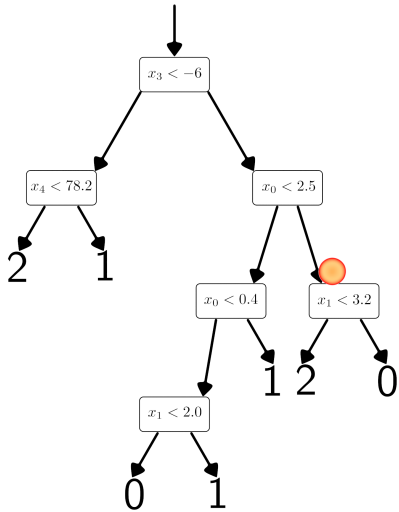
- Start at top
- Move to first split
- Go **left** or **right** based on test
- Move to next split, and so on...



Decision trees III

To evaluate the function, $f_\theta(x)$:

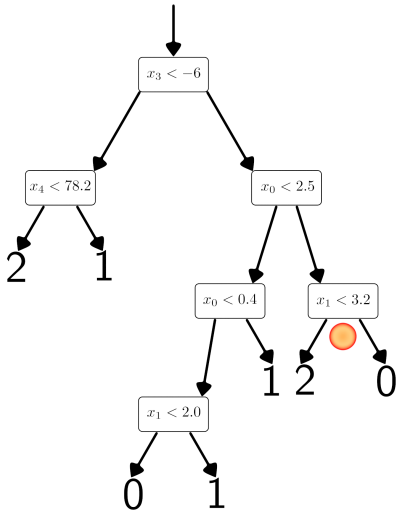
- Start at top
- Move to first split
- Go **left** or **right** based on test
- Move to next split, and so on...



Decision trees III

To evaluate the function, $f_\theta(x)$:

- Start at top
- Move to first split
- Go **left** or **right** based on test
- Move to next split, and so on...

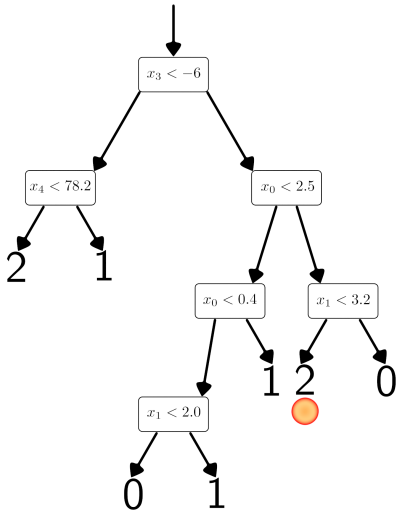


Decision trees III

To evaluate the function, $f_\theta(x)$:

- Start at top
- Move to first split
- Go **left** or **right** based on test
- Move to next split, and so on...

- Stop at leaf, return its answer



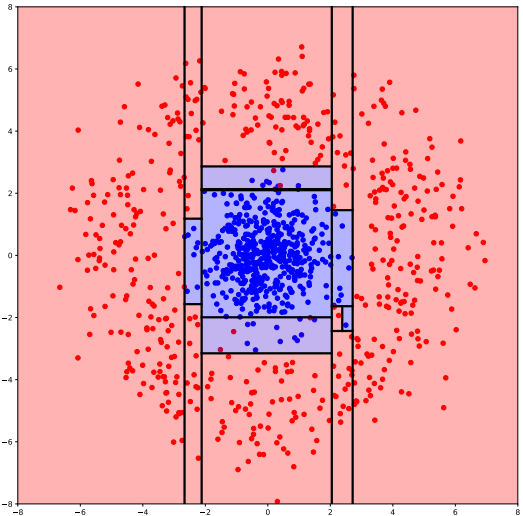
Decision trees IV

- Greedy optimisation of parameters (θ)
  (brute force at each node)

