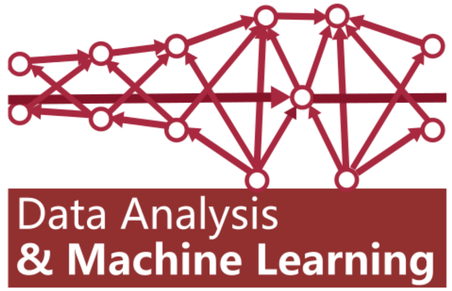


Data Analysis and Machine Learning 4 (DAML)

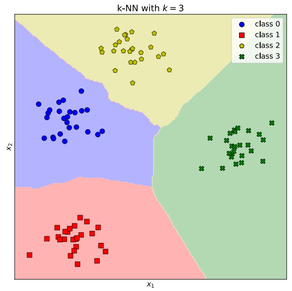
Week 8: Classification and Regression Trees

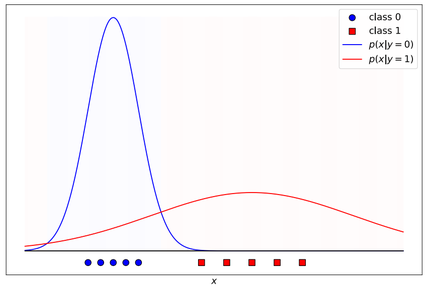
Elliot J. Crowley, 11th March 2024

THE UNIVERSITY *of* EDINBURGH

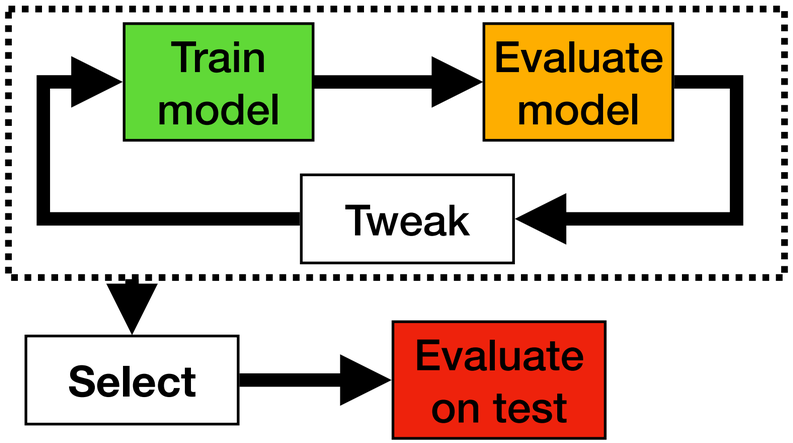
Recap

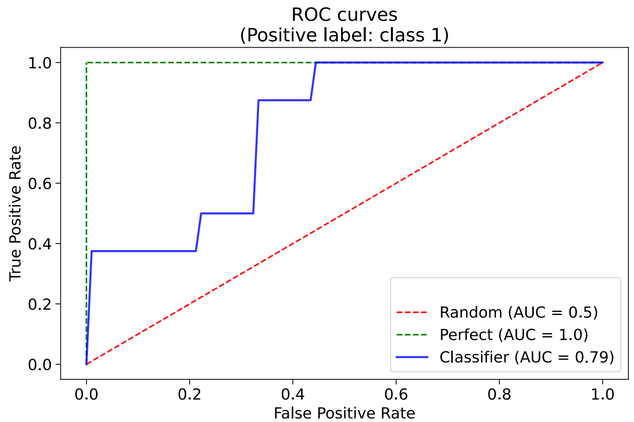
- We looked at different models for classification





- We considered model selection and a typical ML workflow
- We looked at different ways to evaluate classifiers





Non-parametric models

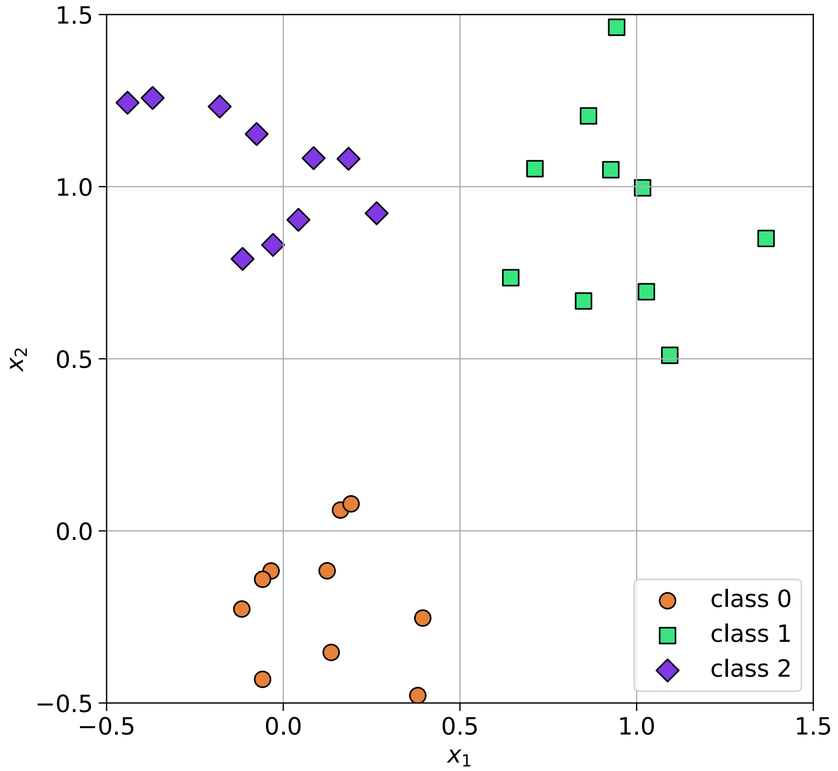
- Parametric models are represented by a function with a fixed number of parameters i.e. they have a fixed capacity
- Non-parametric models are not!
- The capacity of a non-parametric can scale with the number of data points
- In this lecture we will look at classification and regression trees (CARTs) which are non-parametric models
- These are also known as **decision trees**

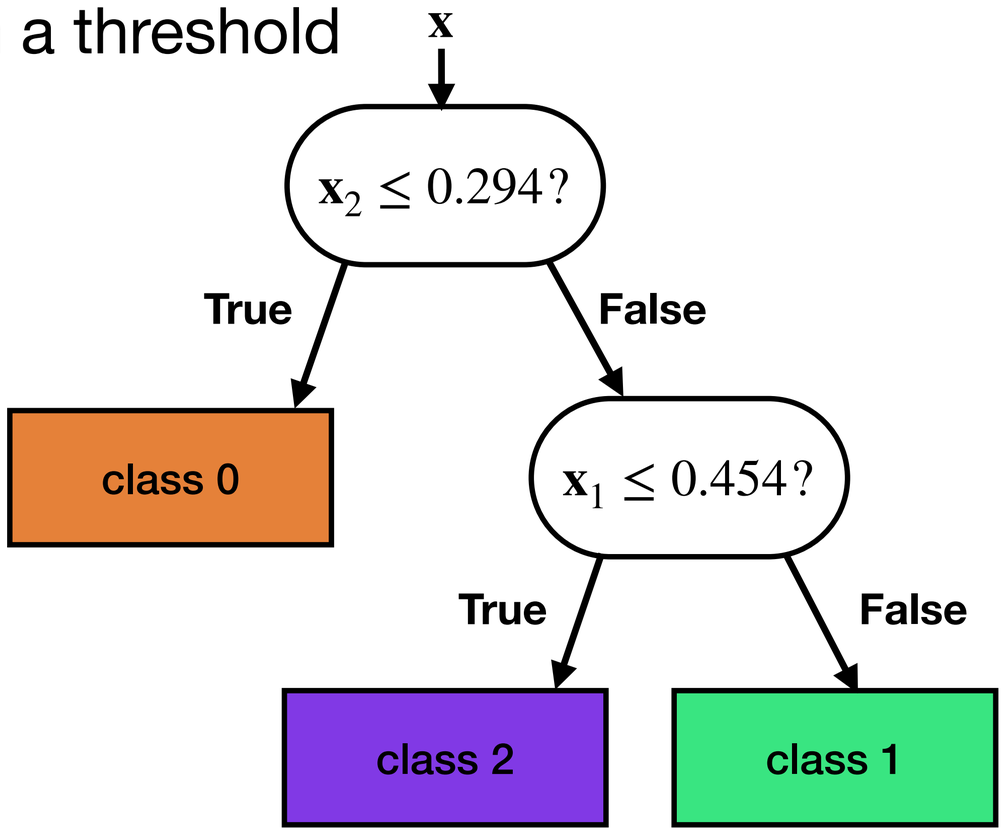
Credit to Joe Mellor for the description

Classification trees

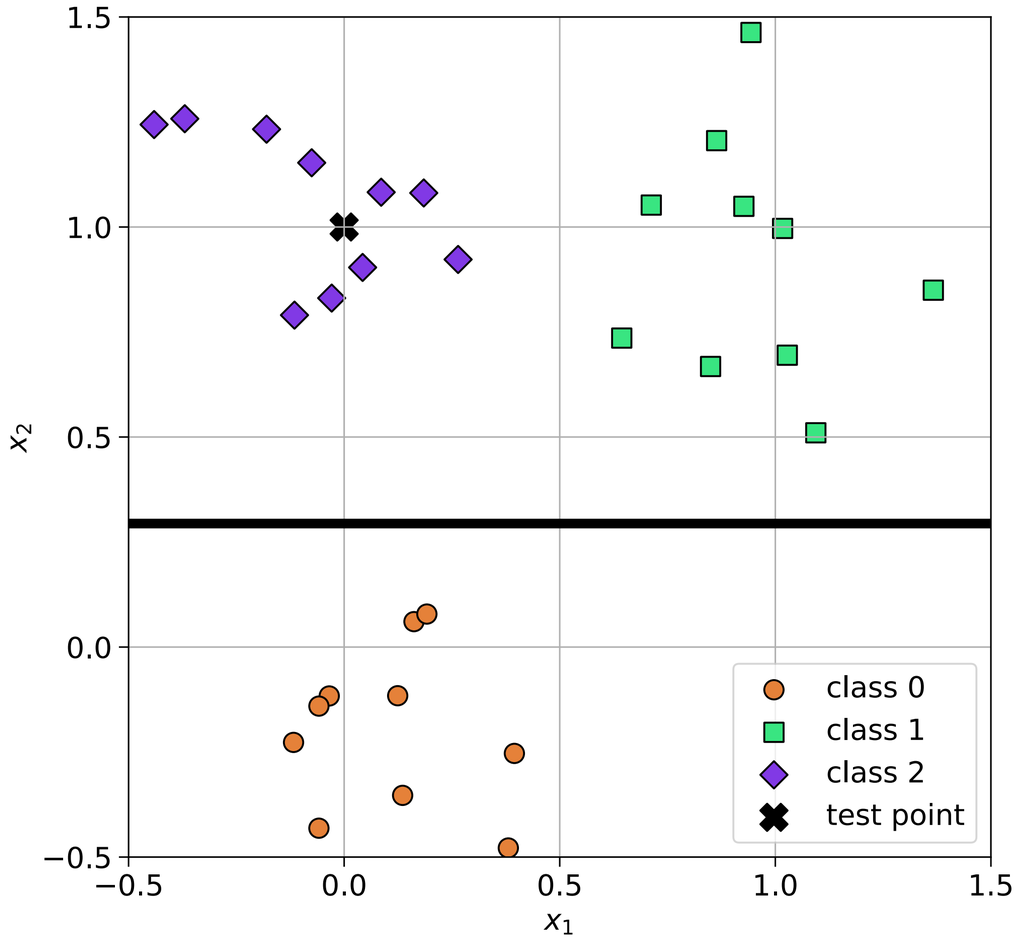
Classification trees

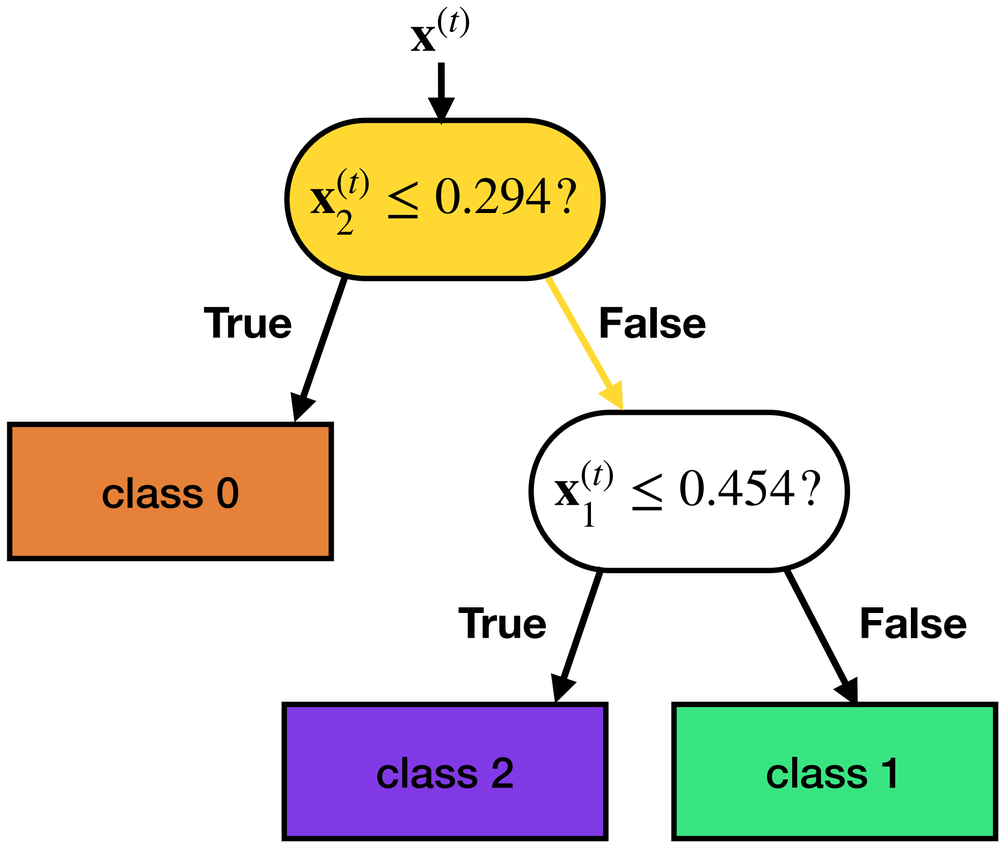
- An upside-down tree consisting of nodes — leaf nodes are at the bottom
- Each node splits incoming data based on a threshold for a single feature



