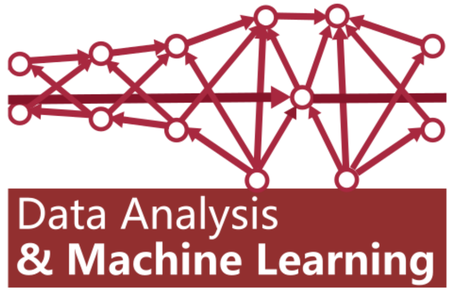


Data Analysis and Machine Learning 4

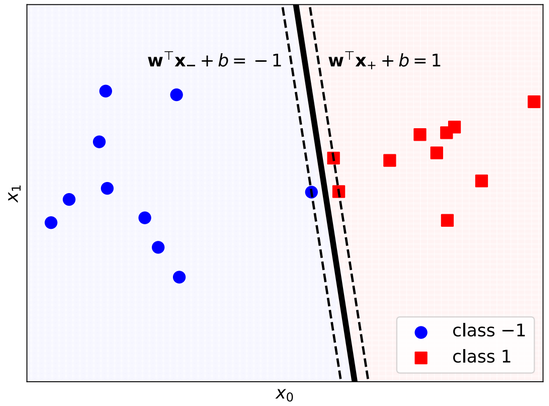
Week 8: Non-parametric models

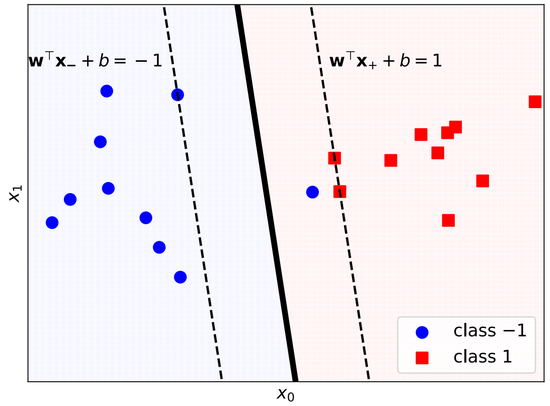
Elliot J. Crowley, 13th March 2023

THE UNIVERSITY *of* EDINBURGH

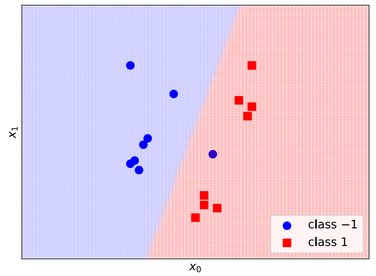
Recap

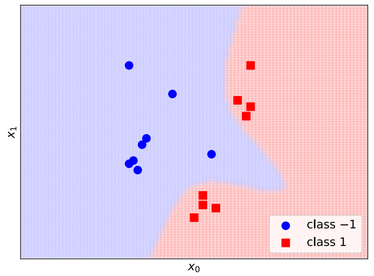
- We motivated the hard-margin SVM formulation to obtain a max-margin classifier and relaxed it to allow for margin violations

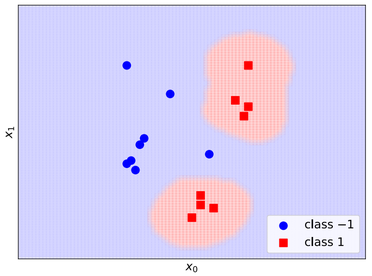




- We introduced the dual form of the SVM and showed how the kernel trick lets us implicitly map features to higher dimensions







Non-parametric models

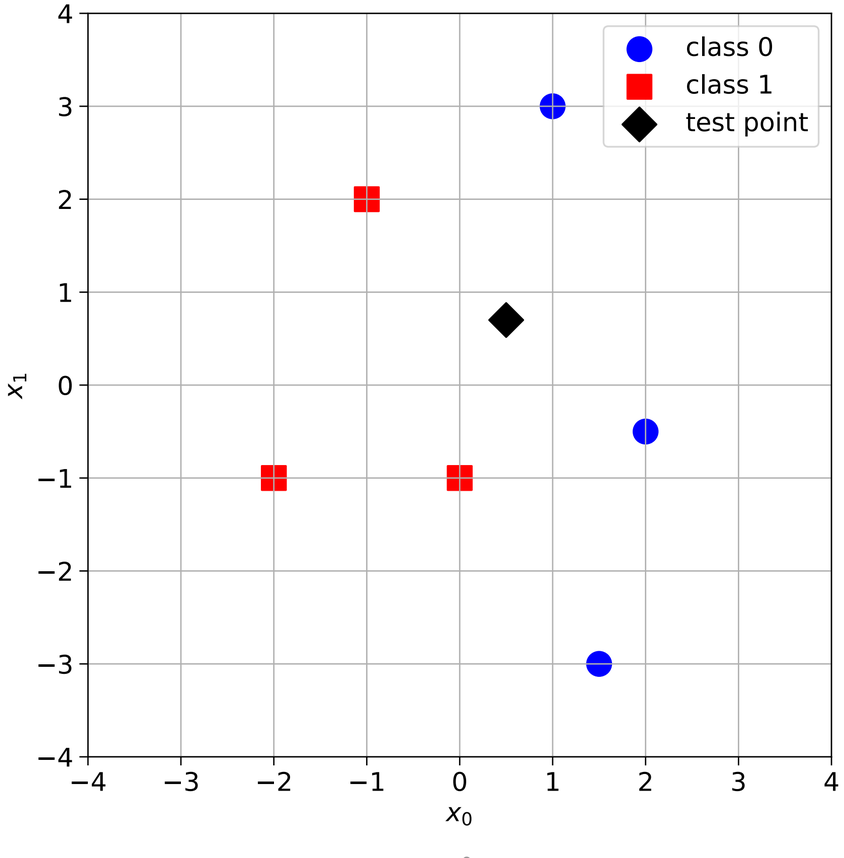
- Parametric models are represented by a function with a fixed number of parameters i.e. they have a fixed capacity
- Non-parametric models are not!
- The capacity of a non-parametric can scale with the number of data points
- In this lecture we will consider k-nearest neighbour and decision tree classifiers which are both non-parametric models
- We will also look at a random forest, which is an *ensemble* of decision trees

Credit to Joe Mellor for the description

k-nearest neighbours

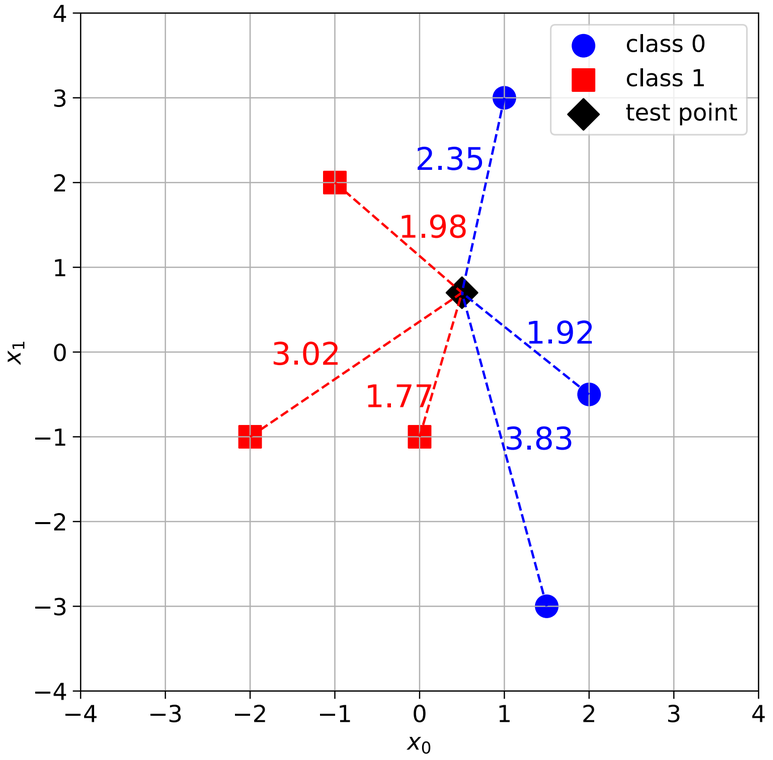
k-NN classification

- k-nearest neighbours (k-NN) is a simple algorithm for classification
- It has no parameters, and a single hyperparameter k
- Consider a training set $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ where data points are 2D $\mathbf{x} \in \mathbb{R}^2$ and labels are binary $y \in \{0,1\}$
- Let's use k-NN with $k = 3$ to perform **binary classification** on a test point $\mathbf{x}^{(t)}$
- i.e. classify $\mathbf{x}^{(t)}$ as either class 1 or class 0

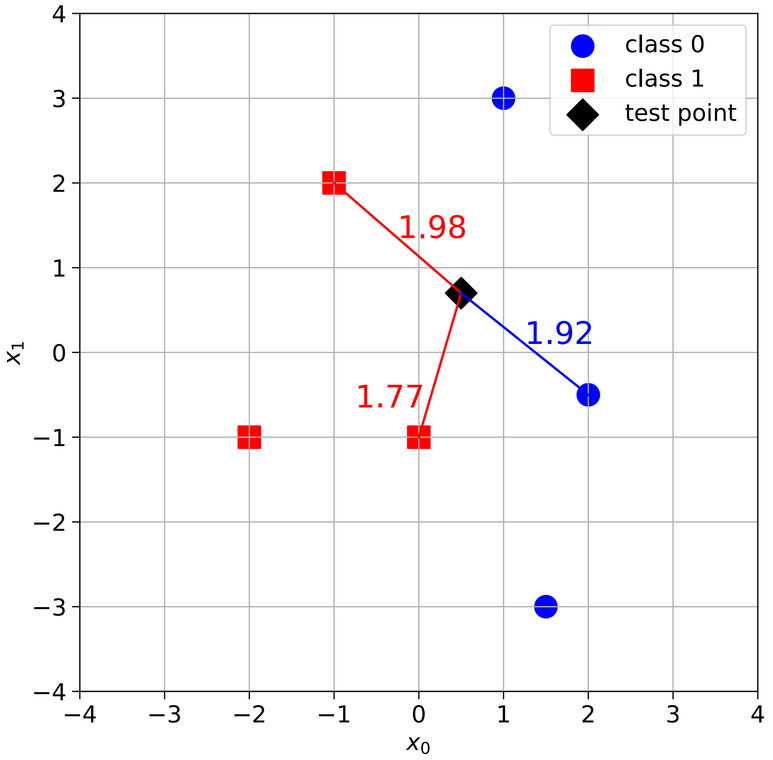


k-NN algorithm for classifying a test point

1. Compute distances between training points and test point

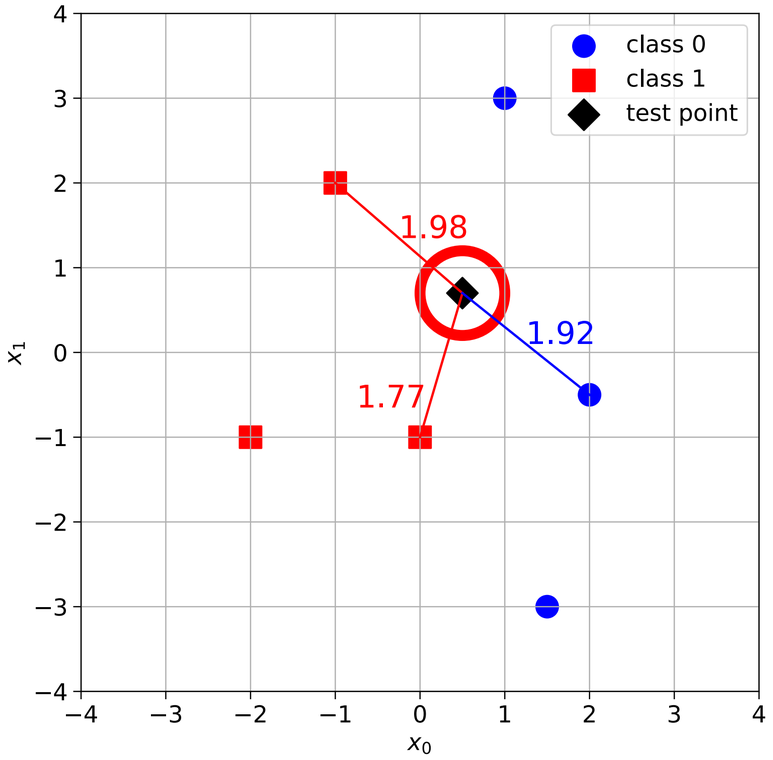


2. Isolate k-nearest training points



Here $k = 3$

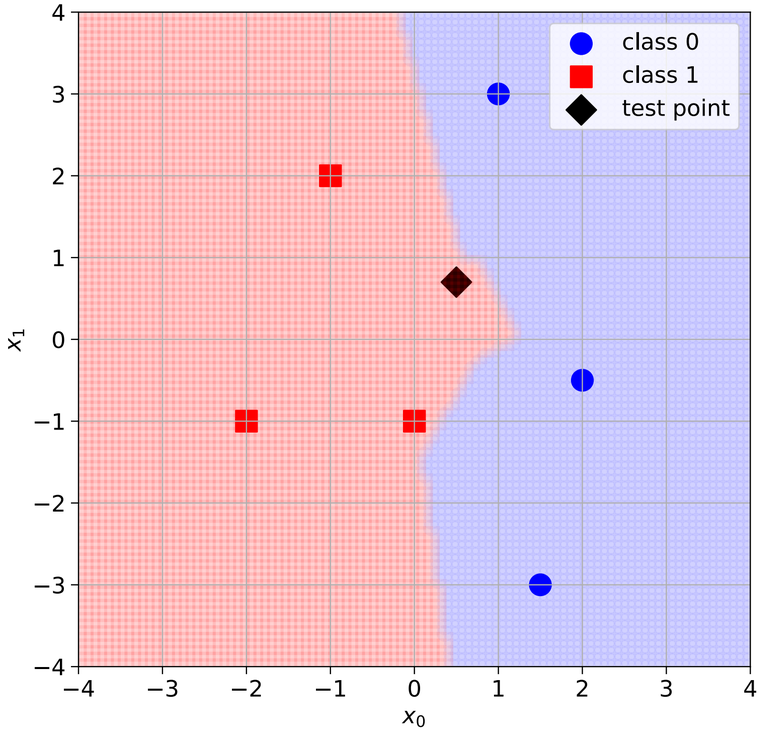
3. Classify as mode of the labels of the k-nearest points



