eecs4404/5327
Introduction to Machine Learning and Pattern Recognition

Lecture 2 - Classification

Amir Ashouri

September 11, 2019

Recap of last class

Reading: UML Chapter 1: 1.1,1.2,1.3

Recap

Lesson 1:
Machine Learning can only work if "there is something we can learn" Patterns must exist!
(Learning can not overcome inherent randomness in data generation).

Lesson 2:
We need to have an idea what we are looking for
($\rightarrow$ **inductive bias** or **prior knowledge**)

Lesson 3:
We need to have data. No data, no learning!

Predicting tastiness of Papayas

Task:
Use machine learning to generate a program that predicts whether a **papaya** is **tasty** or **not tasty**.

Predicting tastiness of Papayas

Step 1 Choose feature representation

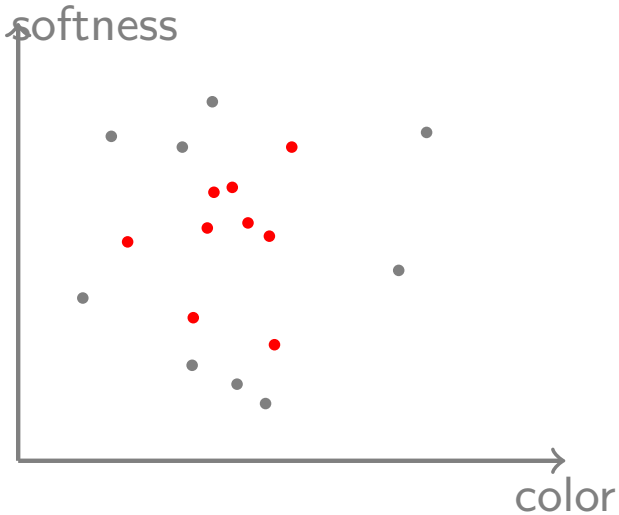
Step 2 Choose class of predictors
Class H of hypotheses
$h : X \to Y$

Step 3 Collect data
$(x_1, y_1), \ldots (x_n, y_n)$

Step 4 Fit a model
Choose a predictor h from the class that has minimal empirical error
This is called *Empirical Risk Minimization*

Step 5 Predict
$y = h(x)$

Feature space $X \subseteq \mathbb{R}^d$
Label space $Y = \{-1, 1\}$



Predicting tastiness of Papayas

Step 1 Choose feature representation

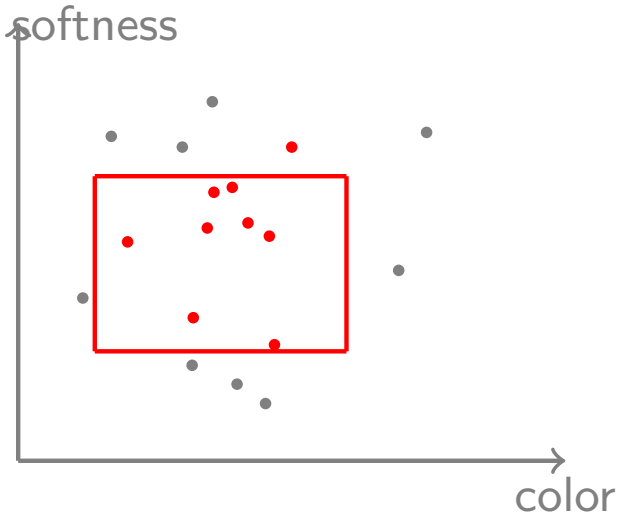
Step 2 Choose class of predictors
Class H of hypotheses
$h : X \to Y$

Step 3 Collect data
$(x_1, y_1), \ldots (x_n, y_n)$

Step 4 Fit a model
Choose a predictor h from the class that has minimal empirical error
This is called *Empirical Risk Minimization*

Step 5 Predict
$y = h(x)$

Feature space $X \subseteq \mathbb{R}^d$
Label space $Y = \{-1, 1\}$



1. **Supervised** Learning
2. **Unsupervised** Learning
3. **Reinforcement** Learning

Supervised Learning

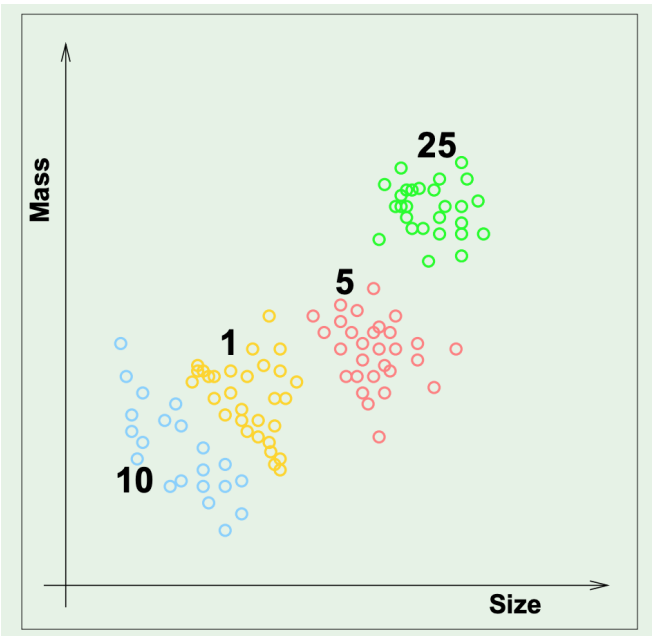
Training data comprises examples of the input vectors $x_1, \ldots, x_n \in \underline{X}$ along with their corresponding target label $\mathcal{Y}$
Examples:

