Introduction to Random Forests

Content

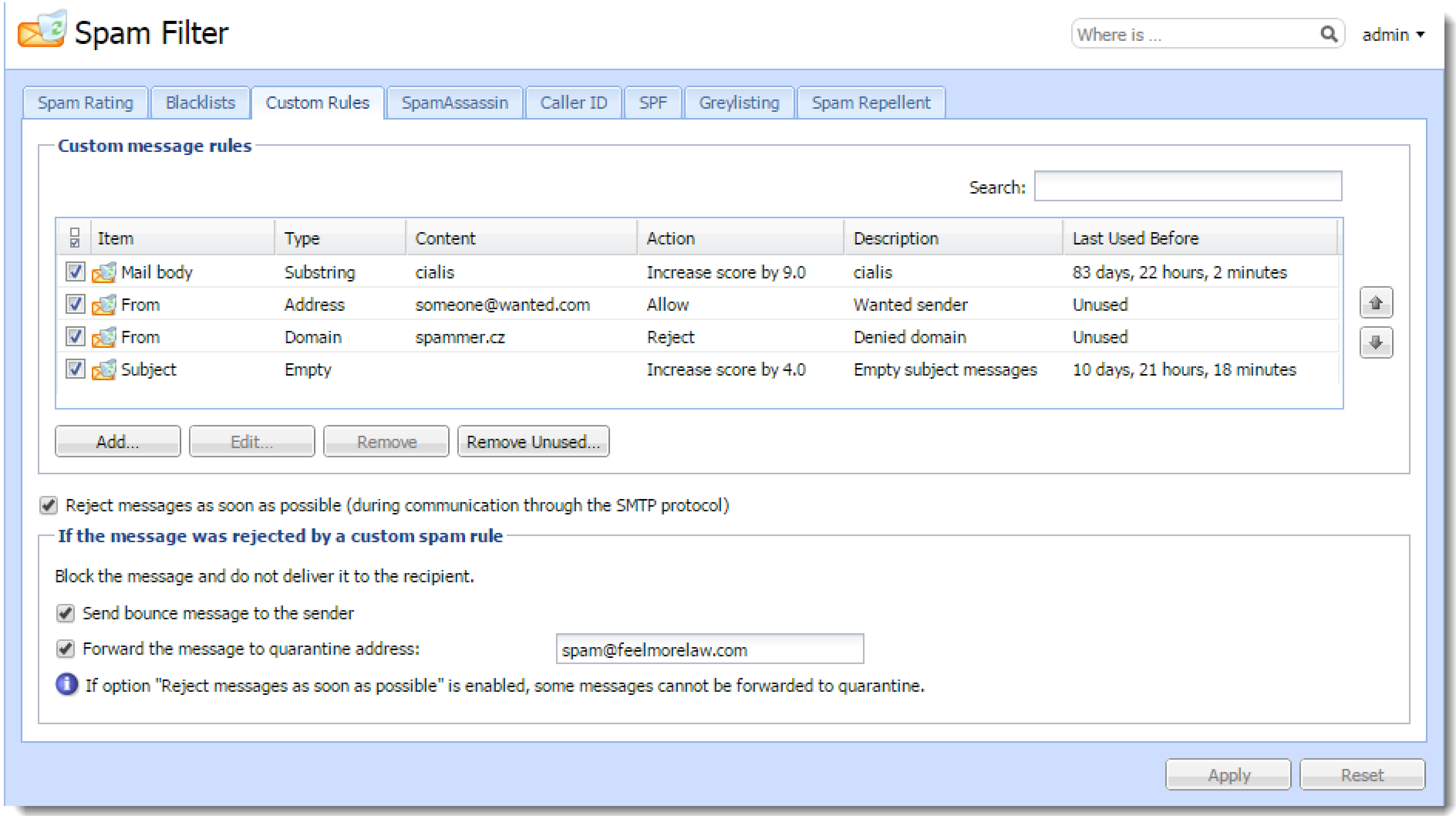
1. **Machine learning**
    - classification
    - regression
    - overfitting, regularization, hyperparameters
    - datasets
2. **Decision trees**
3. **Random Forests**

Machine learning

- make the computer learn how to perform some task, instead of instructing it on how to do the task
- learn from *data* and *automatically*
- typical tasks are
  - **classification**
  - **regression**
  - clustering
  - dimensionality reduction

Example classification

The old way: rule-based



With machine learning

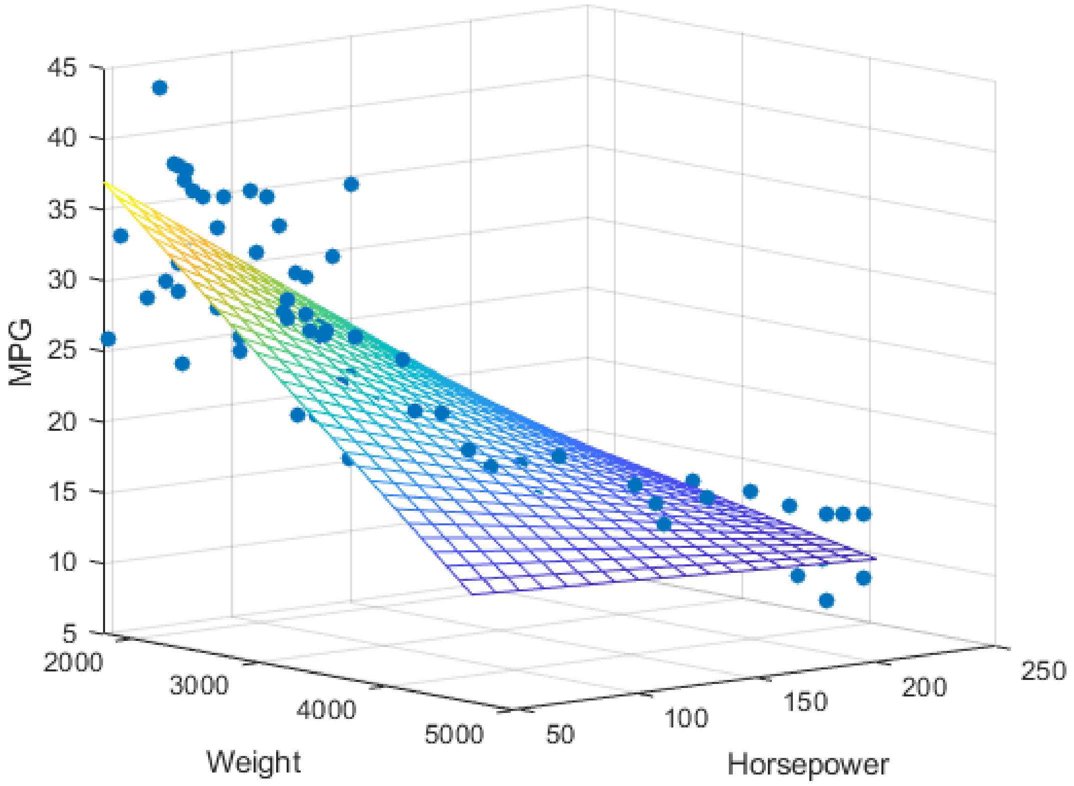
- Learn automatically from the *data*.
- Data may be the table of word counts + class of email :

buy	offer	credit	honey	...	hello	regards	is spam ?
1	1	1	0	...	0	0	yes
0	1	0	0	...	1	0	no
:							:
0	0	0	2	...	1	1	no

- each row is an email and its class : a *sample*, pair (x_i, y_i)
- $x_1 = [0, 0, 1, 3 \ldots 2, 0]$, $y_1 = 0$, `yes` $= 1$, `no` $= 0$
- *supervised* learning : find the parameters of a mapping $f(x) = \hat{y}$ such that $f(x_i) = \hat{y}_i$ **equal** to y_i for most of i's
- $y_i, \hat{y} \in \{0, 1\}$, **discrete**
- f is a **classifier**