Predict as class 1

k-NN decision boundary

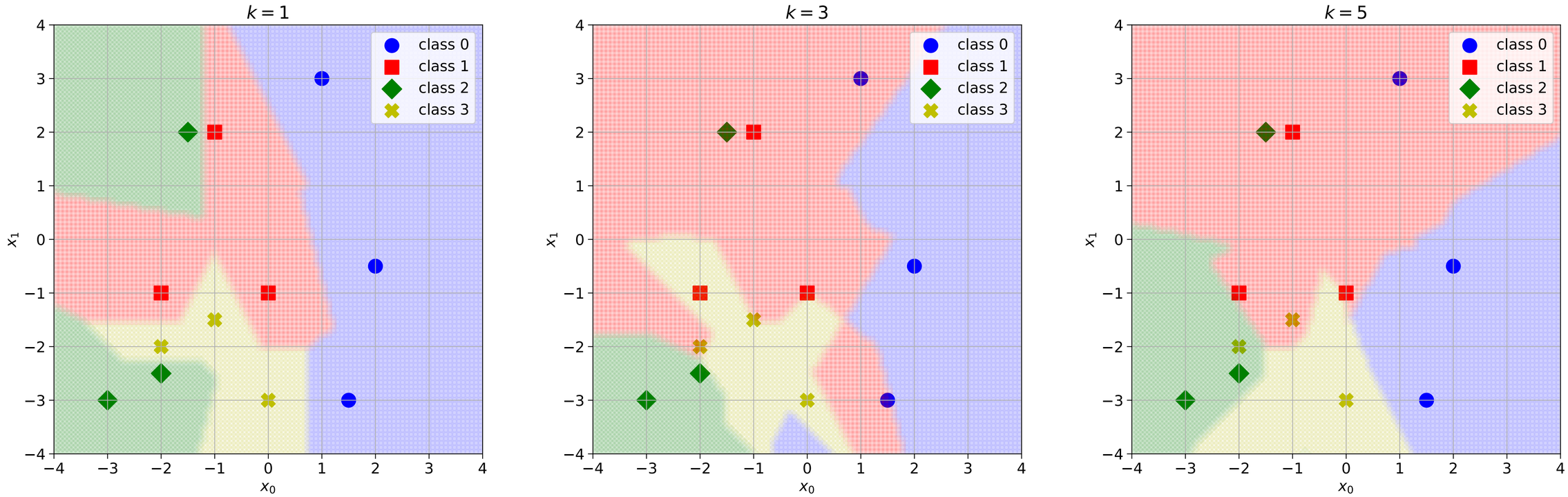
- Consider how an arbitrary point will be classified in this space
- There will be a region where it is predicted as class 0, and another for class 1
- The border of these two regions is the classifier's **decision boundary**
- This is non-linear for k-NN



Multi-class classification with k-NN

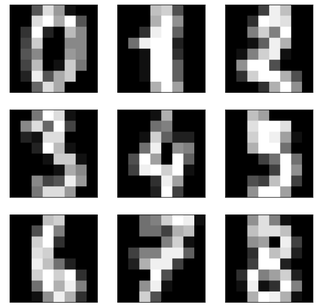
- We have considered binary classification with k-NN
- The algorithm works for multi-class ($K > 2$) classification too

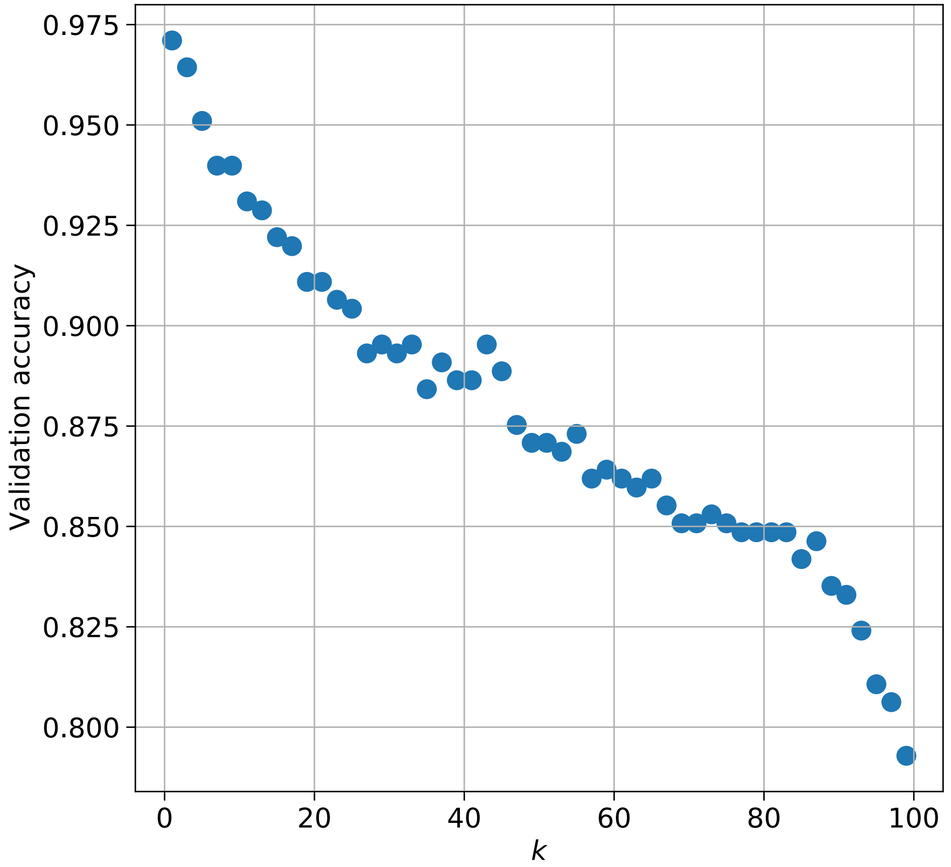
K and k are not the same!



Tuning k by grid search

Classifying digits on vectorised images $\mathbf{x} \in \mathbb{R}^{64}$ labelled $y \in \mathbb{Z}^{+}_{<10}$