- Medical Diagnosis $\underline{X}$ = list of symptoms $\mathcal{Y} \in \{-1, +1\}$
- Hand-written digits $\underline{X}$ = image of digits $\mathcal{Y} \in \{0, 1, 2, \ldots, 9\}$
- Classifying coins $\underline{X}$ = (size, mass) of coins
  $\mathcal{Y} \in \{5c, 10c, 25c, 50c\}$

Supervised Learning

Training data comprises examples of the input vectors $x_1, \ldots, x_n \in \underline{X}$ along with their corresponding target label $\mathcal{Y}$
Examples:

- Medical Diagnosis $\underline{X}$ = list of symptoms $\mathcal{Y} \in \{-1, +1\}$
- Hand-written digits $\underline{X}$ = image of digits $\mathcal{Y} \in \{0, 1, 2, \ldots, 9\}$
- Classifying coins $\underline{X}$ = (size, mass) of coins
  $\mathcal{Y} \in \{5c, 10c, 25c, 50c\}$



Unsupervised Learning

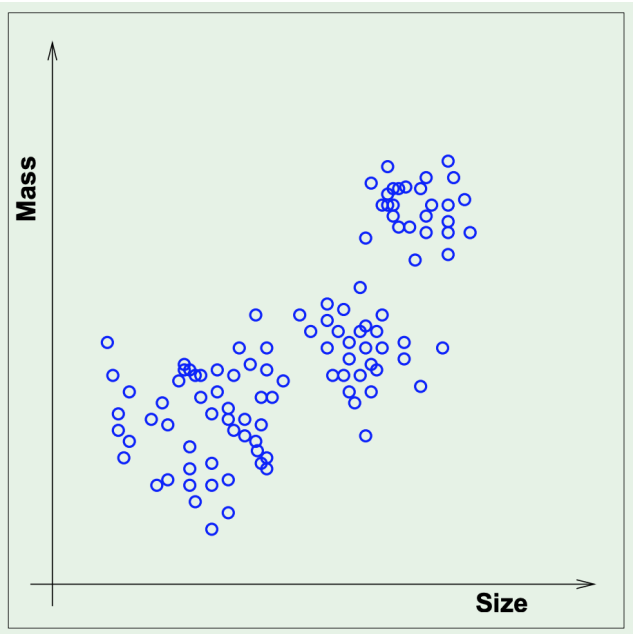
What if we didn't have the labels of our data? Training data comprises examples of the ONLY input vectors $x_1, \ldots, x_n \in \underline{X}$
Examples:

- Reducing dimension of a data set
  $\underline{X} = \{(x_1, y_1), \ldots, (x_{20}, y_{100}\}$ to $\underline{X'} = \{(x_1, y_1, \ldots, \{(x_{20}, y_5)\}$
- Clustering coins $\underline{X}$ using only their size and mass

Unsupervised Learning

What if we didn't have the labels of our data? Training data comprises examples of the ONLY input vectors $x_1, \ldots, x_n \in \underline{X}$
Examples:

- Reducing dimension of a data set
  $\underline{X} = \{(x_1, y_1), \ldots, (x_{20}, y_{100}\}$ to $\underline{X'} = \{(x_1, y_1, \ldots, \{(x_{20}, y_5)\}$
- Clustering coins $\underline{X}$ using only their size and mass



Reinforcement Learning

We have the input training data and *some sort of grading of the data* and yet no label.
Example:

- An agent learns to play a chess game
- A robot Learning to walk

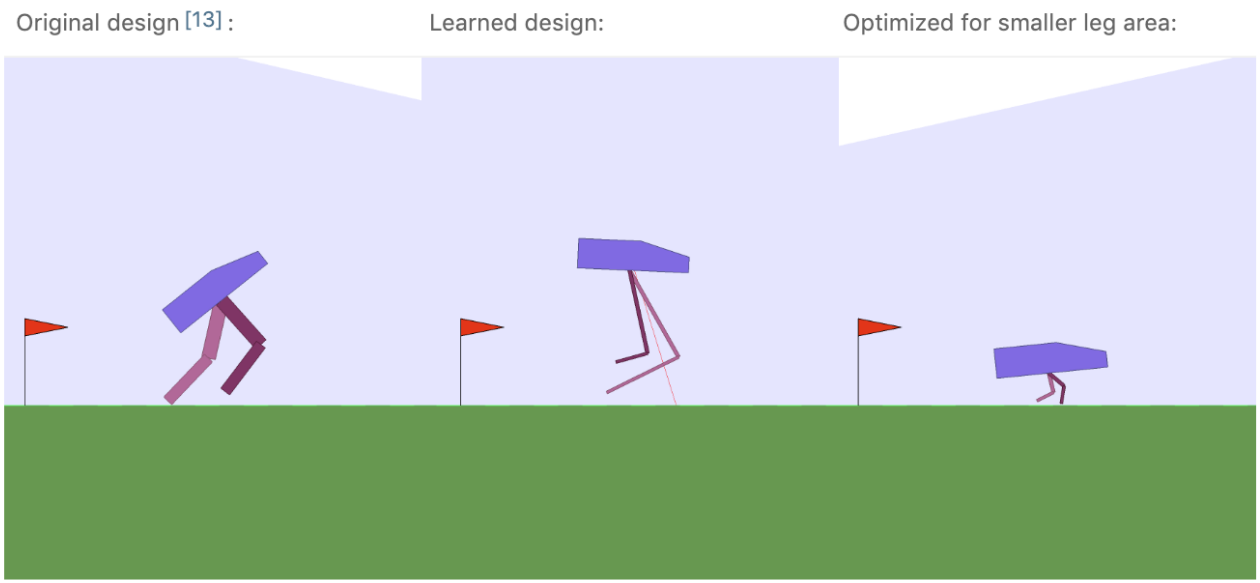


Figure: Source: https://designrl.github.io

Classification

Reading: PRML, Intro to Chapter 4

Classification

We call a prediction problem a classification task, if the output space is finite (as opposed to real-valued outputs as in regression). That is, we have

