Support Vector Machines (SVMs)

Machine Learning for Classification

Ricardo Ñanculef

Bologna Business School 2020

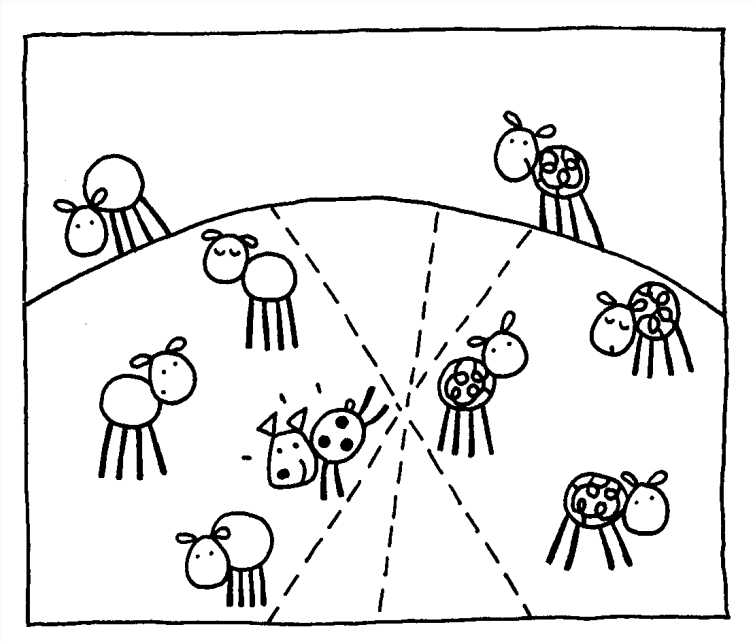
Table of contents

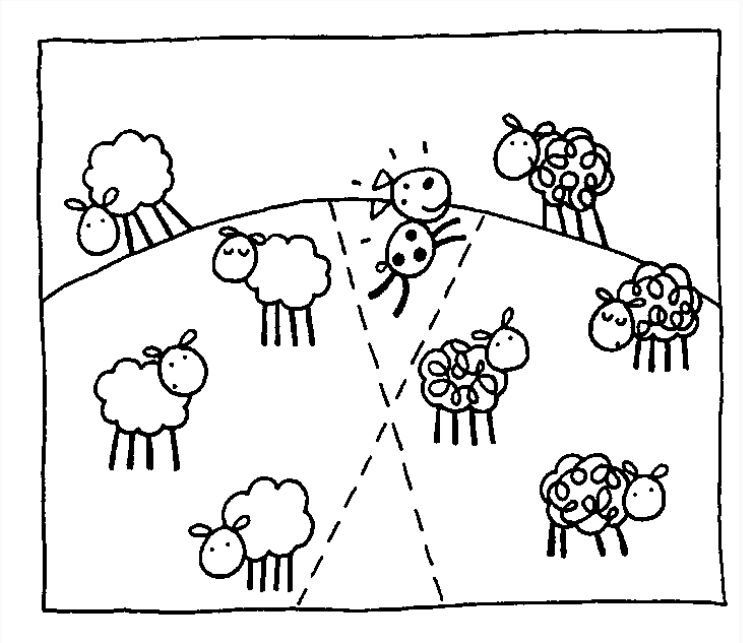
Separating Hyperplanes

Separating the Data using Hyperplanes

Support vector machines (SVMs) are methods primarily designed to address binary classification problems, i.e. classification problems with two possible outcomes .

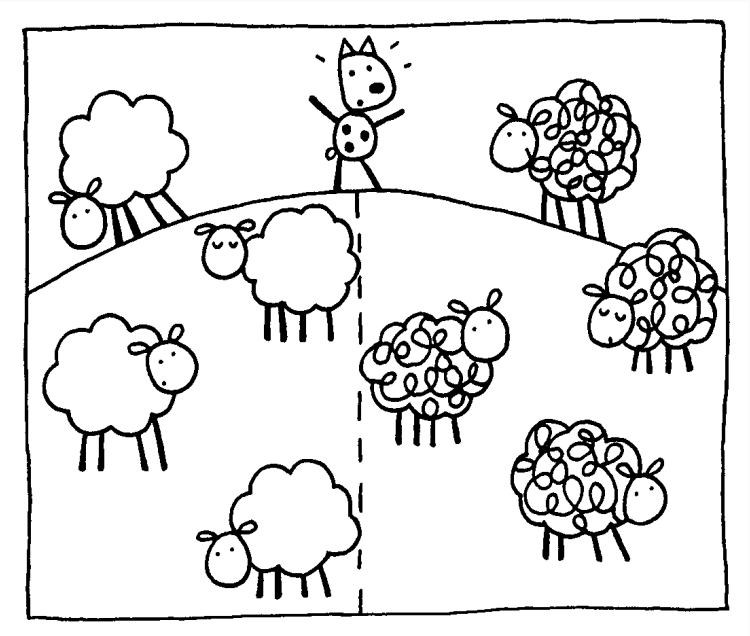
SVMs address problems of this type of problem learning a *hyperplane*, i.e. a "line" in many dimensions, that separates the classes.

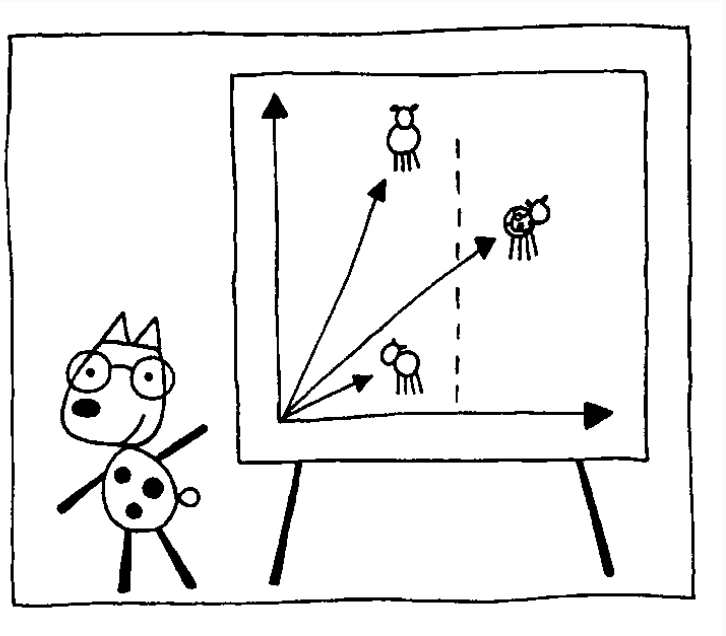




Separating the Data using Good Hyperplanes

Given a set of *training examples* $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{n}$, SVMs can automatically find a separating hyperplane that (as we will see) is strong against overfitting.

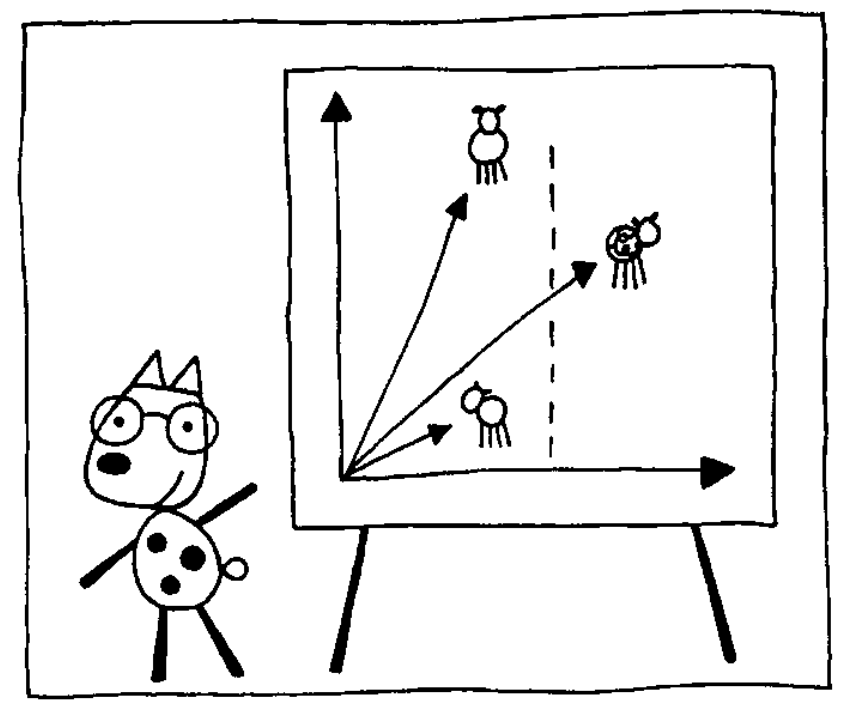




Preparing your Data

To apply SVMs yo need to represent your data such that

1. Any input x is a vector of real-valued attributes $x \in \mathbb{X} \subset \mathbb{R}^d$.
2. Any output y is $+1$ or -1.



Preparing your Data

1. Any input x is a vector of real-valued attributes $x \in \mathbb{X} \subset \mathbb{R}^d$.

   This allows to apply **linear algebra** to manipulate your data. In particular, it allows to formally introduce the idea of "hyperplane".

