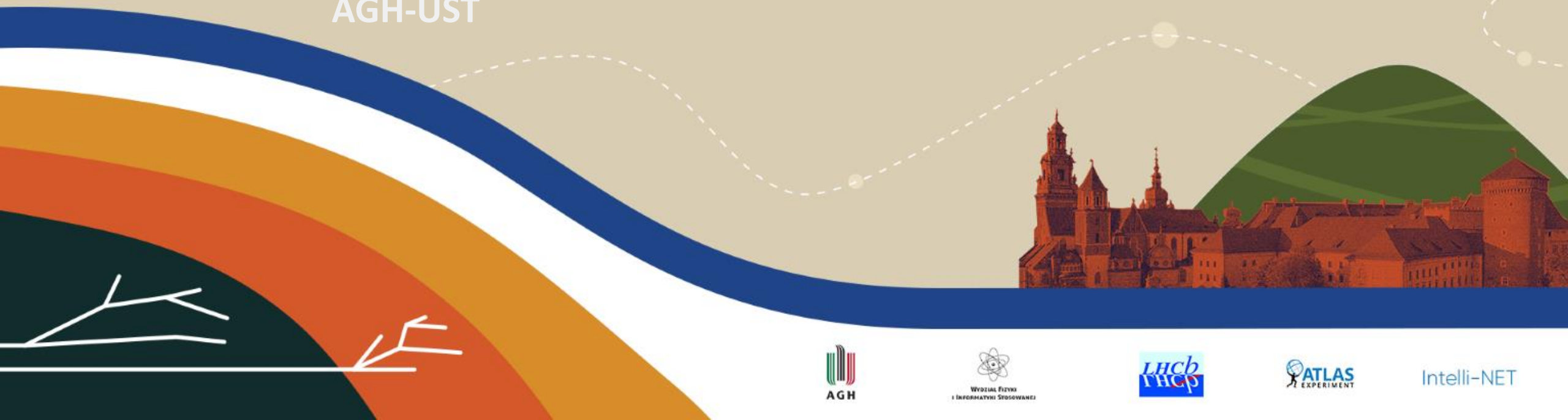Krakow Applied Physics and Computer Science
ONLINE
Summer School '21
July 1 - 28 2021

Tomasz Szumlak
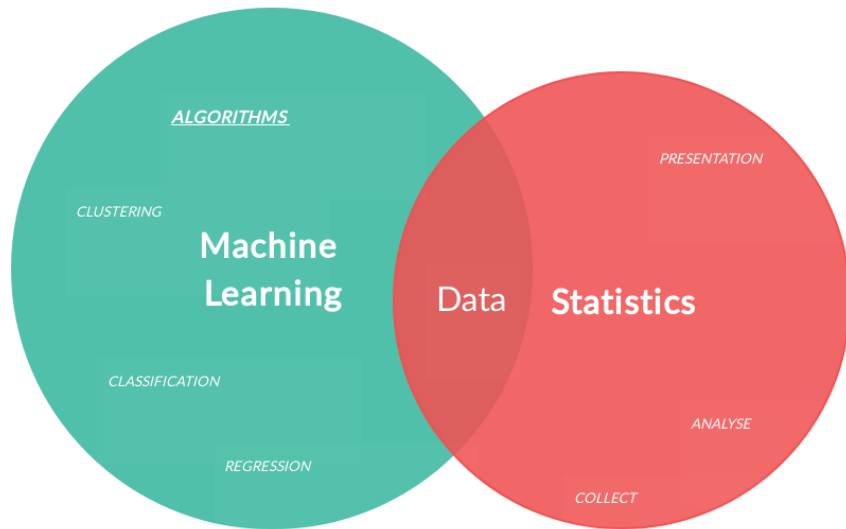
AGH-UST

Machine Learning
Introduction

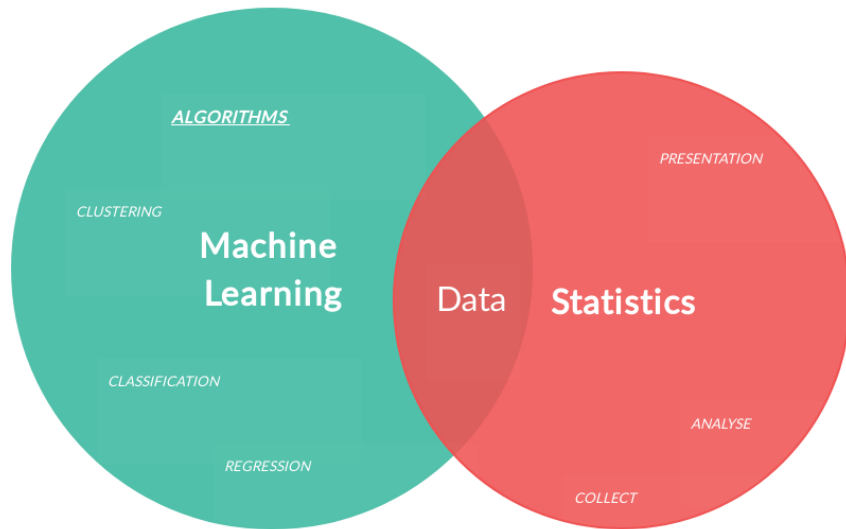AGH · Wydział Fizyki i Informatyki Stosowanej · LHCb · ATLAS EXPERIMENT · Intelli-NET
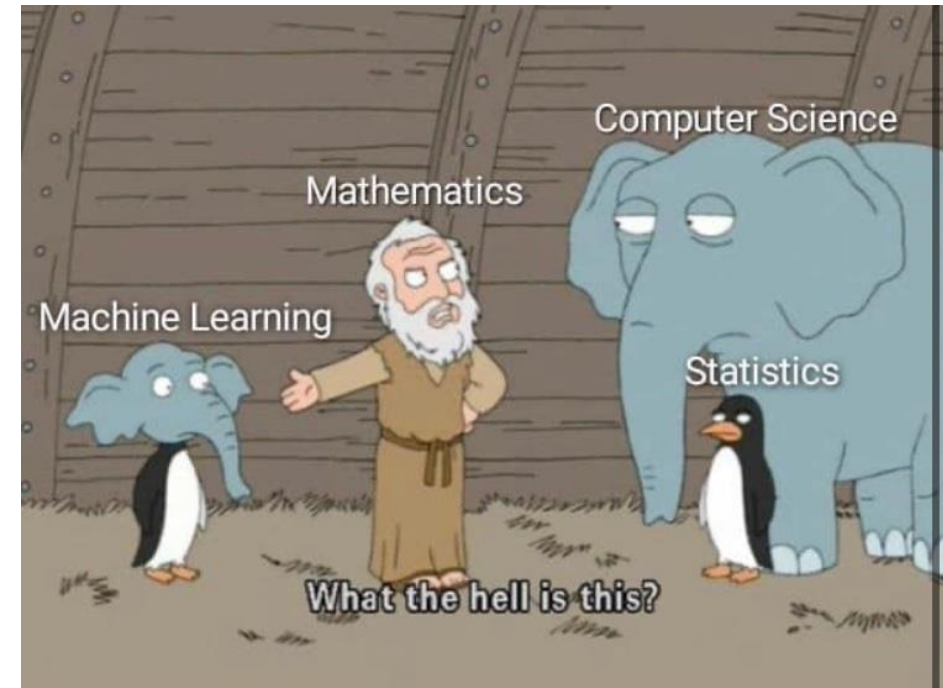
A quick overwiev – ML is a large beast!