Input/feature space: $\mathcal{X} \subseteq \mathbb{R}^d$.
Output/label space: $\mathcal{Y} = \{1, 2, \ldots, k\} = [k]$

$\rightarrow$ binary classification if two possible outputs:

Output/label space: $\mathcal{Y} = \{+1, -1\}$ or $\mathcal{Y} = \{0, 1\}$

Binary Classification

We want to predict with a (D-dimensional) linear function:

$$y_{\mathbf{w}}(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) = b + w_1 x_1 + w_2 x_2 + \ldots w_D x_D$$

Binary Classification

We want to predict with a (D-dimensional) linear function:

$$y_{\mathbf{w}}(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) = b + w_1 x_1 + w_2 x_2 + \dots w_D x_D$$

$$y_{\mathbf{w}}(\mathbf{x}) = h(x) = \mathbf{sign}(\sum_{\mathbf{i=1}}^{\mathbf{D}} \mathbf{w_i x_i} - \mathbf{b})$$

- the w_i are called "weights"
- b is called "bias" or "offset" or "threshold"
  (nothing to do with statistical bias)

Classification tasks examples

Examples of classification tasks are:

- Email spam detection $\mathcal{Y} = \{+1, -1\} = \{\text{spam}, \text{not spam}\}$
- Medical diagnosis, eg $\mathcal{Y} = \{+1, -1\} = \{\text{diabetes}, \text{healthy}\}$
- Papaya example $\mathcal{Y} = \{+1, -1\} = \{\text{tasty}, \text{not tasty}\}$
- Image classification
  $\mathcal{Y} = \{1, 2, \ldots, k\} = \{\text{cat}, \text{dog}, \text{zebra}, \ldots, \text{tiger}\}$
- Text classification
  $\mathcal{Y} = \{1, 2, \ldots, k\} = \{\text{sports}, \text{news}, \text{arts}, \ldots, \text{science}\}$

Examples of classes of classifiers

A predictor $h : \mathcal{X} \to \mathcal{Y}$ is also called a classifier.

A set H of classifiers is called a hypothesis class.

Examples of hypothesis classes for classification are:

- Decision trees
- Rectangles (as in the papaya example)
- Neural networks (can be used for classification but also for other tasks)
- Linear classifiers

Loss function for classification

We measure the quality of a predictor by means of a loss function

$$\ell(h, (x, y(predicted - label))) = \mathbf{1}[h(x) \neq t(true - label)]$$

Linear classification

Reading: PRML, Intro to Chapter 4

Linear classifiers

The class of linear classifiers is defined as:

$$H = \{y_{\mathbf{w},b} \mid \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

where

$$y_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \in \{+1, -1\}$$

for $\mathbf{x} \in \mathbb{R}^d$.

Linear classifiers

A binary linear classifier $y_{\mathbf{w},b} : \mathbb{R}^d \to \{+1, -1\}$ maps one halfspace of $\mathbb{R}^d$ to $+1$ and its complement to -1. The class of linear classifiers is therefore also called halfspaces or halfspace classifiers. The decision boundary is the hyperplane (subspace of dimension $d-1$, the subspace where $\langle \mathbf{w}, \mathbf{x} \rangle = 0$) defining the two halfspaces.

Getting rid of b...

We can reduce classification with a linear classifier in $\mathbb{R}^d$ to classification with homogeneous linear classifiers in $\mathbb{R}^{d+1}$ via the following identity:

$$y_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \operatorname{sign}(\langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}} \rangle) =: y_{\tilde{\mathbf{w}}}(\tilde{\mathbf{x}})$$

where $\tilde{\mathbf{x}} = (1, \mathbf{x}) \in \mathbb{R}^{d+1}$ and $\tilde{\mathbf{w}} = (b, \mathbf{w}) \in \mathbb{R}^{d+1}$.
For binary classification, this is equal to

$$h(x) = \mathbf{sign}(\sum_{\mathbf{i=1}}^{\mathbf{D}} \mathbf{w_i x_i} - \mathbf{b})$$

$$h(x) = \mathbf{sign}(\sum_{\mathbf{i=0}}^{\mathbf{d}} \mathbf{w_i x_i}) = \mathbf{sign}(\mathbf{w}^{\mathsf{T}} \mathbf{x})$$

where w_0 is an artificial coordinate, $x_0 = 1$

Linearly separable data

A dataset $D = ((\mathbf{x}_1, y_1), \ldots (\mathbf{x}_N, y_N))$ with $\mathbf{x}_i \in \mathbb{R}^d$ is called (linearly) separable if there exists a vector $\mathbf{w} \in \mathbb{R}^d$ with

$$h_{\mathbf{w}}(\mathbf{x}_i) = y_i$$

for all $i \in [N]$.