2. Any output y is $+1$ or -1.

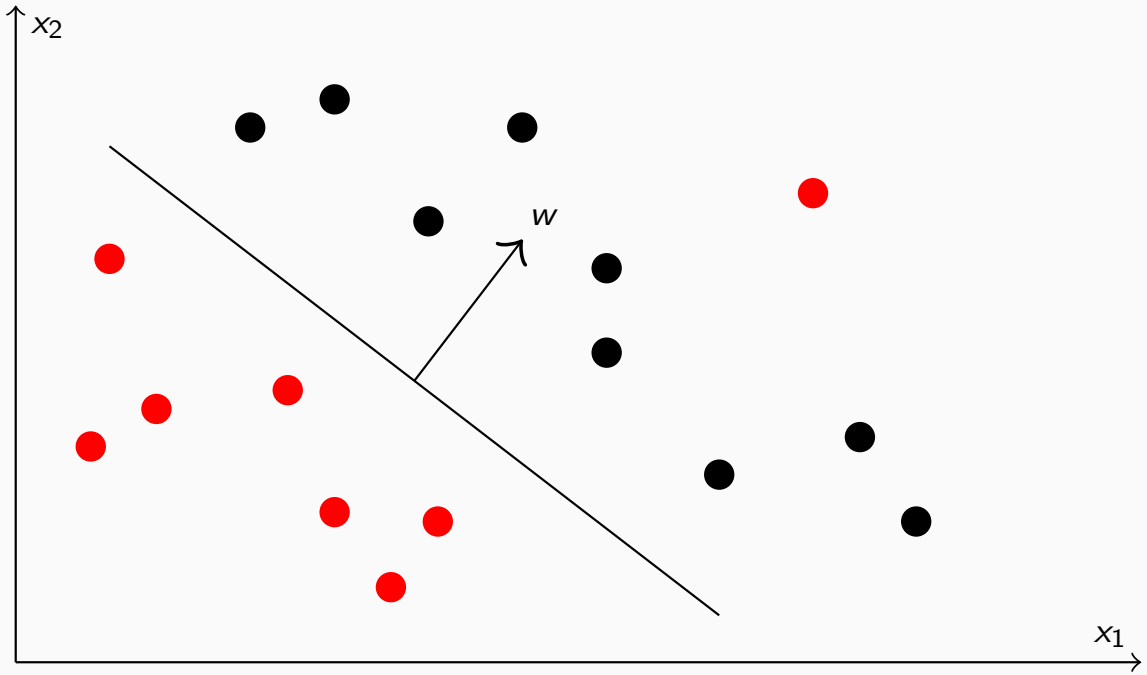
   This allows to use the positive side of the hyperplane to put one class and the negative side to put the other.

X is called the feature space.

Hyperplane Parametrization

Mathematically, a *hyperplane* $\mathcal{H}$ in a d-dimensional feature space ($\mathbb{X} \subset \mathbb{R}^d$) is a surface defined by two parameters:

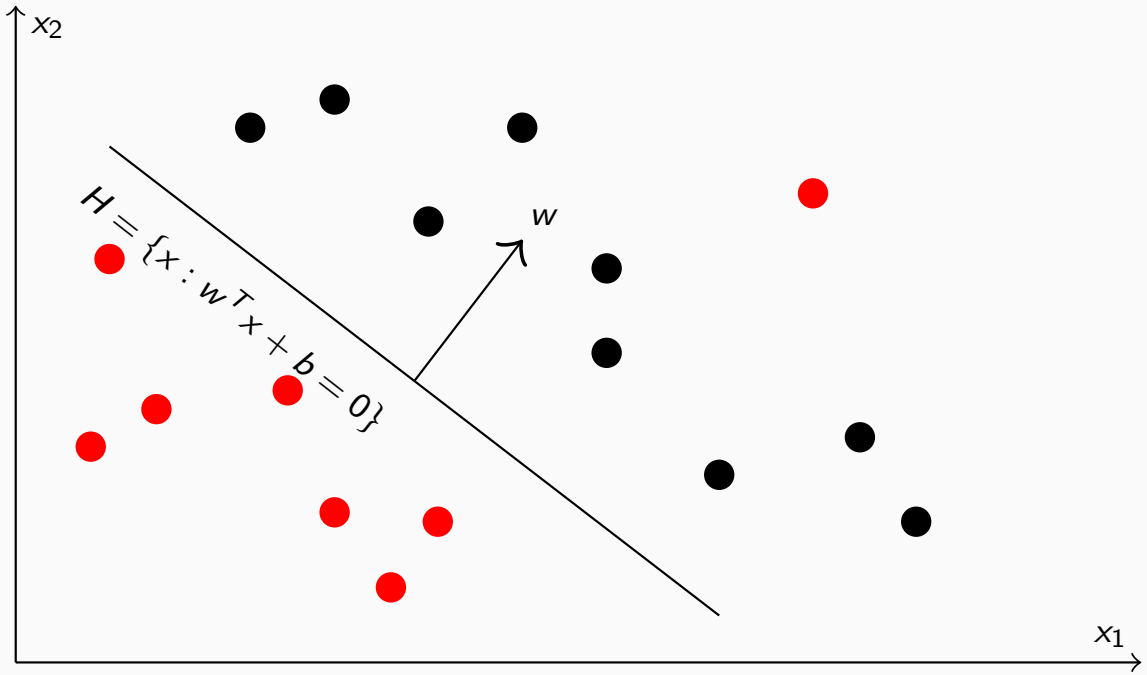
- A vector $w \in \mathbb{R}^d$, determining its orientation in the space (the surface defined by $\mathcal{H}$ is orthogonal to w).
- A scalar $b \in \mathbb{R}$, determining its "distance" to the origin.



Hyperplane Parametrization

More precisely, a *hyperplane* $\mathcal{H}$ with parameters $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ is the set of all the points x such that

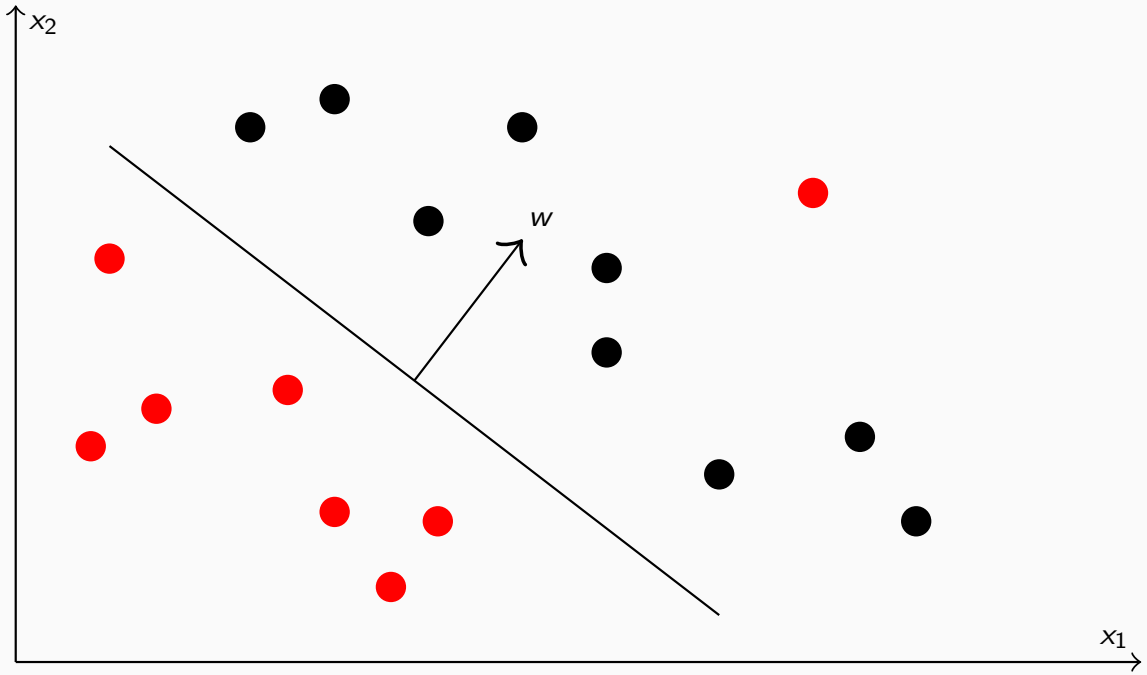
$$w^T x + b = \sum_i w_i x_i + b = 0 \tag{1}$$



Sides of a Hyperplane

If, for some point x, we have $w^T x + b \neq 0$, we have two possibilities,

$$w^T x + b > 0 \;\Rightarrow\; \text{we say that } x \text{ is on the "positive side" of } \mathcal{H}. \quad (2)$$
$$w^T x + b < 0 \;\Rightarrow\; \text{we say that } x \text{ is on the "negative side" of } \mathcal{H}$$



Hyperplanes as Classifiers

Basic Idea: If we encode the classes as ± 1, we can use the sides of the hyperplane to identify the classes.