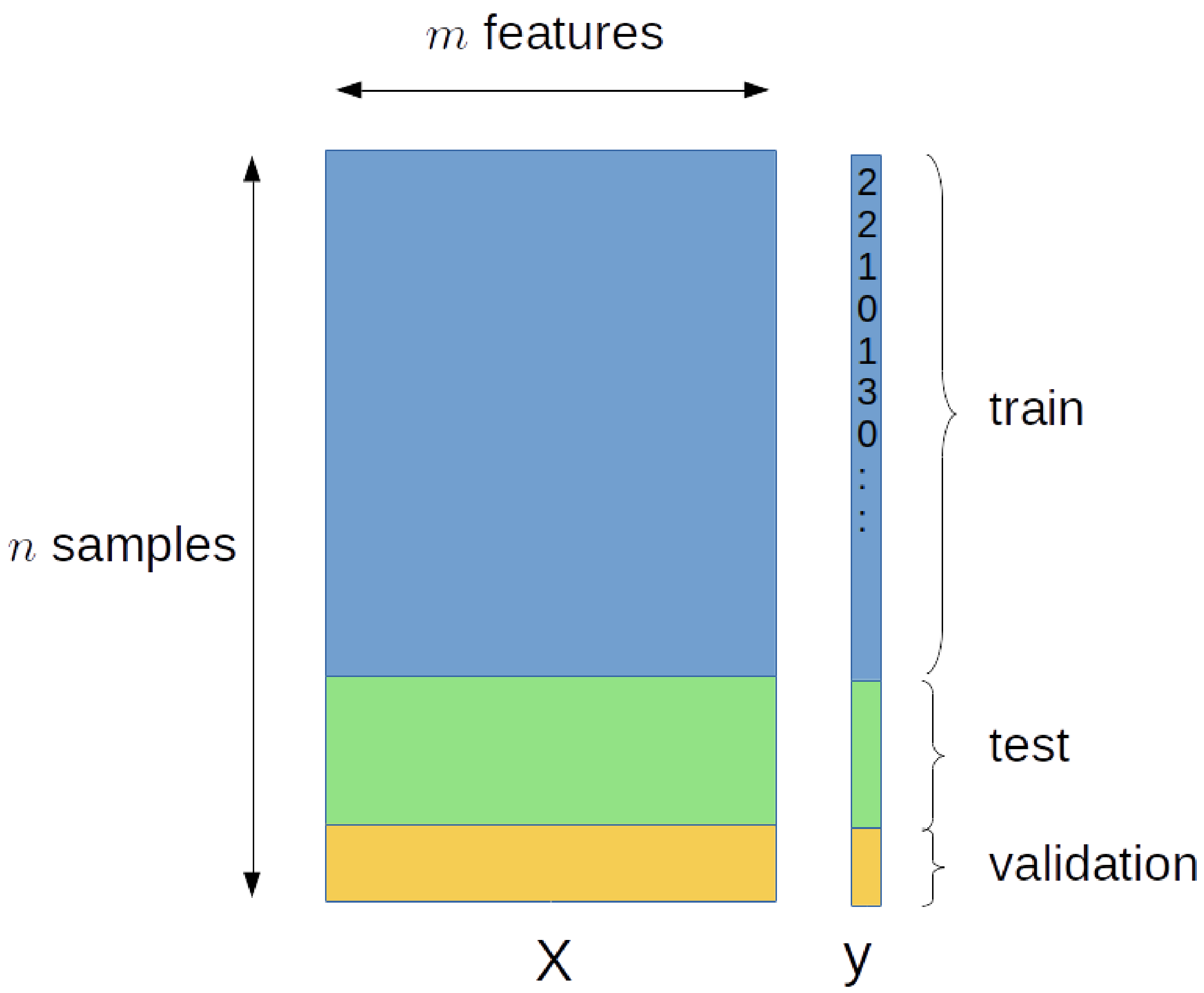
Example regression

- MPG = milles per gallon vs. car weight and horsepower
- $f(hp, w)$ predicts mpg
- we want $\hat{y}_i$ **close** to y_i
- y_i and $\hat{y}$ **continuous**
- f is a **regressor**



Dataset (X, y)

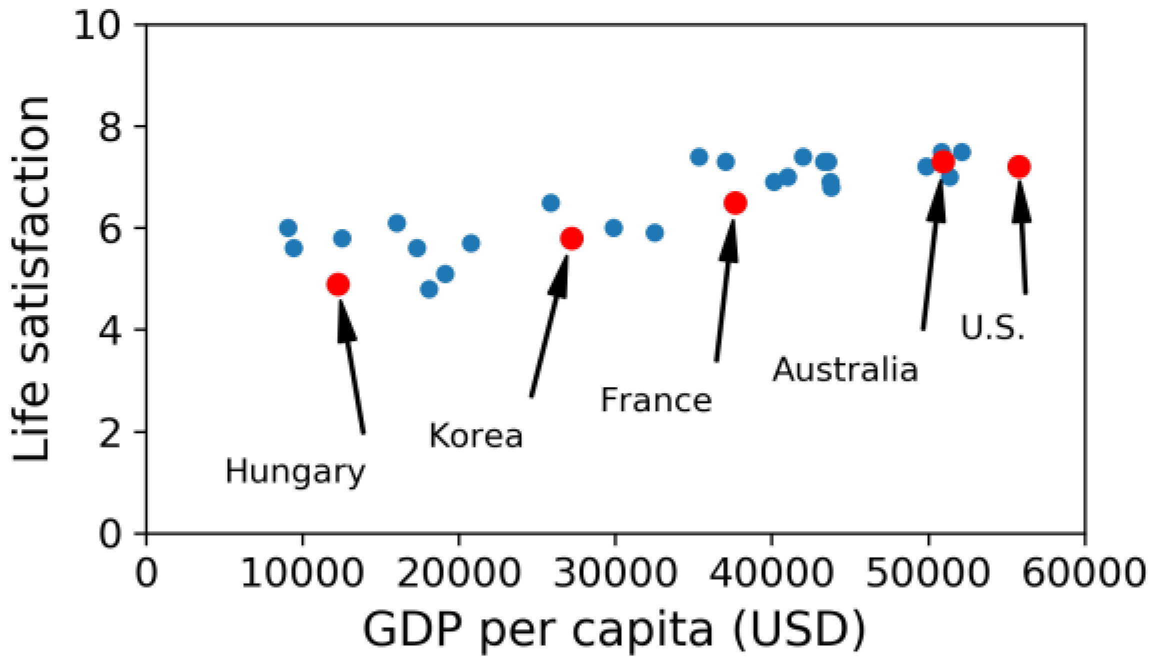
X is a matrix, y is a vector (*groundtruth*)



- train : find parameters of f
- test : does f perform well on *unseen* samples ?
- why a test set ? consider a k-Nearest Neighbors classifier:
  - memorizes all the training samples x_i
  - for an unseen x find its k nearest neighbors $x_{j_1}, x_{j_2} \dots x_{j_k}$ in the training set
  - $\hat{y} =$ most frequent label in $\{y_{j_1}, y_{j_2} \dots y_{j_k}\}$
  - 1-NN is perfect on the training set
- validation : to set the value of *hyperparameters*, but we won't use it for the sake of simplicity and use instead the test set

Hyperparameters

Does money make happiness (at country level) ?



source

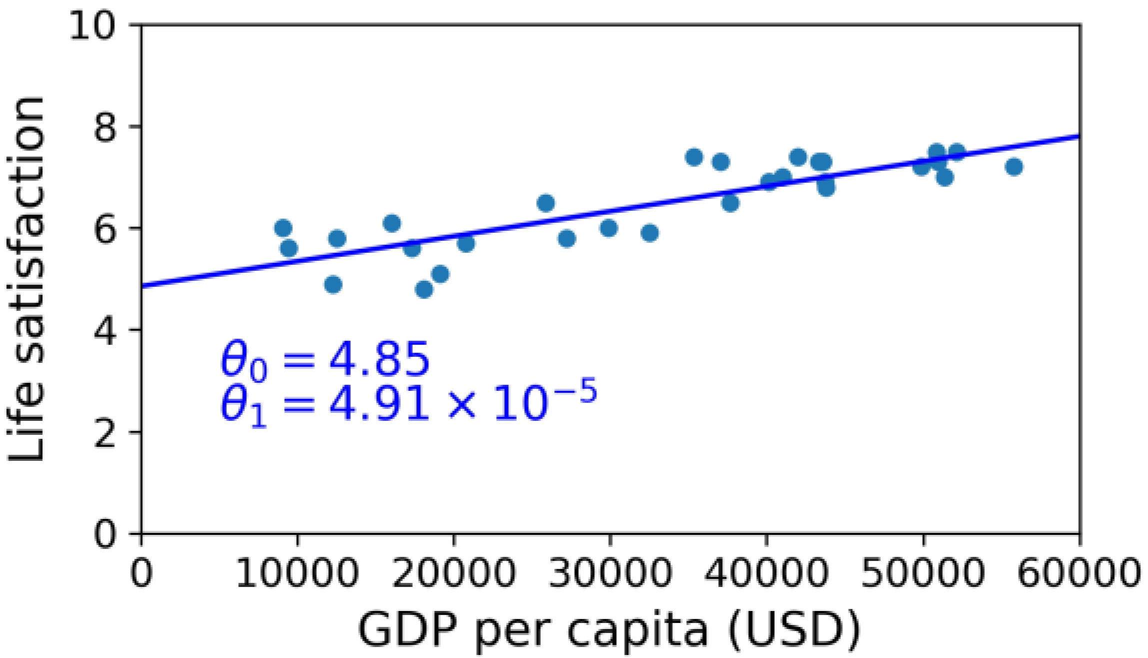
GDP = gross domestic product = *producte interior brut*

GDP per capita = *renta per càpita* = GDP / population

Least squares *linear* regression : θ_0, θ_1 that minimize sum of squared vertical differences