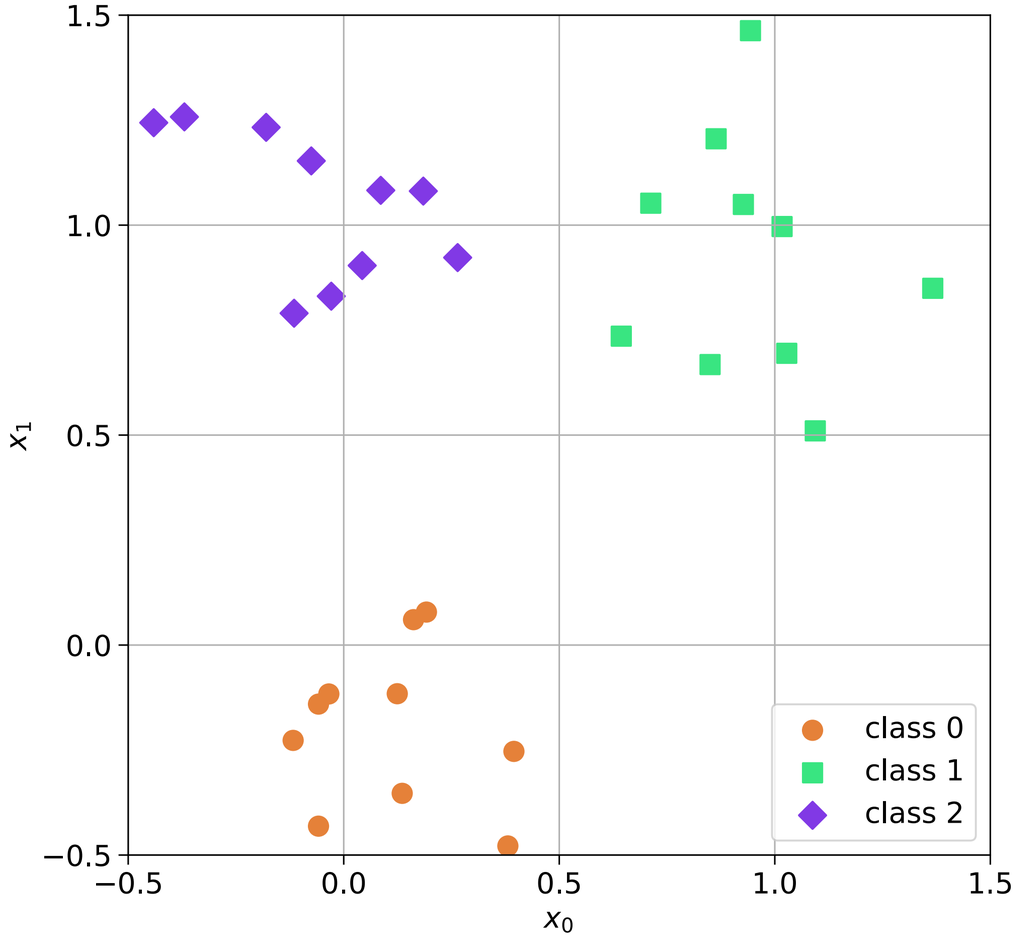
Highest validation accuracy for $k = 1$

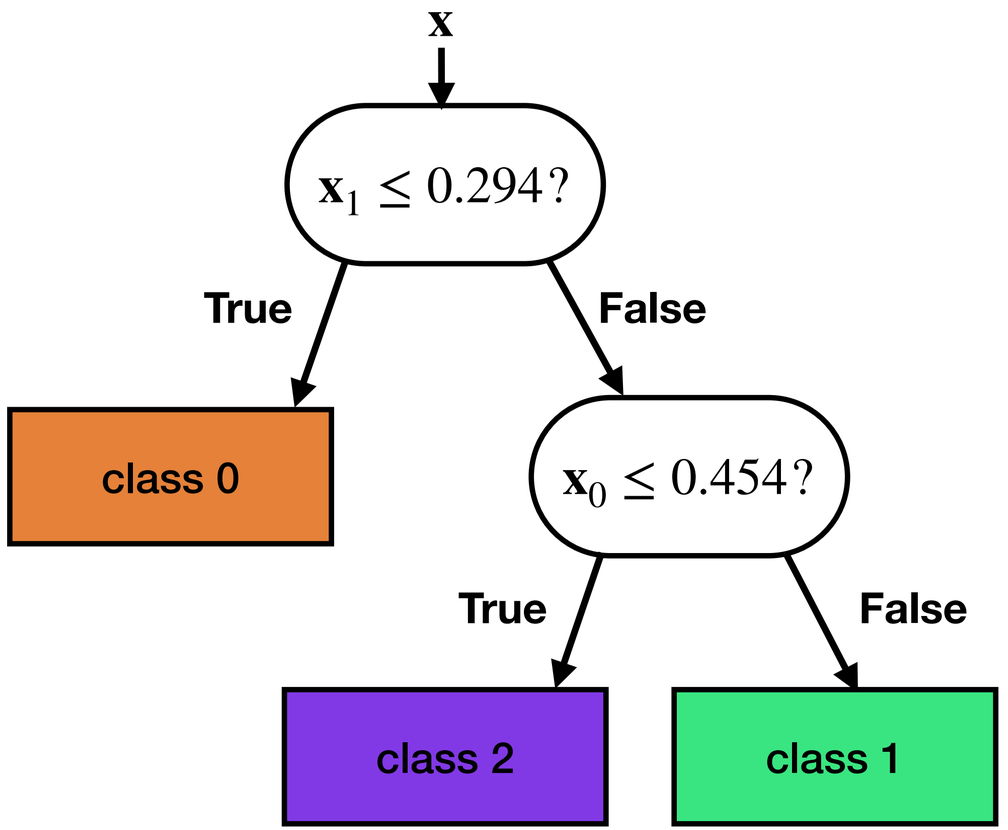
Use this for test evaluation

Decision trees

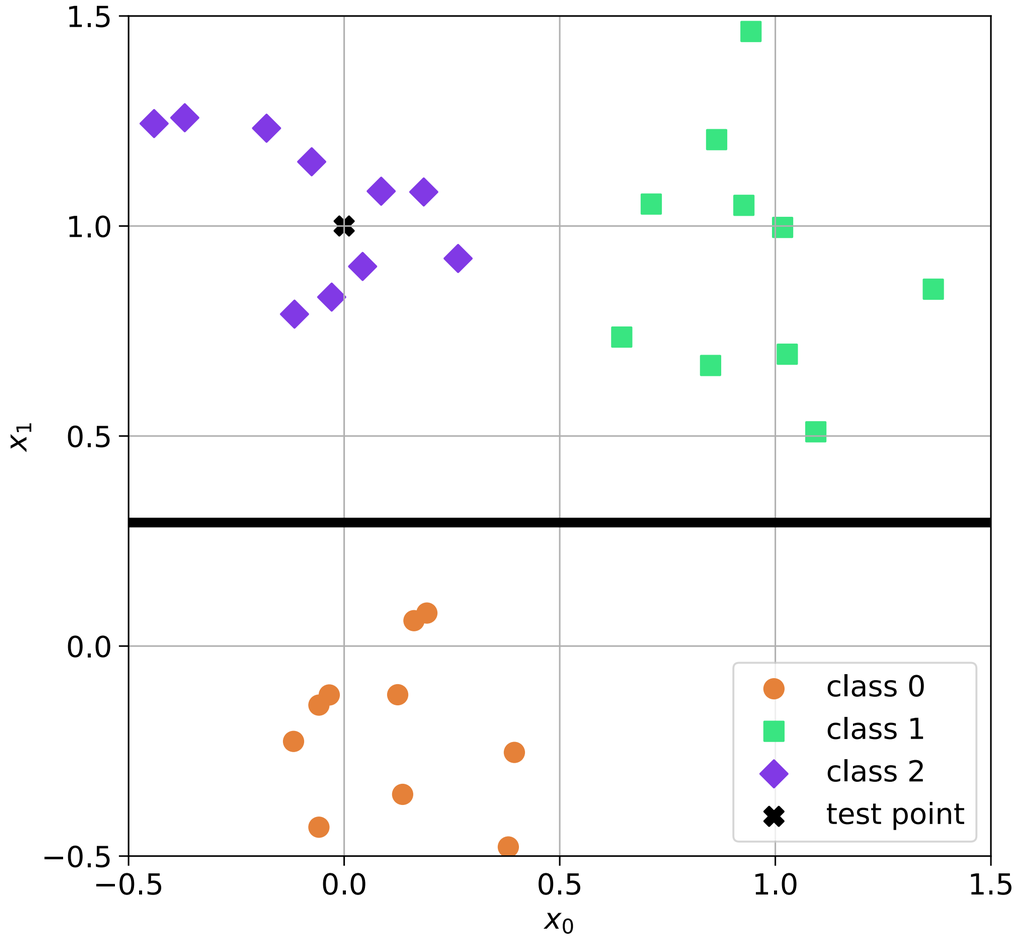
Decision trees for classification

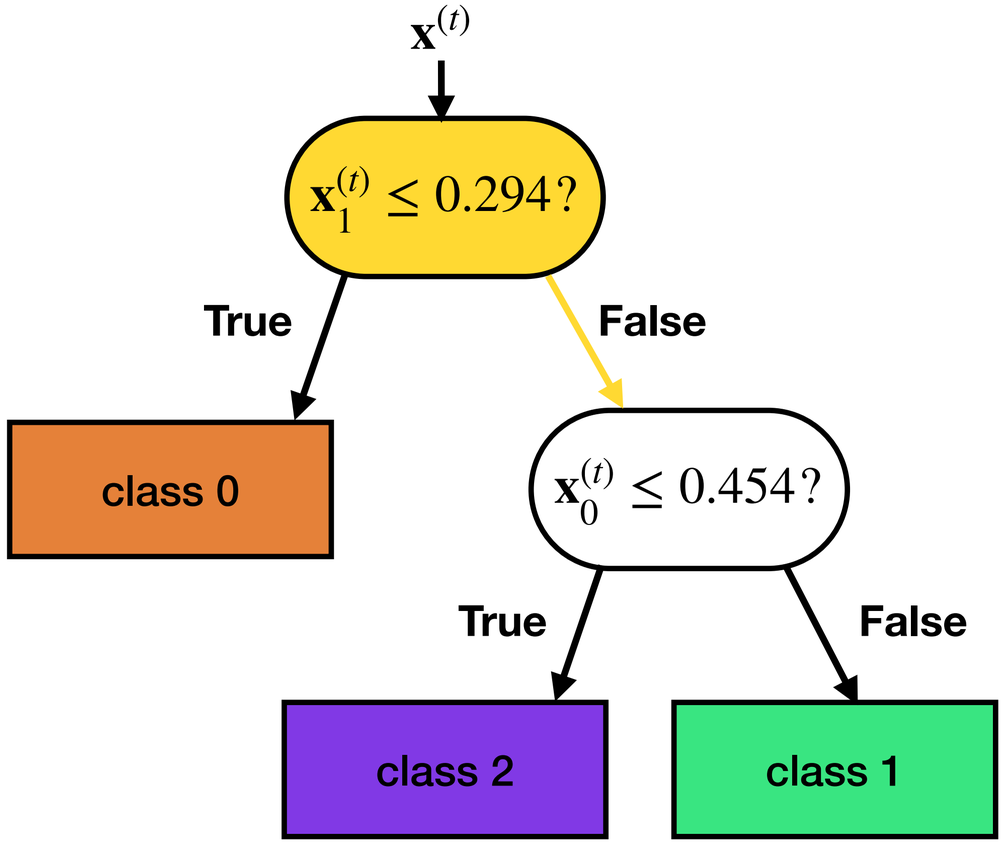
A learnt tree of simple binary rules based on thresholding feature values