- Tree construction:
  1. Training data contains only one class $\rightarrow$ generate leaf node
  2. Otherwise:
     - Try all features / splits
     - Select best (lowest total loss)
     - Recurse (build another tree) for left and right children
       (train each tree with data that reaches it)

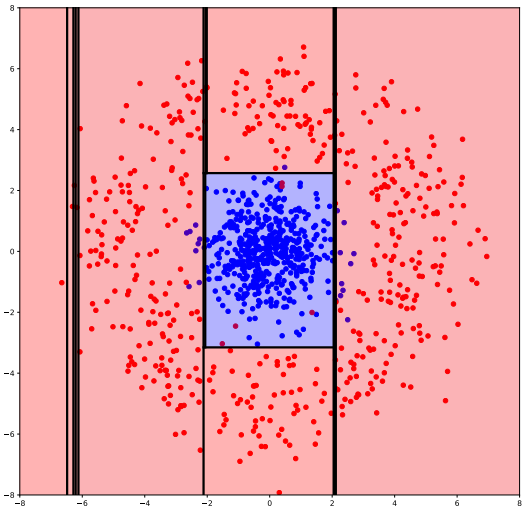
Decision trees V

- Need $L(y_i, f_\theta(x_i))$, the loss function



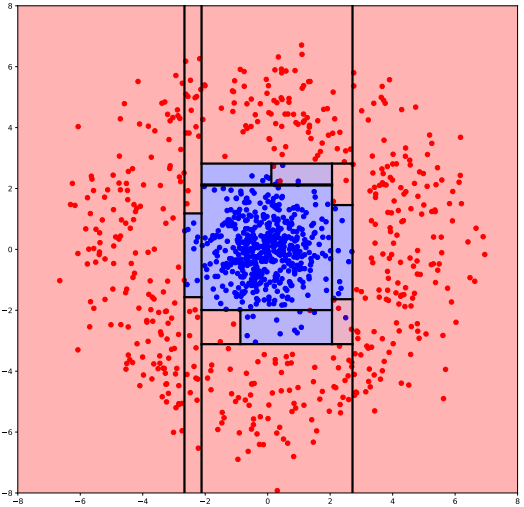
Decision trees V

- Need $L(y_i, f_\theta(x_i))$, the loss function
- **0–1 loss** works for decision stumps...
  ...but fails for decision trees!
  (wasted splits, can't refine initial rectangle)



Decision trees V

- Need $L(y_i, f_\theta(x_i))$, the loss function
- **0–1 loss** works for decision stumps...
  ...but fails for decision trees!
  (wasted splits, can't refine initial rectangle)
- Two that work:
  - Gini impurity (first image on this slide)
  - Information gain (current image)



Gini impurity

- "*Probability that if you select two items from a data set at random (with replacement) they will have a different class*"
  - Lowest (0): When there is only one class
  - Highest (< 1): When every data point has a different class

Gini impurity

- "*Probability that if you select two items from a data set at random (with replacement) they will have a different class*"
  - Lowest (0): When there is only one class
  - Highest (< 1): When every data point has a different class

- Let $p_i = P(\text{selecting class } i \text{ from data set})$ i.e. given some (sub)dataset, let p_i is the Probability of selecting class i from the dataset. Then the Gini impurity $G(p)$ is given as follows :

$$G(p) = \sum_i p_i(1 - p_i) = 1 - \sum_i p_i^2$$

Gini impurity

- "*Probability that if you select two items from a data set at random (with replacement) they will have a different class*"
  - Lowest (0): When there is only one class
  - Highest (< 1): When every data point has a different class

- Let $p_i = P(\text{selecting class } i \text{ from data set})$ i.e. given some (sub)dataset, let p_i is the Probability of selecting class i from the dataset. Then the Gini impurity $G(p)$ is given as follows :

$$G(p) = \sum_i p_i(1 - p_i) = 1 - \sum_i p_i^2$$

- Works because it **values partial success** (unlike 0–1 loss)