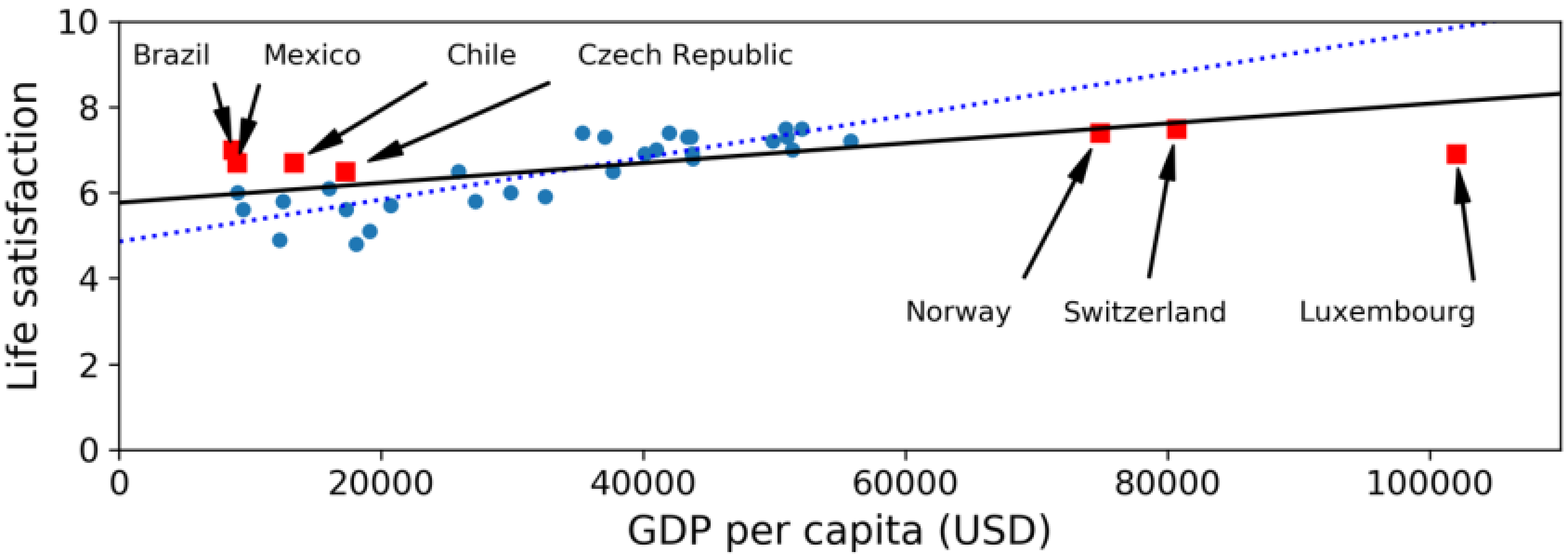
$$\arg\min_{\theta_0,\theta_1} \sum_i ||\theta_1 x_i + \theta_0 - y_i||^2$$

θ_1 slope, θ_0 intersect 4.8



The goal of regression is to predict life satisfaction for *new countries*

New samples ■ arrive once we have learned the model

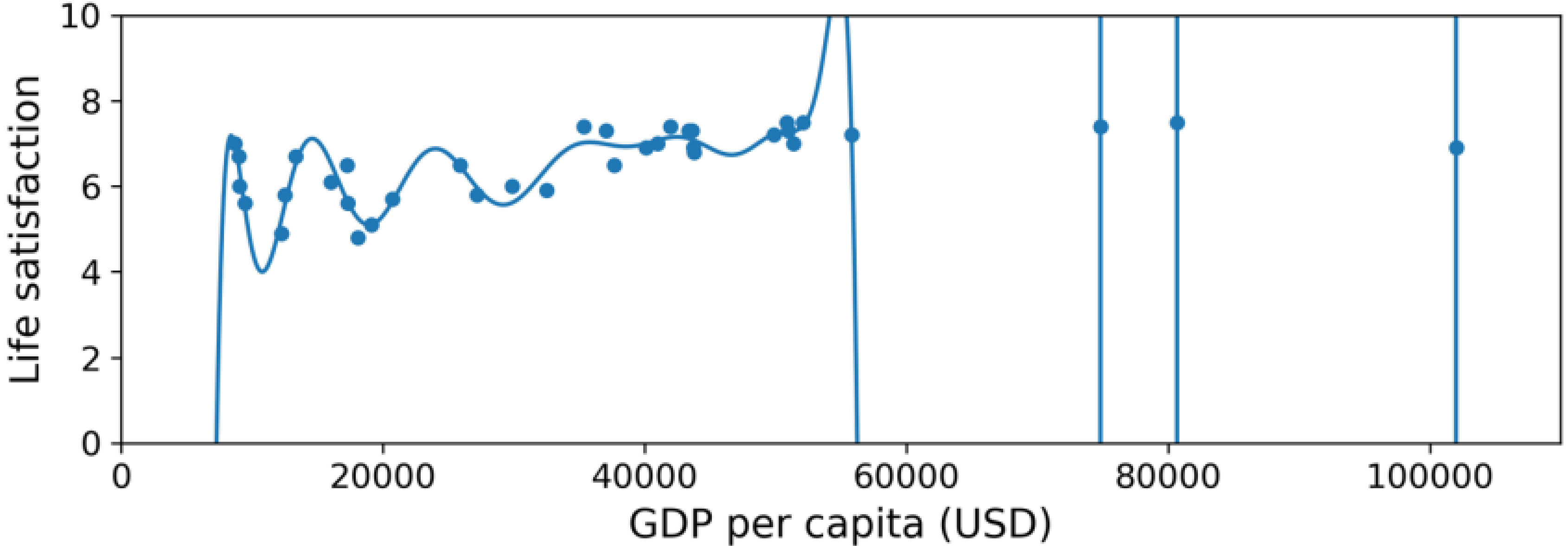


The 3 new richest countries heavily change the learned model (dotted blue line).
Maybe they are **outliers**, samples not following the "normal" distribution.

Is the model form, a straight line, is too simple for this problem ?

Let's try a high-degree polynomial of degree $p = 8$ instead of degree 1 :

$$\text{Life satisfaction} = \sum_{k=0}^{p} \theta_k \cdot GDP^k$$

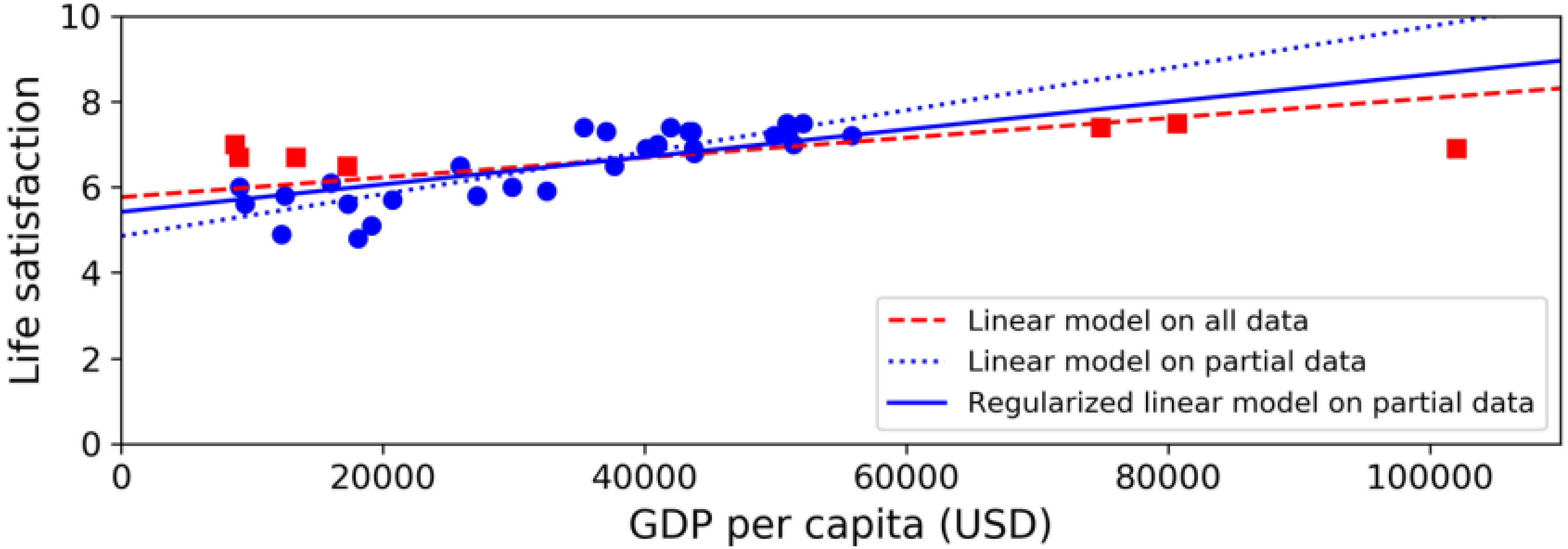


Overfitting: model fits well to training data but won't fit to new samples with GDP $>$ 60K USD or $<$ 10K. The former simple model of a line is better.

Regularization: *constrain* the model f to reduce overfitting and influence of outliers. A simple model is also a way to regularize. Another is hyperparmeters tunning.

Hyperparameters control the amount of regularization, for a given model :

$$\arg\min_{\theta_0,\theta_1} \sum_i ||\theta_1 x_i + \theta_0 - y_i||^2 + \lambda\,\theta_1^2 \tag{1}$$