Can start on a serious note…

A quick overwiev – ML is a large beast!

Can start on a
serious note…

…or not so serious

A quick overwiev

- ❏ The first part will shortly introduce the key concepts behind ML

- ❏ We start slow, but I will try to inject advanced topics into the course

- ❏ The point is to understand the innerworkings as much as possible, then if you wish you can jump into PyTorch, Keras, Tensorflow, … what-not

- ❏ The second part will be more practical, we discuss a few fundamental applications such as ANN, PCA, GAN and TMVA framework

- ❏ I will provide the ready-to-use code – have a lot of fun with it!

Setting the scene (I)

❑ We are living in interesting times – data come in **abundance** and ability to **process** them and **gain knowledge** is of great value: data is **very precious resource** (like iron, gold or water)

❑ We want to process the data fast and in a robust way

❑ Cutting out all the technobabble and buzzwords we can say that Machine Learning (ML), which is a part of data mining business, allows us to use **computer algorithms** to **make sense of data** or to turn them into knowledge

❑ What is more exciting we have a lot of **open source** libraries that implements the most sophisticated algorithms on the market and **they are free**!

❑ In this lecture we cover: the generic concepts of ML, terminology (knowing the vocabulary is always good!) and different types of learning, architecture of ML system and finally some information on Python toys used for ML
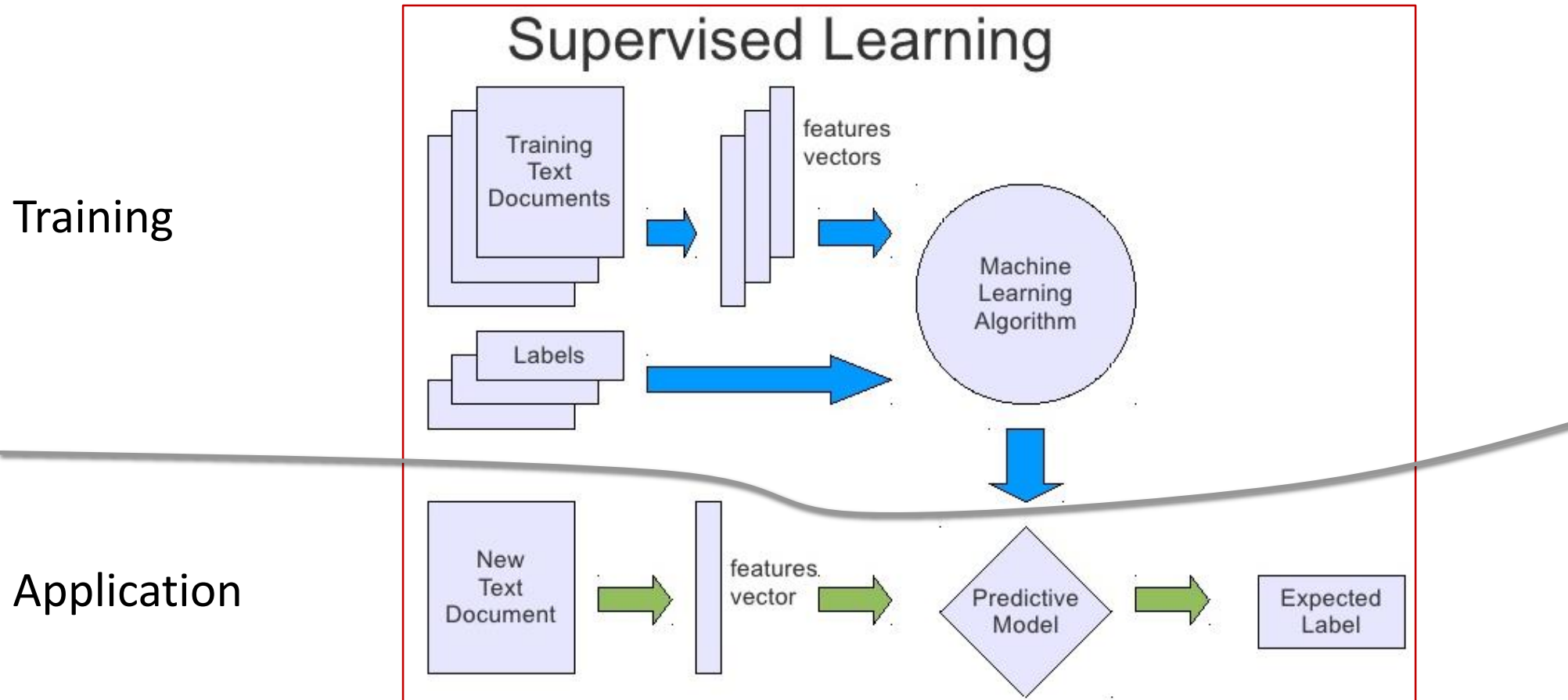
Setting the scene (II)

❑ Large (structured and un-structured) data set are produced in many fields

    ❑ Scientific experiments

    ❑ Social media

    ❑ Data bases (customers, patients, ect.)