Classification Rule: We can decide/predict the class of an input $x \in \mathbb{X}$, evaluating $f(x) = w^T x + b$. If $f(x) > 0$ we predict that the class of x is $+1$. Otherwise the prediction for x is -1.

The classifier takes thus the form:

$$f(x) = \text{sign}\left(w^T x + b\right) \tag{3}$$

Interpretation of the Classification Rule

Classification Rule:

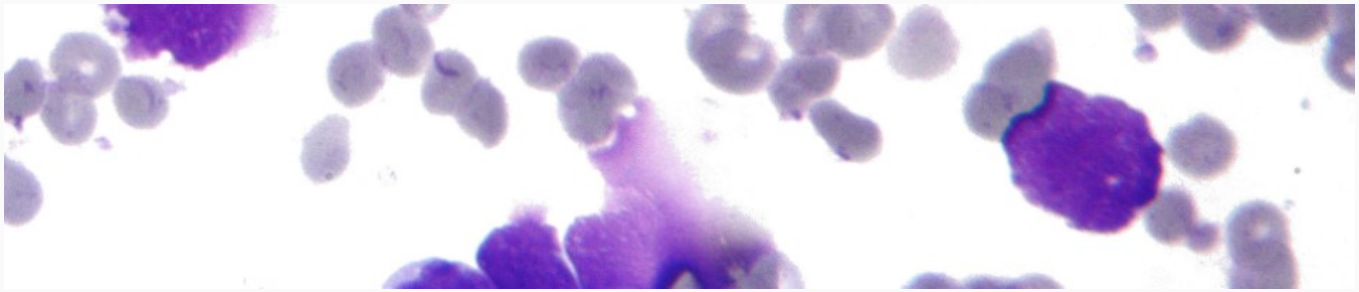
$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_i w_i x_i + b\right)$$

- If $w_i > 0$, a large value of the attribute x_i increases the chance of x being classified into the positive class.
- If $w_i < 0$, a large value of the attribute x_i increases the chance of x being classified into the negative class.
- If $w_i = 0$, the attribute x_i is not relevant for classifying x.
- b changes the "bias" of the classifier to classify points into one of the two classes.

Example

Wisconsin Breast Cancer Problem

You have ($n = 569$) cell images diagnosed as cases of breast cancer.
Some of them are malignant and other are benign.
You are asked to learn a classifier to predict the type of future images.



Example

Wisconsin Breast Cancer Problem

Images have to be represented as vectors of features that describe the characteristics of the cell present in the image.

Classes are encoded as $+1$ (malignant) or -1 (benign).

For instance, we can work with 2 attributes: *cell radius* x_1 and *texture* (standard deviation of gray-scale values) x_2.

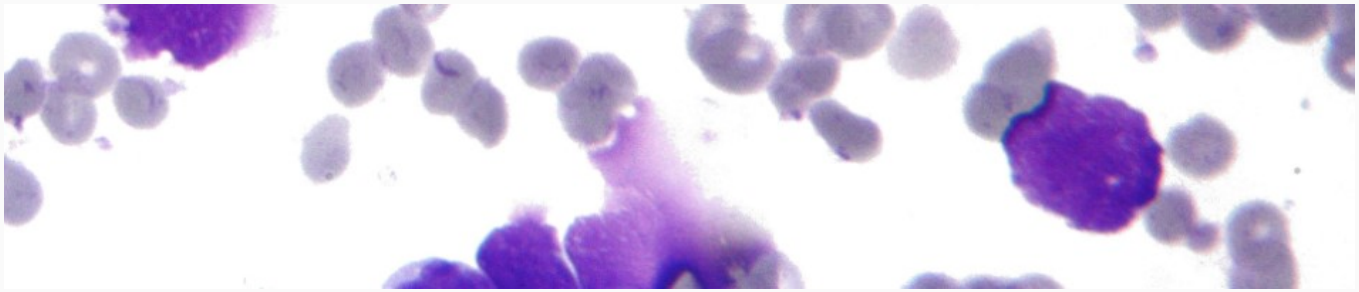
The classification rule for this case takes the form

$$f(x) = \text{sign}\left(w_1 x_1 + w_2 x_2 + b\right)$$

Example

Wisconsin Breast Cancer Problem If your hyperplane has parameters $w_1 = 2(> 0)$, $w_2 = -1(< 0)$ and $b = 1(> 0)$.

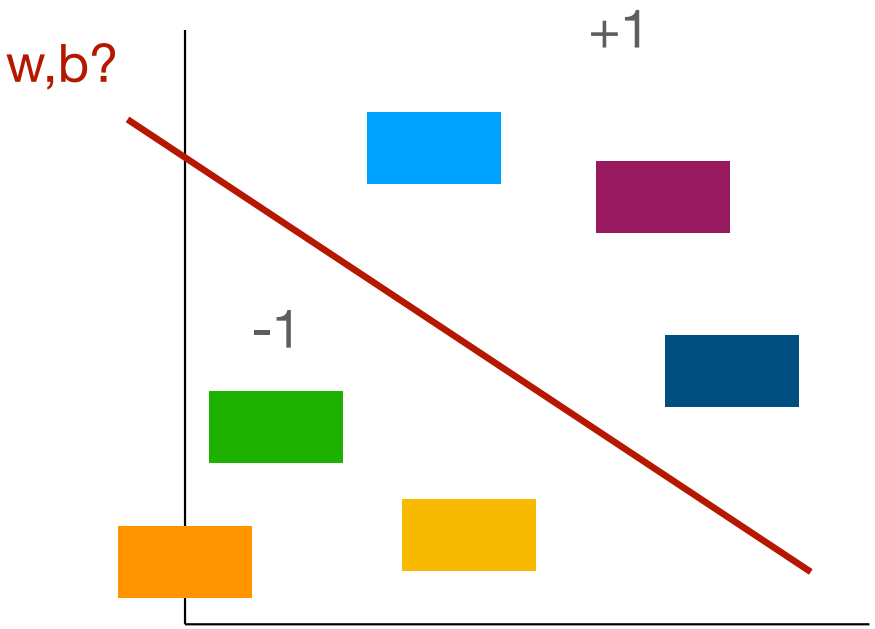
- A larger *radius* increases the chance of the image being classified as *malignant*.
- A highly *textured* image increases the chance of the image being classified as *benignant*.
- The classifier has a "bias" towards the *malignant* class.



Challenge: Learning the Hyperplane

How can we automatically find values for w and b using a set of *training examples* $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{n}$?

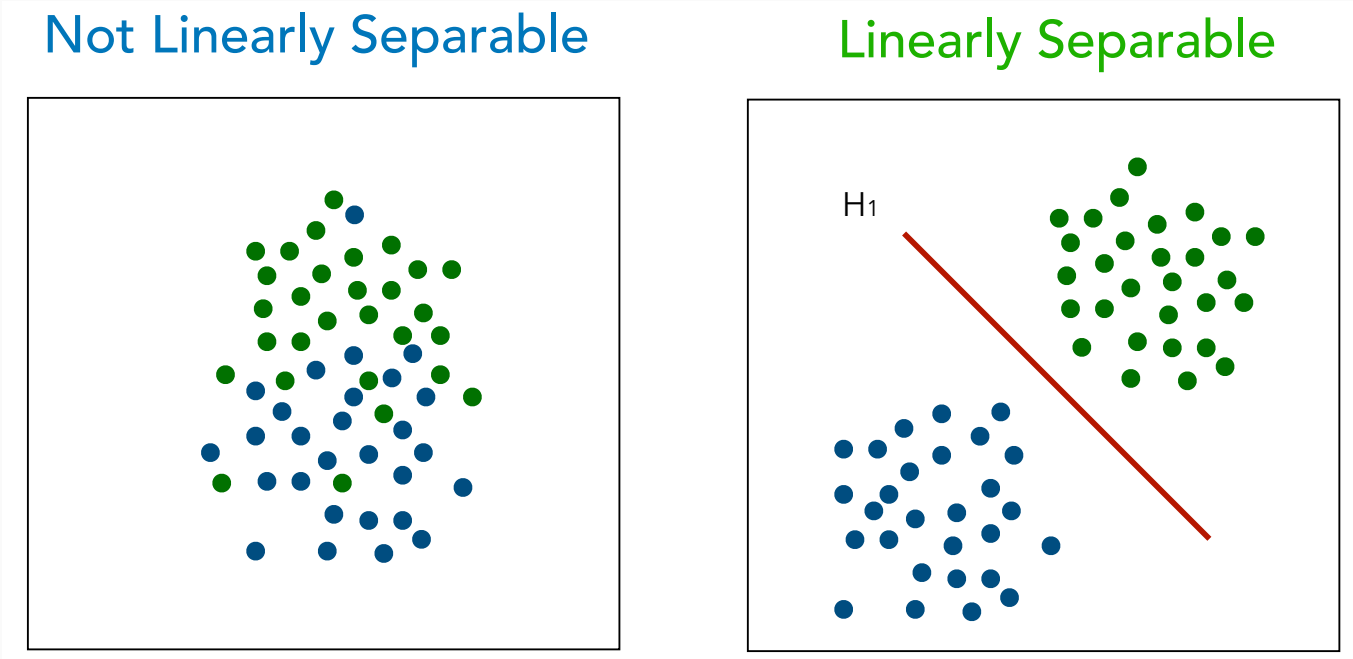
opinions with correct label



Large Margins or ... How to Learn a Good Hyperplane?

