Machine Learning 1.02: Decision Trees

Tom S. F. Haines
T.S.F.Haines@bath.ac.uk

What kinds of problems can ML solve?
(a glossary)

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)

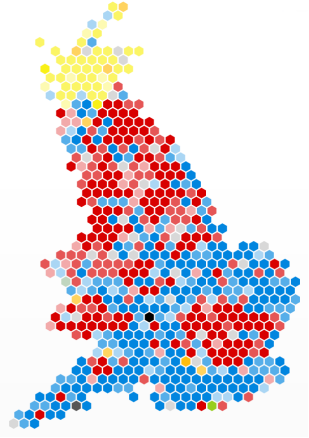
Supervised learning: Classification

- 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 $\rightarrow$ Predict election winner

(peccary)



(YouGov, 2017-06-07)

UNIVERSITY OF BATH

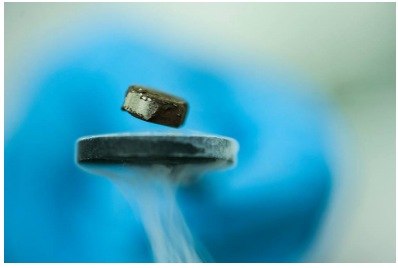
Supervised learning: Regression

- 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

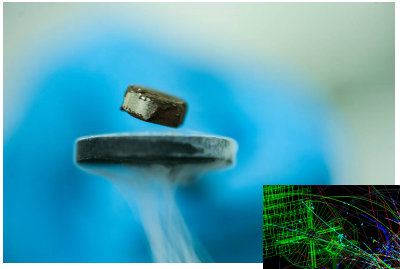


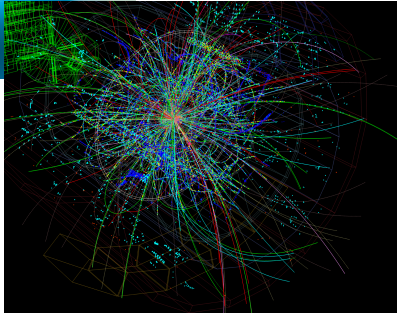
Supervised learning: Regression

- 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

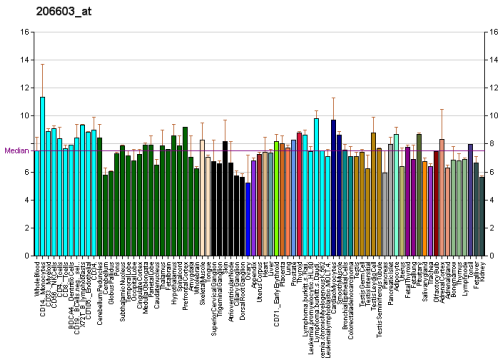
Unsupervised learning: Clustering

- 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

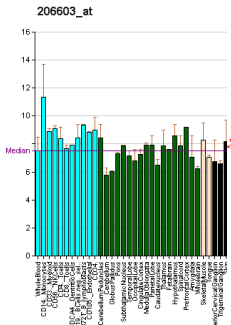
(Gene expression of SLC2A4)

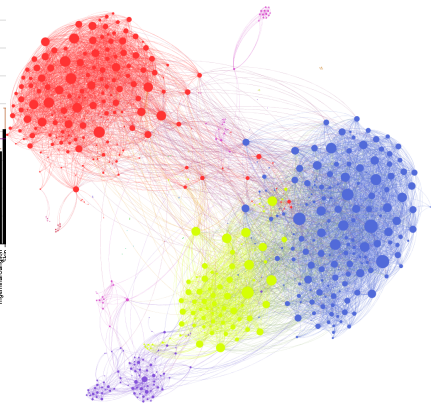


Unsupervised learning: Clustering

- 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

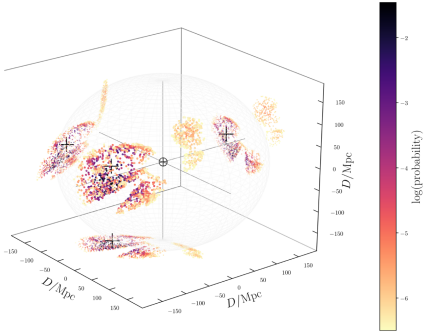
Unsupervised learning: Density estimation

- Density estimation:
  - Learns distribution of data
  - i.e. $x_i \sim P$

Unsupervised learning: Density estimation

- Density estimation:
  - Learns distribution of data
  - i.e. $x_i \sim P$

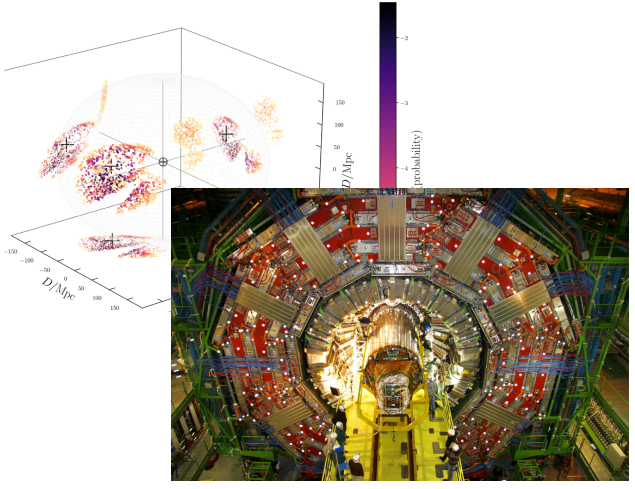
- Finding coalescing binary neutron stars with LIGO
  - Input: Locations that explain detection
  - Output: Search order for optical follow up



Unsupervised learning: Density estimation

- Density estimation:
  - Learns distribution of data
  - i.e. $x_i \sim P$
- Finding coalescing binary neutron stars with LIGO
  - Input: Locations that explain detection
  - Output: Search order for optical follow up
- Detecting LHC problems
  - Input: Normal outputs
  - Output: Possible failures