❑ ML evolved as a part of **artificial intelligence** – self-learning algorithms to gain knowledge from data and able to make predictions based on that knowledge

❑ Decisions made by trained algorithms are **data driven** – can evolve in time and improve

❑ Significant impact on our everyday life: spam filters, high quality Web search engines, efficient text and speech recognition, games, banking, self-driving cars, cameras (e.g. face recognition), you name it!

Types of machine learning

☐ In general we have three types of learning:

   ☐ Supervised learning

   ☐ Unsupervised learning

   ☐ Reinforcement learning

☐ The **supervised learning** in a nutshell – our s/w learns a model using **labelled training data** set, this will allow it to make a **prediction** regarding **new unseen data** in the future

☐ By supervised learning in this context we mean **using the training samples** for which the required output signals (labels) **are already known**

☐ For instance we want to train a **spam killer algorithm** – for this we can use a number of spam e-mails and a number of genuine messages

Supervised learning

Training

Application

Supervised Learning

Training Text Documents → features vectors → Machine Learning Algorithm

Labels →

New Text Document → features vector → Predictive Model → Expected Label

Supervised learning

☐ In case of a classification task we want to have a system that can **predict** the **categorical class labels** of new instances (new data samples) **using past experiences** (training data)

☐ These **discrete labels**, **without any particular ordering**, can be understood as the **group membership tags** of the new data samples (instances)

☐ For instance, the e-mail spam killer is an example of a **binary classification task**, since we have two possible classes: genuine messages and spam e-mails

☐ We can also have non-binary sets of class labels – text recognition will have as many classes as letters

☐ Here an interesting problem can be encountered: say, we train an algorithm to recognise letters – **what is going to happen if we pass a number to it**?

Binary classification

❑ The idea of a binary classification can be understood using the following example: say, we have given 30 training samples – half of them is **negative** (noise) and half positive (signal)
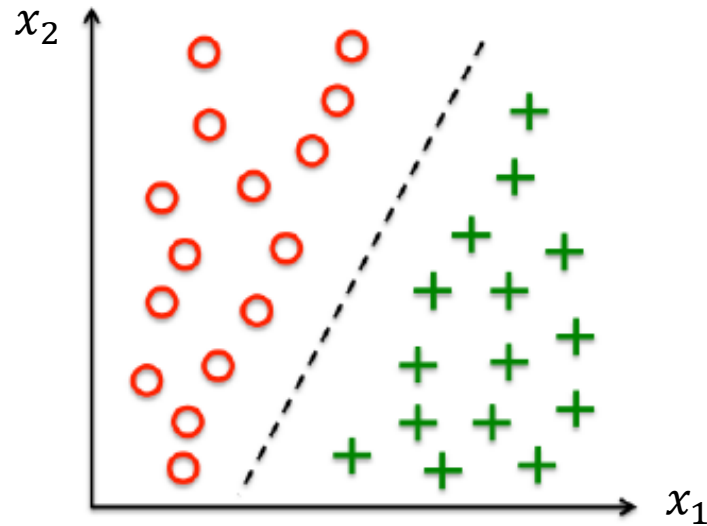
❑ 2D data set – each data instance has two values (x_1, x_2) associated with it

❑ Using them separately is going to yield poor results!

❑ Try to imagine we project the data on the respective axes

❑ Our algorithm must learn a rule to separate these two classes and classify a new instance into one of these classes given values x_1, x_2

❑ This rule is also called **decision boundary** (black dashed line)

Regression

☐ Regression is a second type of supervised learning and is used to predict **continuous outcomes**

☐ Here, we are given a number of **predictor variables** (explanatory variables) and a **continuous response variable** (outcome)

☐ The goal is to find a **model** that **describes a relationship** between **predictors and outcome** – having that we could predict an outcome for any future data

☐ For instance imagine the following: a University Principal asked if it is possible to predict the outcome of Physics final exams based on previous experience?

    ☐ Let's assume that there is a relationship between the time spent studying for the final exams and previous partial scores and the final score

    ☐ We could use this „training data" to teach the model and make predictions regarding the scores for new students

Regression

- The term regression was devised by Francis Galton in 1886 as a conclusion of his studies on height variance in a population
- The plot below shows an idea of **linear regression** with predictor x and response y
- The model can be described now by the intercept and slope (gradient) – can be used to predict the outcome of new data
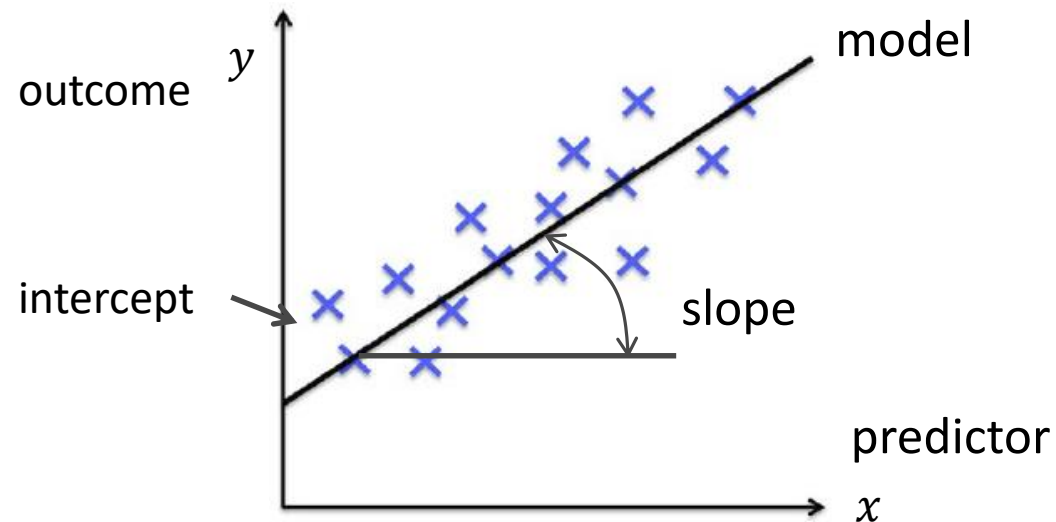
Reinforcement learning

