Machine Learning 1.02: Decision Trees

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

Types of machine learning

- **Supervised** – Learn from labelled (input, output) pairs.
- **Unsupervised** – Input only, no answer.

UNIVERSITY OF BATH

Types of machine learning

- **Supervised** – Learn from labelled (input, output) pairs.
- **Unsupervised** – Input only, no answer.

- **Semi-supervised** – Some data is labelled, some is not.
- **Weakly-supervised** – Data is labelled, but not with the output we want, rather something related.
- **Reinforcement learning** – Algorithm outputs a sequence of actions within an environment, and may not found out how well it did until the very end.

⋮

Supervised learning

- **Supervised** – Learn from labelled (input, output) pairs.

UNIVERSITY OF BATH

Supervised learning

- **Supervised** – Learn from labelled (input, output) pairs.
- output $= f(\text{input})$ – goal is to learn function $f(\cdot)$.

Types of supervised learning

- **Classification** – Output of $f(\cdot)$ is discrete,
  e.g. $0, 1, \ldots$ or $\mathrm{cat}, \mathrm{dog}, \mathrm{parrot}, \ldots$
- **Regression** – Output of $f(\cdot)$ is continuous,
  e.g. $5.0, 6.36, \pi, \ldots$

UNIVERSITY OF BATH

Types of supervised learning

- **Classification** – Output of $f(\cdot)$ is discrete,
  e.g. $0, 1, \ldots$ or $\mathrm{cat}, \mathrm{dog}, \mathrm{parrot}, \ldots$
- **Regression** – Output of $f(\cdot)$ is continuous,
  e.g. $5.0, 6.36, \pi, \ldots$

- **Multi-label classification** – Output is a set of labels, rather than just one,
  e.g. objects seen in an image.
- **Structured prediction** – Output is "something else",
  e.g. a graph, or a sentence.