  Example of *abnormality detection*



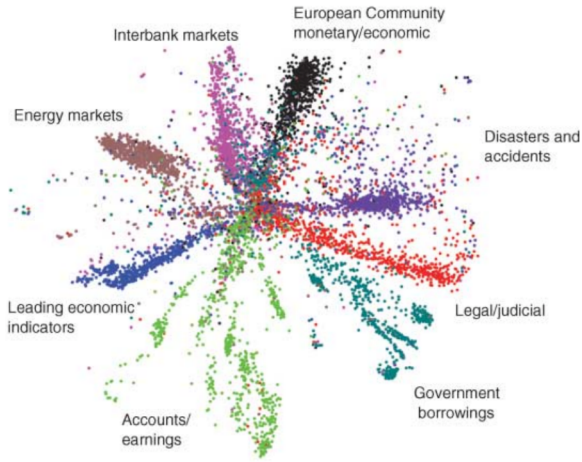
UNIVERSITY OF BATH

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



*-supervised

- Collecting data cheap
- Labelling data expensive

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

*-supervised

- Collecting data cheap
- Labelling data expensive

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

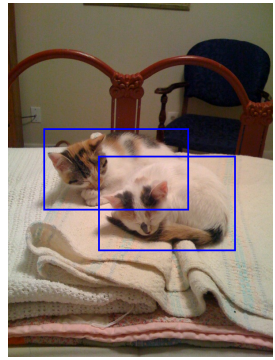
- Precise labels expensive, inaccurate labels cheap

- **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



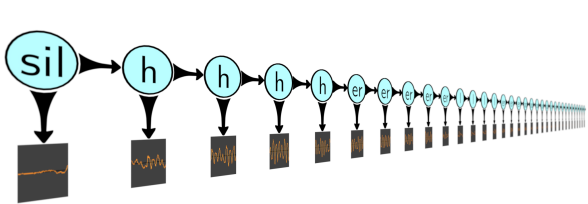
Graphical models

- **Represents structure** by drawing relationships...

Graphical models

- **Represents structure** by drawing relationships...

- Voice recognition
- Hidden Markov model, used twice:
  1. Align phonemes with audio (weakly-supervised)
  2. Recognition using language model (structured prediction)



Graphical models

- **Represents structure** by drawing relationships...

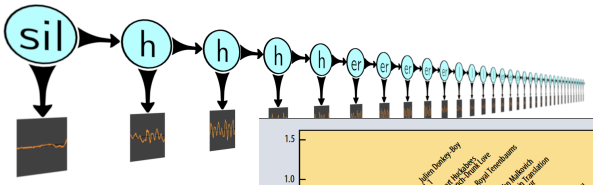
- Voice recognition
- Hidden Markov model, used twice:
  1. Align phonemes with audio (weakly-supervised)
  2. Recognition using language model (structured prediction)

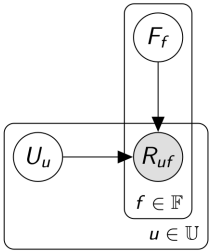
- Recommender systems

  e.g. for films

  - Input: User ratings (sparse)
  - Output: Omitted ratings

  (estimating missing values)





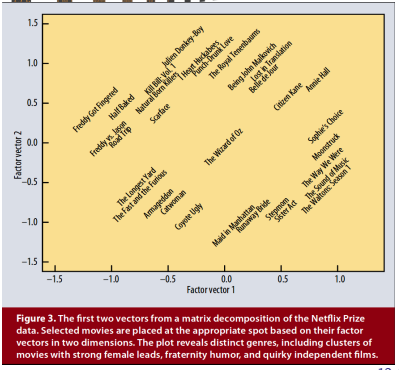


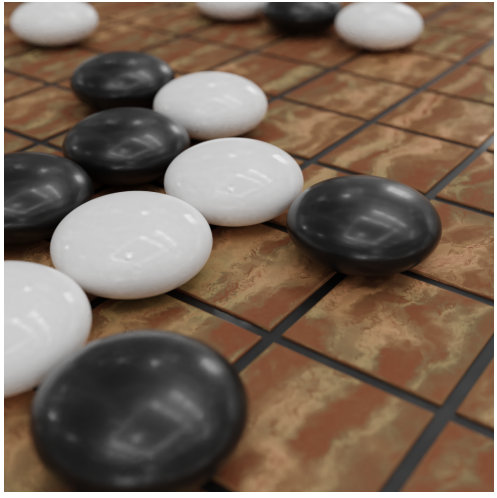
Figure 3. The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films.

UNIVERSITY OF BATH

Reinforcement learning

- *Actions* in an *environment*
- *Delayed reward*

- Examples:
  - Games, e.g. *Alpha Go*
  - Agents (inc. working together)
  - Robots



UNIVERSITY OF BATH

Glossary

- **Supervised**
  - Classification
  - Regularisation
  - Multi-label classification
  - Structured prediction
- Semi-supervised
- Weakly-supervised
- Graphical models

(incomplete!)

- **Unsupervised**
  - Clustering
  - Density estimation / abnormality detection
  - Dimensionality reduction / manifold learning
- **Reinforcement learning**

UNIVERSITY OF BATH

Further categories

Can also classify ML algorithms by...

UNIVERSITY OF BATH

Further categories

Can also classify ML algorithms by...

- Answer quality:
  - Point estimate
    e.g. "*You have cancer*"
  - Probabilistic
    e.g. "*60% chance you have cancer*"
  - Bayesian
    e.g. "*5% chance you have cancer*"

UNIVERSITY OF BATH

Further categories

Can also classify ML algorithms by...