UNIVERSITY OF BATH

Weighting the split

- Can calculate $G(p^{(\texttt{left})})$ and $G(p^{(\texttt{right})})$ for halves of a split...
  ...but need a single number

- Weight by exemplar count:

$$L(\text{split}) = \frac{n_l}{n} G(p^{(\text{left})}) + \frac{n_r}{n} G(p^{(\text{right})})$$

n = total exemplar count
n_l = exemplars traveling down left branch
n_r = exemplars traveling down right branch

UNIVERSITY OF BATH

Weighting the split

- Can calculate $G(p^{(\texttt{left})})$ and $G(p^{(\texttt{right})})$ for halves of a split...
  ...but need a single number

- Weight by exemplar count:

$$L(\text{split}) = \frac{n_l}{n} G(p^{(\text{left})}) + \frac{n_r}{n} G(p^{(\text{right})})$$

  n = total exemplar count
  n_l = exemplars traveling down left branch
  n_r = exemplars traveling down right branch

- Intuitively: more data in a branch $\implies$ more important

- Information gain is weighted the same...

UNIVERSITY OF BATH

Information gain

- Conceptually: How much you *learn* from traversing a split
- Entropy:

$$H(p) = -\sum_i p_i \log(p_i)$$

  - Important – internet wouldn't work without it!
  - = bits required on average to encode data with given PDF
    (*bits* assumes $\log_2$; if $\log_e$ the unit is *nats*)

UNIVERSITY OF BATH

Information gain

- Conceptually: How much you *learn* from traversing a split
- Entropy:

$$H(p) = -\sum_i p_i \log(p_i)$$

  - Important – internet wouldn't work without it!
  - $=$ bits required on average to encode data with given PDF
    (*bits* assumes $\log_2$; if $\log_e$ the unit is *nats*)

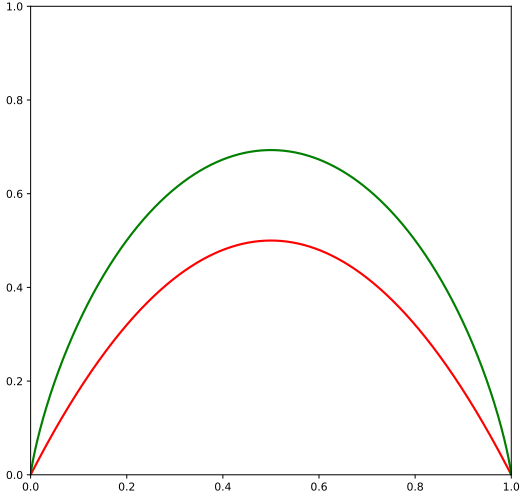
- Information gain $=$ number of bits/nats obtained from traversing the split:

$$I(\text{split}) = H(p^{(\text{parent})}) - \frac{n_l}{n} H(p^{(\text{left})}) - \frac{n_r}{n} H(p^{(\text{right})})$$

(you maximise this one!)

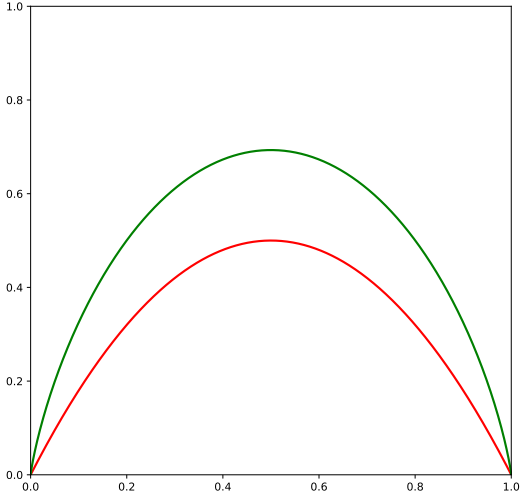
Which loss function?