Linearly separable data

A dataset $D = ((\mathbf{x}_1, y_1), \dots (\mathbf{x}_N, y_N))$ with $\mathbf{x}_i \in \mathbb{R}^d$ is called (linearly) separable if there exists a vector $\mathbf{w} \in \mathbb{R}^d$ with

$$h_{\mathbf{w}}(\mathbf{x}_i) = y_i$$

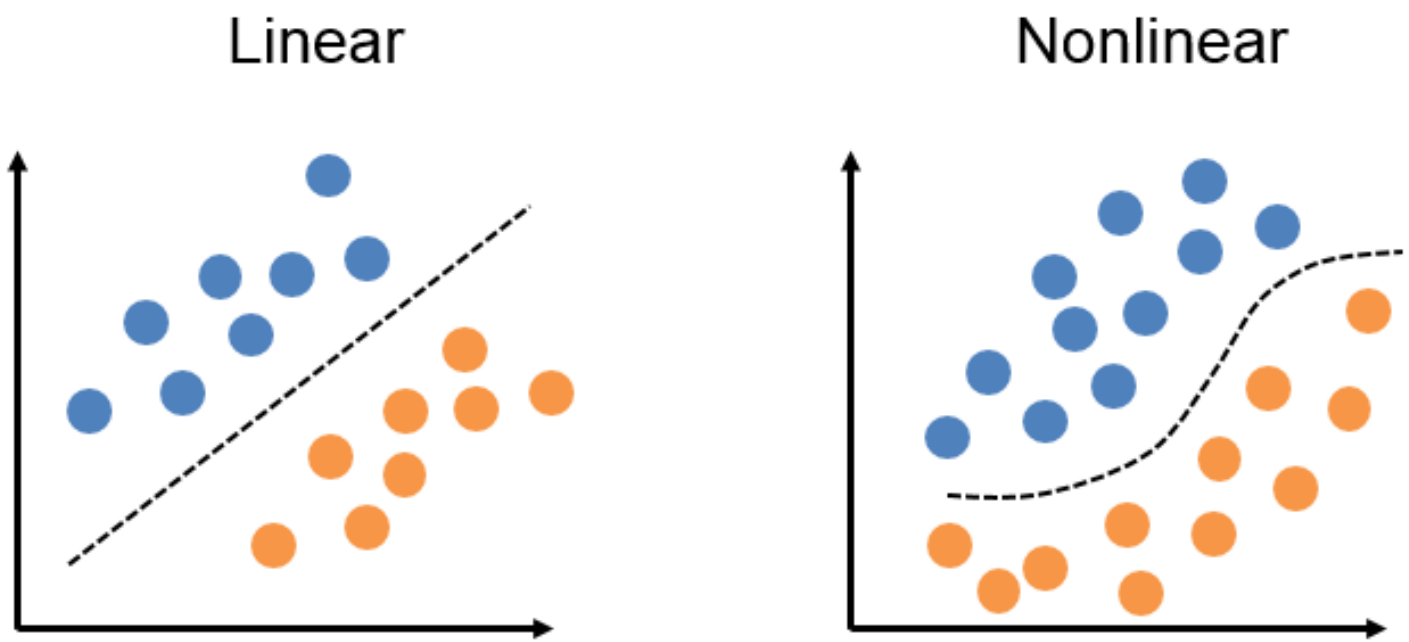
for all $i \in [N]$.

Note that in vector space this is equivalent to

$$y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$$

for all $i \in [N]$.

Linearly separable data



Perceptron Learning Algorithm (PLA)

The perceptron algorithm is a simple, iterative procedure to find a separating hyperplane to a dataset $D = ((\mathbf{x}_1, t_1), \ldots (\mathbf{x}_N, t_N))$ if D is separable.

Perceptron Learning Algorithm (PLA)

The perceptron algorithm is a simple, iterative procedure to find a separating hyperplane to a dataset $D = ((\mathbf{x}_1, t_1), \ldots (\mathbf{x}_N, t_N))$ if D is separable.

That is, the perceptron algorithm implements the Empirical risk minimization (ERM) rule for the class of linear classifiers if the input data is linearly separable.

Perceptron Learning Algorithm (PLA)

The perceptron algorithm is a simple, iterative procedure to find a separating hyperplane to a dataset $D = ((\mathbf{x}_1, t_1), \ldots (\mathbf{x}_N, t_N))$ if D is separable.

That is, the perceptron algorithm implements the Empirical risk minimization (ERM) rule for the class of linear classifiers if the input data is linearly separable. The runtime of the perceptron

algorithm depends on "how separable" the data is: the larger the margin, the faster the algorithm finds a separator.

A Misclassified Point

In a binary linear classification, there are two possibilities:

$$sign(w^T x_i) \neq y_i$$

1. $y_i = +1$ for a $t_i = -1$
2. $y_i = -1$ for a $t_i = +1$

Perceptron Learning Algorithm (PLA)

Input: $D = ((\mathbf{x}_1, t_1), \ldots (\mathbf{x}_N, t_N))$

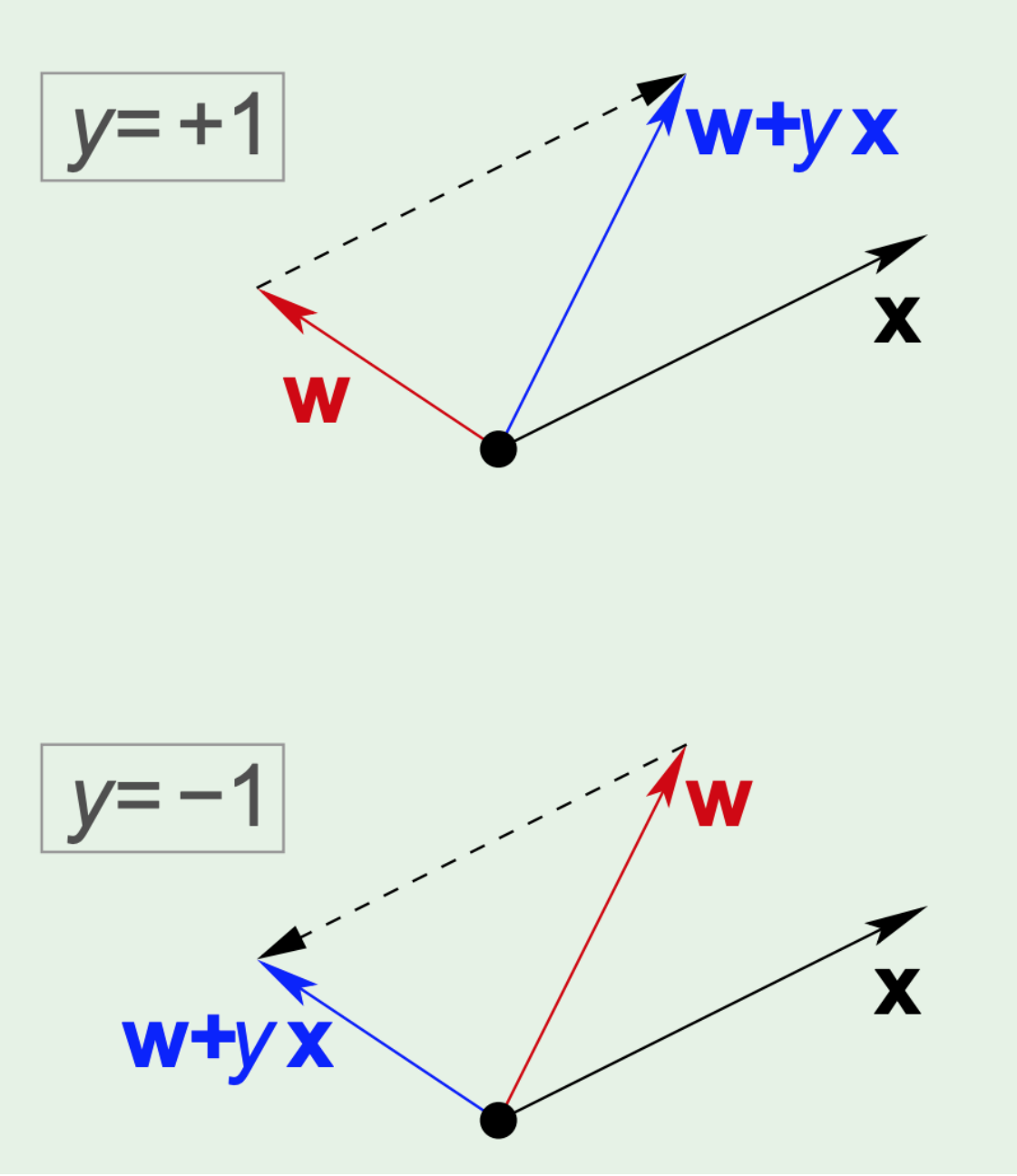
Initialize: $\mathbf{w}^1 = \mathbf{0}$

For $t = 1, 2, \ldots$**:**

> **If** there exits an i with $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \leq 0$ (a misclassified point)
> **then update:** $\mathbf{w}^{y+1} = \mathbf{w}^y + y_i \mathbf{x}_i$

Output: $\mathbf{w}^y$

PLA Update Rule



Perceptron Learning Algorithm (PLA)

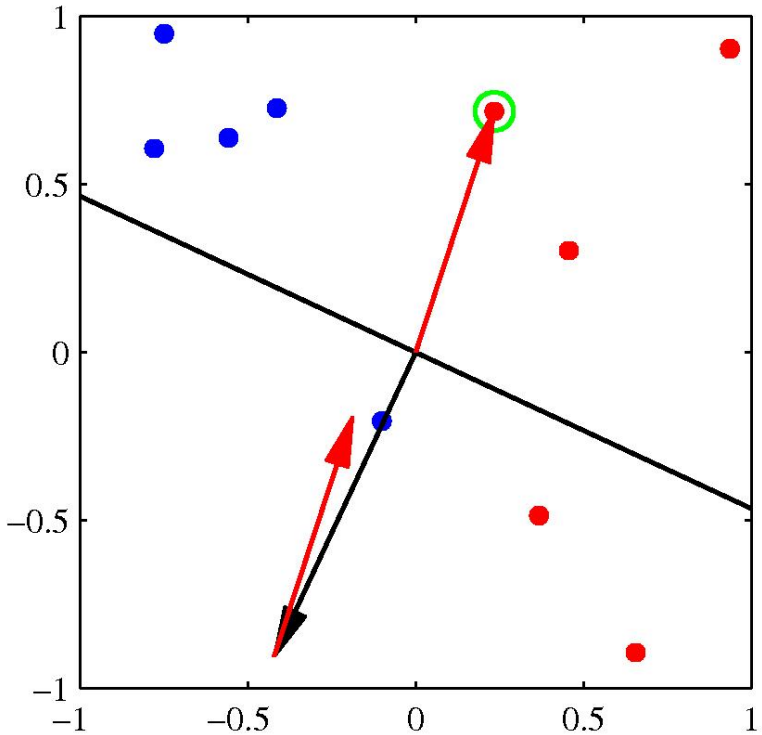


Figure: Figure 4.7 from C. Bishop: PRML

Perceptron Learning Algorithm (PLA)

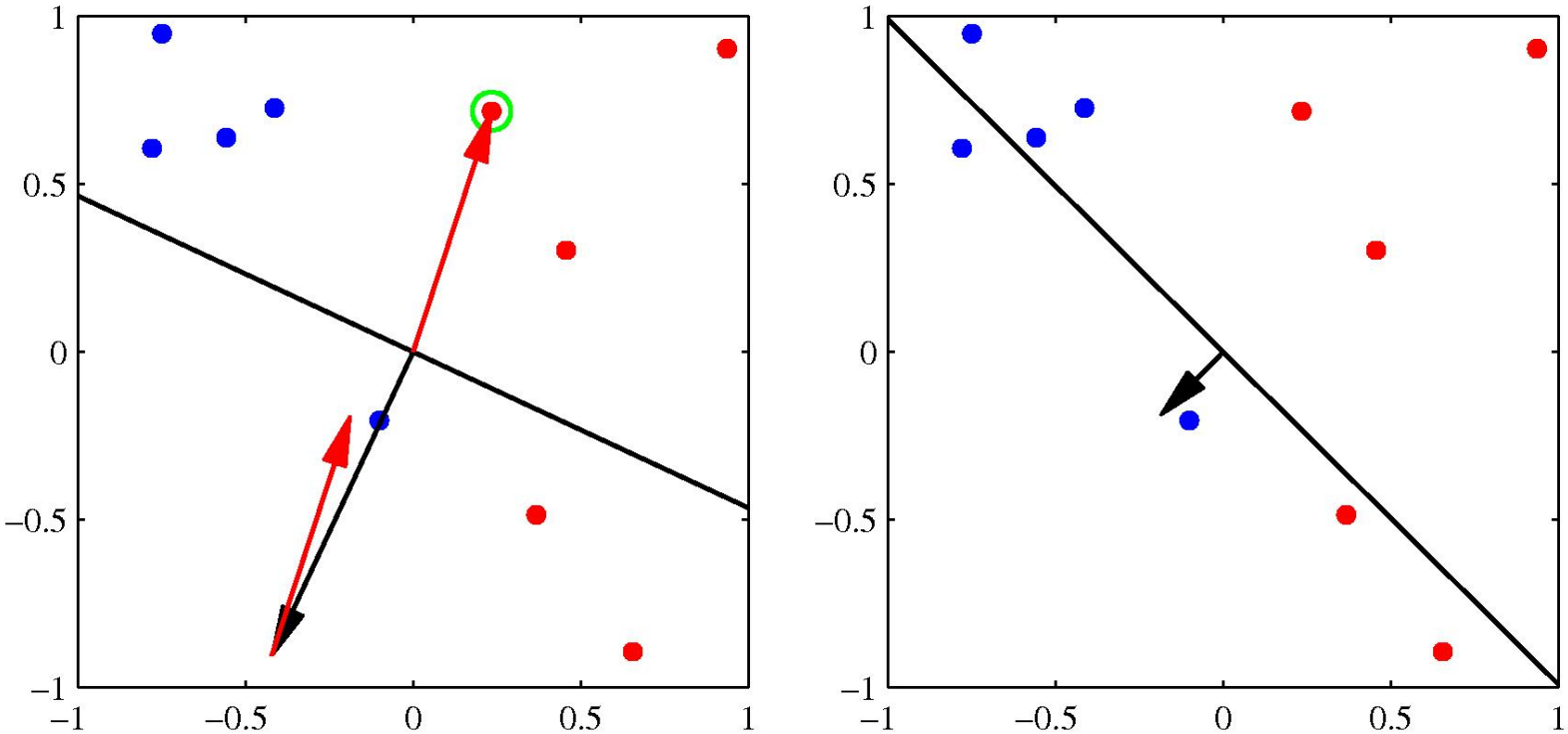


Figure: Figure 4.7 from C. Bishop: PRML

PLA

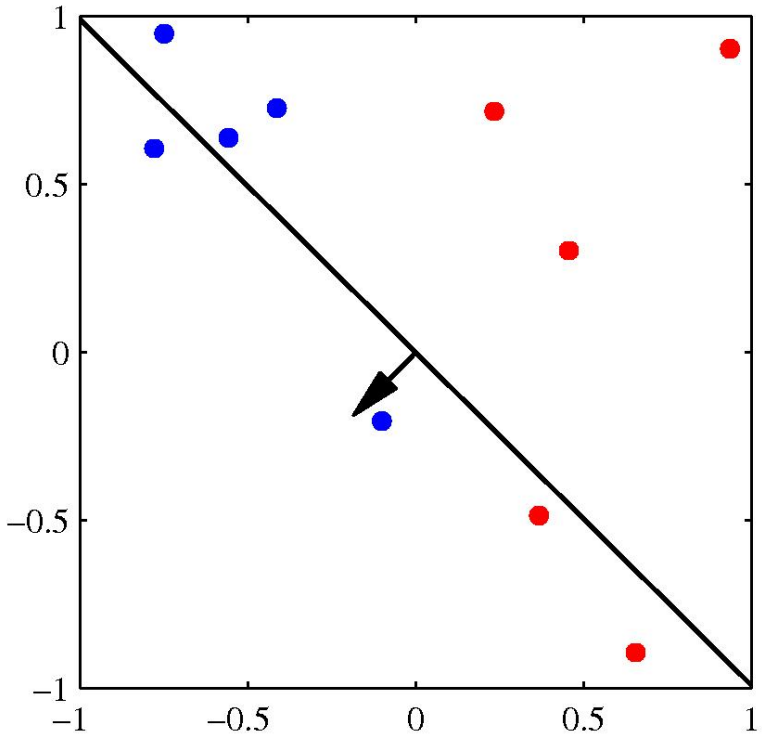


Figure: Figure 4.7 from C. Bishop: PRML

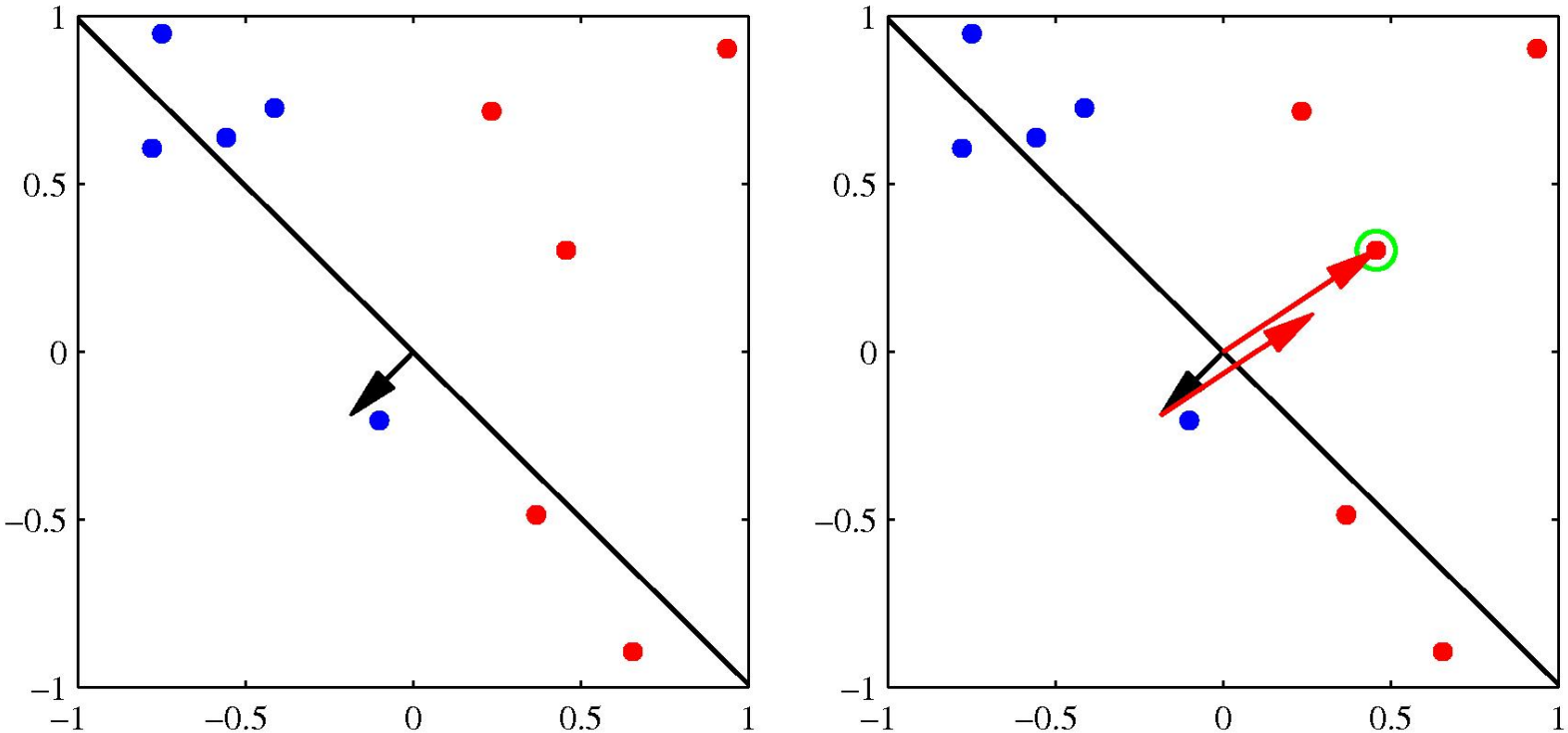


Figure: Figure 4.7 from C. Bishop: PRML

PLA

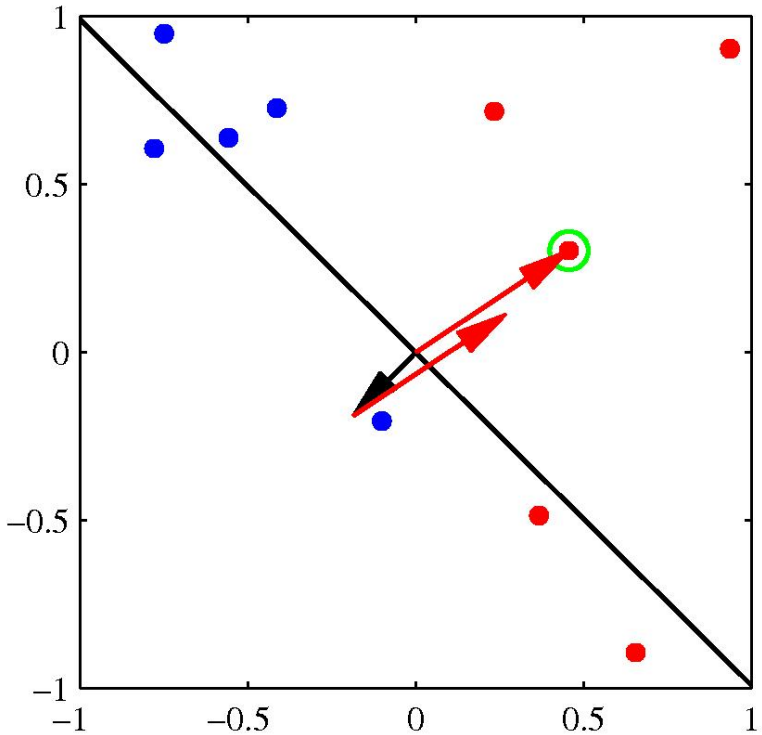


Figure: Figure 4.7 from C. Bishop: PRML

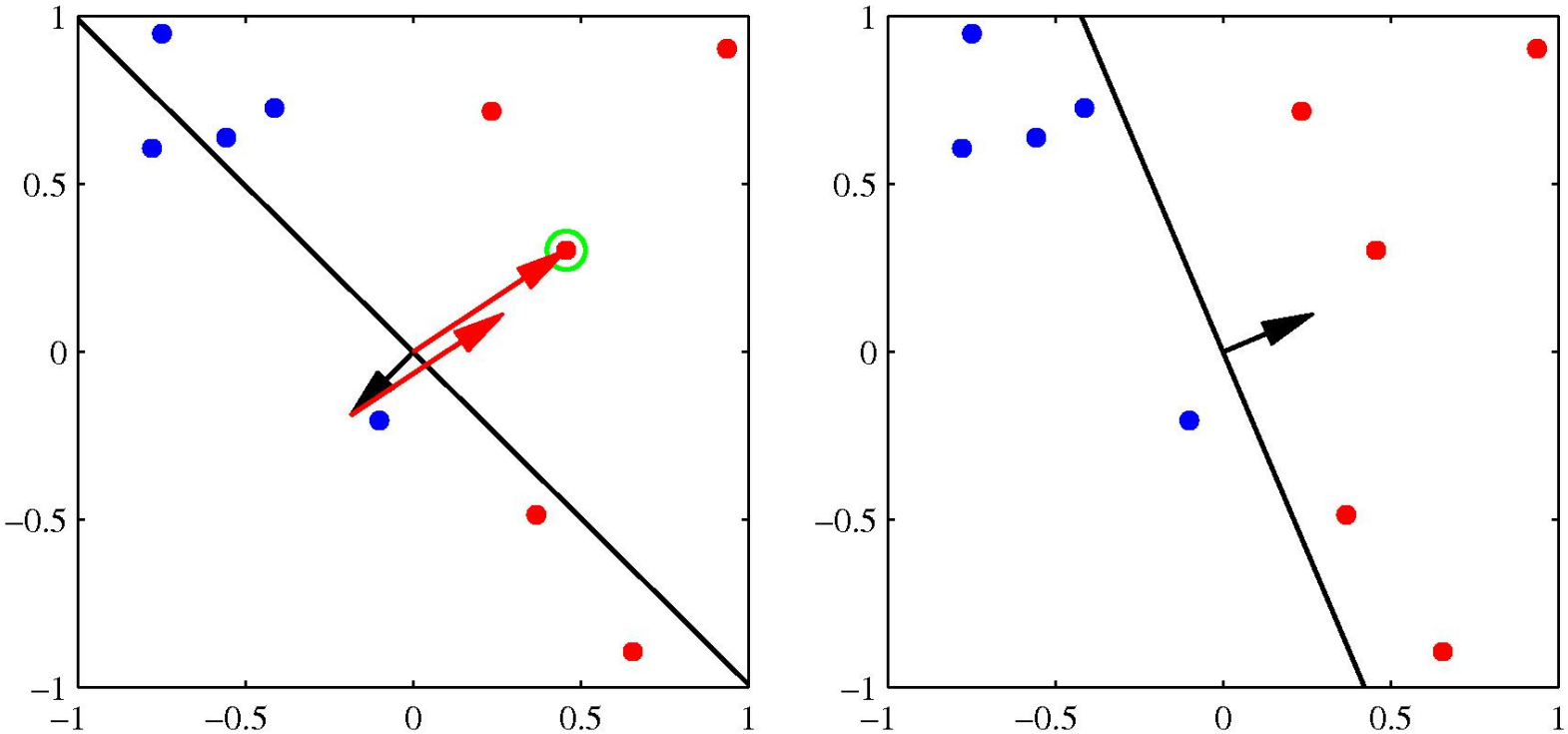


Figure: Figure 4.7 from C. Bishop: PRML

Margin of a dataset

We say that a dataset $D = ((\mathbf{x}_1, t_1), \dots (\mathbf{x}_N, t_N))$ with $\mathbf{x}_i \in \mathbb{R}^d$ is called (linearly) separable with margin γ if there exists a vector $\mathbf{w} \in \mathbb{R}^d$ with

$$y_{\mathbf{w}}(\mathbf{x}_i) = t_i$$

and

$$\left\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x} \right\rangle \geq \gamma$$

for all $i \in [N]$.

Perceptron Learning Algorithm (PLA)

We have the following runtime guarantee:

Theorem

Let the input data $D = ((\mathbf{x}_1, t_1), \ldots (\mathbf{x}_N, t_N))$ *be separable with margin* γ*. Then the perceptron algorithm stops after at most*

$$\frac{R^2}{\gamma^2}$$

updates (and outputs a separating hyperplane), where $R = \max\{\|\mathbf{x}_i\| \mid (\mathbf{x}_i, t_i) \in D\}$ *is the maximum norm of an input data point.*

Perceptron Learning Algorithm (PLA)
comment on runtime guarantee

The runtime guarantee is $\frac{R^2}{\gamma^2}$ where γ is the margin of the data and R the maximum norm of the datapoints. Note that the norm of the datapoints is required here: otherwise, we may artificially inflate the margin of the data by multiplying all input vectors $\mathbf{x}_i$ with some large constant. This would increase the margin. However, the data is still "essentially the same" (just stretched) and we should not expect the learning algorithm to run faster due to this stretch. The factor R^2 in the nominator of the bound reflects this. If we stretch the data artificially, the bound $\frac{R^2}{\gamma^2}$ remains the same.

If we want to compare two datasets, we can normalize the input points so that the maximum norm is 1 for both datasets. Comparing the margins after this normalization then reflects the difficulty of the respective datasets.

PLA summary

- Due to Frank Rosenblatt (1962), see story in PRML p193
- Very simple algorithm that works well if the data is actually separable; the larger the margin, the faster it converges
- If the data is not separable, then one can define some stopping criteria (but there is no guarantee of the quality of the output)
- Is actually a gradient descent method (we will see this later)