☐ Can be considered as a type of supervised learning, but the supervision here is much more subtle (we do not use labels)

☐ In case of reinforcement learning we **train an agent** that evolves by **interacting** with the **environment**

☐ An agent can asses its actions using a reward function

☐ Using an exploratory trial-and-error method (or planning) an agent learns a series of actions that maximises the **reward function**

☐ For instance: a chess engine – an agent observes the board (the environment), the reward is win/lose of the game
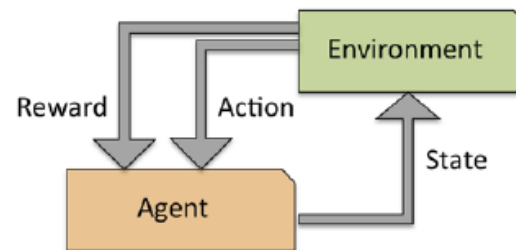
Unsupervised learning

❑ In both supervised and reinforcement learning we kind of know the right answer beforehand (via labels or reward function)

❑ In unsupervised learning we **do not know** the **structure of data** (unlabelled data) – we can extract knowledge without guidance of a known outcome or reward function

❑ Main exploratory technique of data analysis is **clustering**, which makes possible to organise the data into meaningful sub-groups (clusters) **without any prior** information of their membership

  ❑ Each set of objects that share a **certain degree of similarity** can be assign to a cluster

  ❑ Objects assigned to a given cluster are, in turn, **more dissimilar** to objects in other groups (clusters)

  ❑ Clustering is often called **unsupervised classification**

  ❑ The best example is creating distinct marketing programs

Unsupervised learning

❑ The idea of unsupervised classification can be described by a simple sketch as below:
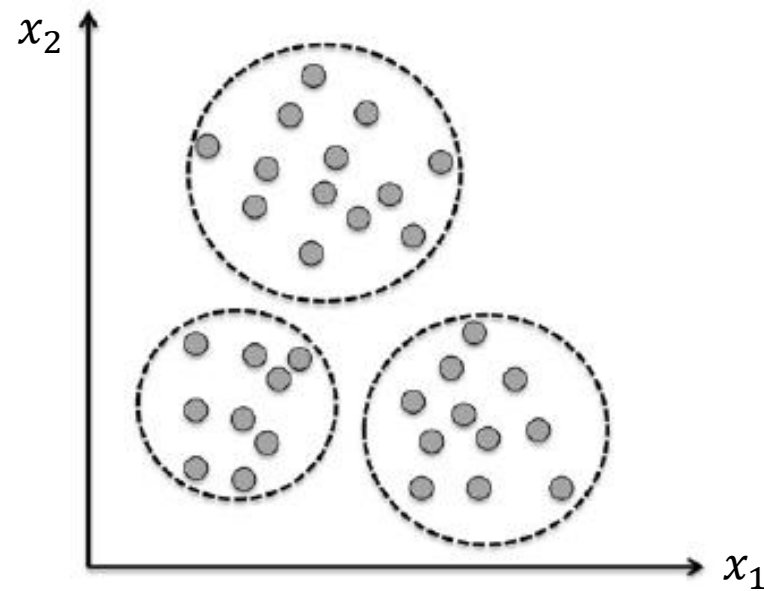
❑ In case of clusterisation we can encounter problem of dissimilarities within groups and between groups

❑ Not a trivial matter!

Terminology

☐ Will not reinvent the wheel and use one of the most popular example with flower iris (see for example: Yu Yang „A study of pattern recognition of Iris flower based on Machine Learning")

☐ Imagine, each flower sample can be stored as rows and each variable is stored as column (i.e., matrix notation is helpful!) in feature matrix $\mathcal{F}$

$$\mathcal{F} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{pmatrix}$$

Terminology
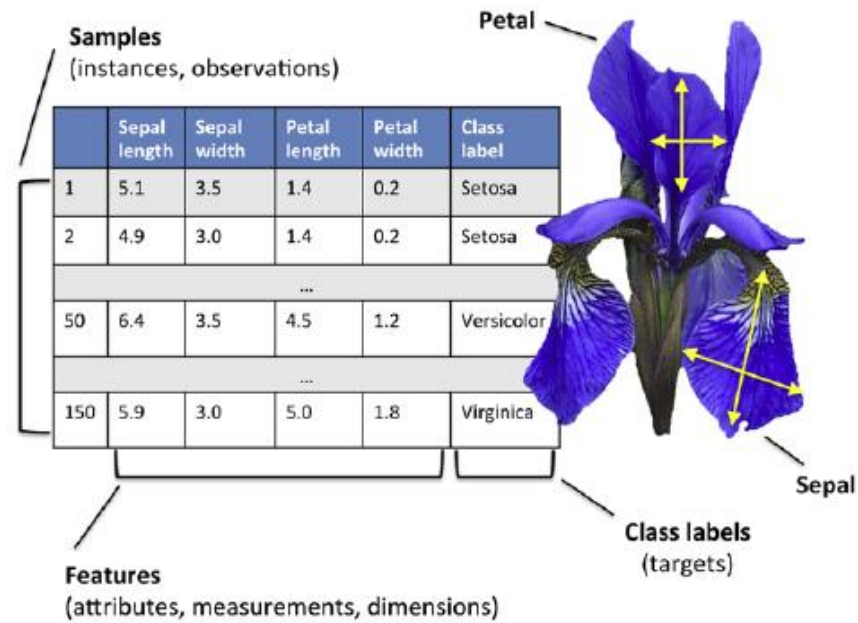
$$x^{(i)} = \left(x_1^{(i)}, \ldots, x_n^{(i)} \right) \text{ - one instance}$$

$$x_j = \begin{pmatrix} x_j^{(1)} \\ . \\ . \\ . \\ x_j^{(m)} \end{pmatrix}$$

A deeper look…

❑ The whole story began in **1943** with McCullock-Pitts **neuron model**

❑ They described a **nerve cell** using a simple **binary logic gate** with binary output(s) – such **perceptron** (or artificial neuron) accepts multiple input signals, **integrates** (combines) them and if signal exceeds a certain threshold the perceptron is able to produce an **output signal**

  ❑ Using the biological language, the signal arrives via **dendrites**, is then processed by a **cell body** and the output is **propagated via axon**