$\vdots$

This lecture

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

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
```

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

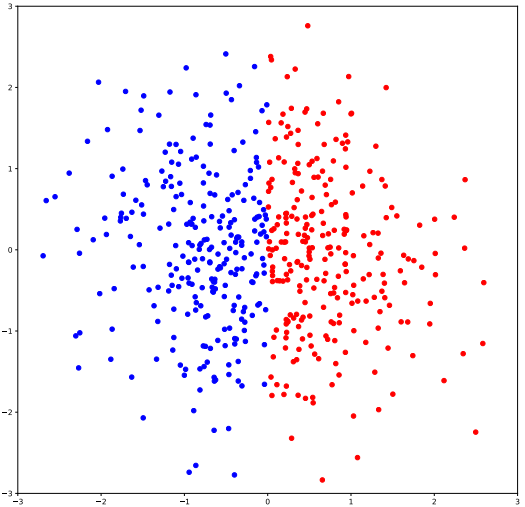
- What is the space of functions it learns?

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)

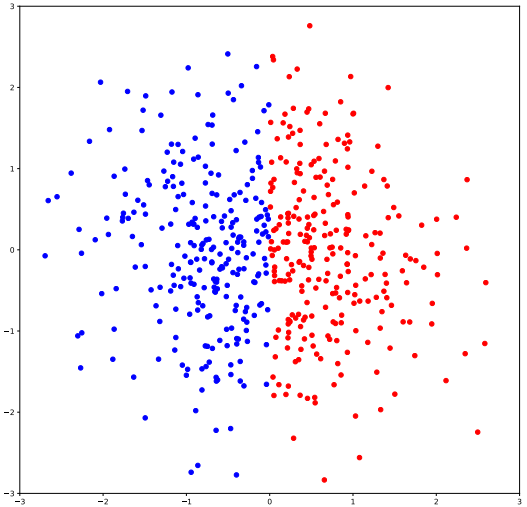


Continuous decision stump I

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

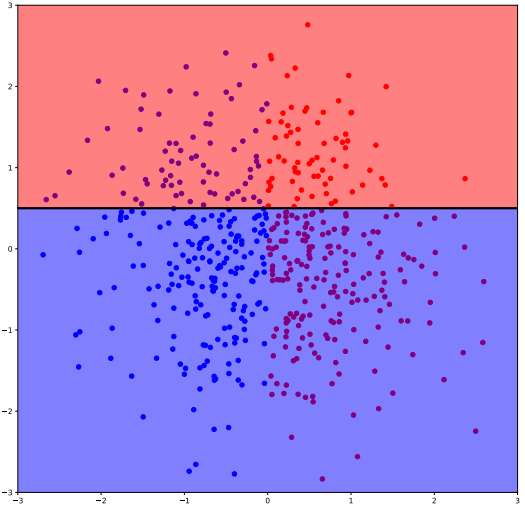
$$f_\theta(x) = \begin{cases} 0 & x_{\texttt{feature}} < \texttt{split} \\ 1 & x_{\texttt{feature}} \geq \texttt{split} \end{cases}$$

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



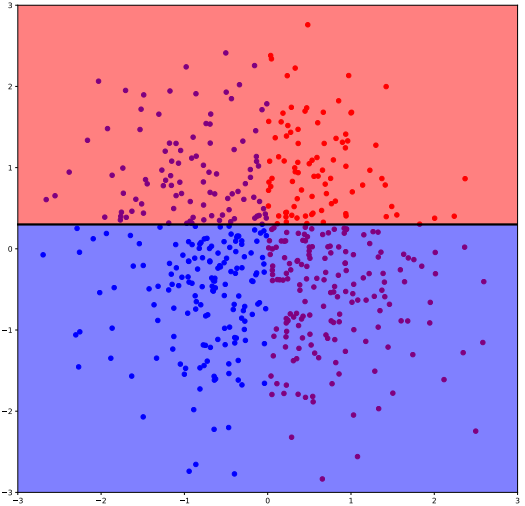
Continuous decision stump II

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



Continuous decision stump II

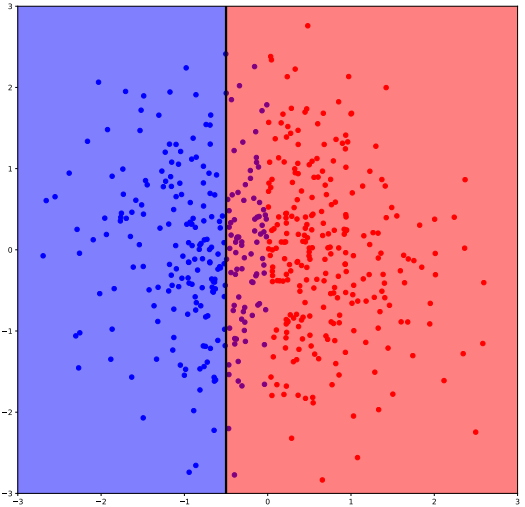
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$



UNIVERSITY OF BATH

Continuous decision stump II

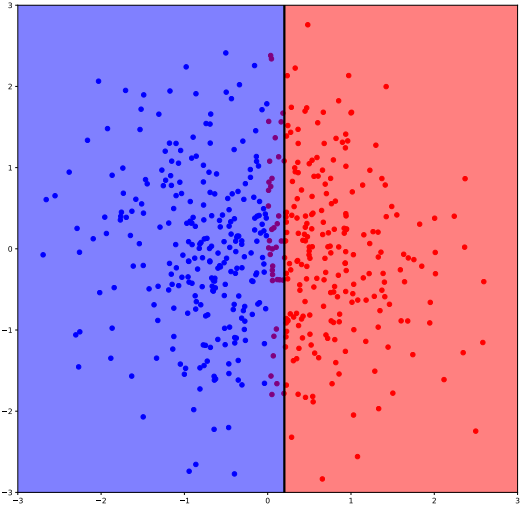