- Typical conditions $\implies$ almost identical!
- Unsurprising as graphs very similar
  Red = Gini impurity
  Green = information gain



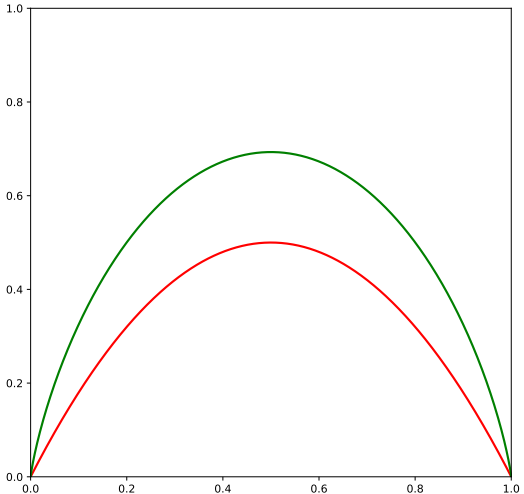
Which loss function?

- Typical conditions $\implies$ almost identical!
- Unsurprising as graphs very similar
  Red = Gini impurity
  Green = information gain
- Gini: Faster to compute (no log) $\therefore$ default.



Which loss function?

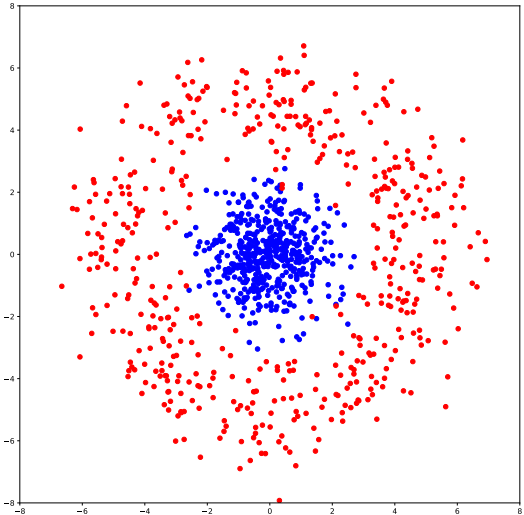
- Typical conditions $\implies$ almost identical!
- Unsurprising as graphs very similar
  Red = Gini impurity
  Green = information gain
- Gini: Faster to compute (no log) $\therefore$ default.
- Information gain:
  - better for some problems
  - works when Gini cannot, e.g. regression
  - has theory!



UNIVERSITY OF BATH

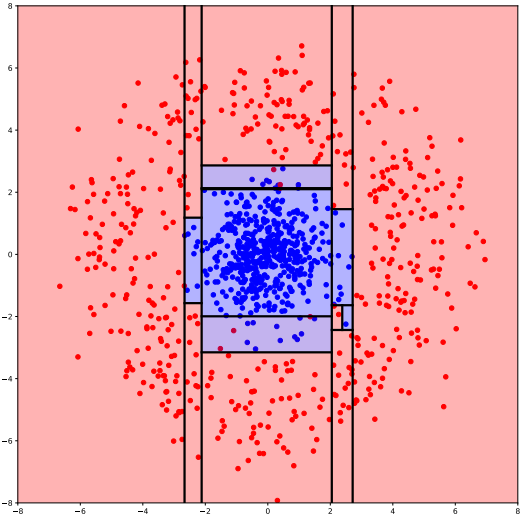
Overfitting

- Boundary is a circle...



Overfitting

- Boundary is a circle...
- but it does this...