  ❑ Axon can have **many terminals** connecting the perceptron with others

❑ Next in 1957 **Rosenblatt** came with an brilliant idea on how to efficiently **train** such perceptrons

❑ With this learning rule it is possible to determine automatically the best weights that decide if the perceptron fires or not
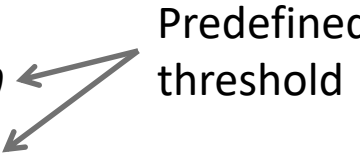
Perceptron model

inputs

weights

1

x_1

x_2

x_3

x_4

w_1

w_2

w_3

w_4

weighted sum

unit step function

Σ
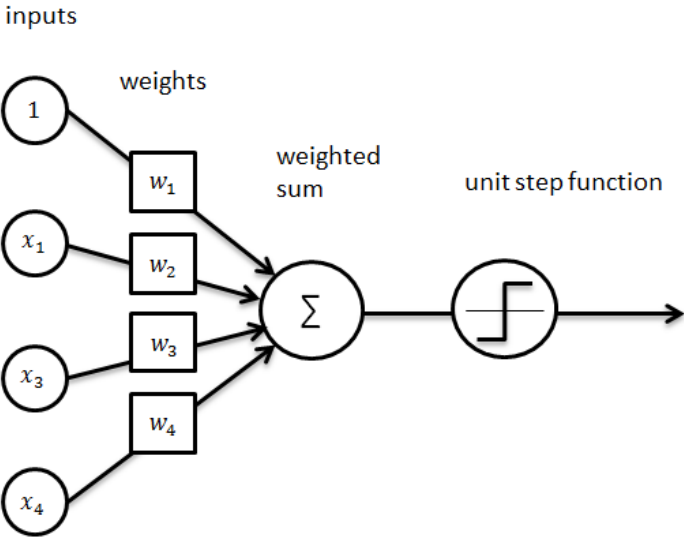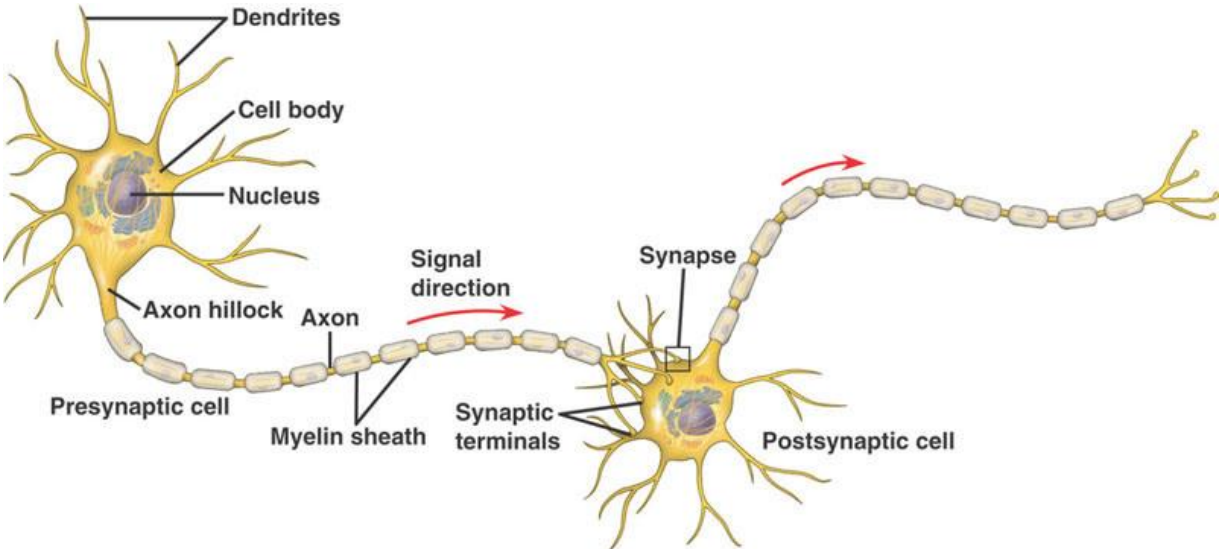
Rosenblatt approach

❑ With Rosenblatt method we use the supervised learning to train a set of weights

❑ They are then multiplied with the input data (features) and based on the result a decision is made: fire/not fire

❑ Trained perceptron algorithm can be subsequently used to make a decision regarding a sample membership (classification)

❑ In order to provide a bit more formal description, let's assume that we are dealing with a binary classification task with two classes, we then call one a **positive** $(+1)$ and the other a **negative** (-1)

❑ In order to define our perceptron behaviour we need to define an **activation function**: $\phi(z)$

Predefined threshold

$$\phi(z) = \begin{cases} +1 \; if \; z \geq \; \theta \\ -1 \; if \; z < \theta \end{cases}$$

Some math... (I)

☐ The activation function, defined in the previous slide as the Heaviside **step function** (NOTE! **This is not a unique choice**) takes a linear combination of given input values and a weight vector

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}, \vec{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_k^{(i)} \end{bmatrix}$$

☐ Using the two vectors we can calculate the **net input** z:

$$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_k x_k^{(i)} = \sum_{j=1}^{j=k} w_j x_j^{(i)} = \vec{w}^T \vec{x}^{(i)}$$

☐ Usually, we also move the threshold to the left side to facilitate the notation:

$$z^{(i)} = w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_k x_k^{(i)}, w_0 = \theta, x_0^{(i)} = 1$$

Some math... (II)

☐ Here, a critical part is to understand two things:

   ☐ How the **input information** (which may have many components) is **translated** into a binary information $(+1/-1)$?

   ☐ How it is used to make the classification?