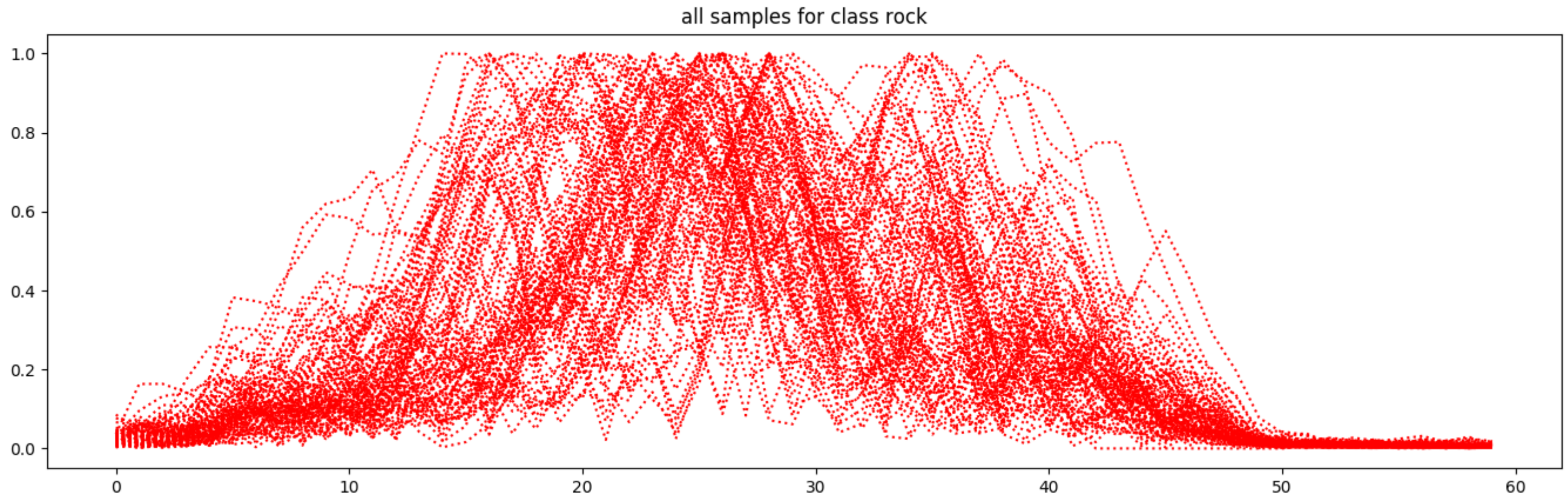
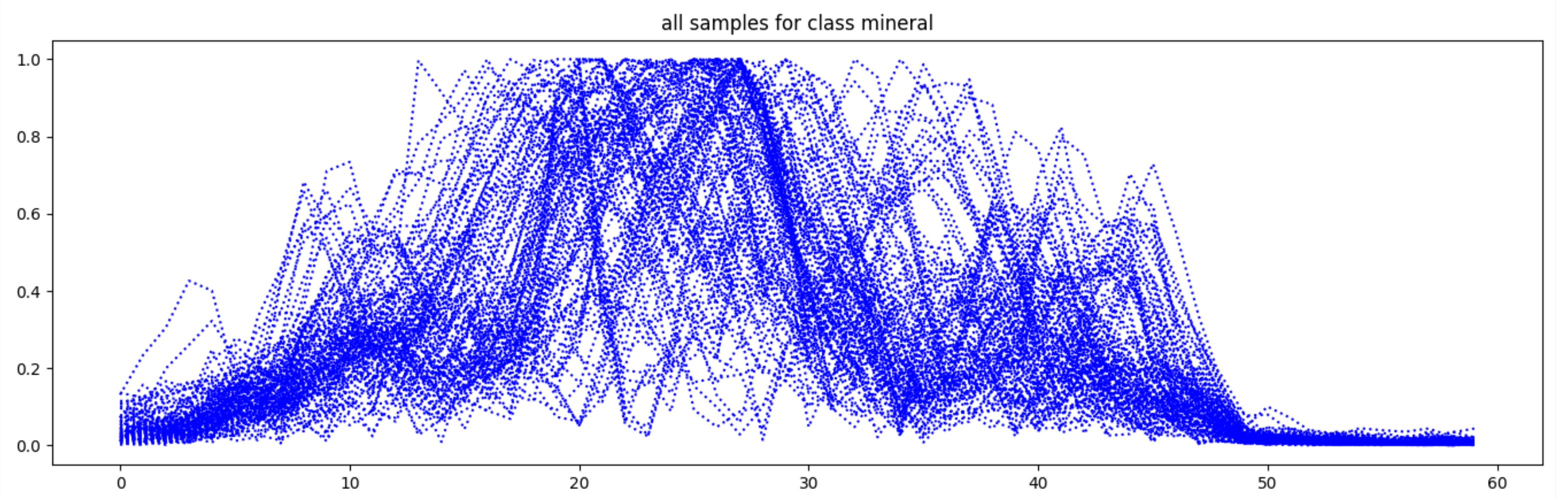
λ is an hyperparameter: a small, positive value makes θ_1 smaller, better fit new data.

Our datasets

The Sonar dataset

- To predict whether or not an object is a mine or a rock given the strength of sonar returns at 60 angles, read description
- 208 samples (rows), 61 columns = 60 angles plus class
- CSV file `sonar.all-data`, class is character `M` or `R`
- used to illustrate a certain messy implementation of random forests in Python that we will improve and extend

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def load_sonar():
  df = pd.read_csv('/home/joan/Downloads/sonar.all-data',header=None)
  X = df[df.columns[:-1]].to_numpy()
  y = df[df.columns[-1]].to_numpy(dtype=str)
  y = (y=='M').astype(int) # M = mine, R = rock
  return X, y

X, y = load_sonar()
idx_rocks = y==0
idx_mines = y==1
plt.close('all')
plt.figure(), plt.plot(X[idx_rocks].T,'b'), plt.title('all samples of class rock')
plt.figure(), plt.plot(X[idx_mines].T,'r'), plt.title('all samples of class mine')
```

all samples for class rock 1.0 0.8 0.6 0.4 0.2 0.0 0 10 20 30 40 50 60	rock
all samples for class mineral 1.0 0.8 0.6 0.4 0.2 0.0 0 10 20 30 40 50 60	mine

The Iris dataset

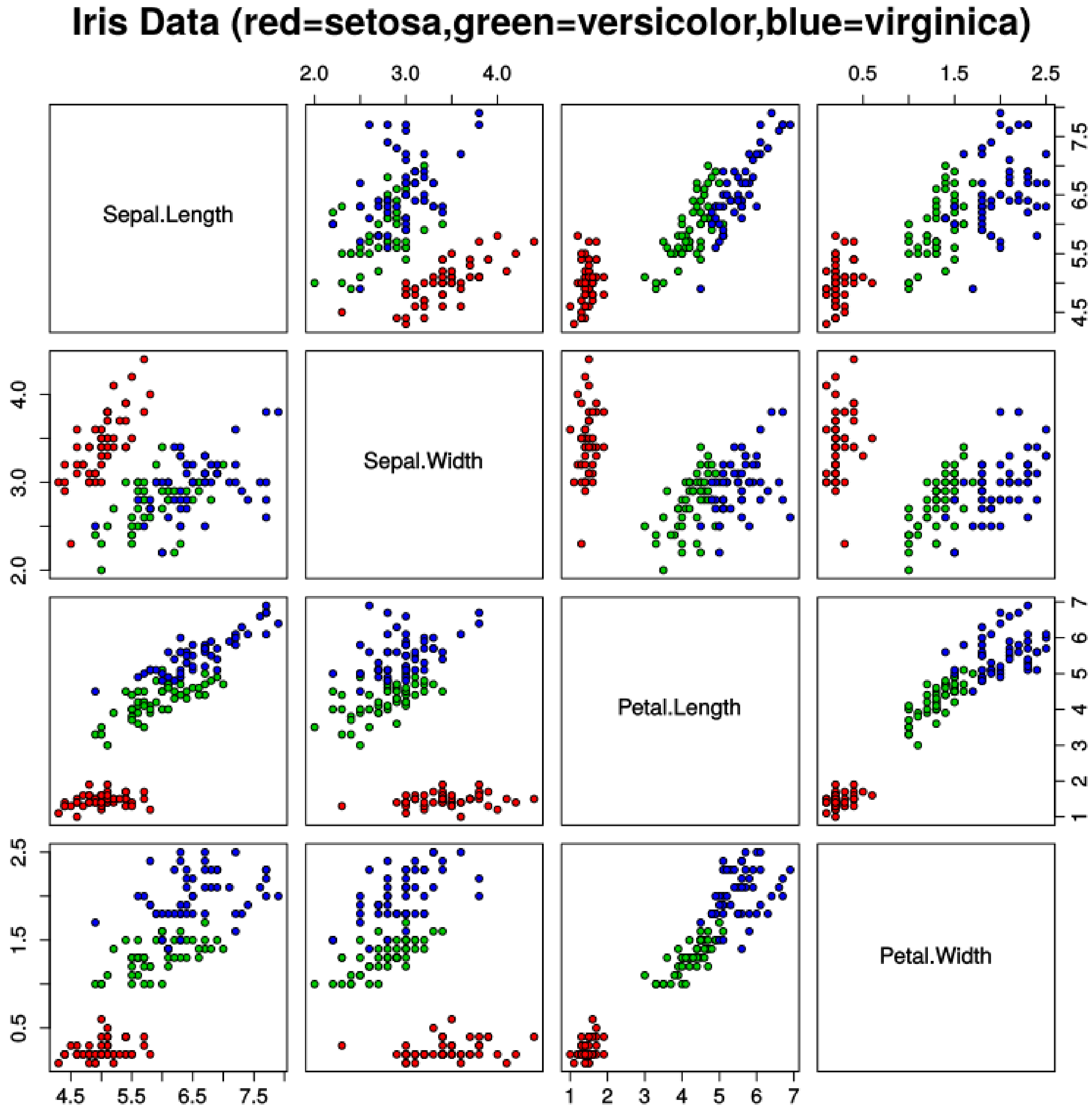
Of a set of 150 flowers, description

- 4 measures or *features* : length, width of sepal, petal
- 3 classes : iris setosa, iris virginica, iris versicolor
- 50 samples per class.

Iris Versicolor **Iris Setosa** **Iris Virginica**

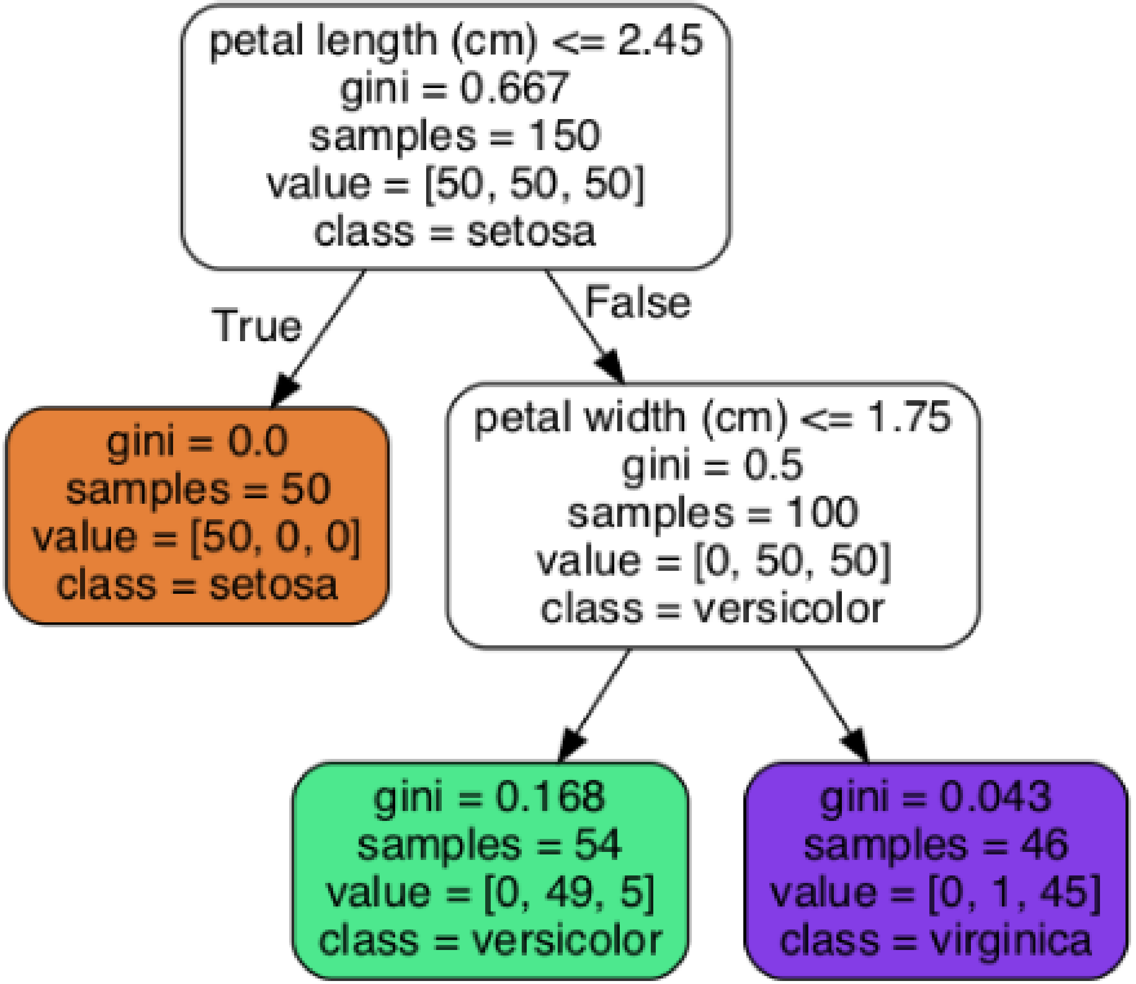
```python
import numpy as np
import sklearn.datasets