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$



UNIVERSITY OF BATH

Continuous decision stump II

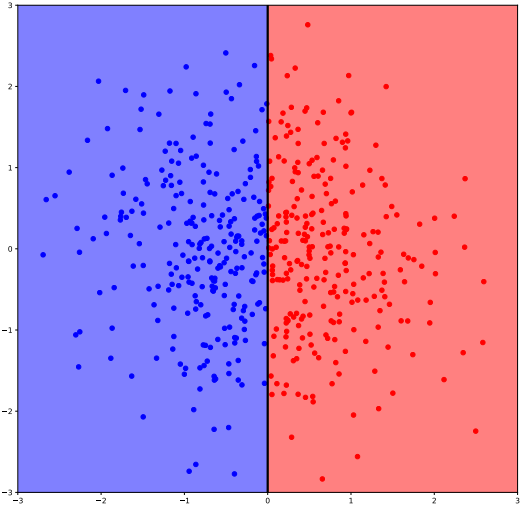
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$
- $\texttt{feature} = x, \texttt{split} = 0.2$
  accuracy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



UNIVERSITY OF BATH

Continuous decision stump II

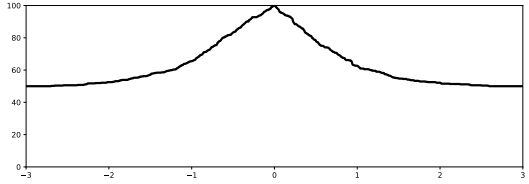
- $\texttt{feature} = y, \texttt{split} = 0.5$
  accuracy $= 50.8\%$
  (accuracy = one minus average 0–1 loss)
- $\texttt{feature} = y, \texttt{split} = 0.3$
  accuracy $= 52.1\%$
- $\texttt{feature} = x, \texttt{split} = -0.5$
  accuracy $= 83.2\%$
- $\texttt{feature} = x, \texttt{split} = 0.2$
  accuracy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



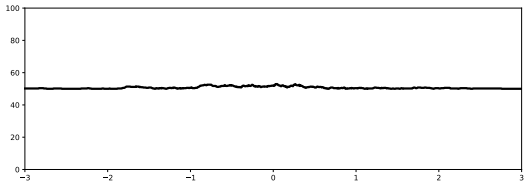
Continuous decision stump III

- 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:

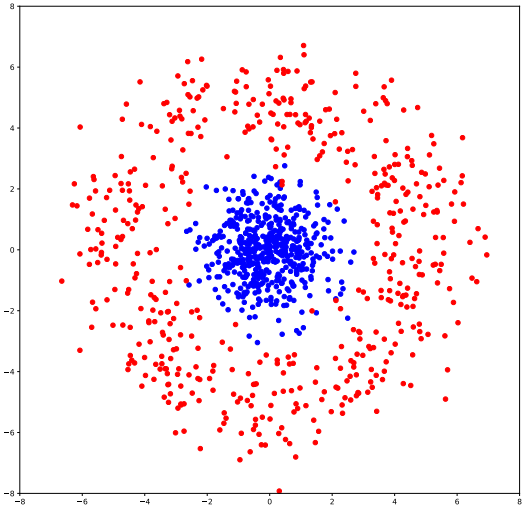


y:



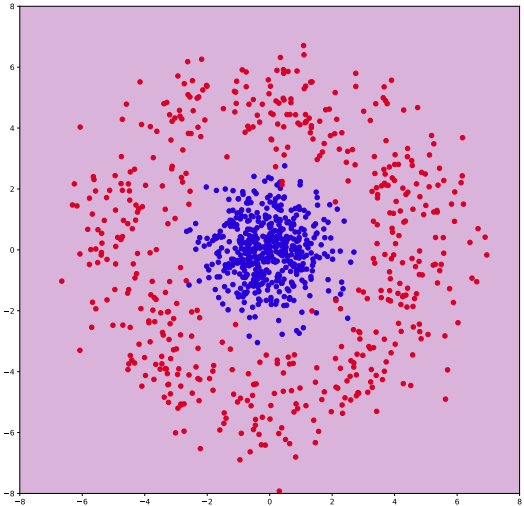
Continuous decision stump IV

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



Decision trees I