☐ Note, that the formulas we wrote in the last slide are identical to the vector dot product – we reduce the space dimension!
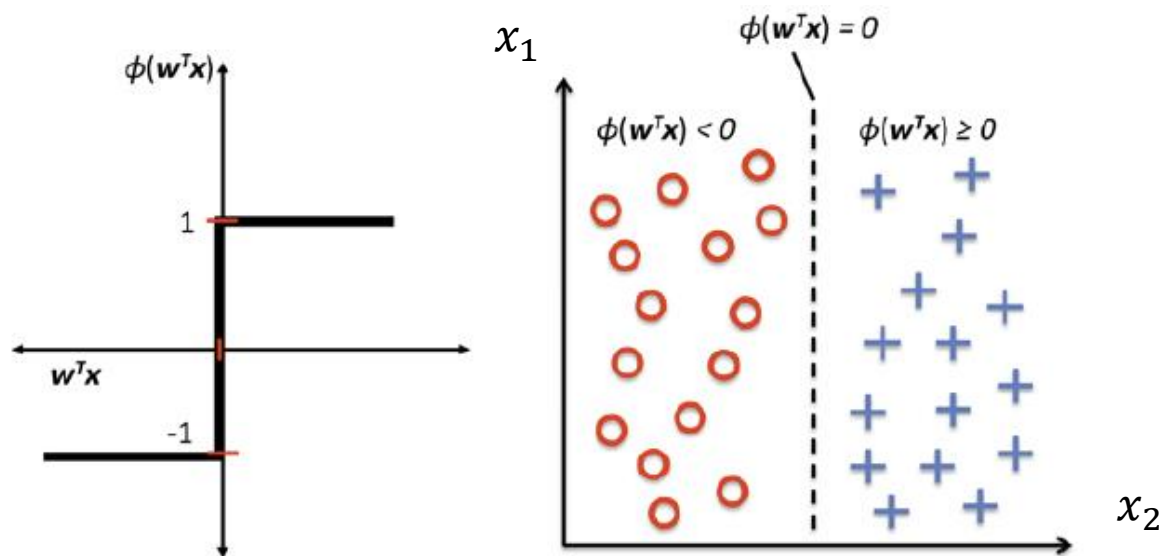
The algorithm (I)

☐ First we reduce the input data and mimic the behaviour of a single neuron (just like in the brain): fire/not fire

☐ The perceptron algorithm, then goes like that:

   ☐ **Initialise the weights vector to 0 or „something small"**

   ☐ **For each training data sample $\vec{x}^{(i)}$ do:**

      ☐ **Get the output value (class label) $\widetilde{y}^{(i)}$, using the unit step function**

      ☐ **Update the weights accordingly (update concerns all the weights in one go)**

☐ We can write

$$w_j = w_j + \Delta w_j$$

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \widetilde{y}^{(i)} \right) \cdot x_j^{(i)}$$

☐ The second formula is called **perceptron learning rule**, and the η is called the learning rate (just a number between 0 and 1)

The algorithm (II)

☐ To update the weight vector we need to know the true class label $y^{(i)}$ and calculate the predicted one: $\tilde{y}^{(i)}$

☐ All the weights are updated at once, we do not re-compute the class label before all Δw_j are updated

☐ Looking at our 2D example problem (see slide 7) we would write:

$$\Delta w_0 = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right)$$

$$\Delta w_1 = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_1^{(i)}$$

$$\Delta w_2 = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_2^{(i)}$$

☐ Before we devour our keyboards in order to implement the perceptron experiment, let's make the following „gedanken" experiment to get the better feeling about it

The algorithm (III)

❑So, let's check first what happens to the weights if we predict the class label correctly/incorrectly:

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_j^{(i)} = \eta \cdot \left(-1^{(i)} - \left(-1^{(i)}\right)\right) \cdot x_j^{(i)} = 0$$

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_j^{(i)} = \eta \cdot \left(+1^{(i)} - \left(+1^{(i)}\right)\right) \cdot x_j^{(i)} = 0$$

❑If the prediction is wrong, however, we have:

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_j^{(i)} = \eta \cdot \left(1^{(i)} - \left(-1^{(i)}\right)\right) \cdot x_j^{(i)} = 2 \cdot \eta \cdot x_j^{(i)}$$

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_j^{(i)} = \eta \cdot \left(-1^{(i)} - \left(1^{(i)}\right)\right) \cdot x_j^{(i)} = -2 \cdot \eta \cdot x_j^{(i)}$$

❑So, the algorithm will try to push weights toward the value of the target class label

❑And what about the value of the sample components?

The algorithm (IV)

❑ Consider the following:

$$y^{(i)} = +1, \tilde{y}^{(i)} = -1, \eta = 1$$

❑ So, the true label is $+1$, however we predicted -1, with the fixed learning rate it turns out that the weight update is proportional to the value of the variable

$$\Delta w_j = \eta \cdot \left(1^{(i)} - \left(-1^{(i)} \right) \right) \cdot x_j^{(i)} = 1 \cdot 2 \cdot x_j^{(i)}$$

❑ Next time we encounter such event the weight will be more positive (or negative, depending on the value of $x_j^{(i)}$)

❑ The convergence of the perceptron algorithm is only possible when the two **classes are linearly separable** (and with a small learning rate)

❑ If the above is not true we need to make a decision on a maximum **number of epochs** (or how many times we go over a training data) and a **maximum number of acceptable errors** in classification
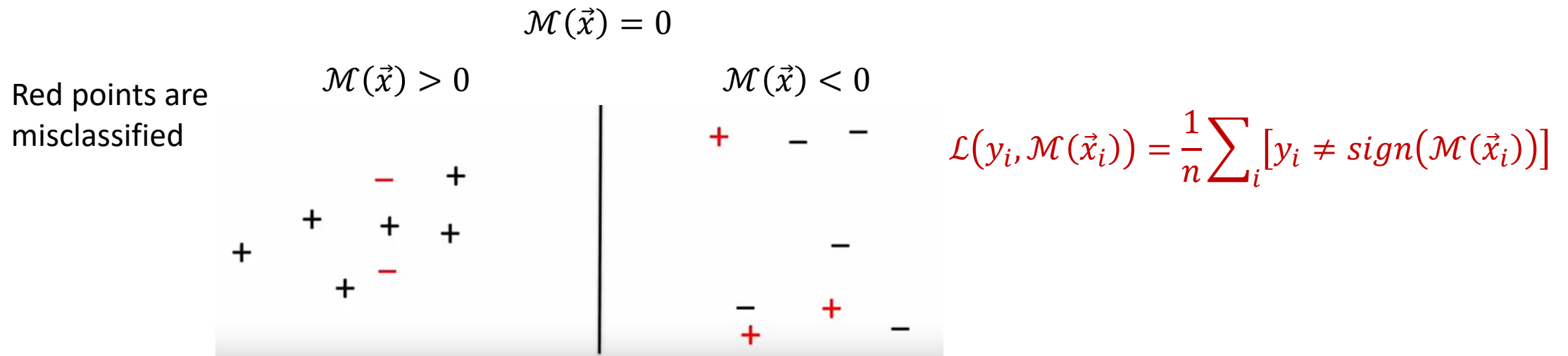
The overview