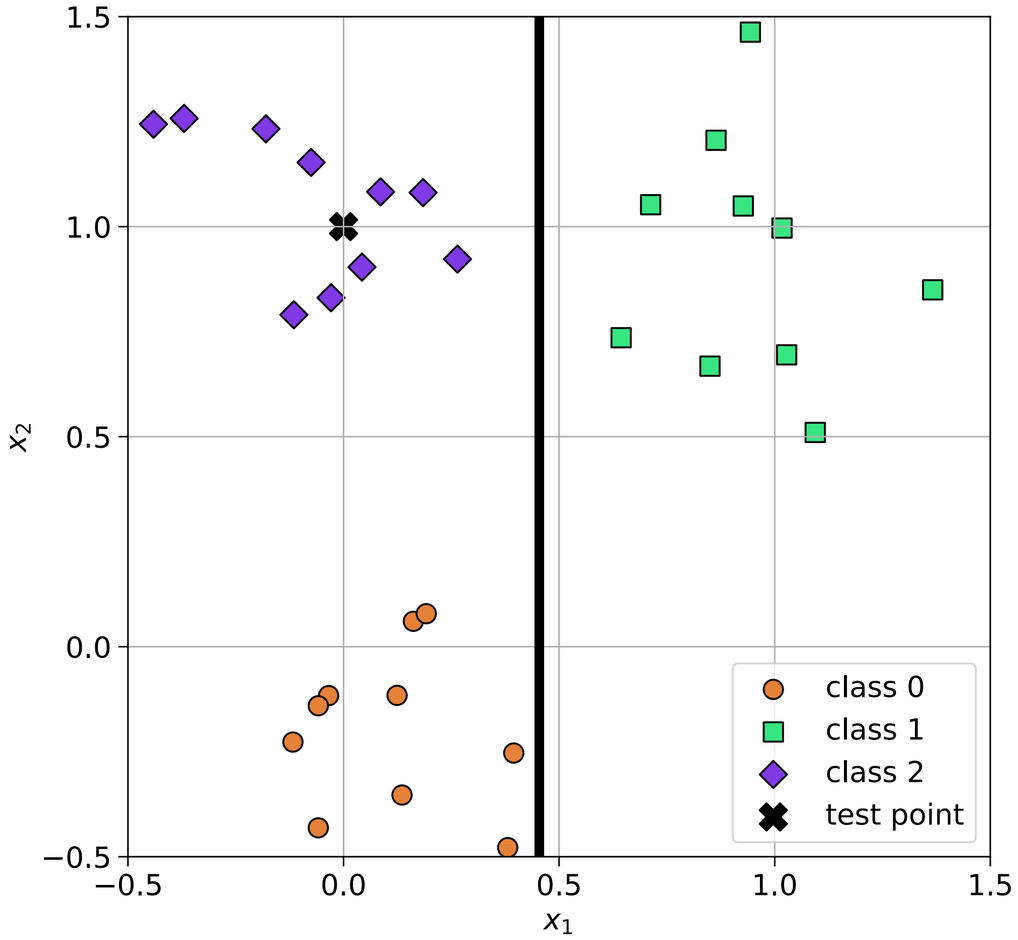


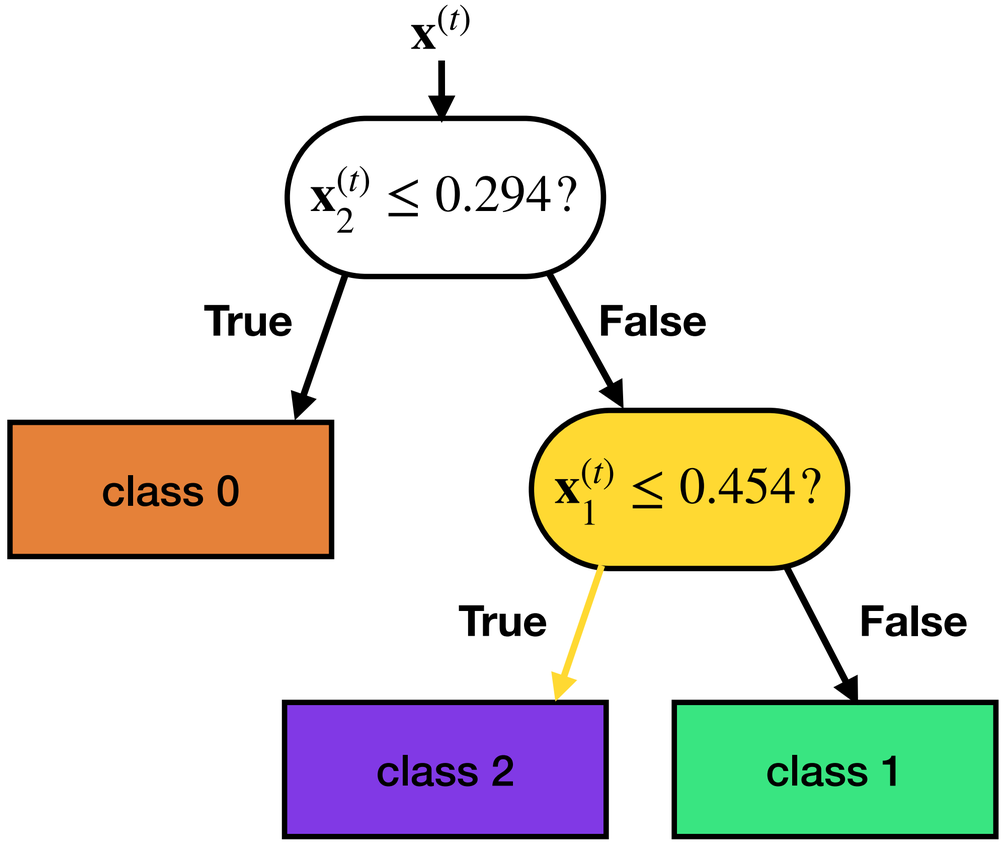
Classifying a point $\mathbf{x}^{(t)}$





Classifying a point $\mathbf{x}^{(t)}$

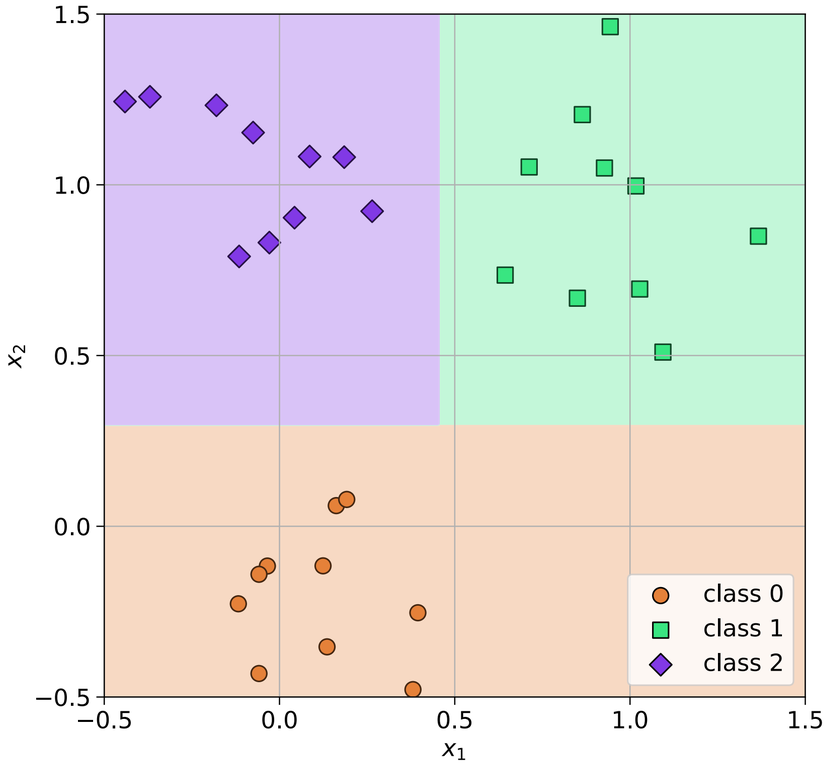


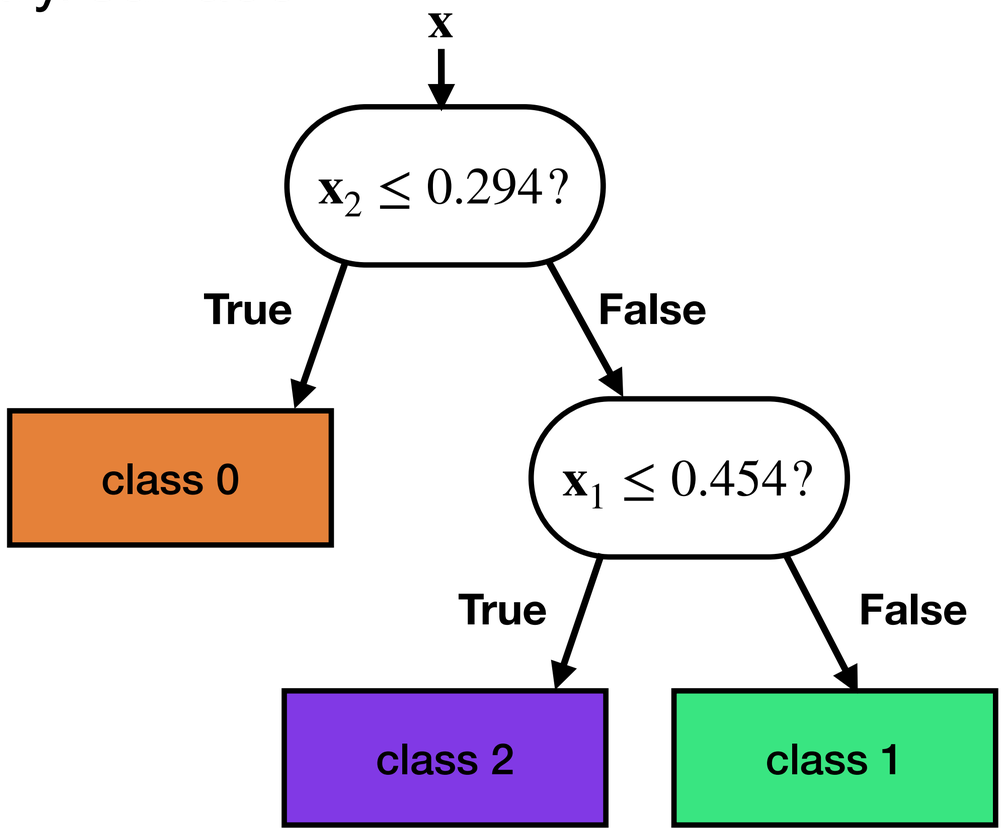


$\mathbf{x}^{(t)}$ is classified as class 2

What is the classification tree doing?

