Classification with machine learning: part B

Dr Maria Anastasiadi
(m.anastasiadi@cranfield.ac.uk)

14th January 2025

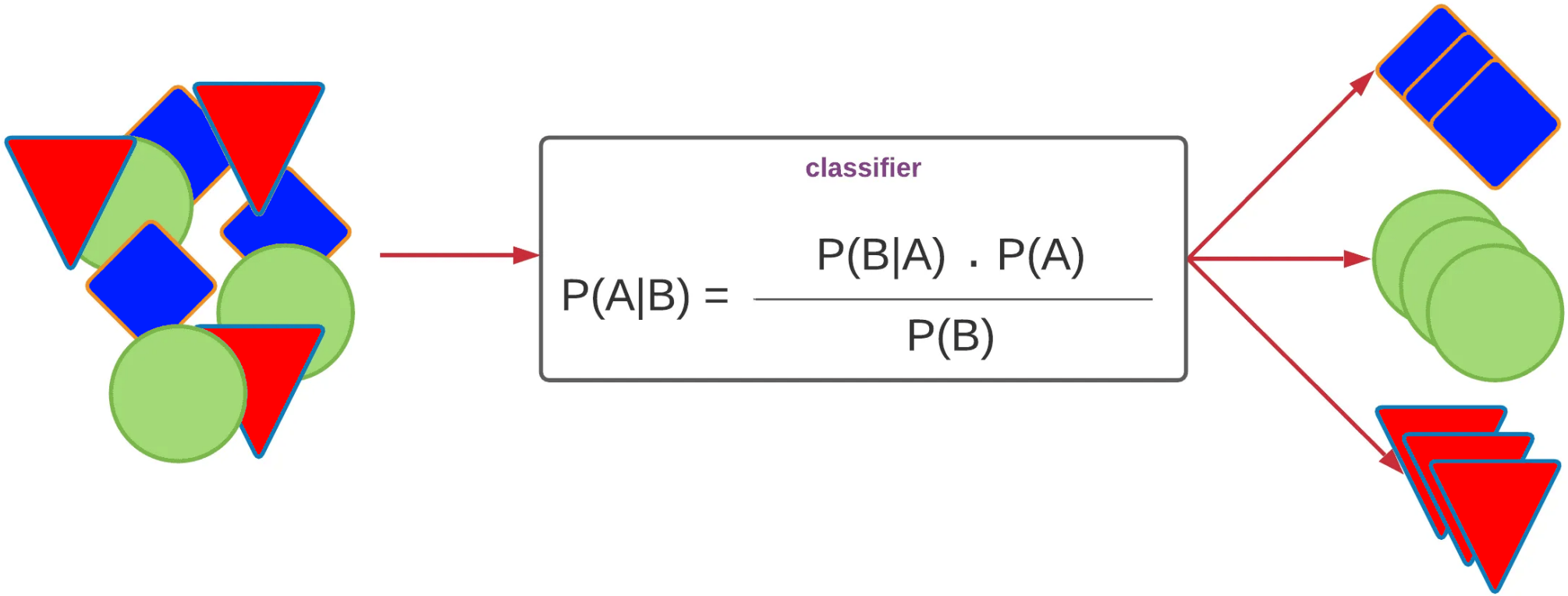
www.cranfield.ac.uk

Intended learning outcomes (ILOs)

- Be able to describe the main principles behind
  - Naïve Bayes
  - SVM
- Provide examples of applications for each algorithm.
- Become familiar with the concept of ML model tuning and optimisation

Probabilistic learning

Naive Bayes Classifier



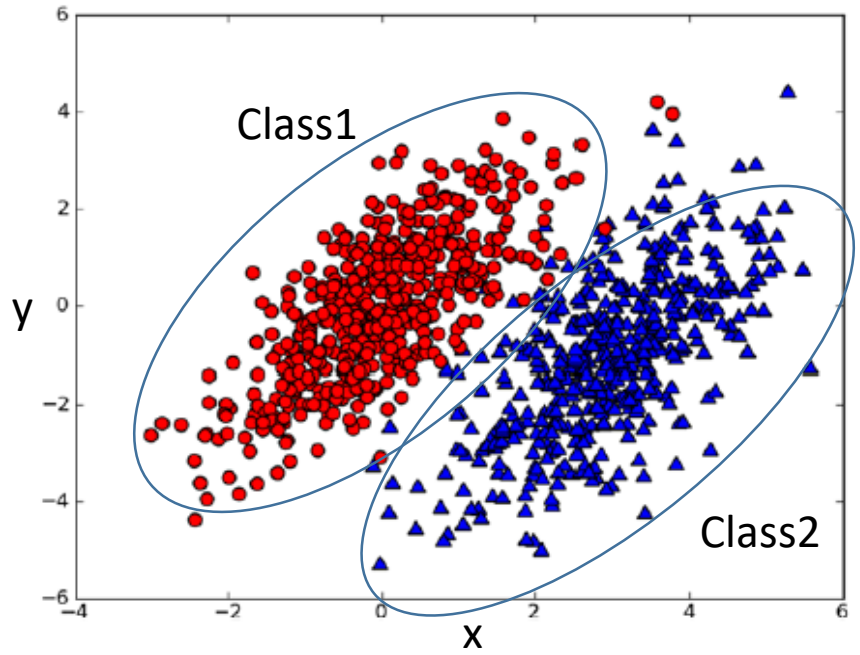
Classification Using Naive Bayes

Naïve Bayes in a nutshell:

1. Assume the data in a dataset belong to two classes Class1 and Class2.
2. Assume we have an equation for the probability of a sample belonging to Class 1 $p_1(x, y)$ and an equation for the probability of a sample belonging to Class 2 $p_2(x, y)$.
3. To classify a new measurement with features (x, y) we use the following rules:

   a) If $p_1(x, y) > p_2(x, y)$, then the class is 1.

   b) If $p_2(x, y) > p_1(x, y)$, then the class is 2.



Choose the decision with the highest probability!

Maximum A Posteriori (MAP)

Bayesian probability

joint probability

1. Independent events

The probability of events A and B occurring at the same time: $P(A \cap B) = P(A) * P(B)$

2. Dependent events

$P(A \cap B) = P(A) * P(B|A) = P(B) * P(A|B)$

Conditional probability

Bayes Theorem:

likelihood

prior probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

posterior probability

marginal probability

Classifying with conditional probabilities

Using the Bayes theorem, we can calculate the p_1 and p_2 probabilities from the previous example and use it to classify an unseen example.

Given a point identified as x,y.
What is the probability it belongs to c_1?
What is the probability it belongs to c_2?

$$p(C_1|x \cap y) = \frac{p(x \cap y|C_1)p(C_1)}{p(x \cap y)}$$

$$p(C_2|x \cap y) = \frac{p(x \cap y|C_2)p(C_2)}{p(x \cap y)}$$

As the number of features increases , the formula can get very complicated and computationally demanding.

For example:

$$p(C_1|x \cap y \cap z \cap k) = \frac{p(x \cap y \cap z \cap k|C_1)p(C_1)}{p(x \cap y \cap z \cap k)}$$

The Naive Bayes algorithm

Naive Bayes assumes **class-conditional independence**, which means that events are independent so long as they are conditioned on the same class level. Also, the denominator can be treated as a constant value. Thus:

$$p(C_1|x \cap y \cap z \cap k) \propto p(x|C_1)\, p(y|C_1)\, p(z|C_1)\, p(k|C_1)$$

The general formula for the Naive Bayes classification algorithm can be summarized by the following equation:

$$P(C_L|F_1, \ldots\ldots, F_n) = \frac{1}{Z} p(C_L) \prod_{i=1}^{n} p(F_i|C_L)$$

Where:

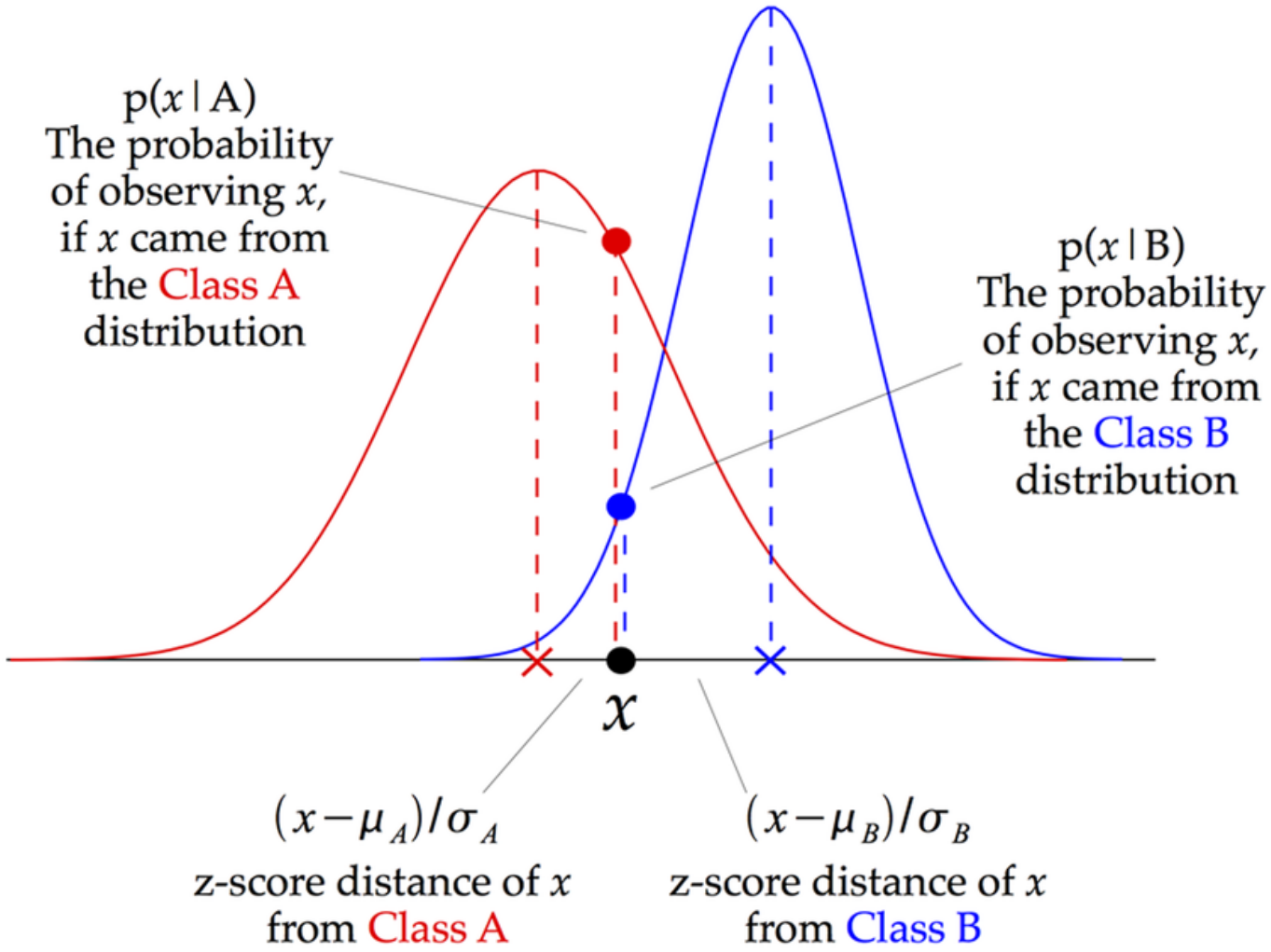
C_L = the level L of class C,

$P(C_L)$ = the prior probability of the class level

F_i = feature i,

1/Z = a scaling factor

Gaussian Naïve Bayes



The Naive Bayes algorithm

The Naive Bayes algorithm makes some "naive“ assumptions about the data. In particular, Naive Bayes assumes that all of the features in the dataset are equally important and independent, which is rarely true in most real-world applications.

However, in most cases when these assumptions are violated, Naive Bayes still performs fairly well.

There are different types of Naive Bayes Algorithms like GaussianNB, MultinomialNB, BernoulliNB.

Strengths	Weaknesses
Simple, fast, effective	Relies on an often faulty assumption of equally important and independent features
Perform well with noisy and/or missing data	Not ideal for datasets with mainly numeric features
Requires relatively few examples for training	Estimated probabilities are less reliable than the predicted classes
Easy to obtain the estimated probability for a prediction	

Naïve Bayes

How it works

1. Collect data
2. Prepare the data
3. Train a model on the data: Calculate the conditional probabilities of the independent features.
4. Evaluate model performance: Test set. Calculate the error rate.
5. Improve model performance.

Example: Classifying text messages (spam / non-spam) – A classical application of NB.

Naïve Bayes: Applications

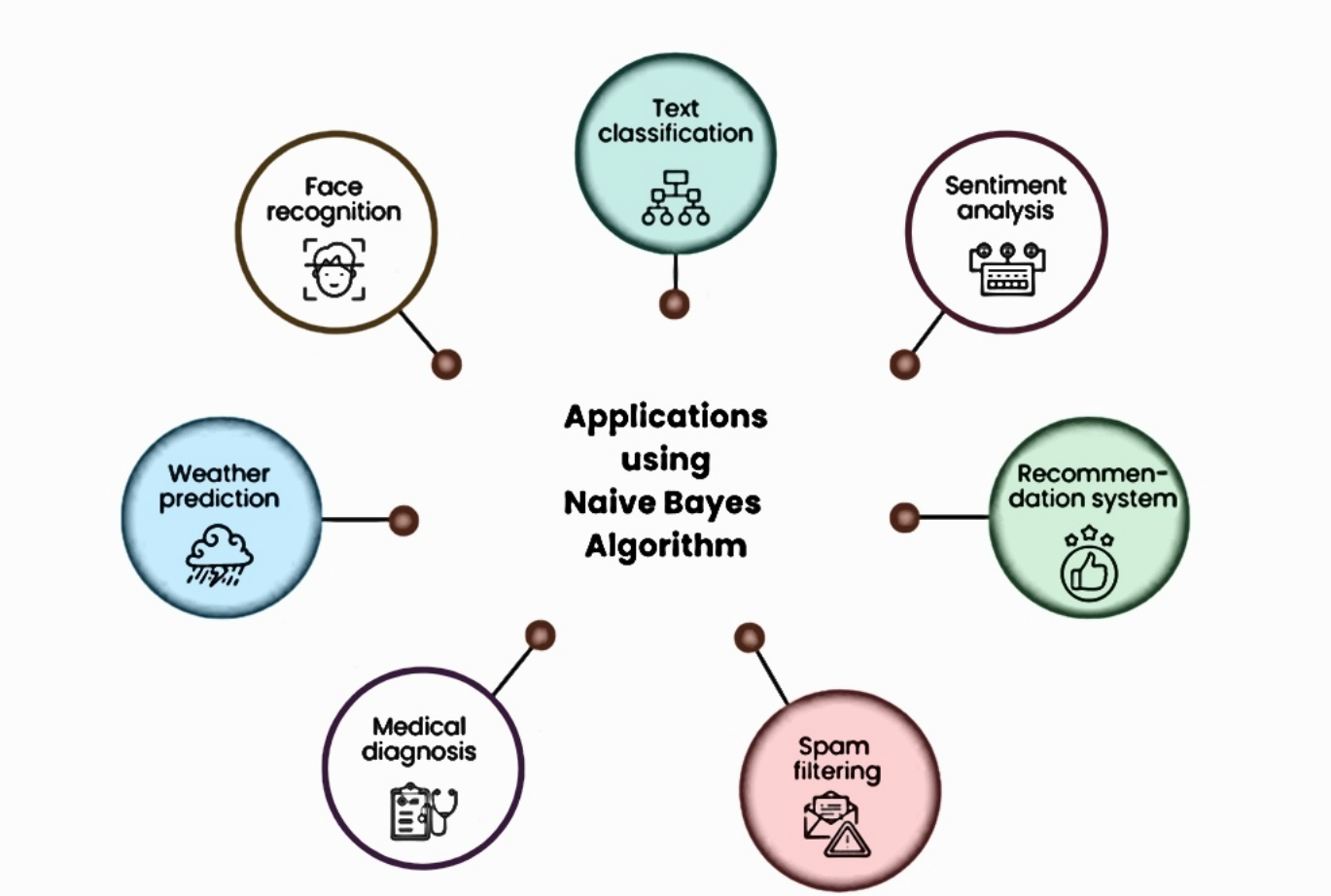
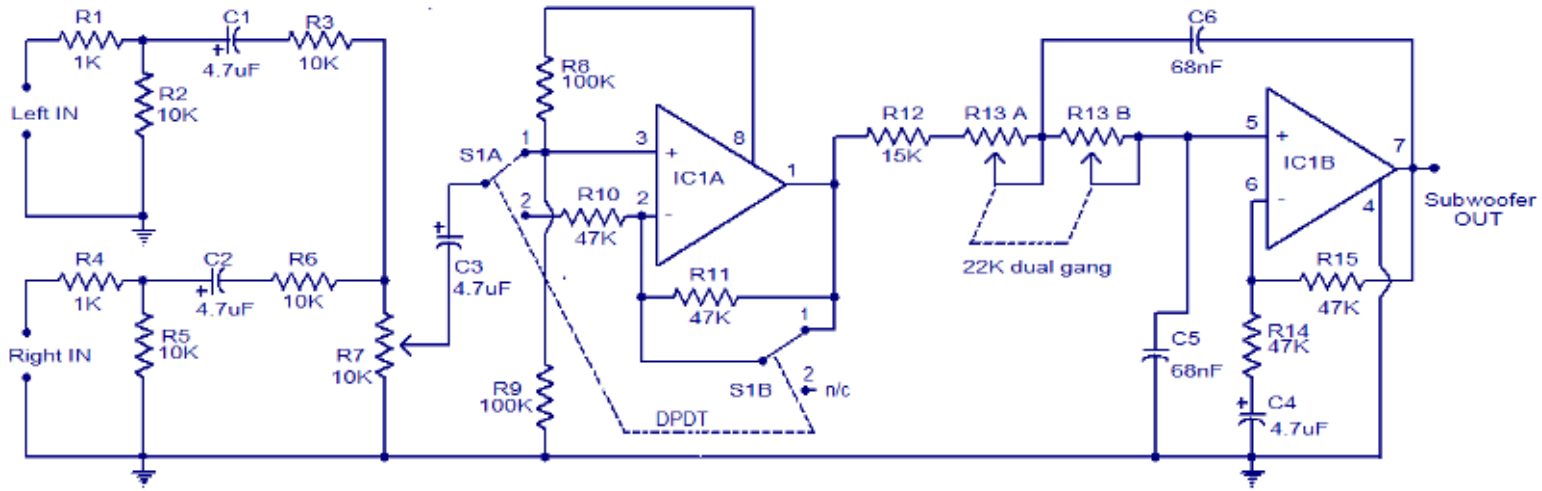


Image source: turing.com

ML: Black Box methods

The black box metaphor

Imagine a box with a set of inputs (switches/buttons) and a set of outputs (lights)



Can you explain ***how*** the box works?
Can you explain ***why*** each component has been placed there?

complexity arises from the interaction of many simple components

ML: Black Box methods

In the case of machine learning, the ‘black box’ metaphor is used for some methods due to the complex mathematics allowing them to function.

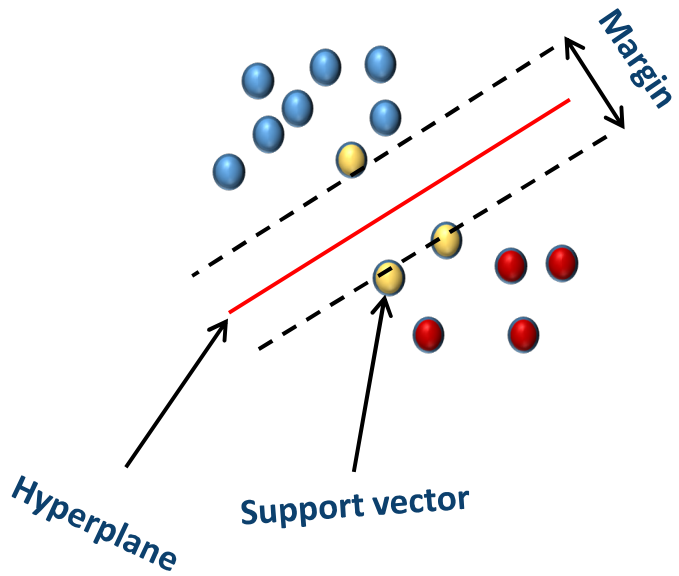
For example:

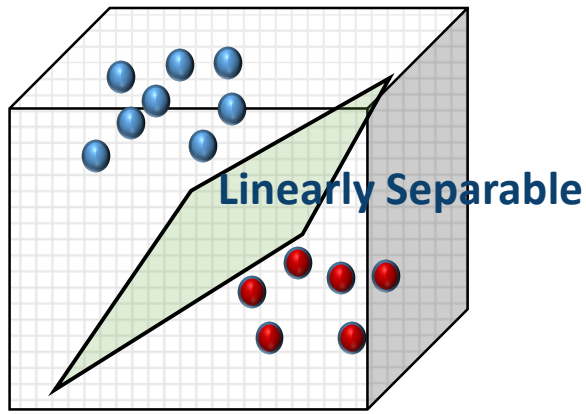
1) **Neural networks** mimic the structure of animal brains to model arbitrary functions.
2) **Support vector machines** use multidimensional surfaces to define the relationship between features and outcomes.

ML: Support Vector Machines (SVM)

SVM aims to create a **linear decision surface** (**hyperplane**) that can separate classes and has the largest distance (margin) between border-line samples ("**support vectors**").

SVMs has traditionally been applied for binary classification.

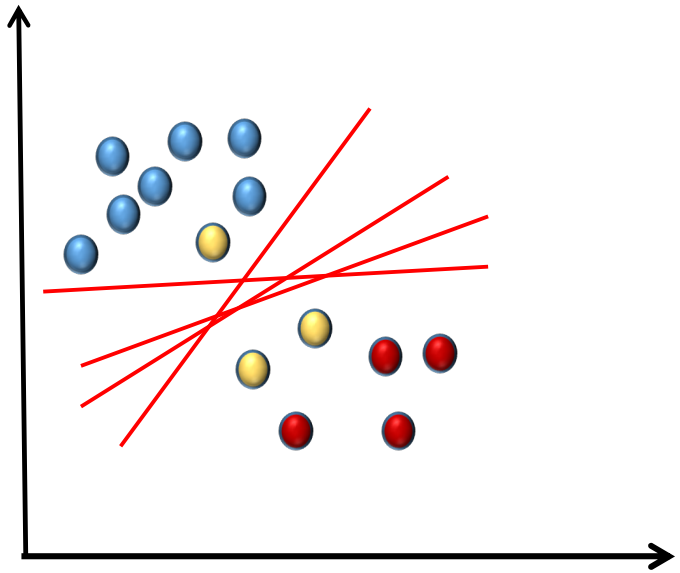


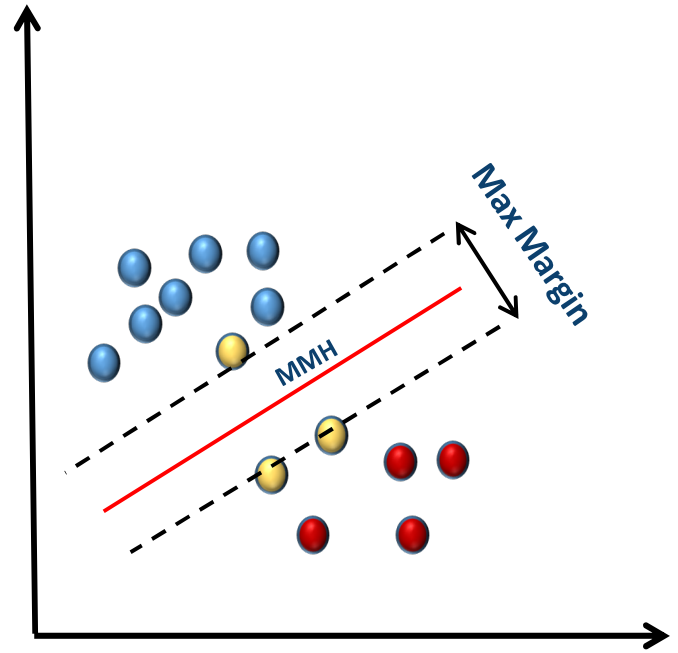


Feature space

ML: Support Vector Machines (SVM)

- An infinite number of hyperplanes may exist.
- How does the algorithm choose?
- Search for the **Maximum Margin Hyperplane (MMH)** that creates the greatest separation between the two classes.
- Important! Each class must have at least one support vector.
- Using the support vectors alone, it is possible to define the MMH.





ML: Support Vector Machines (SVM)

Methods to find MMH

Case1: Linearly separable data

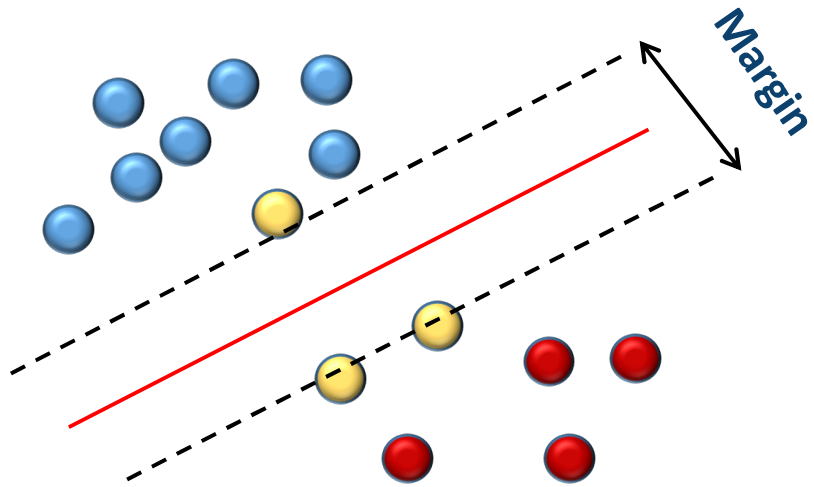
Strategy:

a) Get your dataset

b) Find a hyperplane which separates the data linearly.

c) Select two hyperplanes which separate the data with no points between them.

d) Maximise their distance.



ML: Support Vector Machines (SVM)

Step 1: Get your dataset.

Most of the time your data will be composed of n vectors x_i containing the variable values for each samples.

Each x_i will also be associated with a value y_i indicating if the element belongs to the class (+1) or not (-1).

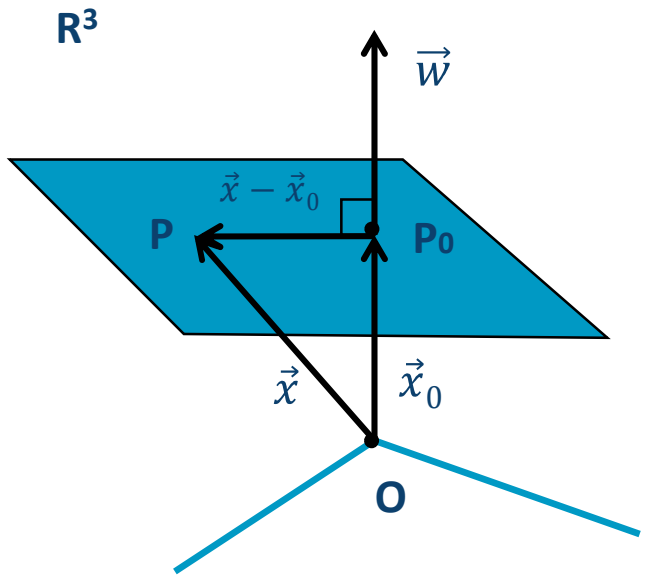
Remember that y_i can have two possible values -1 or +1.

The initial dataset is defined as :

$$\boldsymbol{D = \{(x_i, y_i) |\, x_i \in \mathbb{R}^p, y_i \epsilon \{-1, 1\}\},\ i = 1, \ldots n}$$

ML: Support Vector Machines (SVM)

Step 2. Find the equation for the hyperplane linearly separating the data.



An equation of a hyperplane (H_0) is defined by a point (P_0) and a vector ($\vec{w}$) perpendicular to the plane.

Let P be an arbitrary point on a hyperplane.

Then $\vec{x}_0 = \overrightarrow{OP_0}$ and $\vec{x} = \overrightarrow{OP}$

A condition for P to be on the plane is that the vector $\vec{x} - \vec{x}_0$ is perpendicular to $\vec{w}$:

$\vec{w} \cdot (\vec{x} - \vec{x}_0) = 0$ or

$\vec{w} \cdot \vec{x} - \vec{w} \cdot \vec{x}_0 = 0$ Let us define $-\vec{w} \cdot \vec{x}_0 = b$

$\vec{w} \cdot \vec{x} + b = 0$

$\vec{w}$ is a vector of n weights
And b is a number known as bias

Holds for R^n when n>3

ML: Support Vector Machines (SVM)

Step 3. Select two hyperplanes which separate the data with no points between them.

We have already found a hyperplane H0 separating the dataset and satisfying $\vec{w} \cdot \vec{x} + b = 0$.

Now, lets define two other hyperplanes H1 and H2 which also separate the data and have the following equations:

$$\vec{w} \cdot \vec{x} + b = +1$$

$$\vec{w} \cdot \vec{x} + b = -1$$

The selected hyperplanes must meet the following constraints:
For each vector xi either:

$\vec{w} \cdot \vec{x} + b \geq +1$, for xi belonging to class +1

or

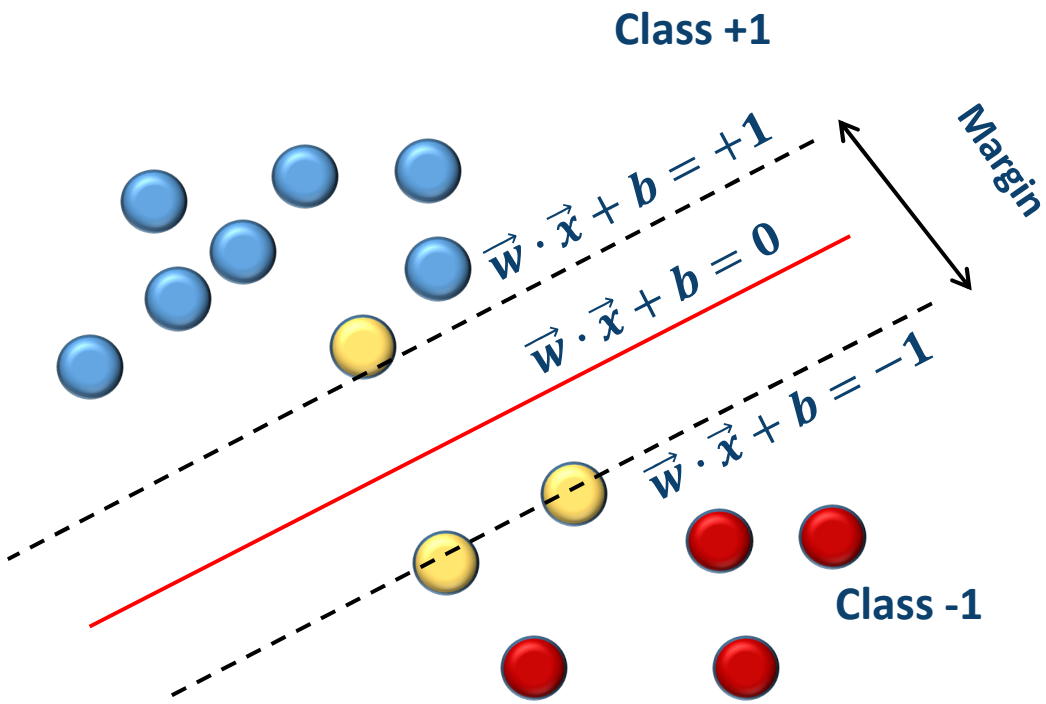
$\vec{w} \cdot \vec{x} + b \leq -1$, for xi belonging to class -1

ML: Support Vector Machines (SVM)

If we multiply both sides of the previous constraints with yi (+1 for class 1 and -1 for class -1) they can be combined in one equivalent constraint:

$$y_i(\vec{w} \cdot \vec{x} + b) \geq 1$$

This constraint also makes sure there will be no points between the two hyperplanes.



Support Vector Machines (SVM)

Step 4. Maximise the distance between H1 and H2.

The distance between these two planes is: D = $\frac{2}{\|\vec{w}\|}$,

where $\|\vec{w}\|$ is the **Euclidean norm** (the distance from the origin to vector w).

Therefore, to maximise distance we need to minimise $\|\vec{w}\|$.

The task is typically re-expressed as a set of constraints, as follows:

$$min\frac{1}{2}\|\vec{w}\|^2$$

Subject to constraint: $y_i(\vec{w}\cdot\vec{x_i}+b)\geq 1, \mathbf{for\ any\ i} = (1, \ldots . n)$

ML: Support Vector Machines (SVM)

Case 2: Non-linearly separable data – Soft Margins

If our dataset isn't linearly separable we can apply **soft margin SVM:**

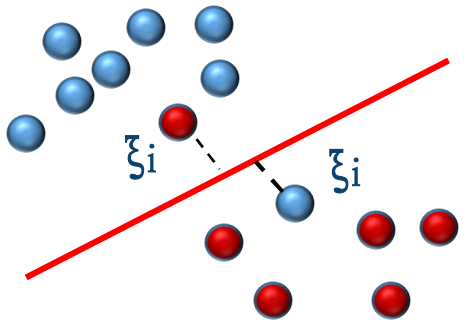
we allow some samples to stray over the line into the margin and be misclassified.

This is done using the **slack variable (ξ)**. A **cost value (C)** is applied to all points that violate the constraints, and rather than finding the maximum margin, the algorithm attempts to **minimise the total cost**.

$$min\,\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

Subject to constraint:

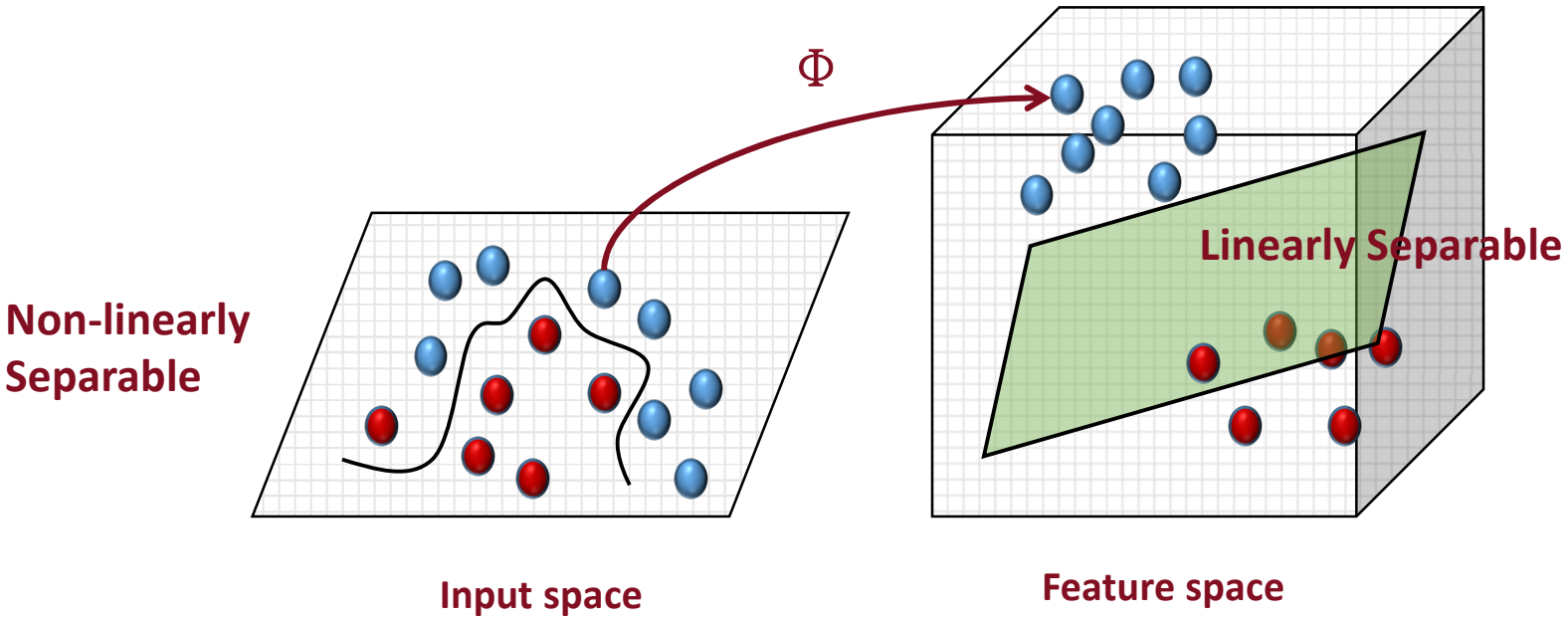
$$y_i(\vec{w}\cdot\vec{x_i}+b) \geq 1-\xi_i, \text{ for any i} = (1,\ldots.n), \xi_i \geq 0$$



ML: Support Vector Machines (SVM)

Case 2: Non-linearly separable data - Kernels

When separation is non-linear, the data is plotted into a higher dimensional feature space (“hypercube”) using a kernel function (the kernel trick).



ML: Support Vector Machines (SVM)

Non-linearly separable data

Kernel functions

The kernel “trick” is used to apply some transformation to the feature vectors xi and xj and combines them using the dot product, which takes two vectors and returns a single number.

General kernel function

$$K\left(\vec{x}_i, \vec{x}_j\right) = \varphi\left(\overrightarrow{x_i}\right) \cdot \varphi\left(\overrightarrow{x_j}\right)$$

ML: Support Vector Machines (SVM)

Non-linearly separable data

Kernel functions

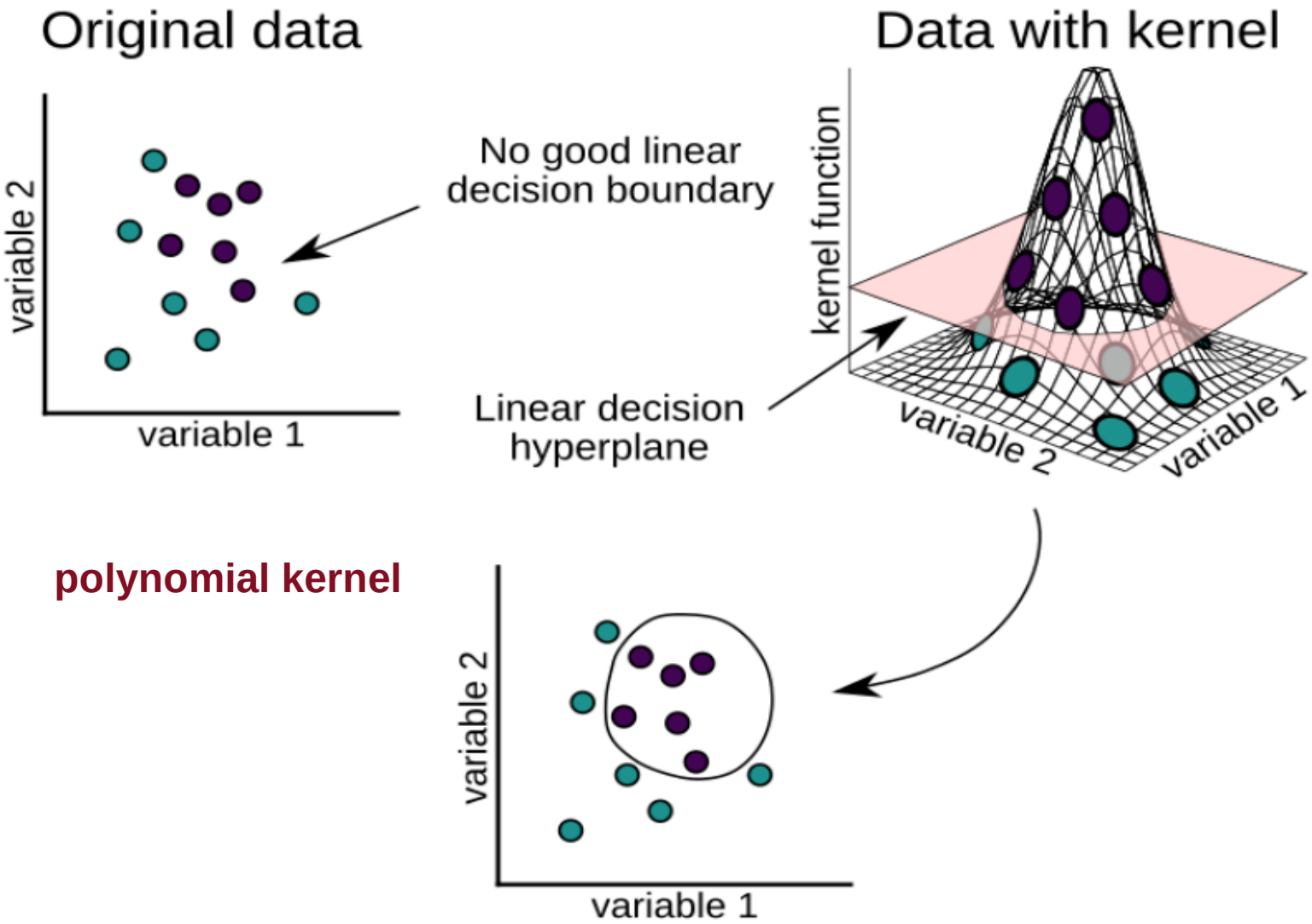
1. The linear kernel. Equivalent to no kernel.

$$K(\vec{x}_i, \vec{x}_j) = \vec{x_i} \cdot \vec{x_j}$$

2. The polynomial kernel of degree *d* . It adds a simple nonlinear transformation of the data.

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x_i} \cdot \vec{x_j} + 1)^{\mathrm{d}}$$

ML: Support Vector Machines (SVM)



Schematic: Hefin Rhys (2019)

ML: Support Vector Machines (SVM)

Non-linearly separable data

Kernel functions

3. The Sigmoid kernel. Results in a SVM model somewhat analogous to a neural network using a sigmoid activation function.

The Greek letters **kappa** and **delta** are used as kernel parameters:

$$K(\vec{x}_i, \vec{x}_j) = \tanh(\kappa \vec{x}_i \cdot \vec{x}_j - \delta)$$

ML: Support Vector Machines (SVM)

Non-linearly separable data

Kernel functions
4. The Gaussian Radial Basis Function (RBF) kernel.

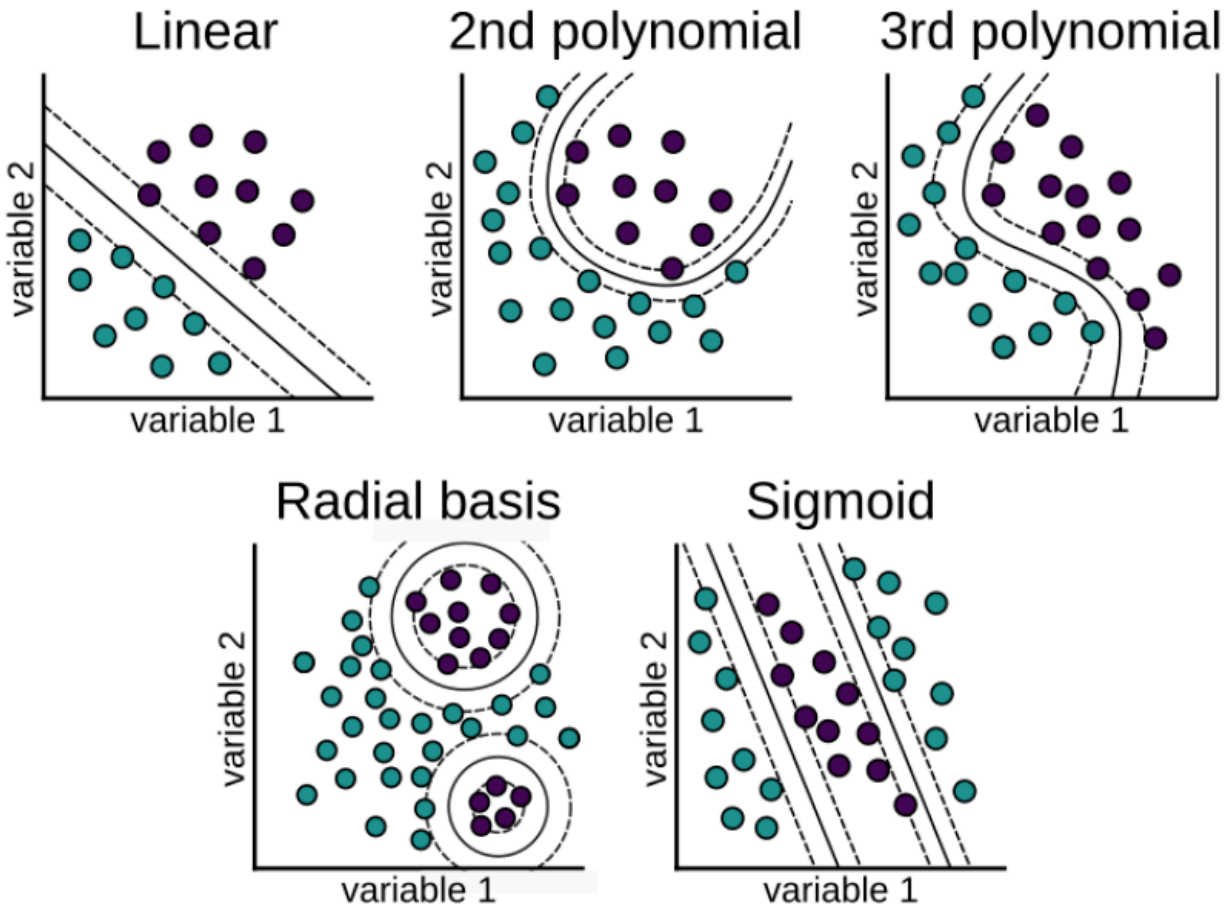
$$K\left(\vec{x}_i, \vec{x}_j\right) = e^{\frac{-\left\|\vec{x_i}-\vec{x_j}\right\|^2}{2\sigma^2}}$$

- Requires the optimisation of two parameters: **cost** and **gamma**.
- Gamma tends to be the reciprocal of the number of columns in the dataset, or can be varied (e.g. between 2^{-4} and 2).

ML: Support Vector Machines (SVM)

Kernel functions

For each Kernel type, the solid line indicates the decision boundary projected back onto the original feature space.



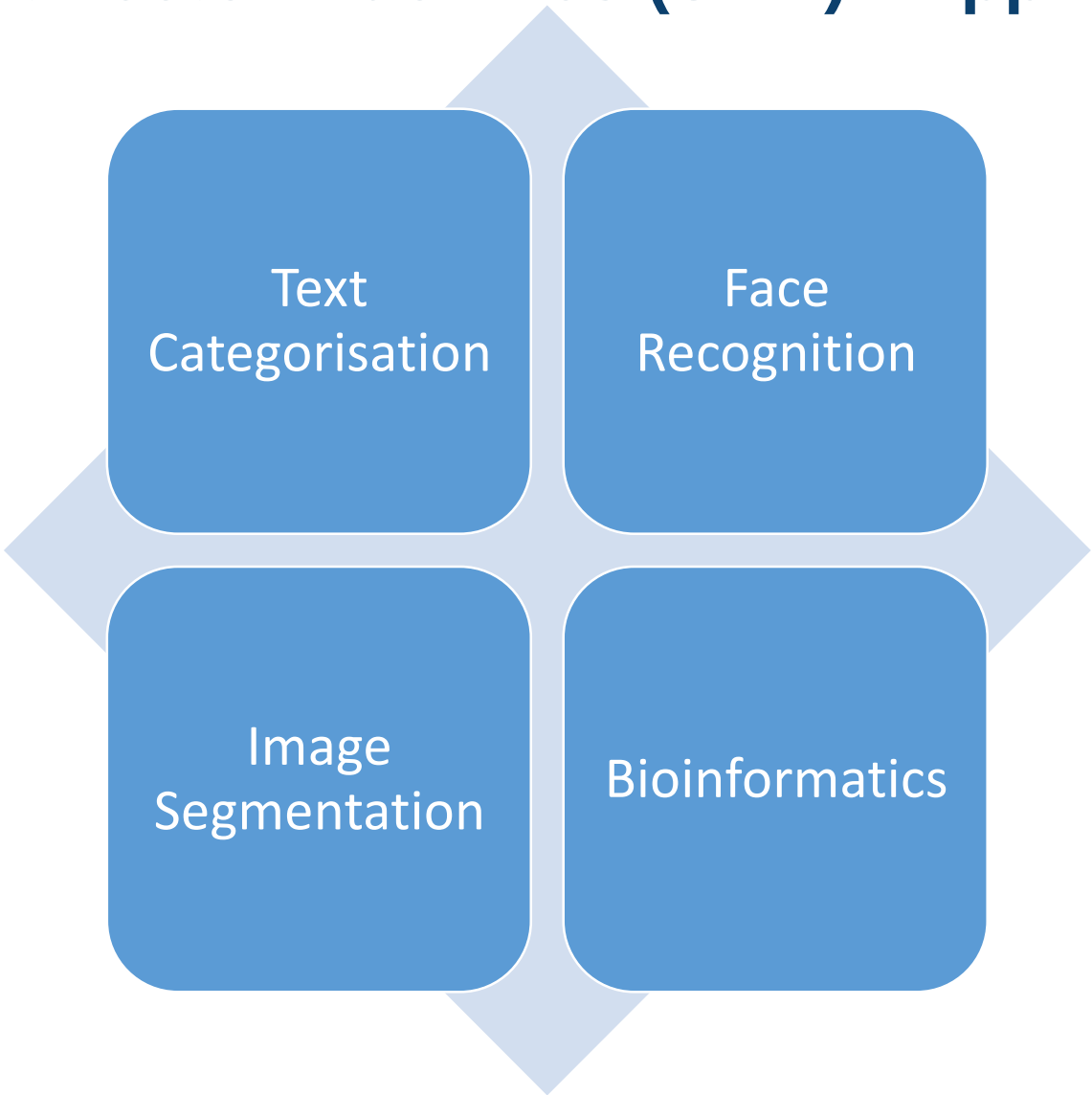
Schematic: Hefin Rhys (2019)

ML: Support Vector Machines (SVM)

Kernels pros & cons

Strengths	Weaknesses
Dual use (classification & regression)	Can take time to train for large datasets
Not very prone to overfitting	Various combinations of kernels and model parameters need to be tested
Can cope well with noisy data	Results very difficult to interpret
Widely applied compared to NN	

ML: Support Vector Machines (SVM) : Applications



ML: Model Optimisation

Model Hyperparameters

Hyperparameters are all the parameters in a ML model that the user defines their values or choose their configuration before training begins.

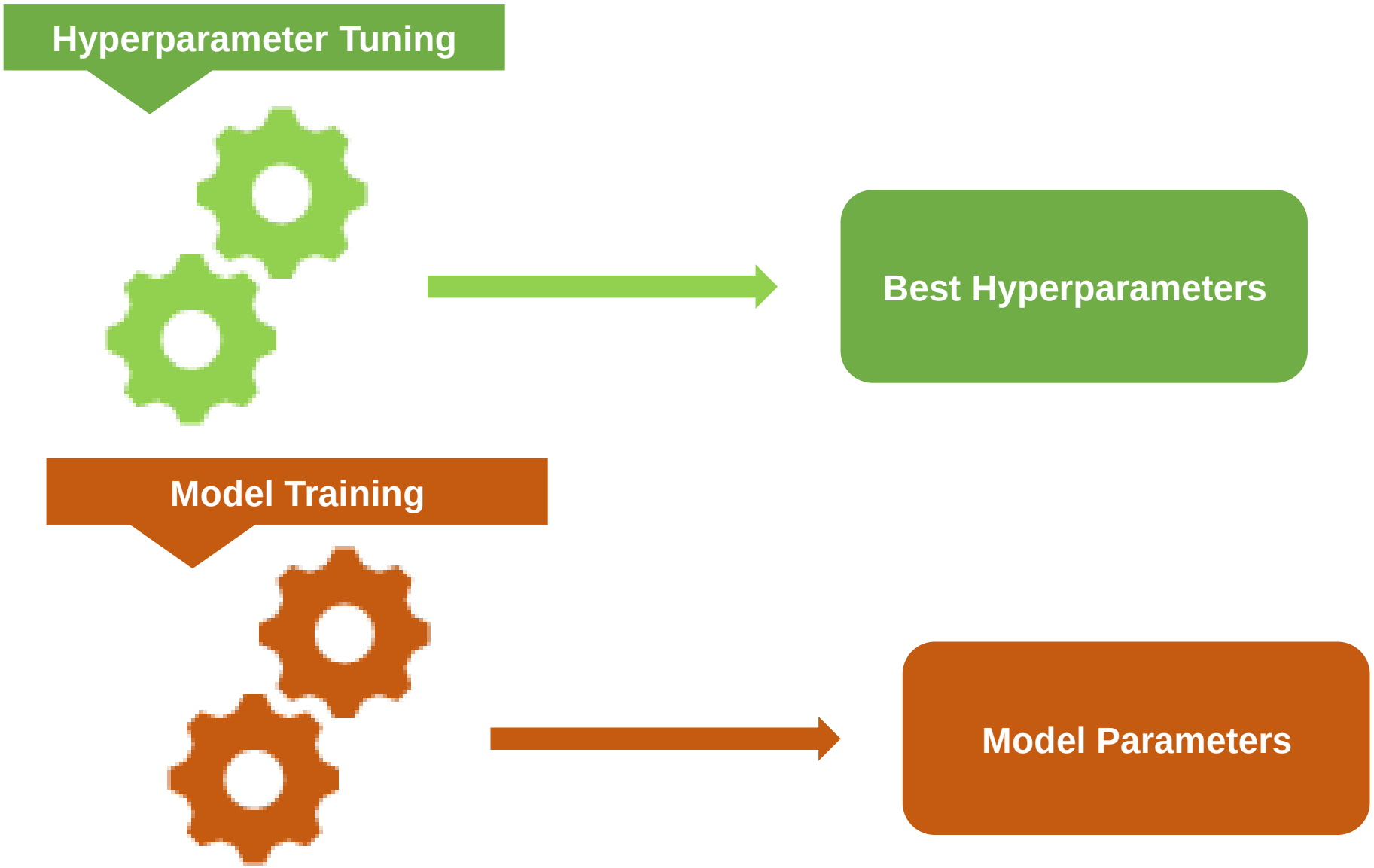
E.g. The number of branches in a decision tree, the number of nodes and layers in a NN, the train-test split ratio.

Model Parameters

Parameters are internal to the model. They are either learned or estimated purely from the data during training as the algorithm used tries to learn the mapping between the input features and the labels or targets.

E.g. The coefficients (or weights) of linear and logistic regression models, weights and biases of a NN.

ML: Model Optimisation



ML: Hyperparameter Tuning

Automated hyperparameter tuning

Randomly picking a model's parameter values is a tedious task and not strictly scientific.

- Rather than choosing arbitrary parameter values for each ML model it is better to conduct an automated **grid search** to find the best combination.
- The `caret` package in R provides this option through the **`train()`** function for over 200 models both for classification and regression tasks.

ML: Hyperparameter Tuning

Automated hyperparameter tuning requires the user to consider three questions:

1. **What type of ML model should be trained on the data?**

2. **Which model hyperparameters can be adjusted, and how extensively should they be tuned to find the optimal settings?**

3. **What criteria should be used to evaluate the models to find the best candidate?**

ML: Hyperparameter Tuning

Model	Method name	Hyperparameters
k-NN	`knn`	k
Naïve Bayes	`nb`	fL, usekernel, adjust
Decision Trees	`C5.0`	model, trials, winnow
OneR Rule Learner	`OneR`	None
RIPPER Rule Learner	`JRip`	NumOpt, NumFolds, MinWeights
Linear Regression	`lm`	None
Regression Trees	`rpart`	cp
Model Trees	`M5`	Pruned, smoothed, rules
Neural Networks	`nnet`	size, decay
Support Vector Machines (Linear Kernel)	`svmLinear`	C
Support Vector Machines (Radial Basis Kernel)	`svmRadial`	C, sigma
Random Forests	`rf`	mtry

ML: Hyperparameter Tuning

- The goal of **automatic tuning** is to search a set of candidate models comprising a matrix, or **grid**, of parameter combinations.
- By default, **`caret`** searches at most three values for each of the *p* parameters → max of $\mathbf{3}^{p}$ models.
- For instance, the automatic tuning for k-NN will compare 3 models with k=5, k=7 and k=9 (default values).
- If the default grid is not suitable, caret allows the user to provide a custom search grid and include more combinations with **`expand.grid.()`**.

ML: Hyperparameter Tuning

- To see the available parameters for a model we use the function **`modelLookup()`**.

- For example:

```
> modelLookup("C5.0")
  model parameter                   label forReg forClass probModel
1  C5.0    trials # Boosting Iterations  FALSE     TRUE      TRUE
2  C5.0     model            Model Type  FALSE     TRUE      TRUE
3  C5.0    winnow                Winnow  FALSE     TRUE      TRUE
```

- The **`caret`** package also allows the user to calculate a variety of model performance statistics including accuracy and kappa (for classifiers) and R-squared or RMSE (for numeric models).
- Cost-sensitive measures such as sensitivity, specificity, and area under the ROC curve (AUC) can also be extracted.

ML: Hyperparameter Tuning

Example of using grid search in a gradient boosting machine (GBM) model.

There are four main tuning parameters:

- **n.trees** = number of iterations, i.e. trees,
- **interaction.depth** = complexity of the tree (the maximum depth of each tree). Default is 1.
- **shrinkage** = a shrinkage parameter applied to each tree in the expansion (learning rate: how quickly the algorithm adapts)
- **n.minobsinnode** = the minimum number of observations in the terminal nodes of the trees

```
expand.grid(interaction.depth = c(1, 5, 9),
                        n.trees = (1:30)*50,
                        shrinkage = 0.1,
                        n.minobsinnode = 20)
```

ML: Hyperparameter Tuning

Example of Random Forests hyperparameter tuning with caret

```
1. Define hyperparameter tuning settings
grid <-expand.grid(mtry=2:40)

control <- trainControl(method='repeatedcv',
                        number=10, repeats=5,
                        search='grid')


2. Train a Random Forests classification model
rf.model <- train(Class~., data = TrainSet,
                  method = 'rf',
                  metric = 'Accuracy',
                  ntrees = 800,
                  tuneGrid = grid,
                  trControl = control )
```

Summary

- Introduction to Probabilistic Methods: Naïve Bayes
- Introduction to Black Box Methods: SVM
- Introduction to hyperparameter tuning concept

www.cranfield.ac.uk

T: +44 (0)1234 750111

@cranfielduni

@cranfielduni

/cranfielduni