Hyperplanes and Linearly Separability

- Suppose by now that your training data is **linearly separable** (i.e., there exist a hyperplane that correctly separates the data).



- In this case, it is not hard to devise a method that automatically finds a hyperplane consistent with the dataset.

Hyperplanes and Linearly Separability

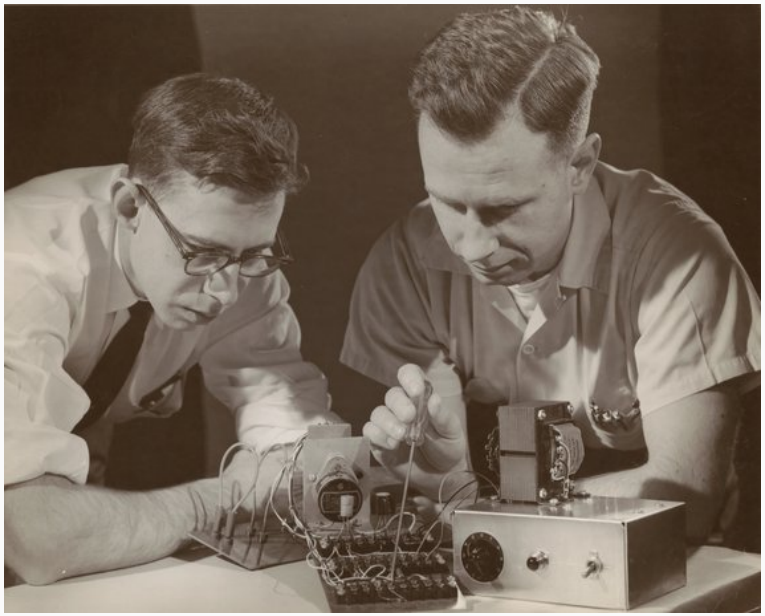
- For instance, a very old method that can make the job is **the perceptron**, an algorithm invented by Frank Rosenblatt in 1957.

Input : A labelled dataset $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{n}$ and $\eta \in \mathbb{R}^+$
Output: A trained hyperplane with parameters w, b

$\boldsymbol{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$;
do
  mistakes $\leftarrow$ `false`
  for $\ell = 1, \ldots, n$ **do**
    if $y^{(\ell)}(w^T x^{(\ell)} + b) \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{y}^{(\ell)} x^{(\ell)}$
      $b \leftarrow b + \eta \boldsymbol{y}^{(\ell)}$
      mistakes $\leftarrow$ `true`
while *mistakes*;

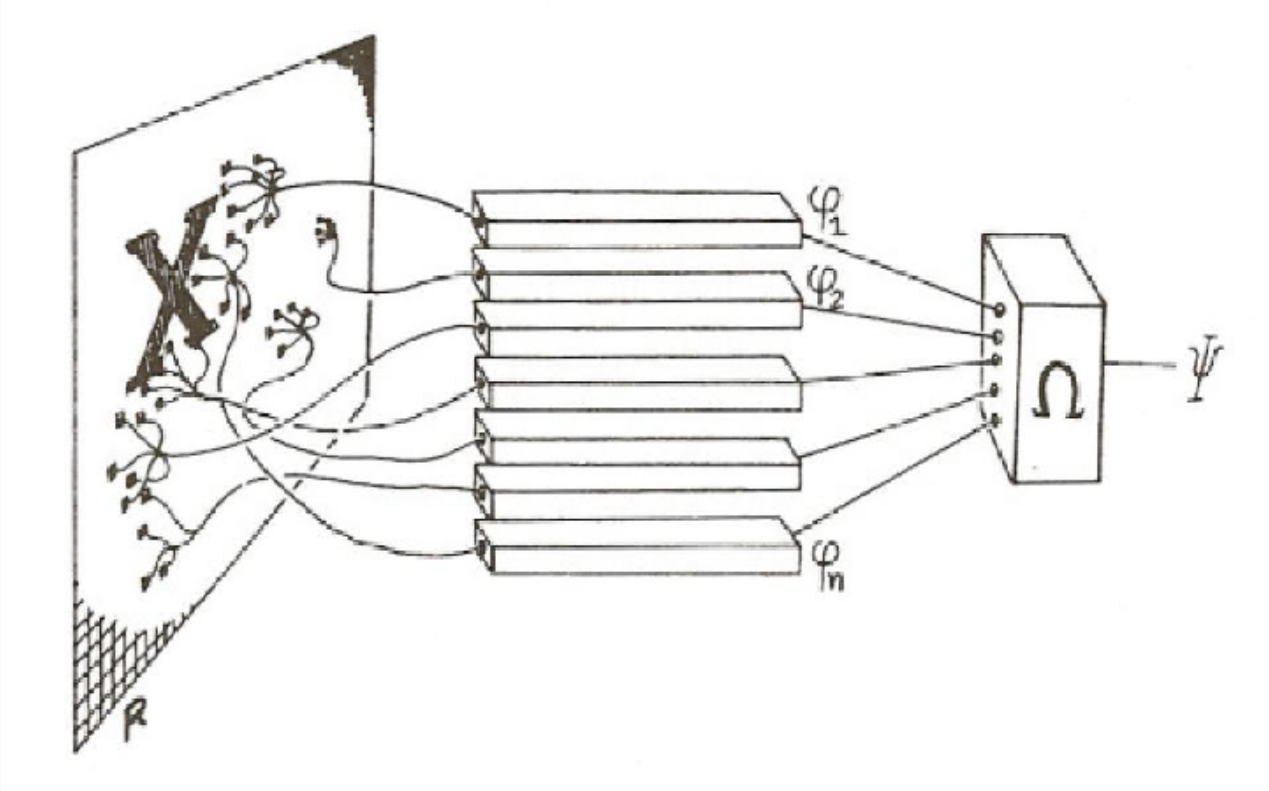
The Mark-I Perceptron

In the 50's, Rosenblatt implemented this algorithm on a physical machine named: The Mark-I Perceptron.



The Mark-I Perceptron

Using the algorithm, the Mark-I Perceptron *learned* to recognise letters of the alphabet from 20×20 images.



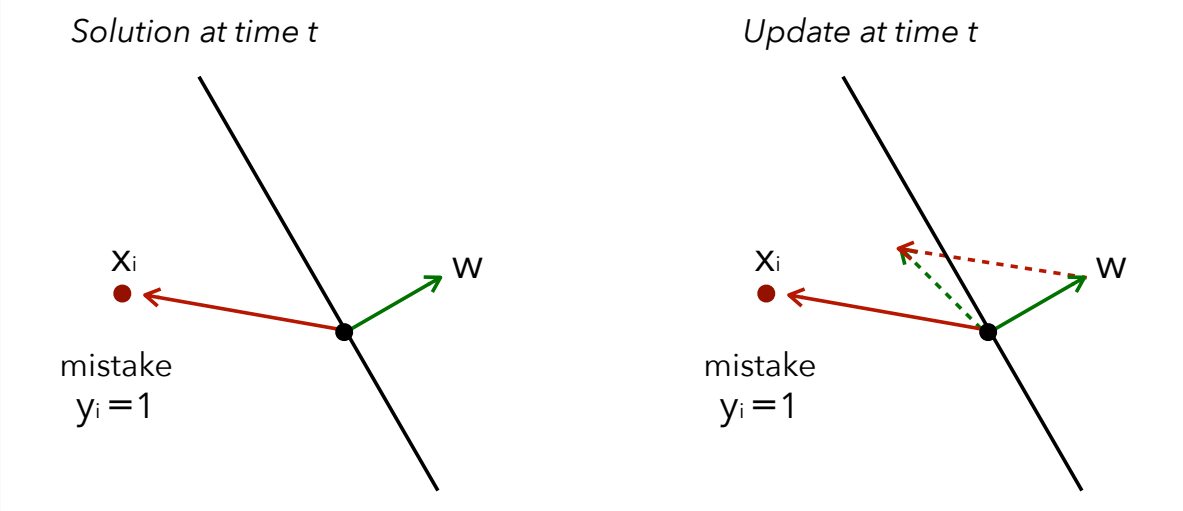
Perceptron Learning

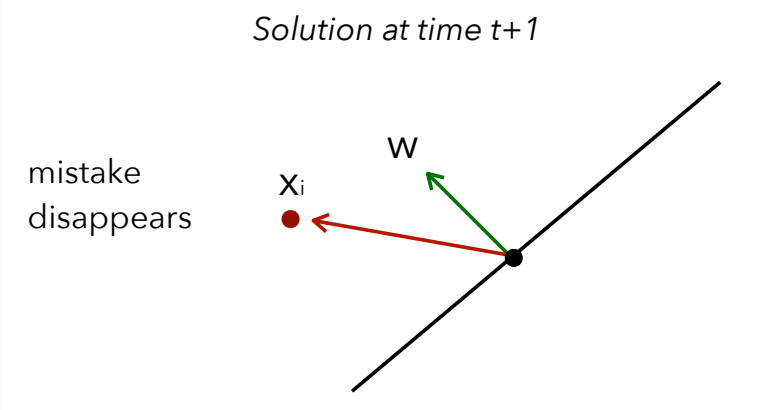
The learning cycles among the training examples, updating the values of w, b when it founds a prediction mistake, i.e. when $y^{(\ell)} f(x^{(\ell)}) < 0$. Recall $f(x^{(\ell)})$ is the algorithm's prediction and $y^{(\ell)}$ the groundtruth corresponding to the example $x^{(\ell)}$.

