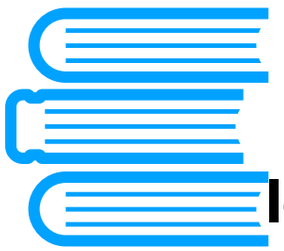
Decision Trees and Ensemble Learning

Supervised

Hisham Ihshaish

June 2021

Bristol, UK



Review + notes

1

```
from sklearn.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_ != 0)))
```

```
Training set score: 0.29
Test set score: 0.21
Number of features used: 4
```

2

```
lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso001.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso001.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso001.coef_ != 0)))
```

```
Training set score: 0.90
Test set score: 0.77
Number of features used: 33
```

3

```
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso00001.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso00001.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso00001.coef_ != 0)))
```

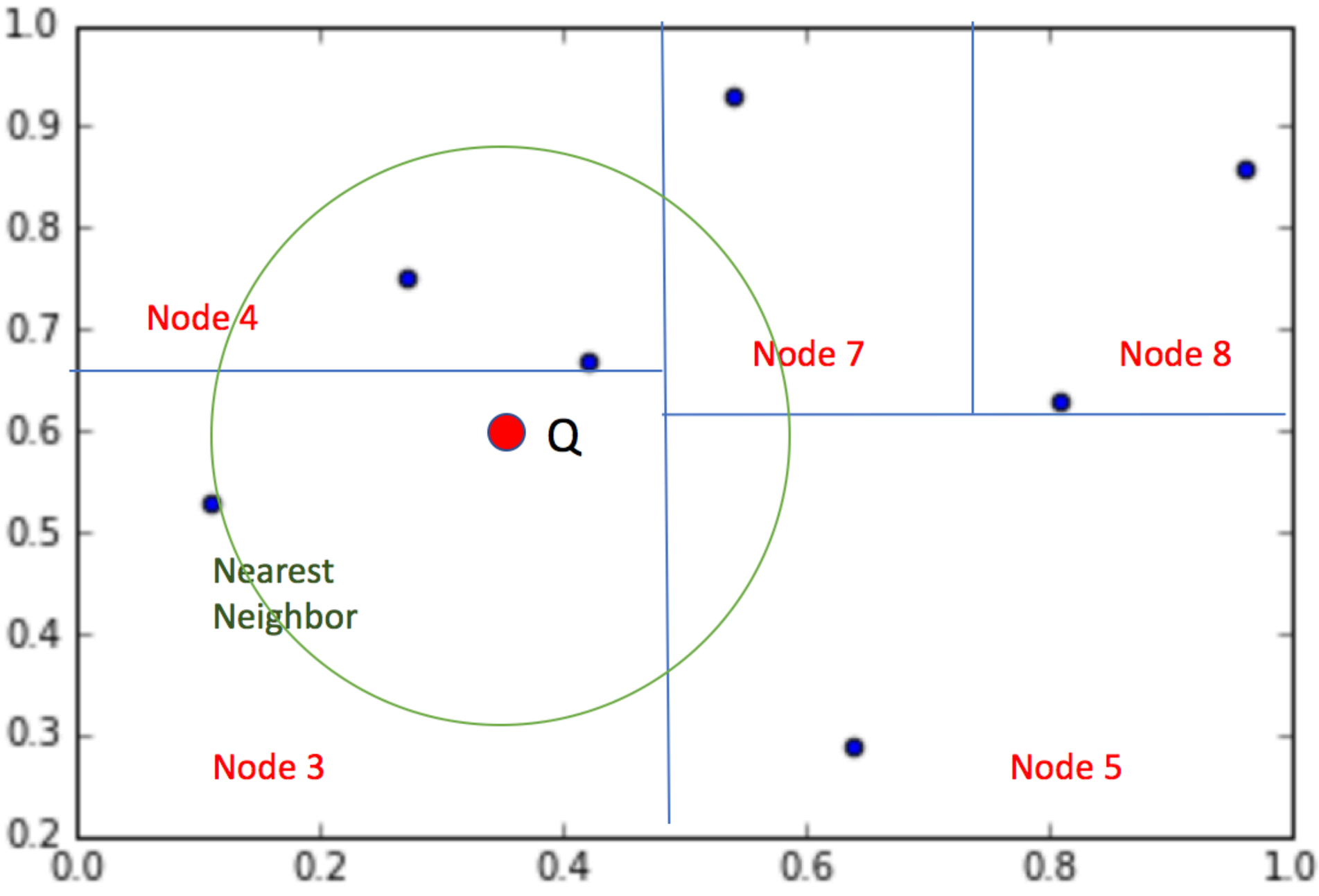
```
Training set score: 0.95
Test set score: 0.64
Number of features used: 94
```

Agenda

- Decision Trees Induction
  - Context
  - Estimation/“design”
  - Properties
- Ensemble Learning
  - Bagging (mainly - I’m hoping to cover more next week)
- Summary
- Tutorial

from kNN lecture

k-d trees approximation



Decision Trees

- Simple (or maybe the simplest), yet **can be** powerful, or amongst most powerful learners, for classification (mainly) and regression problems.

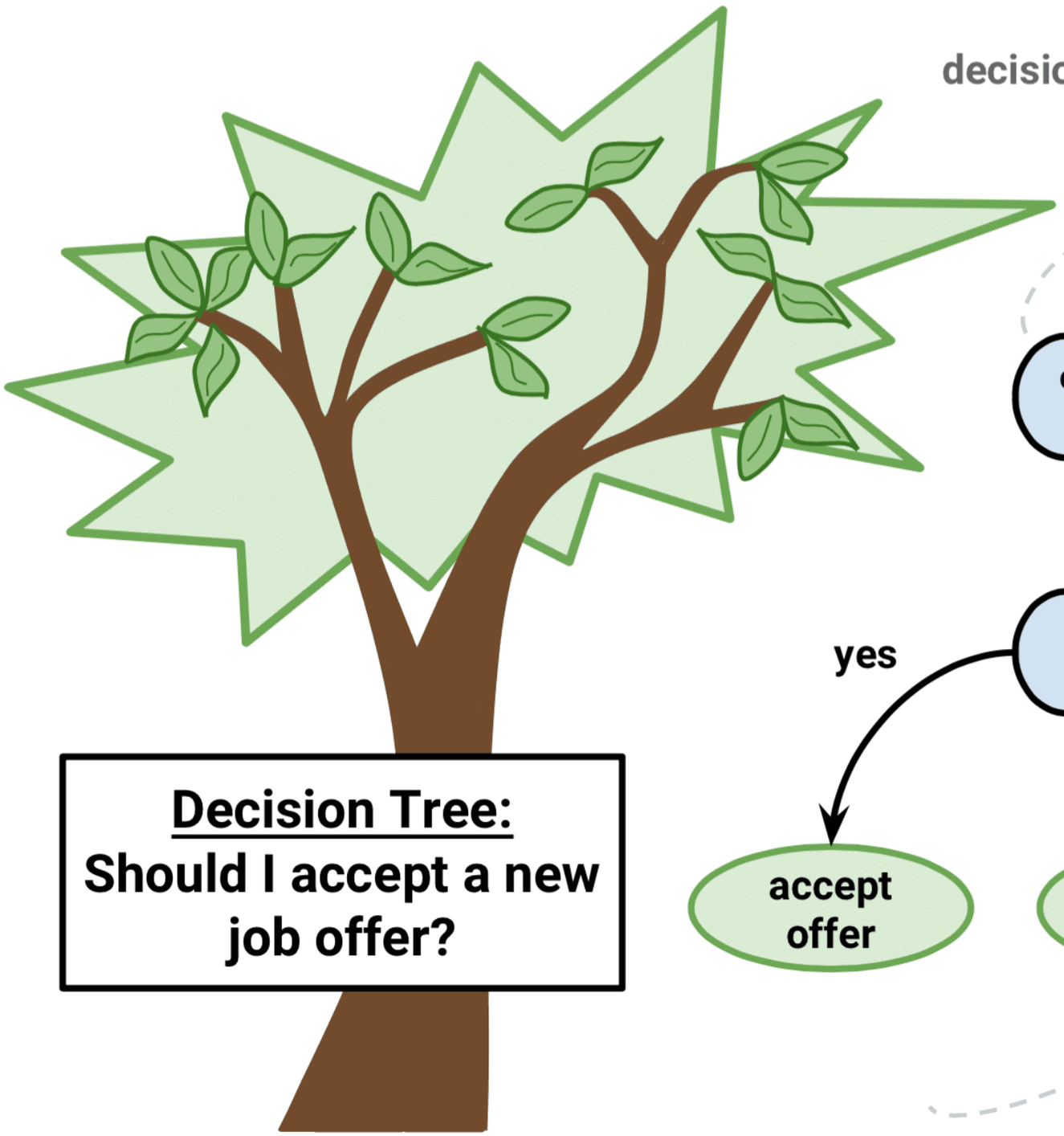
Decision Trees

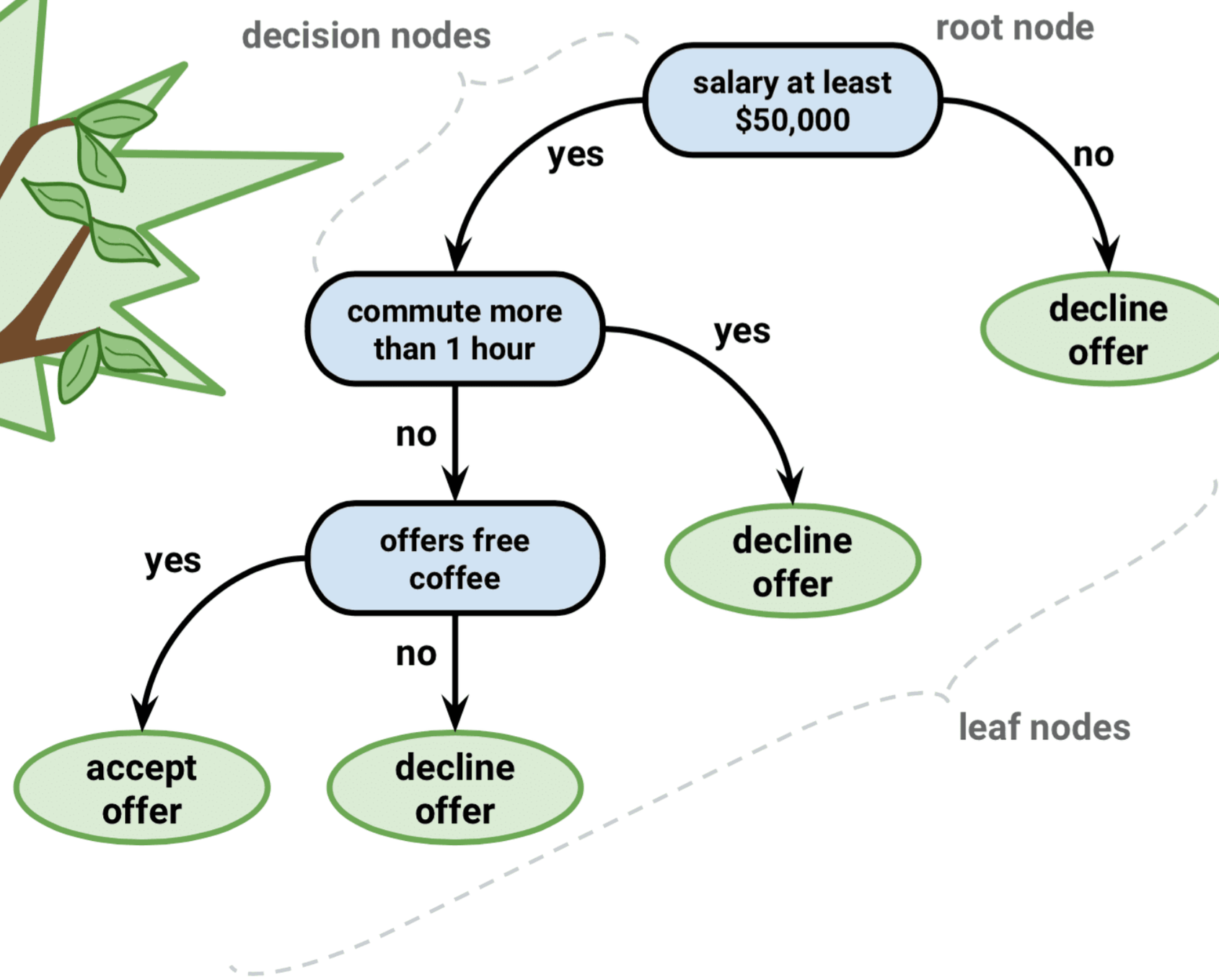
- A very popular model for both **regression** and **classification** tasks — regardless of linearity (huh!)
- Non-parametric.

Decision Trees

- Random Forests (*a variant* - later on in the lecture) is one of the most used models for predictive analytics in practice - because they’re good ;)

Decision Trees



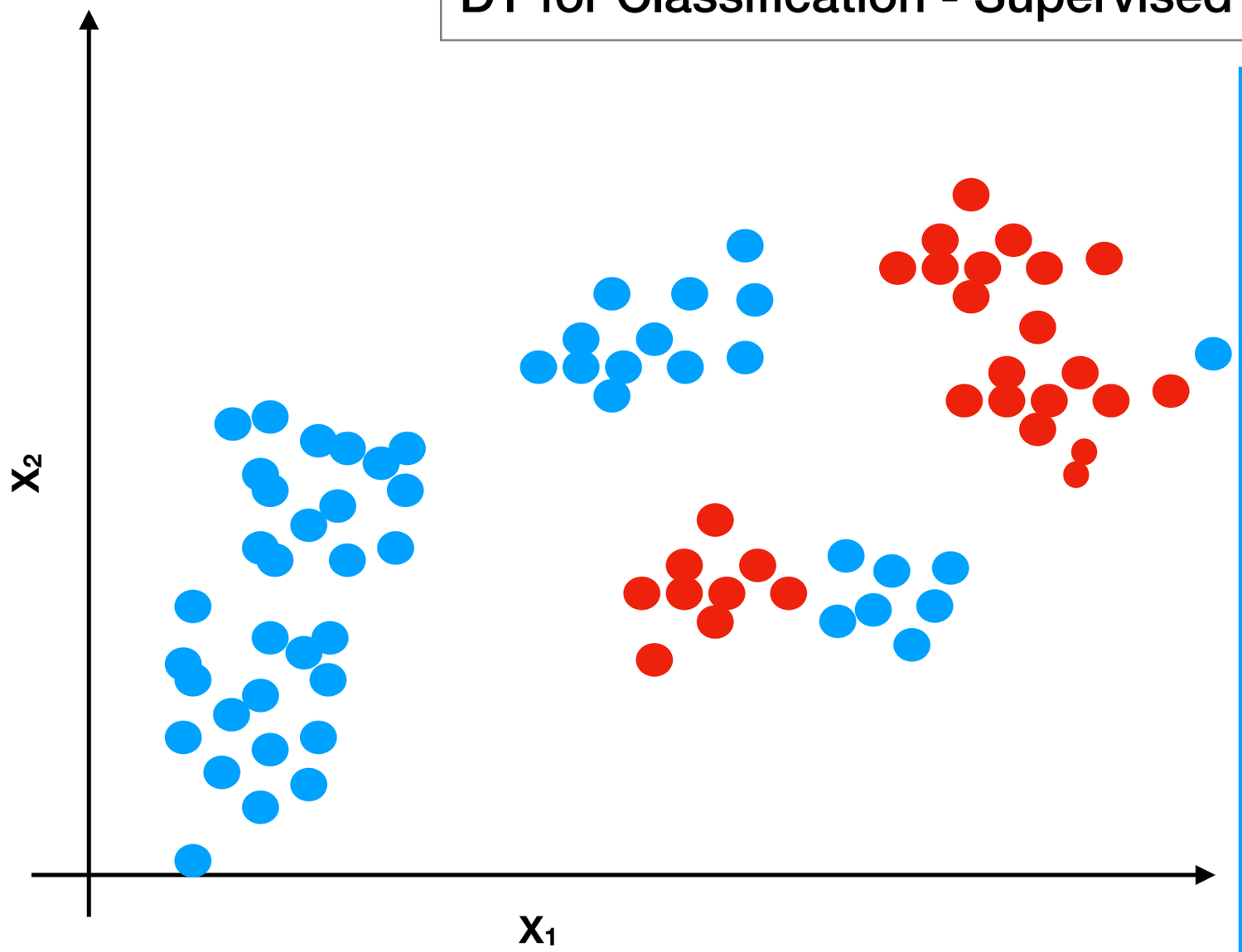


- Decision Trees (simplified) is the task of learning a hierarchy of **if/else** questions, leading to a decision (leaf).

