Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track

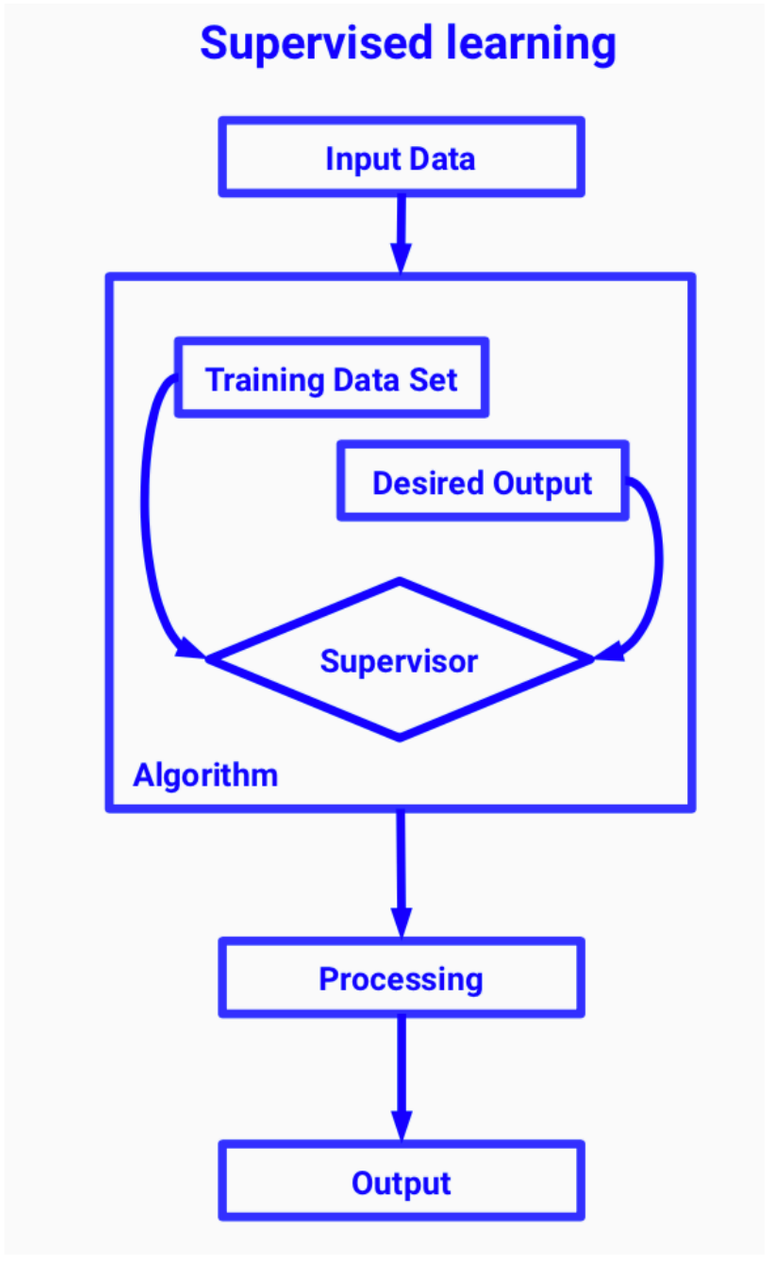
Lecture 3, 14/09/2020

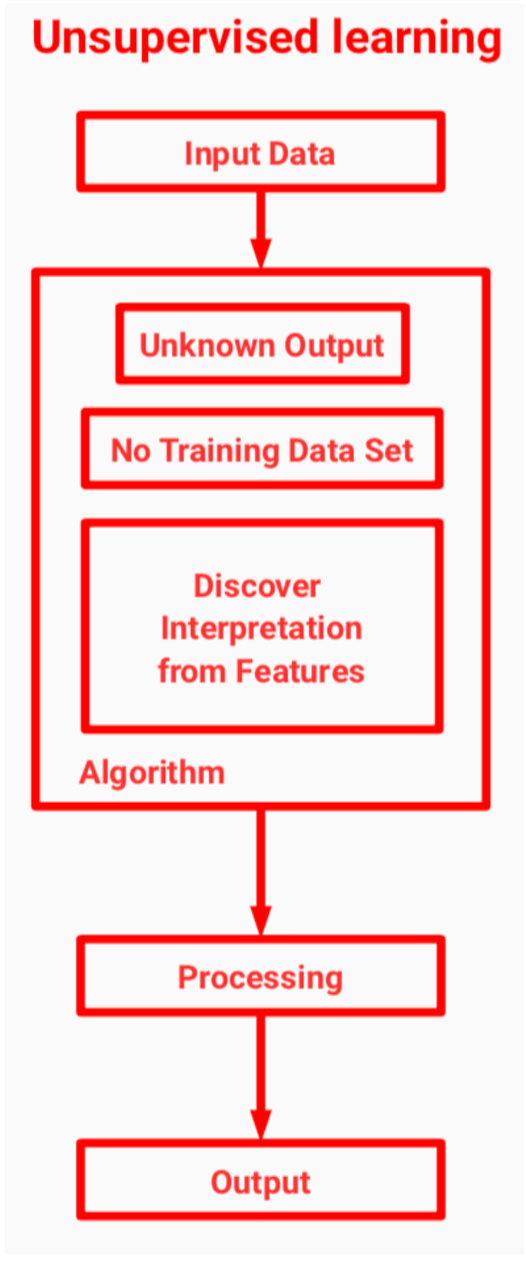
Today's lecture

- Supervised Learning for classification
- Linear classifiers and least squares
- Decision Theory
- Logistic regression and neural nets for classification
- Decision Trees and Random Forests
- Support Vector Machines

Classification with Linear Models

Supervised Learning recap





this lecture ***lecture 4*** ***lecture 5***

Supervised Learning recap

problems in **Supervised Machine Learning** are defined by the following ingredients:

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

The cost function measures how well the model (for a specific choice of its parameters) is able to **describe the input dataset**

example of cost function for single dependent variable: sum of residuals squared

$$C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right) = \frac{1}{n} \sum \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta})\right)^2$$

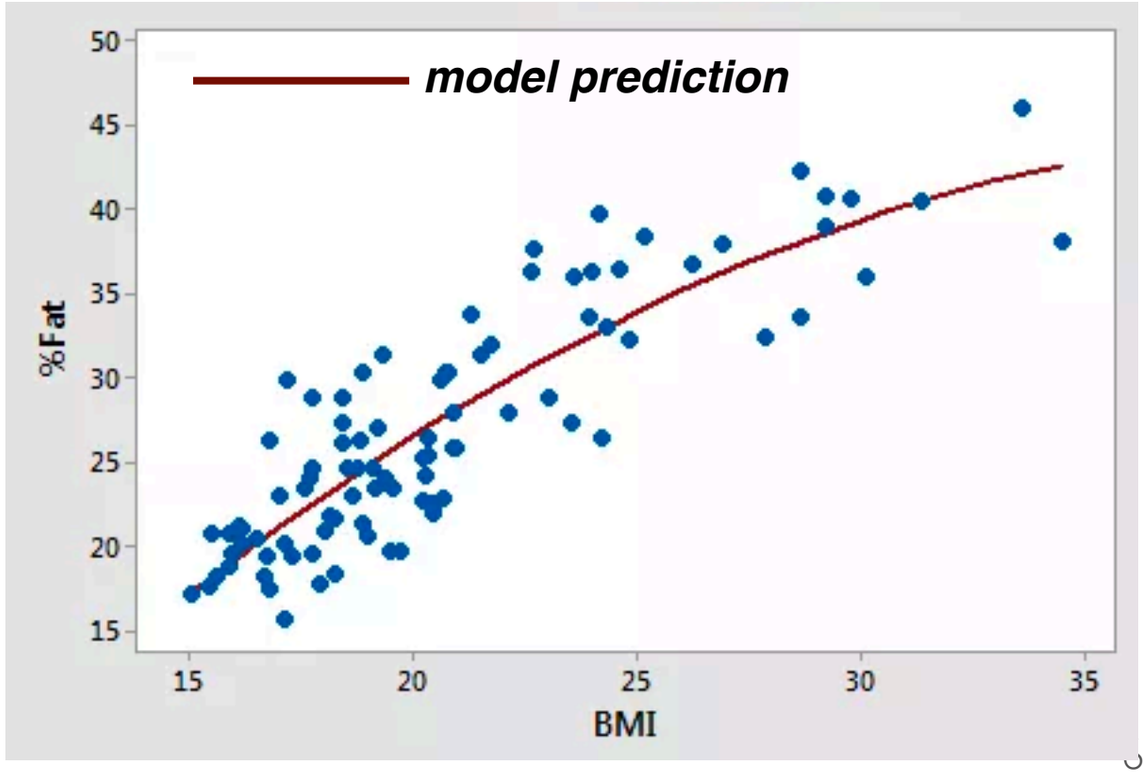
Supervised learning

We denote as **supervised learning** the ML task of **learning a function** that maps a **vector of inputs** to a **vector of outputs** from a finite set of training example

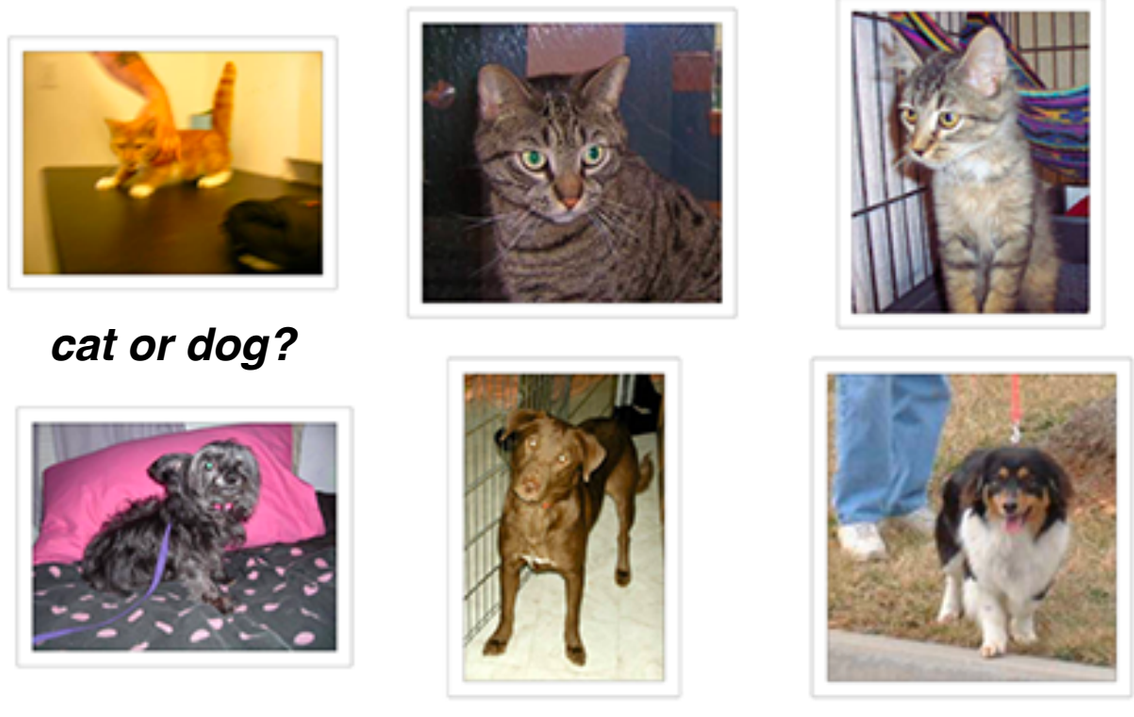
note that some assumptions will be needed: a function is an **infinite-dimensional object** but learning takes place from a **finite number of examples**

main property of supervised learning: the **training samples are labeled**

continuous outputs: regression



discrete outputs: classification



Classification tasks

The goal is to **predict a class label** from a pre-defined list of possibilities

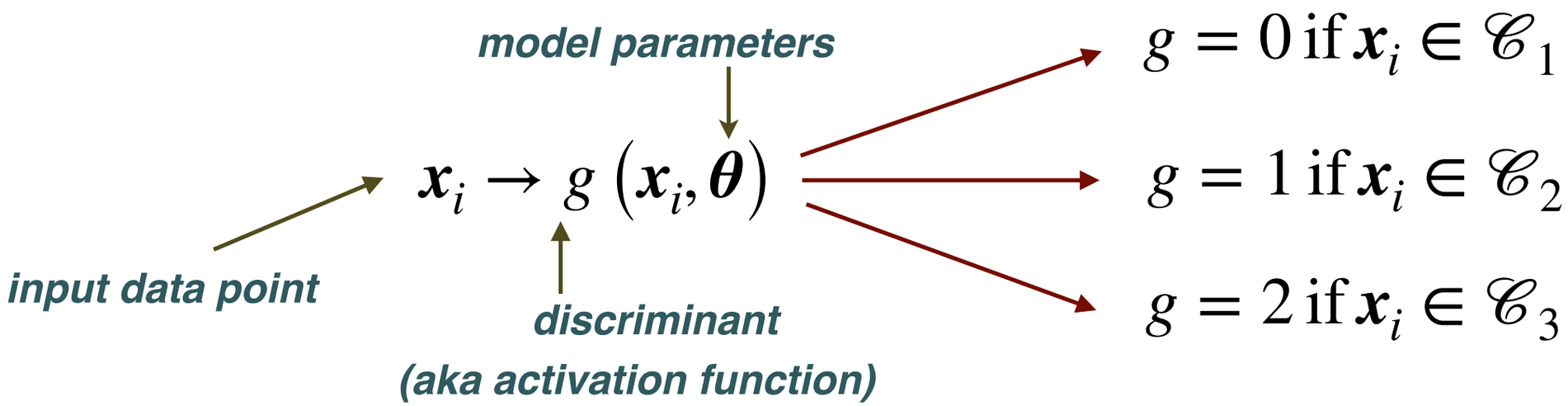
the simplest type of problem is **binary classification** (yes/no problems)

e.g. should I put this email in the spam folder?

but in general one considers **multiclass classification** (> 2 categories)

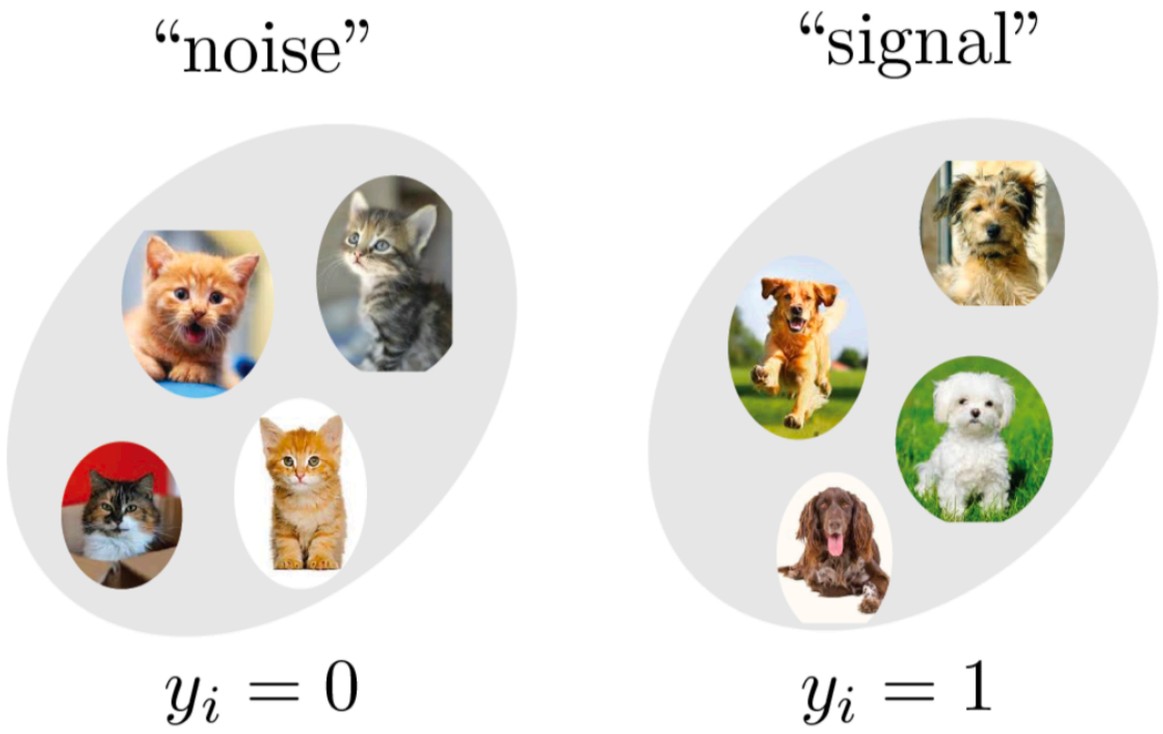
e.g. which type of bird is the one I just photographed?

In the context of ML applications there exist a large number of approaches to classifications tasks. The most basic one is based on assembling a **discriminant function** that maps each input data point to its specific class

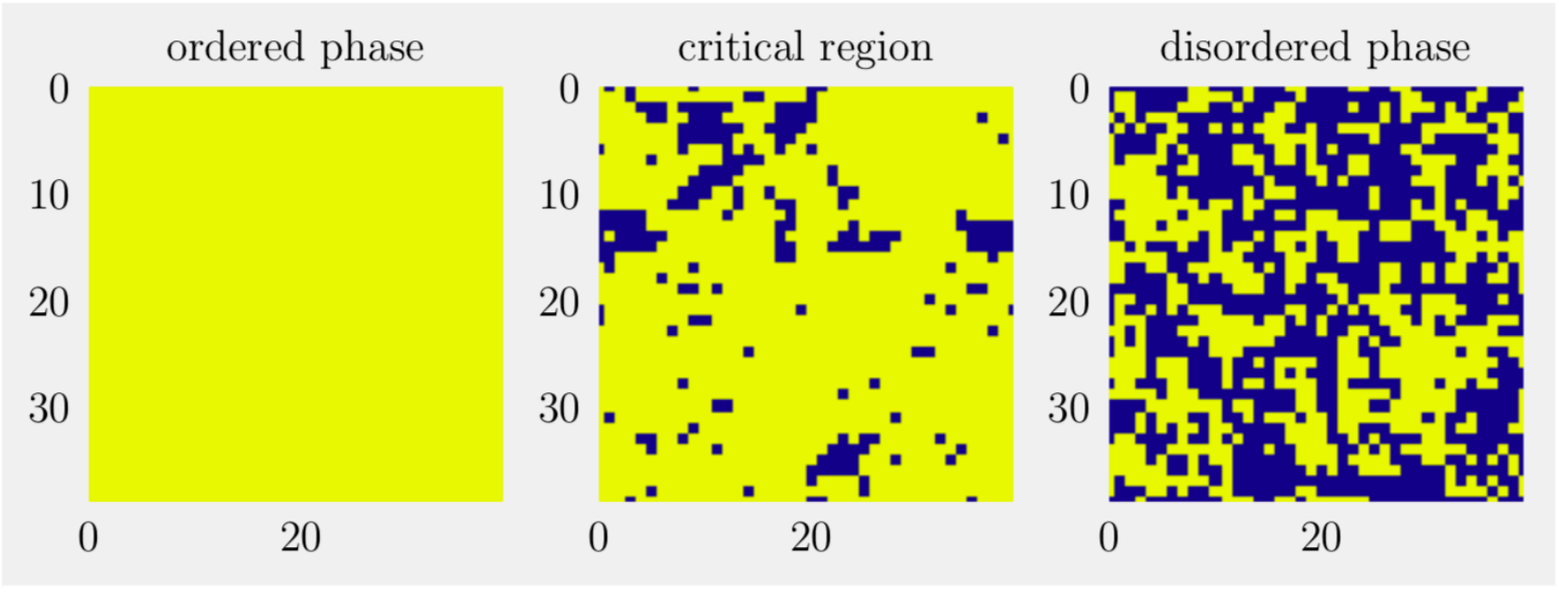


Classification tasks

relevant for Machine Learning applications where outcomes
are discrete variables, eg. **categories in classification problems**



can we tell apart
cats from dogs?



can we identify the phase
(ordered/disordered) of
spin configurations in 2D Ising?

Linear classifiers

In this class of problems, the **dependent variables y_i** are discrete and take values *m=0, …, M-1*, so the index *m* also labels the *M* categories

The goal is to **classify the *n* input samples**, each composed by ***p*** **features,** intro the ***M*** **possible categories** of the problem

A simple classifier is the **perceptron:** a **linear classifier** that categorises examples from a linear combination of the features

$$\sigma(s_i) = \text{sign}(s_i) = \text{sign}\left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right)$$

p features of the samples

model parameters

a perceptron is a **hard classifier** where each sample is assigned to a category with 100% probability

more complex models can be used as classifiers, including NNs

Linear classifiers

In this class of problems, the **dependent variables y_i** are discrete and take values *m=0, …, M-1*, so the index *m* also labels the *M* categories

The goal is to **classify the *n* input samples**, each composed by ***p*** **features,** intro the ***M*** **possible categories** of the problem

A simple classifier is the **perceptron:** a **linear classifier** that categorises examples from a linear combination of the features

$$\sigma(s_i) = \text{sign}(s_i) = \text{sign}\left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right)$$

p features of the samples

model parameters

a perceptron is a **hard classifier** where each sample is assigned to a category with 100% probability

more complex models can be used as classifiers, including NNs

simple linear models are great to gain physical intuition about how ML algorithms work

Linear classifiers

the perceptron (linear classifier) only has **two possible outcomes**

$$\boldsymbol{x}_i \rightarrow g\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right) \begin{cases} g = 1 \text{ if } \boldsymbol{x}_i \in \mathscr{C}_1 \text{ , } \left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right) \geq 0 \\ g = -1 \text{ if } \boldsymbol{x}_i \in \mathscr{C}_1 \text{ , } \left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right) < 0 \end{cases}$$

the **decision boundary** of the classifier is then defined by the condition

$$\left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right) = 0$$

the properties of the decision boundary of this problem can be easily evaluated

For two points that belong to this boundary one has

$$\begin{cases} \left(\boldsymbol{x}_1^T \boldsymbol{\theta} + b_0\right) = 0 \\ \left(\boldsymbol{x}_2^T \boldsymbol{\theta} + b_0\right) = 0 \end{cases} \rightarrow \left(\boldsymbol{x}_1^T - \boldsymbol{x}_2^T\right) \boldsymbol{\theta}$$

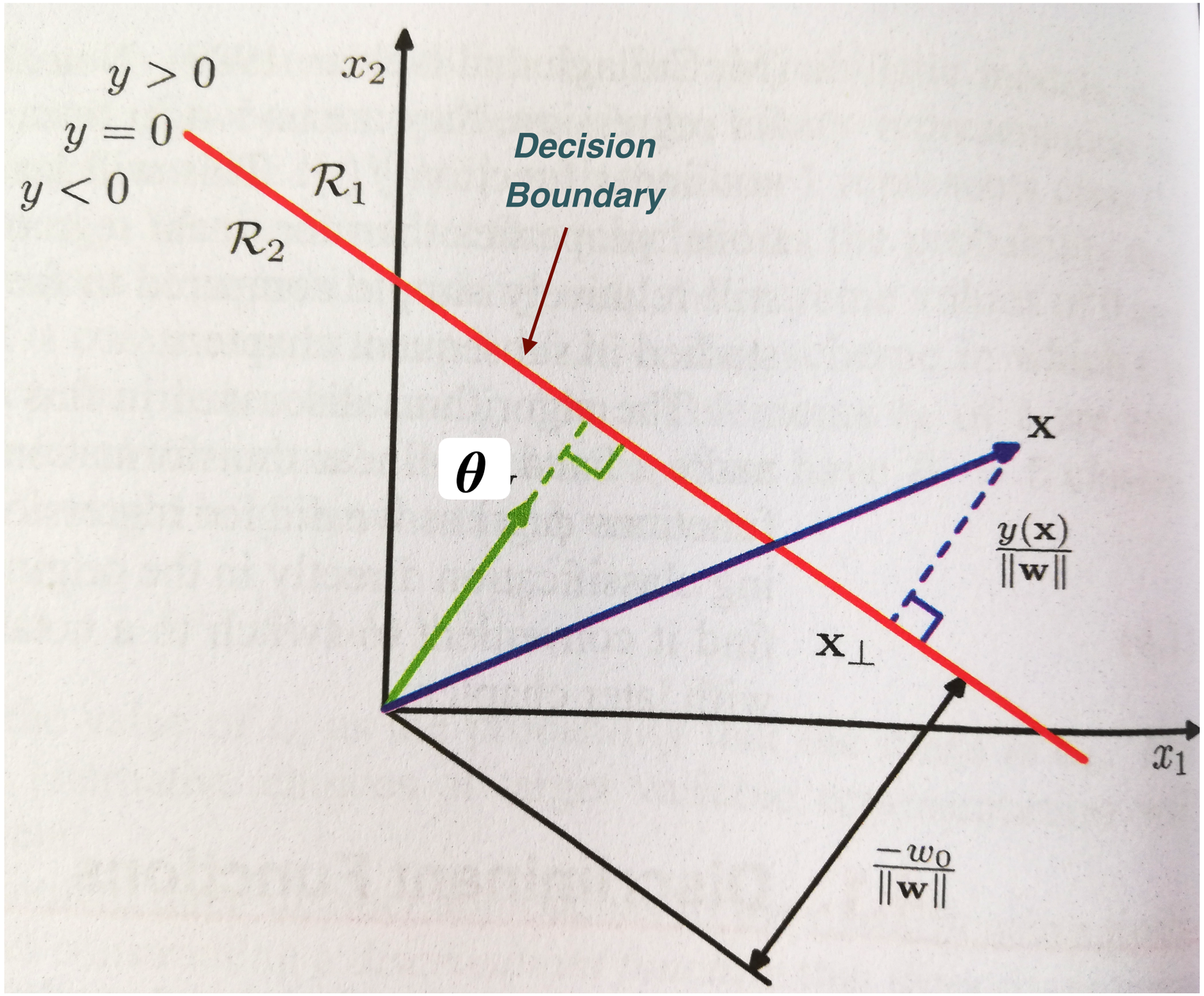
the model parameter vector is perpendicular to the decision boundary!

several other geometric properties can be evaluated

recall that both x and θ live in the p-dimensional feature space

Linear classifiers

example of linear classification for 2d datasets



Training the model

as in the case of regression problems, also for classifications we need to introduce a **figure of merit** (error function) to be optimised

the simplest option is based again on **least squares.** Let's consider a binary linear classifier

the sign function is added back after the training

$$g\left(\boldsymbol{x}_i\right) = \left(\boldsymbol{x}_i^T\boldsymbol{\theta} + b_0\right)$$

recall that x_i and θ have p features each

and one is given a **training dataset** with *N* instances

$$\left\{\boldsymbol{x}_i^T, t_i\right\}, \quad i = 1,\ldots,N, \quad t_i \in (-1,1)$$

in this case the error function depending on the **model parameters** is

$$E_{\mathrm{tr}}(\boldsymbol{\theta}, b_0) = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\boldsymbol{x}_i^T\boldsymbol{\theta} + b_0\right) - t_i\right)^2$$

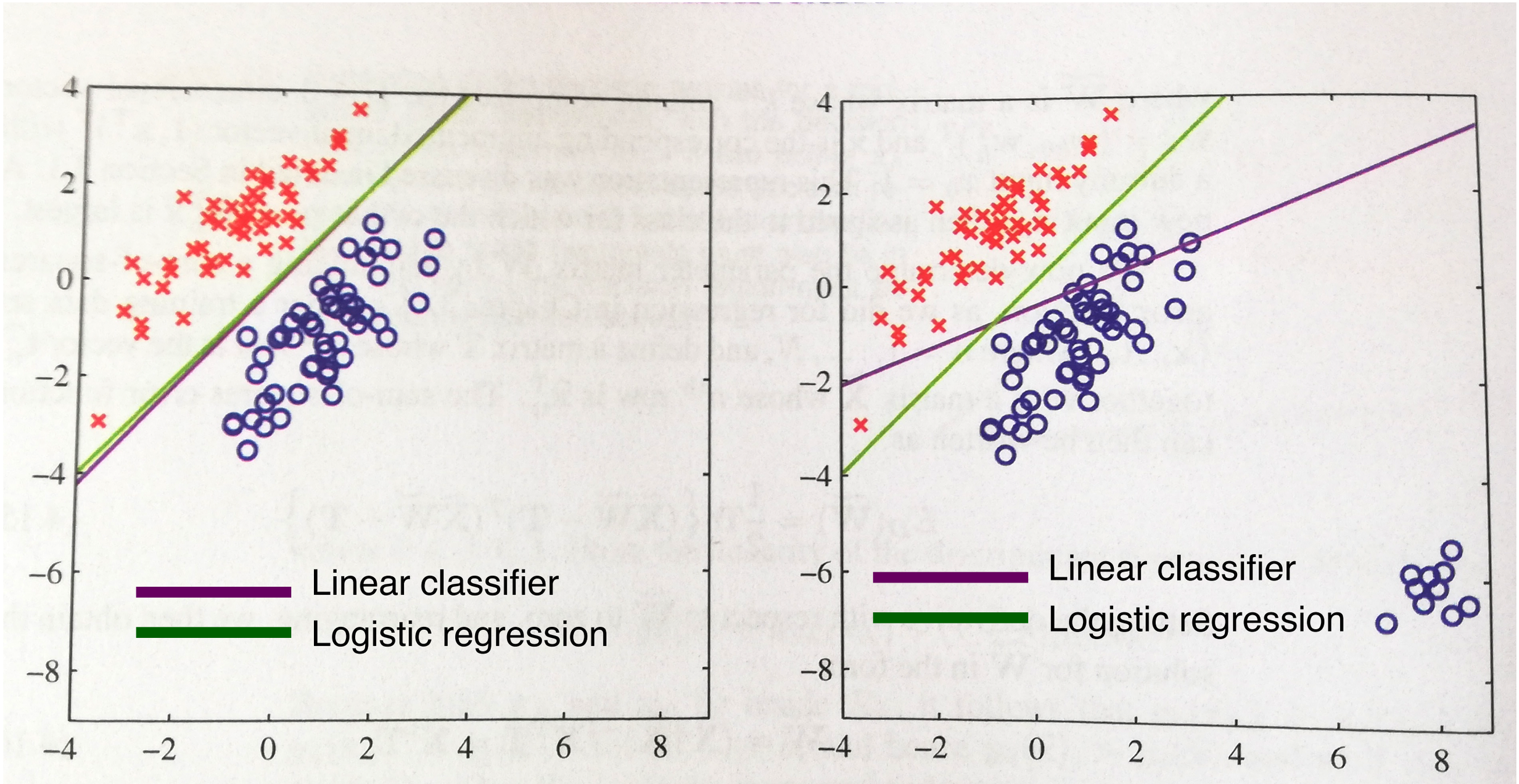
Which is amenable to analytic optimisation, for example

linear system of equations for the model params

$$\frac{\partial}{\partial b_0}E_{\mathrm{tr}} = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\boldsymbol{x}_i^T\boldsymbol{\theta} + b_0\right) - t_i\right) = 0 \rightarrow b_0 = \frac{1}{N}\sum_{i=1}^{N}\left(t_i - \boldsymbol{x}_i^T\boldsymbol{\theta}\right) = 0$$

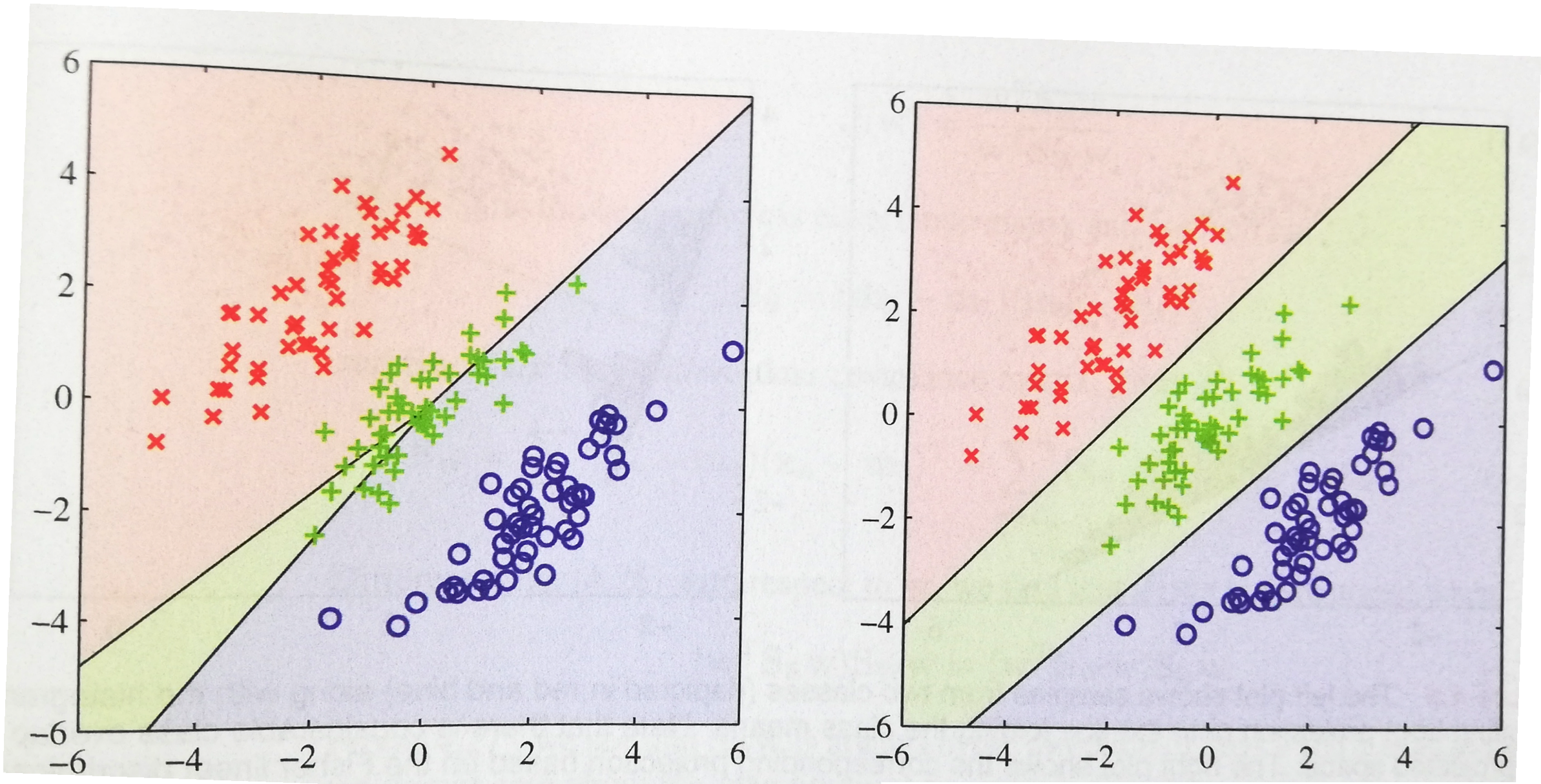
Limitations of the linear classifier

☑ As in regression problems, a linear classifier trained with least squares lacks **robustness with respect to the presence of outliers**



Limitations of the linear classifier

☑ Also the categorisation for multiclass problems can be problematic even when the **decision boundaries** should be easy to find



Linear classifier

Logistic regression

Logistic Regression

Logistic regression

in many cases a **soft classifier,** that **outputs the probability** of a given category, is advantageous over a hard classifier. Also, non-linear models exhibit many positive features for classification problems

In **logistic regression** the probability that a data point $\boldsymbol{x}_i$ belongs to a category y_i is given by

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}} = \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})$$

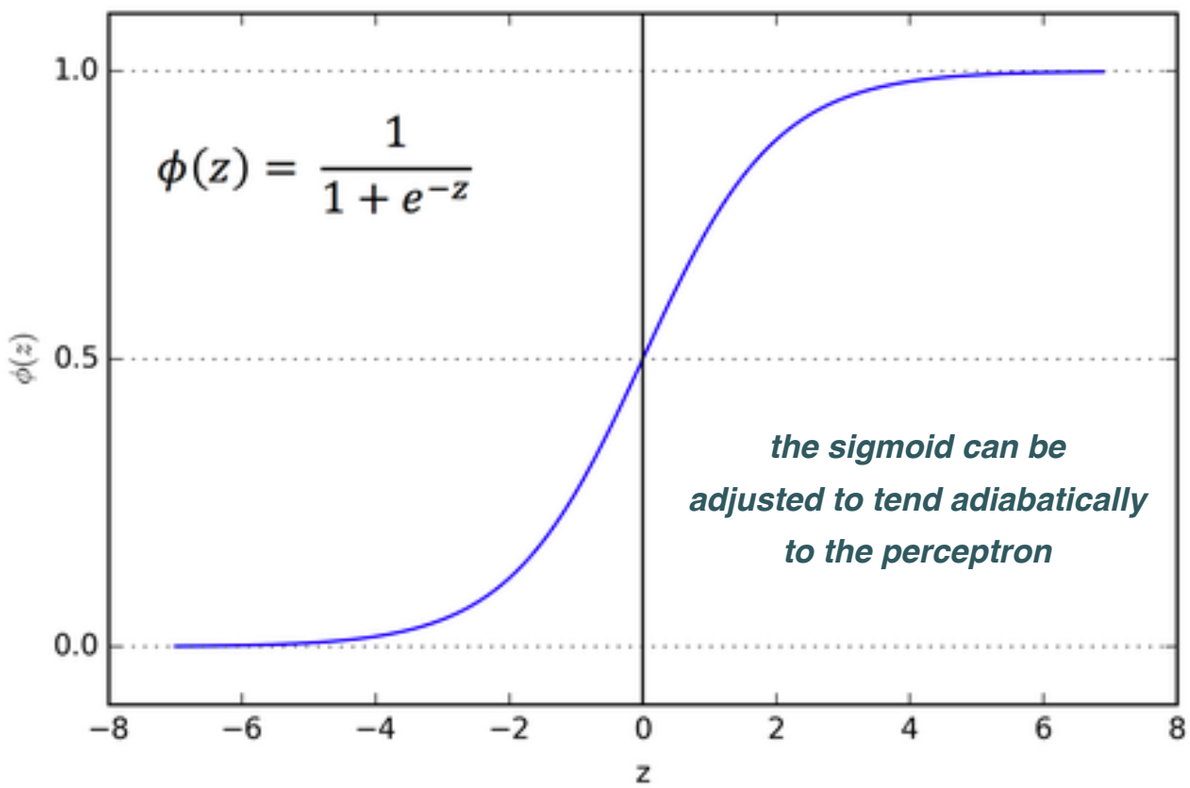
non-linear sigmoid function

$$P(y_i = 0 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta}) = \sigma(-\boldsymbol{x}_i^T \boldsymbol{\theta})$$

where we have used the logistic (or sigmoid) function:

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

$$\sigma(-s) = 1 - \sigma(s)$$



Logistic regression

cost function for logistic regression from **Maximum Likelihood Estimation** (MLE):
choose parameters that maximise the probability of seeing the observed data

$$P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \Big(P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta})\Big)^{y_i} \times \Big(P(y_i = 0 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta})\Big)^{1-y_i}$$

``probability interpolation': 'recovers limits when y_i=0, y_i=1

$$P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \Big(\sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{y_i} \times \Big(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{1-y_i}$$

assuming that all observations are Bernoulli **independent**, the total likelihood is

$$\mathscr{L}(\boldsymbol{\theta} \,|\, \mathscr{D}) = \prod_{i=1}^{n} P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \prod_{i=1}^{n} \Big(\sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{y_i} \times \Big(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{1-y_i}$$

note that this is soft classification: the probability is not only 0 or 1 but any value in between is possible

Logistic regression

cost function for logistic regression derived from Maximum Likelihood Estimation (MLE): choose parameters that maximise the probability of seeing the observed data

$$\mathscr{L}(\boldsymbol{\theta} \,|\, \mathscr{D}) = \prod_{i=1}^{n} P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \prod_{i=1}^{n} \left(\sigma(\boldsymbol{x}_i^T \boldsymbol{\theta} \right)^{y_i} \times \left(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta} \right)^{1-y_i}$$

Since the cost function is the negative log-likelihood, we find that for logistic regression

$$E(\boldsymbol{\theta}) = -\log \mathscr{L} = \sum_{i=1}^{n} - \left(y_i \log \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta}) + (1 - y_i) \log(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})) \right)$$

which is known as the cross-entropy function

The parameters of the model are determined by minimising the cross-entropy

$$\widehat{\theta} = \arg\ \min_{\theta} \left\{ \sum_{i=1}^{n} \left(-y_i \log \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta}) - (1 - y_i) \log(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})) \right) \right\}$$

note that no analytic solution is possible, and numerical methods are required

one can use more complex models to parametrise the probabilities, such as NNs

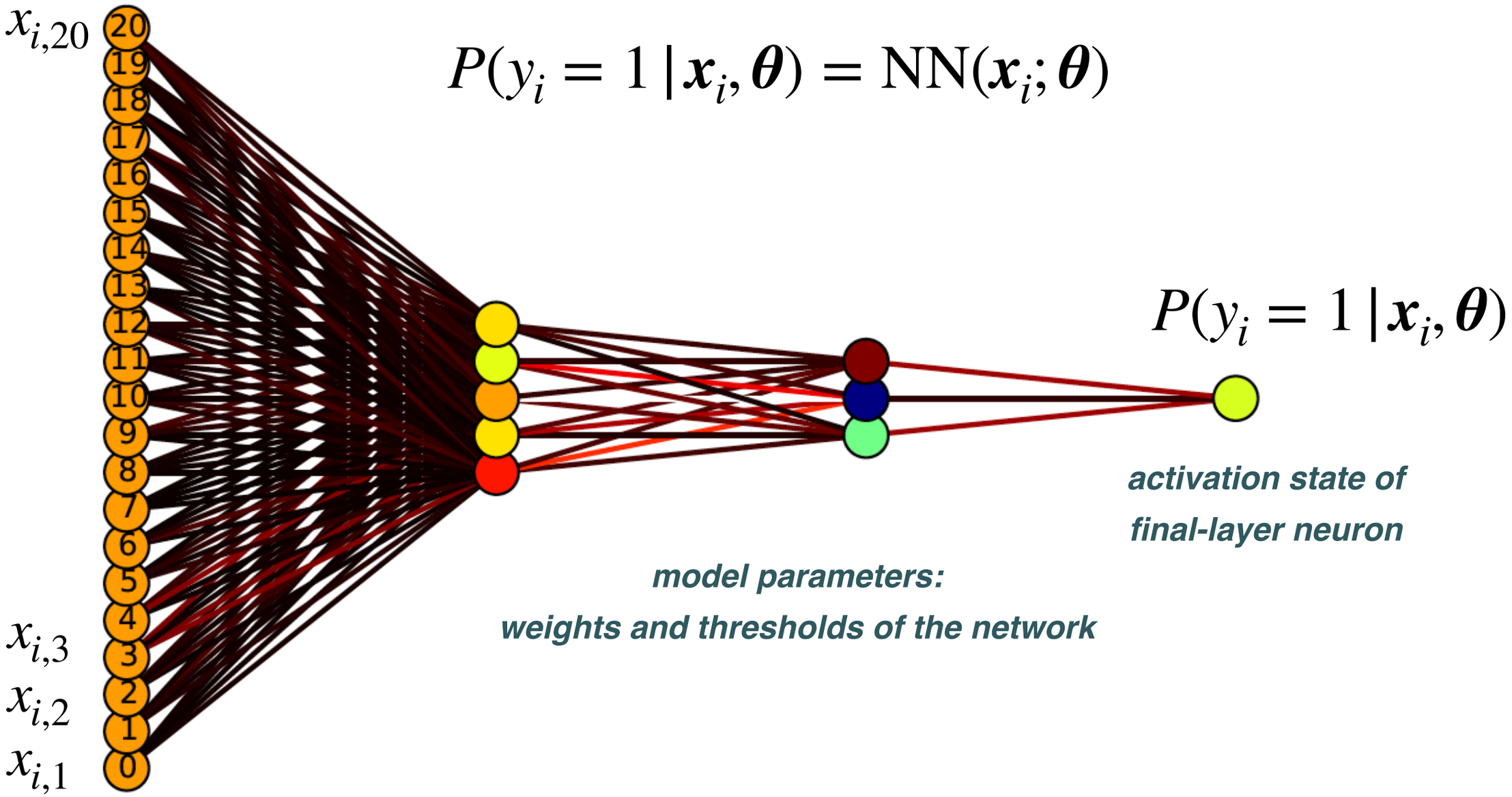
Neural Networks for Classification

instead of modelling the classification probability by a **simple logistic function**

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}}$$

one can adopt more complex models, such as deep neural networks

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i; \boldsymbol{\theta})$$



Decision Theory

Decision Theory

we have discussed several ML models that provide a **probability distribution** for a given outcome given some inputs. However, we still need to **take decisions** based on this info

assume that we have the usual **training dataset**, either for regression or classification

$$\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\right) \ , \qquad \boldsymbol{t} = \left(t_1, \ldots, t_N\right)$$

depending on our application, we can try to achieve different goals

- Given a new input vector ***x,*** make a prediction for *t*
- Given a new input vector ***x,*** **make a decision** based on the expectations for *t*
- Construct the joint probability distribution *p(**x**,t)* associated to the training dataset: this is known as the **inference problem**

the goal of Decision Theory is to help us making **optimal decisions** given our knowledge of the involved probabilities

Decision Theory

for example, our goal could be to **minimise the misclassification rate**

e.g. in clinical diagnosis that require an invasive treatment

we will have trained our classifier on the input examples, and end up with a map of the input space into **decision regions** (assigned to an specific class) separated by **decision boundaries**

the probability of incurring in a **classification mistake** will be given by

$$p(\text{mistake}) = p(\boldsymbol{x} \in \mathscr{R}_1, \mathscr{C}_2) + p(\boldsymbol{x} \in \mathscr{R}_2, \mathscr{C}_1) = \int_{\mathscr{R}_1} p(\boldsymbol{x}, \mathscr{C}_2) d\boldsymbol{x} + \int_{\mathscr{R}_2} p(\boldsymbol{x}, \mathscr{C}_1) d\boldsymbol{x}$$

probability that an input in decision region R_1 is classified as class C_2

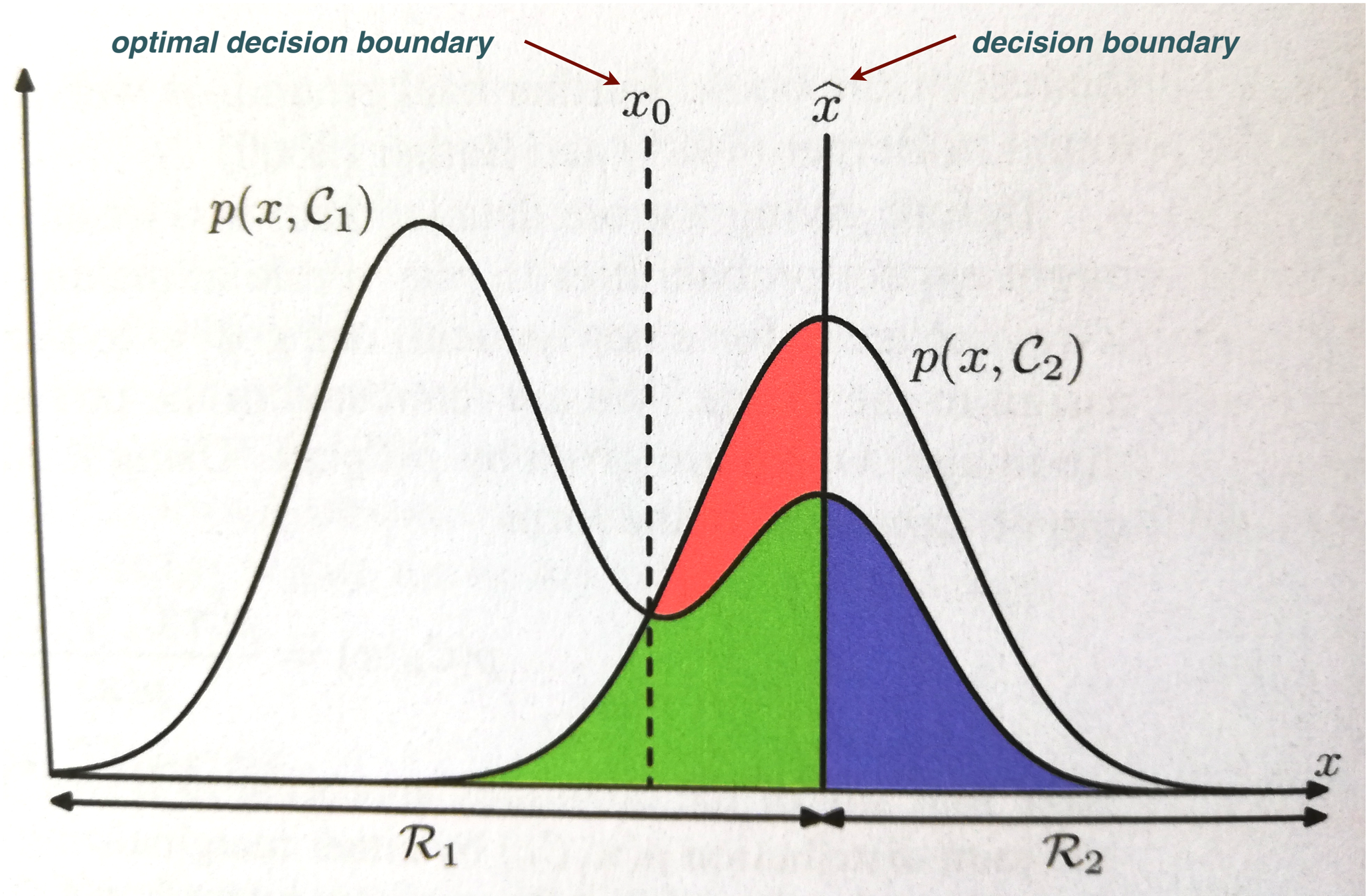
note that we still need to decide the **rule** that assigns a given point ***x*** to each category.

From the product rule of probabilities we have that

$$p(\boldsymbol{x}, \mathscr{C}_k) = p(\mathscr{C}_k | \boldsymbol{x}) p(\boldsymbol{x})$$

The misclassification rate is minimised by the decision boundary defined when each point ***x*** is assigned to the class for which its posterior probability $p(\mathscr{C}_k | \boldsymbol{x})$ is the largest

Decision Theory



red, green, blue regions: **misclassification probability**

green+blue constant as decision boundary varied: optimal choice is that one that minimised red region

Metrics for binary classification

there are many **useful metrics** that are used in ML binary classification problems

example classification problem: 34 training samples, of which

True Positives (TP) e.g. 8	False Positives (FP) e.g. 2
False Negatives (FN) e.g. 4	True Negatives (TN) e.g. 20

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{8 + 20}{8 + 20 + 2 + 4} = 0.823$$

however accuracy is not the right metric for classification problems

defined by a large disparity between classes, e.g. when TN >> TP

$$\text{Accuracy} \simeq \frac{\text{TN}}{\text{TN} + \text{FN}} + \dots$$

Metrics for binary classification

there are other **useful metrics** that are used in ML binary classification problems

example classification problem: 34 training samples, of which

True Positives (TP) e.g. 8	False Positives (FP) e.g. 2
False Negatives (FN) e.g. 4	True Negatives (TN) e.g. 20

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{8 + 20}{8 + 20 + 2 + 4} = 0.823$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 0.80$$

proportion of correct positive classifications

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 0.67$$

proportion of correct actual positive classifications

ROC curve

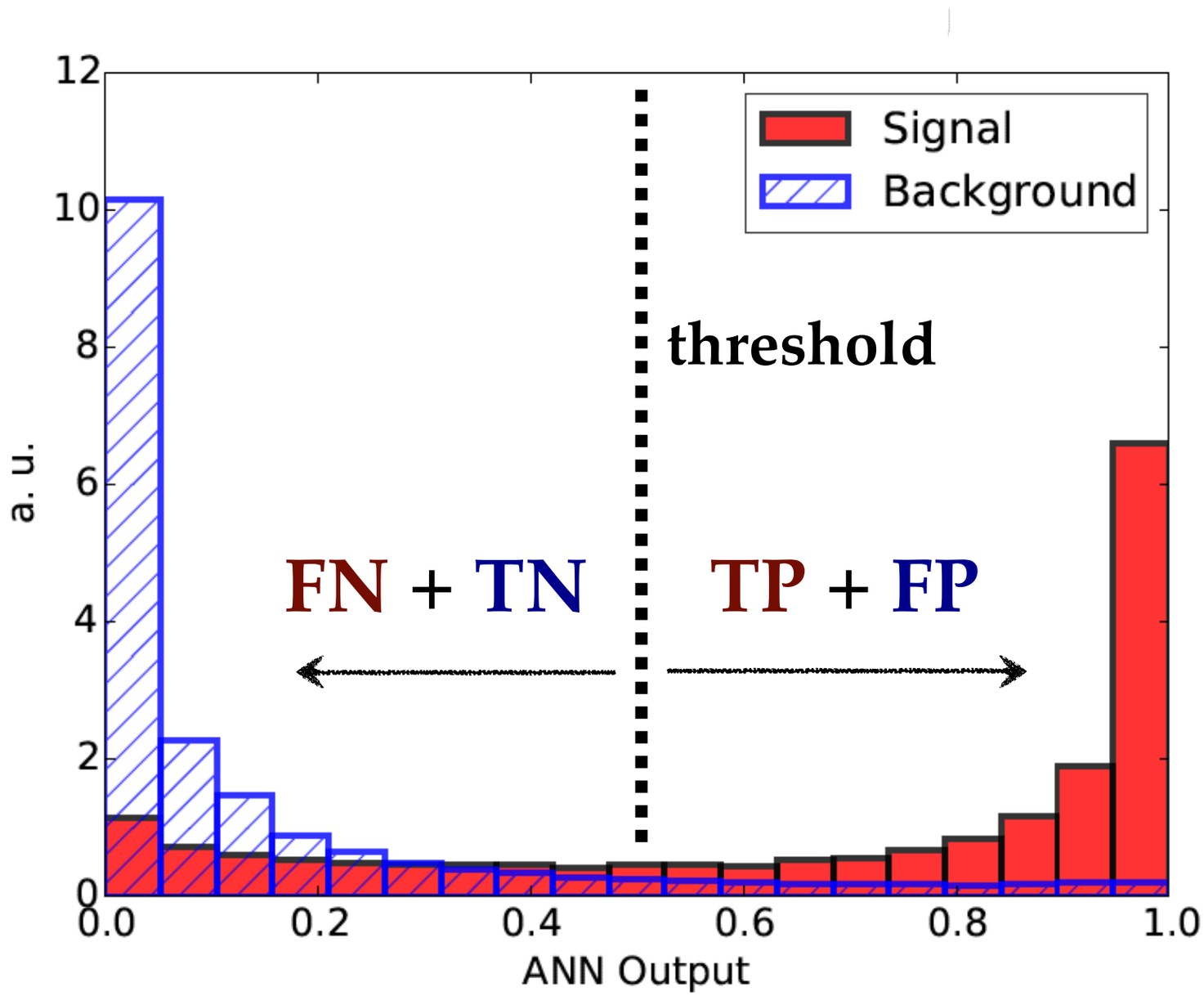
In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

how should we select this threshold for classification?

ROC curve

In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

- ☑ A conservative threshold will **maximise TP**, but also FP might be large
- ☑ An aggressive threshold **reduces FP** but then FN might be large.



ROC curve

In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

- ☑ A conservative threshold will **maximise TP**, but also FP might be large
- ☑ An aggressive threshold **reduces FP** but then FN might be large.

effect of threshold is quantified by the **Receiver Operating Characteristic** (ROC) curve

$$\text{Recall} = \text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

← *proportion of correctly classified positives*

vs

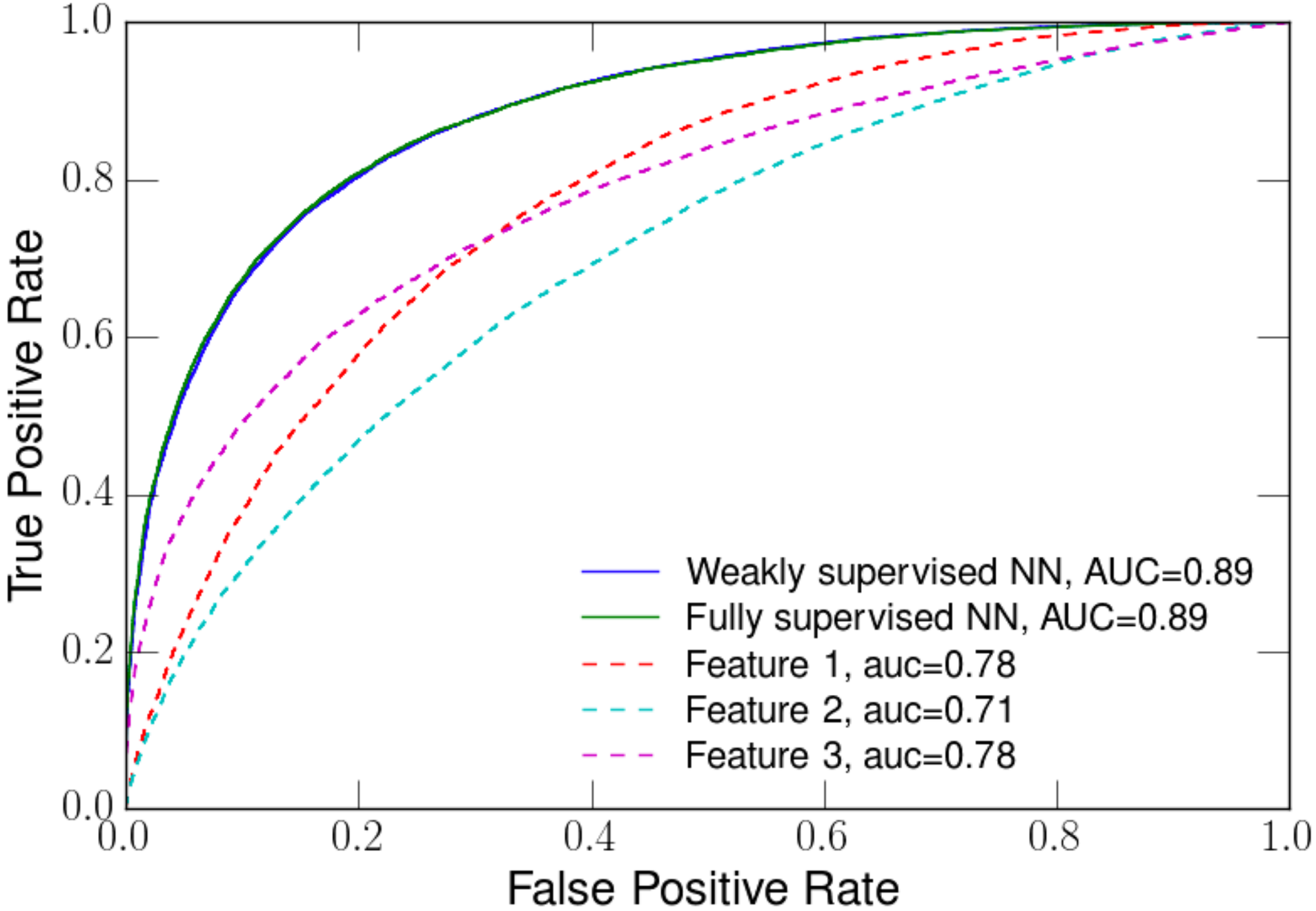
$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

← *incorrectly classified negatives*

a good classifier should maximise TPR while minimising FPR

ROC curve

$$\text{Recall} = \text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

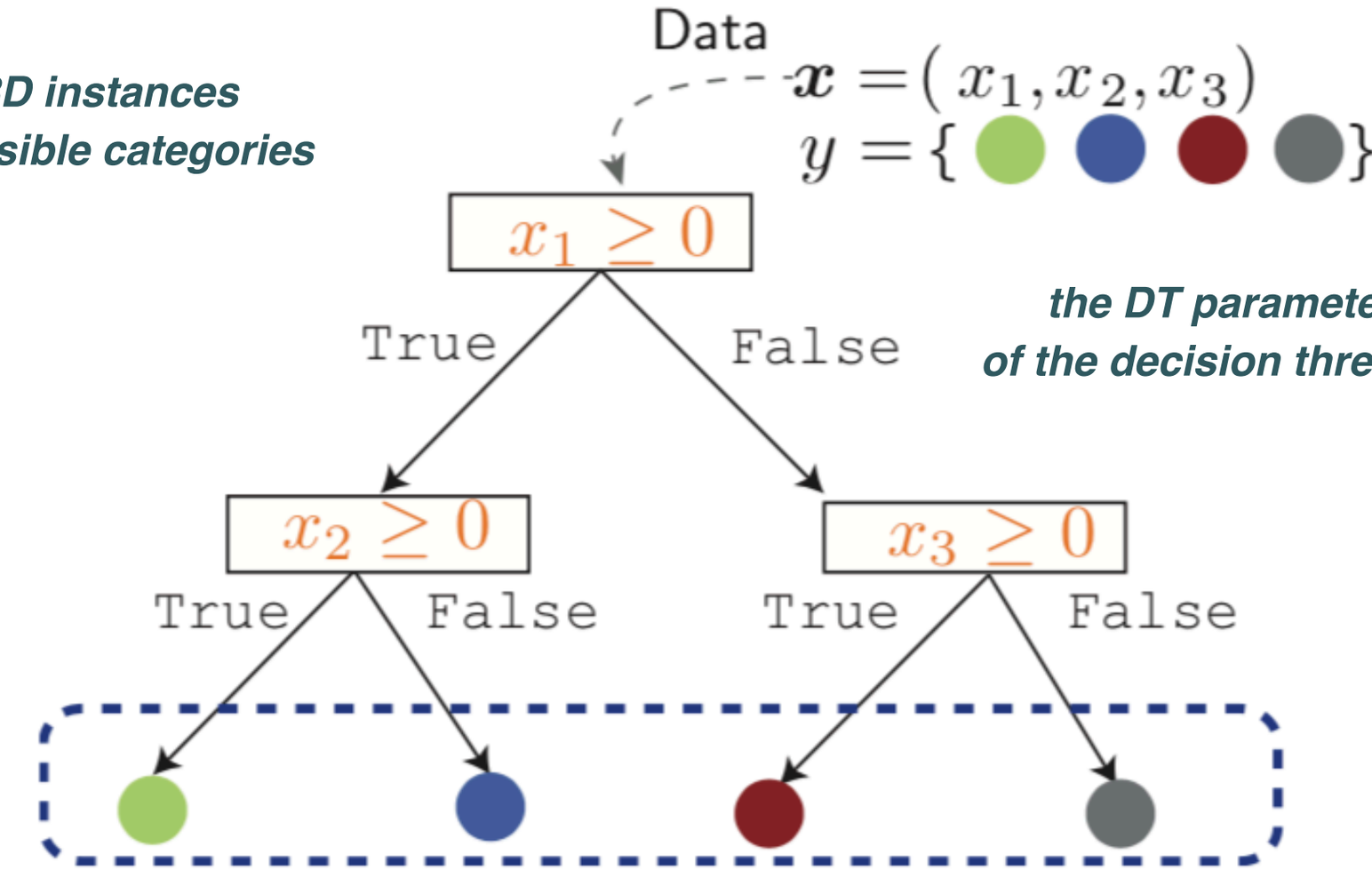


Decision Trees and Random Forests

Decision Trees

Decision Trees are classifiers that work by partitioning the input space into **hypercubes** and then assign a model (e.g. a constant) to each region

classify 3D instances into four possible categories

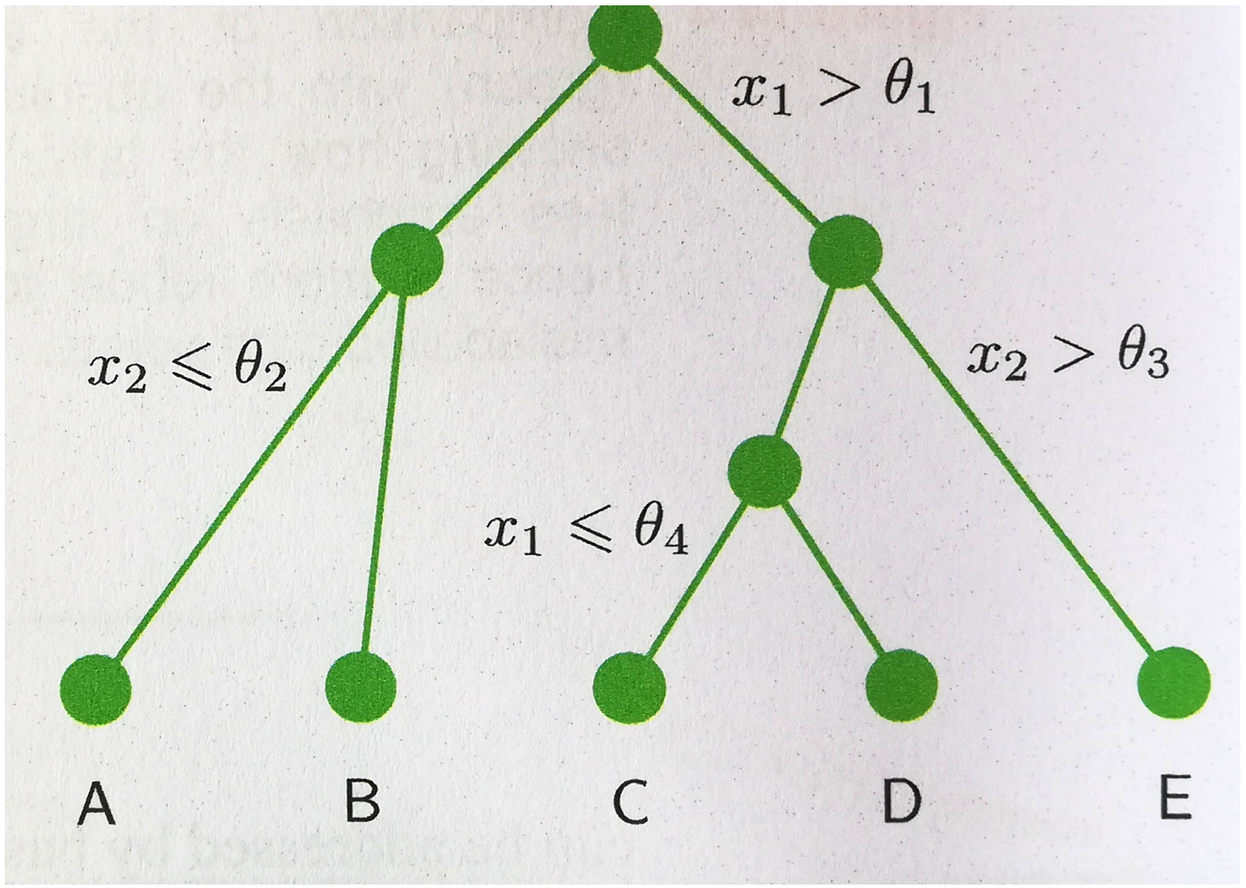


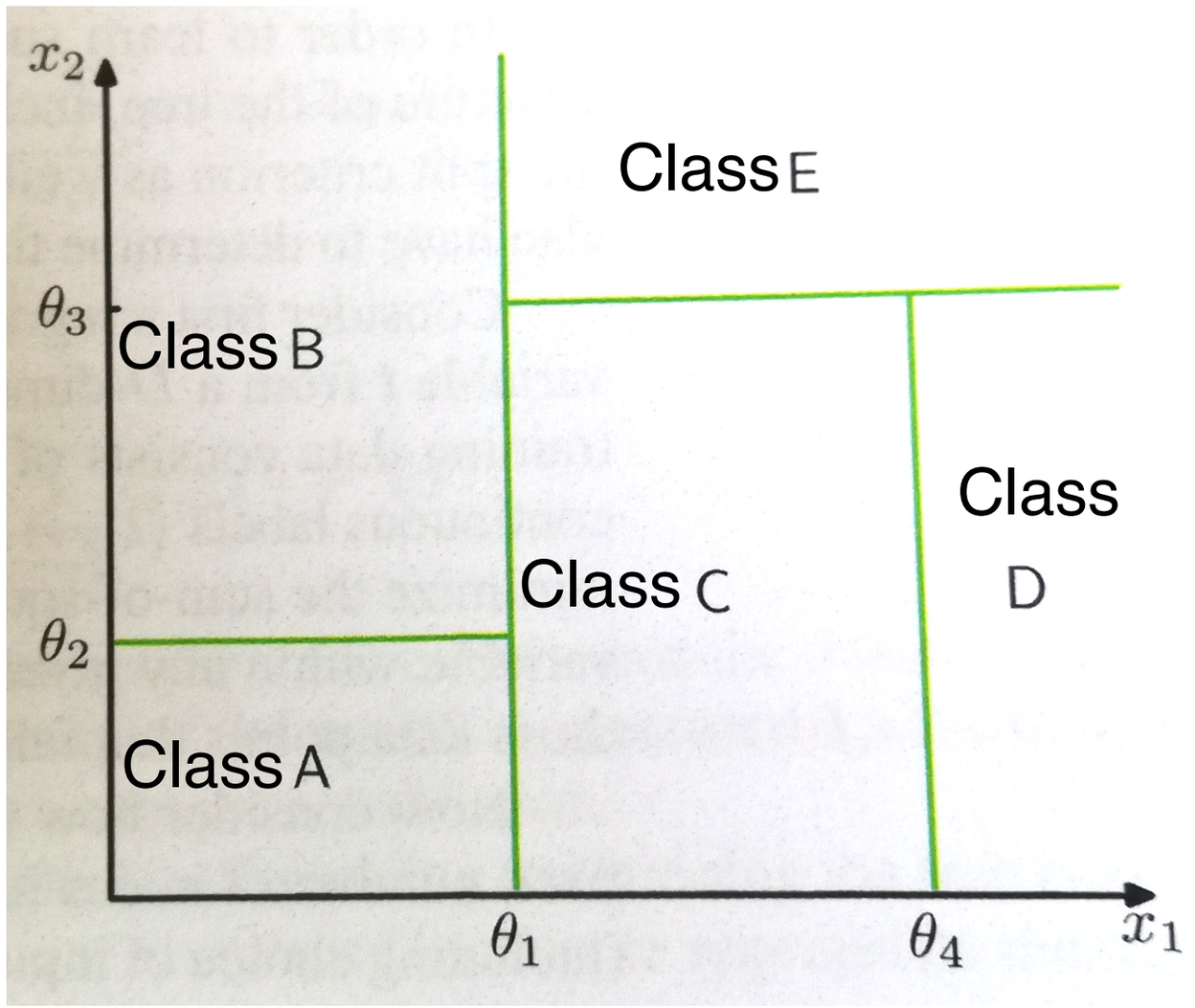
the DT parameters are the values of the decision thresholds at each node

In ML, a **decision tree** is an algorithm which uses a series of questions to hierarchically partition the data, where each branch of the tree splits the data into smaller subsets

Decision Trees

Decision trees have the key property that their output is **human-interpretable:** sequence of binary decisions applied to individual input variables





to train a decision tree we need to determined which **variable is chosen at each node** for the split criterion and the value of the **threshold for the split**

Random Forests

individual trees have often **high variance** and are weak classifiers:

we can improve by incorporating them in an ensemble method

→ We need an ensemble of **randomised decision trees** (minimised correlations)

- ***(1)*** train each decision tree on a different bootstrapped dataset: **bagged decision tree**
- ***(2)*** use different random subset of features at each split: **random forest**

reduces correlations between trees that arise when only few features are strongly predictive

as other ML learning classifiers also Random Forests require some **regularisation**, for example a maximum depth of the tree, to control complexity and prevent overfitting

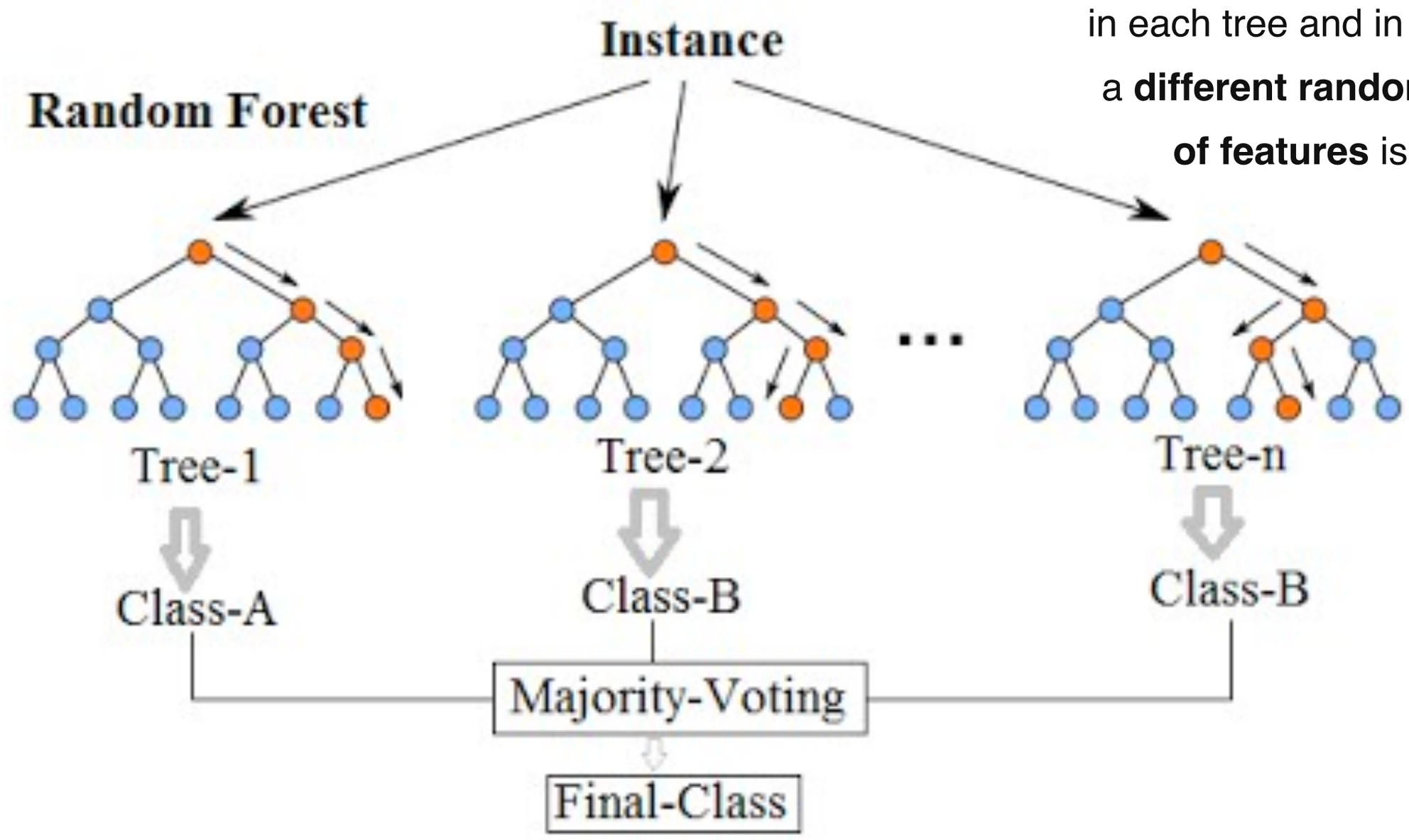
typically, a classification problem with p features, in RFs only $p^{1/2}$ features are used in each split

Random forests have other attractive features, for example, they can be used to **rank the importance of variables** in a regression or classification problem in a natural way

Random Forests

each decision tree in the ensemble is built upon a **random bootstrap sample** of the original data

in each tree and in each split, a **different random subset of features** is used



the final categorisation is assigned *e.g.* from **majority voting**

Support Vector Machines

Support Vector Machines

SVM are ML algorithms popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**

The advantages of support vector machines include:

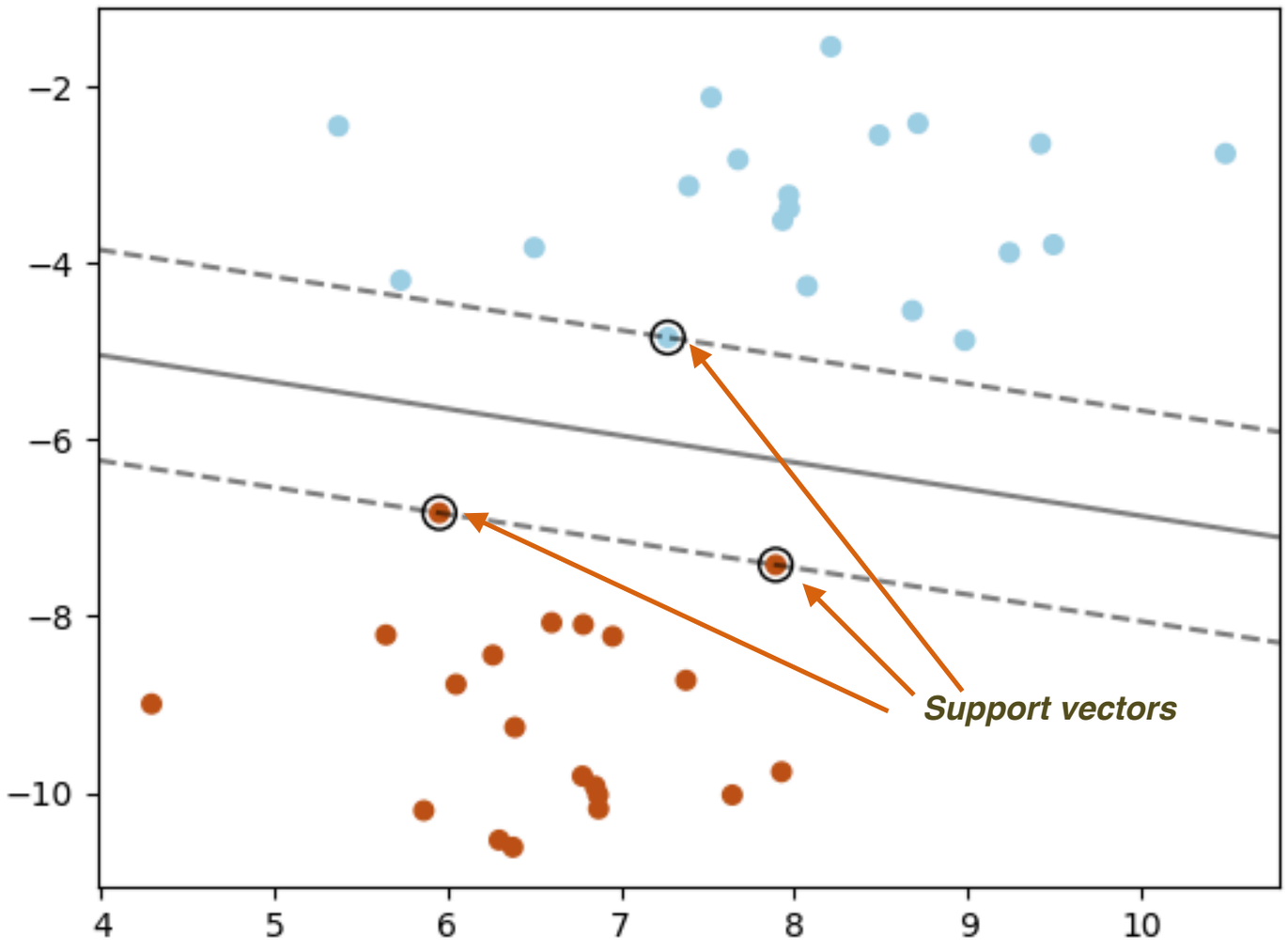
- Effective in **high dimensions.**
- Effective when number of dimensions is greater than number of samples.
- Based on a subset of training points in the decision function (**support vectors**)
- Different **kernels functions** can be used for the figure of merit

In ML, kernel methods operate in a high-dimensional,
implicit feature space without computing the coordinates in data space

Support Vector Machines

SVM are ML algorithms popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**



In SVM classification, the support vectors are samples that lie in the margin boundaries (at least for linearly separable problems)

Support Vector Machines

As other **classification algorithms**, the problem that SVM try to solve is the following. Starting from a training dataset

$$\{\boldsymbol{x}_i^T, t_i\}, \quad i = 1,\ldots,N, \quad t_i \in (-1,1)$$

our goal is to find the **model parameters** such that the prediction from

$$\text{sign}(\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) + b)$$

is correct for most of the samples. Here $\boldsymbol{\phi}$ is the a so-called **feature-space transformation**, where the *feature space* is where our data lives

Example of a feature-space transformation

$\boldsymbol{x}$	$\boldsymbol{\phi}(\boldsymbol{x})$
Height	***Height/Weight***
Weight	***Weight+Height***
Age	***Age * Weight***

Support Vector Machines

the problem then that SVM are aiming to solve is

margin between the two categories

$$\min_{\omega,b,\xi}\left(\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}+C\sum_{i=1}^{n}\xi_i\right)$$

subject to the **additional constraint** that

Class label of sample i → $t_i \times \left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_i)+b\right) \geq 1-\xi_i$ ← *tolerance (regulator)*

which essentially aims to **maximise the margin between the two categories** while including a penalty when when a sample is misclassified or within the margin boundary, where the team proportional to *C* plays the role of a **regulator**

> SVM are an example of a **constrained optimisation problem,** where one tries to minimise a quadratic function subject to **linear constraints.** They are most efficiently solved by means of the Lagrange multiplier method

Support Vector Machines

https://www.youtube.com/watch?v=5zRmhOUjjGY