	Prediction: Sign of $(w^T x^{(\ell)} + b)$	
Groundtruth $y^{(\ell)}$	$+1$	-1
$+1$	correct	mistake
-1	mistake	correct

Perceptron Learning

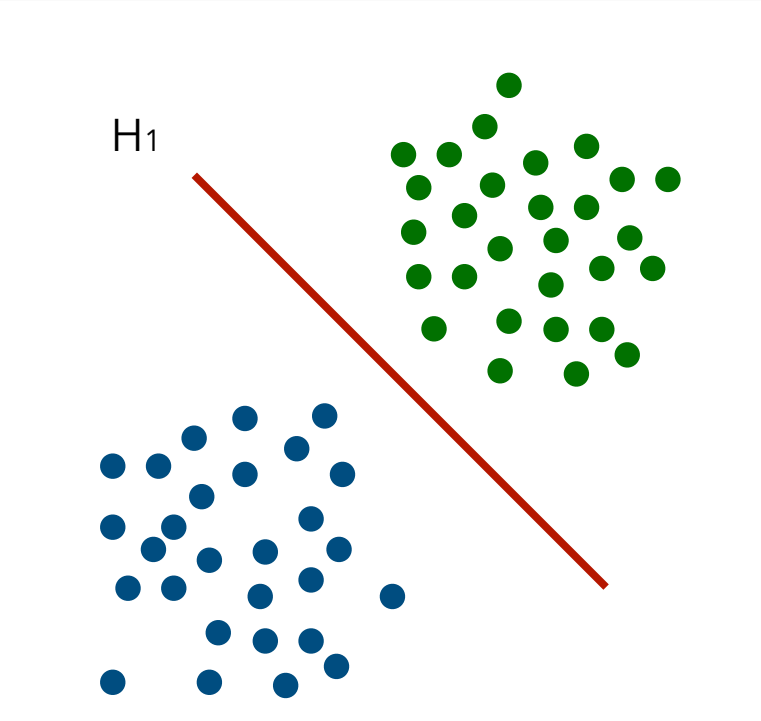
To correct the mistake, the algorithm "moves" w, b in such a way that $y^{(\ell)} f(x^{(\ell)})$ become less negative. In particular, it moves w in the direction of $x^{(\ell)}$ if $y^{(\ell)} = +1$ and away of $x^{(\ell)}$ if $y^{(\ell)} = -1$.





Perceptron Learning

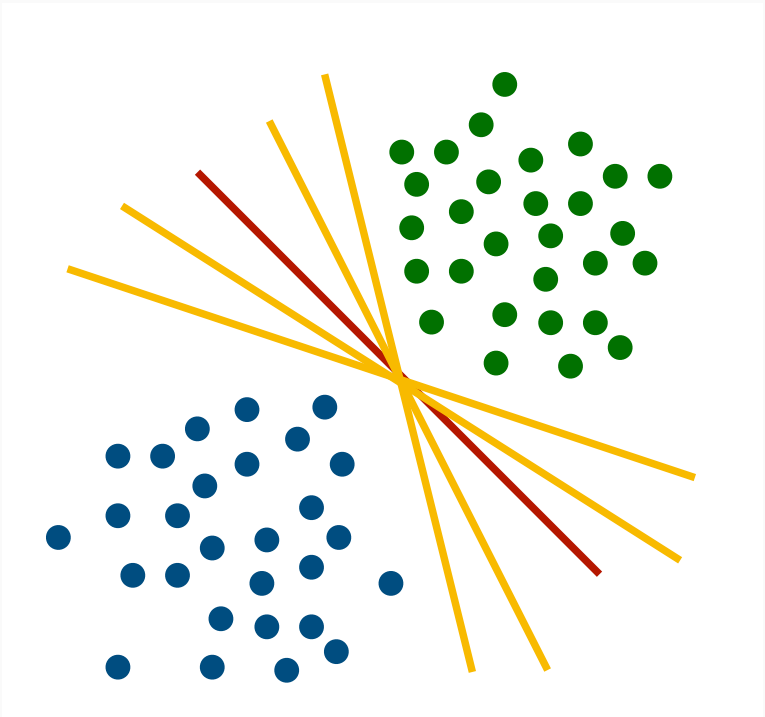
It can be shown that, if the data is linearly separable, the perceptron finds a separating hyperplane after a finite number of steps.



Perceptron Learning

If the data is linearly separable, however, infinitely many solutions exist.

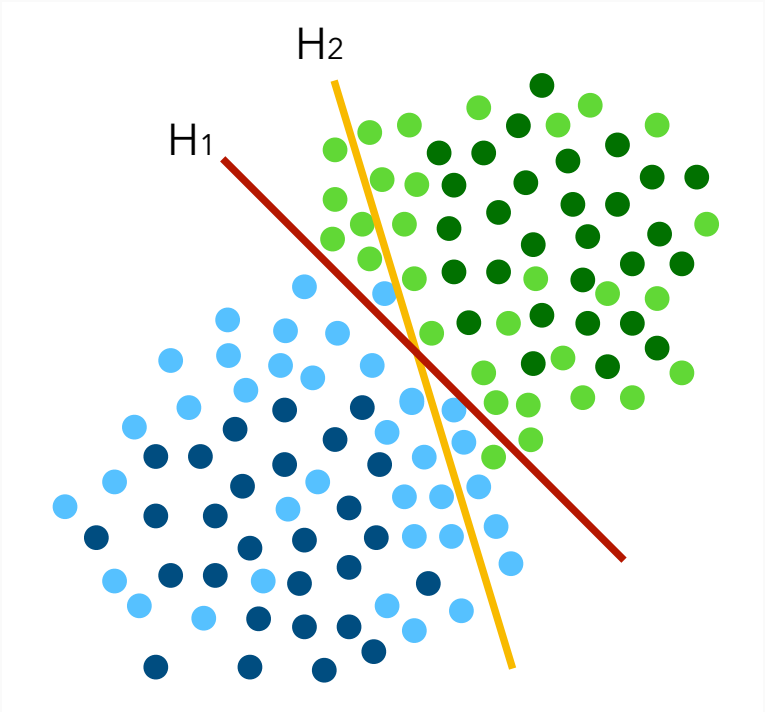
For the perceptron, all of them are equally good, because the perceptron focuses only on reducing the training error.



Prediction versus Training Error

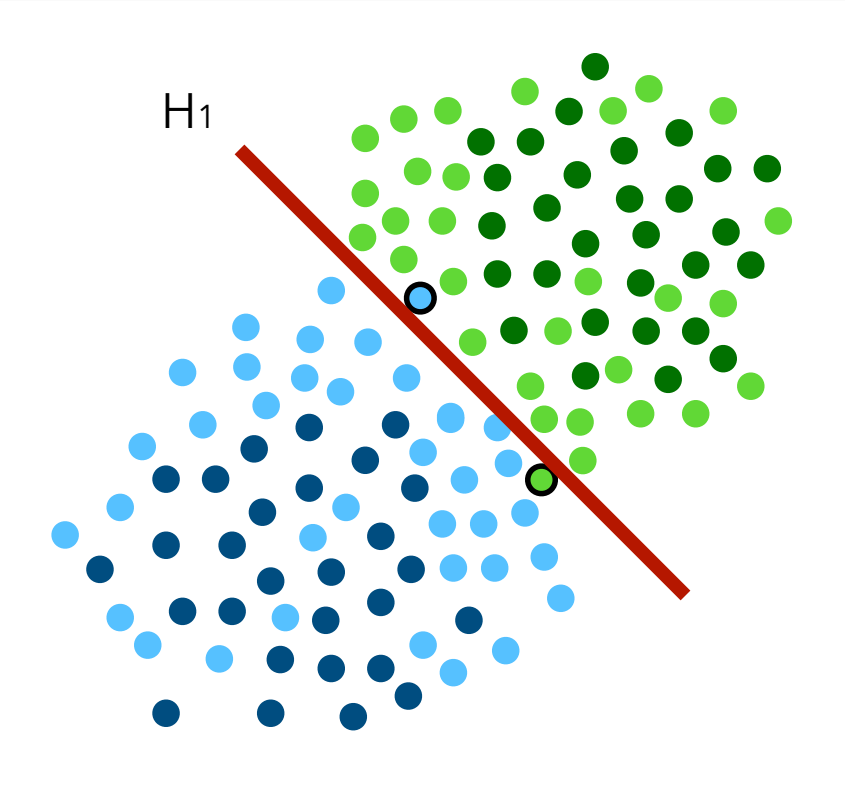
What about the the expected error of these hyperplanes on novel/test data? (clear colors in the picture).

Are the hyperplanes below equally good in terms of prediction error?



Prediction versus Training Error

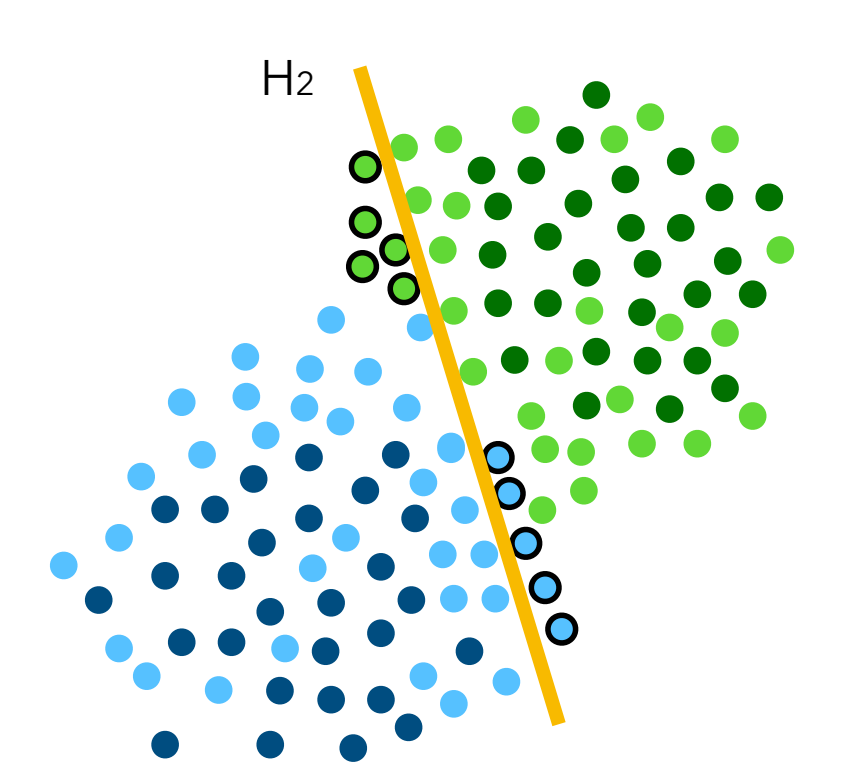
H_1 makes 2 mistakes on novel data.



Prediction versus Training Error

H_1 makes 2 mistakes on novel data.

H_2 makes 10 mistakes on novel data.

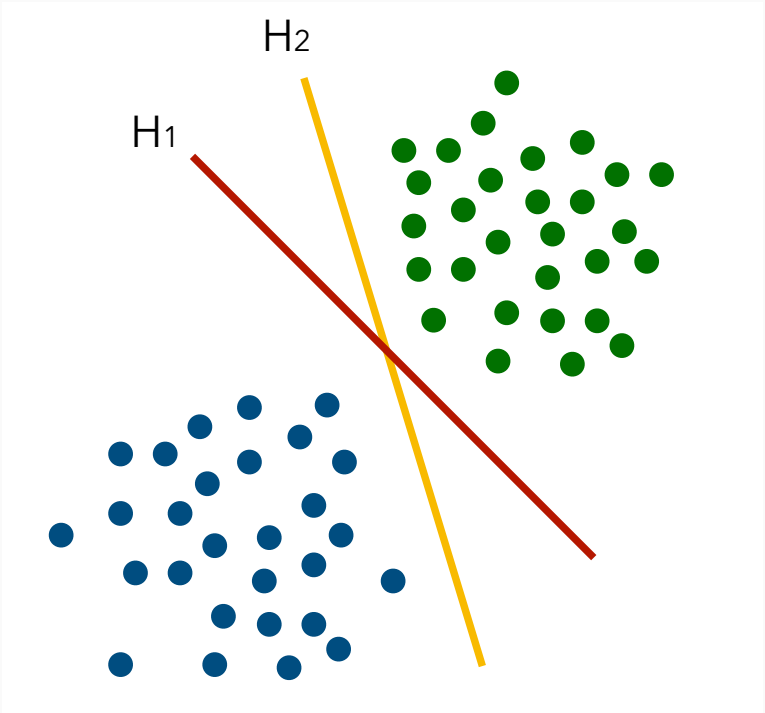


Prediction versus Training Error

Unfortunately, we do not know where test data will appear.

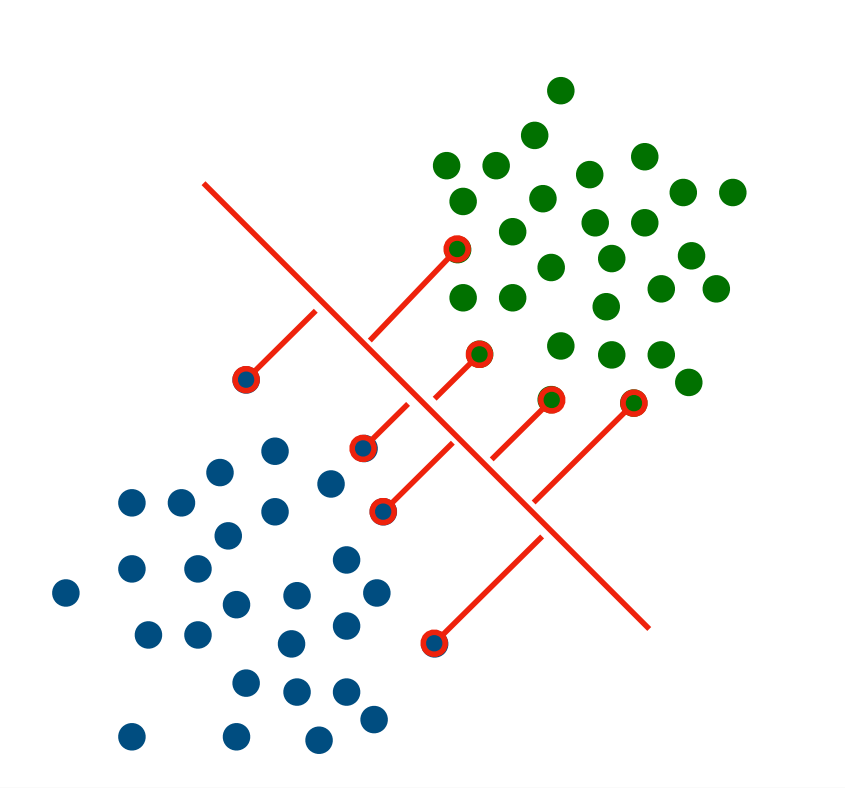
Can we guess where?

Is there a hyperplane with greater probability of being a good classifier on test data?



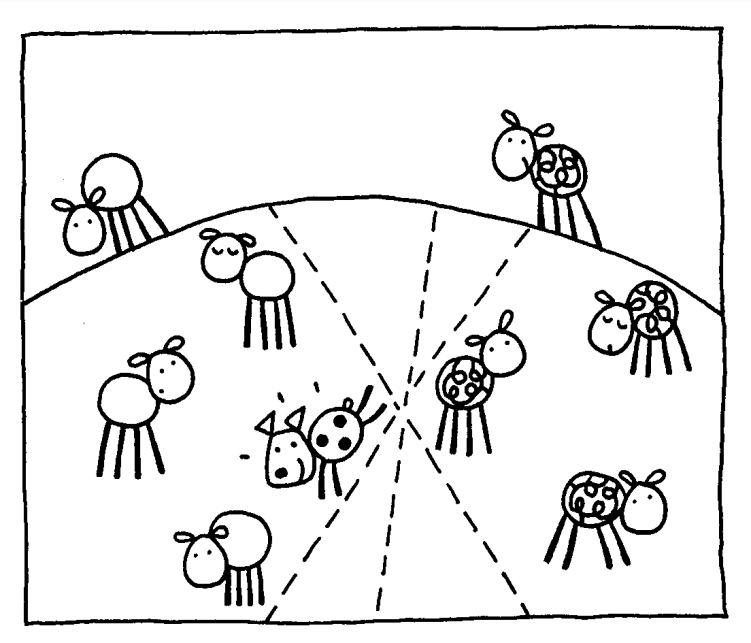
Margin

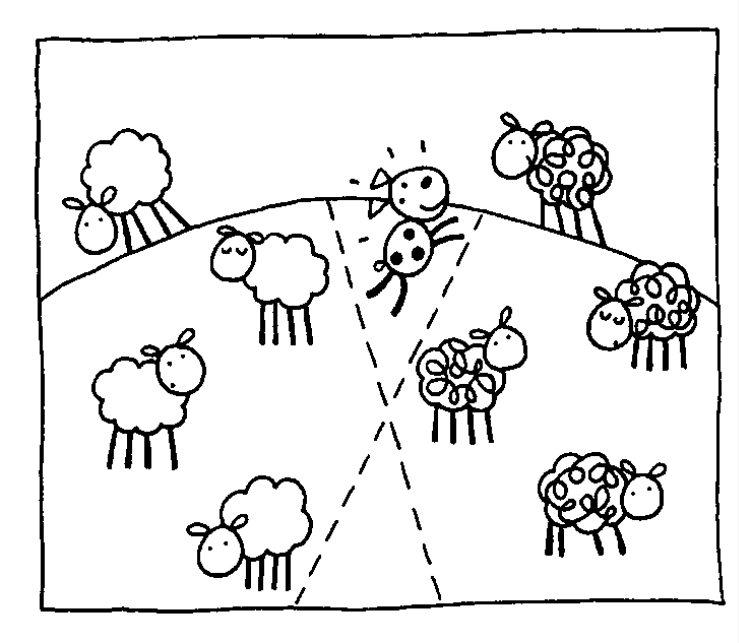
Key Idea: If you have many possible solutions, choose that with the largest **margin**, i.e. the **largest distance to the nearest training point**.