Image credit: www.packtpub.com

DT for Classification

DT for Classification - Supervised Learning



➡ Data:

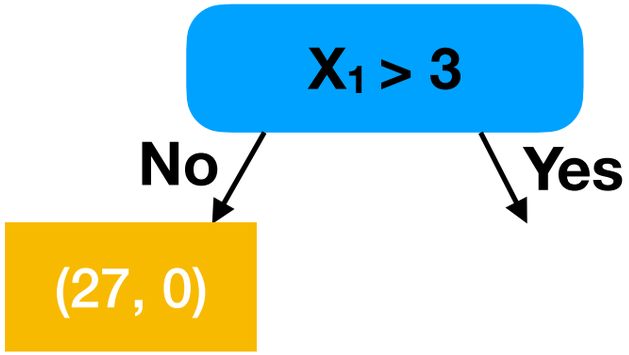
$((x_{i1},x_{i2}), y_i)$, i =1, 2, …,n

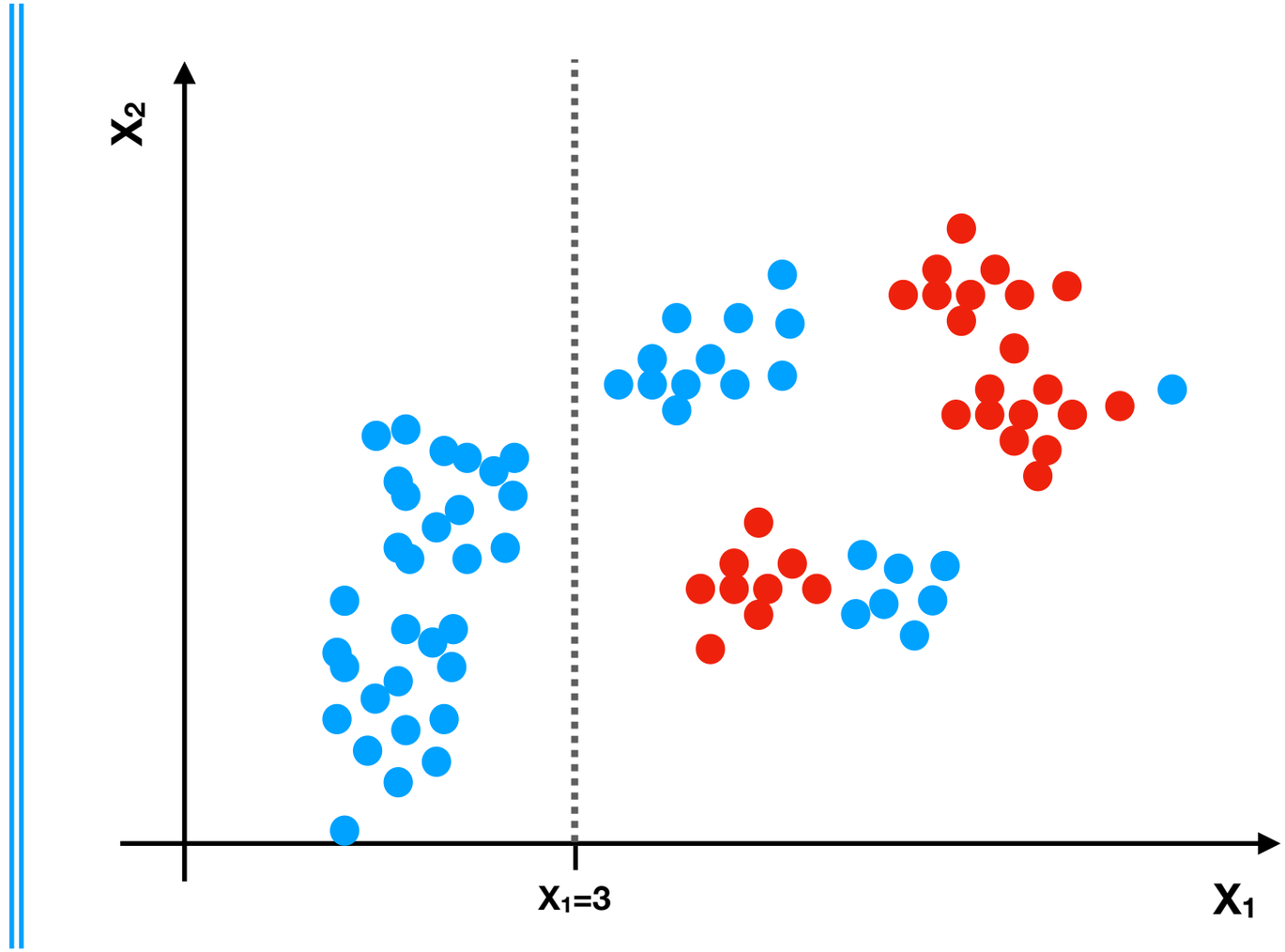
➡ Task:

E (Y: colour | X: x_1,x_2)

such that ***error*** is minimised at each Leaf.

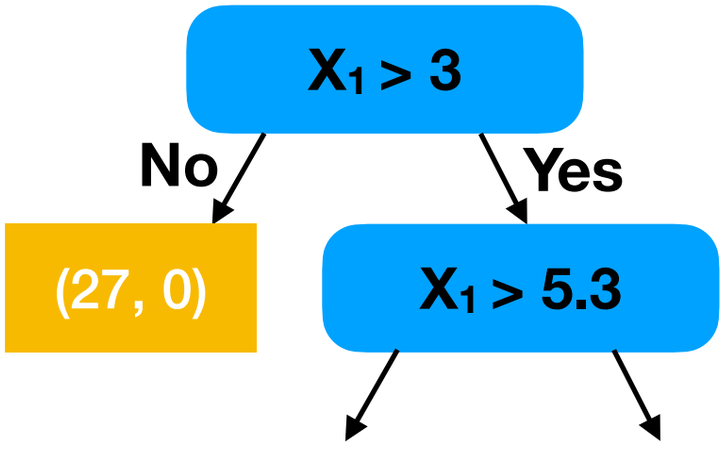
DT for Classification

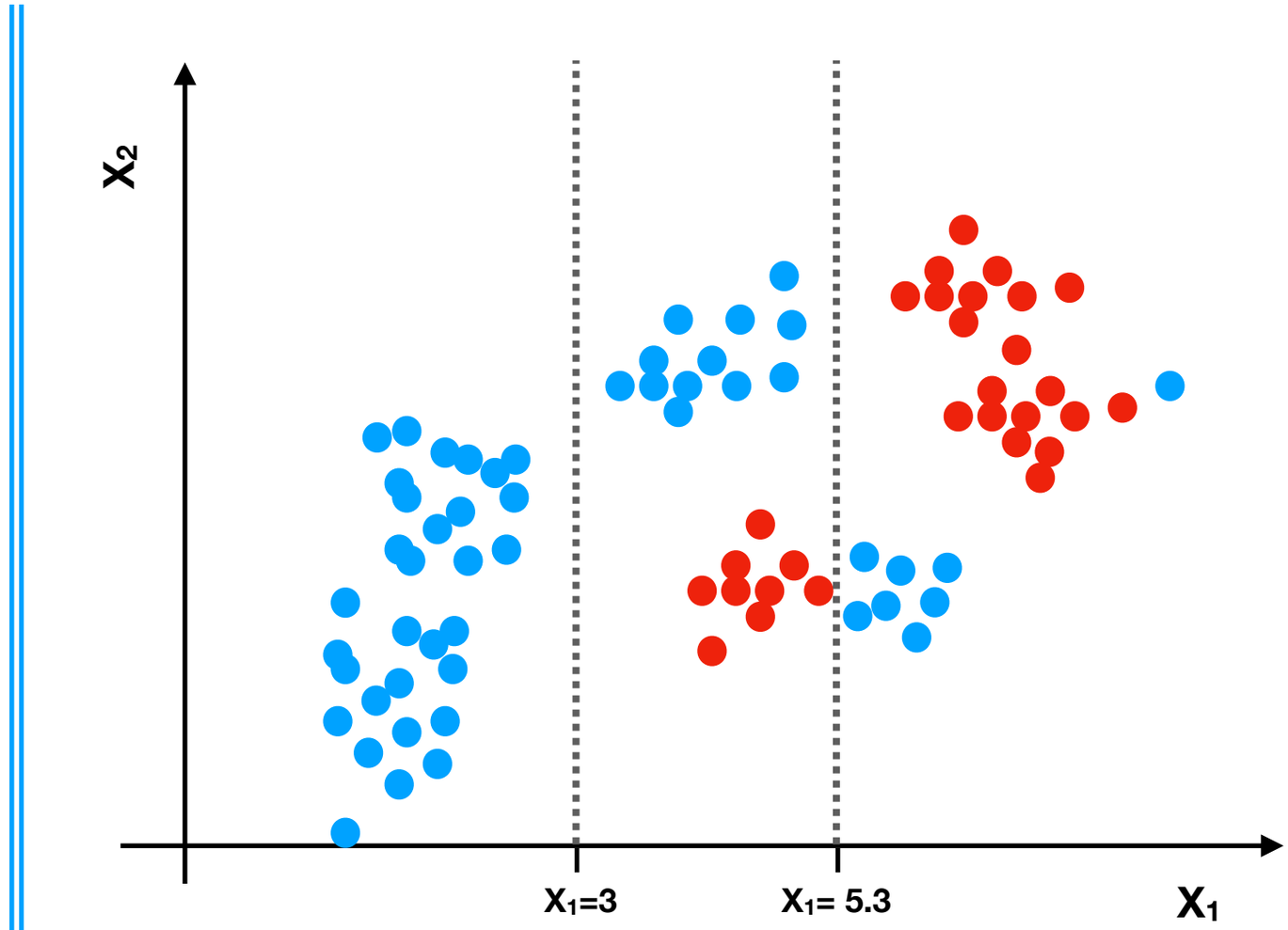




- Depth = 1
- Decision Attribute: X_1

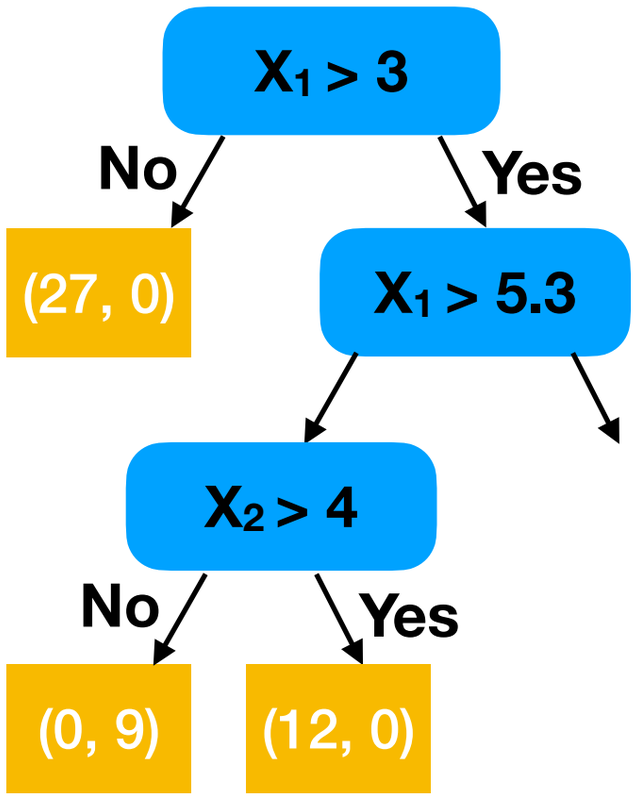
DT for Classification

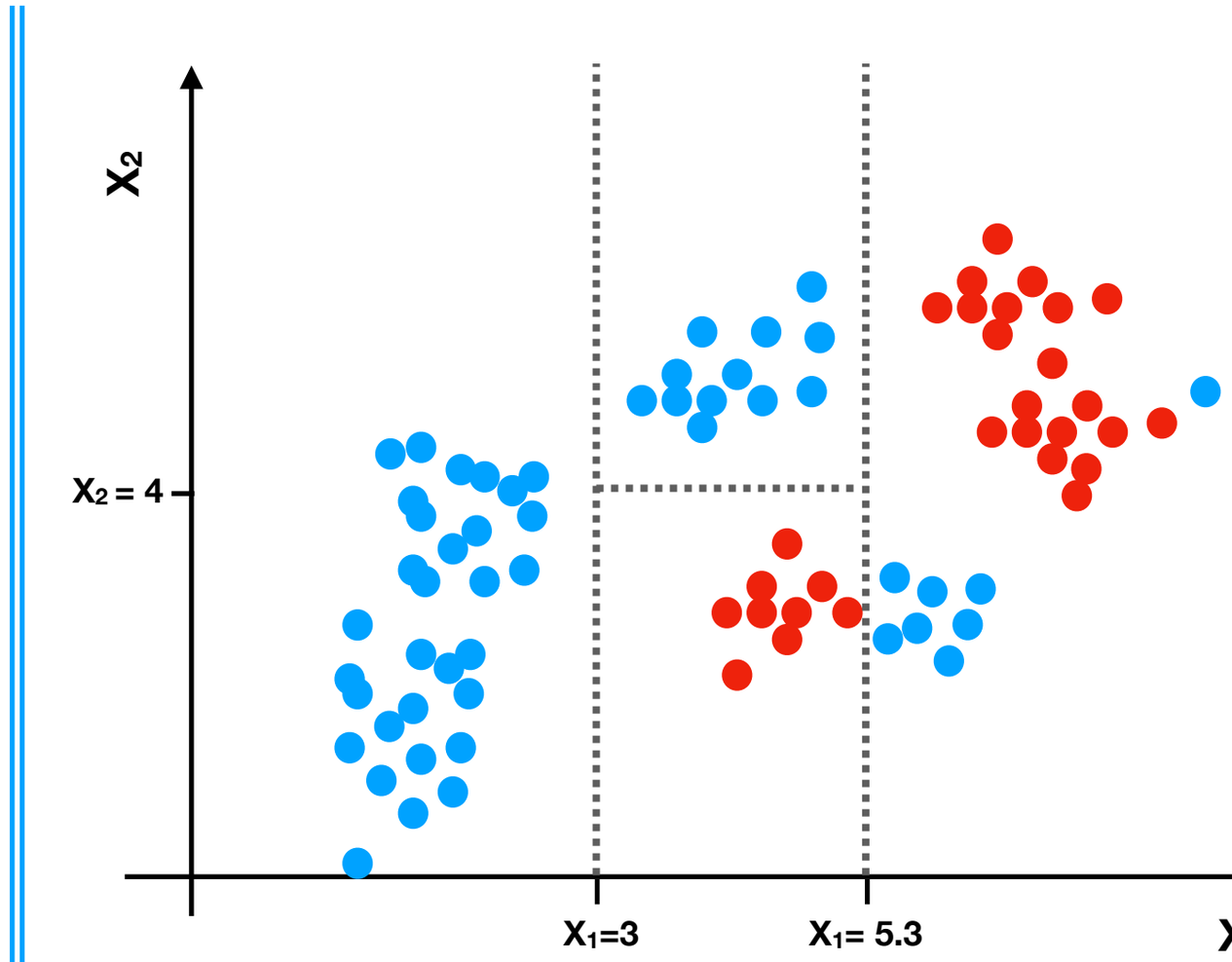




- Depth = 2
- Decision Attribute: X_1

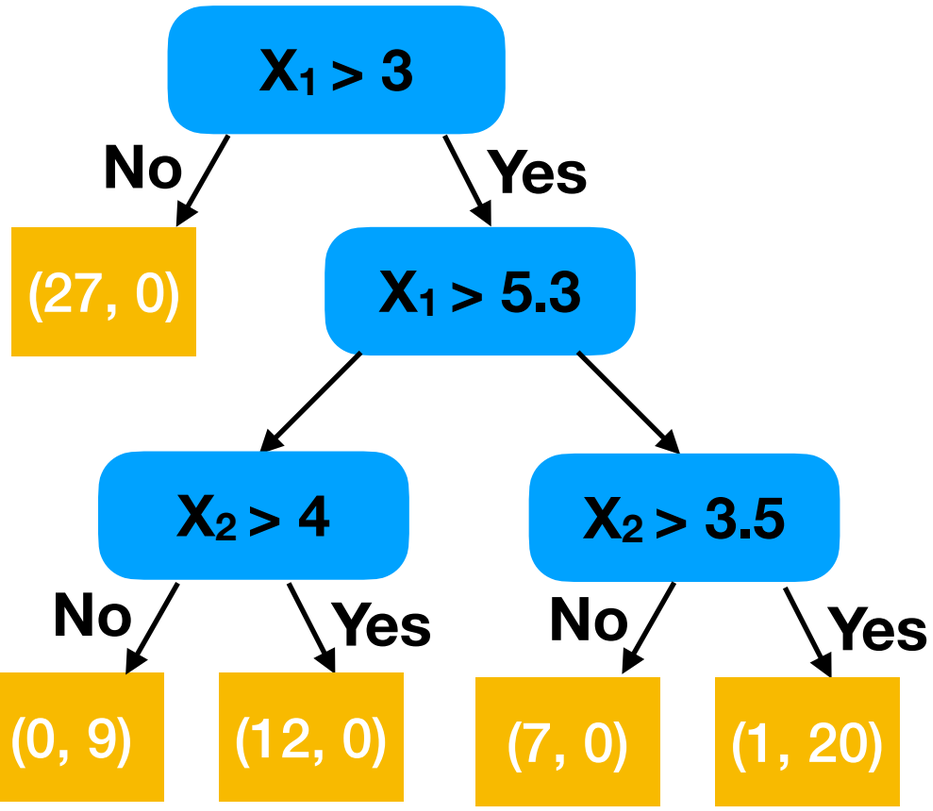
DT for Classification

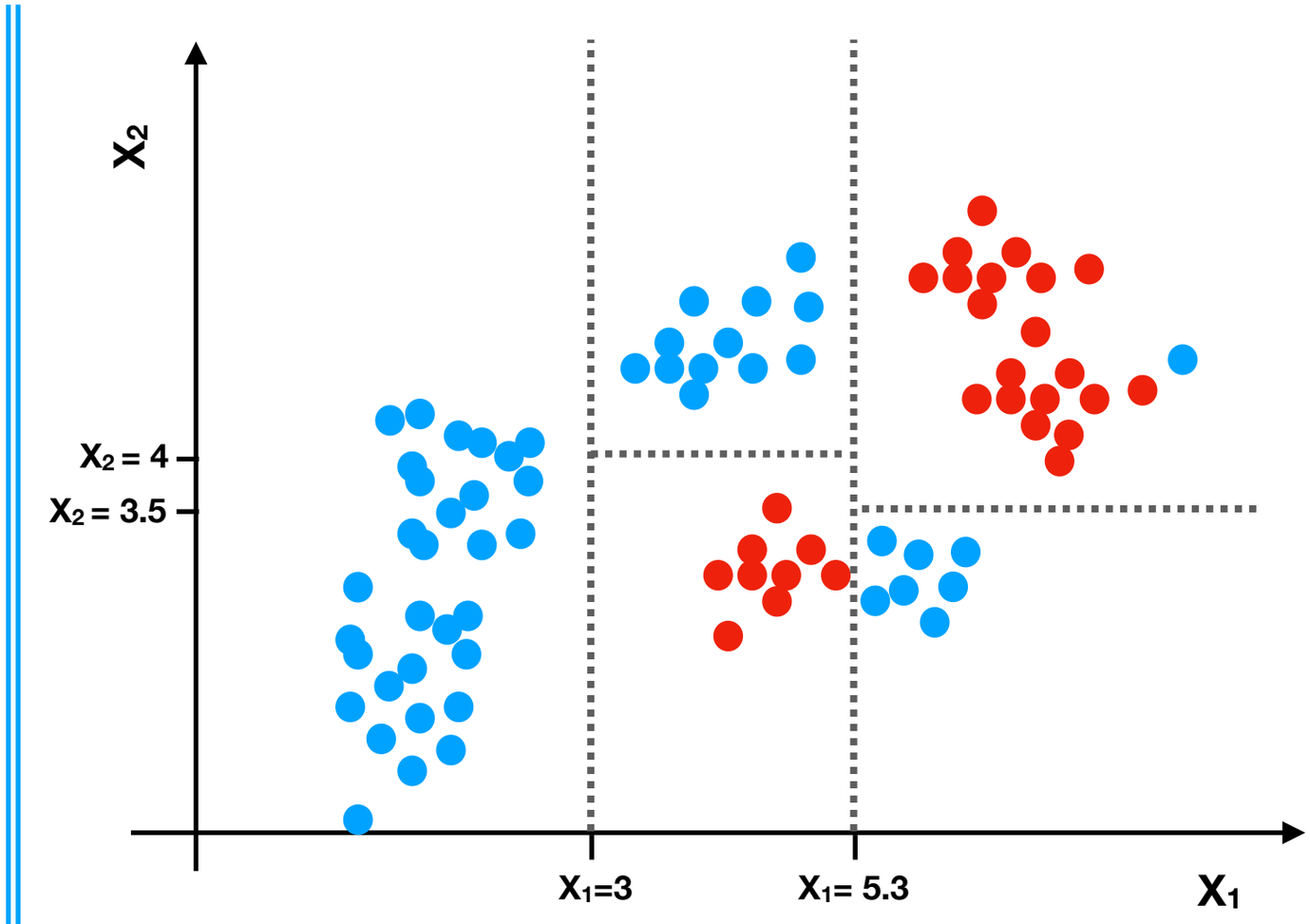




- Depth = 3
- Decision Attribute: X_2

DT for Classification



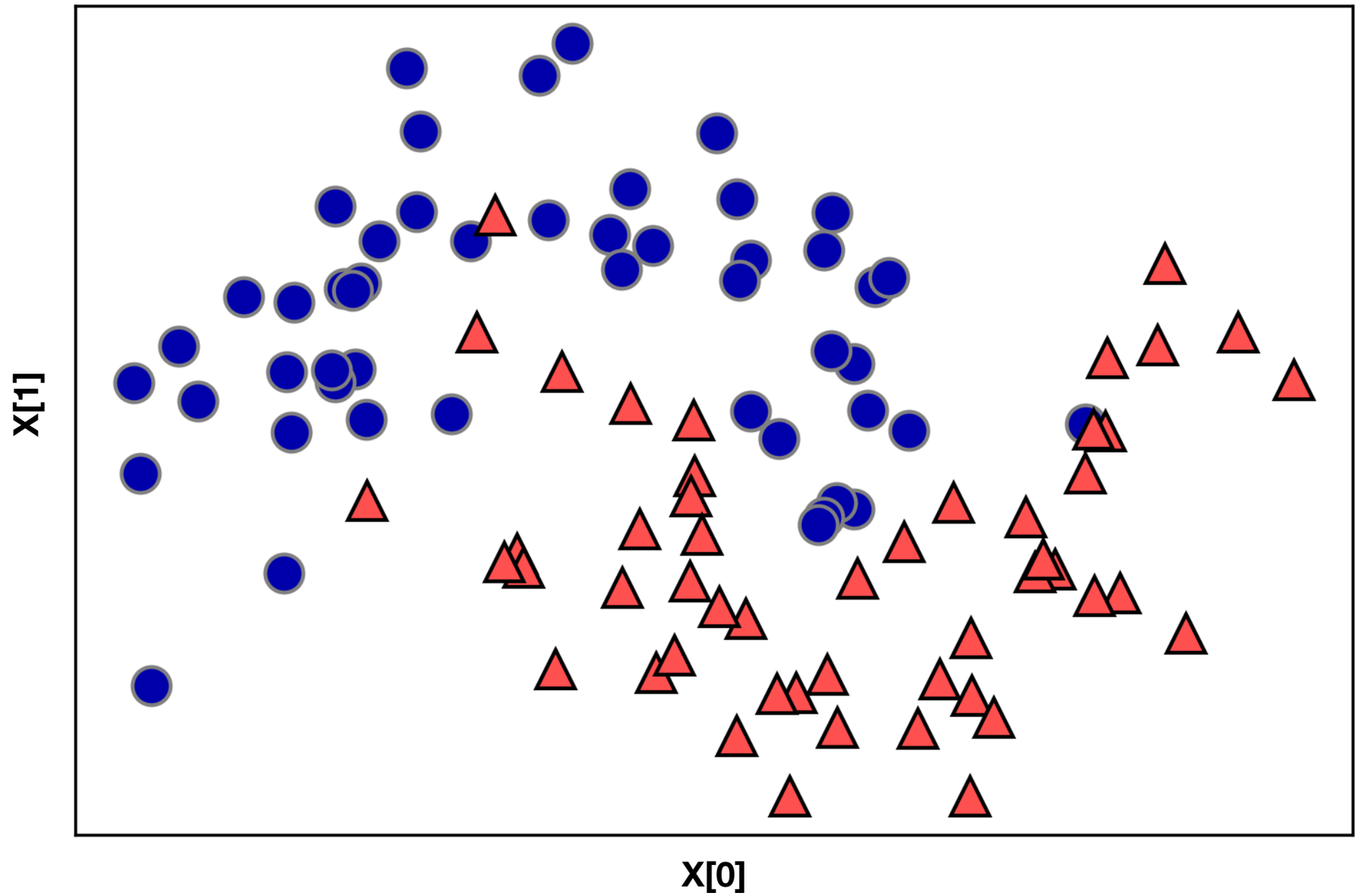


- Depth = 3
- Decision Attribute: X_2

‘two_moons’ toy dataset

- Sklearn dataset, two half moon shapes of a sample (default n =100). Labeled (50%, 50%)
- Task: classify a shape (0: not moon, 1:moon)

```
>> from sklearn.datasets import make_moons
>> moons =make_moons(n_samples=150, random_state=0)
```

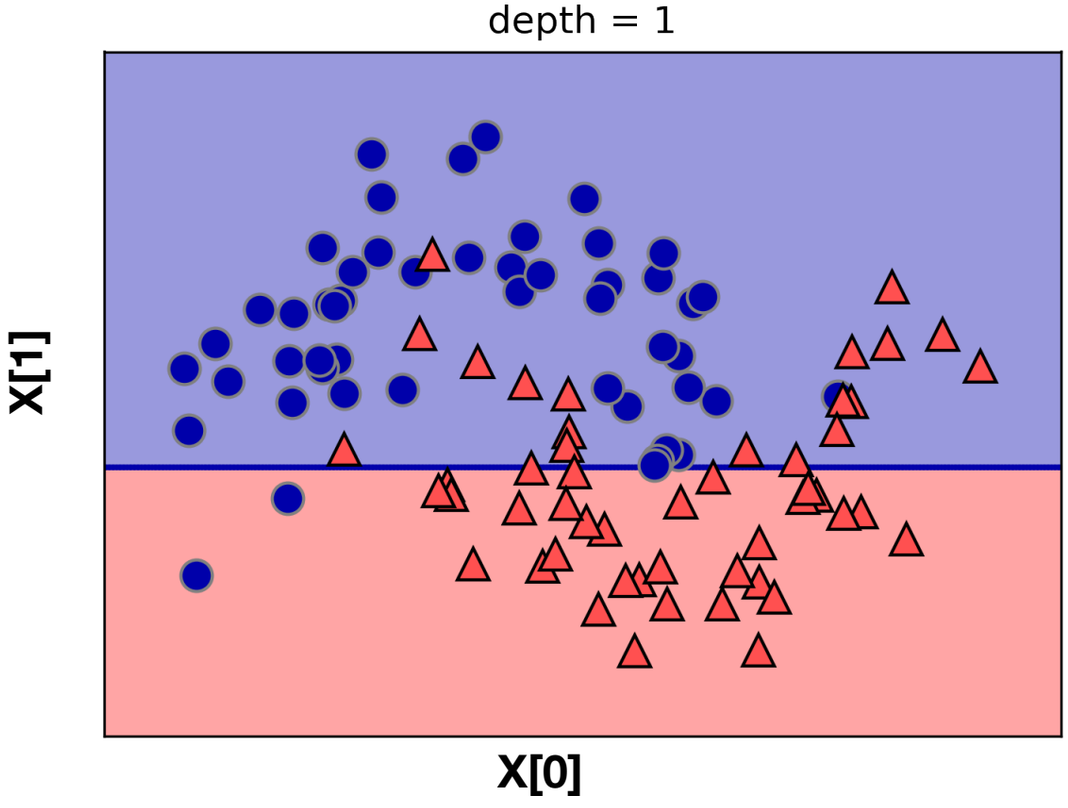


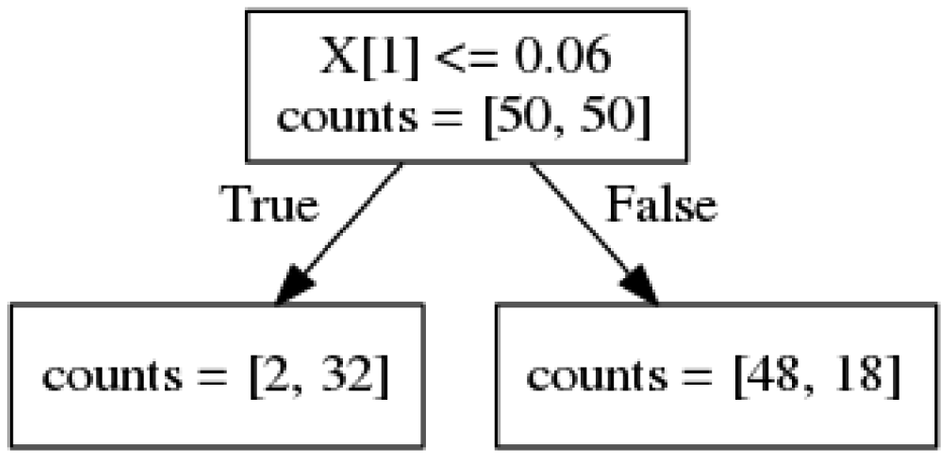
credit: A Muller, mglearn

‘two_moons’ toy dataset

- Sklearn dataset, two half moon shapes of a sample (default n =100). Labeled (50%, 50%)
- Task: classify a shape (0: not moon, 1:moon)

```
>> from sklearn.datasets import make_moons

>> moons =make_moons(n_samples=150, random_state=0)
```



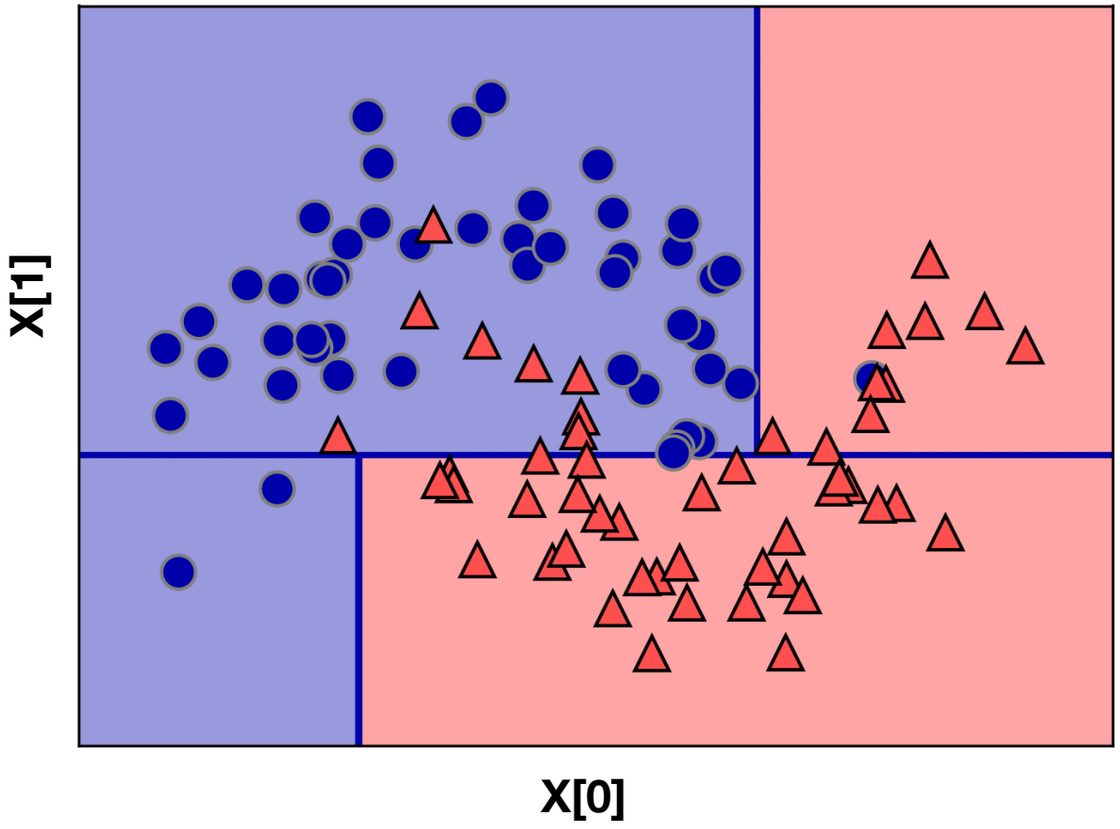


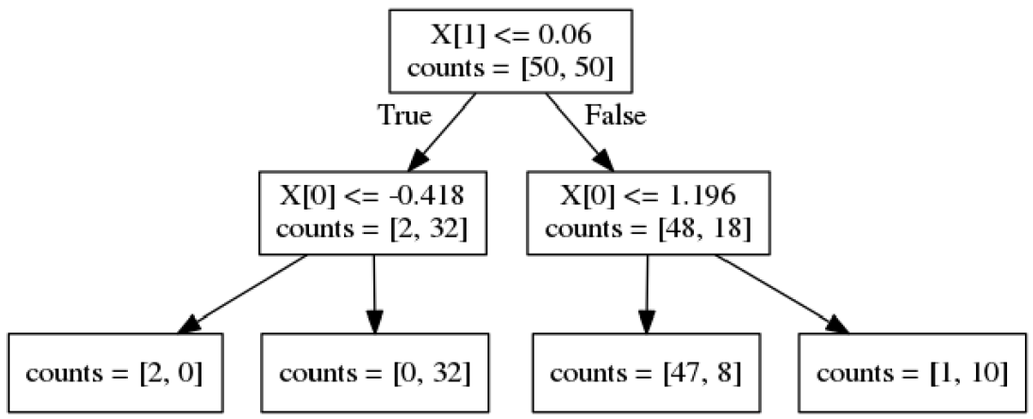
credit: A Muller, mglearn

‘two_moons’ toy dataset

- Sklearn dataset, two half moon shapes of a sample (default n =100). Labeled (50%, 50%)
- Task: classify a shape (0: not moon, 1:moon)

```
>> from sklearn.datasets import make_moons

>> moons =make_moons(n_samples=150, random_state=0)
```



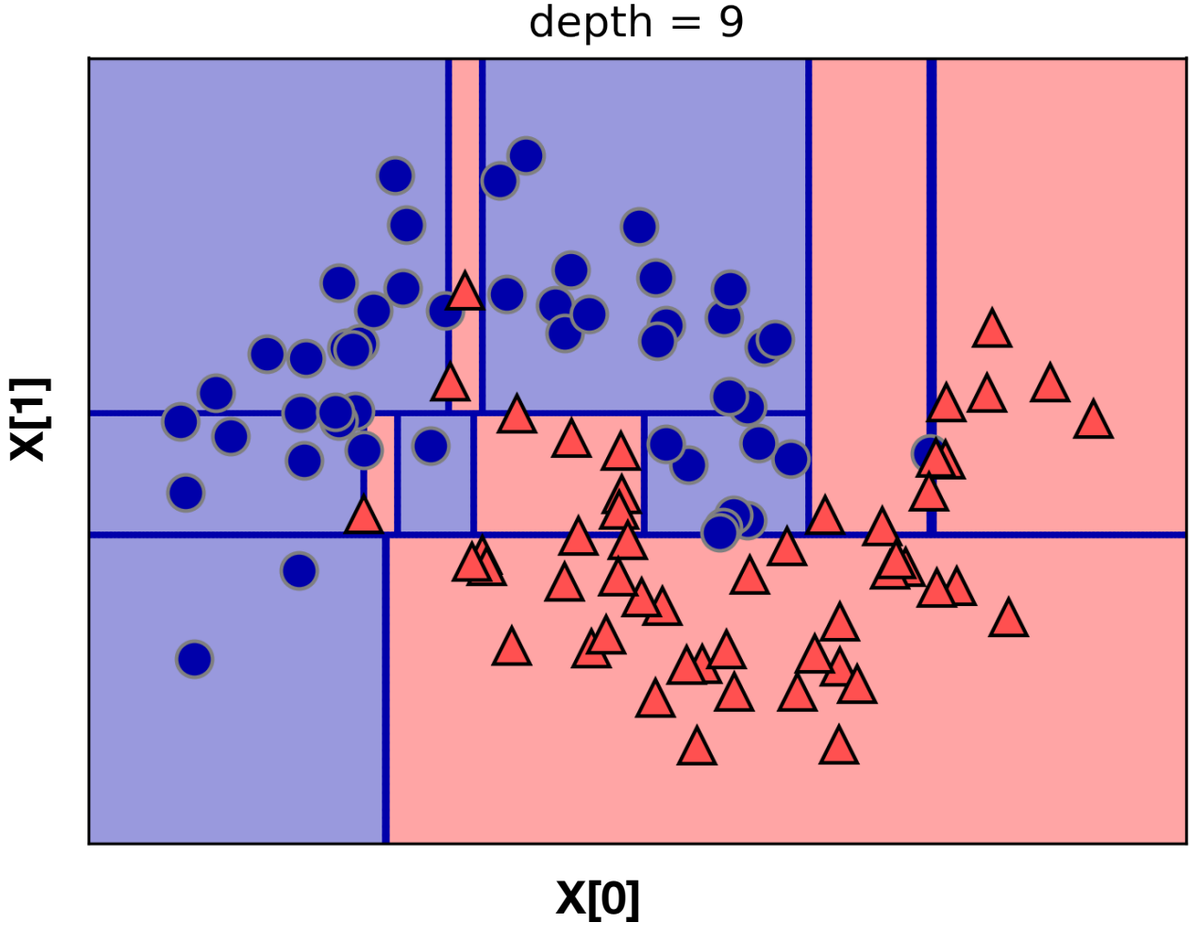


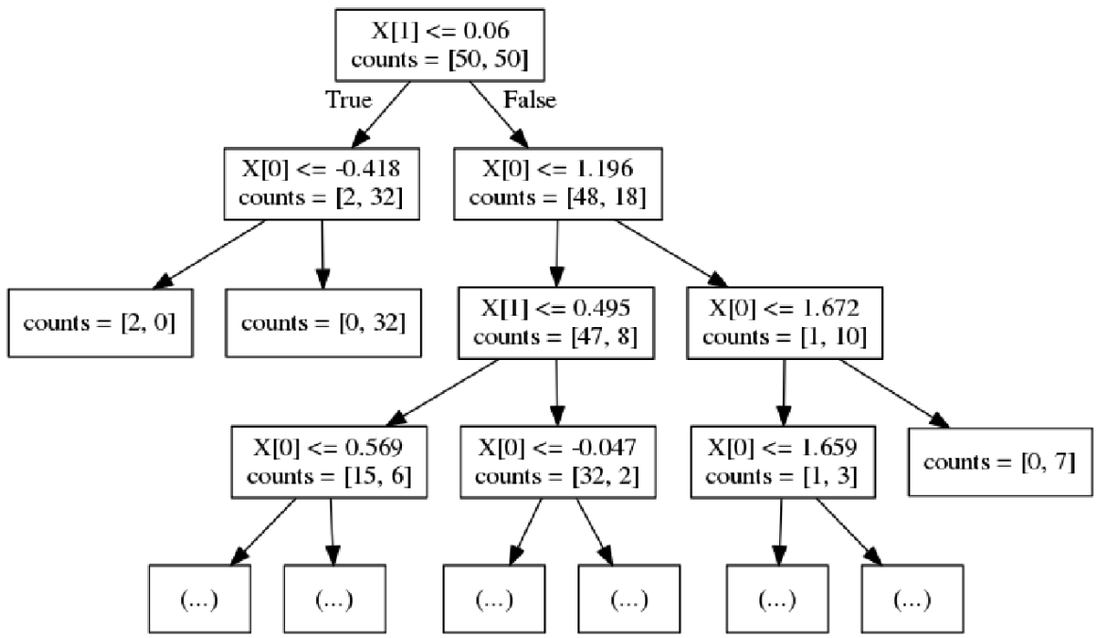
credit: A Muller, mglearn

‘two_moons’ toy dataset

- Sklearn dataset, two half moon shapes of a sample (default n =100). Labeled (50%, 50%)
- Task: classify a shape (0: not moon, 1:moon)

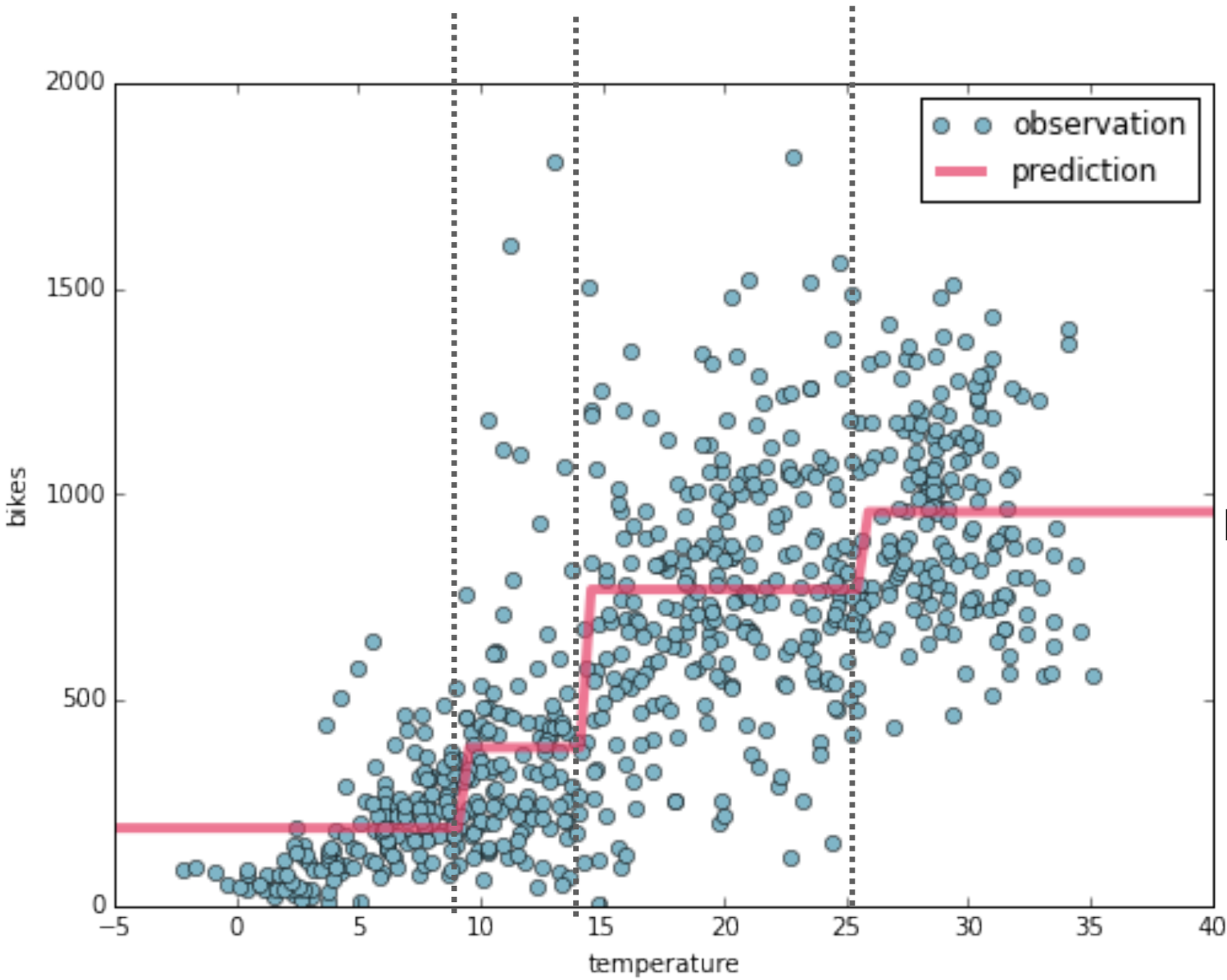
```
>> from sklearn.datasets import make_moons

>> moons =make_moons(n_samples=150, random_state=0)
```





credit: A Muller, mglearn

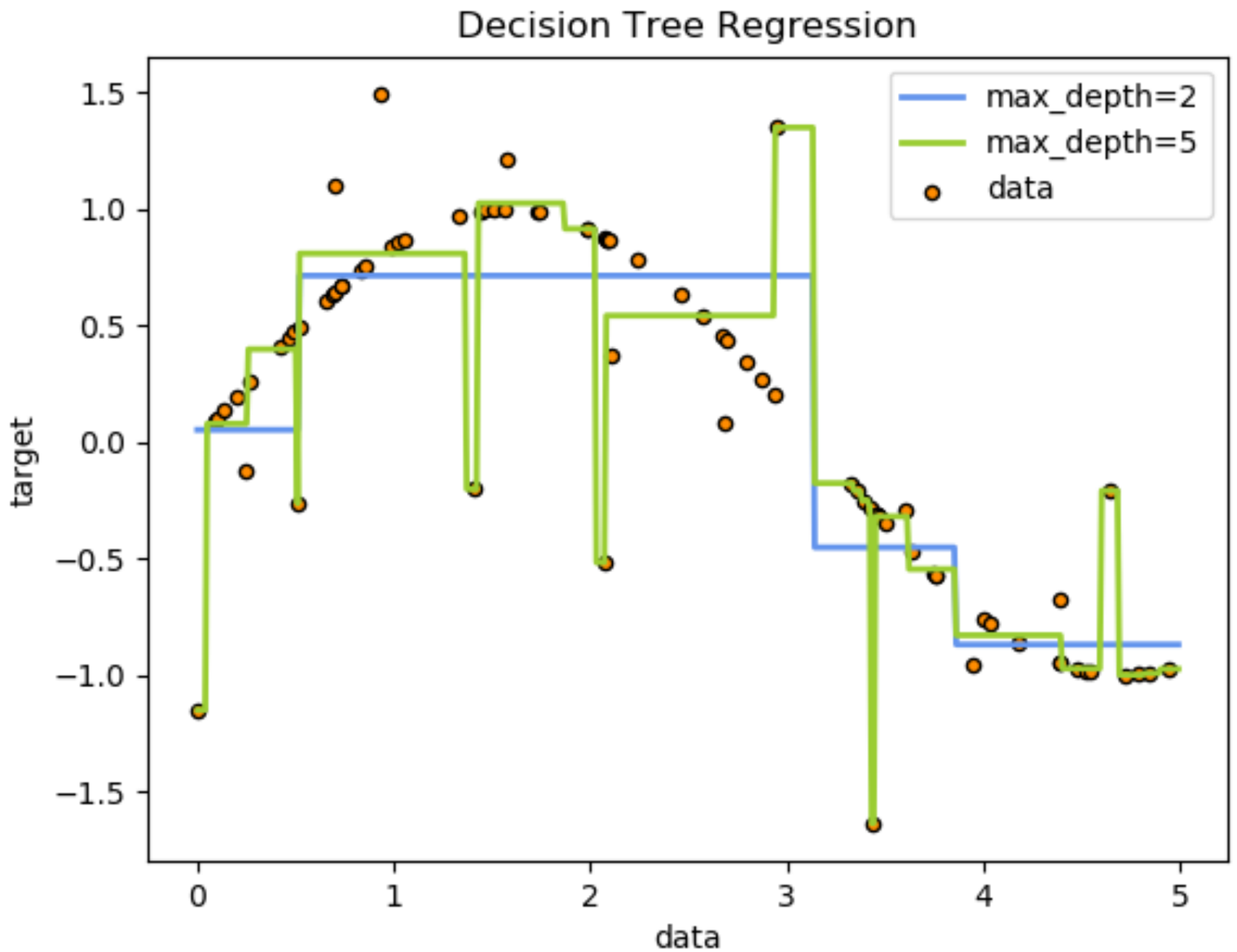
DT for Regression



Minimise MSE (mean square error)