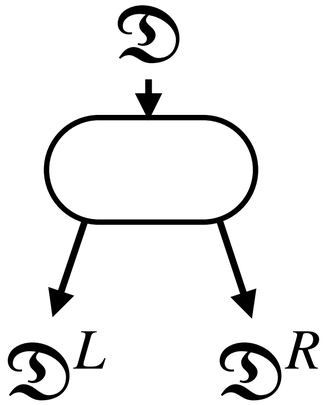
- It is slicing up the feature space using straight lines (/hyperplanes)
- This gives a non-linear decision boundary/surface

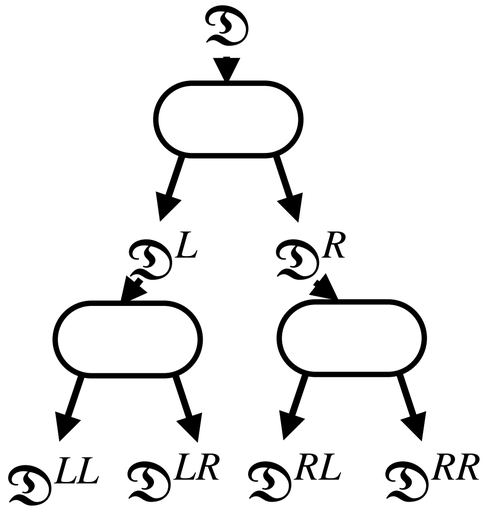


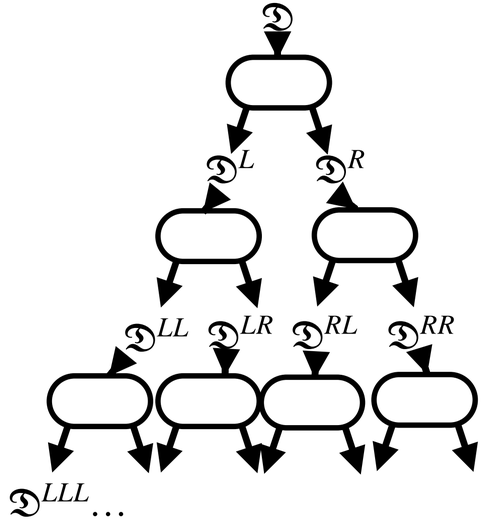


Classification tree learning

- We have a training set $\mathfrak{D} = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N\}$ where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \mathbb{Z}^+_{<K}$
- We start with a single node and learn the best way to split the training set into a left and right split
- Then for each split, we create a node and learn the best way to split it further and so on. This is a **greedy** algorithm!







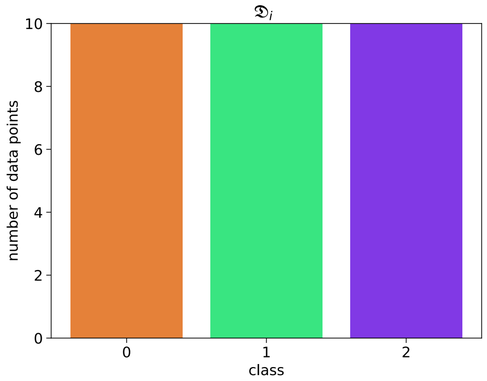
Finding the best split at a node

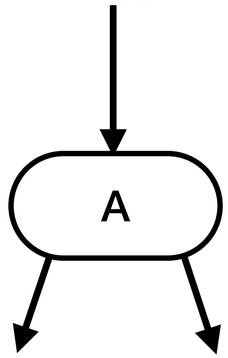
- Consider a subset of data arriving at some node i: $\mathfrak{D}_i = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N_i\}$
- The node will split this into $\mathfrak{D}_i^L = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N_i : x_j^{(n)} \leq t\}$ and $\mathfrak{D}_i^R = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N_i : x_j^{(n)} > t\}$
- We want to find the feature j and threshold t that gives us the best split
- If we had some c that tells us how **bad** a data split is then we could solve:

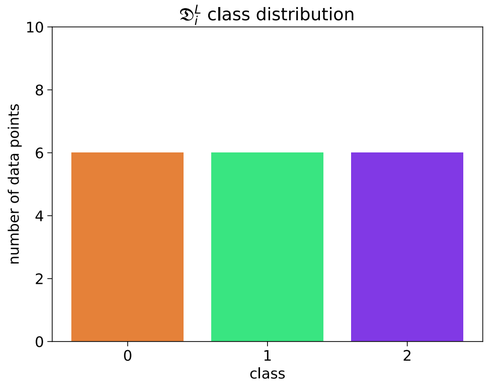
$$\underset{j,t}{\text{minimise}} \ \frac{|\mathfrak{D}_i^L(j,t)|}{|\mathfrak{D}_i|} c(D_i^L(j,t)) + \frac{|\mathfrak{D}_i^R(j,t)|}{|\mathfrak{D}_i|} c(D_i^R(j,t))$$

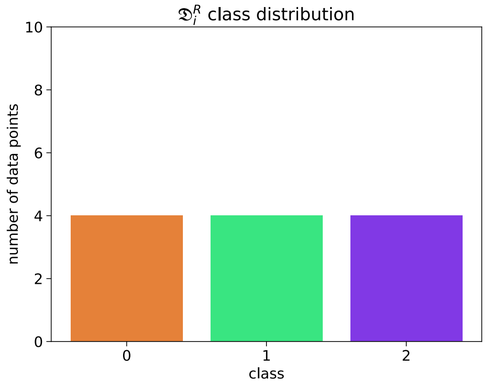
Formulation based on Murphy's book

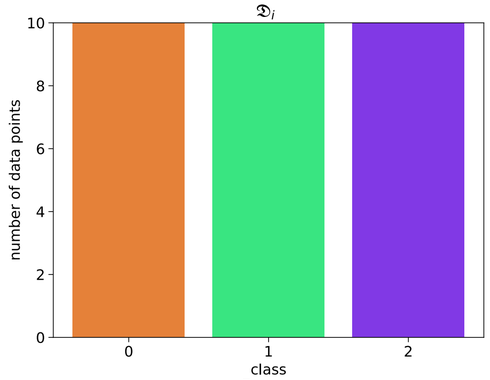
What makes a good split for classification?

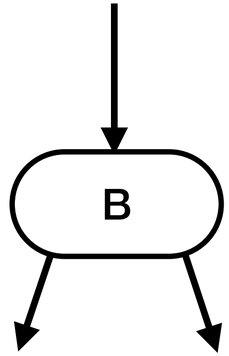


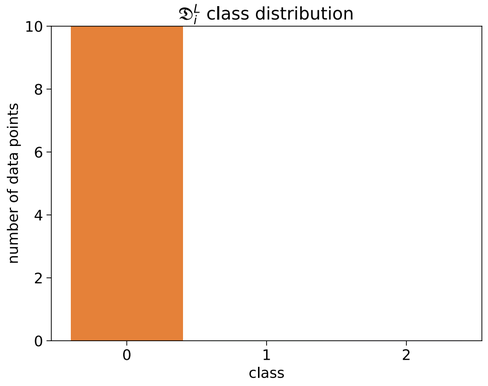


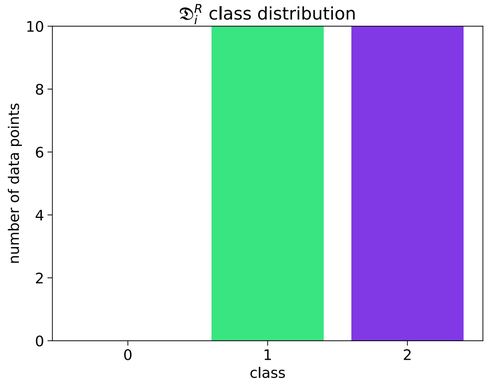








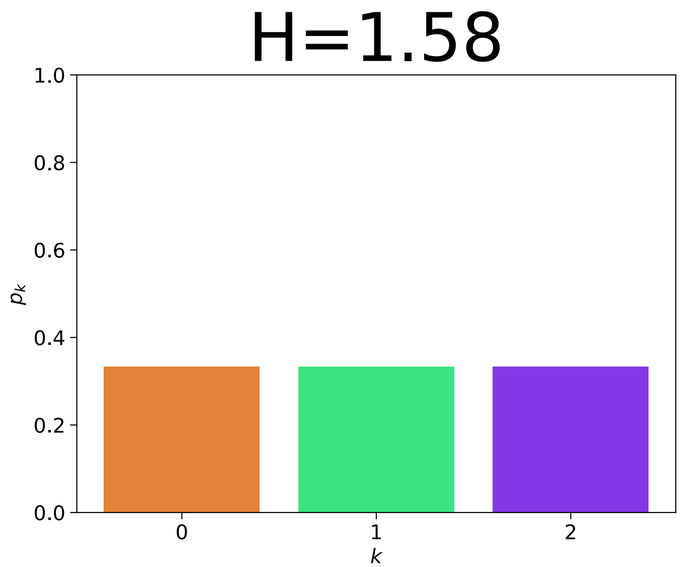


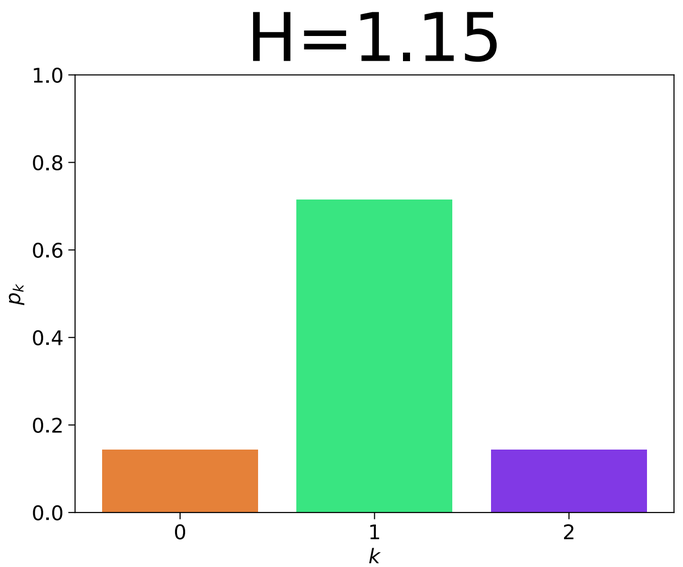


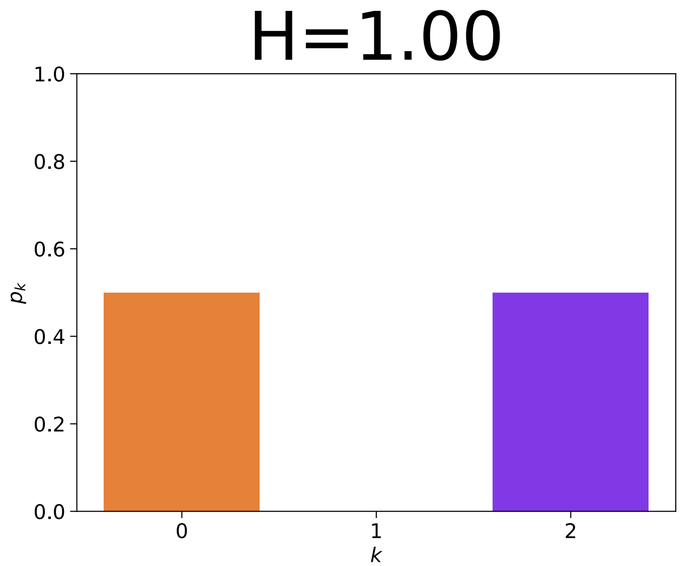
Measuring the quality of a split

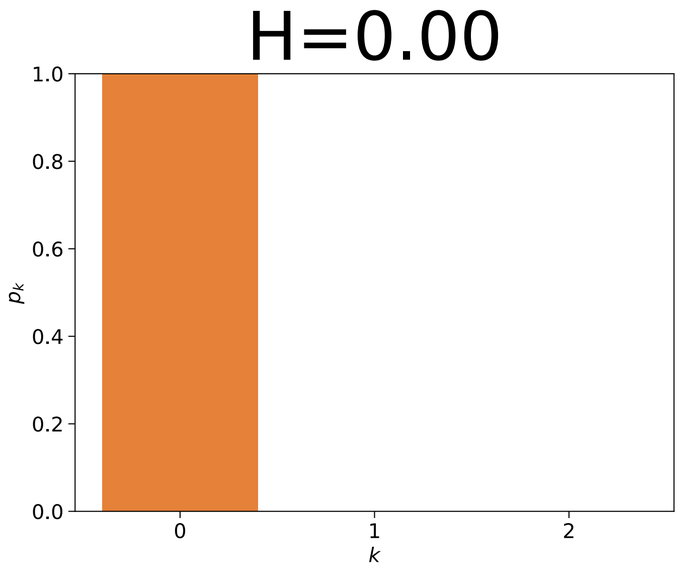
- A good split will separate examples from different classes
- This will make the resulting class distributions **non-uniform**
- Entropy provides a measure of the uniformity of a probability distribution
- For a split, we can divide the number of data points in a class by the total number of points in the split to get empirical probabilities $p_0, p_1, \ldots, p_{K-1}$
- Entropy can then be computed as $H = -\sum_k p_k \log_2 p_k$
- We can use this as c to measure how **bad** a split is