- Answer quality:
  - Point estimate
    e.g. "*You have cancer*"
  - Probabilistic
    e.g. "*60% chance you have cancer*"
  - Bayesian
    e.g. "*5% chance you have cancer*"
- Workflow:
  - Batch learning
    i.e. Collect data then learn
  - Incremental learning
    i.e. Learn as data arrives
  - Active learning
    i.e. Algorithm selects data to learn from!
    (leads to automating science...)

UNIVERSITY OF BATH

Further categories

Can also classify ML algorithms by...

- Answer quality:
  - Point estimate
    e.g. "*You have cancer*"
  - Probabilistic
    e.g. "*60% chance you have cancer*"
  - Bayesian
    e.g. "*5% chance you have cancer*"
- Area:
  - Traditional
  - Computer vision
  - Natural language processing (NLP)
  - Interactive
- Workflow:
  - Batch learning
    i.e. Collect data then learn
  - Incremental learning
    i.e. Learn as data arrives
  - Active learning
    i.e. Algorithm selects data to learn from!
    (leads to automating science...)

UNIVERSITY OF BATH

This lecture

- Glossary
- Decision tree – supervised classification
- A good starter algorithm – easy to understand
- Next lecture is random forests, which build 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
  - $y \in \mathrm{N}$ is the output class

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
  - $y \in \mathrm{N}$ is the output class
- 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?

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: $\sum_{i=1}^{N} L(y_i, f_\theta(x_i))$