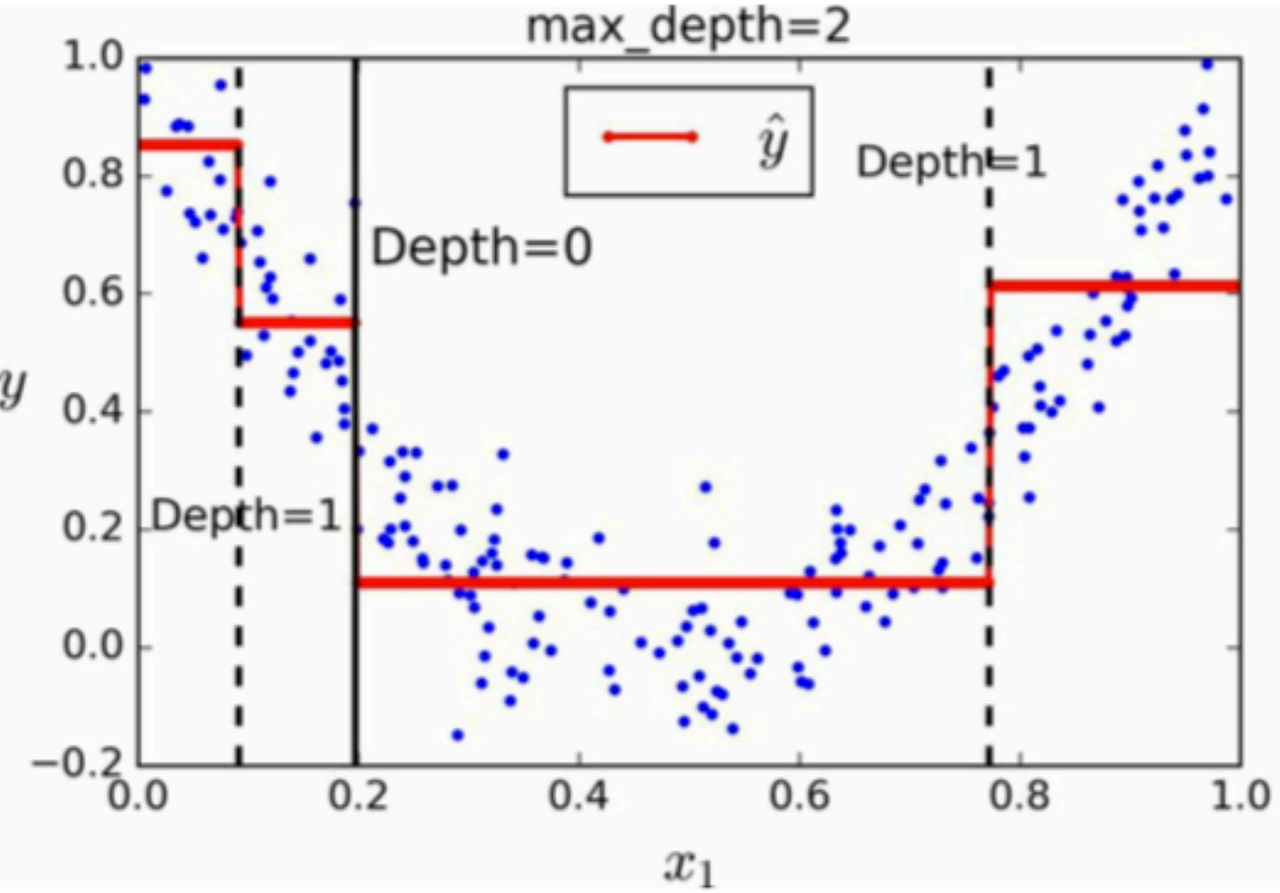
$$\text{MSE} = \frac{SSE = \sum (y_i - \hat{y}_i)^2}{m}$$

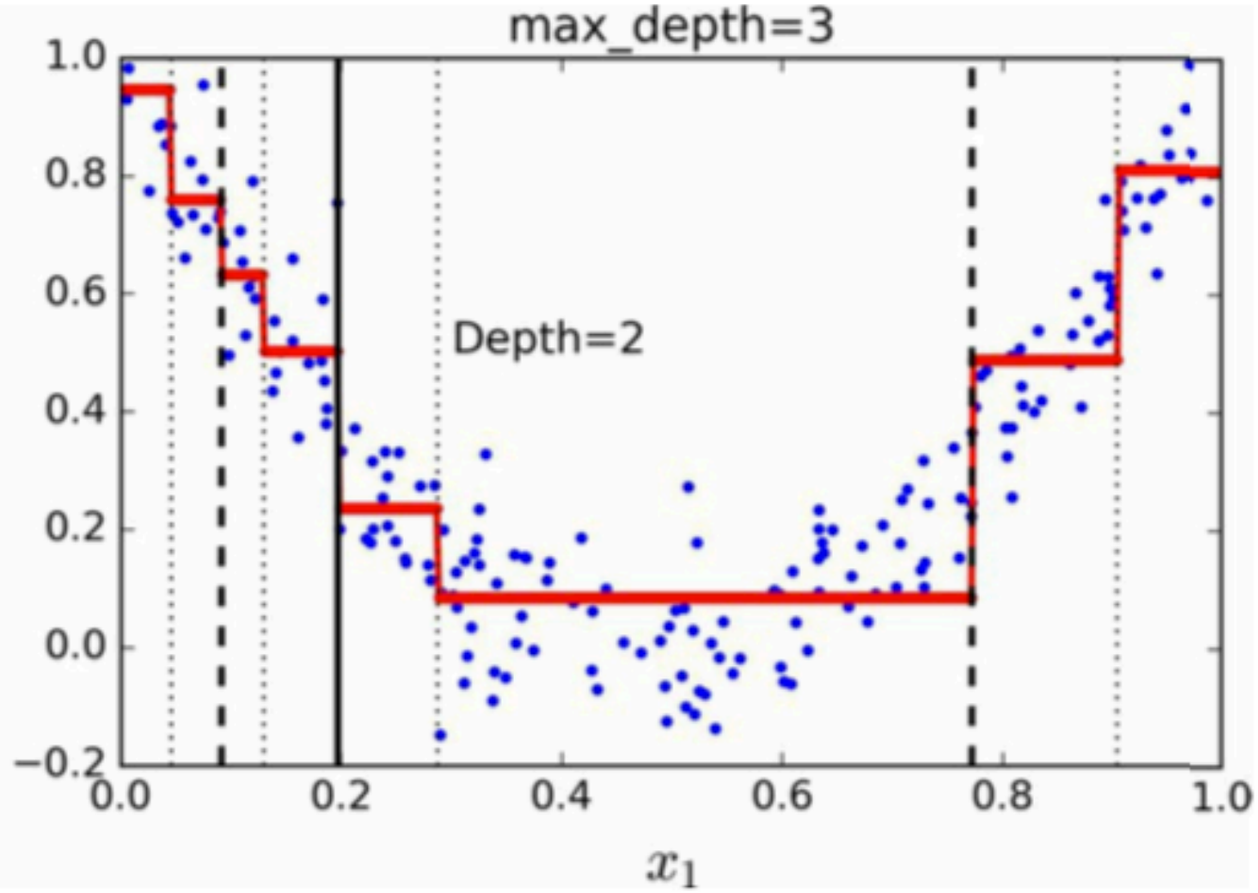
DT for Regression



http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

DT for Linear Regression





From: A Geron, Hands on ML - chapter 6

Top-down Induction of decision trees

The **widely used** technique is a top-down, greedy search approach:

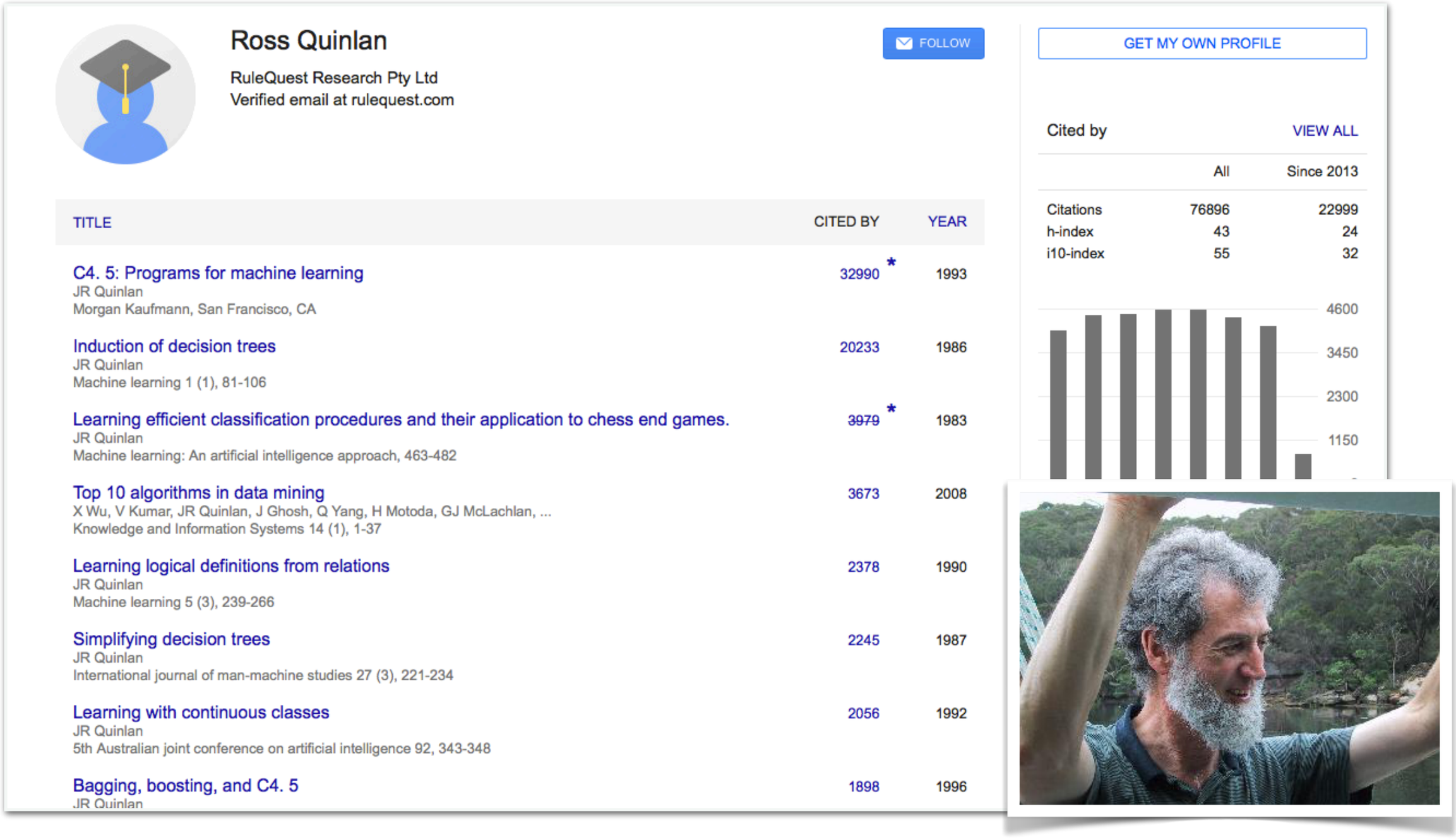
1. Choose the **best** feature x* for the root of the tree.
2. Separate training set **S** into subsets $\{\mathbf{S}_1, \mathbf{S}_2, .., \mathbf{S}_k\}$ where each subset $\mathbf{S}_i$ contains examples having the same value for x*.
3. Recursively apply the algorithm on each new subset until all examples have the same class label.

Choosing (decision) Features

- **ID3** (Iterative Dichotomiser 3) and **C4.5** by Ross Quinlan (1986) based on Information Gain.

- **CART** by Leo Breiman: **Gini Impurity** — measure of the heterogeneoity (or "impurity") of the nodes. If all data points at one node belong to the same class then this node is considered "pure". So by minimising the Gini Impurity the decision tree finds the features they separate the data best.

Features for Decision Nodes



J.R. Quinlan, *Induction of Decision Trees, Machine Learning* 1: 81—106, 1986.

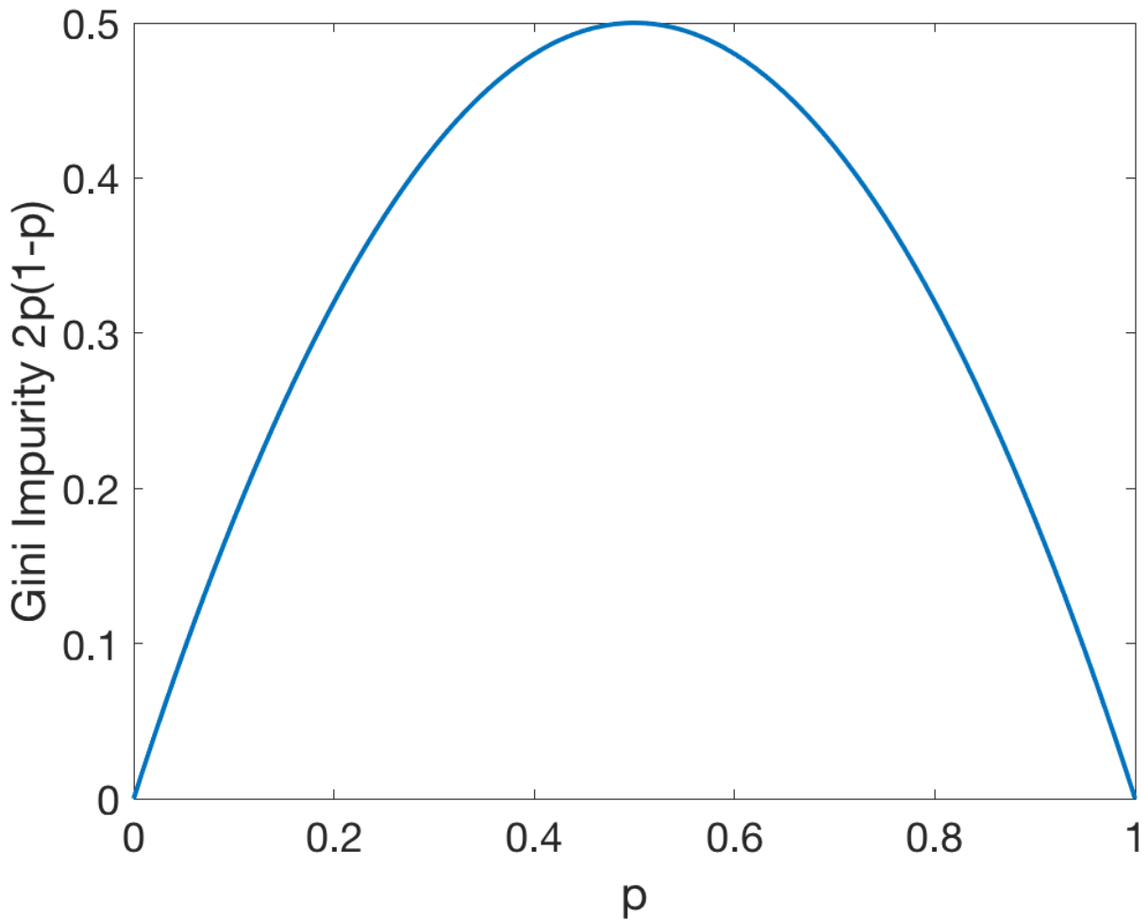
Available: http://hunch.net/~coms-4771/quinlan.pdf

1 Gini Impurity → CART

Let k be the given classes in a dataset,

$$I_G(p) = \sum_{i=1}^{k} p_i(1 - p_i)$$

$$= 1 - \sum_{i=1}^{k} p_i^2$$



	Count		Probability		Gini Impurity
	n_1	n_2	p_1	p_2	$1 - p_1^2 - p_2^2$
Feature A	0	10	0	1	$1 - 0^2 - 1^2 = 0$
Feature B	3	7	0.3	0.7	$1 - 0.3^2 - 0.7^2 = 0.42$
Feature C	5	5	0.5	0.5	$1 - 0.5^2 - 0.5^2 = 0.5$

2 Entropy → ID3

- Let *X* be a random variable with the following probability distribution

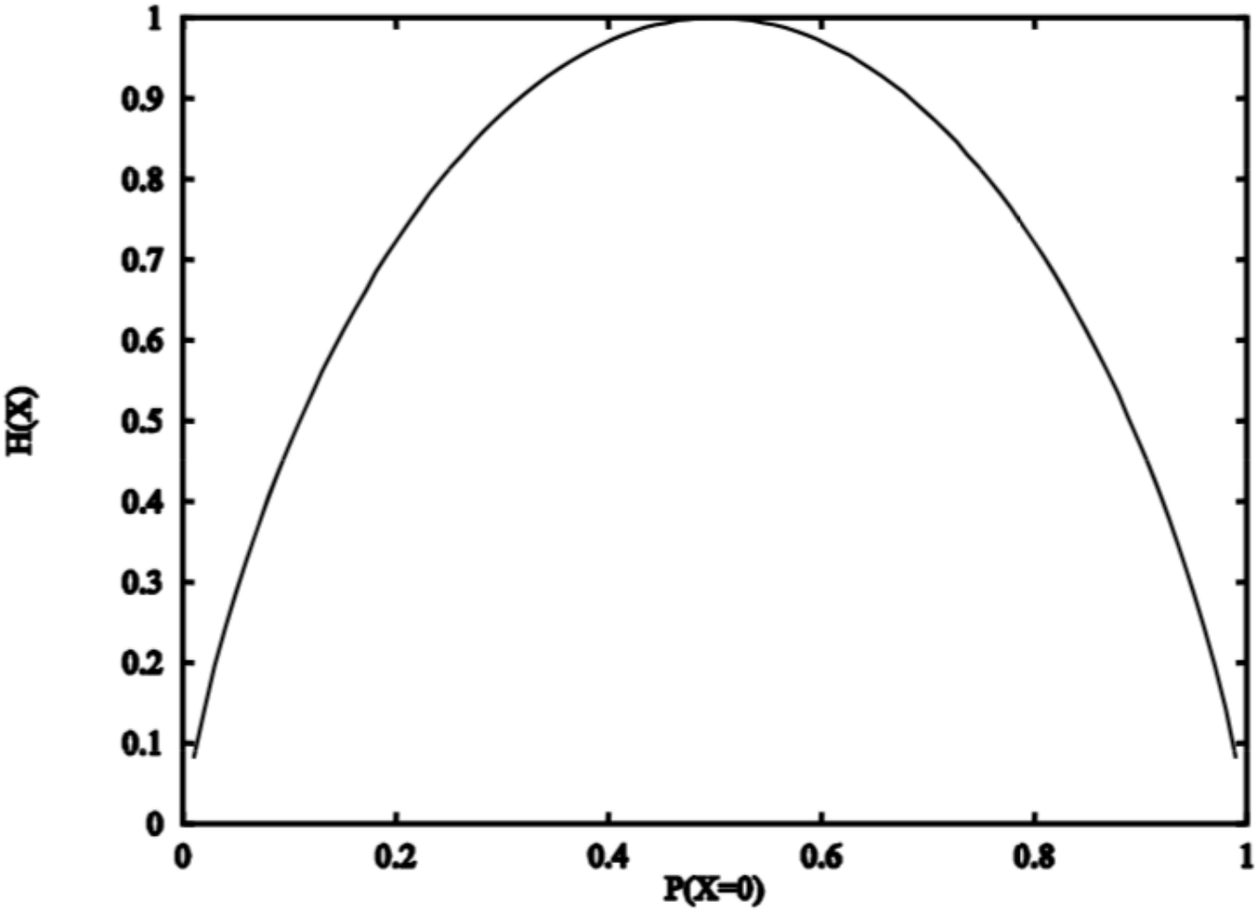
P(X = 0)	*P(X = 1)*
0.2	0.8

- The <u>entropy</u> of ***X***, denoted *H(**X**)*, is defined as

$$H(X) = -\sum_{x} P_X(x) \log_2 P_X(x)$$

- **Entropy** measures the uncertainty of a random variable
- The larger the entropy, the more uncertain we are about the value of *X*
- If P(X=0)=0 (or 1), there is no uncertainty about the value of X, entropy = 0
- If P(X=0)=P(X=1)=0.5, the uncertainty is maximized, entropy = 1

Entropy of random variable X



Entropy is applied as a term of the heterogeneity of a split/sample; If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has entropy of one.

If P(X=0)=0 (or 1), there is no uncertainty about the value of X, entropy = 0

If P(X=0)=P(X=1)=0.5, the uncertainty is maximized, entropy = 1

Mutual Information

The mutual information measure of two random variables *X* and *Y* is defined as:

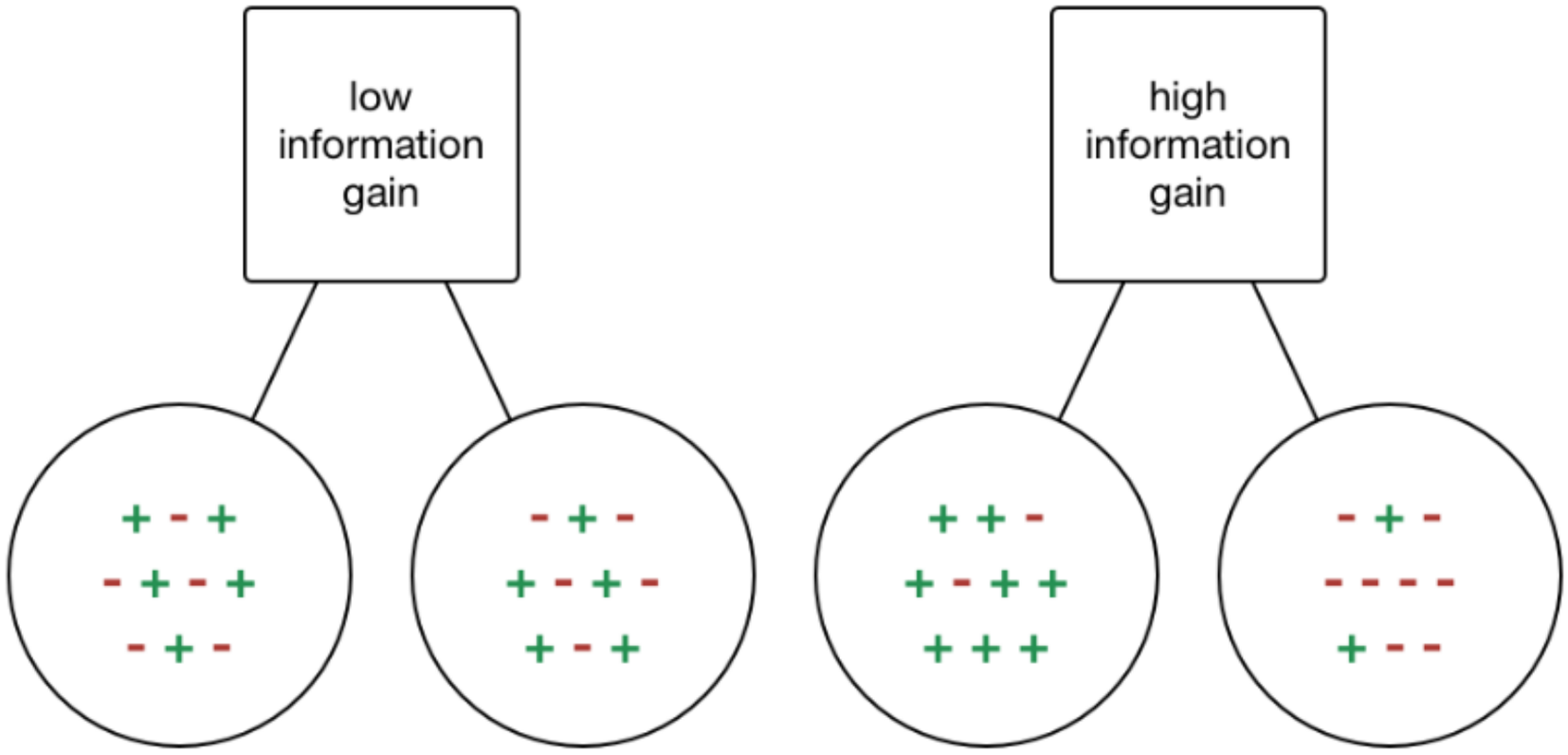
$$I(X,Y) = H(Y) - H(Y|X)$$

the amount of information we learn about Y by knowing the value of X (and vice versa – it is symmetric).

Information Gain (IG) = $\underset{j}{\operatorname{argmax}}\ H(Y) - H(Y|X_j)$

$\Rightarrow \underset{j}{\operatorname{argmin}}\ H(Y|X_j)$

Information Gain



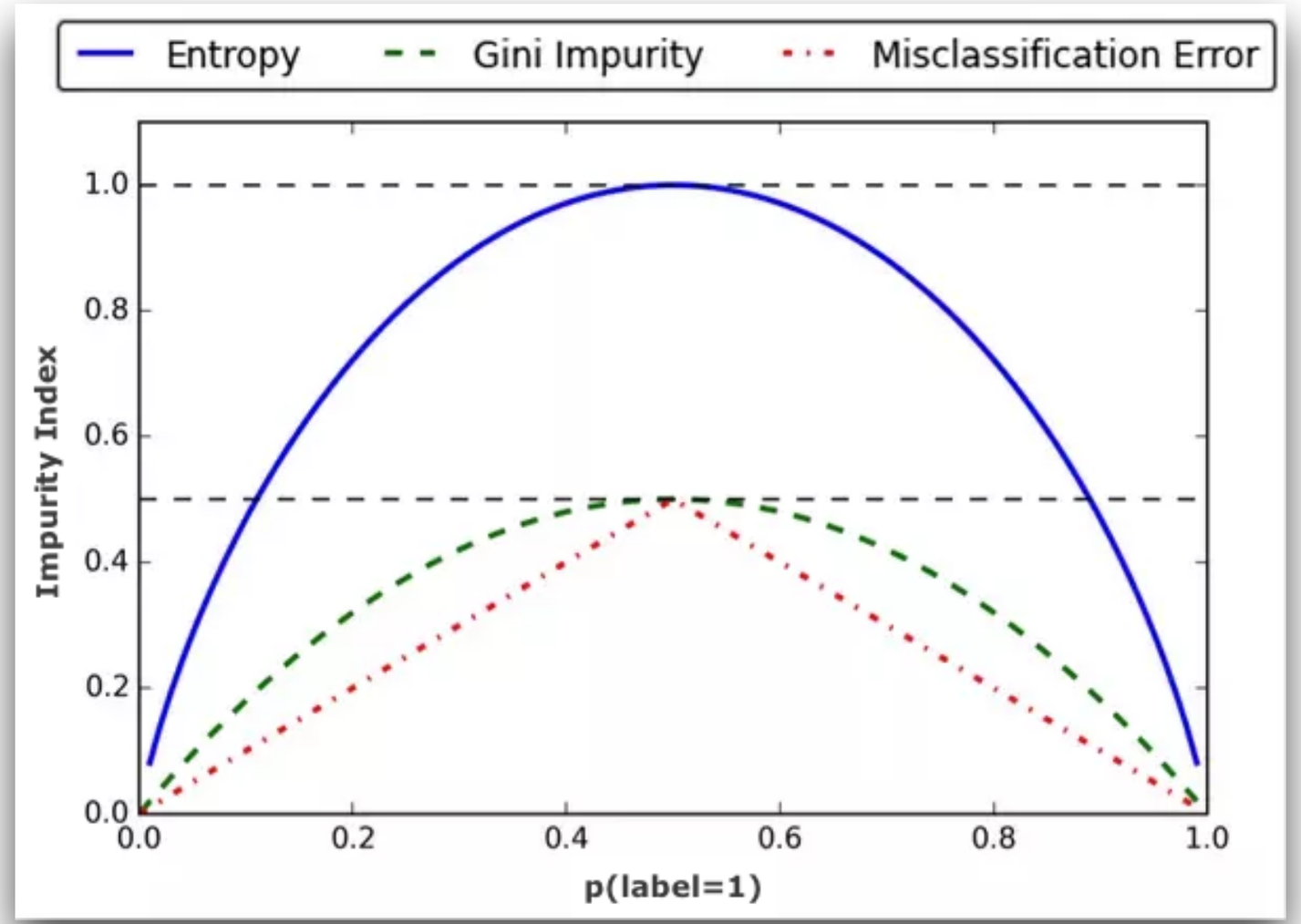
Information Gain (IG) = $\underset{j}{\operatorname{argmax}}\ H(Y) - H(Y|X_j)$

$\Rightarrow$ $\underset{j}{\operatorname{argmin}}\ H(Y|X_j)$

Gini and Entropy

◉ Gini : $I_G(S) = 1 - \sum_{j=1}^{c} p_j^2$

◉ Entropy : $H(S) = -\sum_{j=1}^{c} p_j \log p_j$



Agenda

- ➡ Decision Trees Induction
  - ◉ Context
  - ◉ Estimation/“design”
  - ◉ Properties
- ➡ Ensemble Learning
  - ◉ Bagging (mainly - I’m hoping to cover more next week)
- ➡ Summary
- ➡ Tutorial

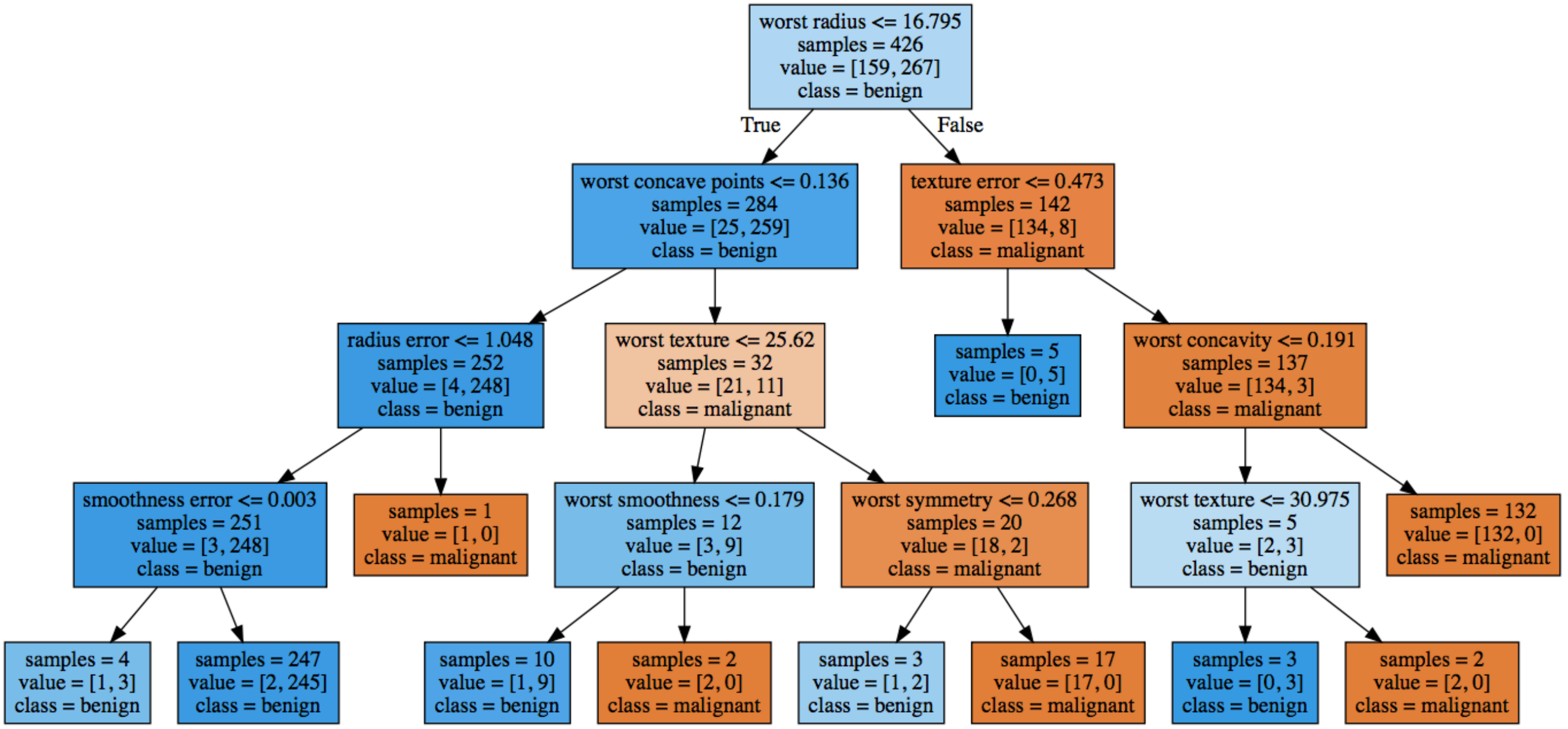
Decision Trees' Summary

➡ very popular:

- Easy to understand
- Easy to implement
- Easy to use
- Computationally cheap (well!)
- Suitable for linearly (& non linearly) separable classes
- Does regression as well.
- Completely invariant to scaling of the data