Examples of different entropies





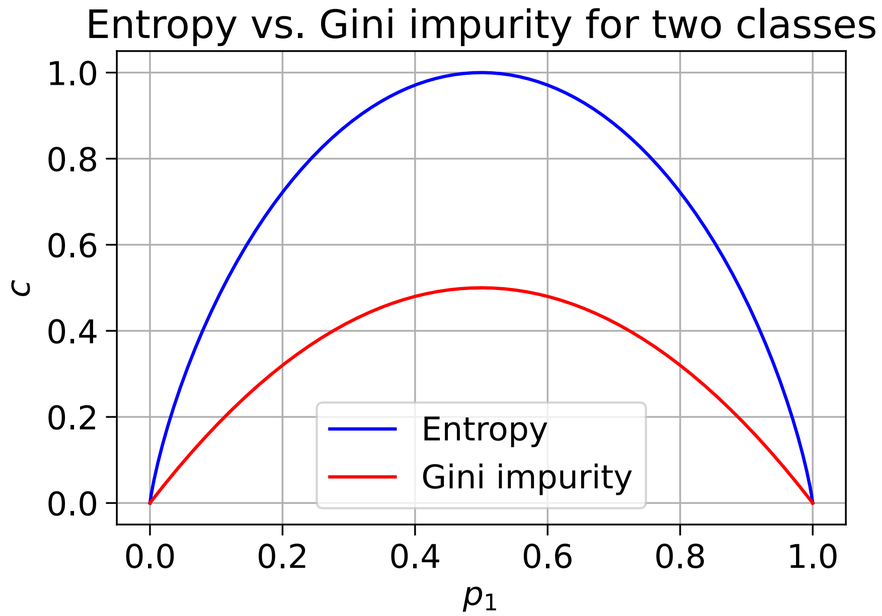




We can also use Gini impurity as c

Gini impurity is the probability of incorrectly classifying a new data point labelled according to the class distribution of that split

$$G = \sum_k p_k(1 - p_k)$$

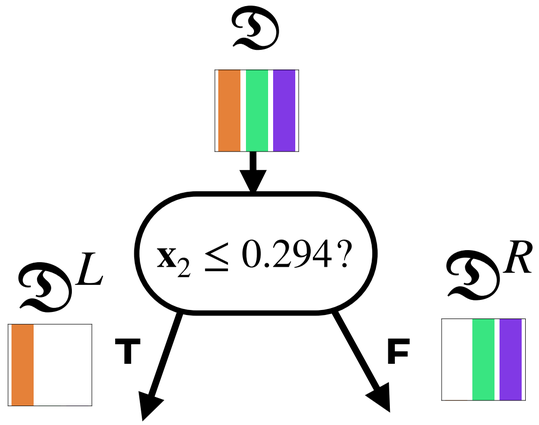


Shapes are very similar

Choice has minimal effect on performance

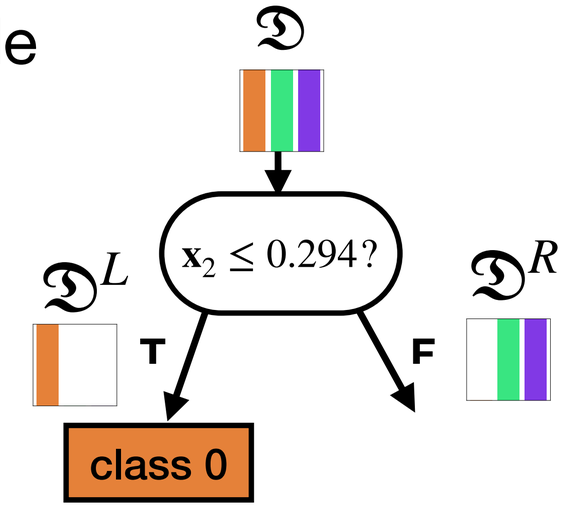
Learning example: Splitting $\mathfrak{D}$

- We will show how the tree from the start was grown using Gini impurity as c
- Let's create a node that best splits $\mathfrak{D}$ into $\mathfrak{D}^L$ and $\mathfrak{D}^R$
- We solve $\underset{j,t}{\text{minimise}} \ \frac{|\mathfrak{D}^L|}{|\mathfrak{D}|}c(\mathfrak{D}^L) + \frac{|\mathfrak{D}^R|}{|\mathfrak{D}|}c(\mathfrak{D}^R)$
- This gives us $j = 2$ and $t = 0.294$



Learning example: Splitting $\mathfrak{D}^L$

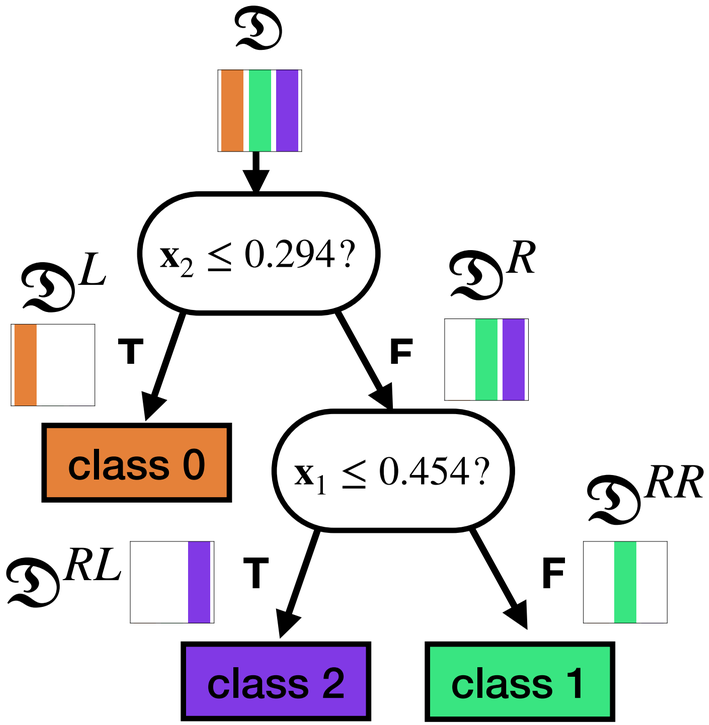
- Now let's create a node that splits $\mathfrak{D}^L$… **or not**
- $\mathfrak{D}^L$ only contains examples from 1 class so we just create a leaf node
- At a leaf node we classify according to the most probable class in the training split at that node
- Here the split contains 10 points from class 0 and 0 points from class 1 or 2
- $p_0 = 1, p_1 = 0, p_2 = 0$ so classify as class 0



This leaf node is **pure** because it only has examples from one class

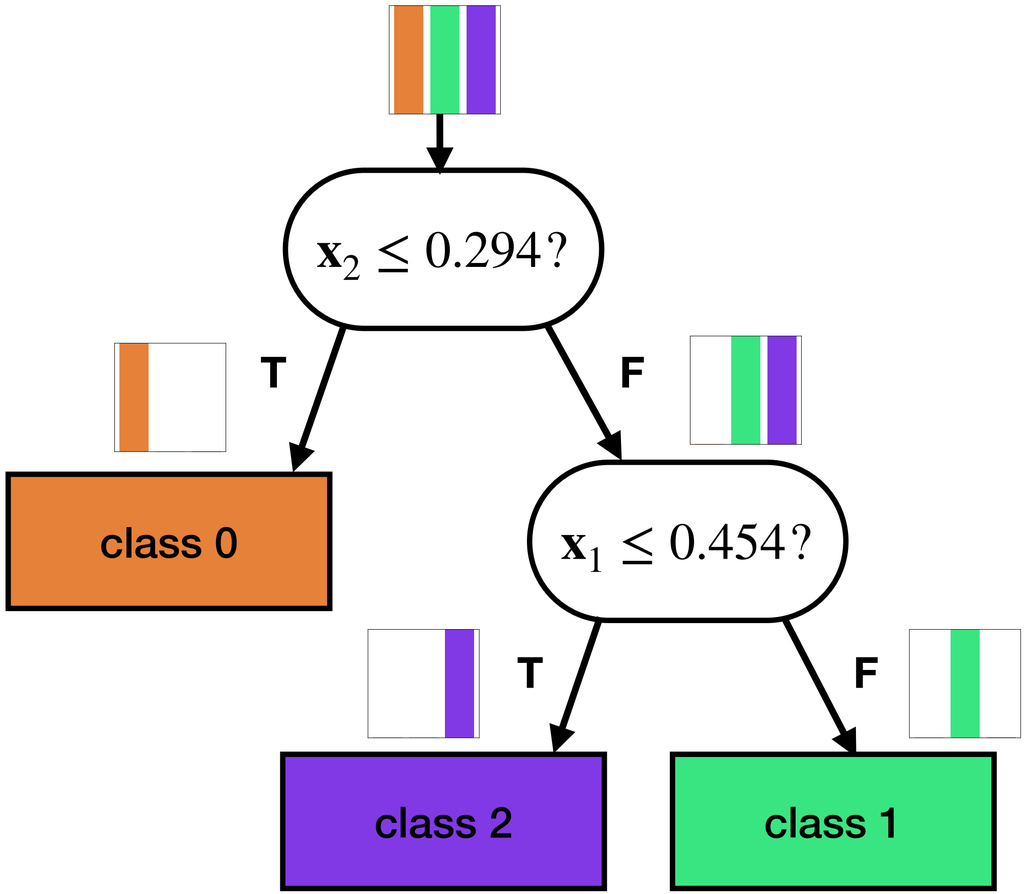
Learning example: Splitting $\mathfrak{D}^R$

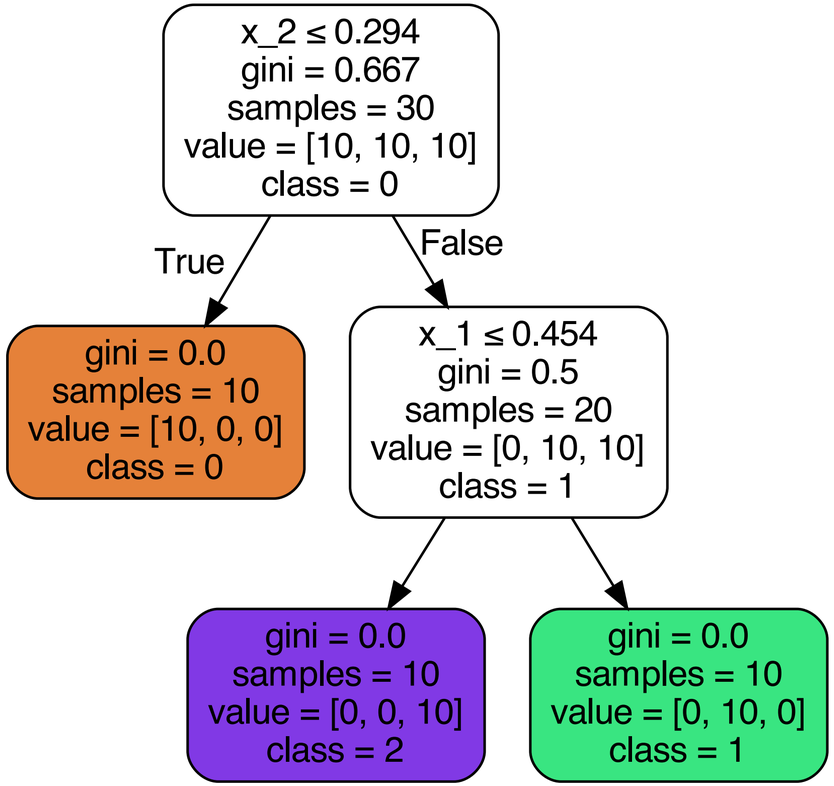
- Now let's create a node that splits $\mathfrak{D}^R$ into $\mathfrak{D}^{RL}$ and $\mathfrak{D}^{RR}$
- We solve $\underset{j,t}{\text{minimise}} \; \frac{|\mathfrak{D}^{RL}|}{|\mathfrak{D}^R|} c(\mathfrak{D}^{RL}) + \frac{|\mathfrak{D}^{RR}|}{|\mathfrak{D}^R|} c(\mathfrak{D}^{RR})$
- This gives us $j = 1$ and $t = 0.454$ which we use to create a node
- Both splits only contain 1 class, so we create **pure** leaf nodes and we're done!



How trees will look in Sklearn

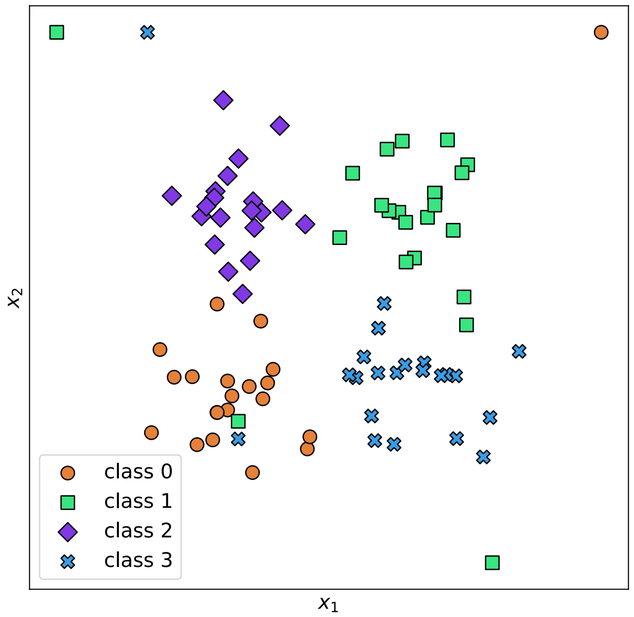
Make sure that you're happy that these trees are the same!