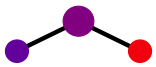
- What if we did this recursively?

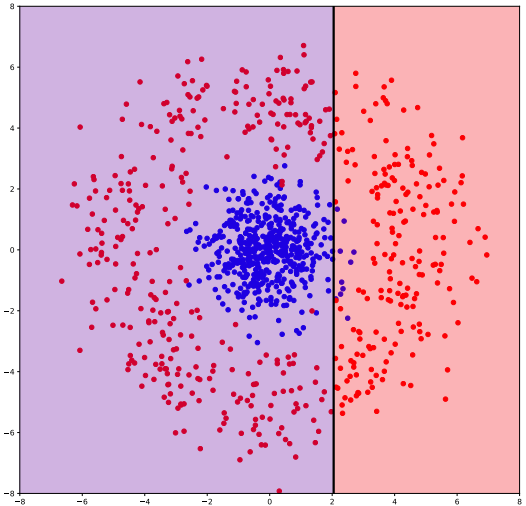


Decision trees I

- What if we did this recursively?

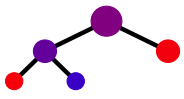
- Background colour – shaded with ratio of red/blue points.
- First split – as before.
- Can be represented as a tree.

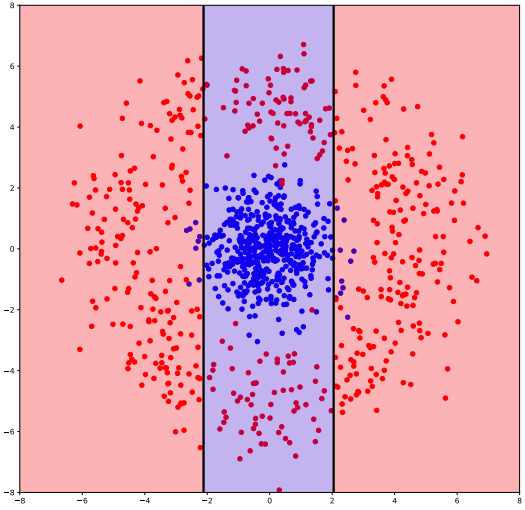




Decision trees I

- 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.



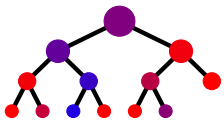


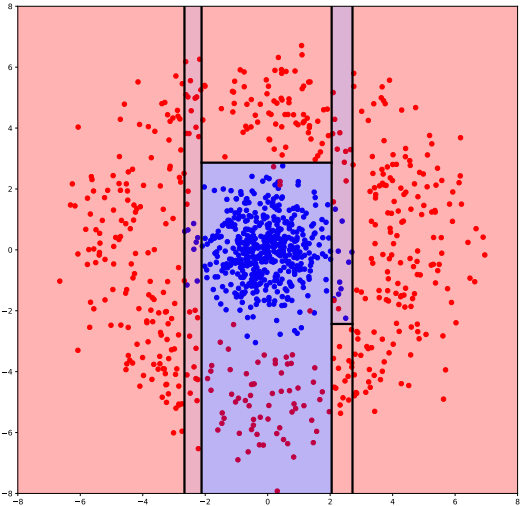
Decision trees I

- 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.

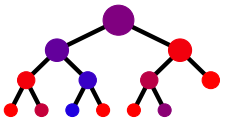
- Have split left half of first split again.
- Jumping ahead...

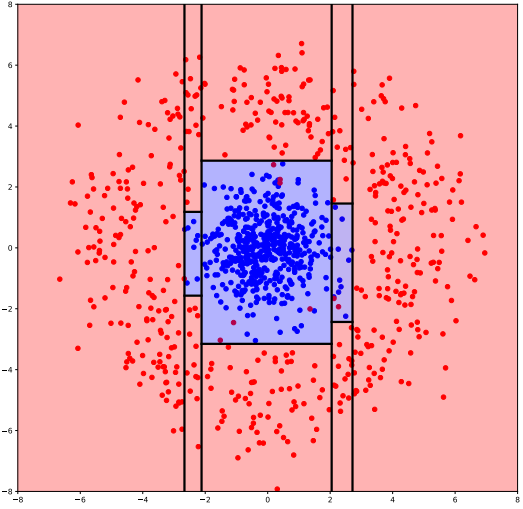




Decision trees I

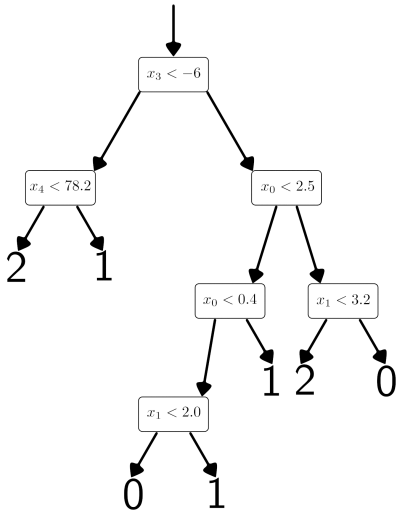
- What if we did this recursively?
- Background colour – shaded with ratio of red/blue points.
- First split – as before.
- Can be represented as a tree.
- Have split left half of first split again.
- Jumping ahead...
- **Decision tree = recursive splitting**





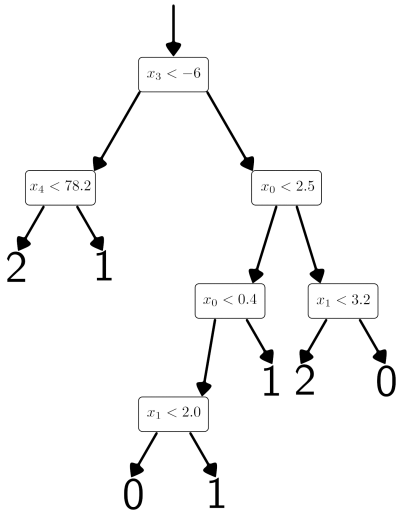
Decision trees II

- The function parameter, θ, is a binary tree



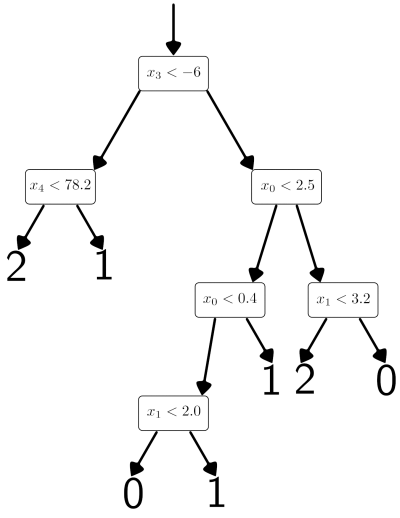
Decision trees II

- 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

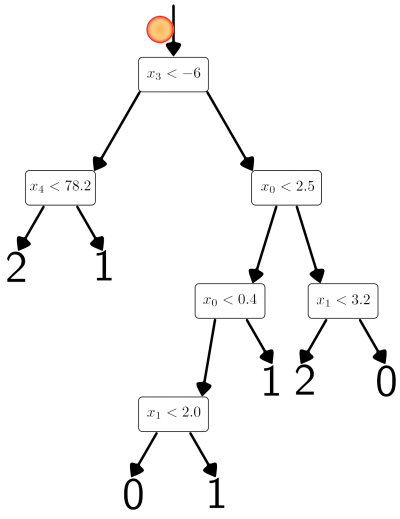
To evaluate the function, $f_\theta(x)$:



Decision trees III

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

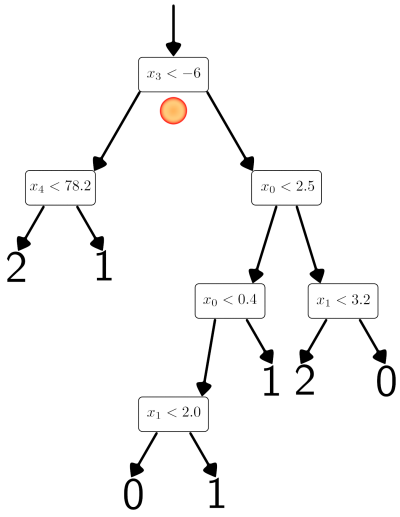
- Start at the top



Decision trees III

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

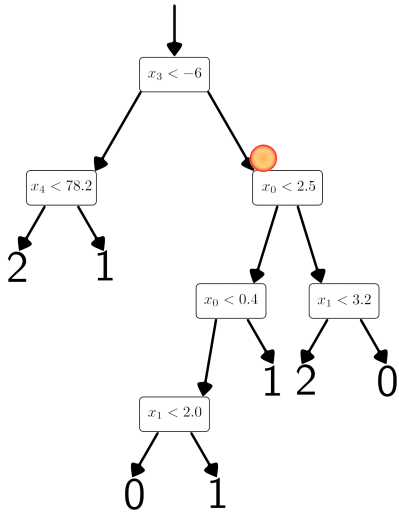
- Start at the top
- Move to the first split



Decision trees III

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

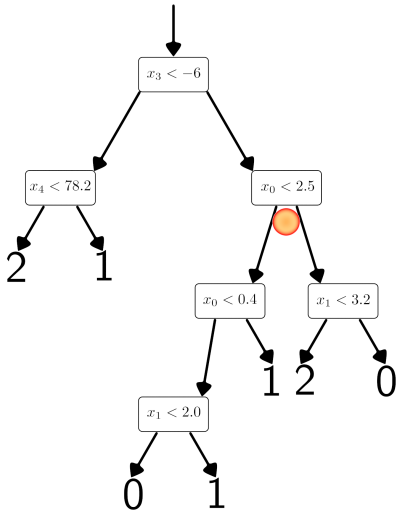
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test



Decision trees III

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

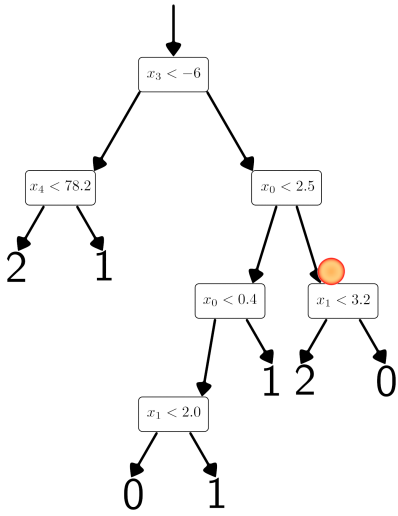
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...



Decision trees III

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

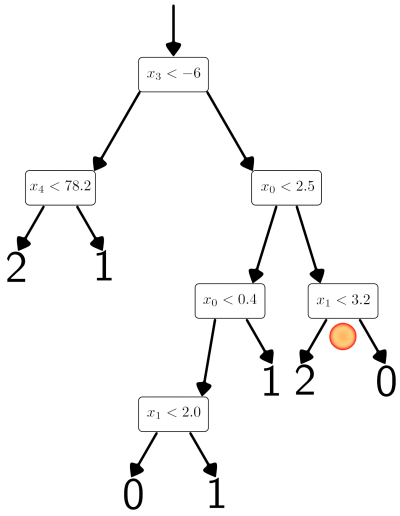
- Start at the top
- Move to the first split
- Head **left** or **right** depending on test
- Move to next split, and so on...



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...

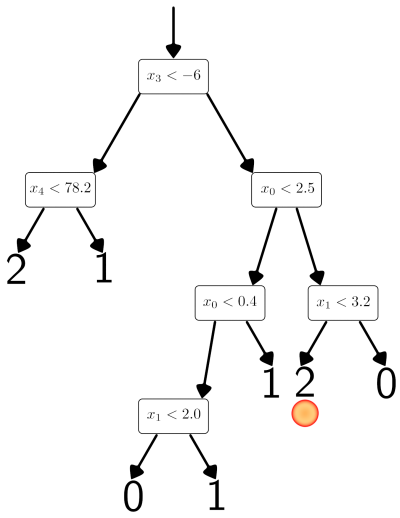


Decision trees III

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.



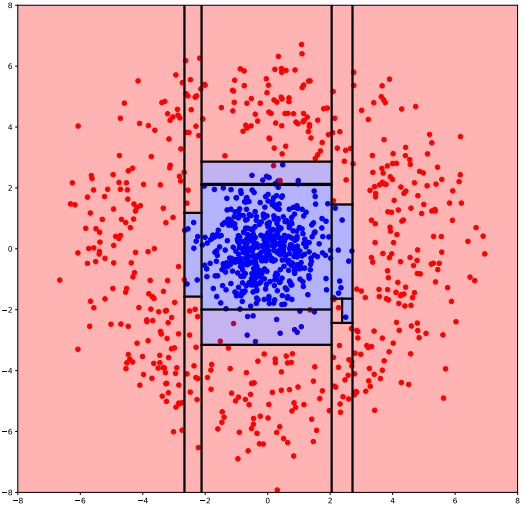
Decision trees IV

- 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.

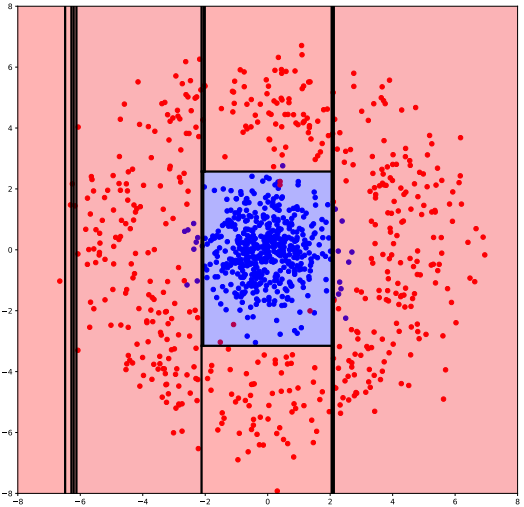
Decision trees V

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



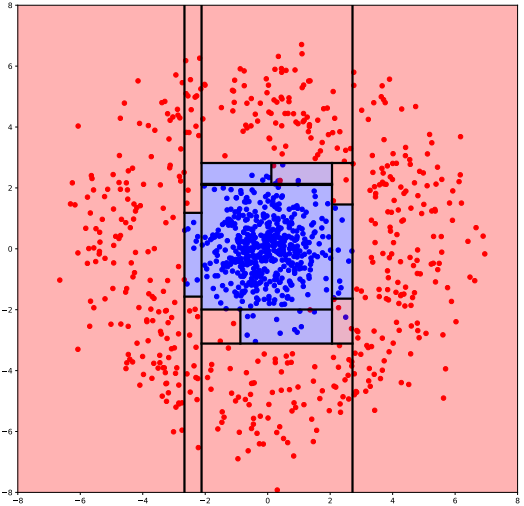
Decision trees V

- 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

- The 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

- The 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

- The 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 assigns value to partial success (unlike 0–1 loss).

Weighting the split

- You can calculate $G(p^{(\texttt{left})})$ and $G(p^{(\texttt{right})})$ for the 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 the 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

- Information gain is, 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

- Information gain is, conceptually, how much you learn from traversing a split.
- Entropy:

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

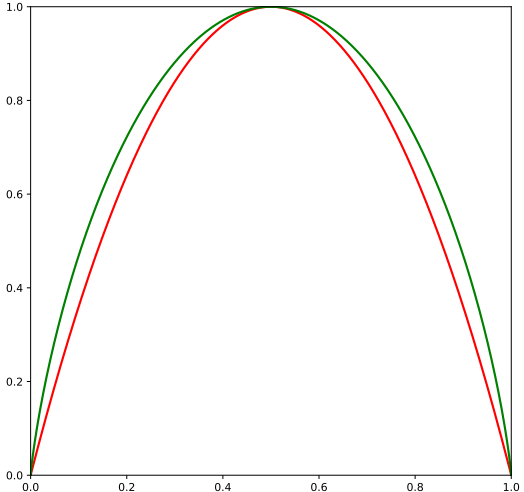
  - Really important – internet wouldn't work without it!
  - Measures how many bits are required on average to encode data with a given probability distribution (bits assumes $\log_2$, if $\log_e$ the unit is nats).

- Information gain is defined as the number of bits/nats obtained from traversing the split:

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

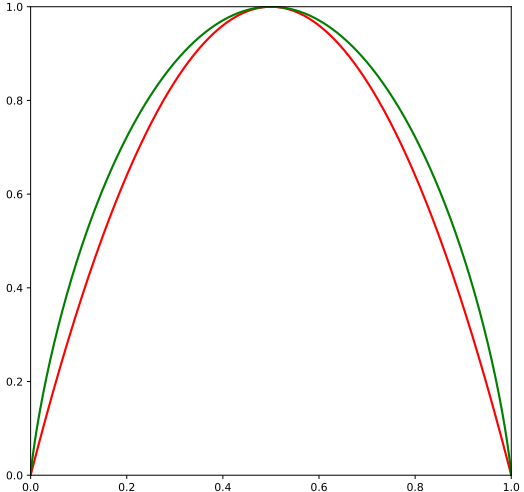
Which loss function?

- 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 is a lot faster to compute (no log), so should be your default.

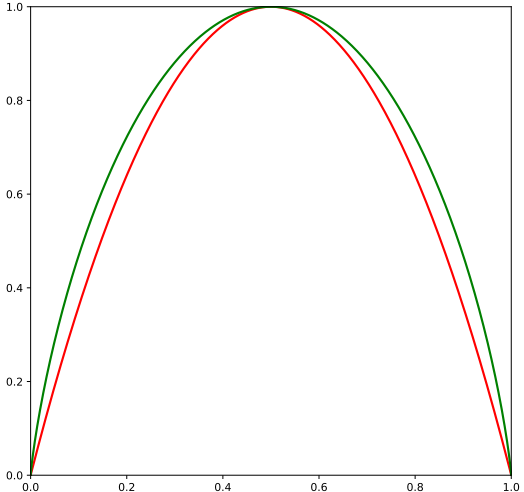


Which loss function?

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

- Gini is a lot faster to compute (no log), so should be your default.

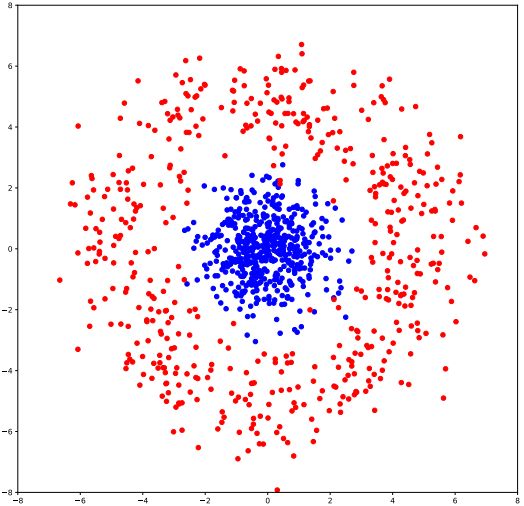
- Information gain can be better for some problems, and works in situations where Gini cannot, e.g. regression.
- Information gain has better theory!



UNIVERSITY OF BATH

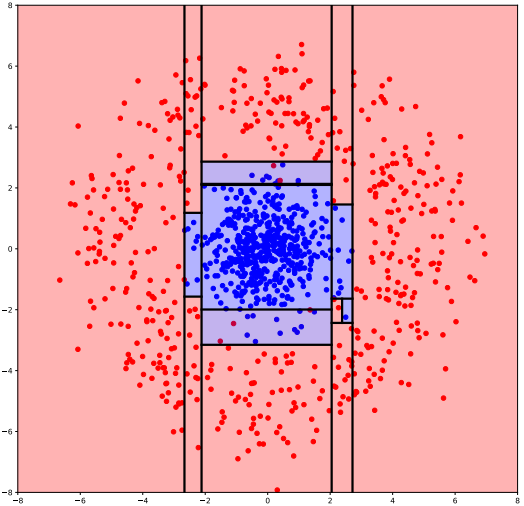
Overfitting

- The boundary is clearly a circle...



Overfitting

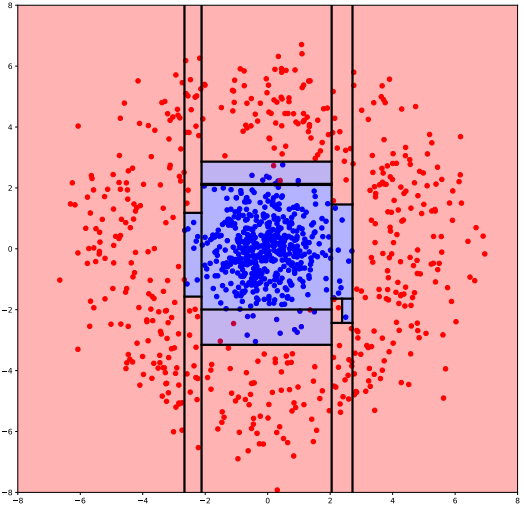
- The boundary is clearly a circle...
- But it does this...



Overfitting

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

- This is called **overfitting**.
  It is modelling the **noise**, not the **signal**.

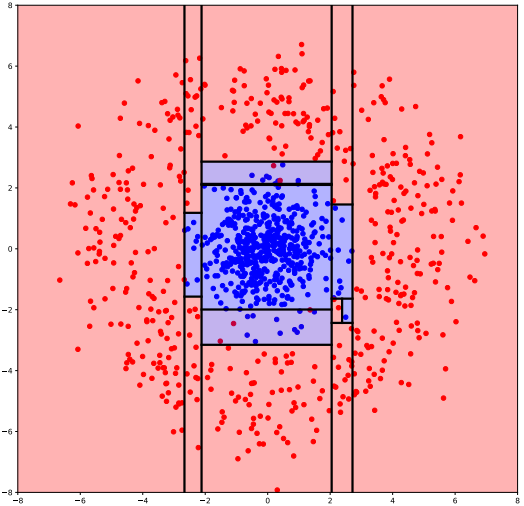


Overfitting

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

- This is called **overfitting**.
  It is modelling the **noise**, not the **signal**.

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



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!

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.
  - Capable of probabilistic answers.
  - 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)