Intuition

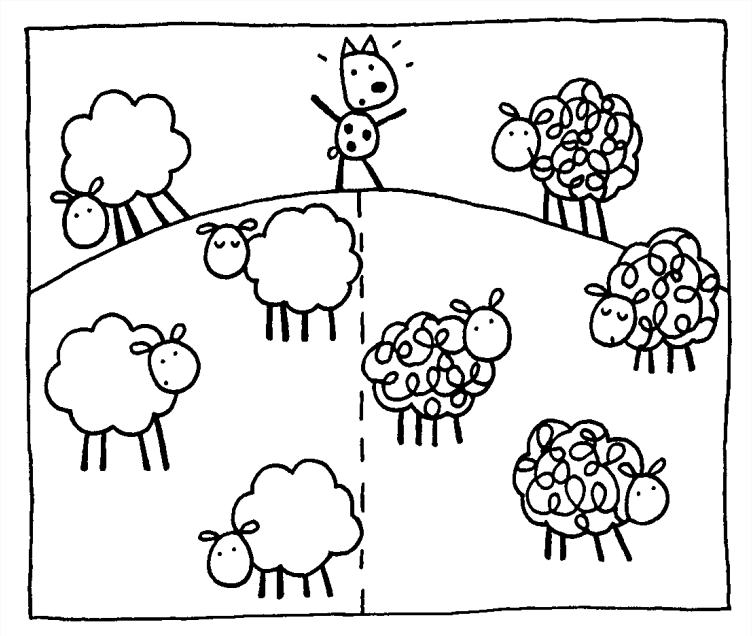
Many classifiers can be consistent with your data (sheeps). However, if your novel data is slightly different (sheeps get fatter), many of these solutions are not longer consistent.

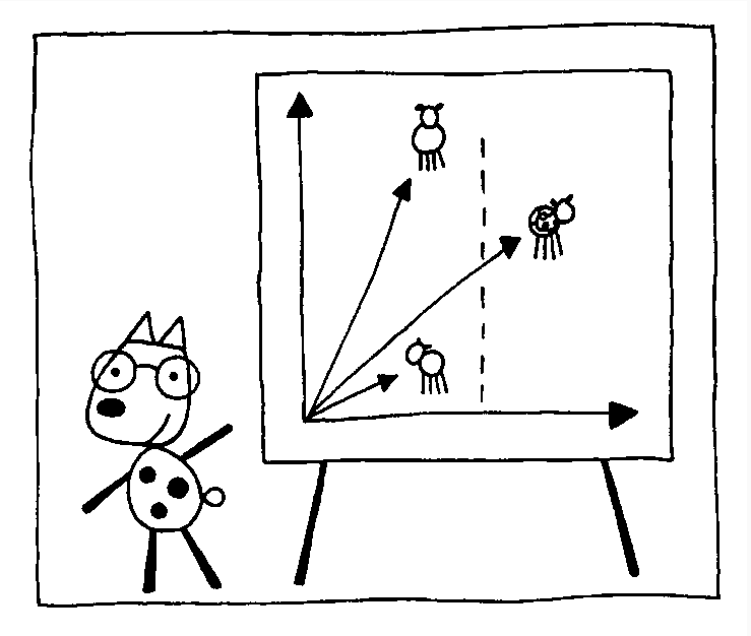




Intuition

The best hyperplane is the one giving your data more space to grow/change in the future.

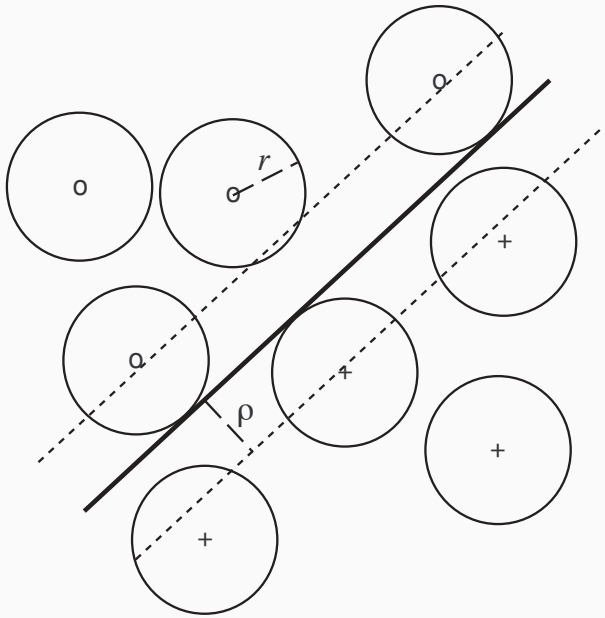




Intuition

More formally, you can expect novel/test data in the neighbourhood of your training items.

The hyperplane with a large margin is a solution more *robust* to changes in the training data and thus it has more chances to perform well on novel/test data.



Margin-Based Generalization Bound

This intuition can be demonstrated to be right.

Theorem*

Suppose your algorithm gets always a separating hyperplane with margin ρ on the training data S. It holds with probability at least δ

$$\textbf{Test Error} \leq \textbf{Train Error}(=0) + \text{cte} \cdot \sqrt{\frac{\frac{R^2}{\rho^2} \ln m + \ln \frac{1}{\delta}}{n}},$$

where cte is a constant independent of the training data.

* Vapnik, W. (1998). *Statistical Learning Theory*. Wiley.

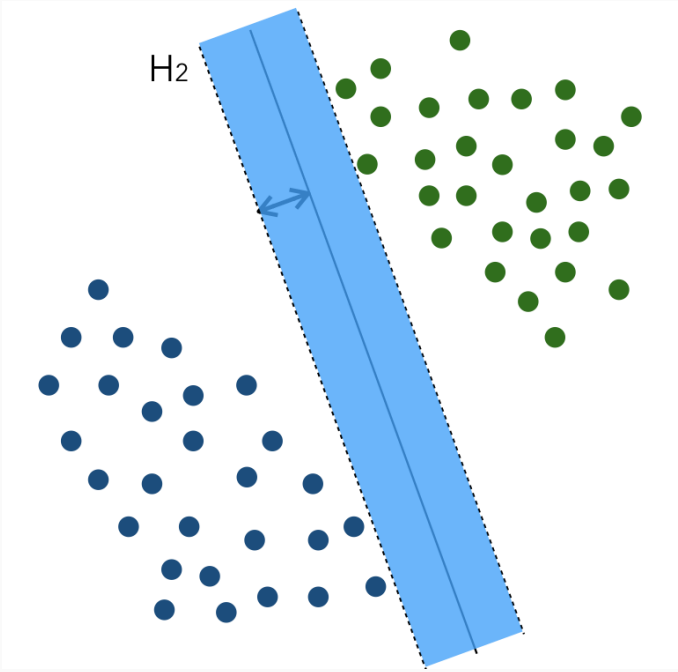
* Zhang, T. (2002). *Covering Number Bounds of Certain Regularized Linear Function Classes*. Journal of Machine Learning Research, 2, 527-550.

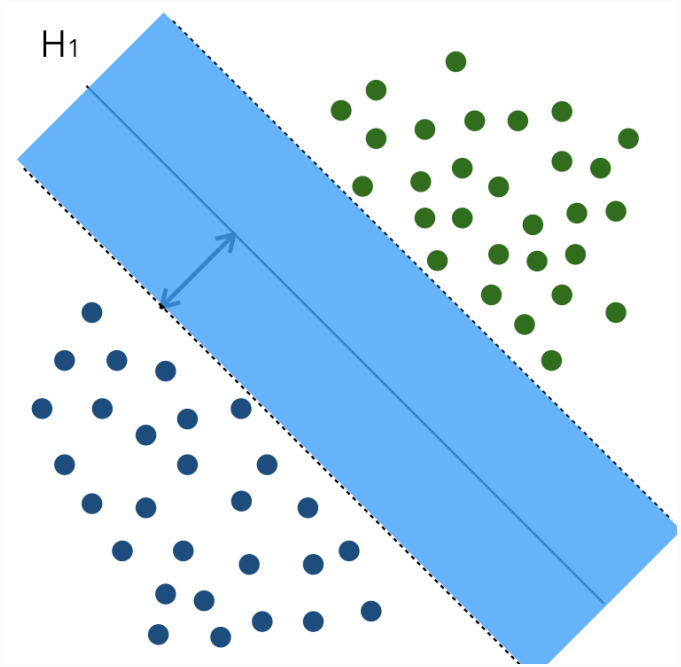
Linear SVMs

Linear SVM

SVM's learns a separating hyperplane by maximizing the margin.