Feature Importance

```
>> load_breast_cancer()
```



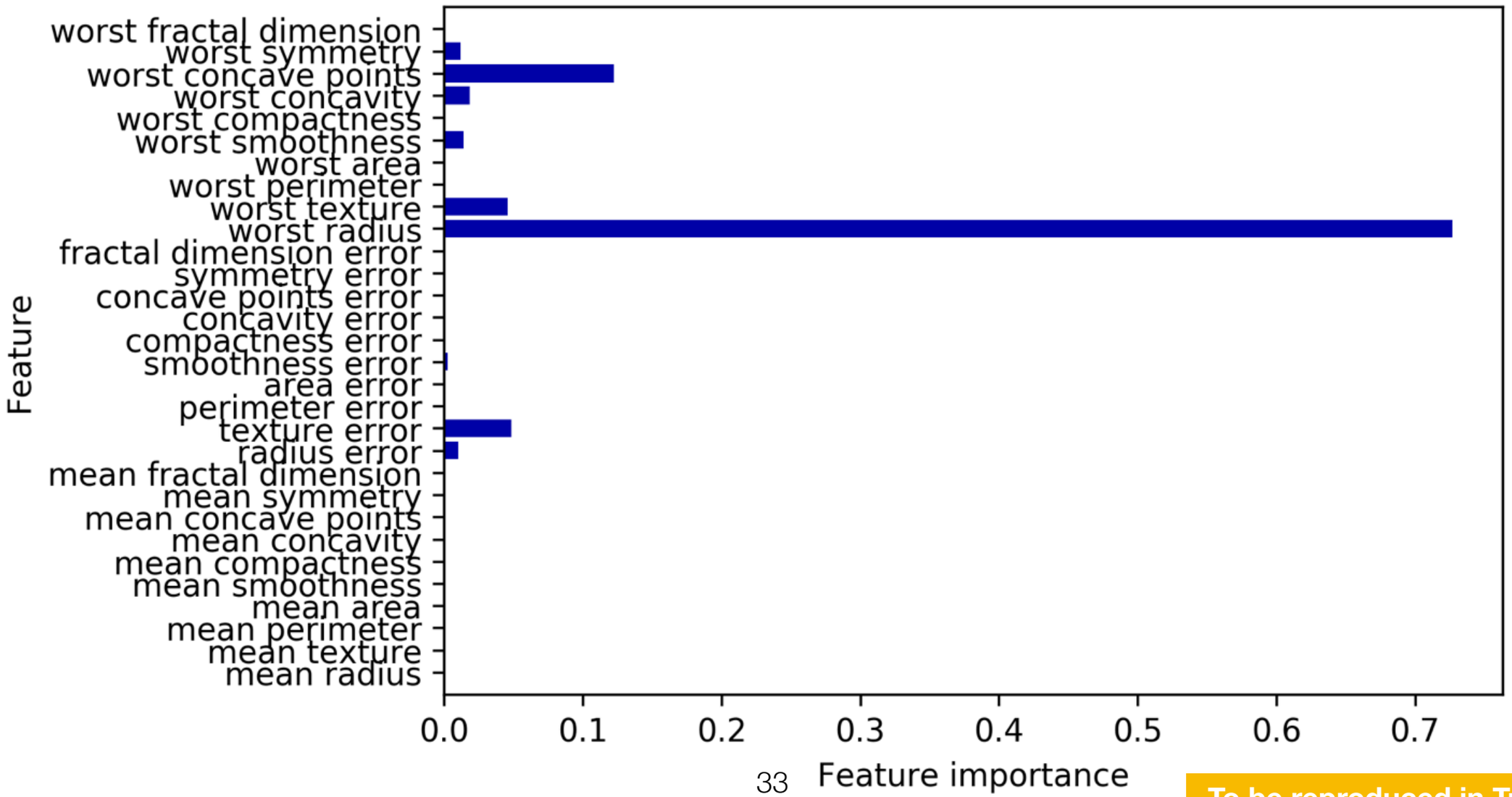
To be reproduced in Tutorial tasks

Feature Importance

```
>> load_breast_cancer()
```

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

```
Feature importances:
[ 0.      0.      0.      0.      0.      0.      0.      0.      0.      0.      0.01
  0.048   0.      0.      0.002   0.      0.      0.      0.      0.      0.727
  0.046   0.      0.      0.014   0.      0.018   0.122   0.012   0.      ]
```



To be reproduced in Tutorial tasks

But it is horrible!

Today’s Puzzle

DT complexity

Decision Tree Depth

Consider a/an (insane) DT where each split (decision node) would peel off one training example, the absolute maximum depth would be ***?***, where *N* is the number of training samples.

- Training Error = ?
- Generalisation (test) error = ?

Decision Trees Main Issue

OVERFITTING

Pruning - Pre and Post

- A common technique to avoid overfitting in Decision Trees is to prune the tree; either whilst it's being constructed (pre-pruning) or after it's been constructed (post-pruning).

- Recursively remove leaves and evaluate Accuracy loss.

✗ NOT supported in scikit-learn

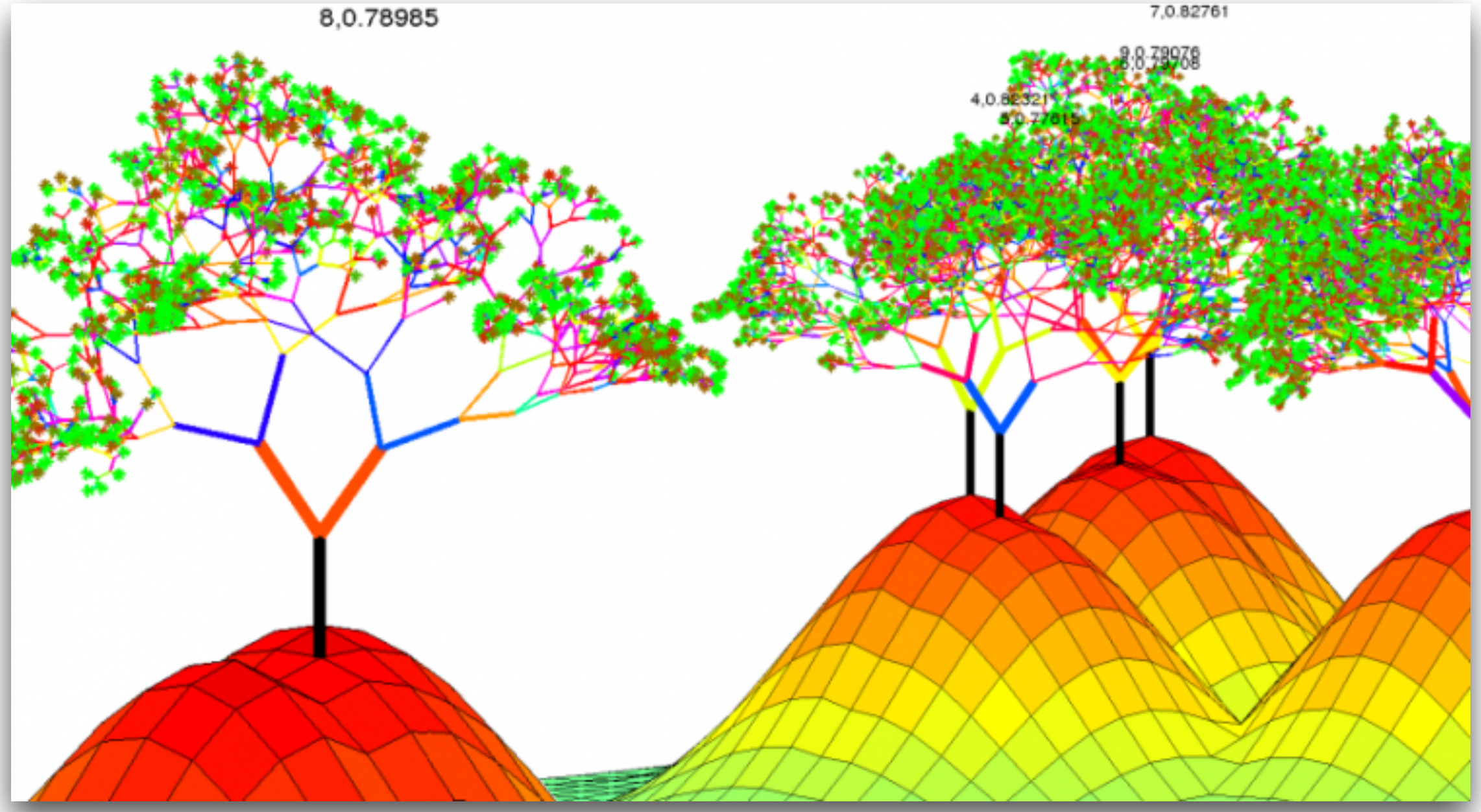
- Limiting maximum depth of the tree
- Limiting maximum number of leaves
- Requiring a minimum number of samples in a node to allow split

✓ supported in scikit-learn

Agenda

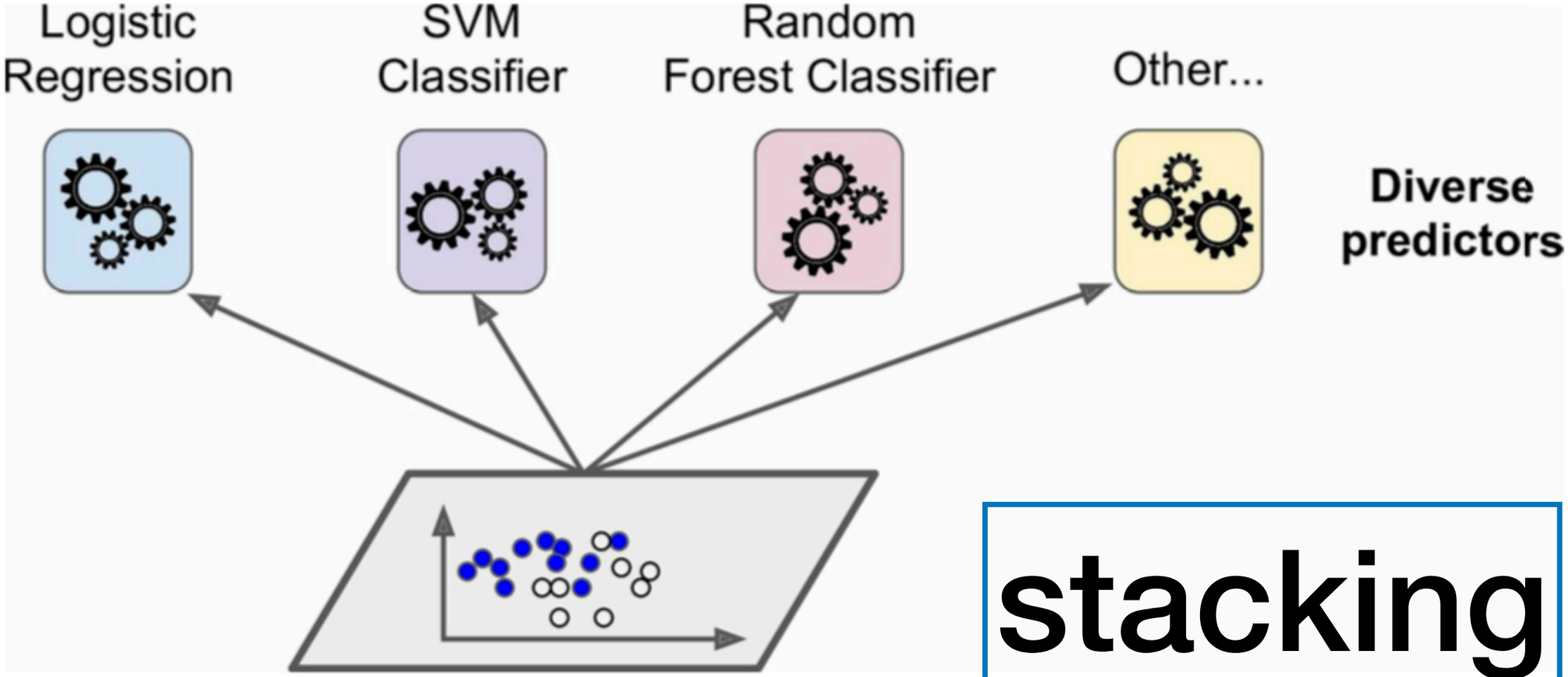
- Decision Trees Induction
  - Context
  - Estimation/“design”
  - Properties
- Ensemble Learning
  - Bagging (mainly - I’m hoping to cover more next week)
- Summary
- Tutorial

Ensemble of DT



Bagging, boosting and stacking!

Why Ensembles



Majority voting → **Hard voting**

Majority weighted voting → **Soft voting**

From: A Geron, Hands on ML - chapter 7

Why Ensembles

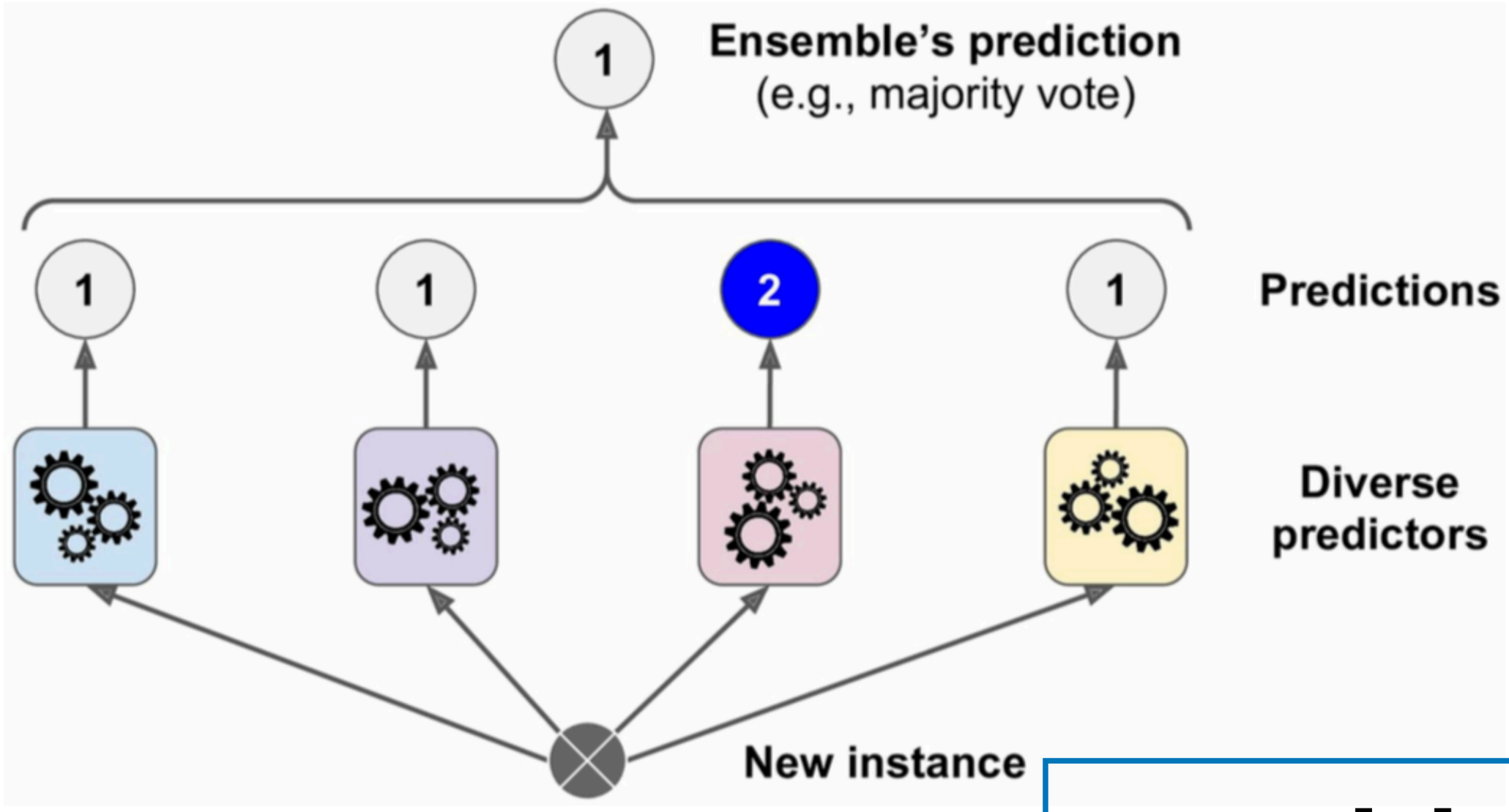


Figure 7-2. Hard voting classifier predictions

stacking

Majority voting → **Hard voting**

Majority weighted voting → **Soft voting**

 From: A Geron, Hands on ML - chapter 7

Ensemble DT

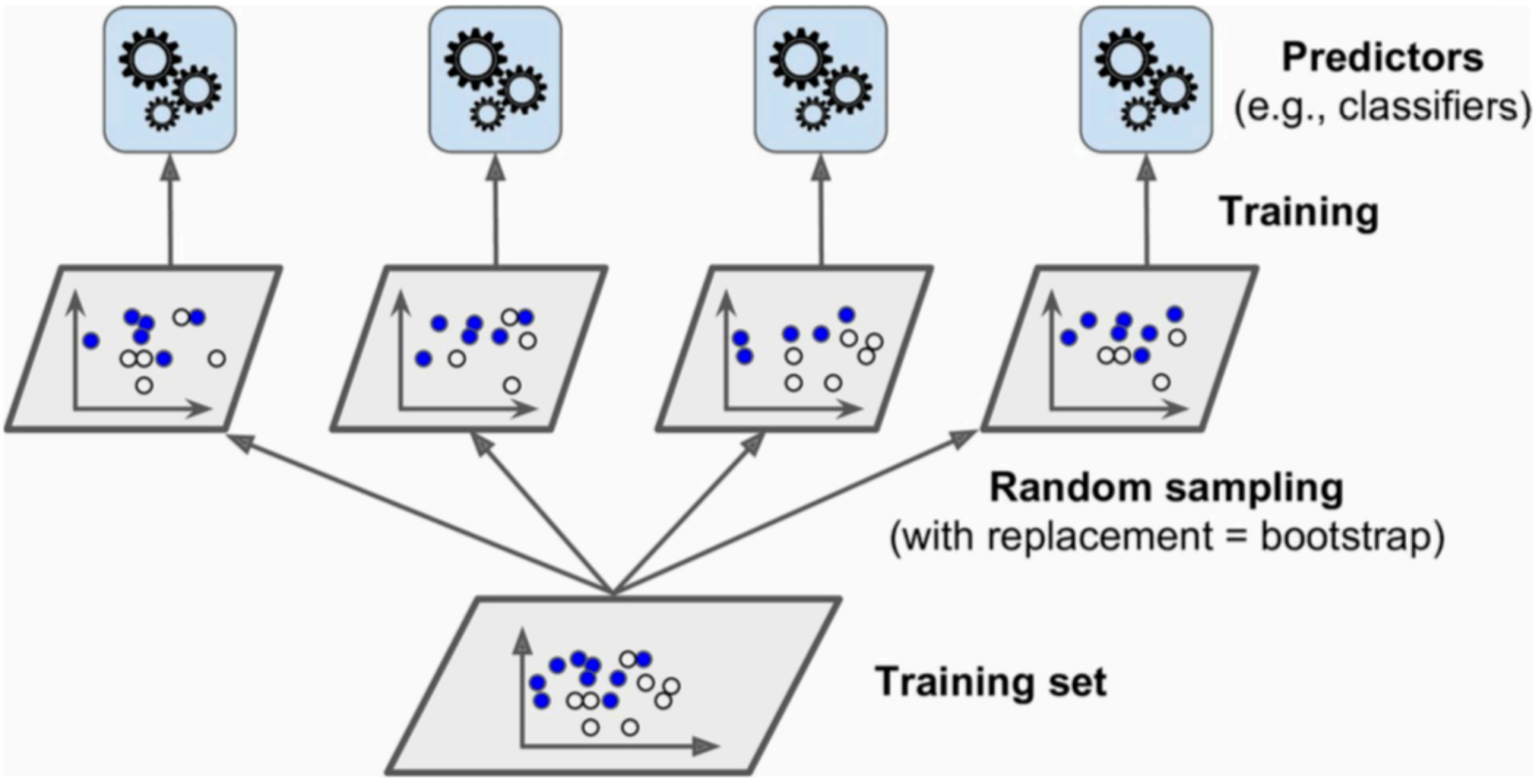


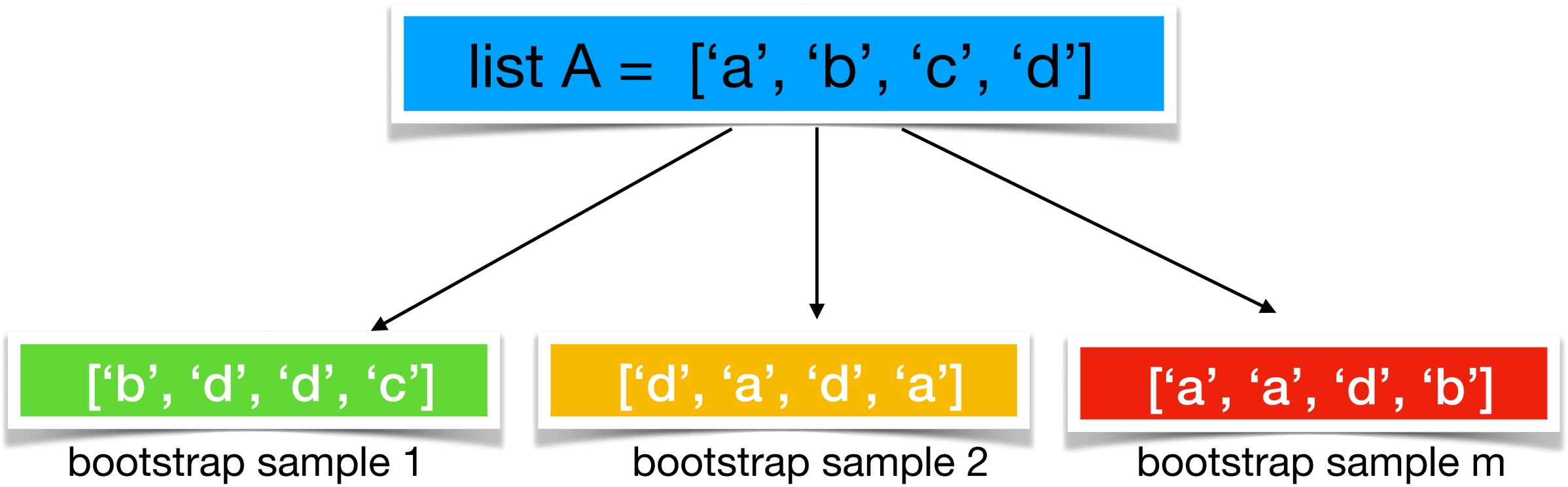
Figure 7-4. Pasting/bagging training set sampling and training

Bagging: random sampling of training set with replacement

Pasting: random sampling of training set without replacement

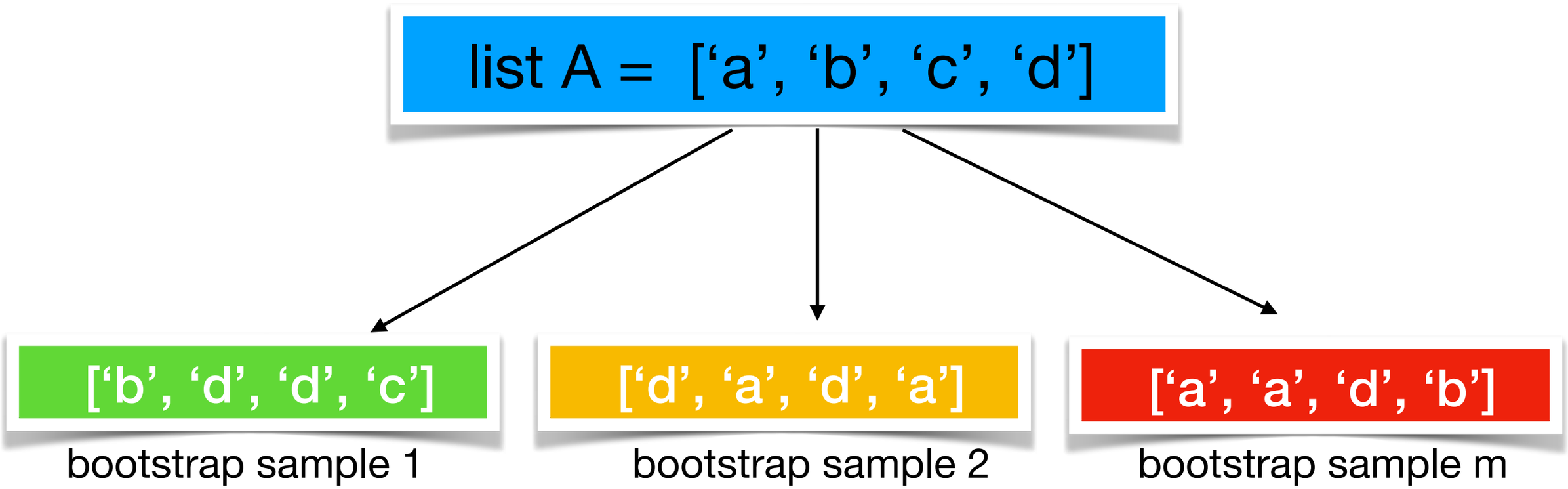
From: A Geron, Hands on ML - chapter 7

Bootstrap (bagging)



Added randomisation

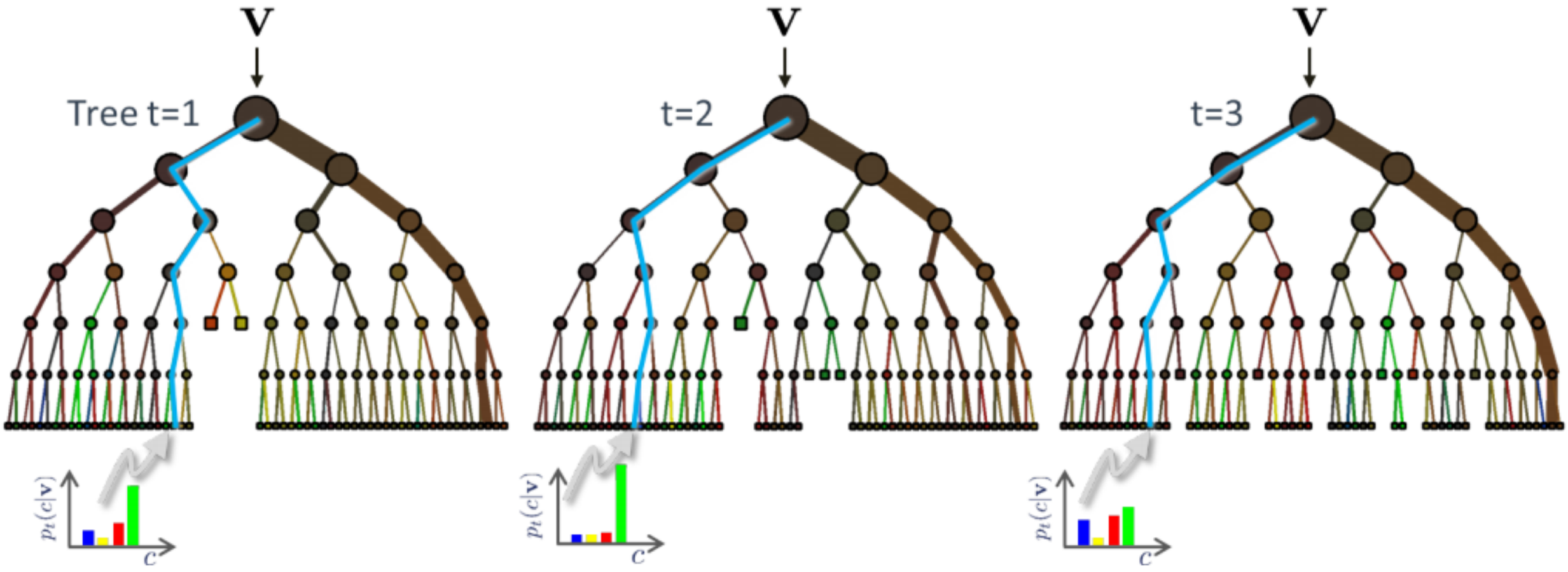
Random Forests



Randomisation:

- **training data**
- **Subset of features**

Random Forests

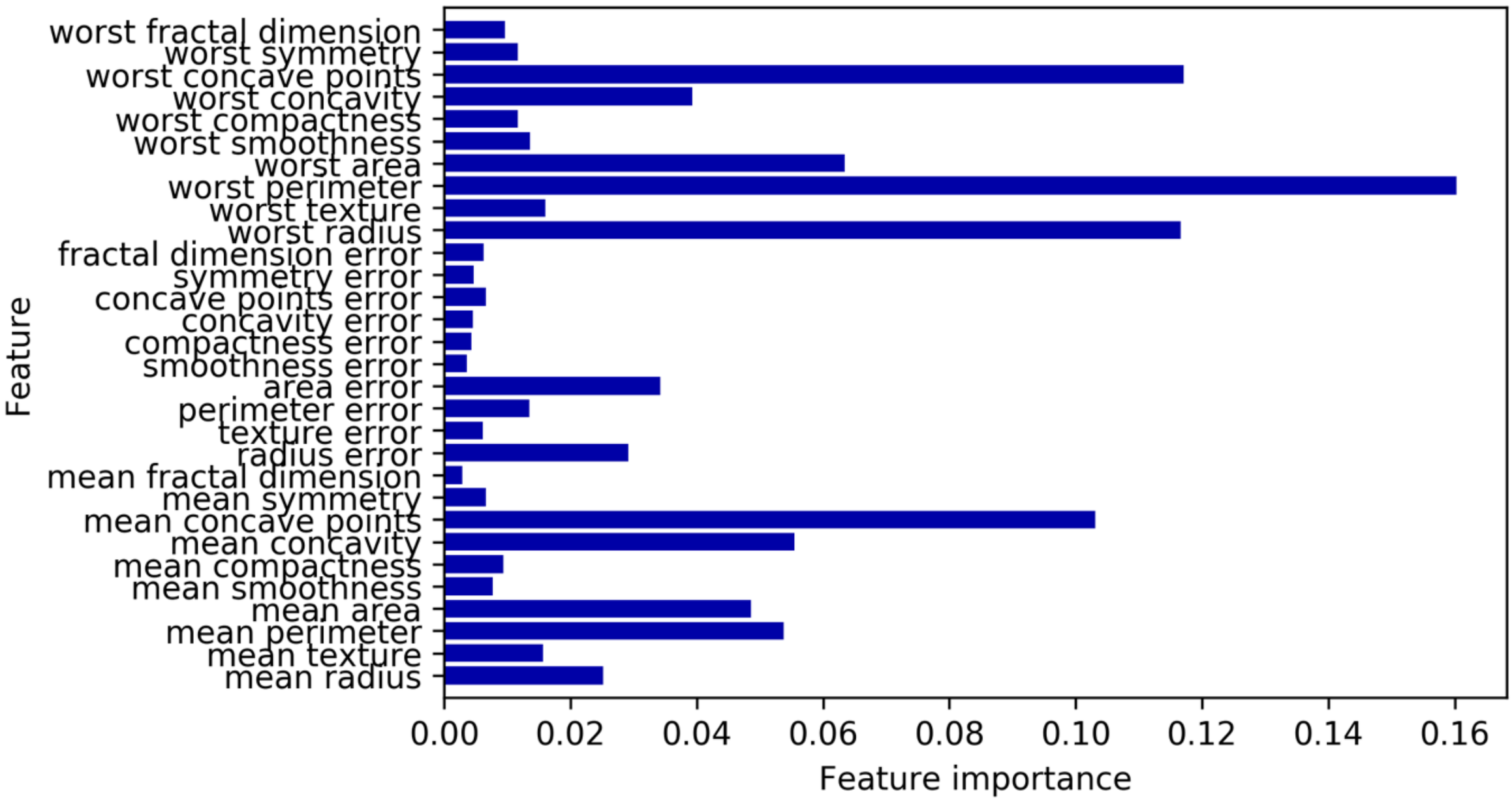


Classification: Majority vote

Regression: Average

Feature Importance (bagging)

```
>> load_breast_cancer()
```



Reading

- A Geron, Hands on ML, Ch. 6 and 7 (pp.167–190)

- Printed chapter on Decision Trees on BB.
- T Hastie, R Tibshirani, J Friedman, "The Elements of Statistical Learning" Sec. 8.7 & Chp. 15 - https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf
- L Breiman "Random Forests", Machine Learning, 45(1), 5-32, 2001 Machine Learning.