Linearly separable

Not linearly separable

Not linearly separable
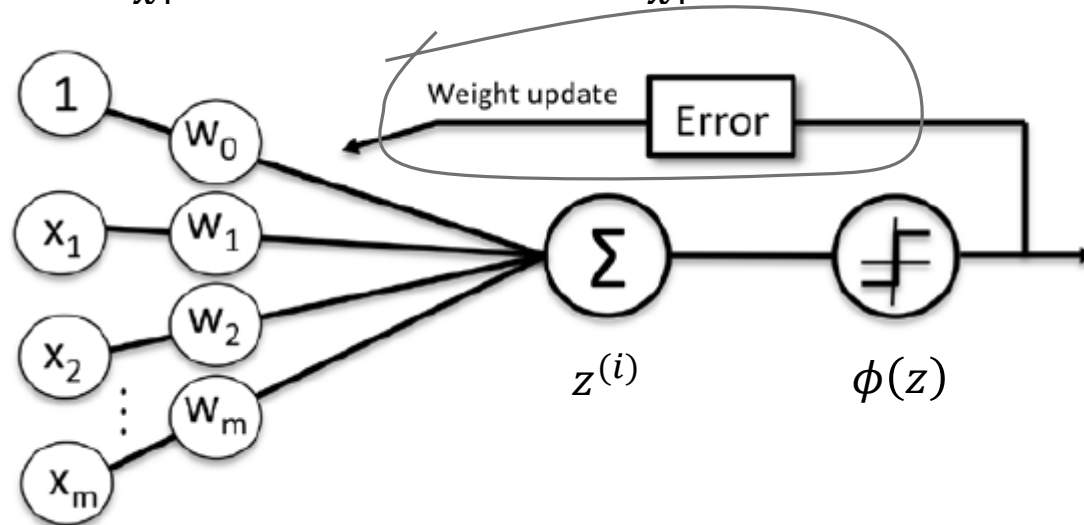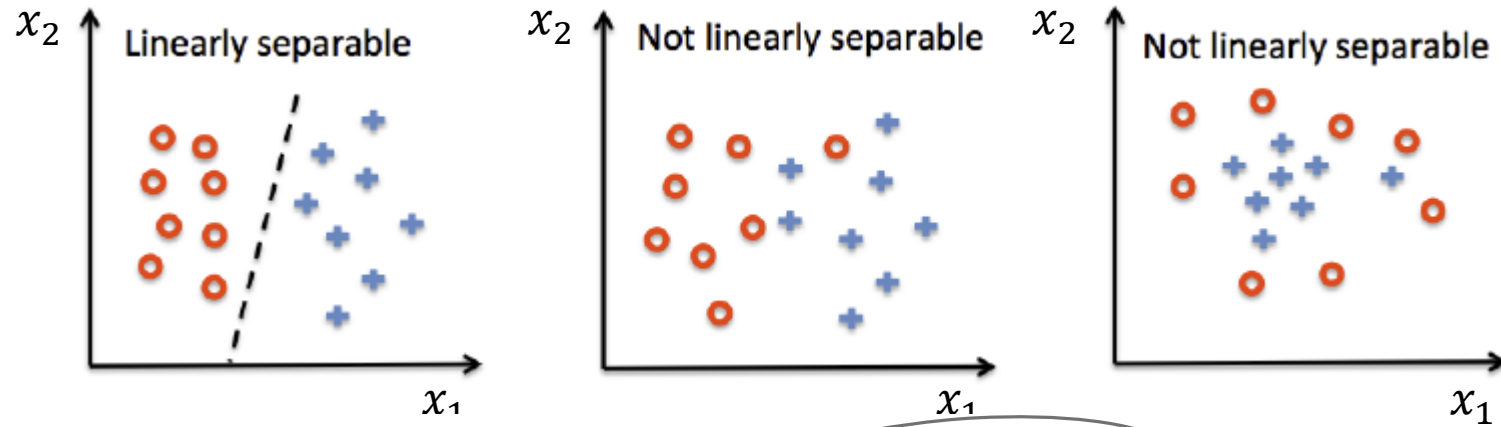
Make some compromise

Loss function (I)

❑ In practice we need to have a very good handle on the performance of our model

❑ Or, in other words we need to have means to penalise the model if it performs poorly and reward if it does good

❑ We could, for instance, just count the number of good and bad decisions and calculate the rates. Imagine the situation below

$$\mathcal{M}(\vec{x}) = 0$$

$$\mathcal{M}(\vec{x}) > 0 \qquad\qquad \mathcal{M}(\vec{x}) < 0$$

Red points are misclassified

$$\mathcal{L}(y_i, \mathcal{M}(\vec{x}_i)) = \frac{1}{n}\sum_i \left[y_i \neq sign(\mathcal{M}(\vec{x}_i))\right]$$
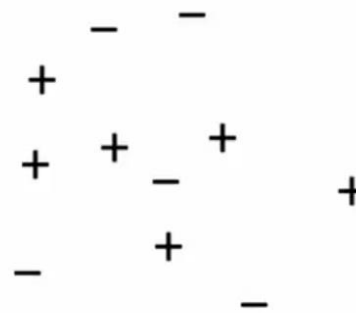
Loss function (II)

☐ Let's create „an universal" formula for the loss function, for that imagine we moved all the correctly classified points on one side and the misclassified on the other (note, we are changing the meaning of the plot from the last slide!