iris = sklearn.datasets.load_iris()
print(iris.DESCR)
X, y = iris.data, iris.target
# X 150 x 4, y 150 numpy arrays
```

No pair of features can perfectly separate the 3 classes



Decision trees

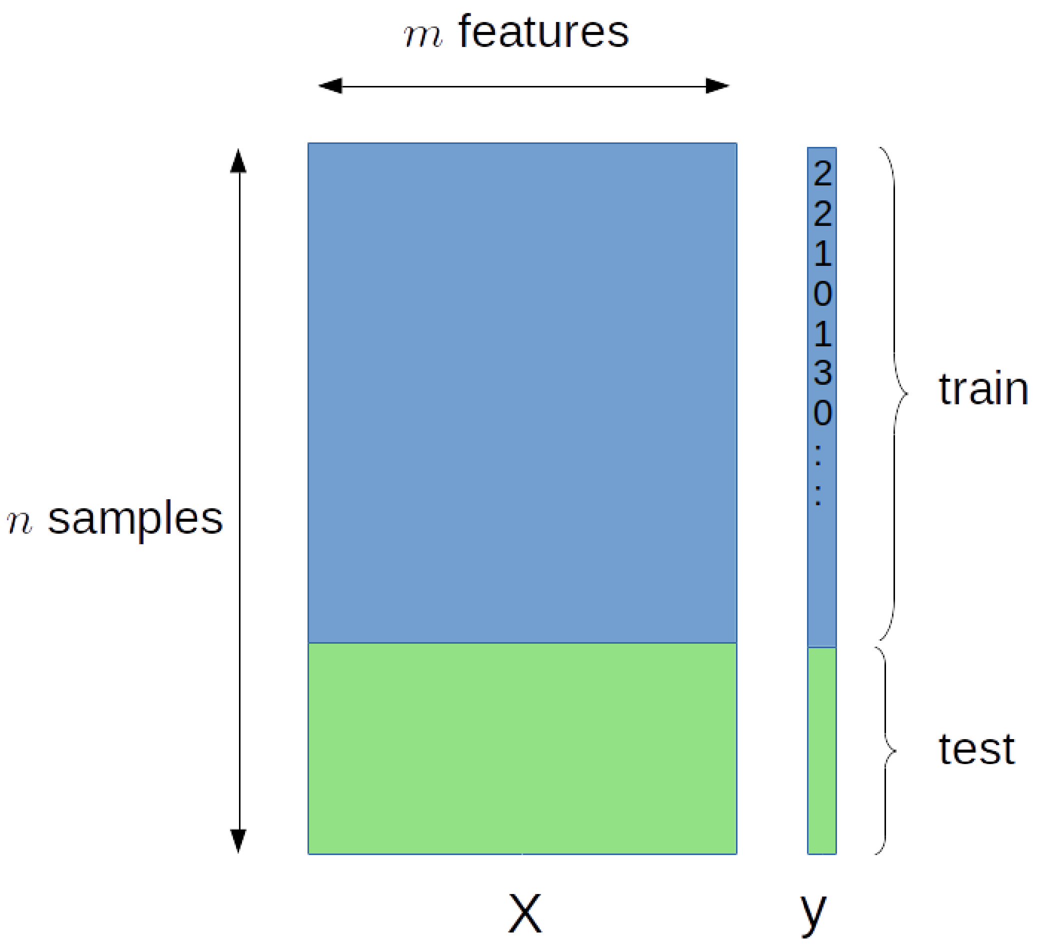
- model for both classification and regression
- a random forest contains a collection of decision trees
- training a decision tree is simple
- classifier is readily interpretable



A decision tree

- parent node: (feature index, threshold)
- leaf node: predicted class

Dataset (X, y) : X is a matrix, y is a vector (*groundtruth*)



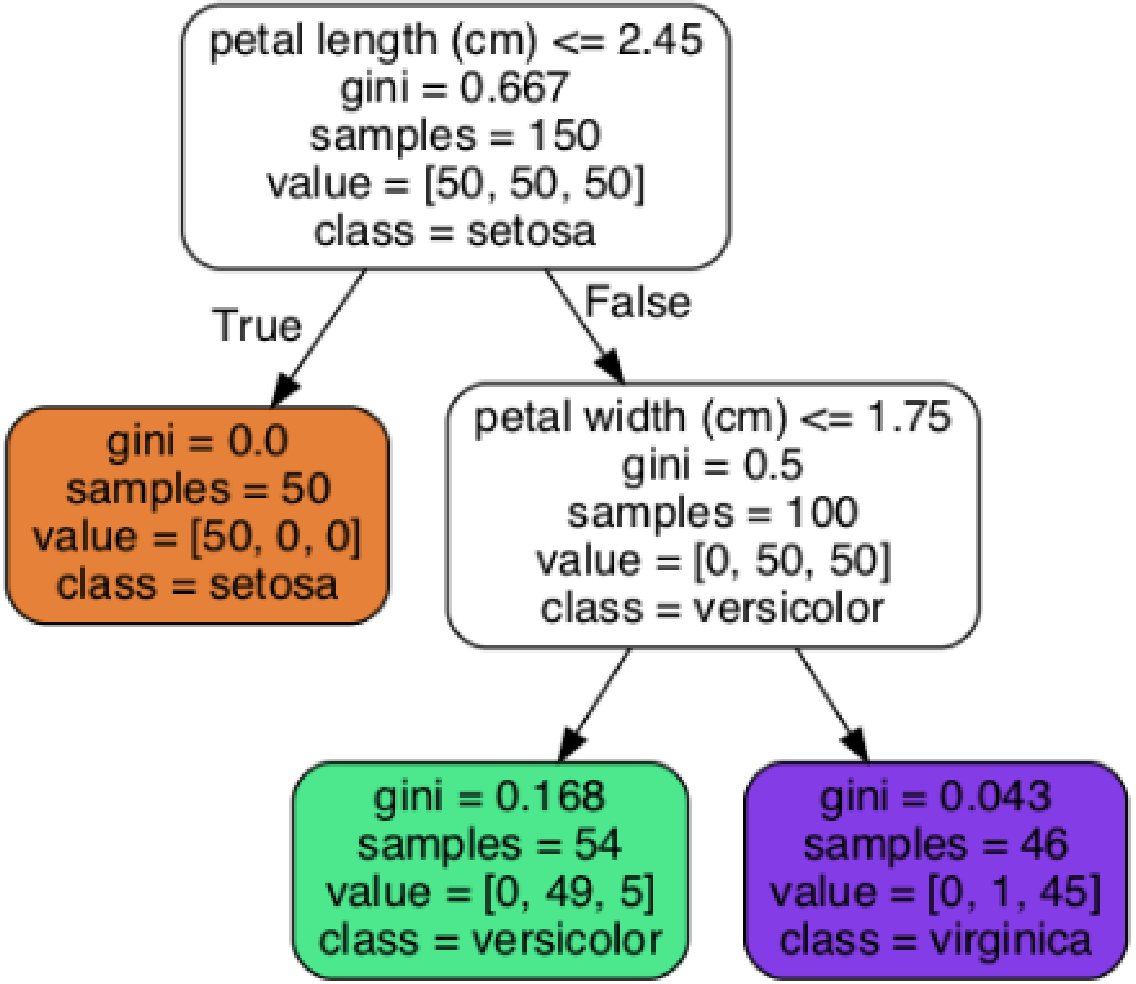
For the sake of simplicity we won't make validation set to optimize the hyperparameters

How to learn = build a decision tree

1. make a root node
2. give it the whole train set (X, y)
3. find the pair feature index k, threshold v, $1 \leq k \leq m$, $v \in X[k]$, such that if

   $$I_{\text{left}} = \{i \mid X[i,k] < v\}\ ,\ I_{\text{right}} = \{i \mid X[i,k] \geq v\}$$

   then $y[I_{\text{left}}]$, $y[I_{\text{right}}]$ are the most "pure" = classes are better separated
4. make a left and right children
5. give $(X[I_{\text{left}}], y[I_{\text{left}}])$ to the left child, and $(X[I_{\text{right}}], y[I_{\text{right}}])$ to the right one
6. for each child, go to step 3 if
   - input dataset has at least `min_size` samples
   - we have not reached `max_depth` depth
   - labels y_I are not all the same



Parameters, learned:

- (petal length, 2.45)
- (petal width, 1.75)

Hyperparameters, set before learning:

- `max_depth` $= 3$
- `min_size` $= 60$

How to measure "most pure" ?

- pure dataset (X, y) : all samples belong to the same class, $y_i = c \; \forall i$
- Gini index is measure of *impurity* of a dataset

$$G = 1 - \sum_{c=1}^{C} p_c^2 \quad , \quad p_c = \frac{1}{m} \sum_{i=1}^{m} 1_{y_i = c}$$

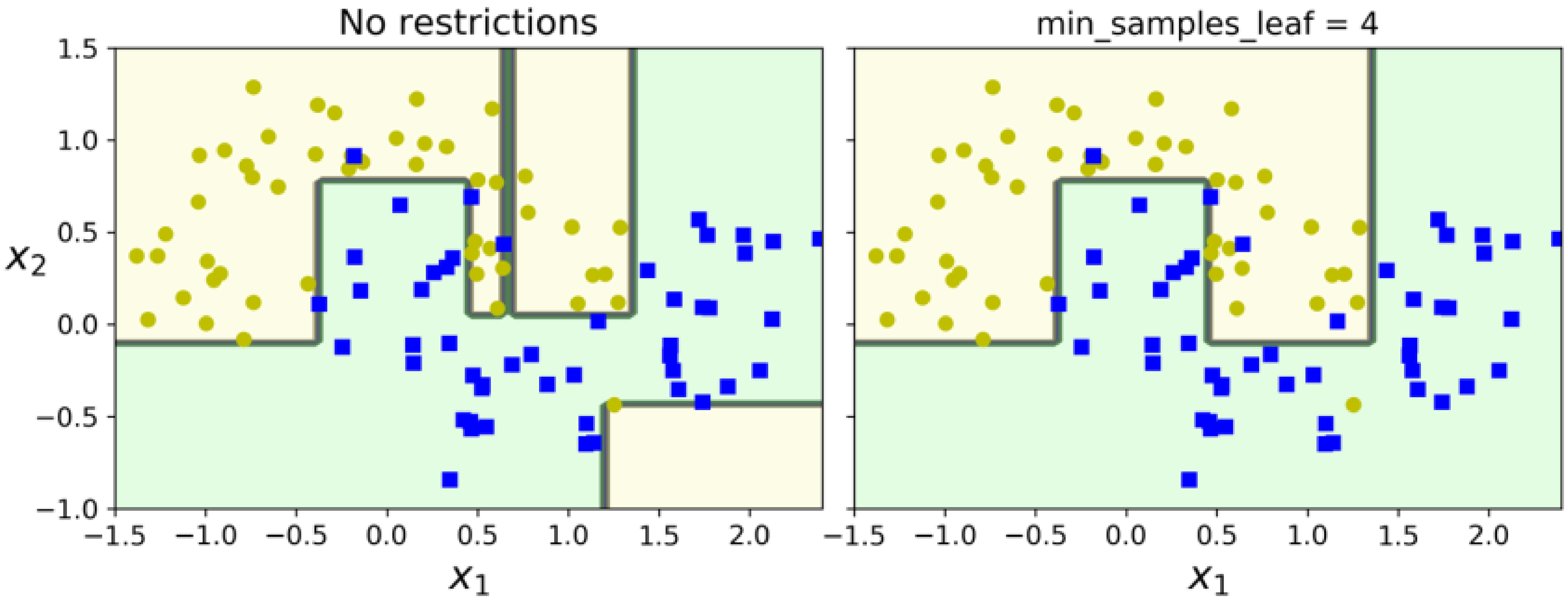
- C number of classes, m number of samples, p_c frequency of class c in y
- best pair (k, v) minimizes this cost function

$$J(k, v) = \frac{m_{\text{left}}}{m_{\text{left}} + m_{\text{right}}} G_{\text{left}} + \frac{m_{\text{right}}}{m_{\text{left}} + m_{\text{right}}} G_{\text{right}}$$

Entropy is an alternative measure of impurity

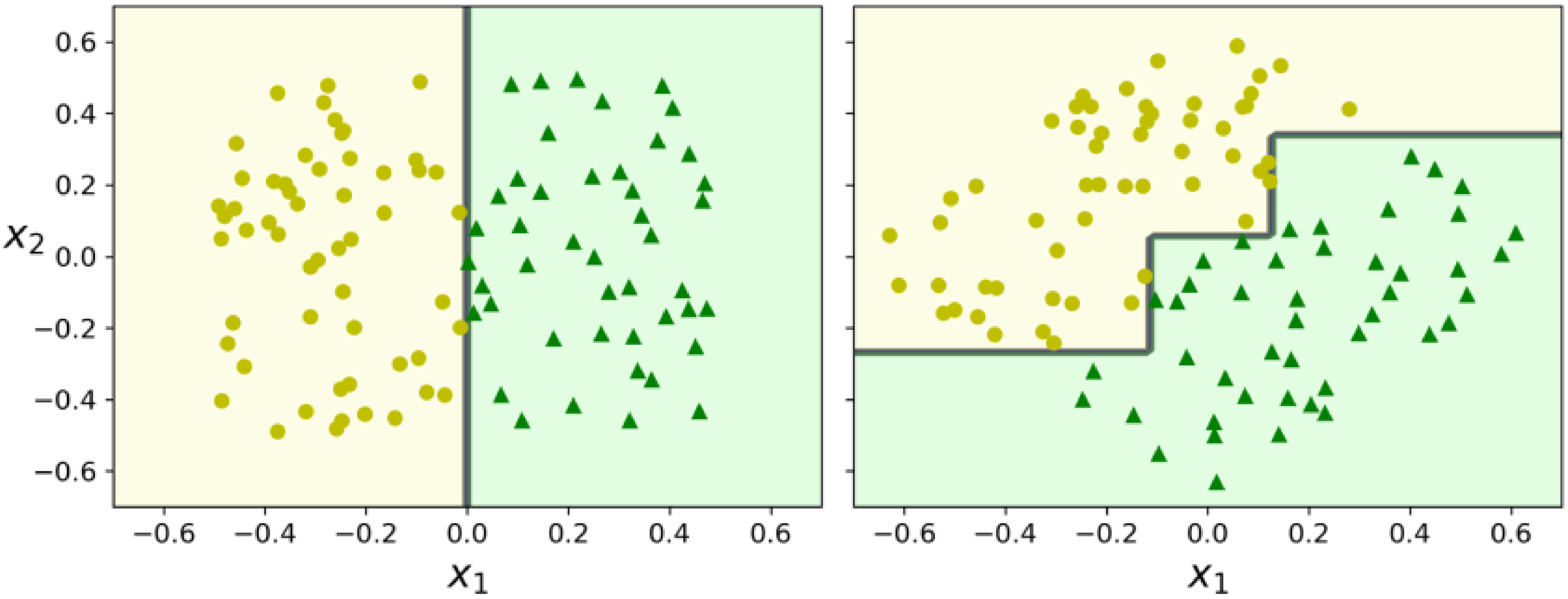
$$H = - \sum_{c=1,\ p_c>0}^{C} p_c \log p_c$$

Regularization in decision trees



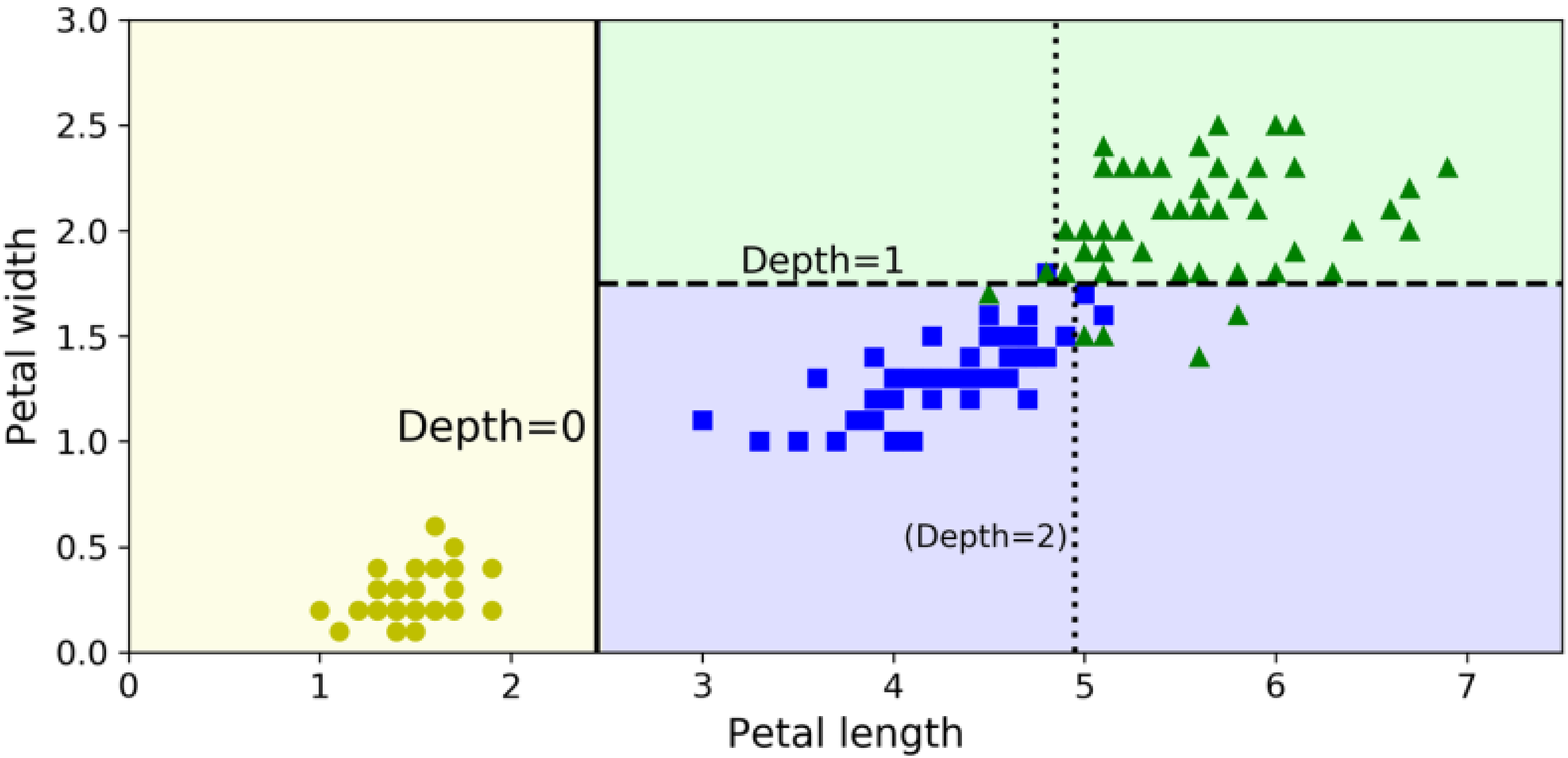