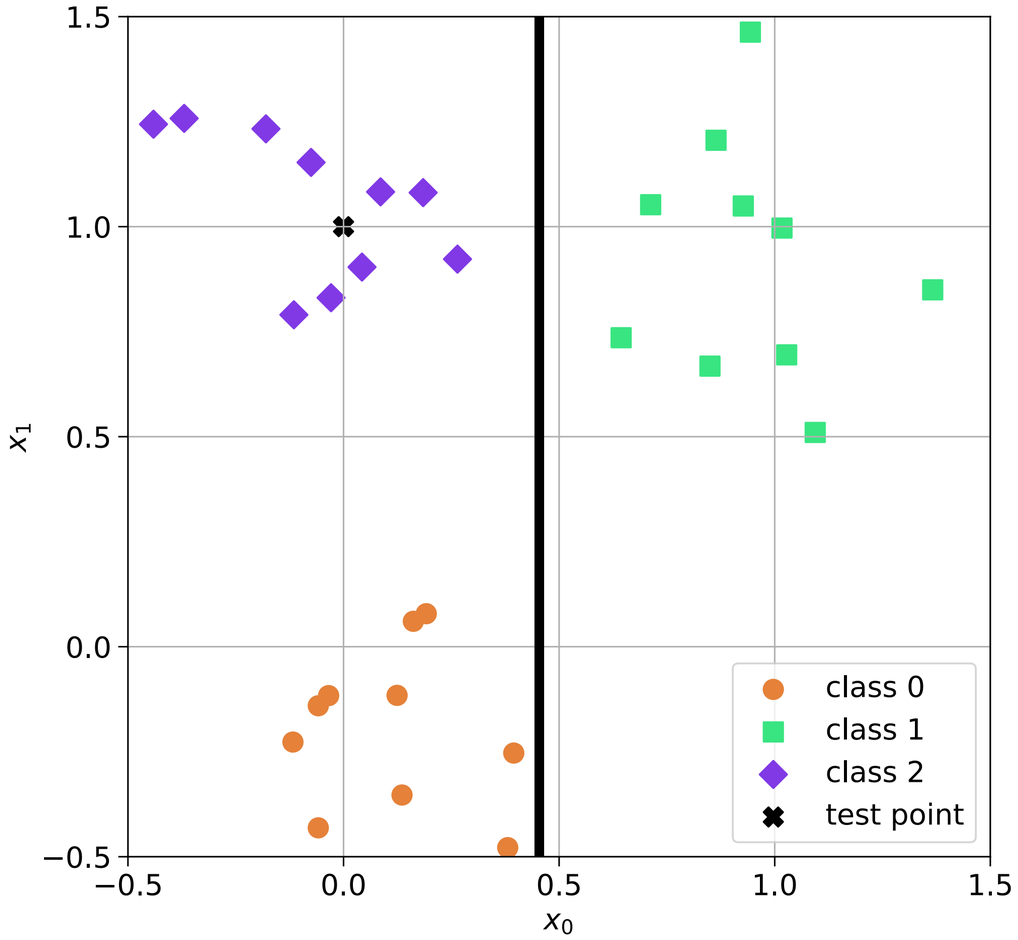


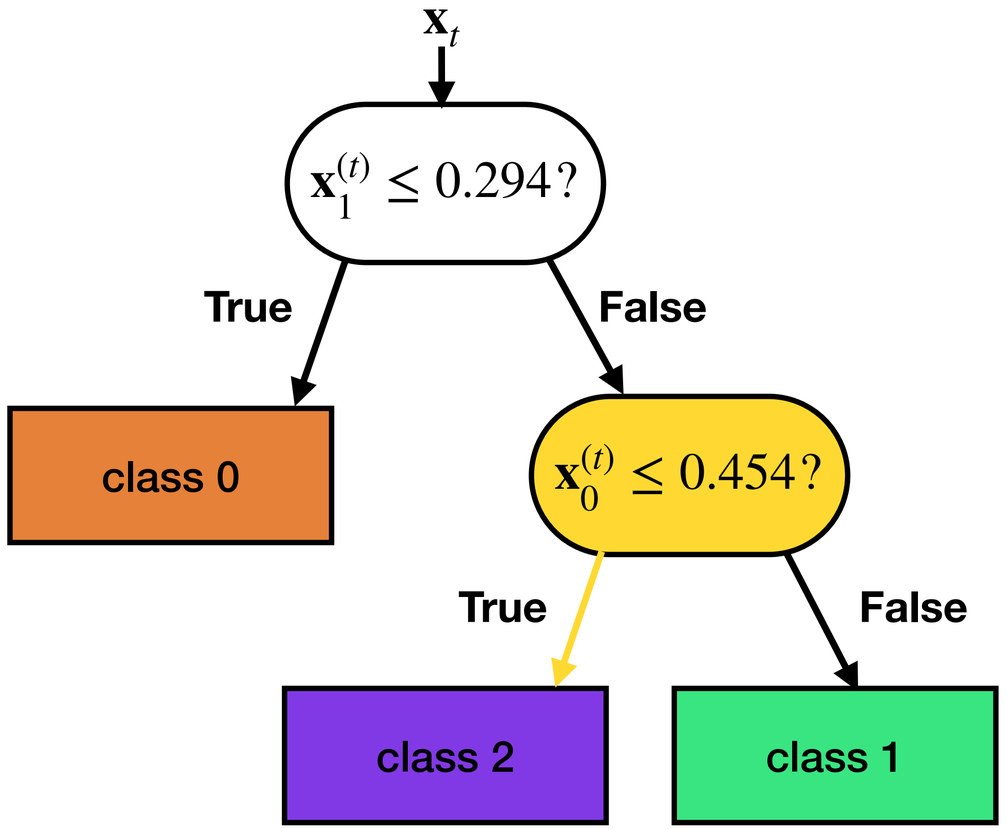
Classifying a point $\mathbf{x}^{(t)}$





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

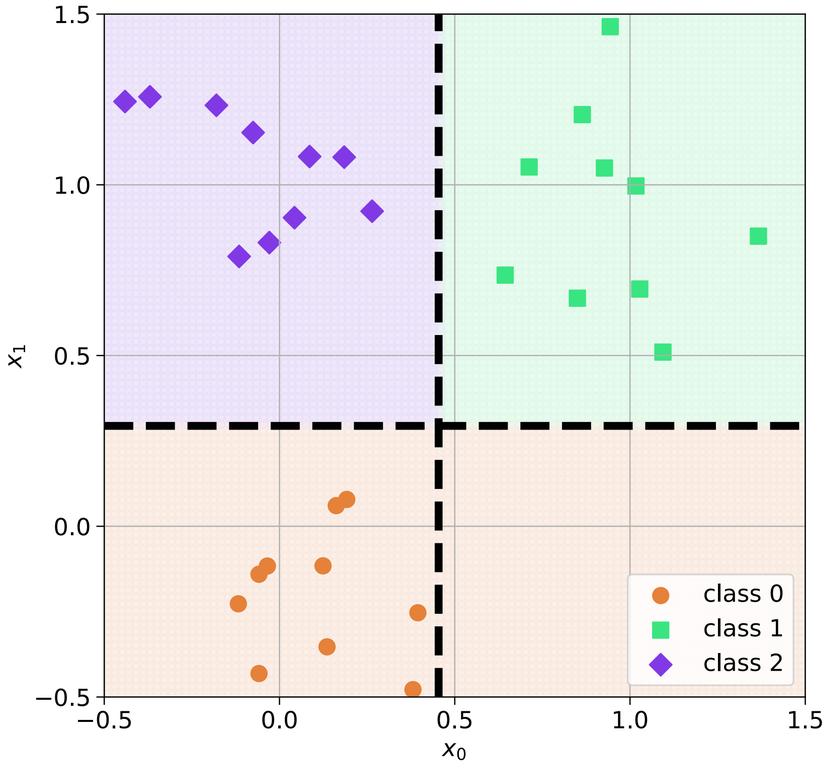


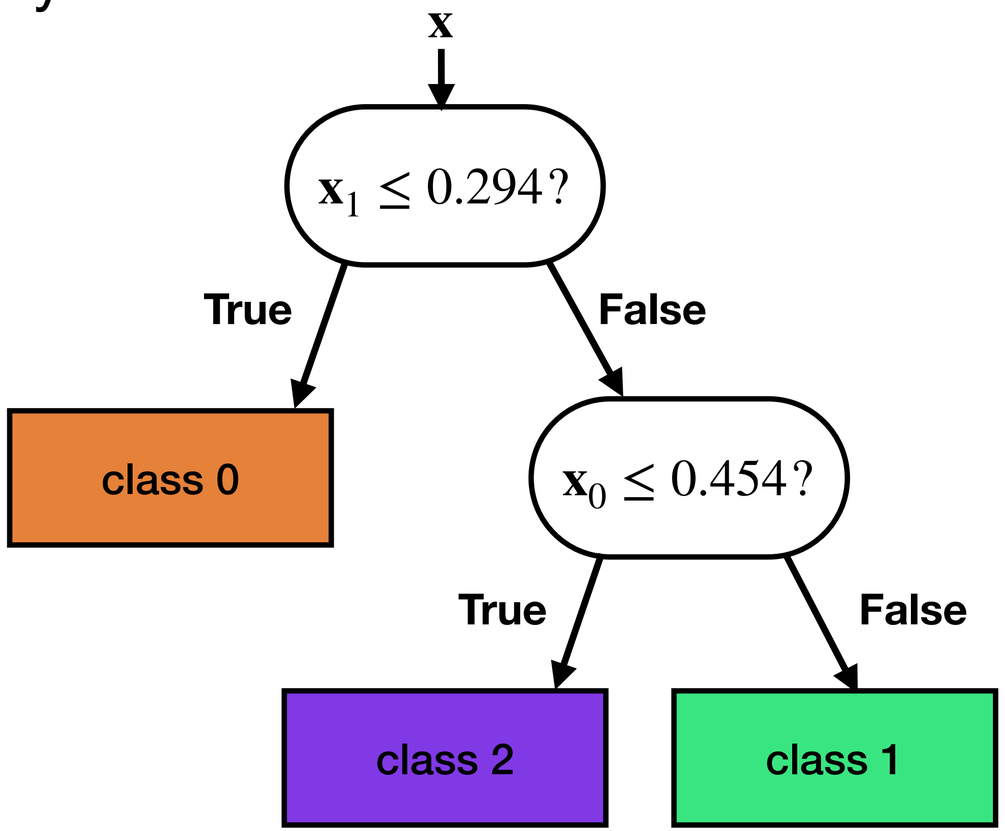


$\mathbf{x}_t$ is classified as class 2

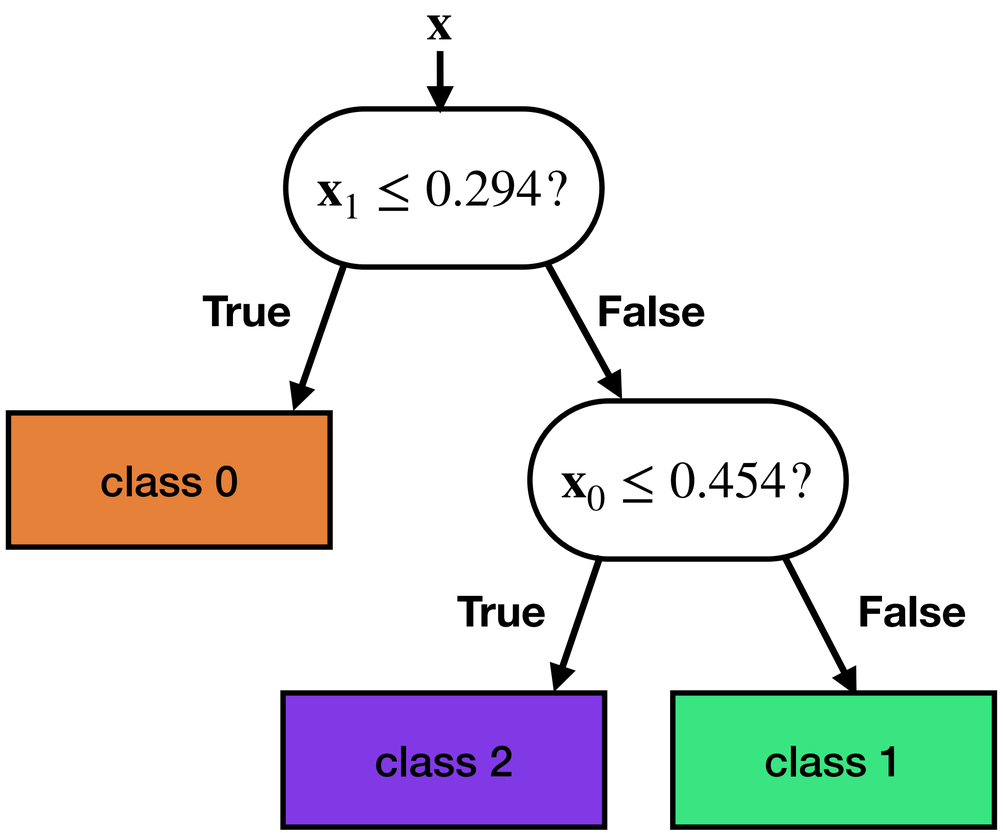
What is the tree doing?

- It is slicing up the feature space using straight lines
- This gives a non-linear decision boundary





Some nomenclature



This tree has depth of 2 as there are two *levels* of feature thresholding

The boxes are all nodes

Nodes at which class decisions are made are leaf nodes

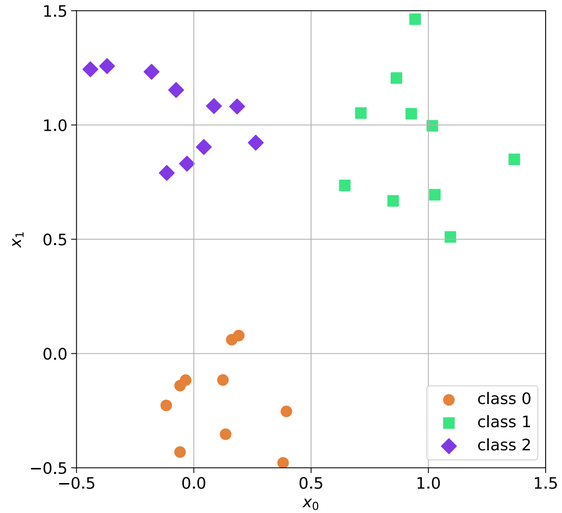
Nodes that are not leaf nodes have a left and right child node

Learning a decision tree from training data

- We have a dataset $Q_0 = \{\mathbf{x}_n, y_n\}_{n=0}^{30-1}$ where $\mathbf{x} \in \mathbb{R}^2$ and $y \in \{0,1,2\}$
- First we decide on the maximum depth for our decision tree (let's say 3)
- Then we decide how to split at the first node
- The first node splits Q_0 into Q_0^{left} and Q_0^{right}according to

$$Q_0^{left}(d, t_0) = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n:x_d^{(n)} \leq t_0}$$

$$Q_0^{right}(d, t_0) = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n:x_d^{(n)} > t_0}$$



Formulation based on https://scikit-learn.org/stable/modules/tree.html#tree-algorithms

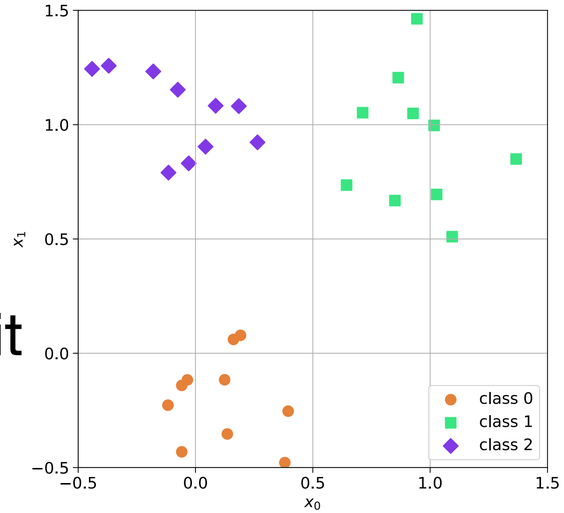
Learning a decision tree from training data

- We have $Q_0^{left}(d, t_0) = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n:x_d^{(n)} \leq t_0}$ and $Q_0^{right}(d, t_0) = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n:x_d^{(n)} > t_0}$
- Let's say we have some function H that tells us how bad a split it
- How much we care about a split should be proportional to its size (e.g. we'd be happy with something that splits 99% of our data well, and 1% badly)
- We can devise a loss function and minimise it

$$L_0 = \frac{n_0^{left}}{n_0} H(Q_0^{left}(d, t_0)) + \frac{n_0^{right}}{n_0} H(Q_0^{right}(d, t_0))$$

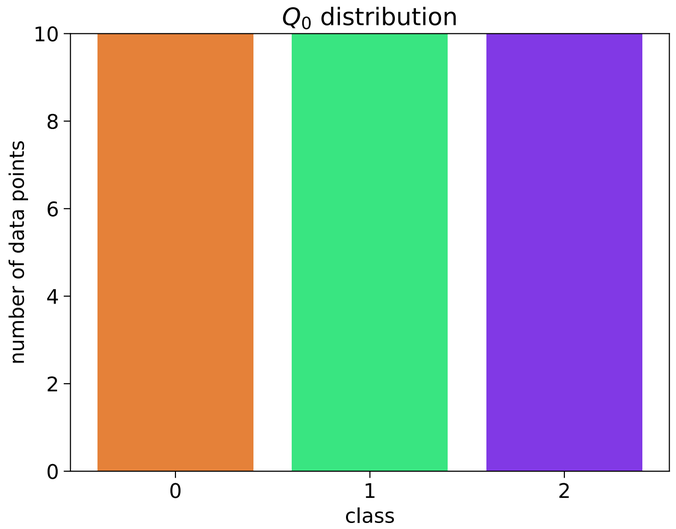
$n_0 = \text{len}(Q_0)$

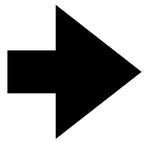
- Solve $\underset{d,t_0}{\text{minimise}}\, L_0$ to get the best feature and threshold for the split

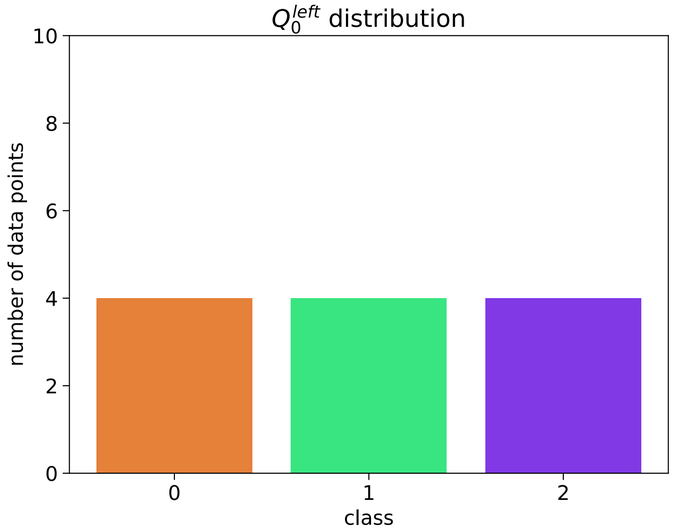


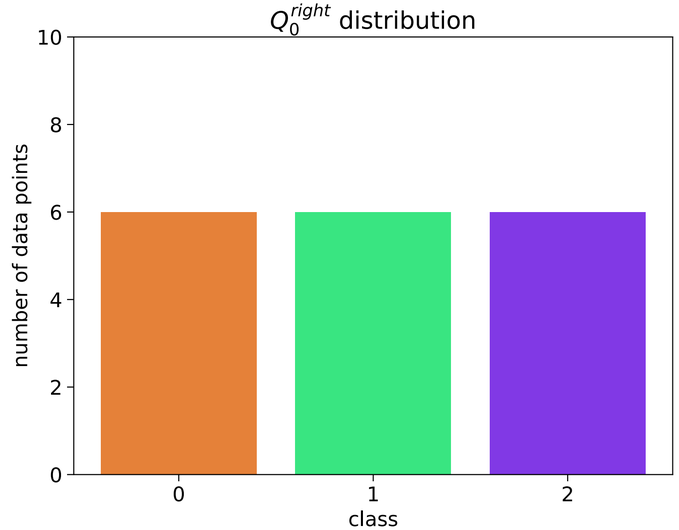
Formulation based on https://scikit-learn.org/stable/modules/tree.html#tree-algorithms

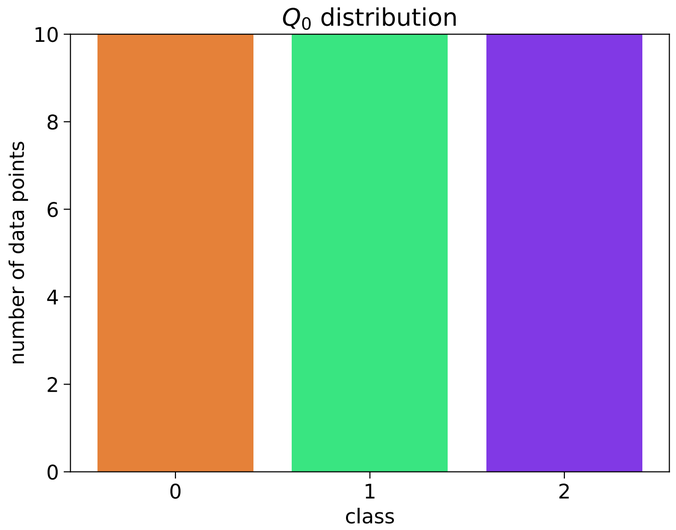
Which split is good and which is bad?

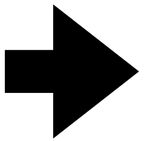


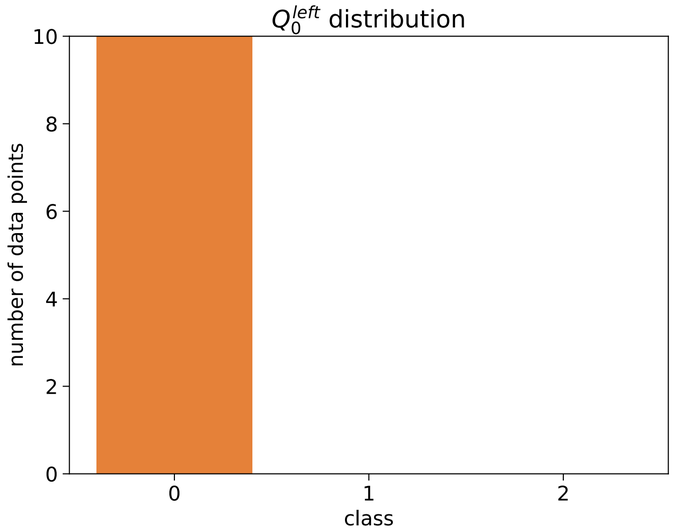


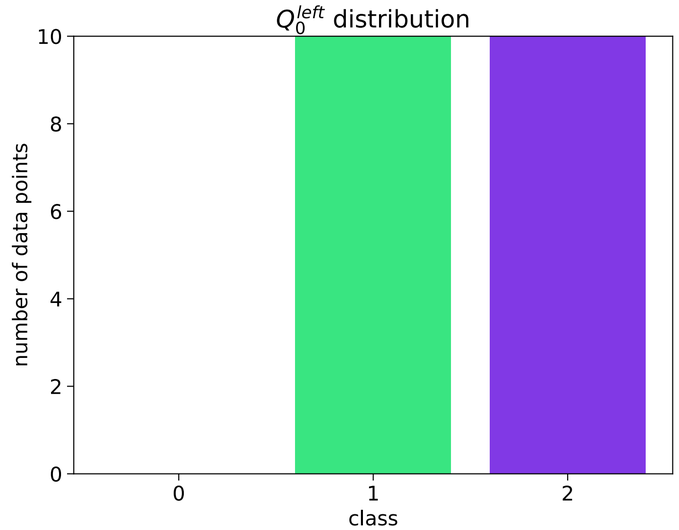










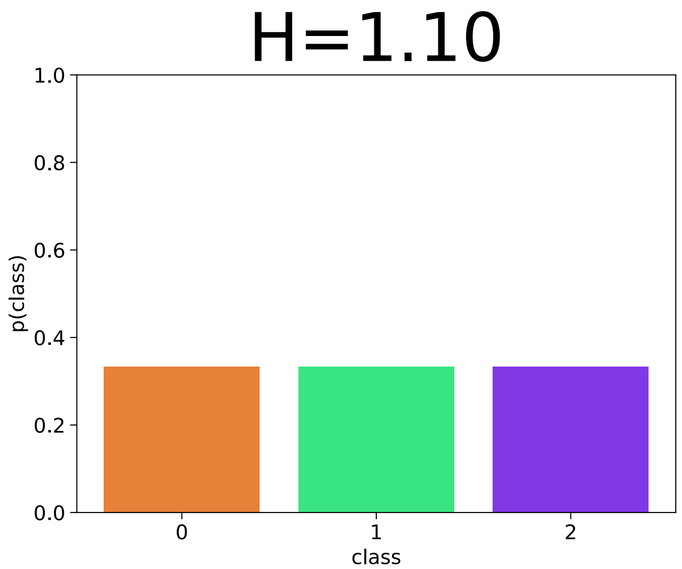


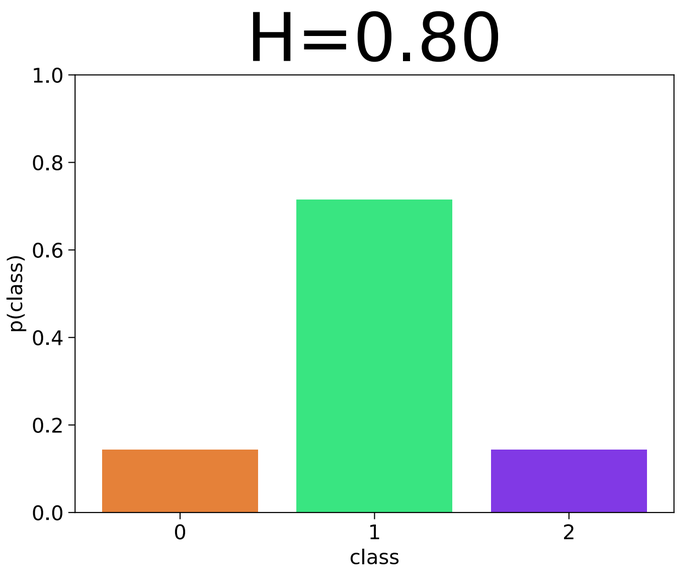
Measuring the quality of a split

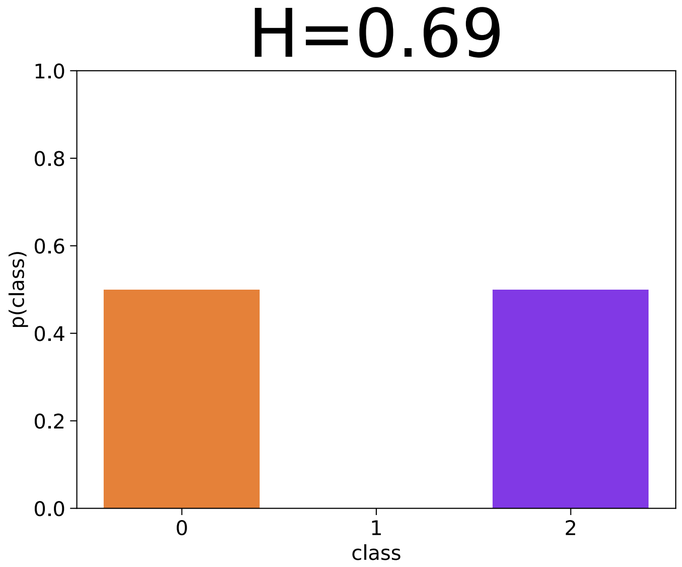
- A good split will separate examples from different classes
- This will make the resulting class distributions **non-uniform**
- Entropy provides a measure of the uniformity of a probability distribution
- For a split, we can divide the number of data points in a class by the total number of data points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

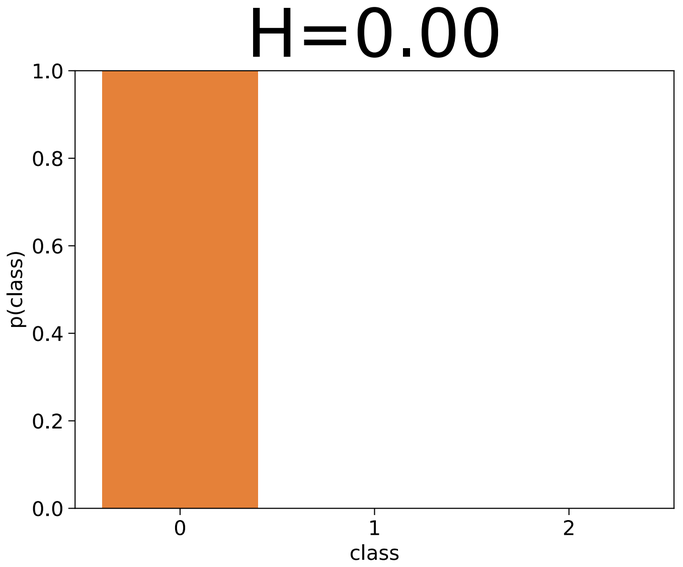
We need to add a small constant to prevent $\log 0$ going to minus infinity

Examples of different entropies



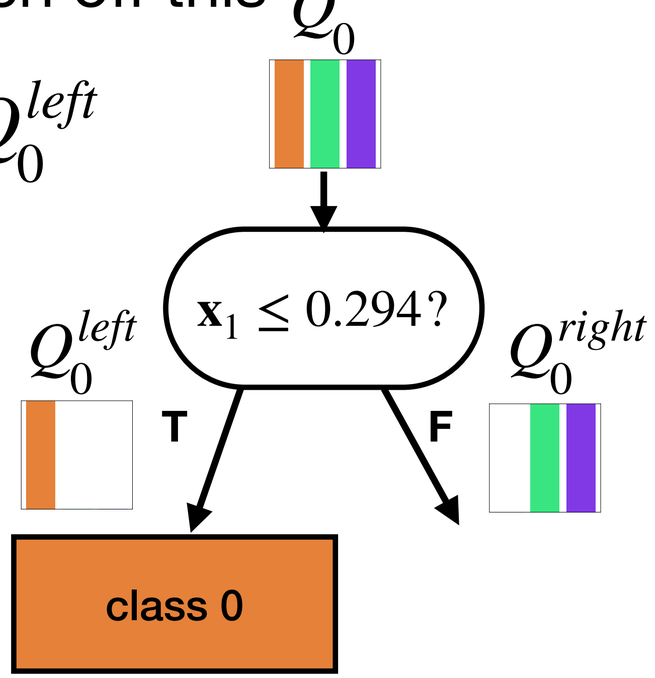






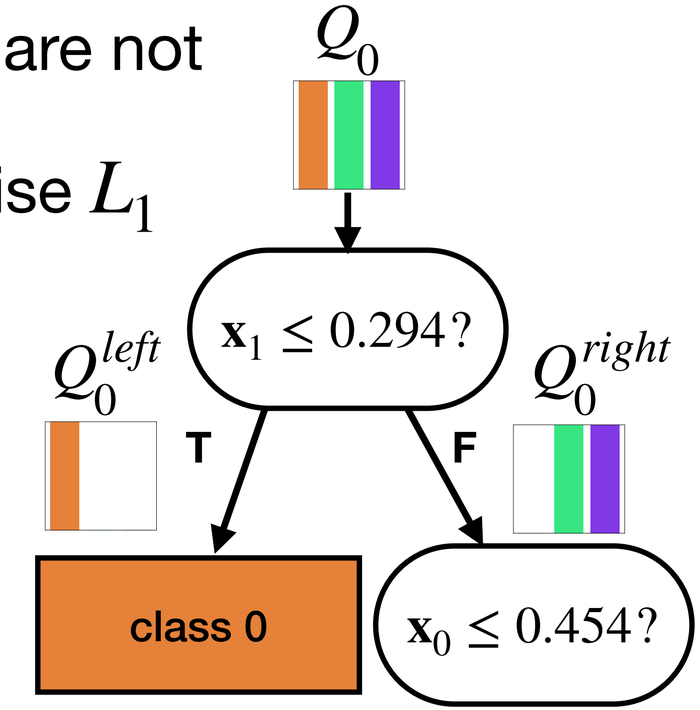
The left child node

- Using H as entropy we can minimise L_0 to get $d = 1$ and $t_0 = 0.294$
- We add a decision node with these values
- Now we need to add a left and right child node that branch off this Q_0
- Let's consider the left child node: we start by looking at Q_0^{left}
- Q_0^{left} only has training points in a single class
- When this happens, we build a leaf node for that class



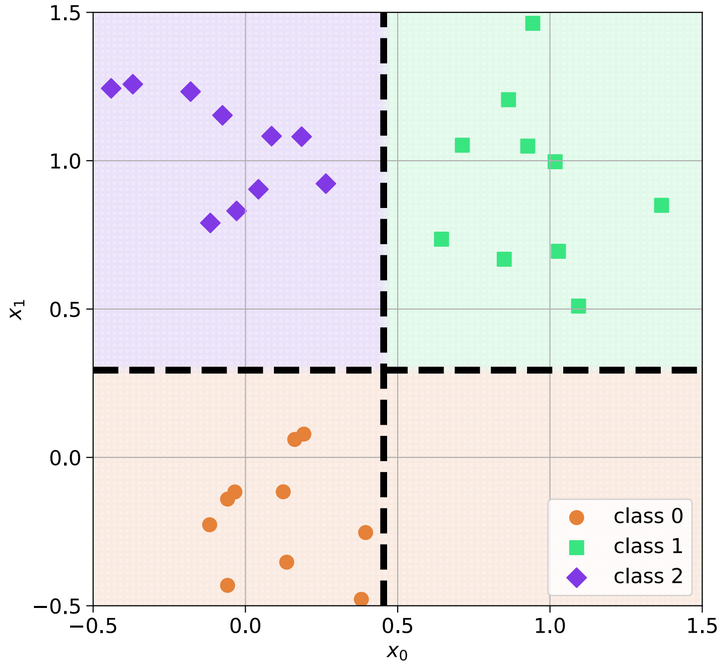
The right child node

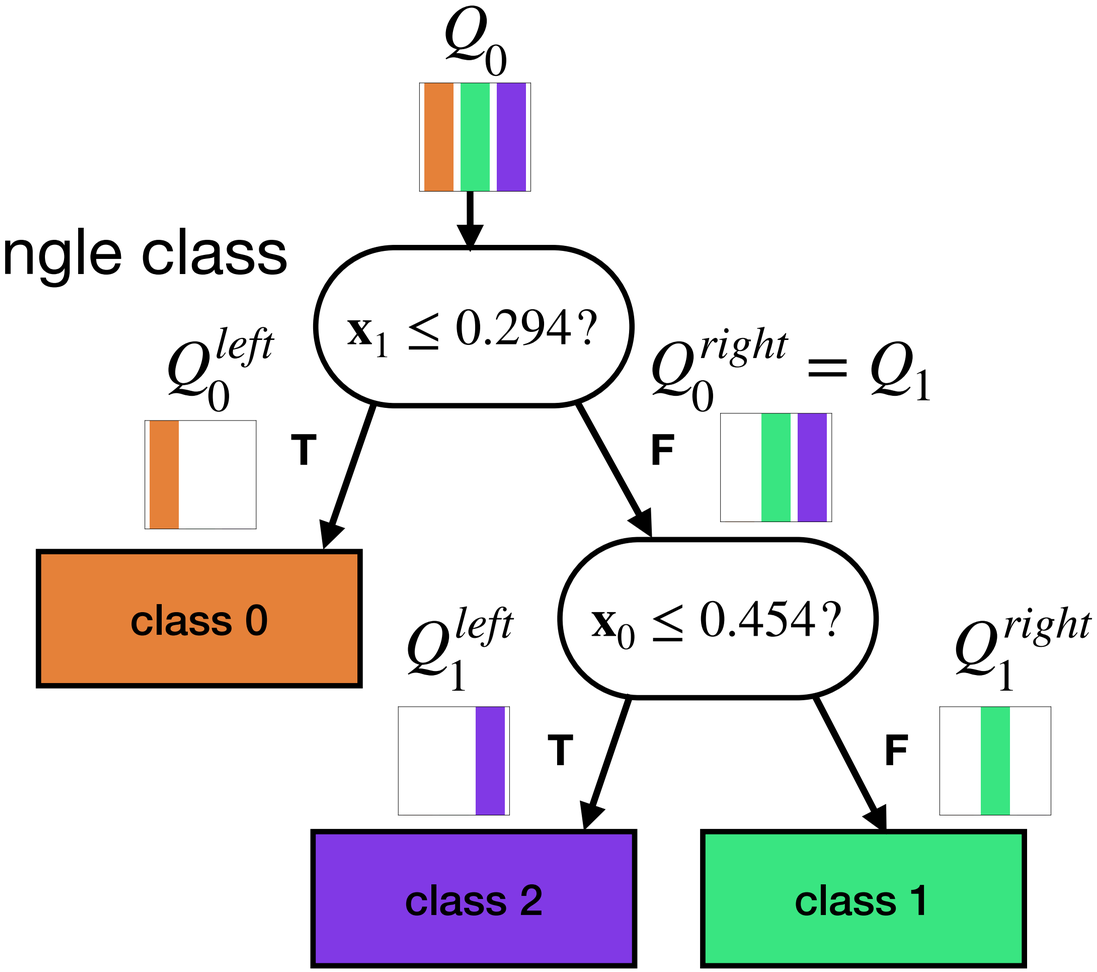
- Let's consider the right child node: we start by looking at Q_0^{right}
- Q_0^{right} has training points in multiple classes
- We first check to see if we are at maximum depth — we are not
- We need to split further, so write $Q_1 = Q_0^{right}$ and minimise L_1
- This gives us $d = 0$ and $t_1 = 0.454$
- We add a decision node with these values



Decision tree complete

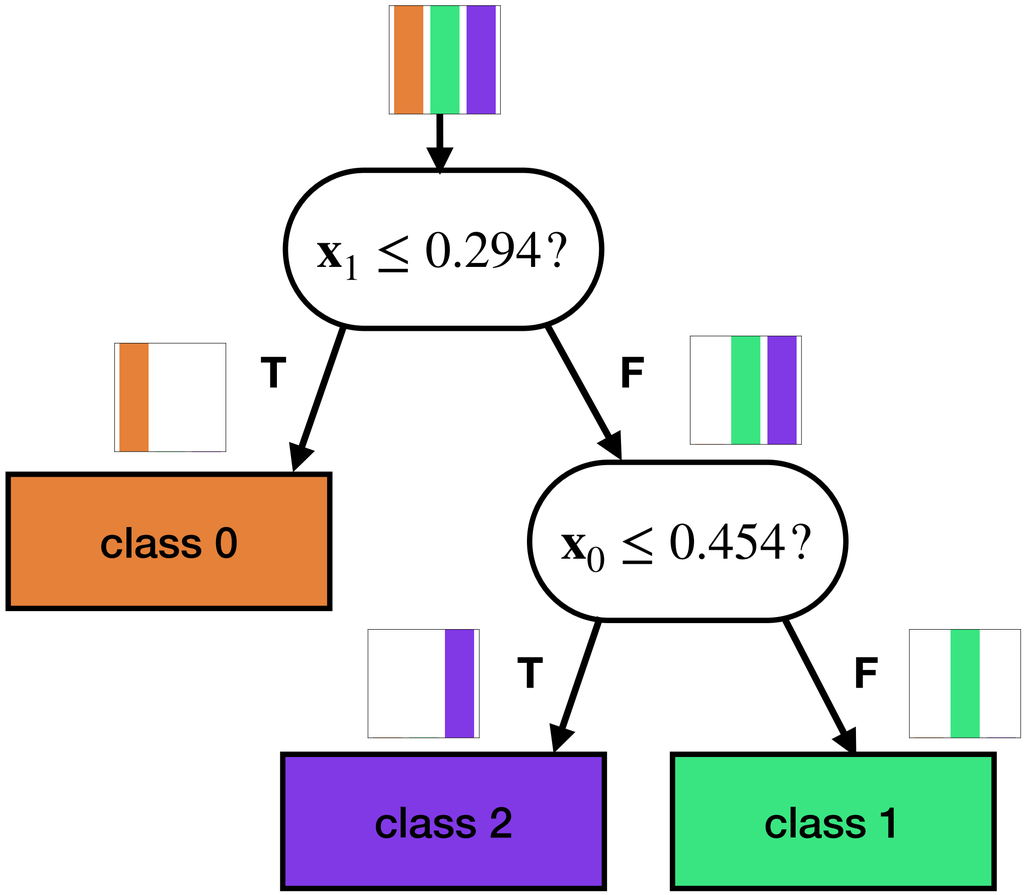
- We now need to add child nodes
- But Q_1^{left} and Q_1^{right} each only contain a single class
- We just add leaf nodes, and we're done

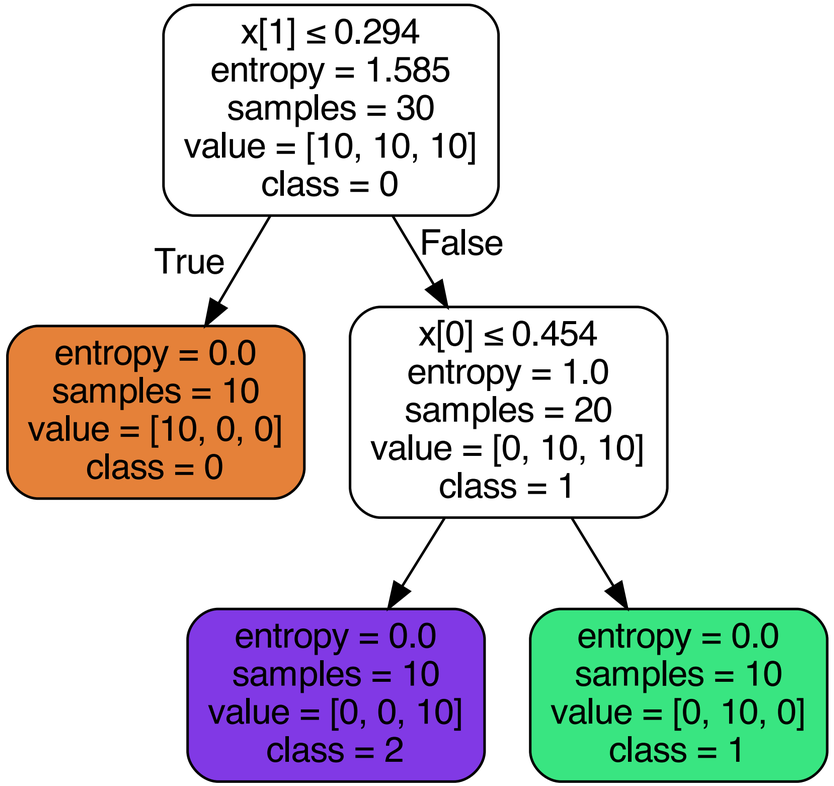




How trees will look in Sklearn

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





Decision tree learning

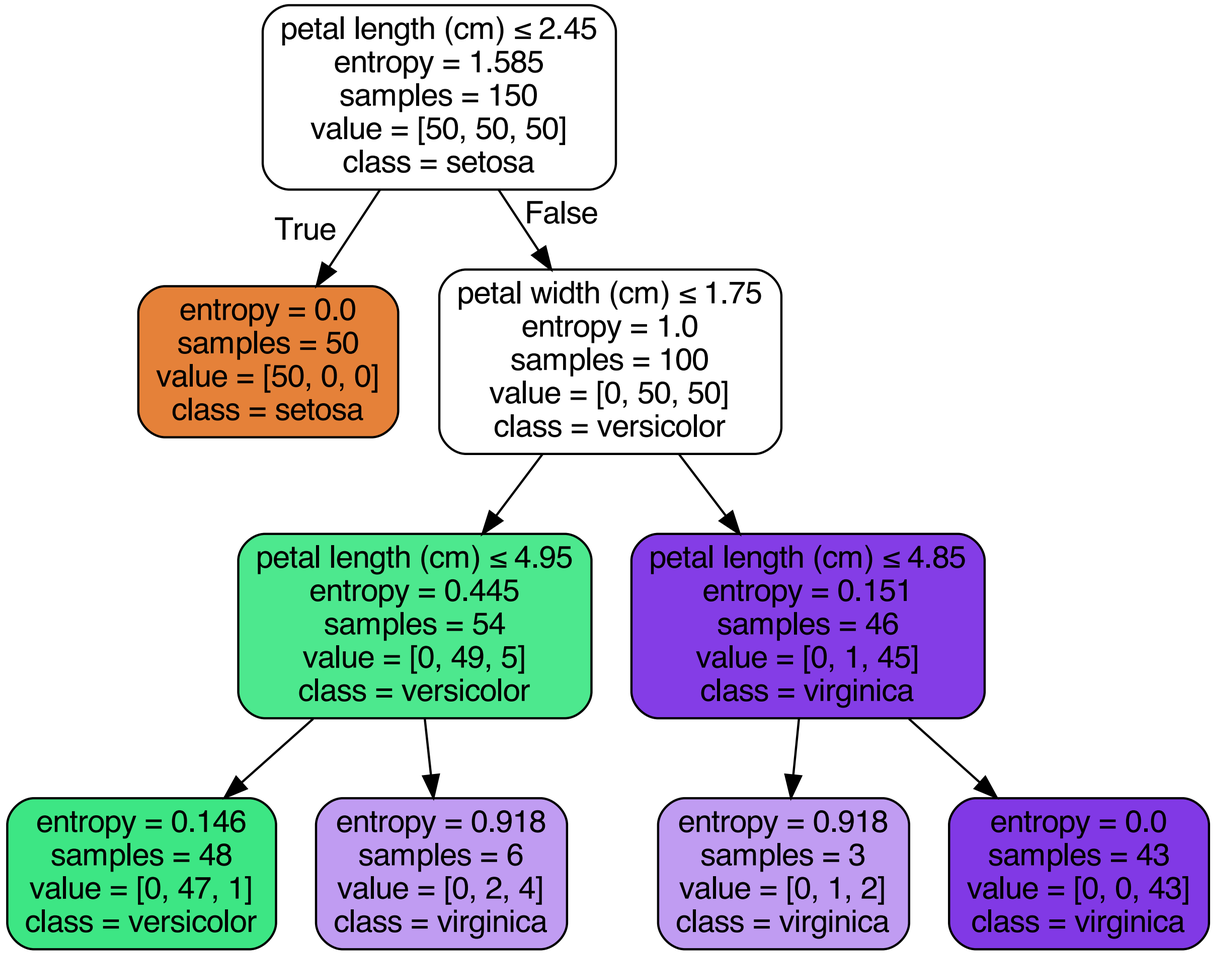
For $Q_0 = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ where $\mathbf{x} \in \mathbb{R}^N$ and $y \in \mathbb{Z}^+_{<K} = \{0,1,\ldots,K-1\}$

- Add decision node with values that minimise L_0
  - Add a left and right child node after this decision node
  - Make each child node a decision node with values that minimises L_1^{left}/L_1^{right} unless at max depth or split only contains 1 class
    - Add a left and right child node after each decision node
    - Make each child node a decision node with values that minimises $L_2^{leftleft}/L_2^{leftright}/L_2^{rightleft}/L_2^{rightright}$ unless…
      - Add a left and right child node after each decision node
      - Make each child node a decision node with values that minimises …

This algorithm is recursive

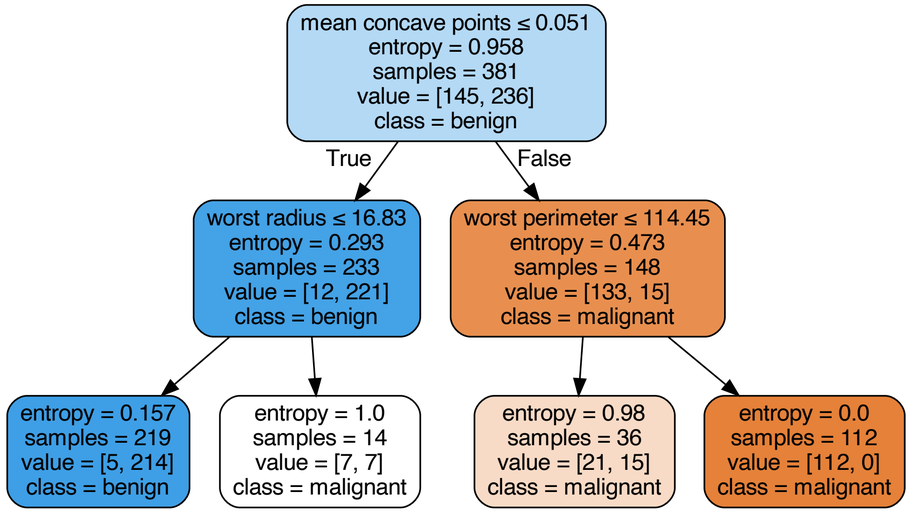
What happens if you reach maximum depth?

- Just create a leaf node that classifies as the highest probability
- To the right we learn a decision tree on the iris dataset with a max depth of 3



Malignancy classification with a small tree

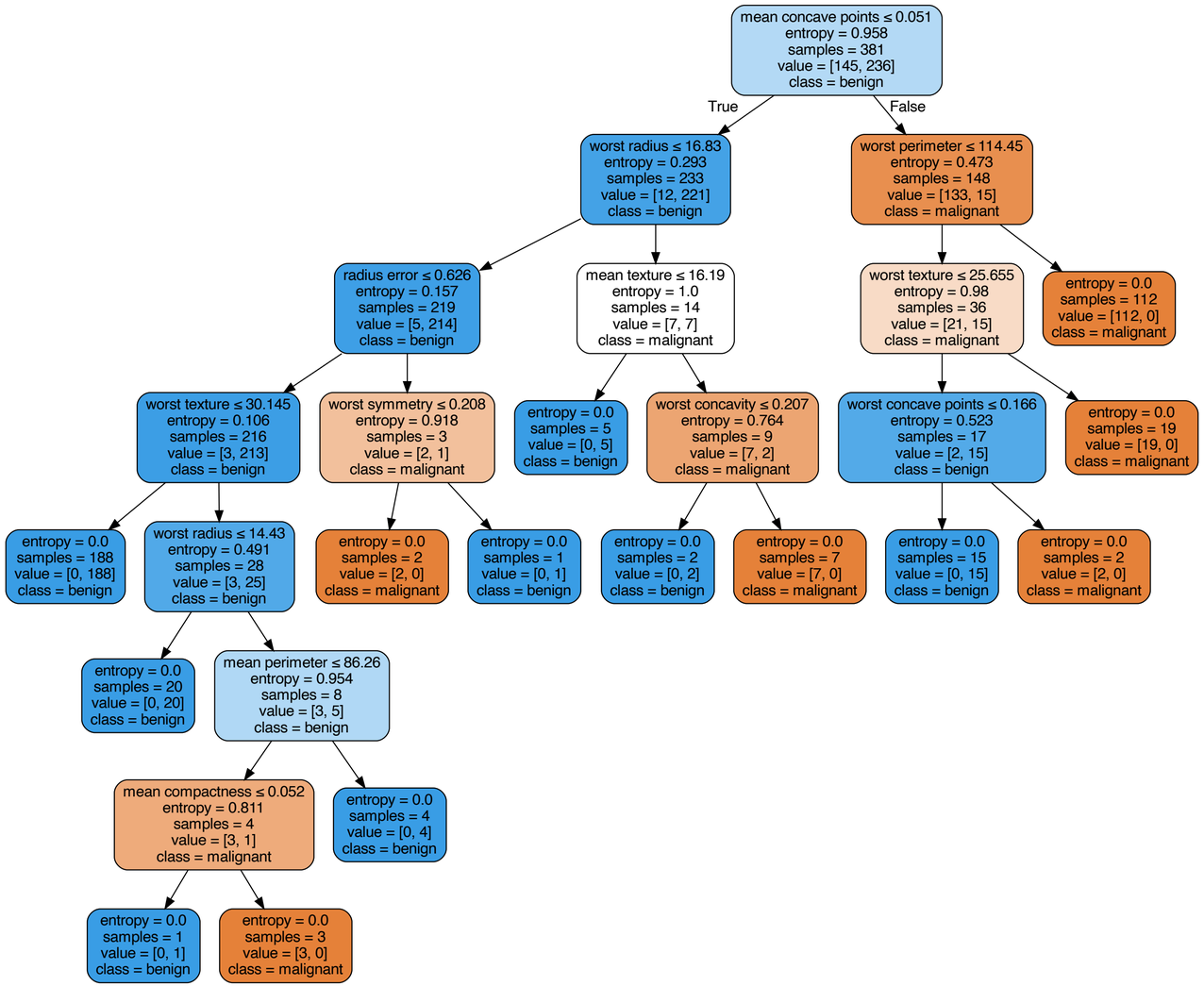
- Breast Cancer Wisconsin dataset has 569 data points $\mathbf{x} \in \mathbb{R}^{30}$ with binary class labels y (malignant / benign)
- Features are measurements from a digitised image of a fine needle aspirate of a breast mass
- Let's split 66%/33% train/val and learn a depth 2 decision tree



This is interpretable and achieves a val 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 val accuracy of 95.2%

This is hard to interpret

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

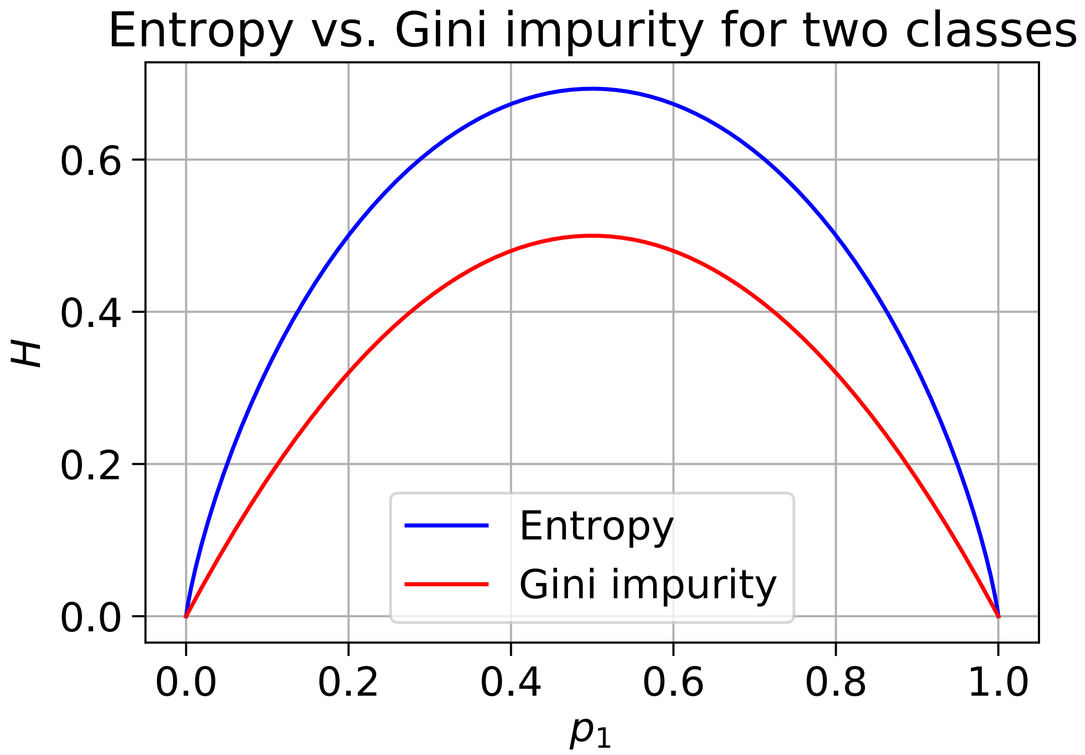
Gini impurity

- For some Q_m we write $L_m = \frac{n_m^{left}}{n_m} H(Q_m^{left}(d, t_m)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(d, t_m))$
- Here H is is entropy $-\sum_k p_k \log p_k$ and we solve $\underset{d,t_m}{\text{minimise}}\, L_m$
- This calculation is being performed a lot. Can we make it cheaper?
- **Yes**. Use Gini impurity $\sum_k p_k(1 - p_k)$ instead of entropy

Don’t have to compute $\log$ any more which is expensive

Gini impurity vs. Entropy

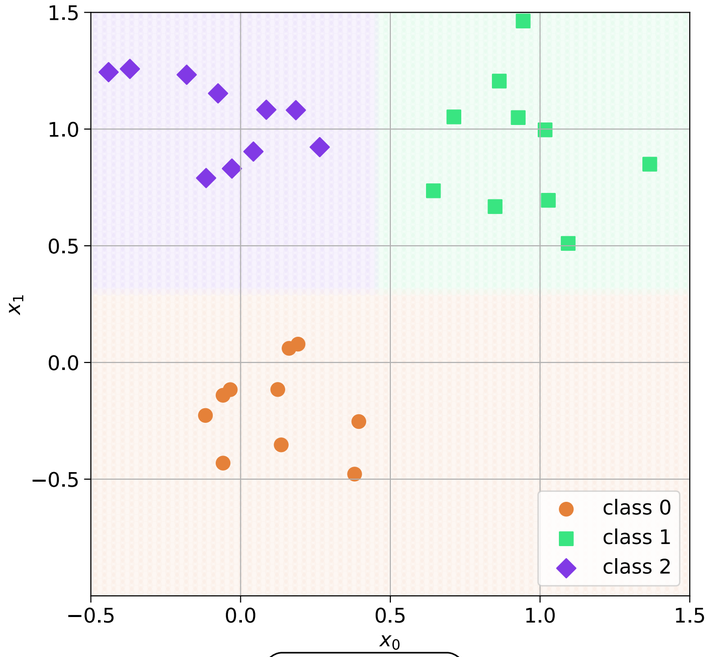
Gini impurity is the probability of incorrectly classifying a new data point labelled according to the class distribution of that split

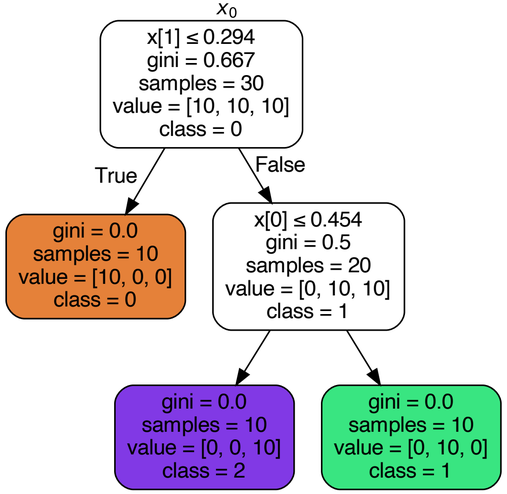


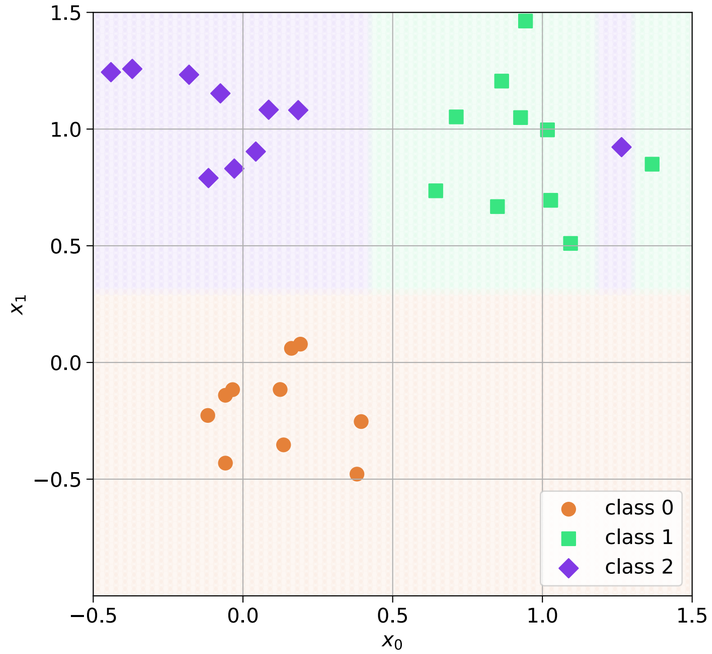
Shapes are very similar

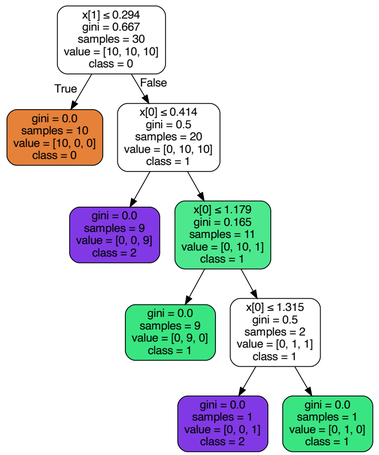
Choice has minimal effect on performance

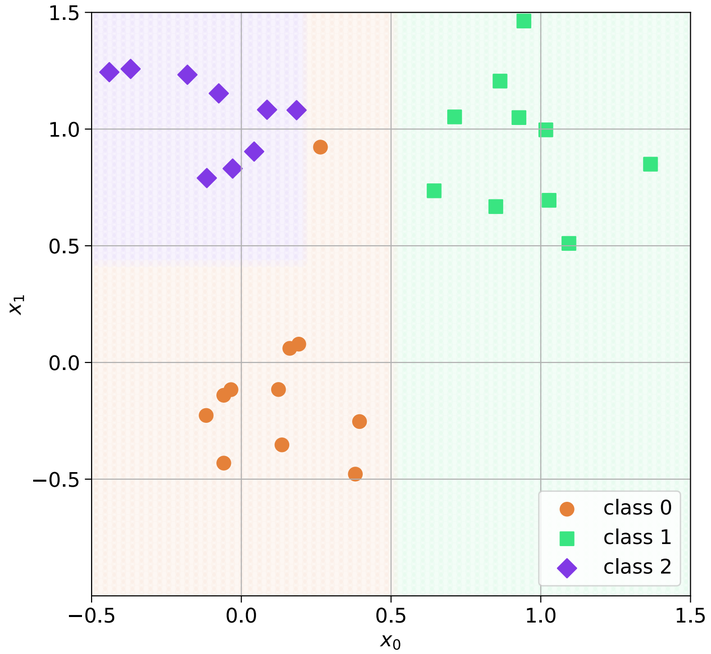
Decision trees tend to overfit and are unstable

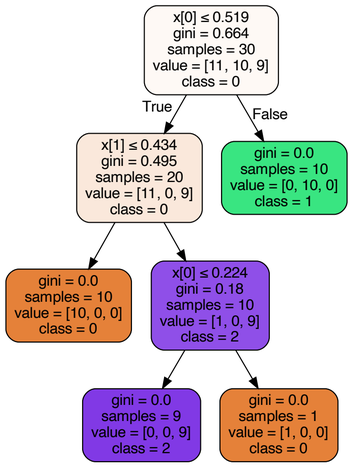






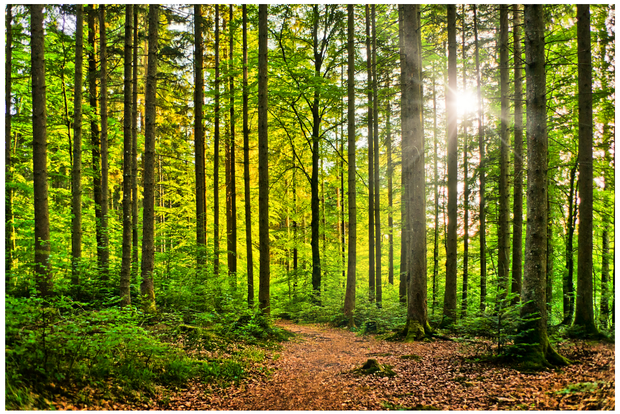






Random Forests

- 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 lots of trees trained on different permutations 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

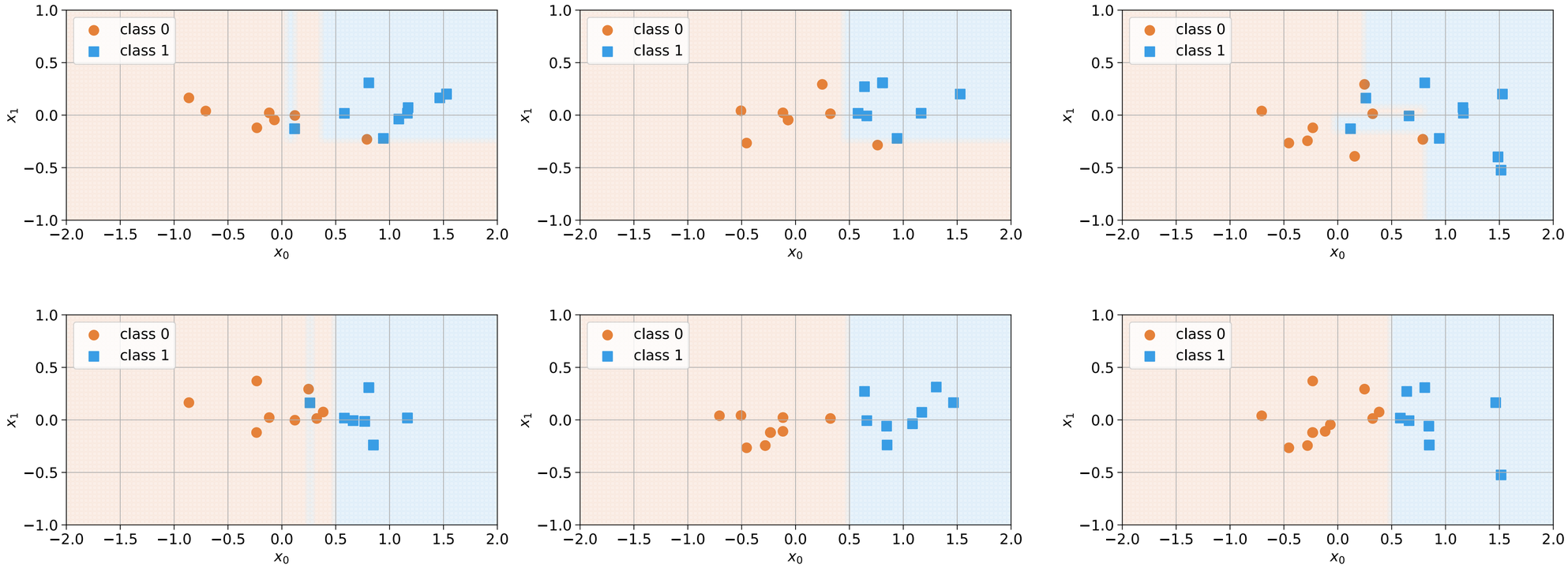


A random forest is an **ensemble model** that consists of multiple (usually 100) decision trees

Bootstrap aggregation (bagging)

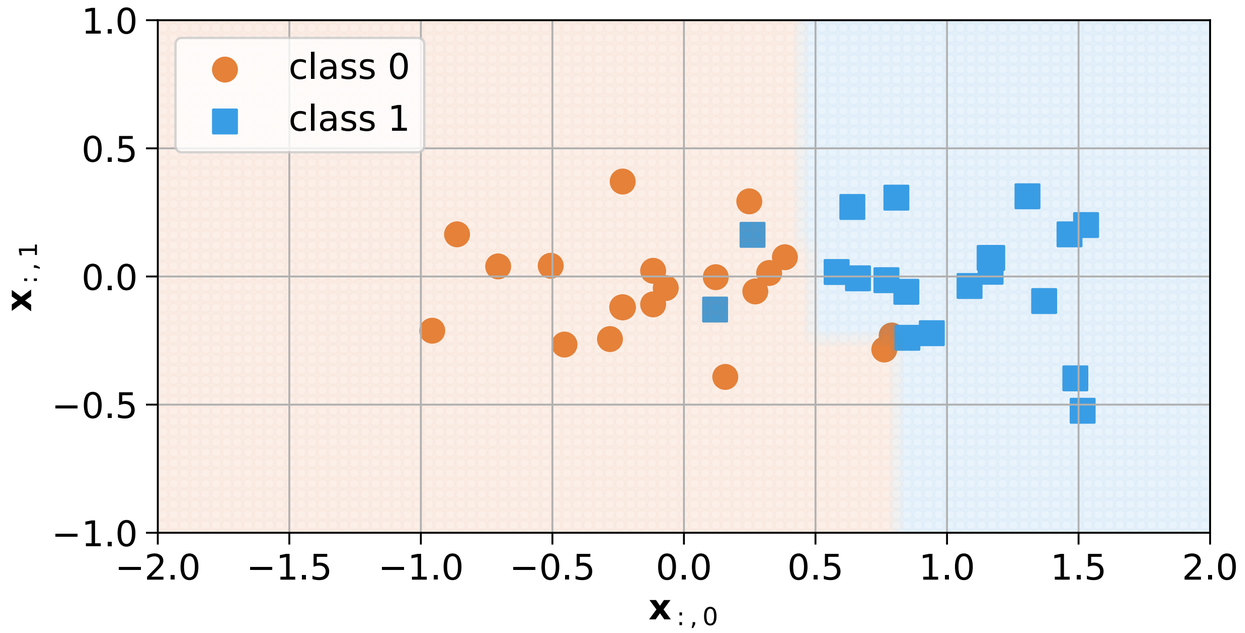
This isn't exclusive to decision tree based classifiers

- We have dataset $Q_0 = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ and want to train t trees
- For each tree, sample M data points at random **with replacement** and train



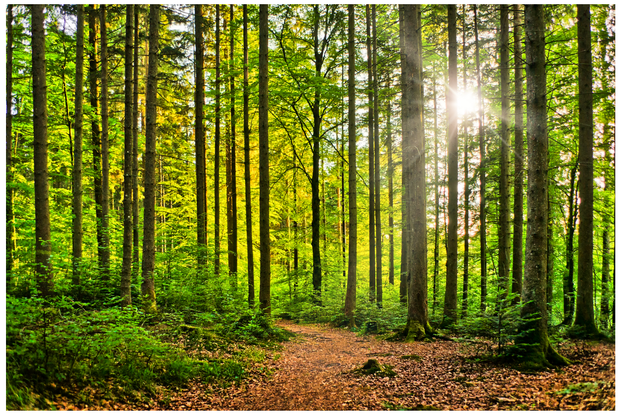
Evaluating the ensemble

- Put a test 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



Is that a random forest?

- Not quite. A random forest is bagging **and feature subsampling at each node**
- For each tree, you perform bagging but also only select a subset of available features **at each node** (this is more meaningful in higher dimensions!)



A random forest is an ensemble of decision trees

Each tree is trained on a random sample of the training data with replacement

At each node in each tree, only a subset of features are available

The 20 newsgroups dataset

- 18k (11k train, 7k test) posts from 20 different newsgroups (think subreddits)
- The task is given a post, classify which news group it belongs to
- A simple way to represent text is as a histogram of word counts (a bag of words)
  e.g. [# “I”, # “like”, # “sausage”, # “hate”]

I like sausage	I hate sausage	sausage sausage
$[1 \quad 1 \quad 1 \quad 0]^\top$	$[1 \quad 0 \quad 1 \quad 1]^\top$	$[0 \quad 0 \quad 2 \quad 0]^\top$

- For 20 newsgroups this gives us $\mathbf{x} \in \mathbb{R}^{130107}$ and we have $y \in \mathbb{Z}^{+}_{<20}$
- With a decision tree we get a test accuracy of **55.7%**
- With a random forest of 100 trees we get a test accuracy of **75.5%**

Coursework 2 (25% of course mark)

- You will perform data analysis and machine learning on “**Sentiment Soup”**: a dataset of 100k text samples drawn from different sources
- You should write a **4-6 page report with an appendix containing code** where you:
  1. Explain what sentiment analysis is and its importance
  2. Summarise and visualise “Sentiment Soup”
  3. Create a set of classification tasks
  4. Train and evaluate classifiers on those tasks, examining different text representations
  5. Select models and examine how they perform on external data that you have found
- The full brief, dataset, submission instructions, and the marking rubric are available on Learn under the “Assessment” tab (after 0950 today). **Deadline 28/3 @ 1600**

Summary

- We have looked at k-nearest neighbours
- We have learnt about decision trees and how they are trained
- We have seen how entropy and Gini impurity allow for good splits
- We have seen how decision trees can overfit
- We have learnt about bootstrap aggregation as a way to form model ensembles
- We have found out that random forests consist of bootstrap aggregation and feature subsampling