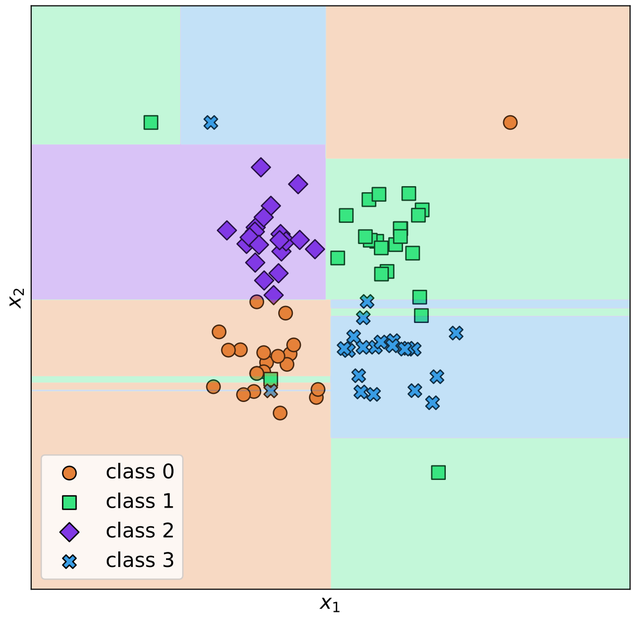


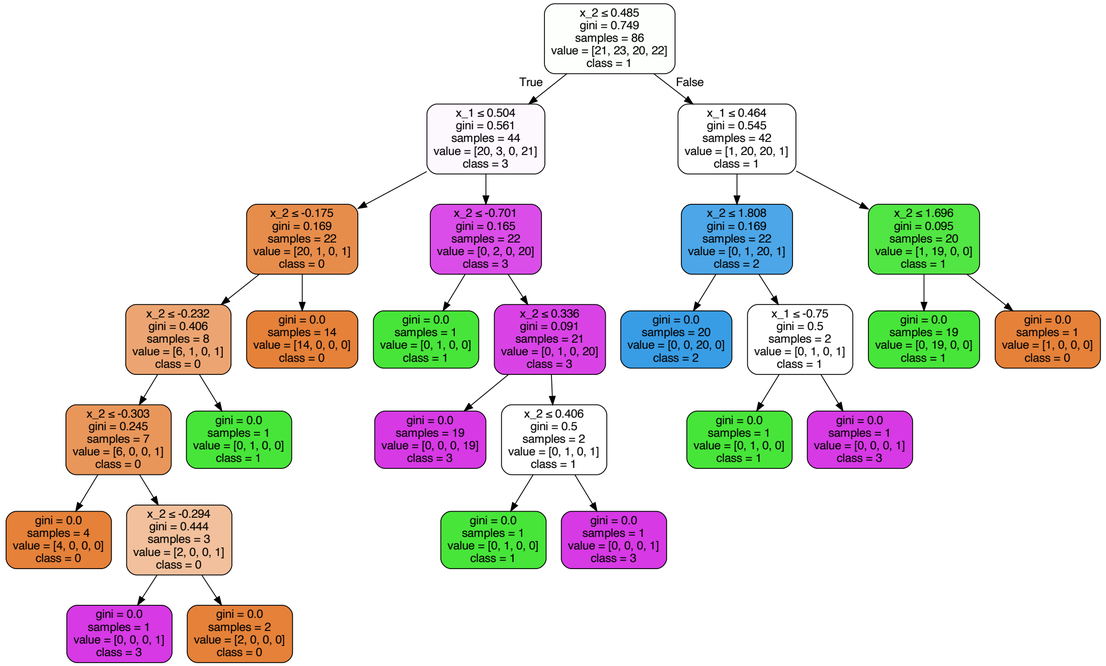


Won’t trees get silly if we keep splitting until nodes are pure?

- Yes. They will massively overfit to the training data

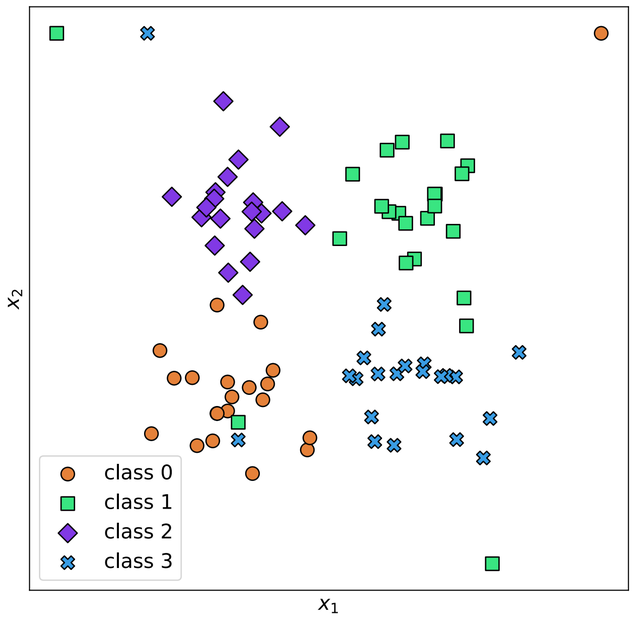


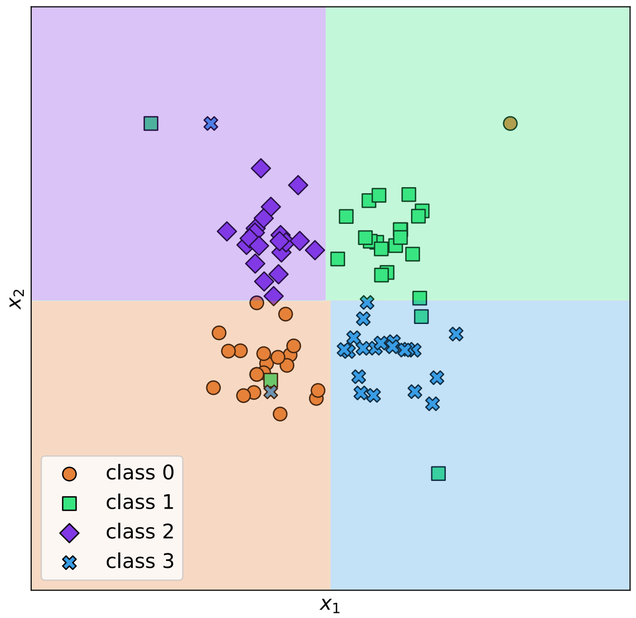


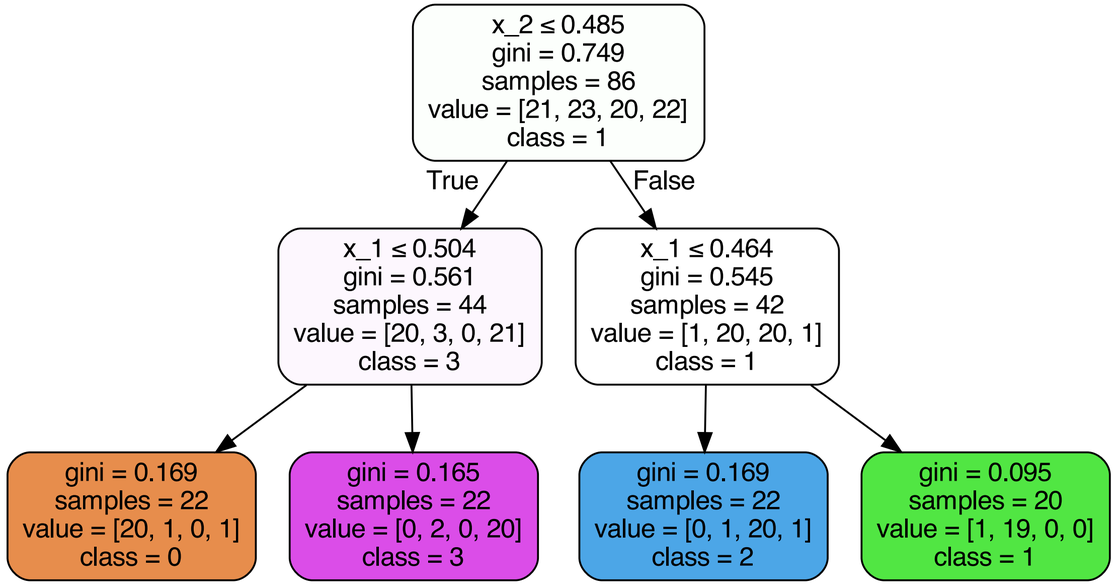


Regularisation in trees

- To combat overfitting we can set a maximum depth for splitting
- Nodes beyond this depth are converted into leaf nodes





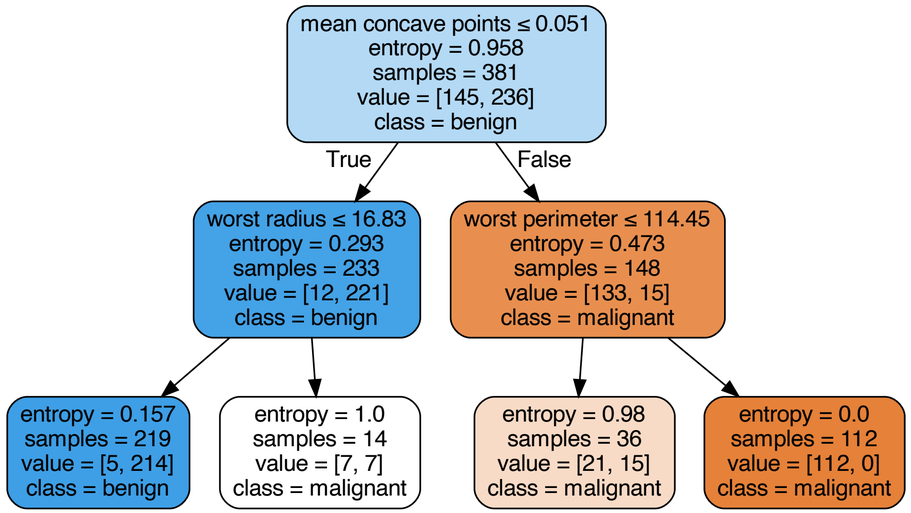


- At a leaf, classify according to most probable class

This tree has a depth of 2 as there are 2 levels of splitting

Malignancy classification with a small tree

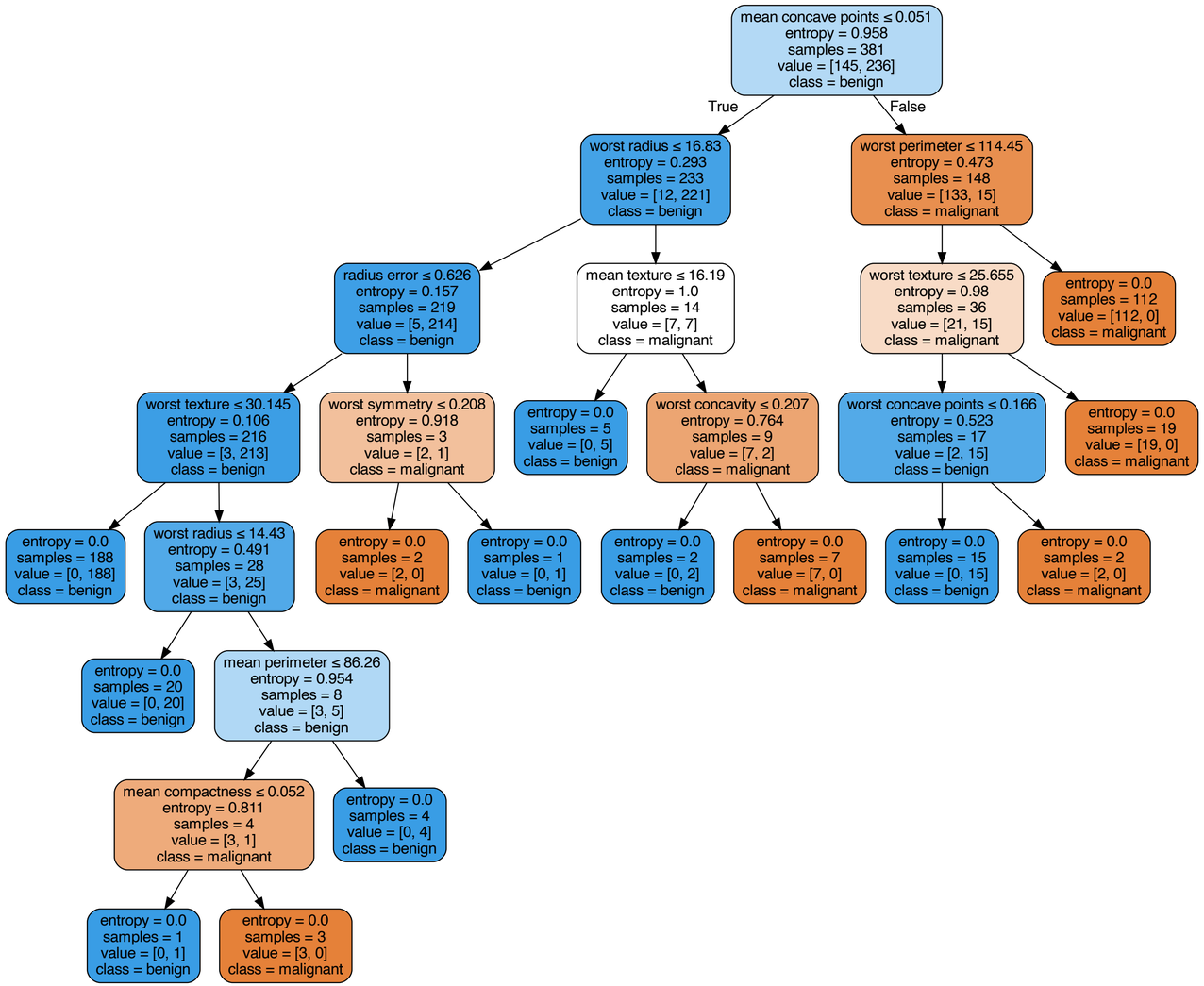
- Breast Cancer Wisconsin dataset has data points $\mathbf{x} \in \mathbb{R}^{30}$ with binary class labels y (malignant / benign)
- Features are measurements from a digitised image of a fine needle aspirate of a breast mass
- Let's fit a classification tree with a max depth of 2



This is interpretable and achieves a validation accuracy of 91.5%

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Malignancy classification with a large tree



Depth 7 tree with a validation accuracy of 95.2%

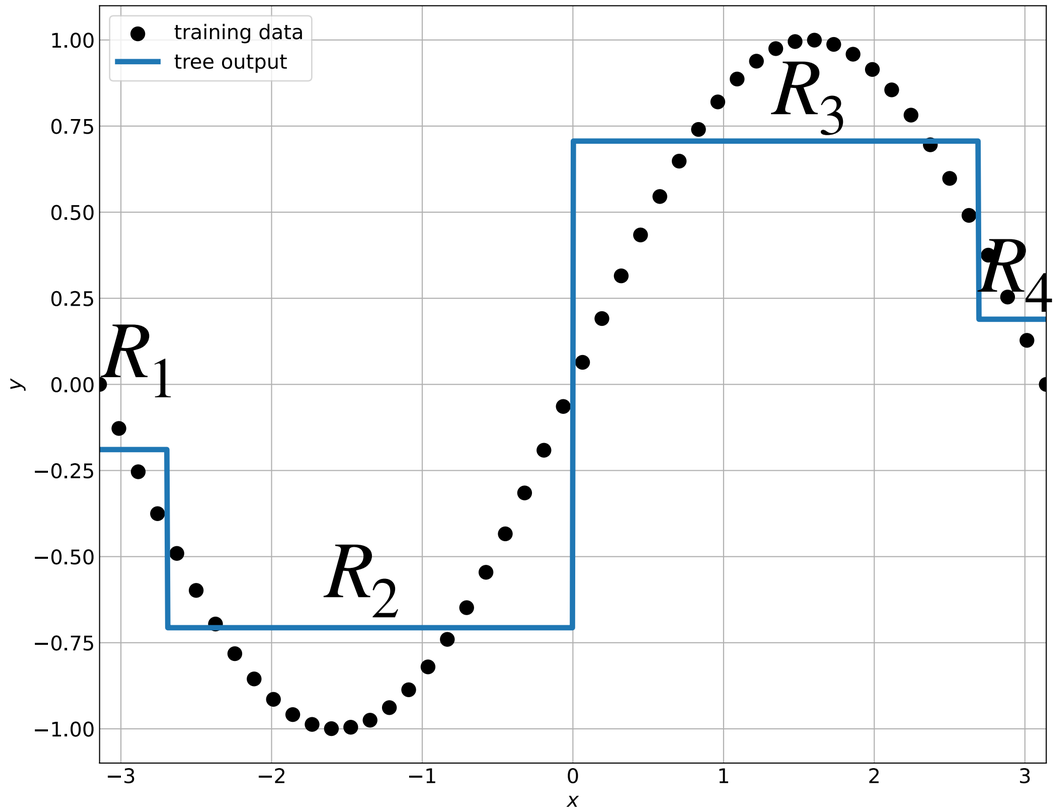
This is hard to interpret

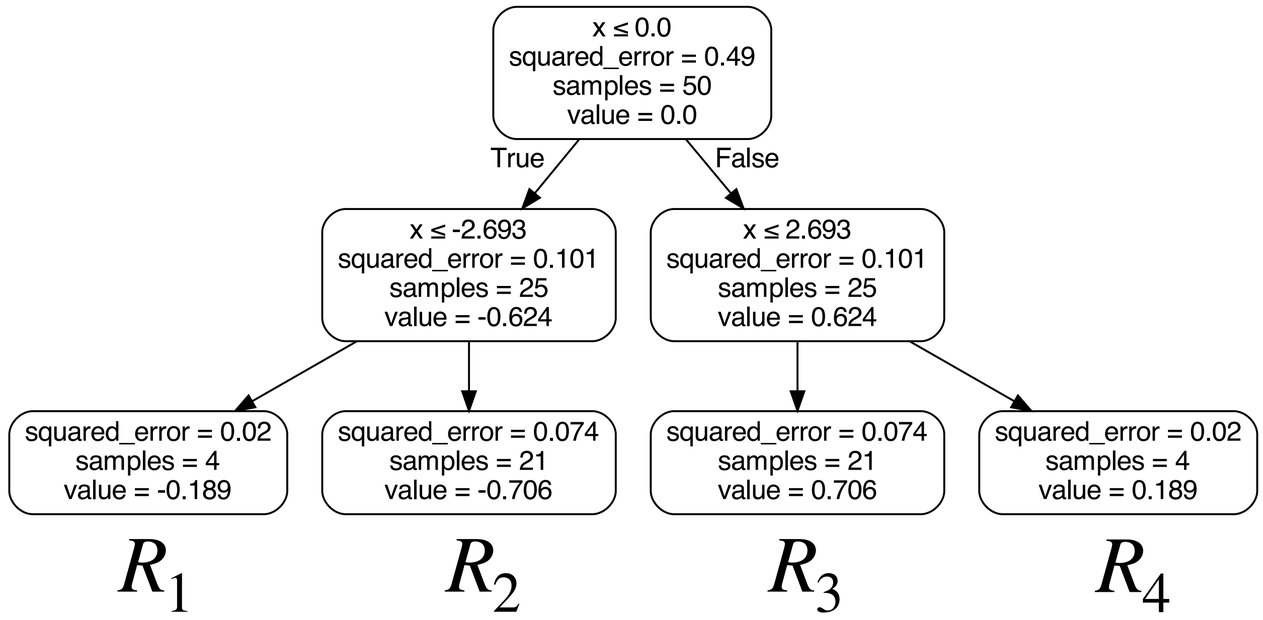
Is this a scenario where you would trade off interpretability for performance?

Regression trees

Regression trees

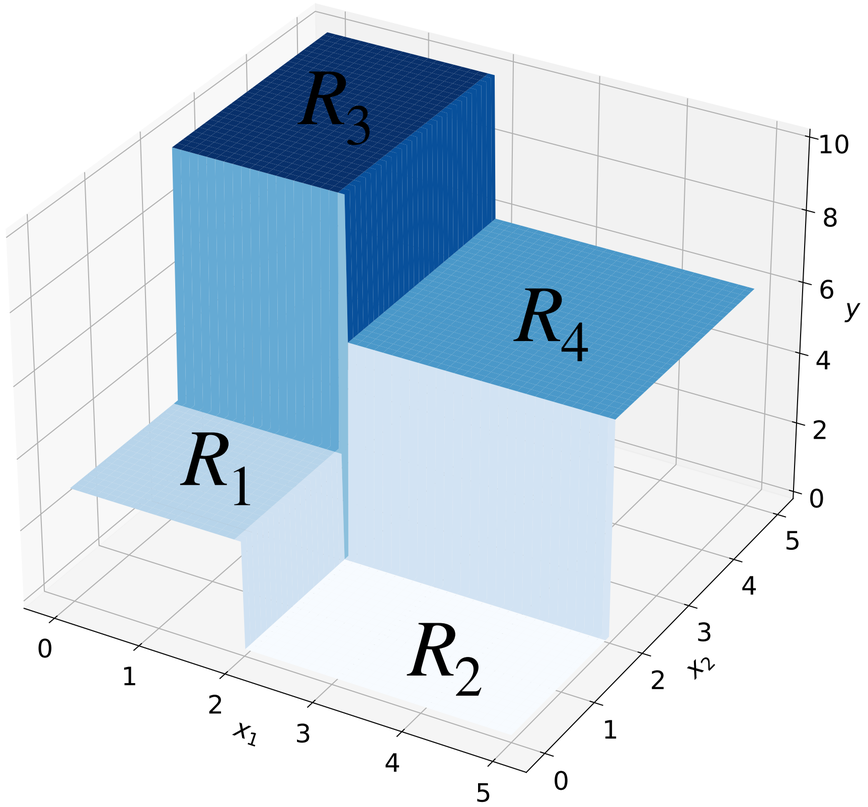
- Very similar to classification trees except the output is real-valued





Regression trees

- The output is a bunch of discrete regions R_j with values w_j
- We can write this functionally as $f(\mathbf{x}) = \sum_j w_j \mathbb{I}(\mathbf{x} \in R_j)$ if we must :)