Note: decision boundaries are perpendicular to axes because at each node we ask if $x[k] < v$ or $x[k] \geq v$ to go left or right

Instability

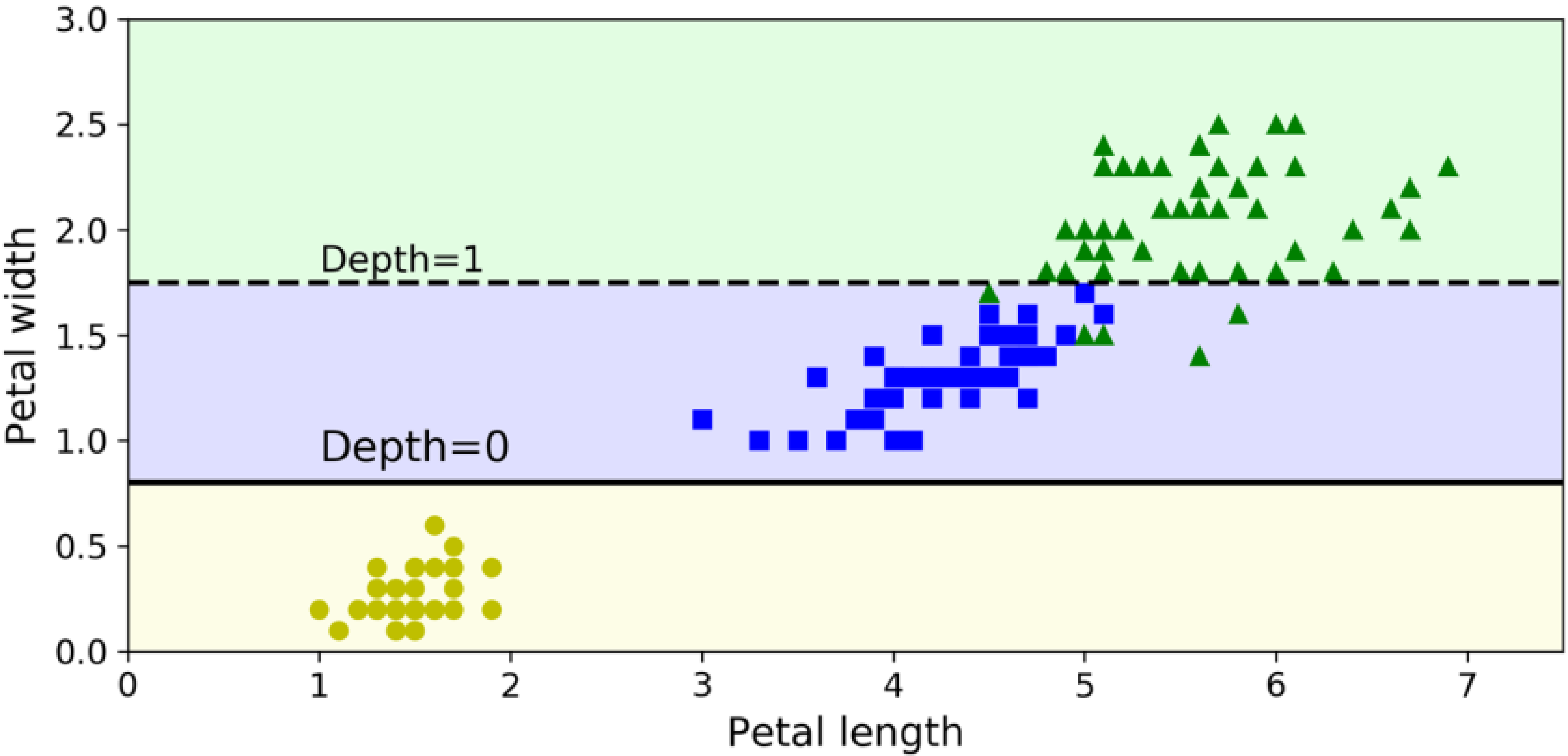


Just rotating the samples makes the problem more difficult, won't generalize well probably

Too sensitive to small data variations

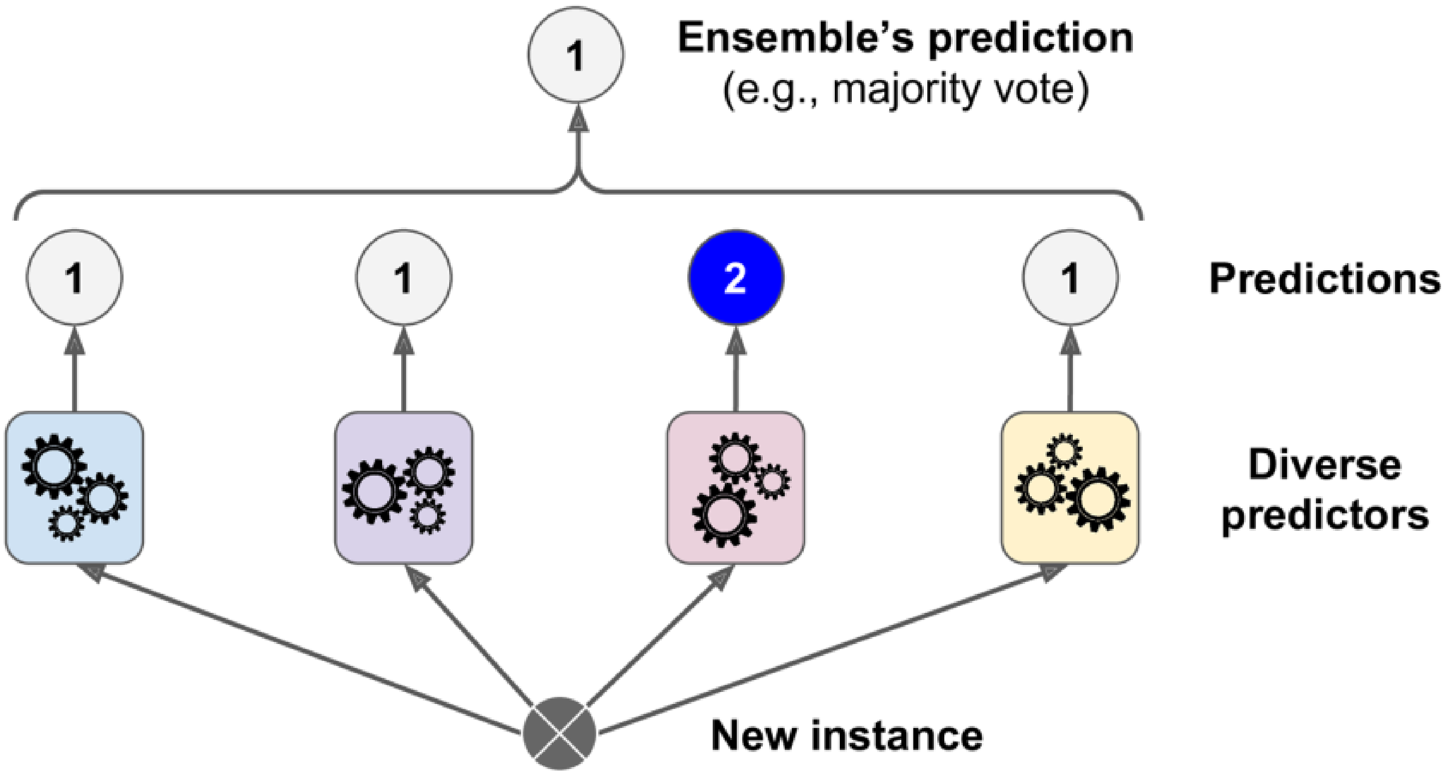


Too sensitive to small data variations



Random Forests

- An **ensemble classifier / regressor** combines the output of a set of different classifiers / regressors
- For instance by **majority voting / average**



Idea : the **combination of predictions from a collection of *different* experts is better that the prediction of the single best expert**

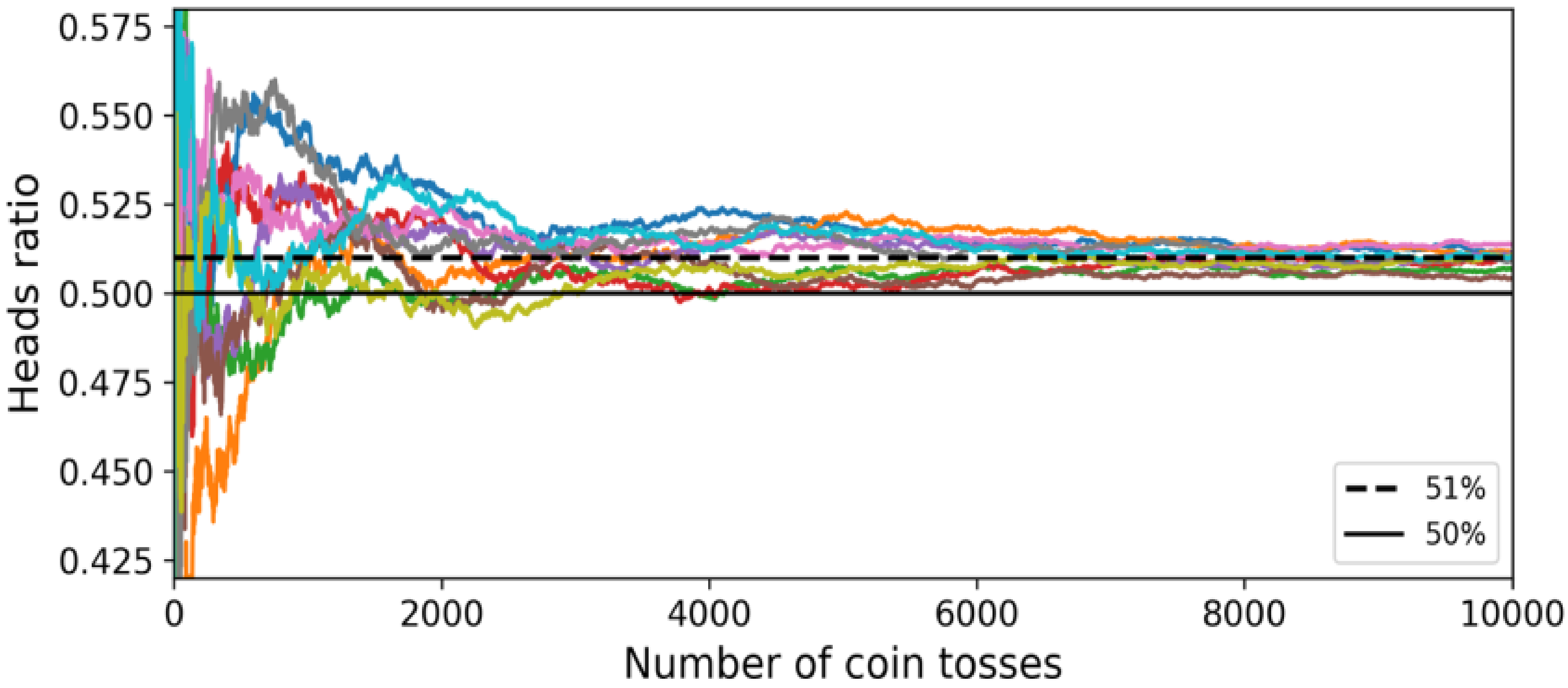
why ?

- supose we have a biased coin[1], $p(head) = 0.51$, $p(tail) = 0.49$
- but you don't know it
- is the coin biased towards heads ?
- toss it 1000 times, $p(\#heads > \#tails) \approx 0.75$
- 10,000 times, $p(\#heads > \#tails) \approx 0.97$

[1] see cumulative distribution function of a binomial distribution

Analogy

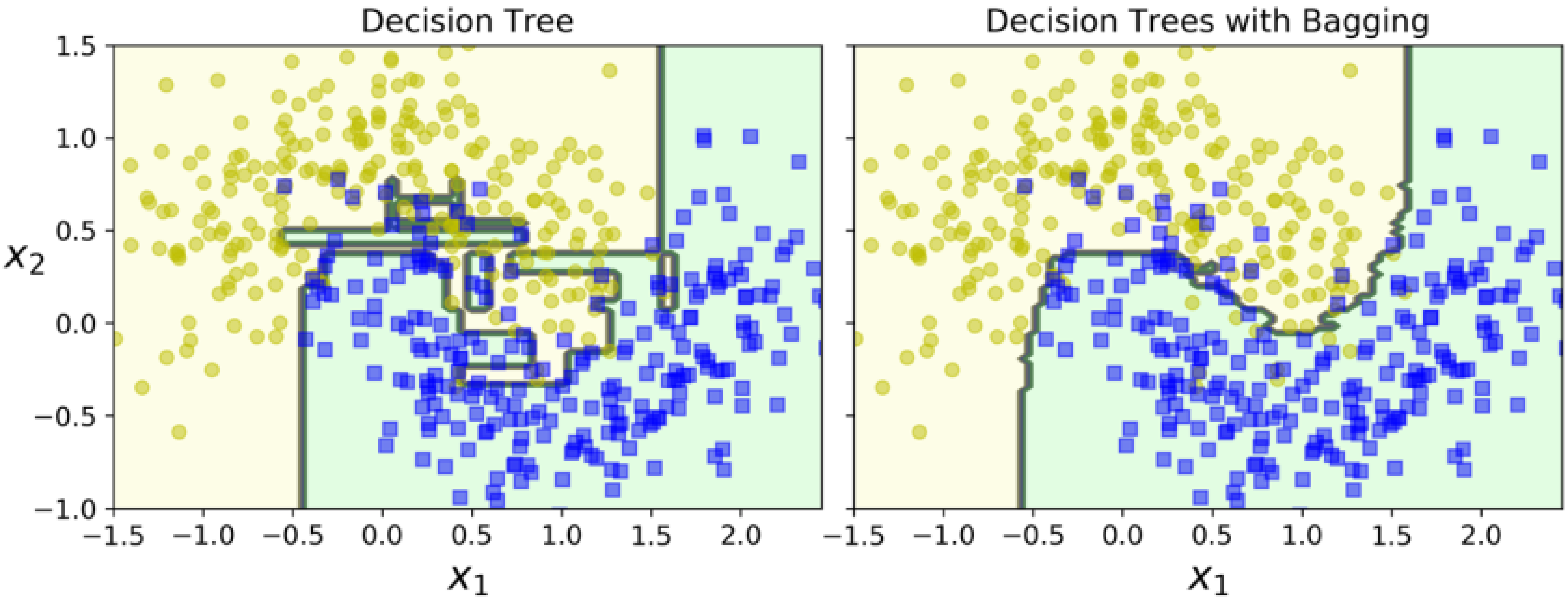
- 1 toss of the coin, if *head* it's biased, if *tail* it's not
- 1 toss $\equiv$ prediction by a weak binary classifier, only 51% accurate
- toss 1000 times, if #heads $>$ #tails it's biased $\equiv$ majority voting of 1000 *independent* binary weak classifiers, 75% accurate
- 10,000 weak *independent* classifiers make a very strong ensemble classifier, 97% accuracy



experiment of tossing the coin 10,000 times repeated 10 times

Random forest classifier

- is a set of (quite) *independent* decision trees
- combines their predictions by majority voting
- works very well in many cases!



1 decision tree vs. ensemble of 500 decision trees

Diversity of classifiers

- This is true only if the decision trees are *independent*, unrelated to each other
- But they are not because they use the same dataset and learning method
- Random forests enforce independence by
  - each decision tree is trained with a different subset of the training set, like 70%
  - each subset is randomly sampled **with replacement** $\rightarrow$ may have repeated samples
  - at each node do not consider all the n features but a random subset, like $\sqrt{n}$ of them

Hyperparameters

- from decision trees
  - `min_size`
  - `max_depth`
- new :
  - `ratio_samples` (0.7)
  - `num_features_node` ($\sqrt{n}$)
  - `num_trees` number of decision trees
  - `criterion_name` : `"gini"` or `"entropy"`