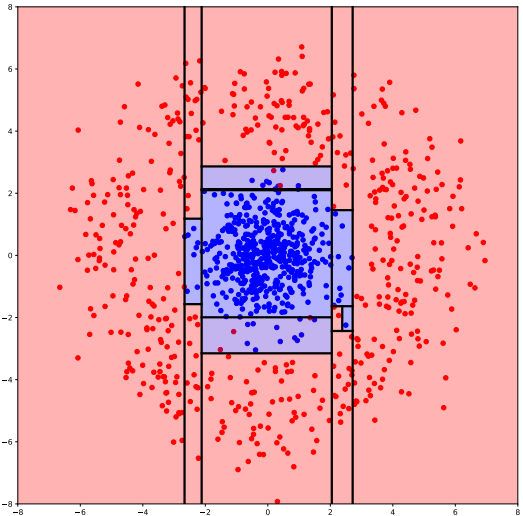


UNIVERSITY OF BATH

Overfitting

- Boundary is a circle...
- but it does this...

- This is called **overfitting**
  Modelling **noise**, not **signal**

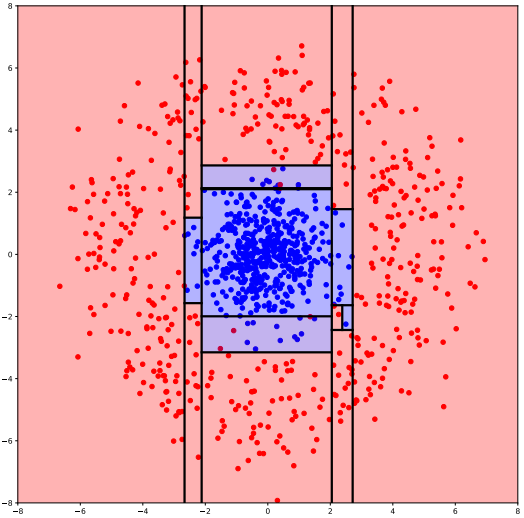


Overfitting

- Boundary is a circle...
- but it does this...

- This is called **overfitting**
  Modelling **noise**, not **signal**

- Can detect overfitting using a **test** set
  (algorithm can't fit noise it hasn't seen)



UNIVERSITY OF BATH

Early stopping

- Can avoid overfitting by *stopping early*
  - Limit on tree depth
  - Minimum leaf node size
- Prevents function getting too complicated!

UNIVERSITY OF BATH

Early stopping

- Can avoid overfitting by *stopping early*
  - Limit on tree depth
  - Minimum leaf node size
- Prevents function getting too complicated!
- These extra parameters are called **hyperparameters**

  (Gini or information gain is also one)
- Also optimised!

  (disturbingly often by hand)

A gloassary of ML problem settings

UNIVERSITY OF BATH

Supervised learning

- Learn a function: $y = f(\vec{x})$
- From (many) examples of input ($\vec{x}$) and output (y)

- Majority of ML:
  Classification or regression...

UNIVERSITY OF BATH

Supervised learning: Classification

- Learn a function: $y = f(\vec{x})$
- Classification: y is **discrete**

Supervised learning: Classification

- Learn a function: $y = f(\vec{x})$
- Classification: y is **discrete**

- Identifying camera trap animals
  - Input: Image
  - Output: Which animal

(peccary)

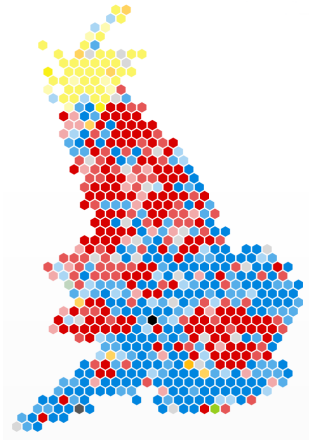
Supervised learning: Classification

- Learn a function: $y = f(\vec{x})$
- Classification: y is **discrete**

- Identifying camera trap animals
  - Input: Image
  - Output: Which animal

- Predicting voting intention
  - Input: Demographics
  - Output: Preferred candidate (probabilistic)
  - Run on entire country → Predict election winner

(peccary)



(YouGov, 2017-06-07)

UNIVERSITY OF BATH

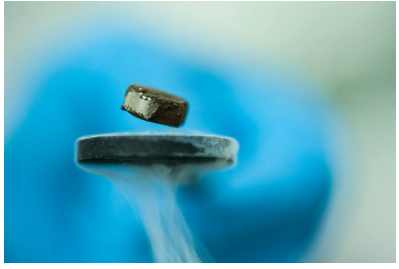
Supervised learning: Regression

- Learn a function: $y = f(\vec{x})$
- Regression: y is **continuous**

Supervised learning: Regression

- Learn a function: $y = f(\vec{x})$
- Regression: y is **continuous**

- Predicting critical temperature of a superconductor
  - Input: Material properties
  - Output: Temperature

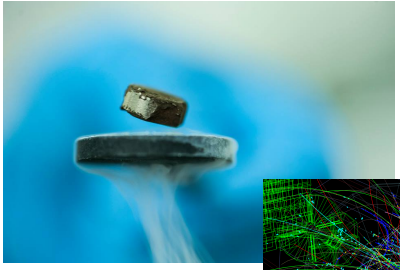


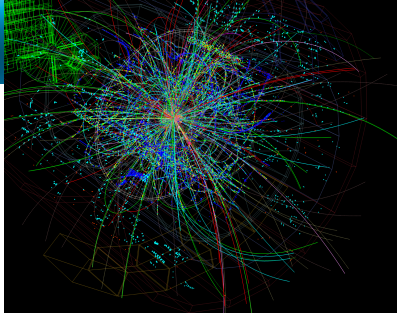
Supervised learning: Regression

- Learn a function: $y = f(\vec{x})$
- Regression: y is **continuous**

- Predicting critical temperature of a superconductor
  - Input: Material properties
  - Output: Temperature

- Inferring particle paths (LHC)
  - Input: Detector energy spikes
  - Output: Particle paths
  - Trained with simulation





UNIVERSITY OF BATH

Supervised learning: Further kinds

- Multi-label classification: y is a **set**
  - e.g. identifying objects in an image
  - e.g. text summarisation (reusing source sentences)

UNIVERSITY OF BATH

Supervised learning: Further kinds

- Multi-label classification: y is a **set**
  - e.g. identifying objects in an image
  - e.g. text summarisation (reusing source sentences)

- Structured prediction: y is anything else!
  - e.g. Sentence tagging: y is a sequence
    (such as part-of-speech tagging)
  - e.g. Automated design: y is a CAD model

UNIVERSITY OF BATH

Unsupervised learning

- No y!
- Finds *patterns* in data

- Examples:
  - Clustering
  - Density estimation
  - Dimensionality reduction

UNIVERSITY OF BATH

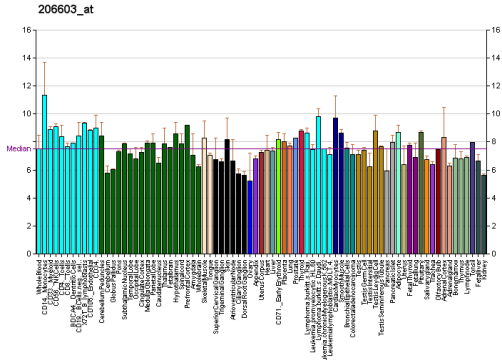
Unsupervised learning: Clustering

- Clustering:
  - Groups “similar” data points
  - Arbitrary similarity definition

Unsupervised learning: Clustering

- Clustering:
  - Groups "similar" data points
  - Arbitrary similarity definition
- Identifying *co-regulated genes*:
  - Input: Many expression level measurements
  - Output: Groups of genes that tend to express at same time

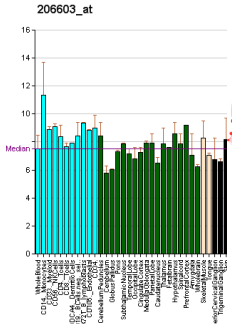
(Gene expression of SLC2A4)

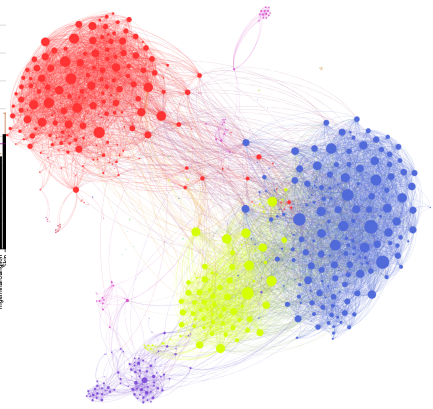


Unsupervised learning: Clustering

- Clustering:
  - Groups "similar" data points
  - Arbitrary similarity definition
- Identifying *co-regulated genes*:
  - Input: Many expression level measurements
  - Output: Groups of genes that tend to express at same time
- Discovering social groups
  - Input: Friend graph
  - Output: Social groups (individuals may belong to several)

(Gene expression of SLC2A4)





(a Facebook friend graph)

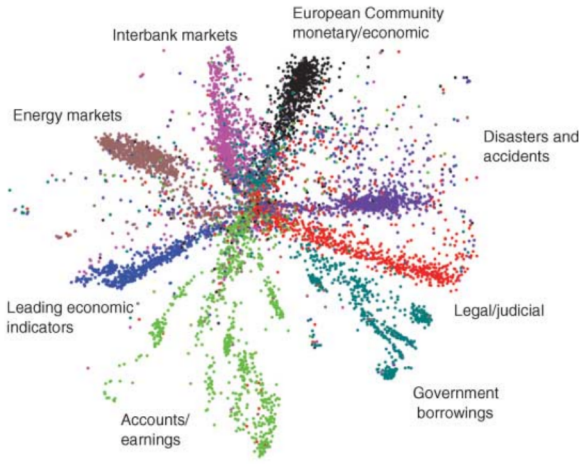
UNIVERSITY OF BATH

Unsupervised learning: Dimensionality reduction

- Dimensionality reduction / manifold learning:
  - Reduce dimensions while preserving information
  - Also used for visualisation (important for verification)

Unsupervised learning: Dimensionality reduction

- Dimensionality reduction / manifold learning:
  - Reduce dimensions while preserving information
  - Also used for visualisation (important for verification)
- Organising news
  - Input: Word vectors
  - Output: Position in layout



UNIVERSITY OF BATH

*-supervised

- Collecting data cheap
- Labelling data expensive

- **Semi-supervised**:
  - Some labelled data
  - Lots of unlabelled data

UNIVERSITY OF BATH

*-supervised

- Collecting data cheap
- Labelling data expensive

- **Semi-supervised**:
  - Some labelled data
  - Lots of unlabelled data

- Precise labels expensive, inaccurate labels cheap

- **Weakly-supervised**:
  - Learns from “weak” labels
  - Outputs “strong” labels

*-supervised

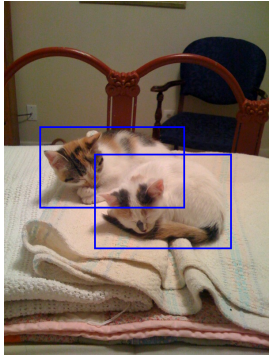
- Collecting data cheap
- Labelling data expensive

- **Semi-supervised**:
  - Some labelled data
  - Lots of unlabelled data

- Precise labels expensive, inaccurate labels cheap

- **Weakly-supervised**:
  - Learns from "weak" labels
  - Outputs "strong" labels

- e.g. finding cats
  - Image contains cat – fast
  - Box around cat – slow



UNIVERSITY OF BATH

Summary

- Introduced decision trees
  - Good at ignoring useless data
  - Can be interpretable
  - Overfits most of the time
- Overfitting, testing, hyperparameters $\rightarrow$ future lectures
- Next lecture: Extending decision trees to random forests (which are much, much better)