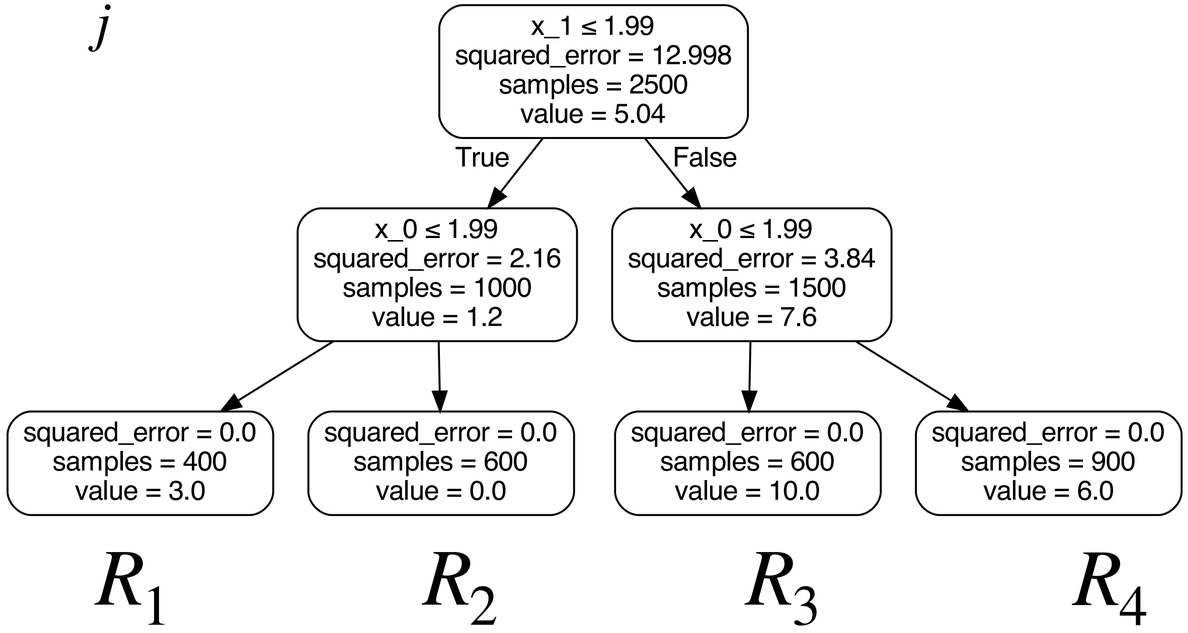


Figure inspired by the one in Murphy

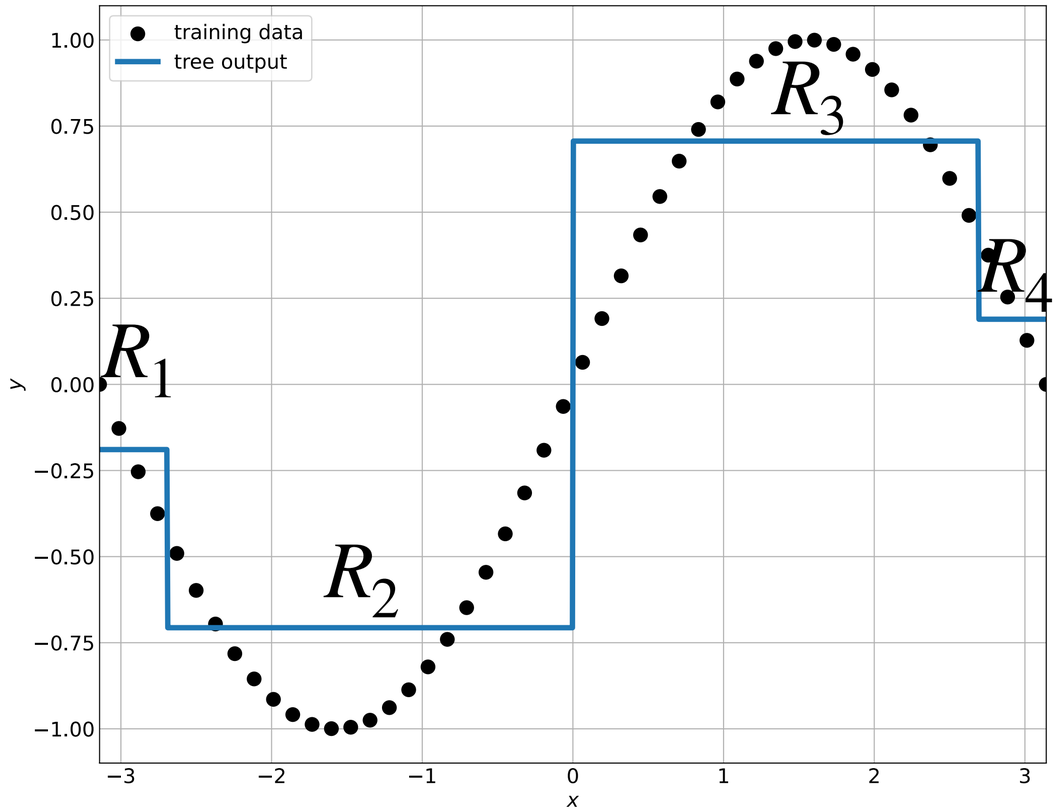
Finding the best split at a node

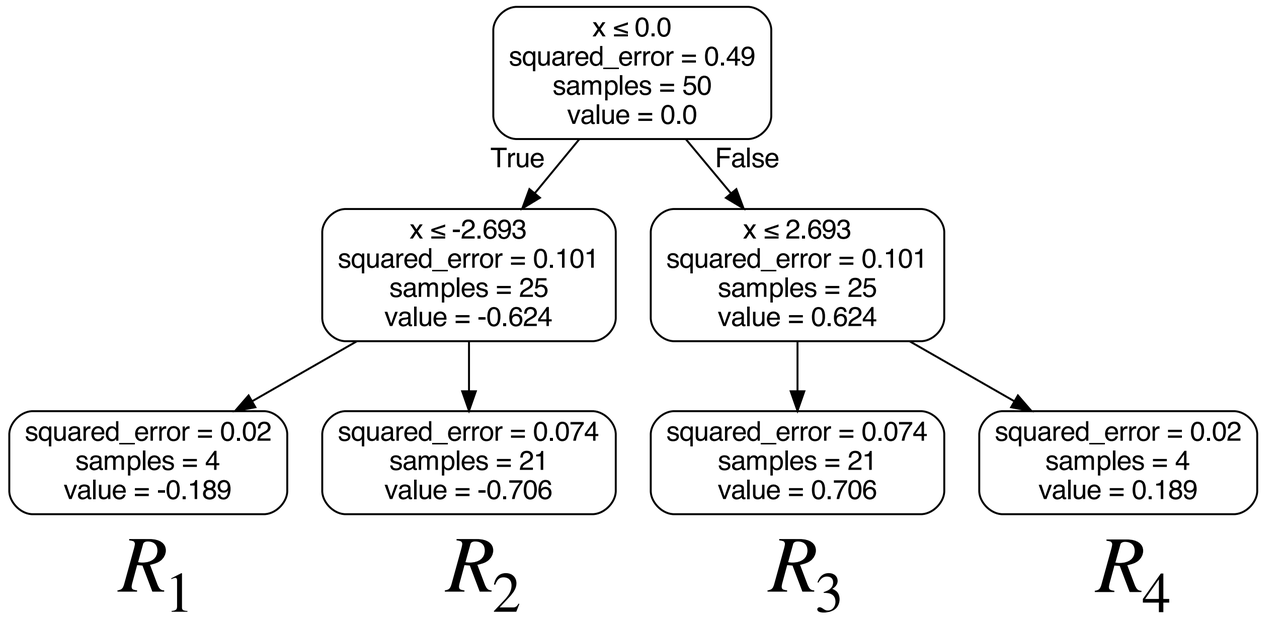
- The learning algorithm is almost the same as for classification trees
- At each node we solve $\underset{j,t}{\text{minimise}} \; \frac{|\mathfrak{D}_i^L(j,t)|}{|\mathfrak{D}_i|} c(D_i^L(j,t)) + \frac{|\mathfrak{D}_i^R(j,t)|}{|\mathfrak{D}_i|} c(D_i^R(j,t))$
- For regression trees we use **mean squared error** for c

$$c(\mathfrak{D}_i) = \frac{1}{|\mathfrak{D}_i|} \sum_{n \in \mathfrak{D}_i} (y^{(n)} - \bar{y})^2$$ where $\bar{y}$ is the mean target in a split

The prediction

- At a leaf node, this is the the mean target associated with its training split





The pros and cons of trees

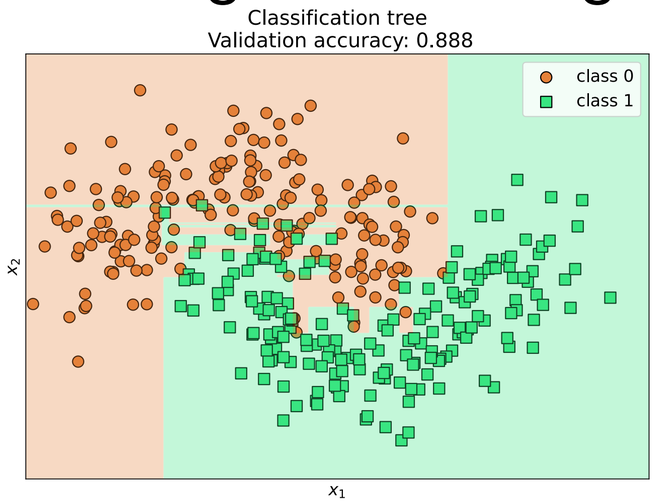
- **Pro:** They are interpretable (-ish)
- **Pro:** They are fast to train
- **Pro:** They don’t require data to be scaled
- **Pro:** They can handle mixed (discrete + continuous) inputs
- **Con:** They **are unstable** (add some noise and get a different tree!)
- **Con:** They typically don’t work as well as other models by themselves

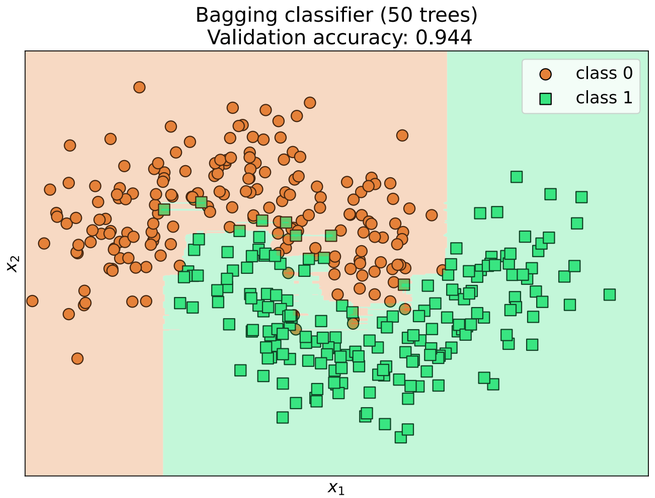
Inspired by Murphy

Bagging and boosting

Bagging

- A decision tree can (and will) overfit to training data, making mistakes that won't give us a model that generalises to held-out data
- But if we have an **ensemble of trees** trained on **different versions of the training data** then they won't all make the same mistakes
- We expect that if we average the decisions of all these trees (the *wisdom of the crowd*) we will get something that generalises better





Bagging (bootstrap aggregation)

Bagging isn't exclusive to tree-based models!

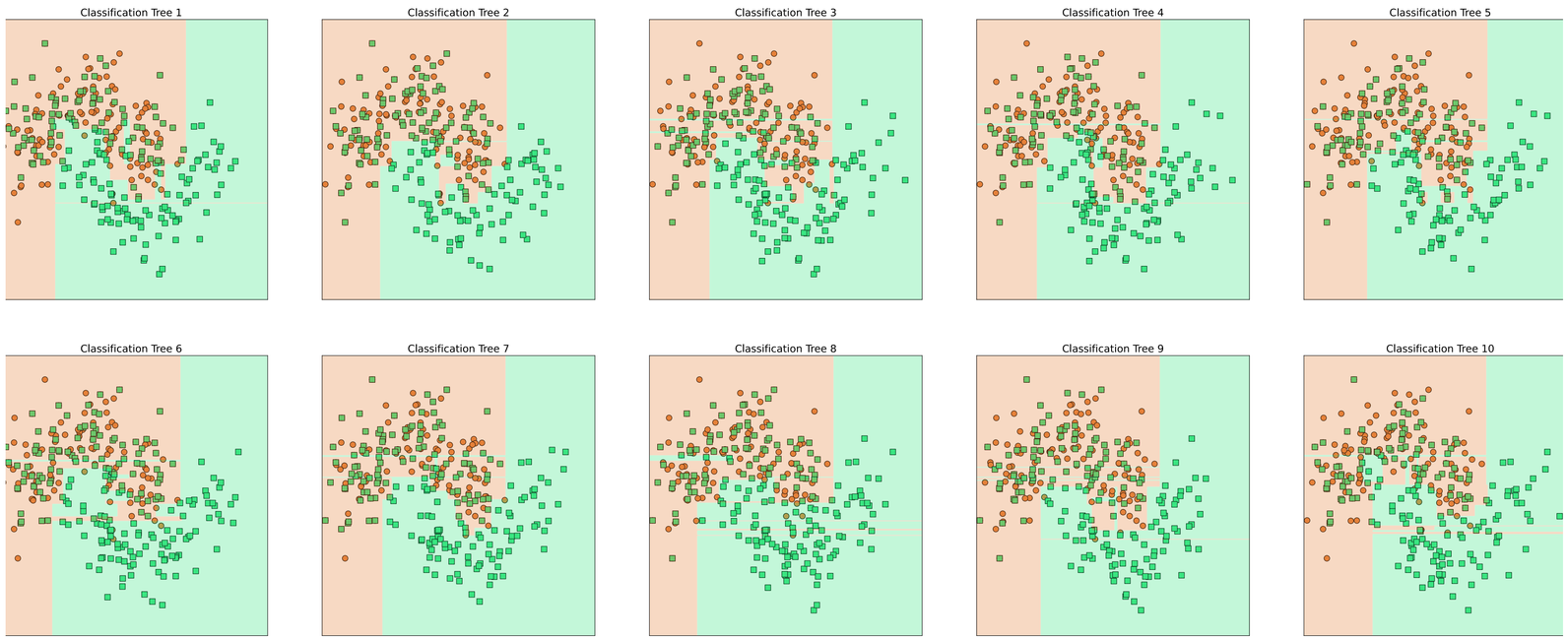
- Goal: We want to train a bagging ensemble on our training set $\mathfrak{D} = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N\}$
- For t in range(T):
  - Create a **bootstrap** dataset $\mathfrak{D}^{(t)}$ from the training set: M points are sampled **with replacement** from $\mathfrak{D}$ (usually $M = N$)
  - Train a model $f_t(\mathbf{x})$ on $\mathfrak{D}^{(t)}$

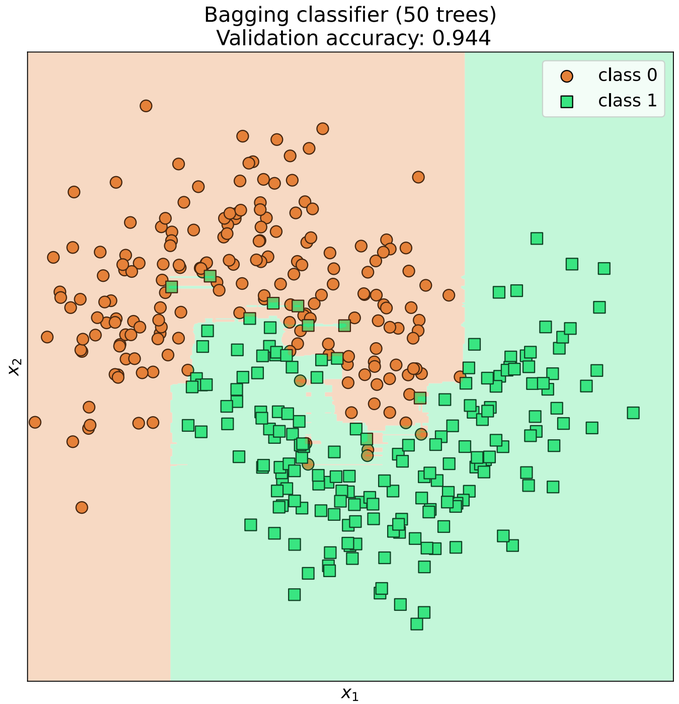
To apply the ensemble to a new point, **average** the outputs of the T models

$$f(\mathbf{x}) = \frac{1}{T}\sum_t f_t(\mathbf{x})$$

Bagging with Classification trees

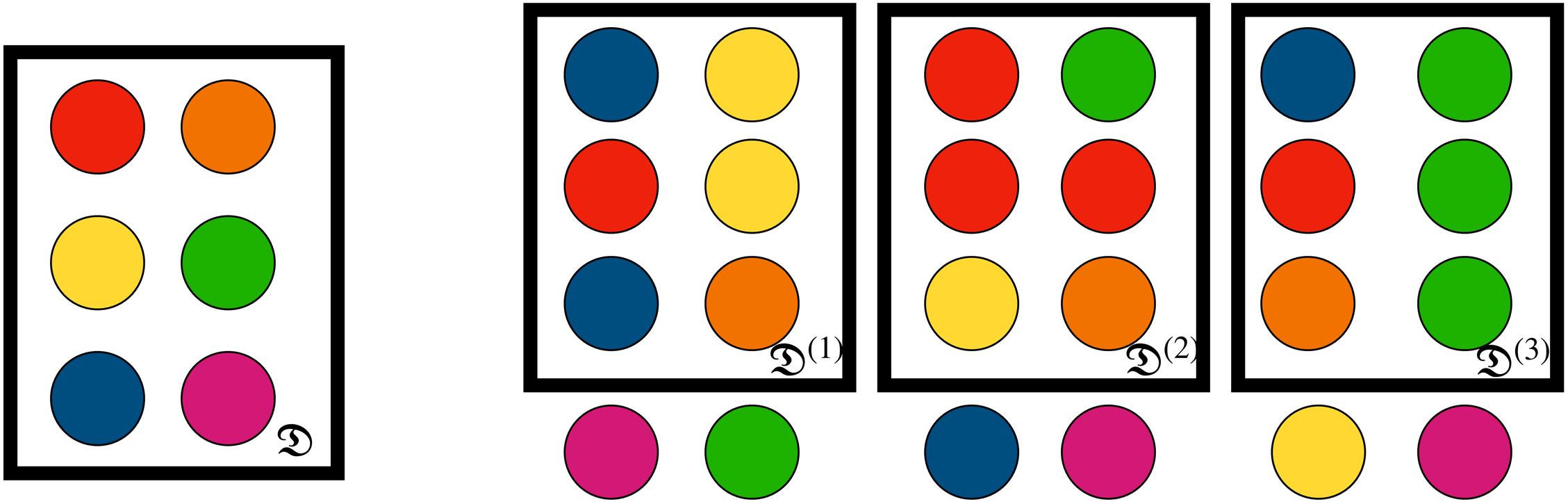
- Put a new point through each tree t to get the class probability distribution $\mathbf{p}_t$
- Then just average all the $\mathbf{p}_t$ and pick the class with the highest probability





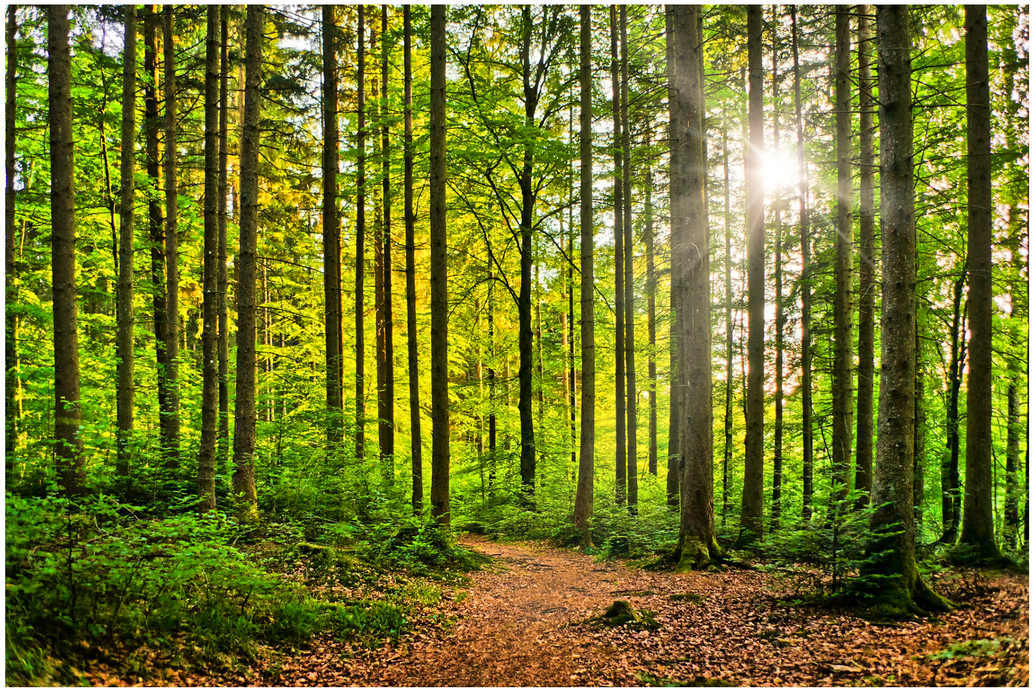
Out of bag (OOB) samples

- Each bootstrap is usually N points sampled with replacement
- The points that aren't in a bootstrap can be used for validation



Random forests

- A random forest is a bagging ensemble where each individual model is a tree
- For each tree, **only a random subset of features are made available at each node for learning each split**
- This encourages variety!



Boosting

Boosting isn't exclusive to tree-based models, but they are the most common weak learner

- Consider a **weak learner** $F_0(\mathbf{x})$ that makes lots of mistakes
- Now consider another weak learner $F_1(\mathbf{x})$ that tries to fix the mistakes made by $F_0(\mathbf{x})$. We can use these in an ensemble: $f_1(\mathbf{x}) = F_0(\mathbf{x}) + F_1(\mathbf{x})$
- This will still be terrible, but what if we add $F_2(\mathbf{x})$ that tries to fix the mistakes made by the previous ensemble and so on…
- This process is known as **boosting**. It creates an ensemble of weak learners that in combination can be very powerful: $f_t(\mathbf{x}) = \sum_t F_t(\mathbf{x})$

Gradient boosting rationale

- This is basically gradient descent in **function space**
- We have a training set $\mathfrak{D} = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N\}$
- Let's represent our function at some step t by its values on the training set:

$$\mathbf{f}_t = \begin{bmatrix} f_t(\mathbf{x}^{(1)}) & f_t(\mathbf{x}^{(2)}) & \dots & f_t(\mathbf{x}^{(N)}) \end{bmatrix}^\top$$

- If there is some loss L we care about minimising (e.g. log loss) then we can evaluate it for each training point:

$$\mathbf{L}(\mathbf{f}_t) = \begin{bmatrix} L(y^{(1)}, f_t(\mathbf{x}^{(1)})) & L(y^{(2)}, f_t(\mathbf{x}^{(2)})) & \dots & L(y^{(N)}, f_t(\mathbf{x}^{(N)})) \end{bmatrix}^\top$$

Gradient boosting rationale continued

- We have a loss at step t given by

$$\mathbf{L}(\mathbf{f}_t) = \begin{bmatrix} L(y^{(1)}, f_t(\mathbf{x}^{(1)})) & L(y^{(2)}, f_t(\mathbf{x}^{(2)})) & \dots & L(y^{(N)}, f_t(\mathbf{x}^{(N)})) \end{bmatrix}^\top$$

- We can compute the gradient of this wrt. $\mathbf{f}_t$:

$$\mathbf{g}_t = \begin{bmatrix} \nabla_{f_t} L(y^{(1)}, f_t(\mathbf{x}^{(1)})) & \nabla_{f_t} L(y^{(2)}, f_t(\mathbf{x}^{(2)})) & \dots & \nabla_{f_t} L(y^{(N)}, f_t(\mathbf{x}^{(N)})) \end{bmatrix}^\top = \begin{bmatrix} g_t^{(1)} & g_t^{(2)} & \dots & g_t^{(N)} \end{bmatrix}$$

- And perform a gradient descent-style update $\mathbf{f}_{t+1} = \mathbf{f}_t - \alpha \mathbf{g}_t$

Gradient tree boosting

- In gradient tree boosting our function at step $f_t(\mathbf{x})$ is an ensemble of regression trees $f_t(\mathbf{x}) = \sum_t F_t(\mathbf{x})$
- We perform a gradient-descent style update by **fitting a new regression tree to the negative gradient**

$$\mathbf{g}_t = \left[\nabla_{f_t} L(y^{(1)}, f_t(\mathbf{x}^{(1)})) \quad \nabla_{f_t} L(y^{(2)}, f_t(\mathbf{x}^{(2)})) \quad \dots \quad \nabla_{f_t} L(y^{(N)}, f_t(\mathbf{x}^{(N)})) \right]^\top = \left[g_t^{(1)} \quad g_t^{(2)} \quad \dots \quad g_t^{(N)} \right]$$

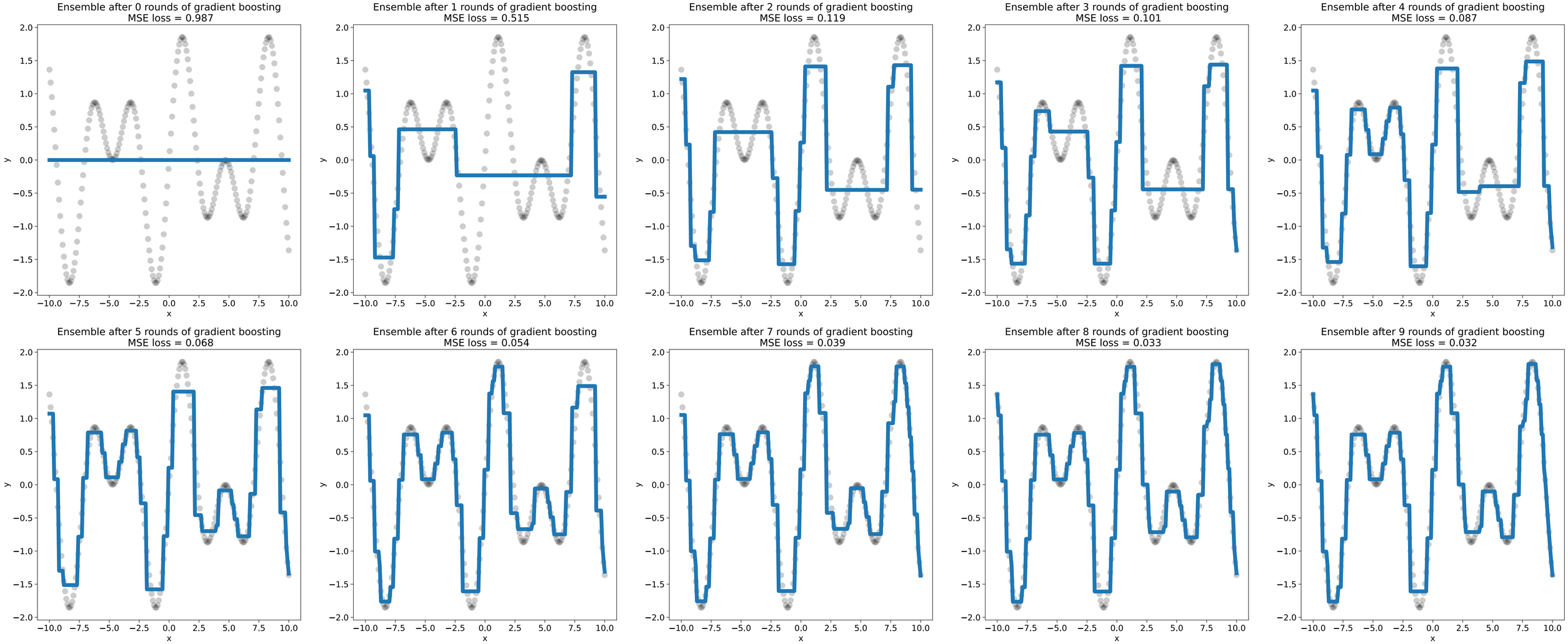
- So usually we fit a regression tree using $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$
- In gradient boosting we fit a regression tree using $\{(\mathbf{x}^{(n)}, -g_t^{(n)})\}_{n=1}^N$

Gradient tree boosting algorithm

- Goal: We want to train a gradient boosted ensemble on our training set $\mathfrak{D} = \{(\mathbf{x}^{(n)}, y^{(n)}) \in N\}$ that minimises some loss L
- Initialise $f_0(\mathbf{x}) = F_0(\mathbf{x})$
- For t in range(T):
  - Compute the gradients $\mathbf{g}_t = \begin{bmatrix} g_t^{(1)} & g_t^{(2)} & \dots & g_t^{(N)} \end{bmatrix}$
  - Fit a regression tree $F_t(\mathbf{x})$ using $\{(\mathbf{x}^{(n)}, -g_t^{(n)})\}_{n=1}^{N}$
  - Update ensemble $f_{t+1}(\mathbf{x}) = f_t(\mathbf{x}) + \alpha F_t(\mathbf{x})$

Sometimes the values at the leaf nodes of each tree are re-computed to minimise the loss after fitting but we omit this detail for simplicity

Gradient tree boosting example



XGBoost (eXtreme Gradient Boosting)

- Famous for winning lots of Kaggle competitions!
- This is gradient tree boosting plus a bag of tricks
- See Murphy p613 for more details

Coursework 2 (30% of course mark)

- You will perform data analysis and machine learning on a dataset of the metadata and reviews for 3777 papers submitted to a top machine learning conference (ICLR 2023)
- You should write a **4-8 page report with an appendix containing code** where you:
  1. Describe the dataset, and what the different columns correspond to
  2. Provide informative summarisations and visualisations of the dataset and discuss these
  3. Propose and carefully define several regression and classification tasks on this dataset
  4. Train and evaluate models for these tasks
  5. Select a model for each task, and discuss its usefulness
- The full brief, dataset, submission instructions, and the marking rubric are available on Learn under the “Assessment” tab (after 1000 today). **Deadline 26/3 @ 1600**

Summary

- We have learnt about classification and regression trees
- We have seen how to formulate node splitting as an optimisation problem
- We have seen how decision trees can overfit
- We have learnt about bagging to create ensembles
- We have learnt about gradient boosting to create ensembles