The opposite signs $\quad y \cdot \mathcal{M}(\vec{x}) < 0 \qquad y \cdot \mathcal{M}(\vec{x}) > 0 \qquad$ The same signs
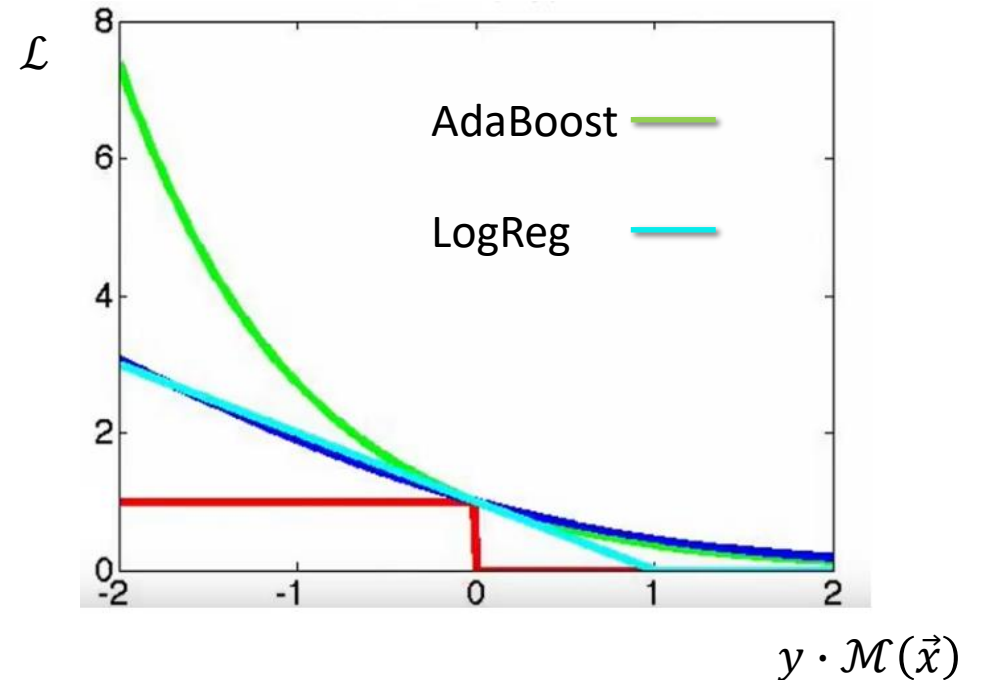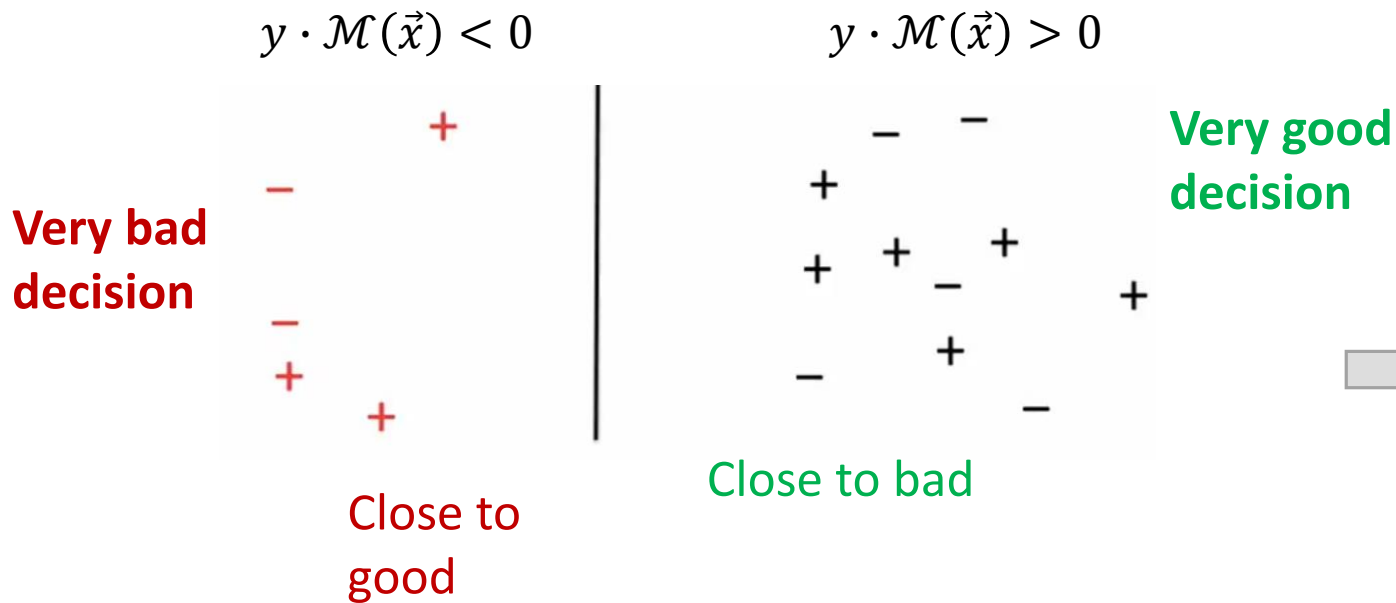
$$\mathcal{L} = \frac{1}{n} \sum_i 1_{[y \cdot \mathcal{M}(\vec{x}_i) < 0]}$$

Max penalty each time!

$y \cdot \mathcal{M}(\vec{x})$

Loss function (III)

☐ In theory such loss function is very powerfull, but in practice we cannot optimise such expression in any easy way and on top of this it has so sensitivity on how bad the decision was, i.e., each time the penalty is maximal

$$y \cdot \mathcal{M}(\vec{x}) < 0 \qquad\qquad y \cdot \mathcal{M}(\vec{x}) > 0$$

Very bad decision

Very good decision

Close to good

Close to bad

Loss function (IV)

❑ There are some tantalising facts regarding the loss function: the whole training process depends on the way we measure its performance – more aggressive approach may be more beneficial, it may determine how long the training process take and if it will be successful at all – **how interesting**

❑ Different loss functions determines upper limits w.r.t $\color{red}{1_{[y \cdot \mathcal{M}(\vec{x}_i) < 0]}}$ one:

$$\mathcal{L}(y_i, \mathcal{M}(\vec{x}_i)) = \frac{1}{n}\sum_i [y_i \neq sign(\mathcal{M}(\vec{x}_i))] = \frac{1}{n}\sum_i 1_{[y \cdot \mathcal{M}(\vec{x}_i) < 0]} \leq \frac{1}{n}\sum_i f_{\mathcal{M}}(y \cdot \mathcal{M}(\vec{x}_i))$$

❑ Our main target then should be: