Lecture 3 – A Tour of Machine Learning Classifiers Using scikit-learn

Andre E. Lazzaretti

Introduction

- Introduction to robust and popular algorithms for classification, such as logistic regression, support vector machines, and decision trees.

- Examples and explanations using the scikit-learn machine learning library, which provides a wide variety of machine learning algorithms via a user friendly Python API.

- Discussions about the strengths and weaknesses of classifiers with linear and non-linear decision boundaries.

Choosing a Classification Algorithm

- No single classifier works best across all possible scenarios

- Main steps:
  - Selecting features and collecting training samples.
  - Choosing a performance metric.
  - Choosing a classifier and optimization algorithm.
  - Evaluating the performance of the model.
  - Tuning the algorithm.

First steps with scikit-learn

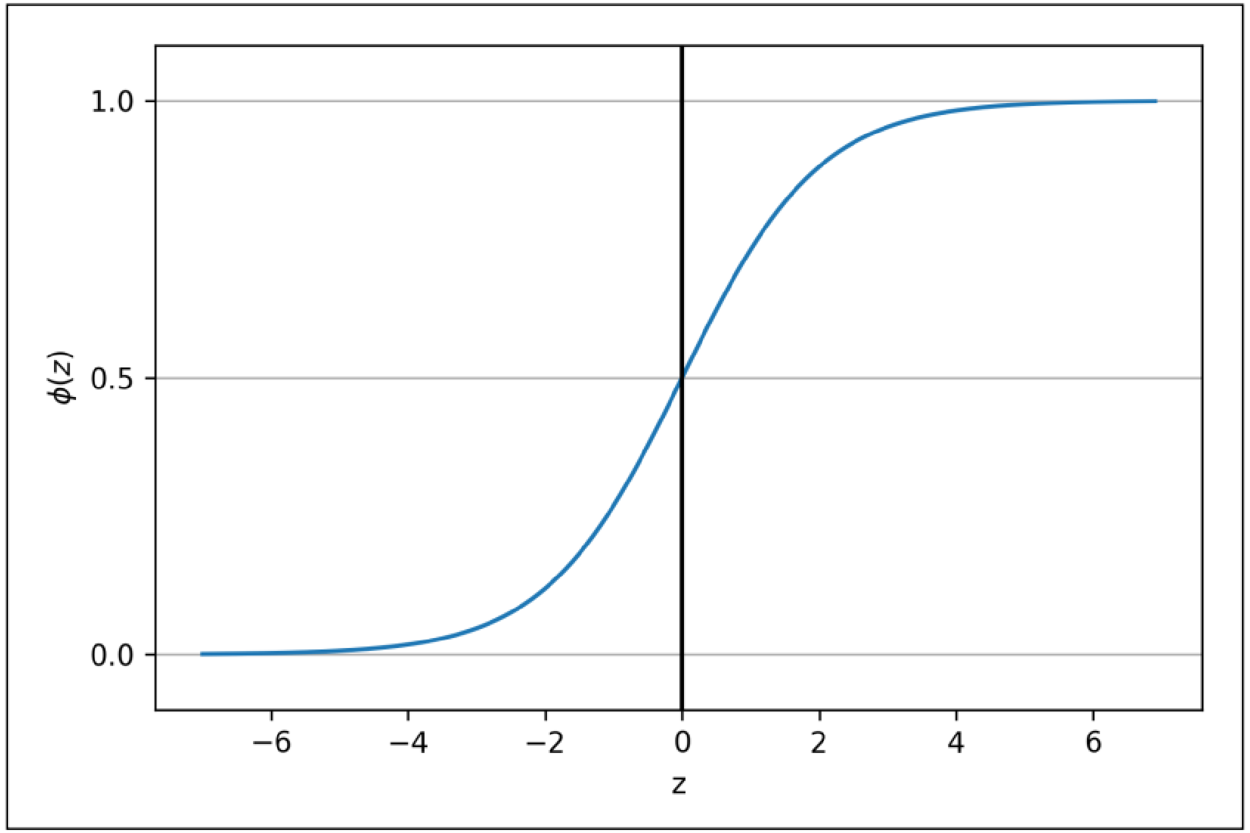
- Train and test perceptron model in Python with scikit-learn API

Logistic Regression

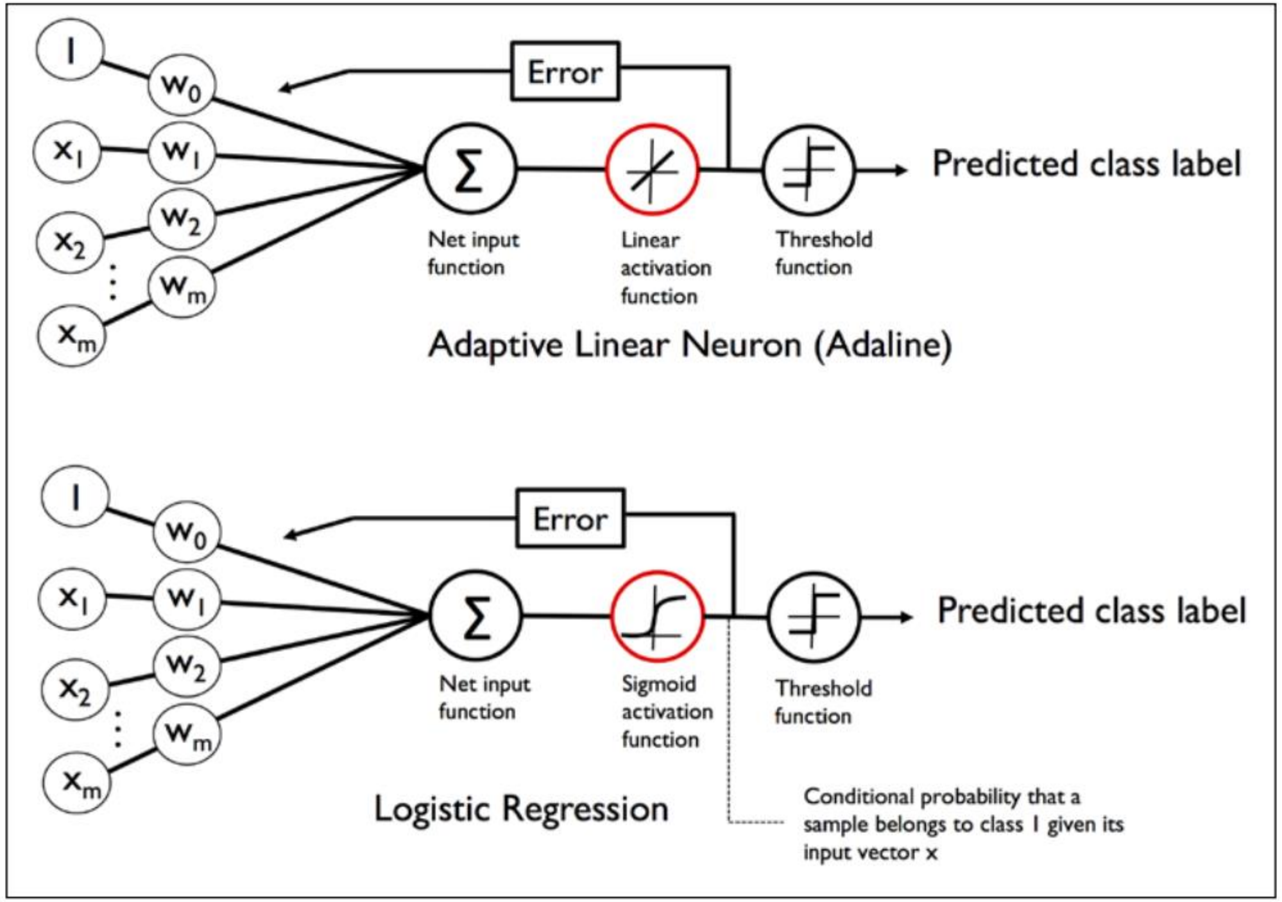
$$\phi(z) = \frac{1}{1+e^{-z}}$$

$$z = \boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m$$

- We are interested in predicting the probability that a certain sample belongs to a particular class.
- To do that, we use the **logistic sigmoid function**, sometimes simply abbreviated to **sigmoid function** due to its characteristic S-shape:



Logistic Regression



Logistic Regression

- The output of the sigmoid function is then interpreted as the probability of a particular sample belonging to class 1, $\varphi(z)=P(y=1|\boldsymbol{x};\boldsymbol{w})$, given its features **x** parameterized by the weights **w**.
- For example, if we compute $\varphi(z)=0.8$ for a particular flower sample, it means that the chance that this sample is an *Irisversicolor* flower is 80%.
- The predicted probability can then simply be converted into a binary outcome via a threshold function:

$$\hat{y} = \begin{cases} 1 & if\, \phi(z) \geq 0.5 \\ 0 & otherwise \end{cases} \qquad \hat{y} = \begin{cases} 1 & if\ z \geq 0.0 \\ 0 & otherwise \end{cases}$$

Logistic Regression – Cost Function

- Likelihood function and log-likelihood:

$$L(\boldsymbol{w}) = P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{w}) = \prod_{i=1}^{n} P\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

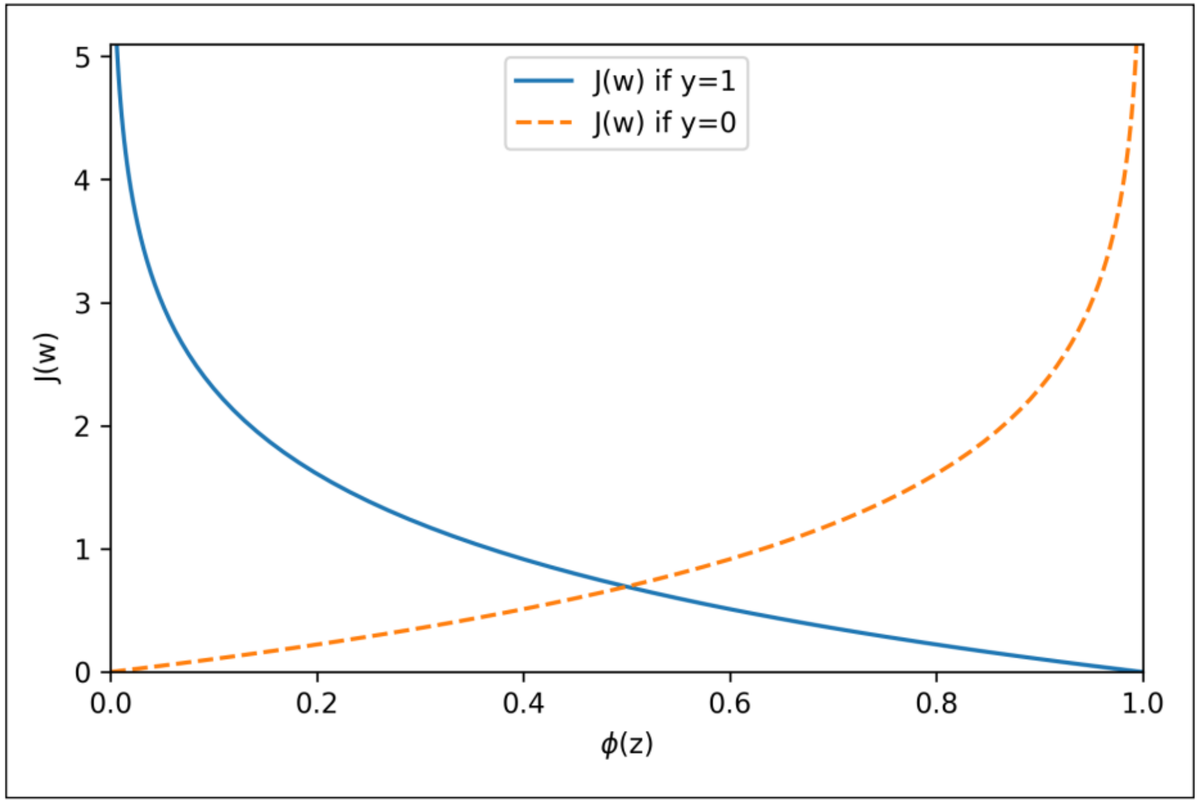
$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left[-y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$

Cost Function – Single Example

$$J(\phi(z), y; \boldsymbol{w}) = -y\log(\phi(z)) - (1-y)\log(1-\phi(z))$$

- Looking at the equation, we can see that the first term becomes zero if *y* = 0, and the second term becomes zero if *y*=1:

$$J(\phi(z), y; \boldsymbol{w}) = \begin{cases} -\log(\phi(z)) & \text{if } y = 1 \\ -\log(1-\phi(z)) & \text{if } y = 0 \end{cases}$$



Gradient for Logistic Regression

$$J\left(\phi(z), y; \boldsymbol{w}\right) = -y\log\left(\phi(z)\right) - (1-y)\log\left(1-\phi(z)\right)$$

$$\frac{\partial}{\partial w_j} l(\boldsymbol{w}) = \left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)}\right)\frac{\partial}{\partial w_j}\phi(z)$$

$$\frac{\partial}{\partial z}\phi(z) = \frac{\partial}{\partial z}\frac{1}{1+e^{-z}} = \frac{1}{\left(1+e^{-z}\right)^2}e^{-z} = \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right)$$
$$= \phi(z)\left(1-\phi(z)\right)$$

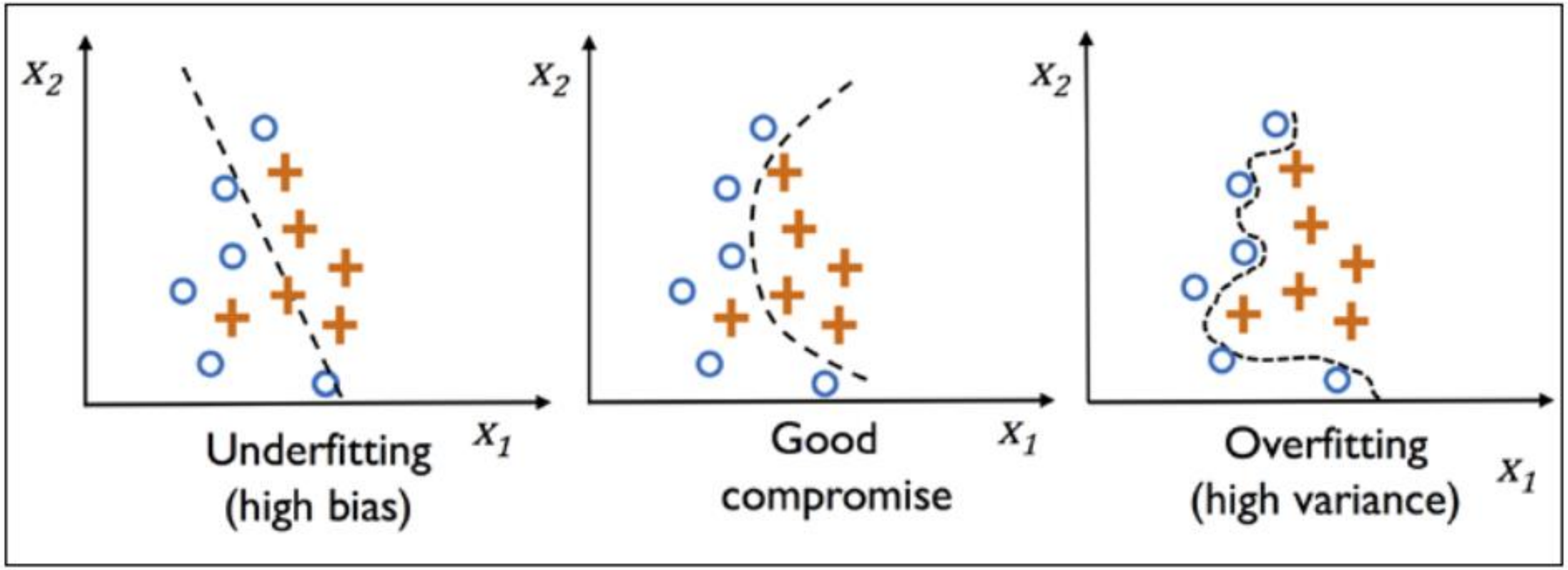
$$\left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)}\right)\frac{\partial}{\partial w_j}\phi(z)$$
$$= \left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)}\right)\phi(z)\left(1-\phi(z)\right)\frac{\partial}{\partial w_j}z$$
$$= \left(y\left(1-\phi(z)\right) - (1-y)\phi(z)\right)x_j$$
$$= \left(y-\phi(z)\right)x_j$$

$$w_j := w_j + \eta\sum_{i=1}^{n}\left(y^{(i)} - \phi\left(z^{(i)}\right)\right)x_j^{(i)}$$

Logistic Regression using scikit-learn

- Train and test LR model in Python with scikit-learn API (IRIS dataset)

Overfitting Regularization



Regularization

- Penalize extreme parameter (weight) values. The most common form of regularization is so-called L2 regularization (sometimes also called L2 shrinkage or weight decay), which can be written as follows:

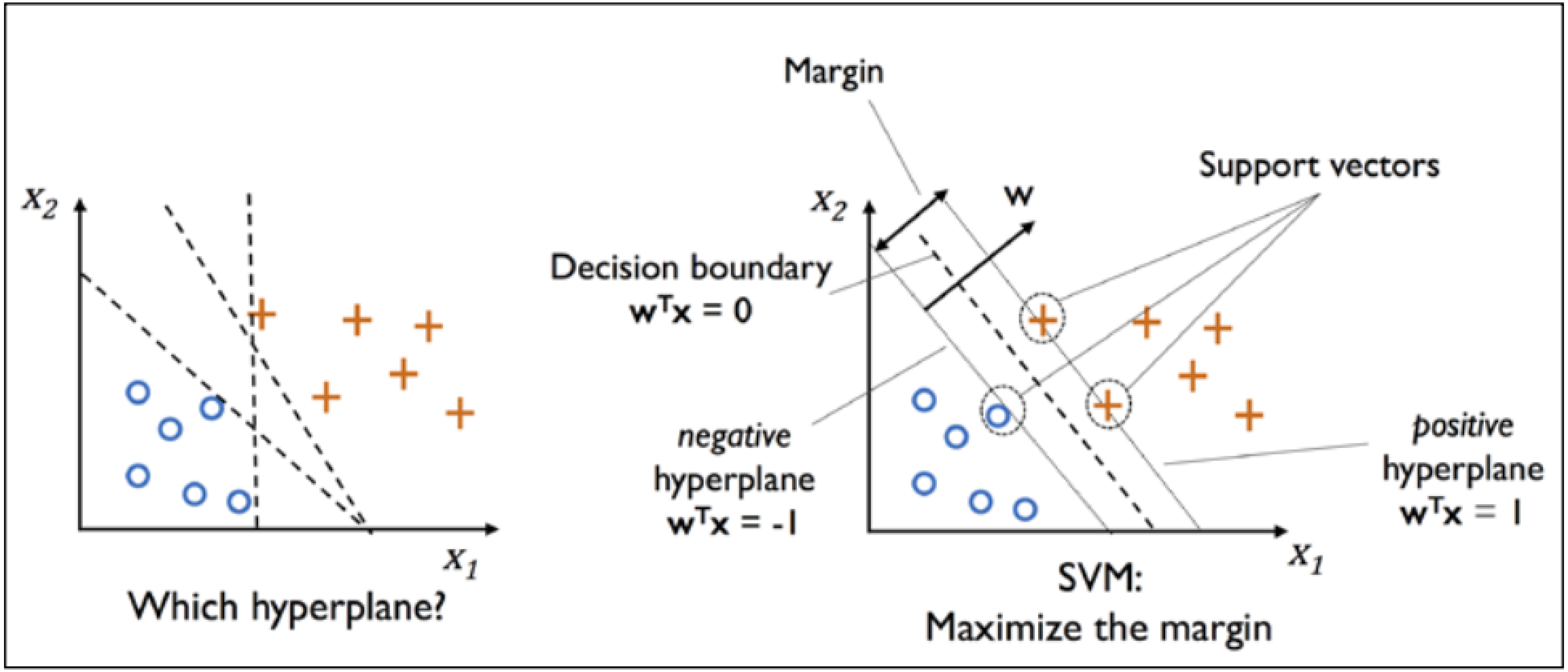
$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

$$J(\boldsymbol{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right] + \frac{\lambda}{2}\|w\|^2$$

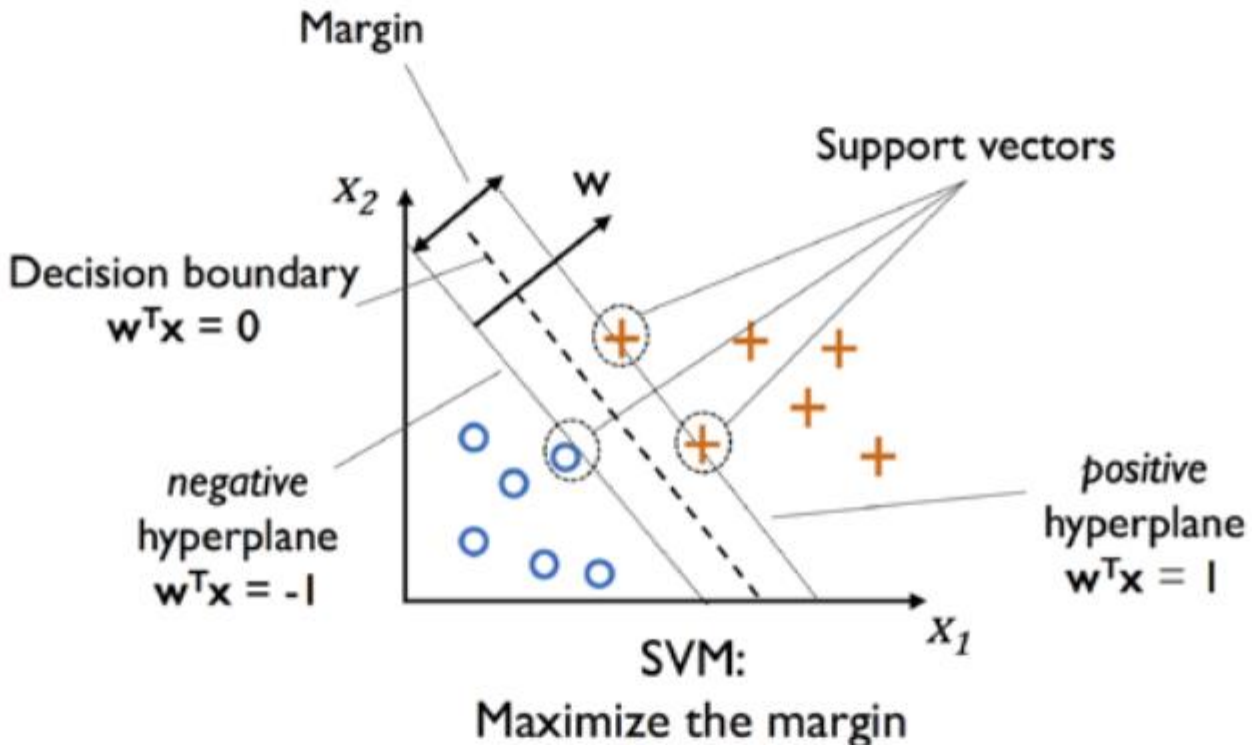
Logistic Regression using scikit-learn

- Including regularization...

Maximum Margin with Support Vector Machines (SVMs)



Maximum Margin Intuition



$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1$$

$$\Rightarrow \boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right) = 2$$

$$\|\boldsymbol{w}\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

$$\frac{\boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

SVM optimization

- The left side of the preceding equation can then be interpreted as the distance between the positive and negative hyperplane, which is the so-called **margin** that we want to maximize. Now, the objective function of the SVM becomes the maximization of this margin by:

$$\text{maximizing } \frac{2}{\|\boldsymbol{w}\|}$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \ if \ y^{(i)} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq -1 \ if \ y^{(i)} = -1$$

$$\text{for } i = 1 \ldots N$$

Dealing with a nonlinearly separable case using slack variables

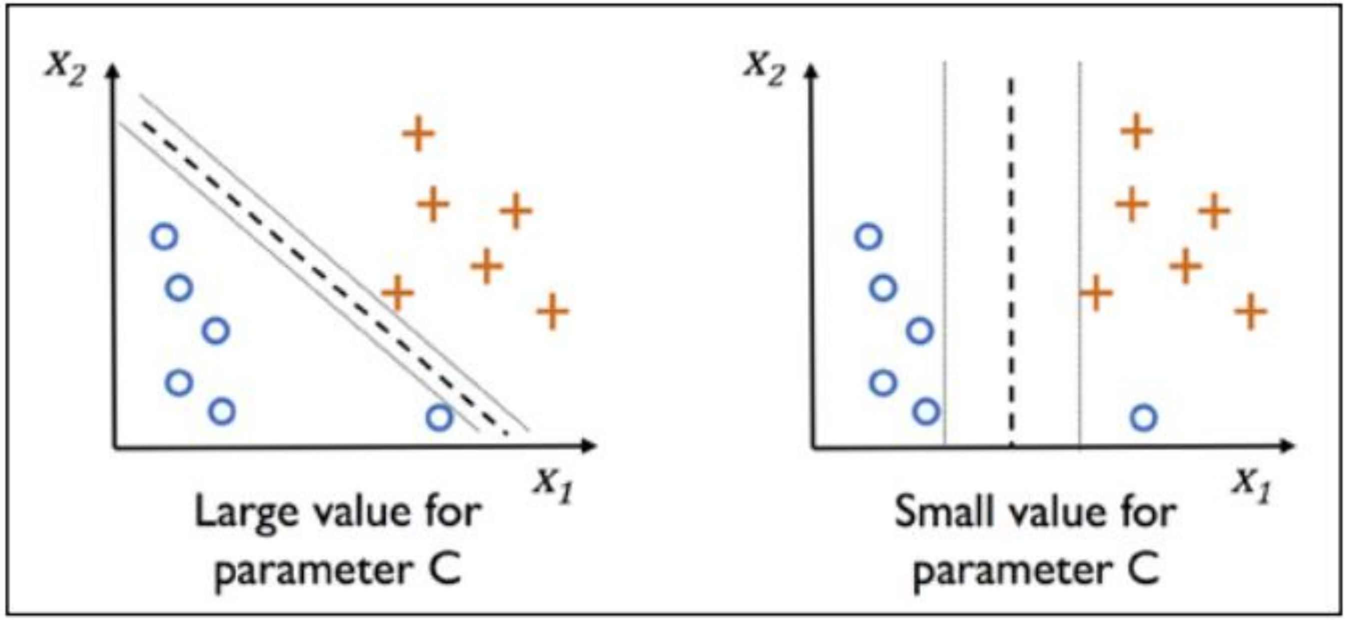
- **Soft-margin classification**:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 - \xi^{(i)} \quad if \;\; y^{(i)} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq -1 + \xi^{(i)} \quad if \;\; y^{(i)} = -1$$

$$\text{for } i = 1 \ldots N$$

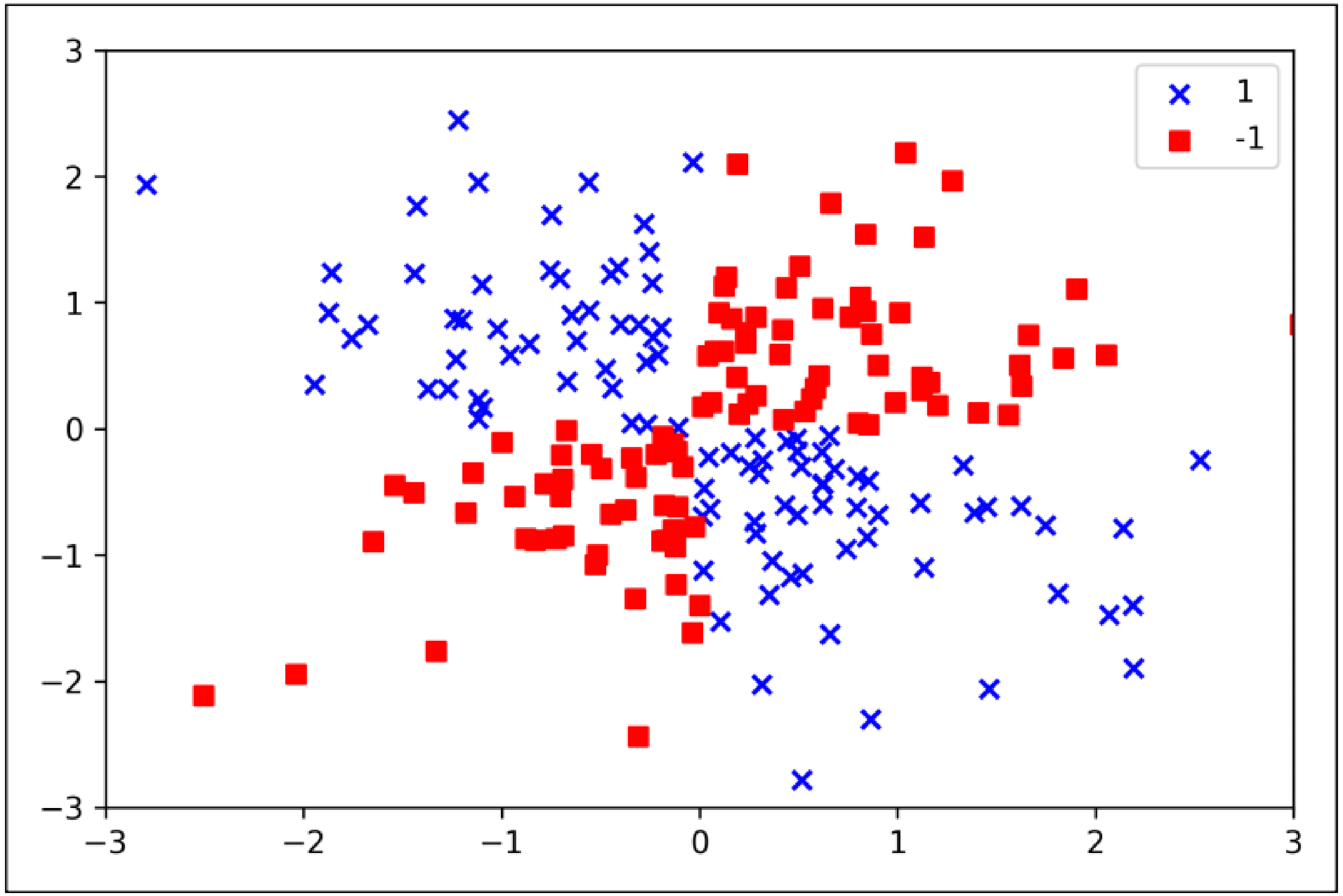
$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)}\right)$$



SVM using scikit-learn

- IRIS dataset...

Solving nonlinear problems using a kernel SVM

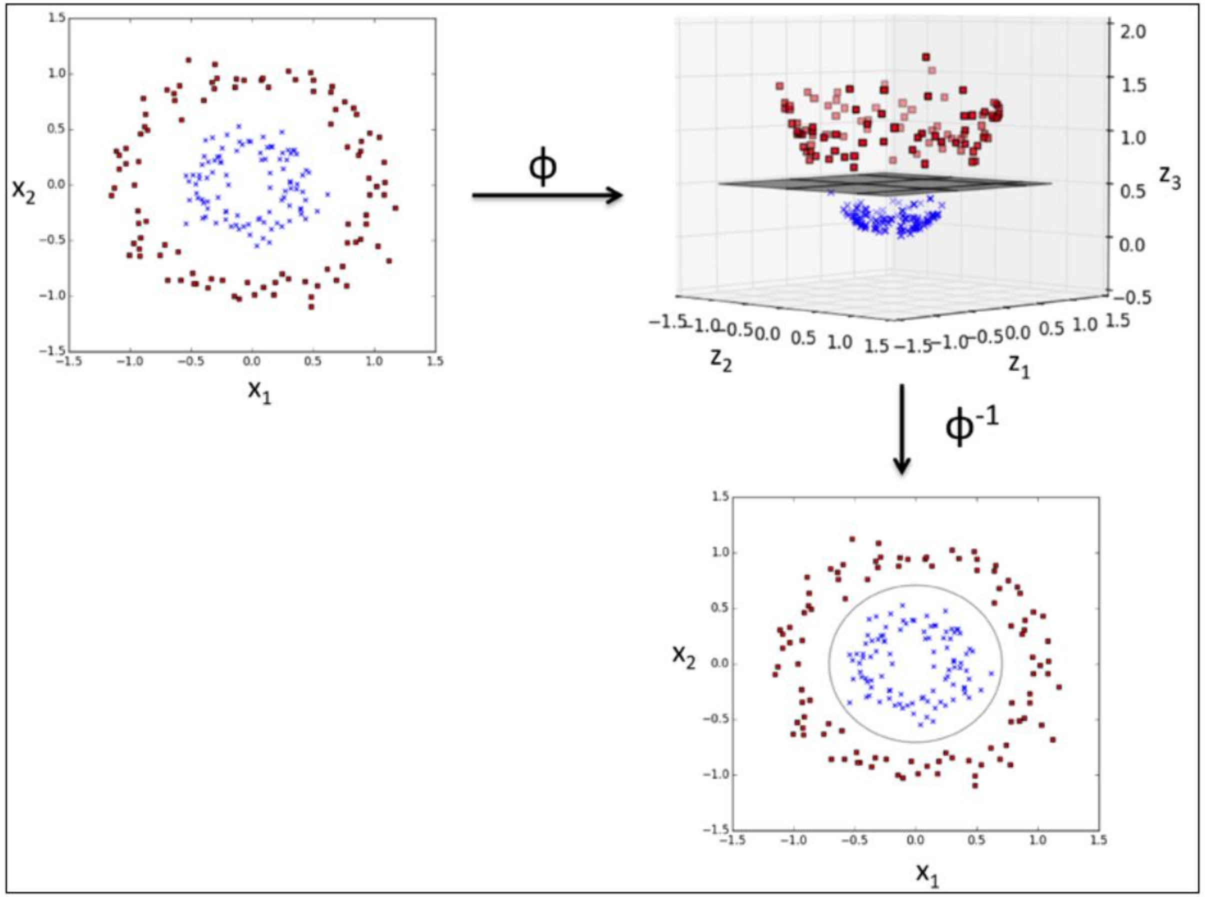


Kernel Methods

- The basic idea behind **kernel methods** to deal with such linearly inseparable data is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping function ϕ where it becomes linearly separable.

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

Kernel Methods



Kernel Trick

- A problem with this mapping approach is that the construction of the new features is computationally very expensive, especially if we are dealing with high-dimensional data.
- This is where the so-called kernel trick comes into play:

$$\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)} \text{ by } \phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right)$$

$$\mathcal{K}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right)$$

Kernel

- Gaussian Kernel:

$$\mathcal{K}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\frac{\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2}{2\sigma^2}\right)$$

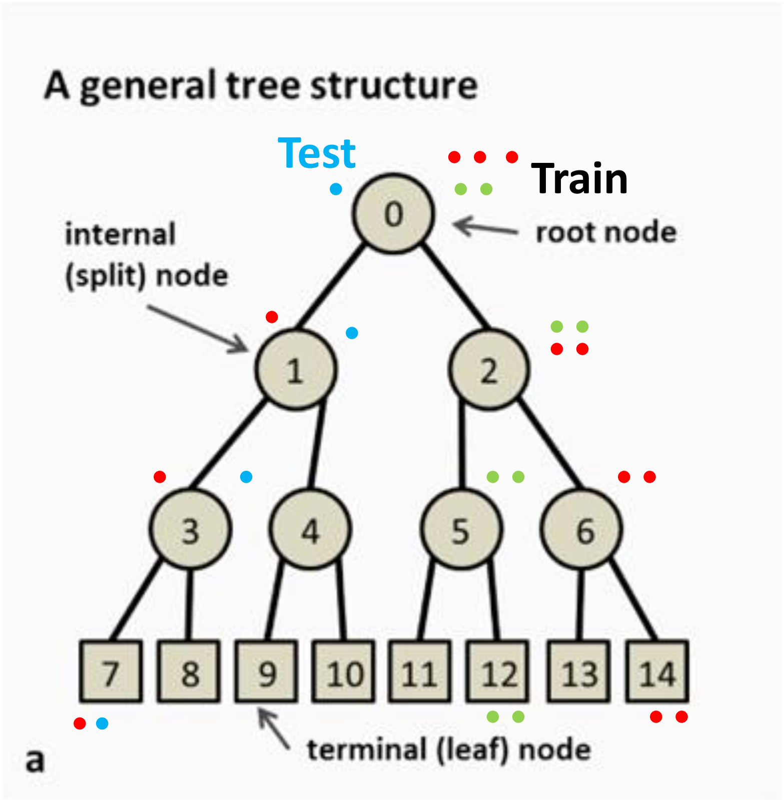
$$\mathcal{K}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\gamma\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right)$$

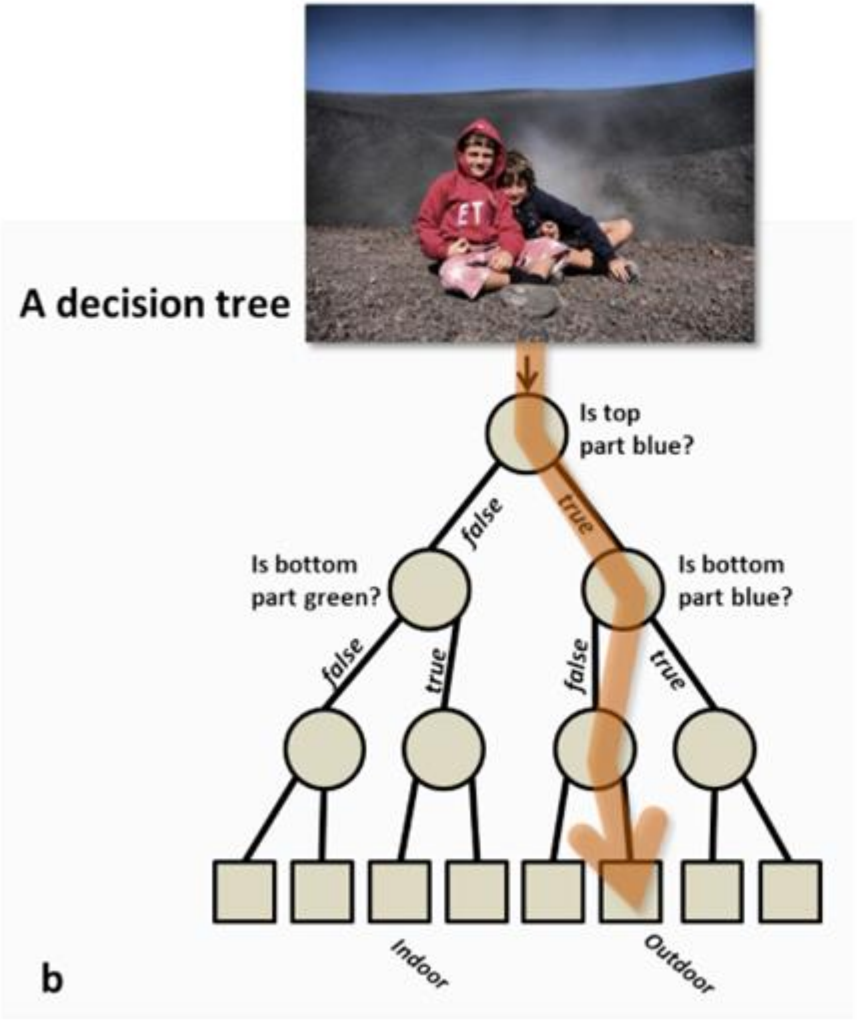
SVM using scikit-learn

- Nonlinear case...

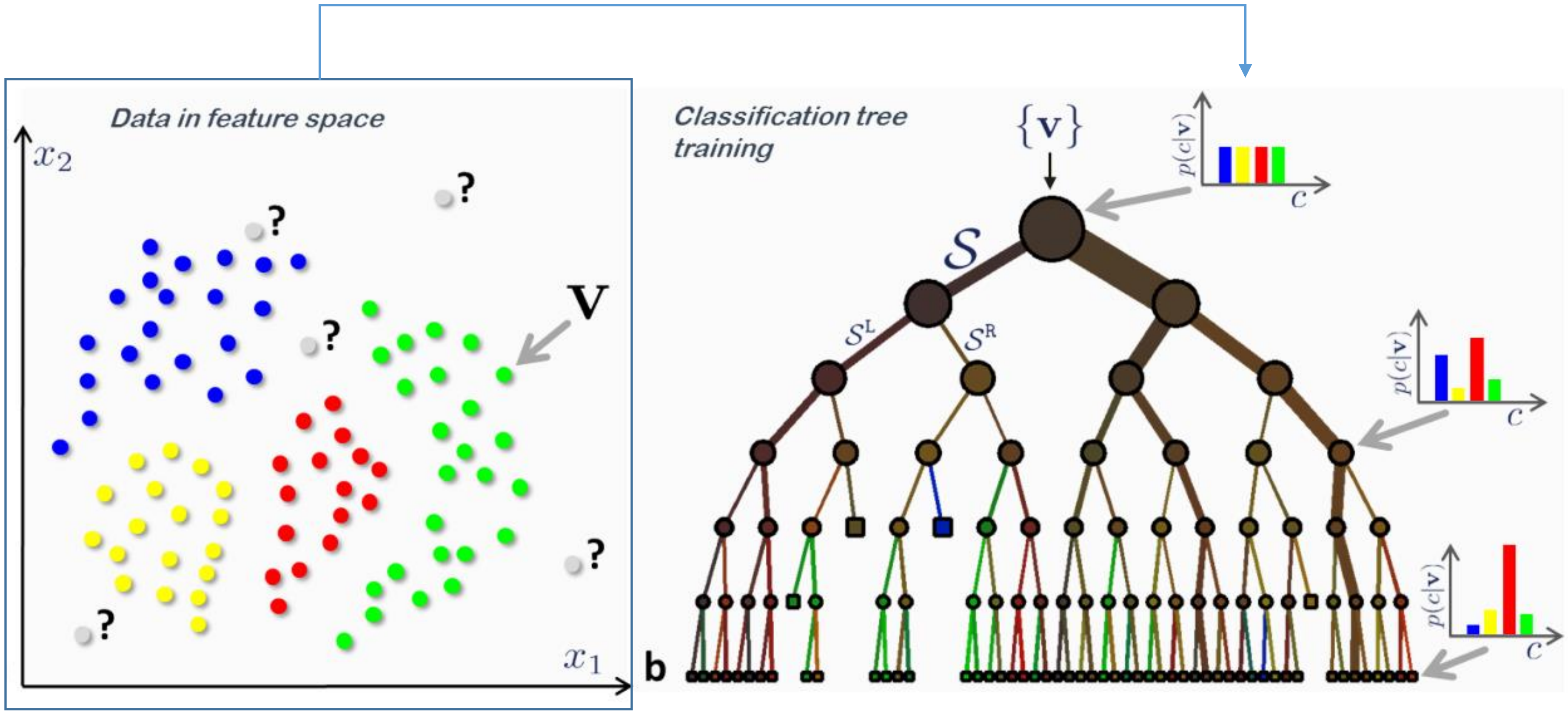
Decision Tree

- General overview:





Decision Trees

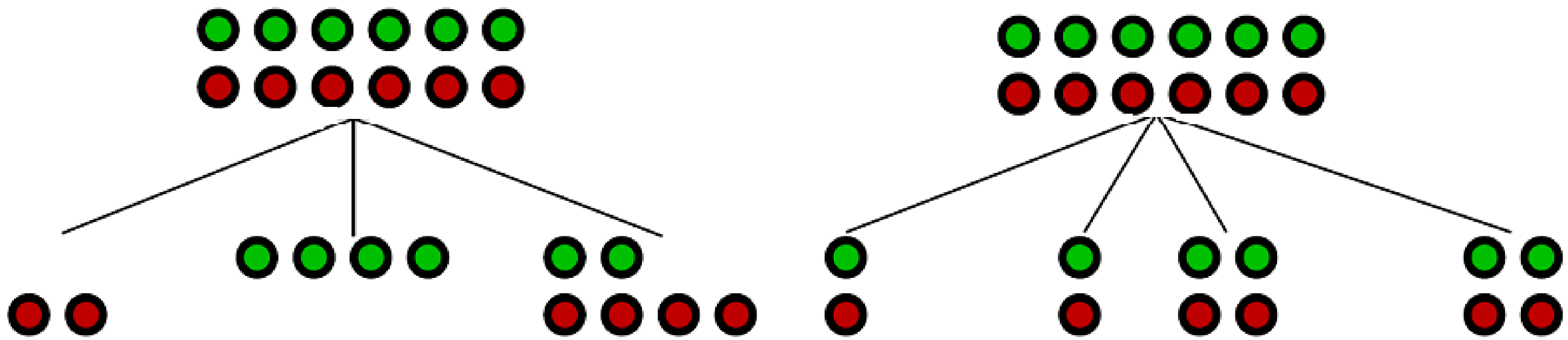


$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$

$$\mathcal{S}_j = \mathcal{S}_j^{\mathrm{L}} \cup \mathcal{S}_j^{\mathrm{R}}$$

Decision trees

- How to build a tree and the splits?
  - A good attribute separates the examples into subsets that, ideally, are all positive and all negative, for example:

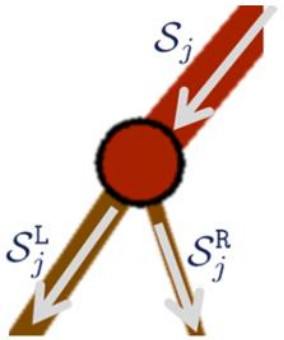


  - For this, the concept of Information Gain or entropy reduction can be used;

Decision Trees

- Training:

|S| - total number of examples

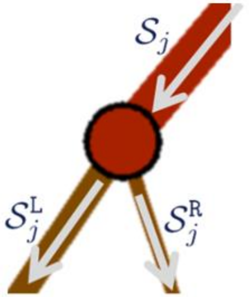


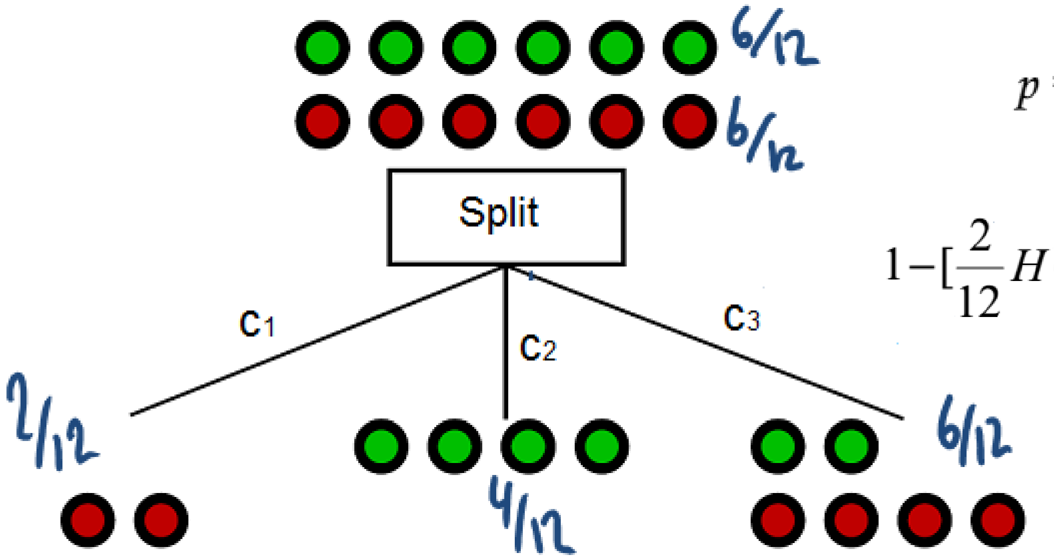
$$I_j = H(\mathcal{S}_j) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i) \longrightarrow$$ **Information Gain**

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log p(c) \longrightarrow$$ **Entropy**

$c \in \{c_k\}$ indexing the class

Ex.: S constains 10 exemples of class c_1
e 10 examples of class c_2:
H(S) = -(H(10/20) + H(10/20)) = -H(10/20, 10/20)
Log2 is normally used!





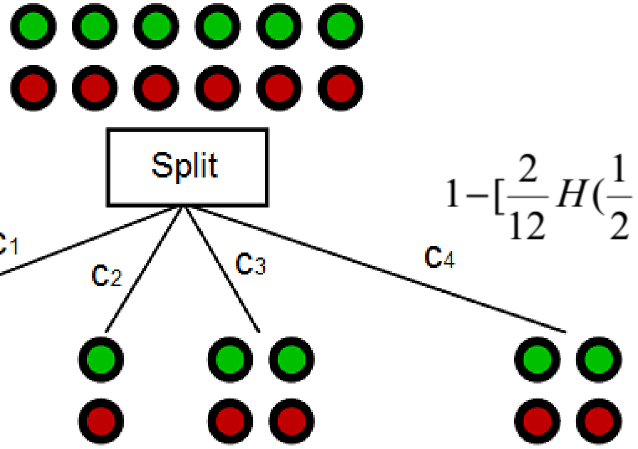
$p = n = 6,\ H(6/12,\ 6/12) = 1$ bit

$$1-\left[\frac{2}{12}H(0,1)+\frac{4}{12}H(1,0)+\frac{6}{12}H\left(\frac{2}{6},\frac{4}{6}\right)\right]=.541 \text{ bits}$$

r g g r

$$I_j = H(\mathcal{S}_j) - \sum_{i\in\{L,R\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i)$$

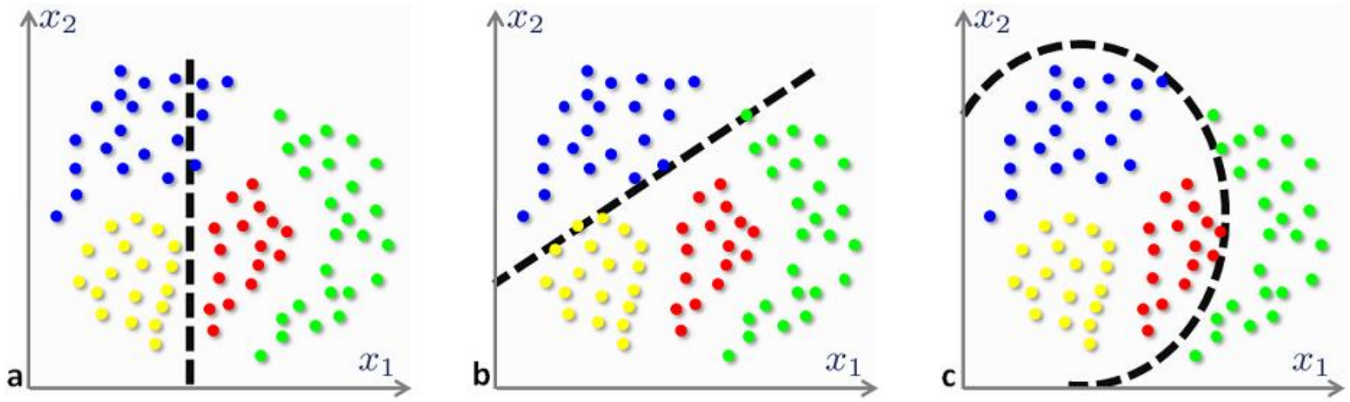
$$H(\mathcal{S}) = -\sum_{c\in\mathcal{C}} p(c)\log p(c)$$



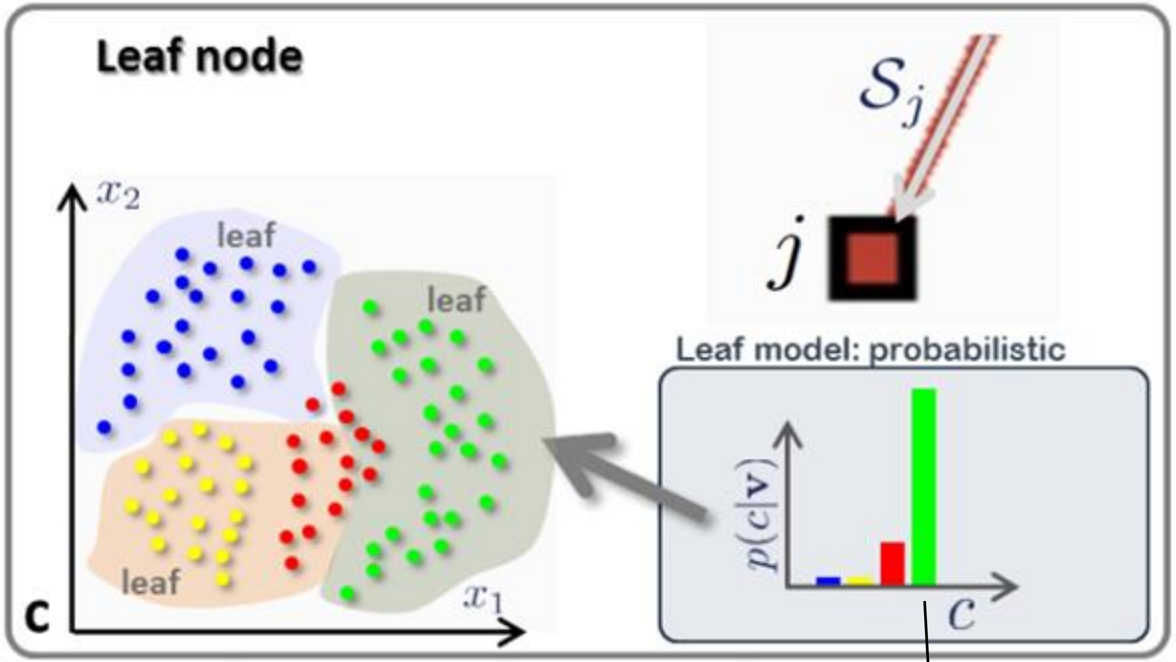
$$1-\left[\frac{2}{12}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{2}{12}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{4}{12}H\left(\frac{2}{4},\frac{2}{4}\right)+\frac{4}{12}H\left(\frac{2}{4},\frac{2}{4}\right)\right]=0 \text{ bits}$$

Decision Trees

- In practical cases (simple version):
  - Calculate the entropy of the full *dataset*
  - For each *feature*:
    - Split into intervals (linear *grid* or histogram grid)
    - Calculate the average information gain considering the intervals
  - Select the *feature* with the highest average information gain
  - Repeat the process (grow the tree) until the stop criterion is reached (maximum size, maximum probability, etc).
- Several other approaches, each with a different algorithm. For instance:



In the test phase



Majoritary class!

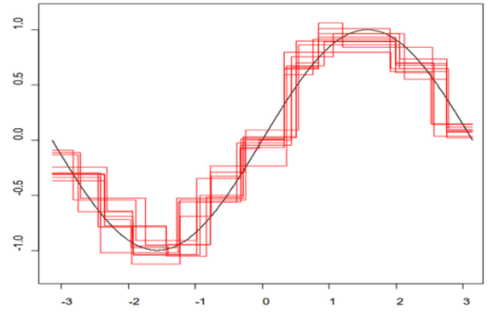
Decision Tree no scikit-learn

- IRIS example...

Random Forests

- Algorithm:
  - For b=1 to B (each tree):
    1. Remove a **Z*** subset of the training data at random.
    2. For each subset, generate a tree of predetermined size, repeating the following steps recursively (*random tree*):
       I. Randomly select a subset of m attributes, of the available p (only a part of the attributes are used).
       II. Determine the best split of the data based on the information gain.
       III. Divide the node into two child nodes.
- The result will be given by the combination of the various trees:

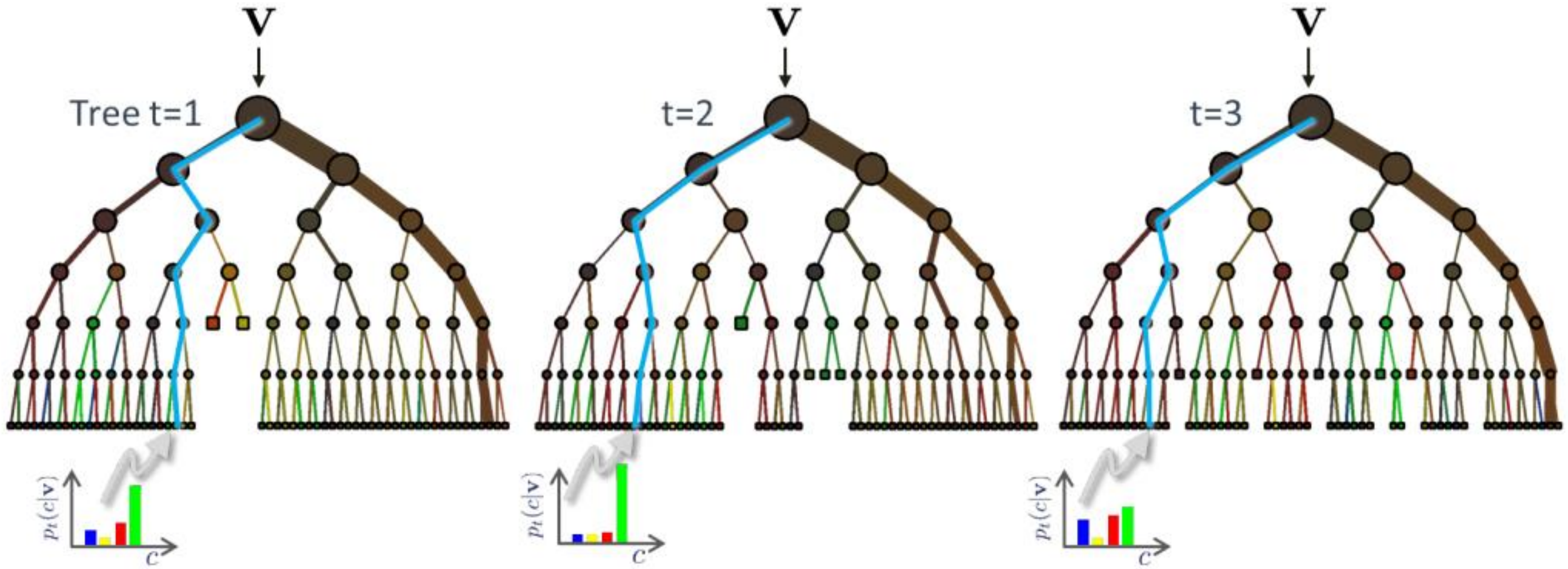
Why does it work?



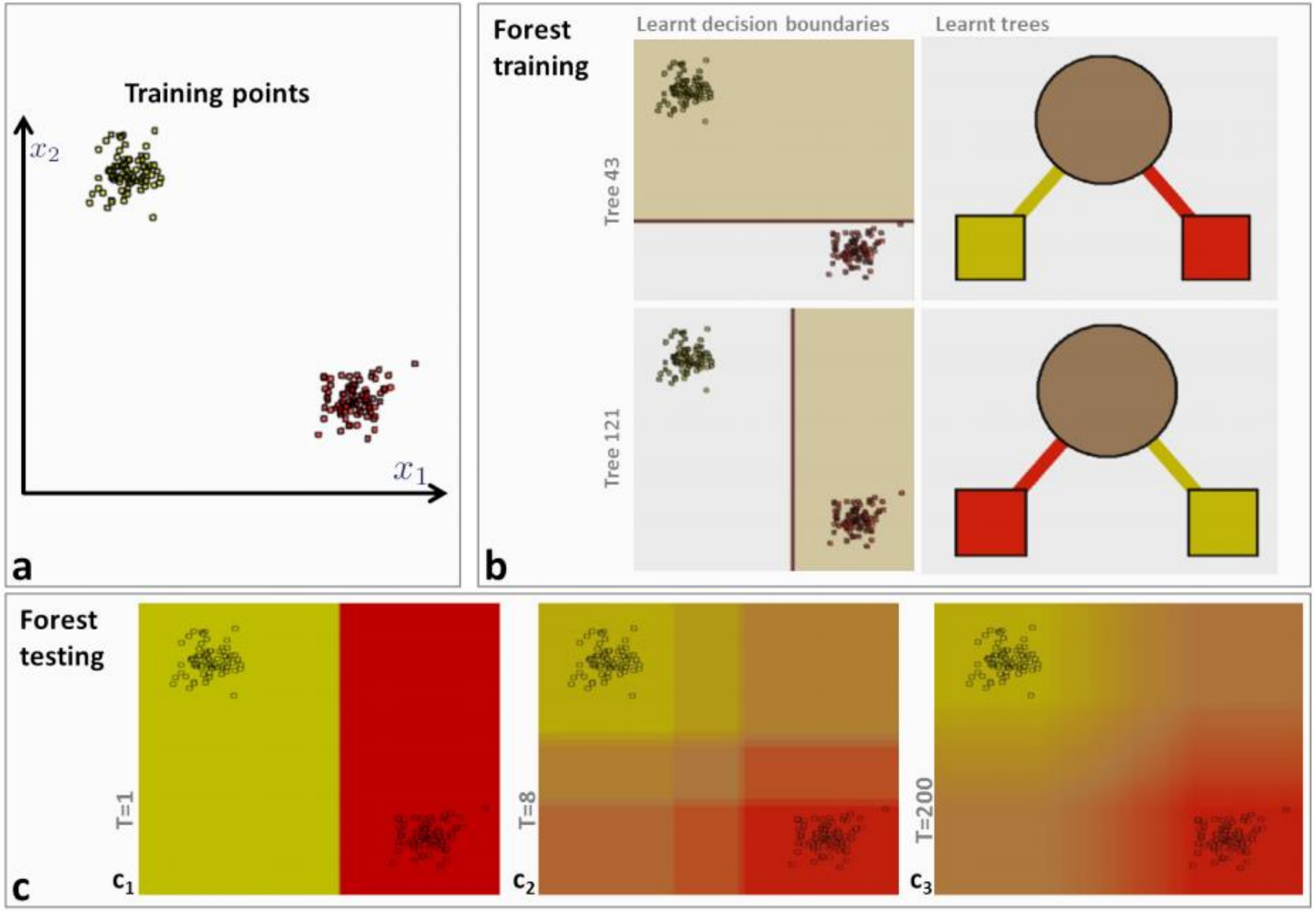
Random Forests

- **Training**: the trees are trained separately and individually (usually in a distributed way).

- **Test**: $$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$



Number of Trees (Impact)



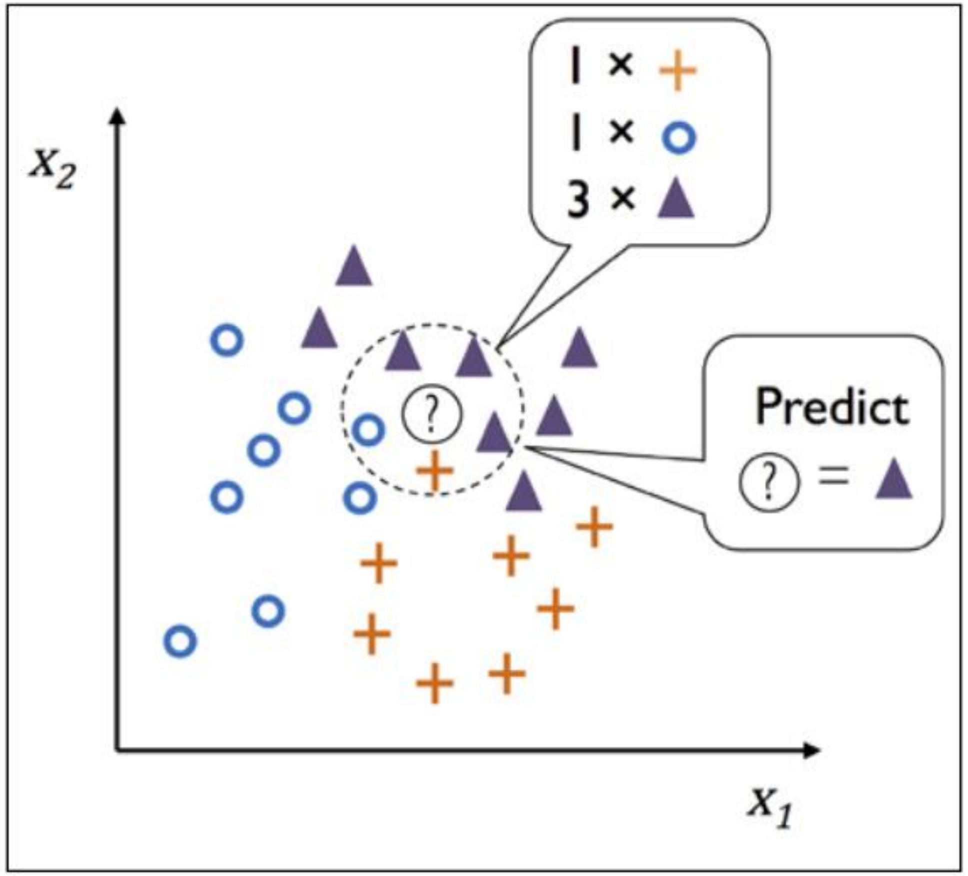
Smoother separation →

Random Forest with scikit-learn

- IRIS example...

K-Nearest Neighbors

1. Choose the number of k and a distance metric.
2. Find the k-nearest neighbors of the sample that we want to classify.
3. Assign the class label by majority vote.



kNN with scikit-learn

- IRIS example...