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
```

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**)

Decision stump III

- What is the space of functions it learns?

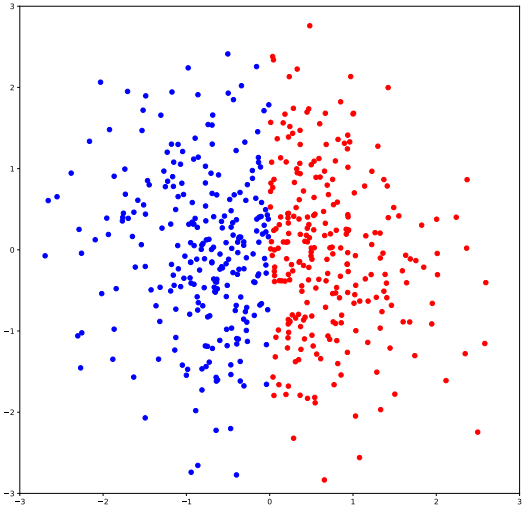
UNIVERSITY OF BATH

Decision stump III

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

Continuous decision stump I

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

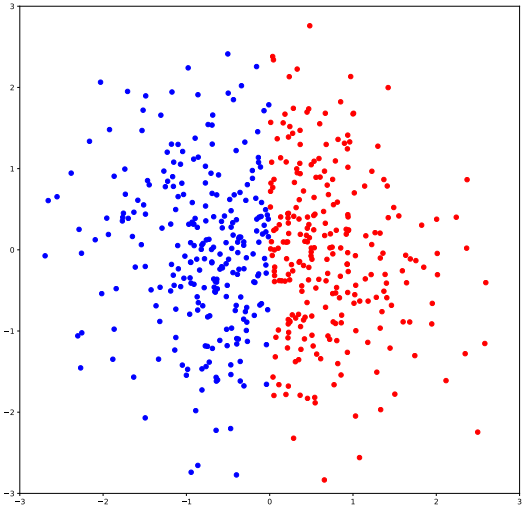


Continuous decision stump I

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

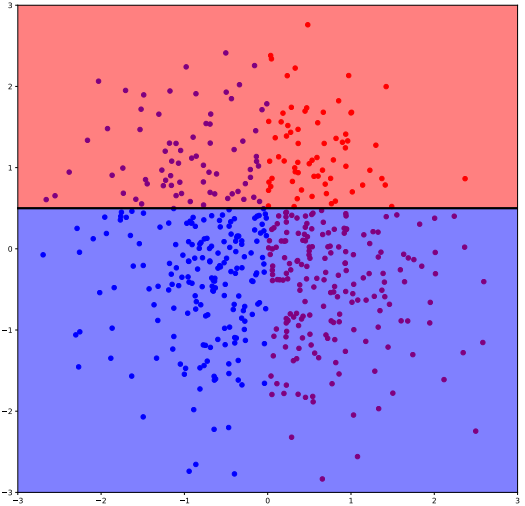
$$f_\theta(x) = \begin{cases} 0 & x_{\texttt{feature}} < \texttt{split} \\ 1 & x_{\texttt{feature}} \geq \texttt{split} \end{cases}$$

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



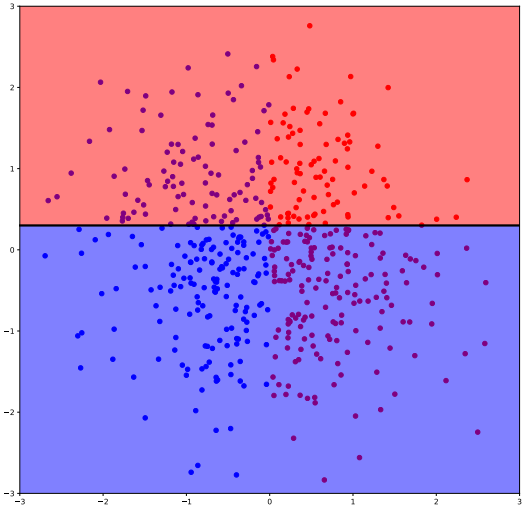
Continuous decision stump II

- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)



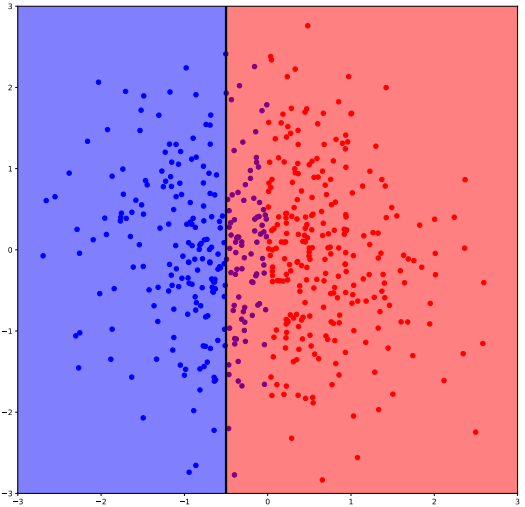
Continuous decision stump II

- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$



Continuous decision stump II

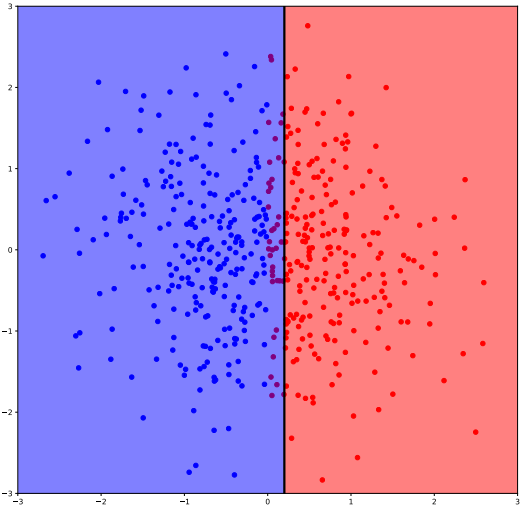
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$



UNIVERSITY OF BATH

Continuous decision stump II

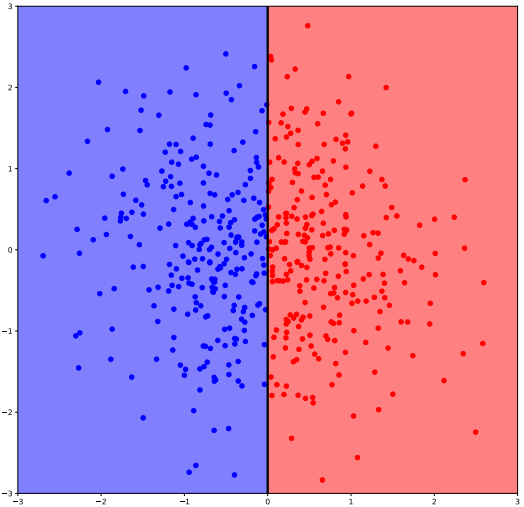
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$
- $\texttt{feature} = x, \texttt{split} = 0.2$
  accuracy $= 92.4\%$



UNIVERSITY OF BATH

Continuous decision stump II

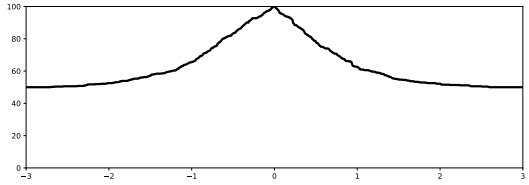
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$
- $\texttt{feature} = x, \texttt{split} = 0.2$
  accuracy $= 92.4\%$
- $\texttt{feature} = x, \texttt{split} = 0.0$
  accuracy $= 100\%$



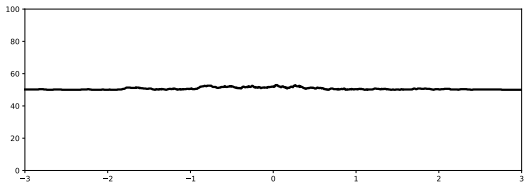
Continuous decision stump III

- So how do we find the best parameters?
- Brute force:
  1. Sweep each dimension, considering 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:

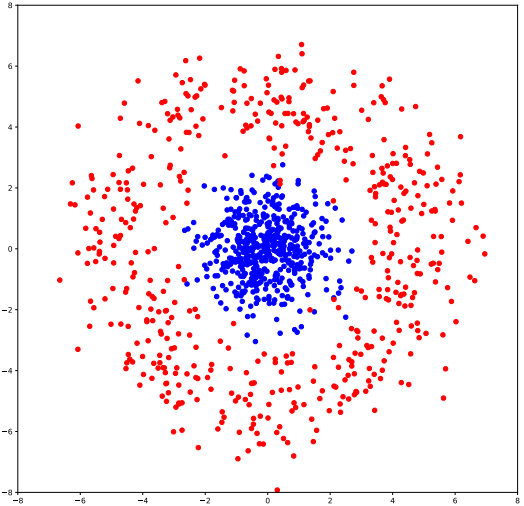


y:



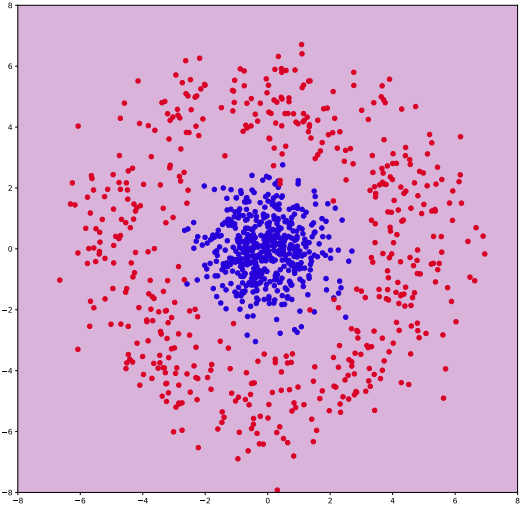
Continuous decision stump IV

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



Decision trees I

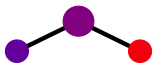
- What if we did this recursively?

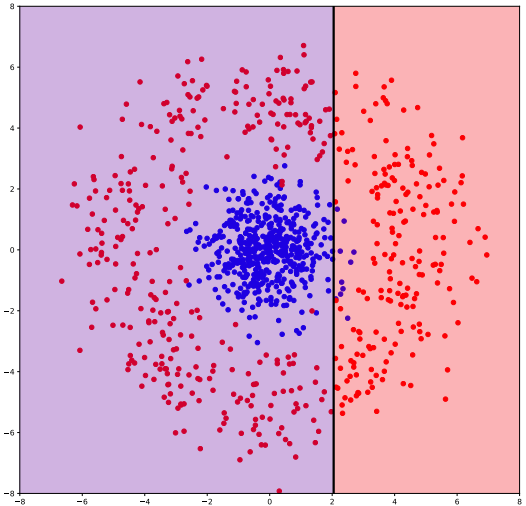


Decision trees I

- What if we did this recursively?

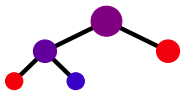
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree

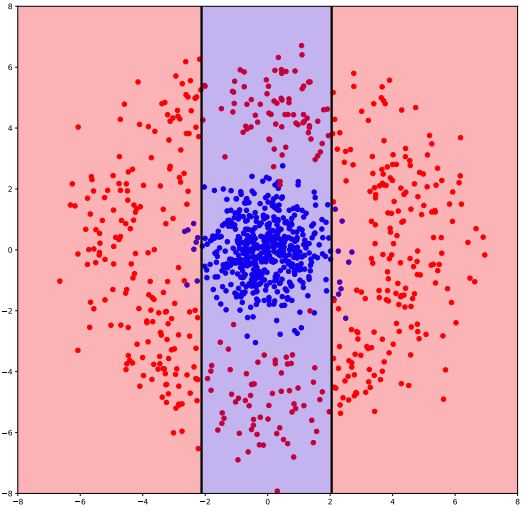




Decision trees I

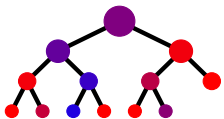
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again

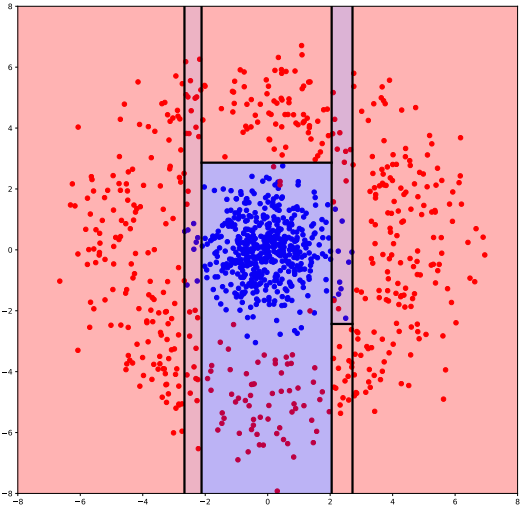




Decision trees I

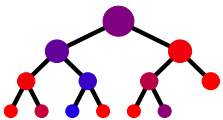
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again
- Jumping ahead...

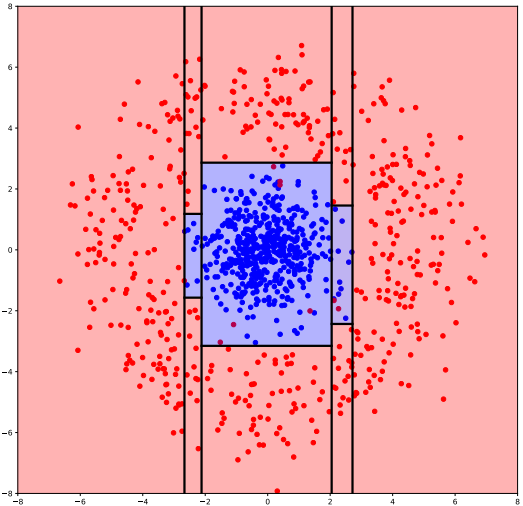




Decision trees I

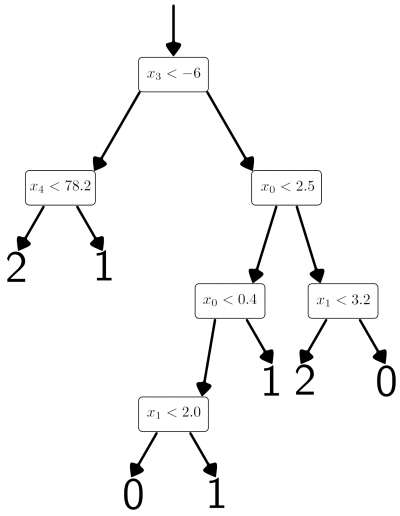
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points
- First split – as before
- Can be represented as a tree
- Have split left half of first split again
- Jumping ahead...
- **Decision tree = recursive splitting**





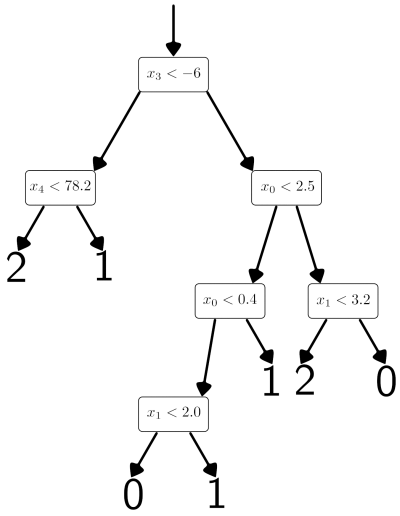
Decision trees II

- The function parameter, θ, is a binary tree



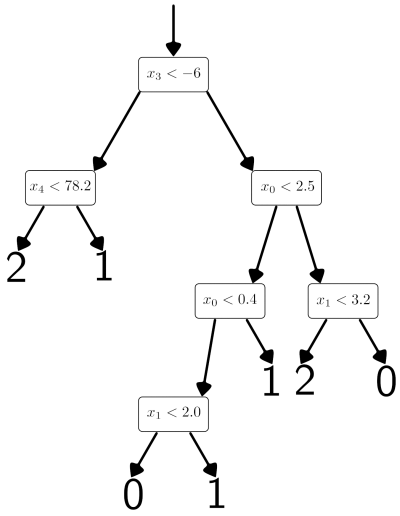
Decision trees II

- The function parameter, θ, is a binary tree
- You have two kinds of node:
  - **internal**, which contain a **split** (rounded rectangles)
  - **leaf**, which contain an **answer** (big numbers)



Decision trees III

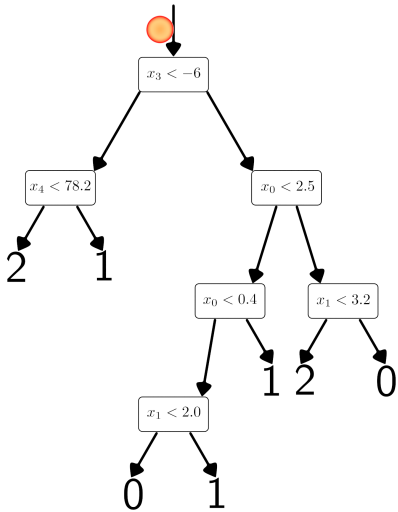
To evaluate the function, $f_\theta(x)$:



Decision trees III

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

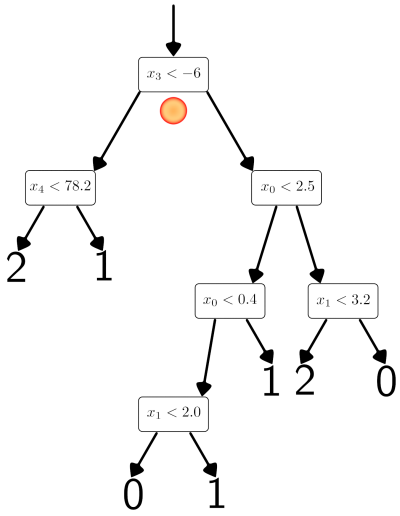
- Start at the top



Decision trees III

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

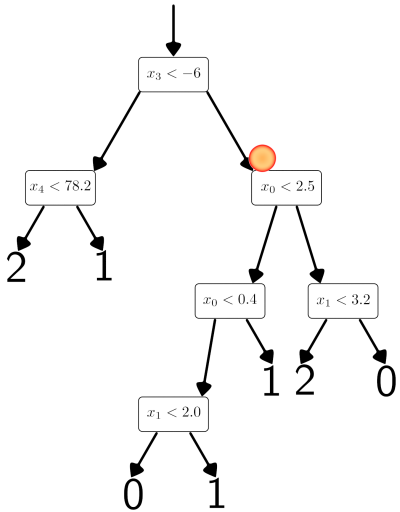
- Start at the top
- Move to the first split



Decision trees III

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

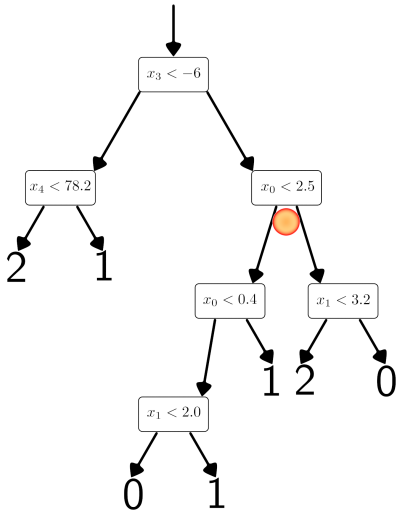
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test



Decision trees III

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

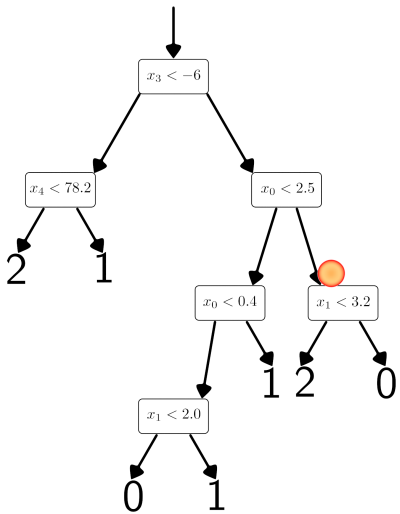
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...



Decision trees III

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

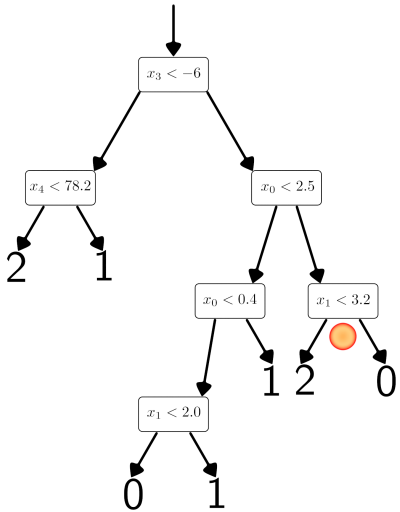
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...



Decision trees III

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

- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...

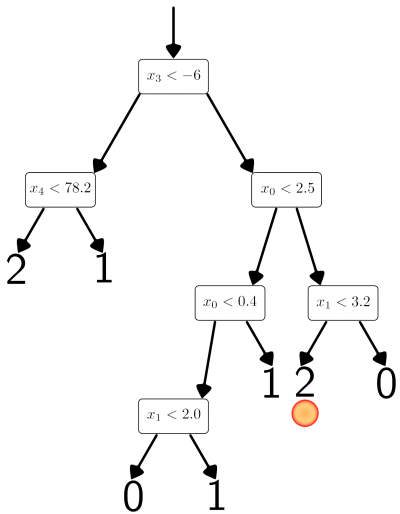


Decision trees III

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

- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...

- Stop when you reach a leaf, and return it's answer



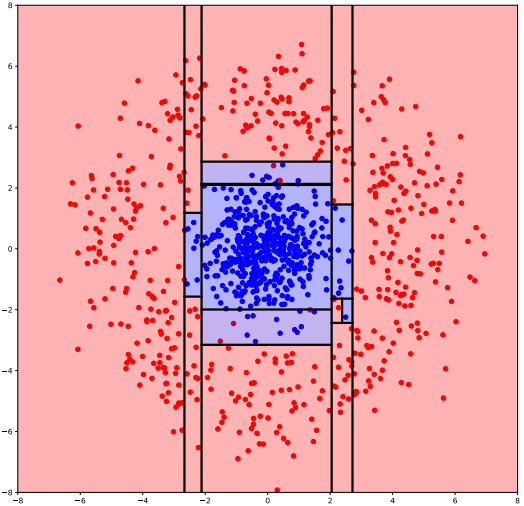
Decision trees IV

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

- Tree construction:
  1. If provided training data only contains one class generate a leaf node
  2. Otherwise, try all features and all splits of provided data
  3. Select feature and split with lowest total loss
  4. Recurse (build another tree) for both the left and right children
     Provide each with the training data that would be sent down that branch

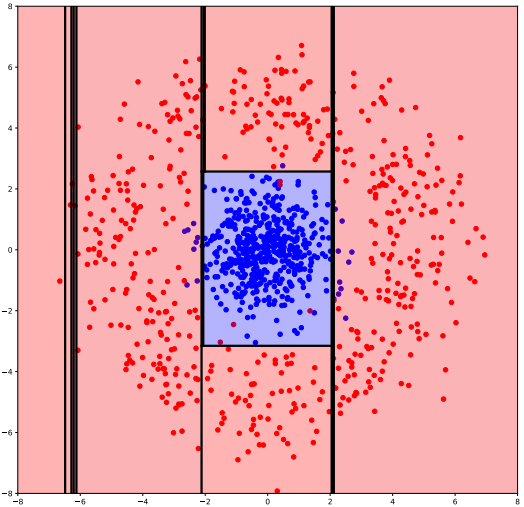
Decision trees V

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



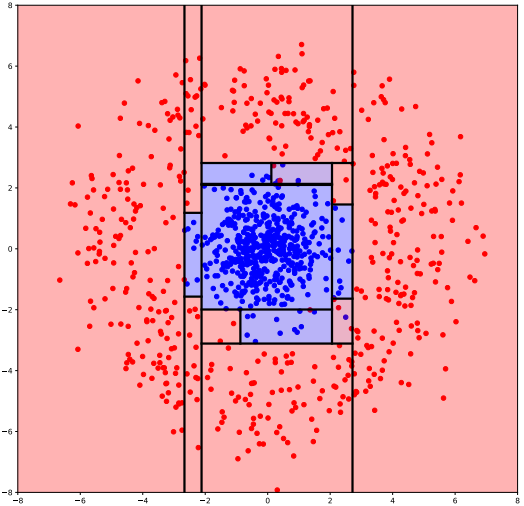
Decision trees V

- Need $L(y_i, f_\theta(x_i))$, the loss function
- **0–1 loss** works for decision stumps...
  ...but fails for decision trees!
  (note the wasted splits and inability to refine the 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!
  (note the wasted splits and inability to refine the initial rectangle)
- Two that actually 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: 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: When every data point has a different class

- p_i = probability of selecting class i from the data set

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

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: When every data point has a different class

- p_i = probability of selecting class i from the data set

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

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

Weighting the split

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

- Simple weighting by exemplar count:

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

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

UNIVERSITY OF BATH

Weighting the split

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

- Simple weighting by exemplar count:

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

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

- Intuitively, the more data is in a branch the more important it is

- Do the same with information gain...

Information gain

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

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

  - Really important – internet wouldn't work without it!
  - Measures how many bits are required on average to encode data with a given probability distribution (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:

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

  - Really important – internet wouldn't work without it!
  - Measures how many bits are required on average to encode data with a given probability distribution (bits assumes $\log_2$, if $\log_e$ the unit is nats)

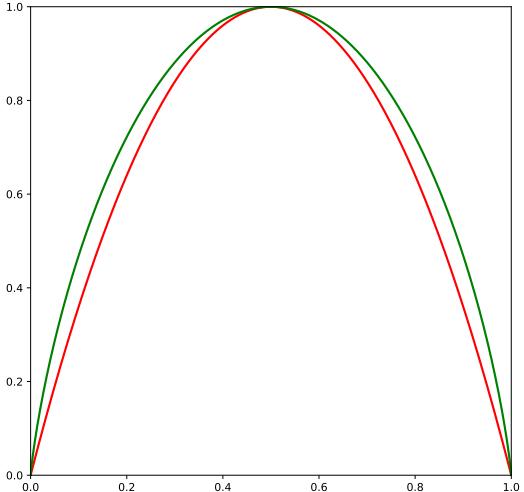
- Information gain = number of bits/nats obtained from traversing the split:

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

(you maximise this one!)

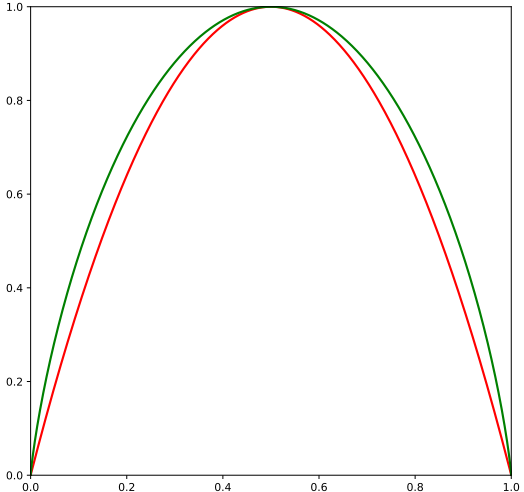
Which loss function?

- Under normal conditions they are basically identical!
- Unsurprising as graphs very similar: Green is Gini impurity, red information gain.



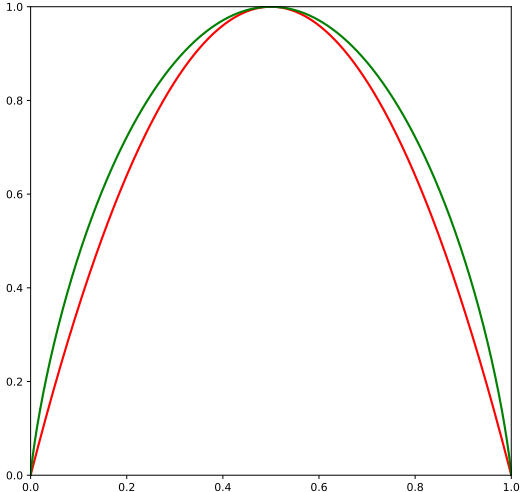
Which loss function?

- Under normal conditions they are basically identical!
- Unsurprising as graphs very similar: Green is Gini impurity, red information gain.
- Gini: Faster to compute (no log) $\therefore$ default.



Which loss function?

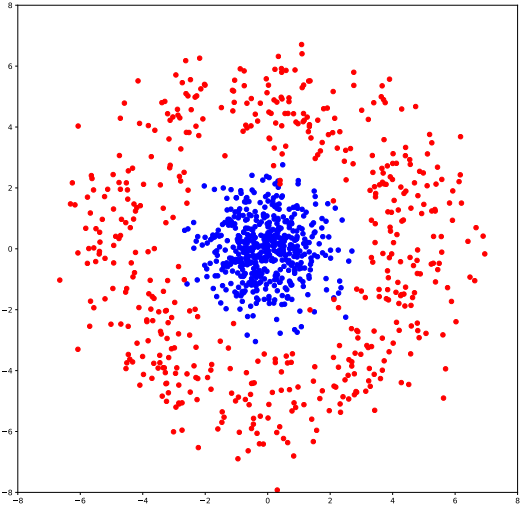
- Under normal conditions they are basically identical!
- Unsurprising as graphs very similar: Green is Gini impurity, red information gain.
- Gini: Faster to compute (no log) $\therefore$ default.
- Information gain can be better for some problems, and works in situations where Gini cannot, e.g. regression
- Information gain has theory!



UNIVERSITY OF BATH

Overfitting

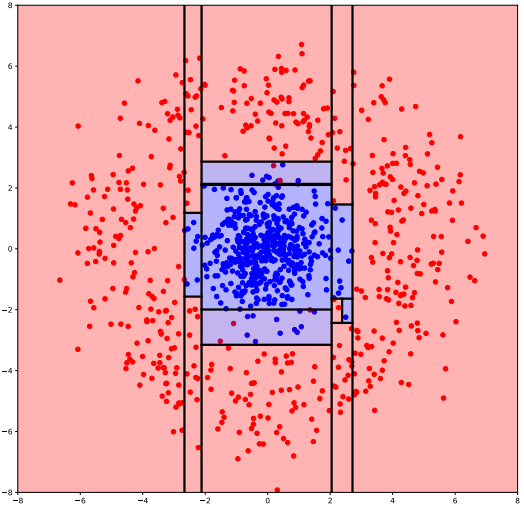
- The boundary is clearly a circle...



UNIVERSITY OF BATH

Overfitting

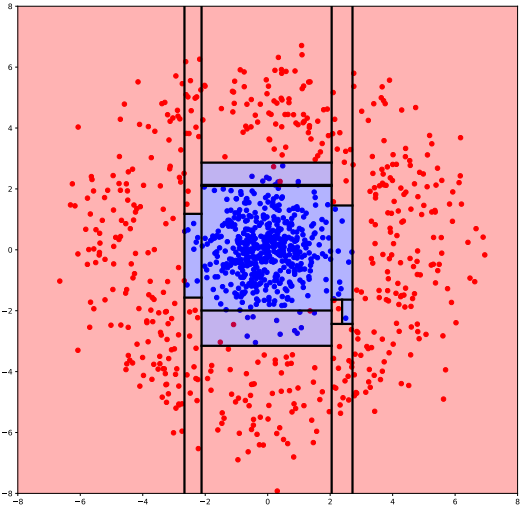
- The boundary is clearly a circle...
- But it does this...



Overfitting

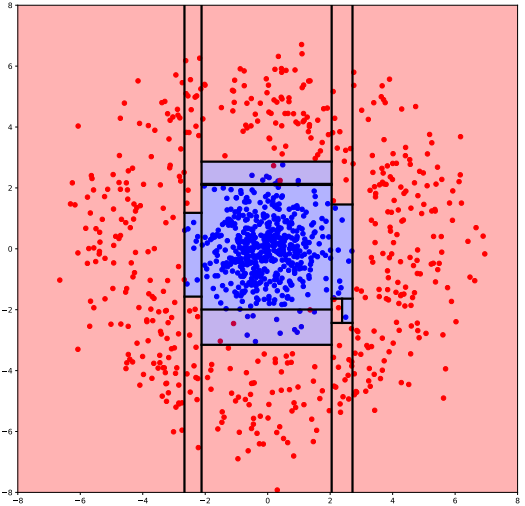
- The boundary is clearly a circle...
- But it does this...

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



Overfitting

- The boundary is clearly 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)



Early stopping

- You can avoid overfitting by stopping early
  - Limit on tree depth
  - Minimum leaf node size
- This prevents the function getting too complicated!

UNIVERSITY OF BATH

Early stopping

- You can avoid overfitting by stopping early
  - Limit on tree depth
  - Minimum leaf node size
- This prevents the function getting too complicated!
- These extra parameters are called **hyperparameters**
  (Gini or information gain is also one)
- They are also optimised!
  (disturbingly often by hand)

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)