With high probability, this solution is better than other separating hyperplanes on test/novel data.

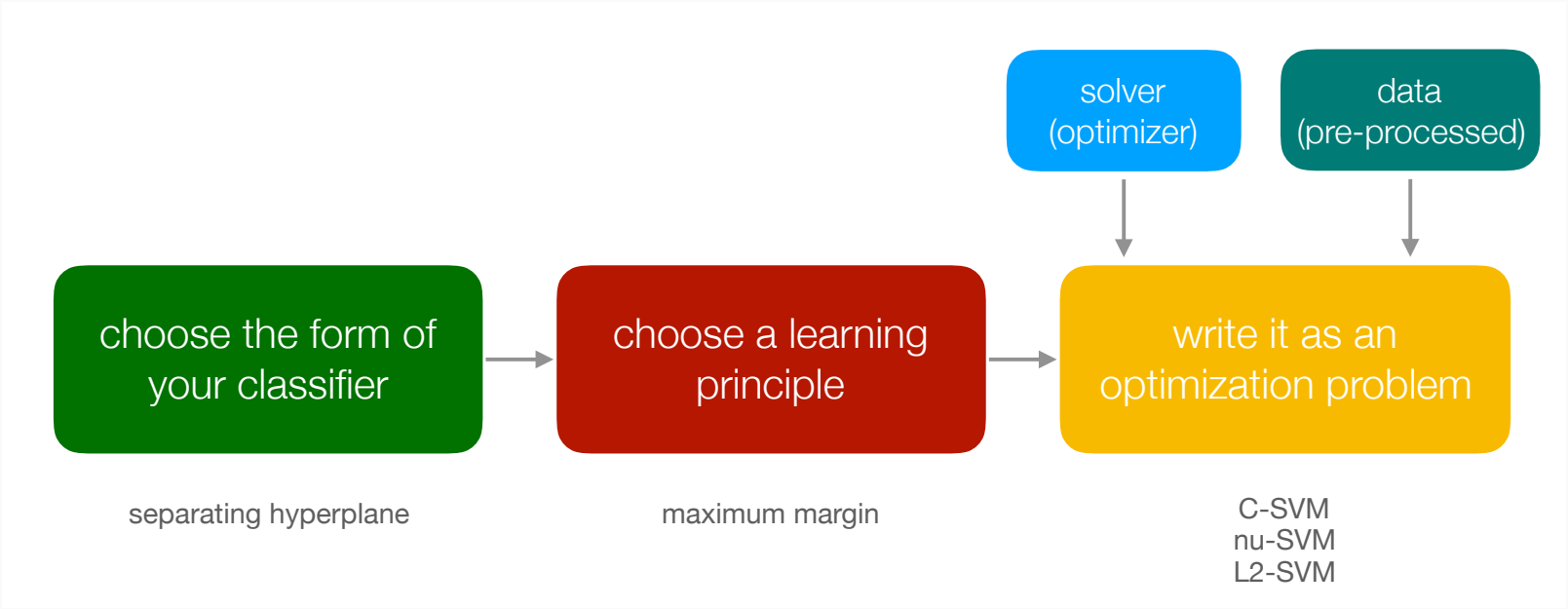




Linear SVM

How to find the hyperplane maximizing the margin on a given dataset?

⋆ Next step: write the maximum margin principle as an optimization problem on the dataset and use a solver/optimizer to find the solution.



The way to perform the last step leads to different types of SVM.

Derivation of the Basic SVM

- Let $\rho^{(\ell)}(w, b)$ denote the (orthogonal) distance of a training example $x^{(\ell)} \in S$ to a hyperplane with parameters w, b.
- The maximum margin hyperplane maximizes the minimum of these distances, that is, it is the solution to the following optimization problem with constraints

$$\max_{w,b} \rho \text{ s.t. } \rho^{(\ell)}(w, b) \geq \rho \tag{4}$$

- Note that by maximizing ρ, all the points $x^{(\ell)} \in S$ are at a distance greater than ρ from the hyperplane:

$$\rho^{(\ell)} \geq \rho \ \forall \ell = 1, \ldots, n. \tag{5}$$

Derivation of the Basic SVM

- It is not hard to show that $\rho^{(\ell)}(w, b)$ can be computed as:

$$\rho^{(\ell)} = \left| \frac{w^T x^{(\ell)} + b}{\|w\|} \right| .$$

- However, if $\mathcal{H}$ is a separating hyperplane we have that $\text{sign}\left(w^T x^{(\ell)} + b\right) = y^{(\ell)}\ \forall \ell$. Therefore, (if the data is linearly separable), the optimal hyperplane can be found by solving

$$\max_{w,b} \rho \text{ s.t. } y^{(\ell)} \frac{\left(w^T x^{(\ell)} + b\right)}{\|w\|} \geq \rho \tag{6}$$

- Note now that adding the constraint $w = \rho$ to the problem does not change the solution. This trick allows to simply the problem. Indeed ...

Basic SVM

- We can find the maximum margin hyperplane by solving the following optimization problem

$$\max_{w,b} \frac{1}{\rho} \text{ s.t. } y^{(\ell)} \left(w^T x^{(\ell)} + b \right) \geq 1 \; \forall \ell$$

  which is equivalent to solve

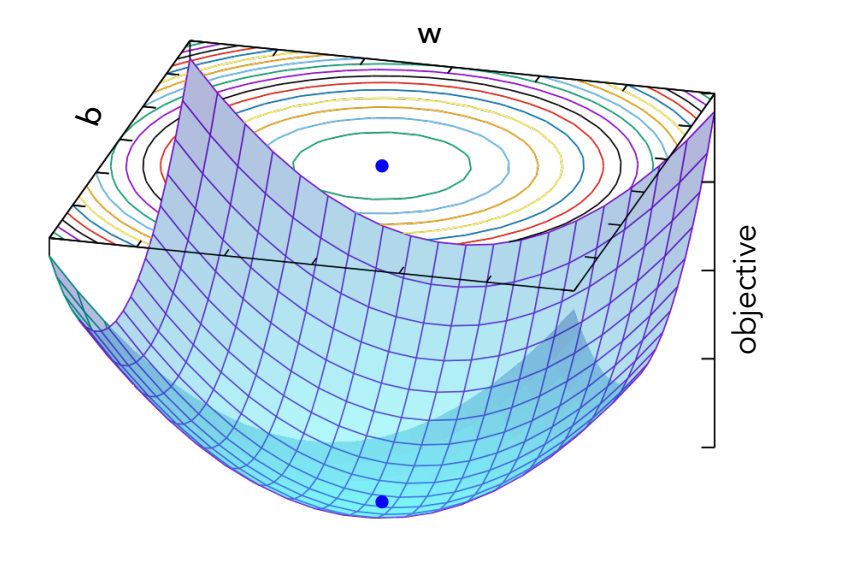
$$\min_{w,b} \frac{1}{2} \|w\|^2 \text{ s.t. } y^{(\ell)} \left(w^T x^{(\ell)} + b \right) \geq 1 \; \forall \ell$$

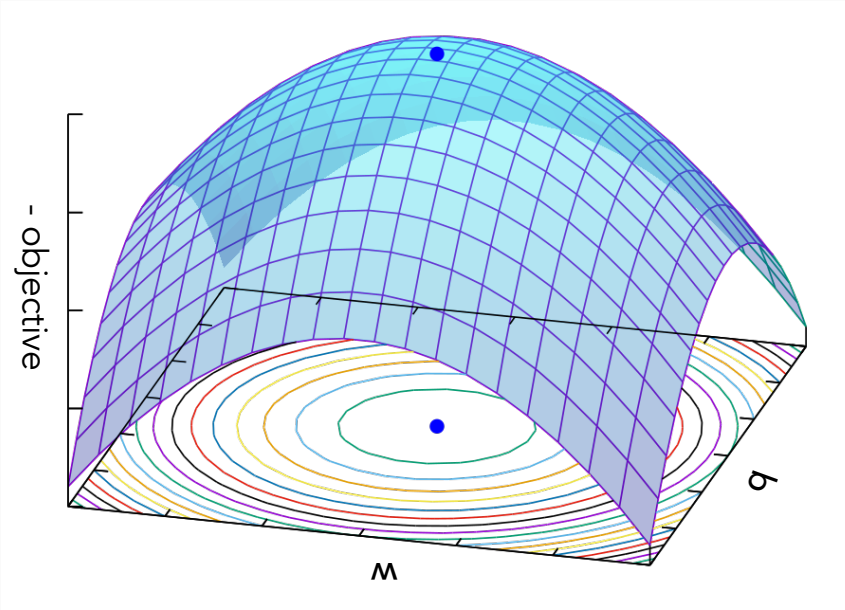
- The latter optimization problem (called Hard-Margin SVM) can be efficiently solved in a computer.

Standard SVM Formulation

⋆ Part of the success of SVMs in practice is due to the fact that they came with good software implementations.

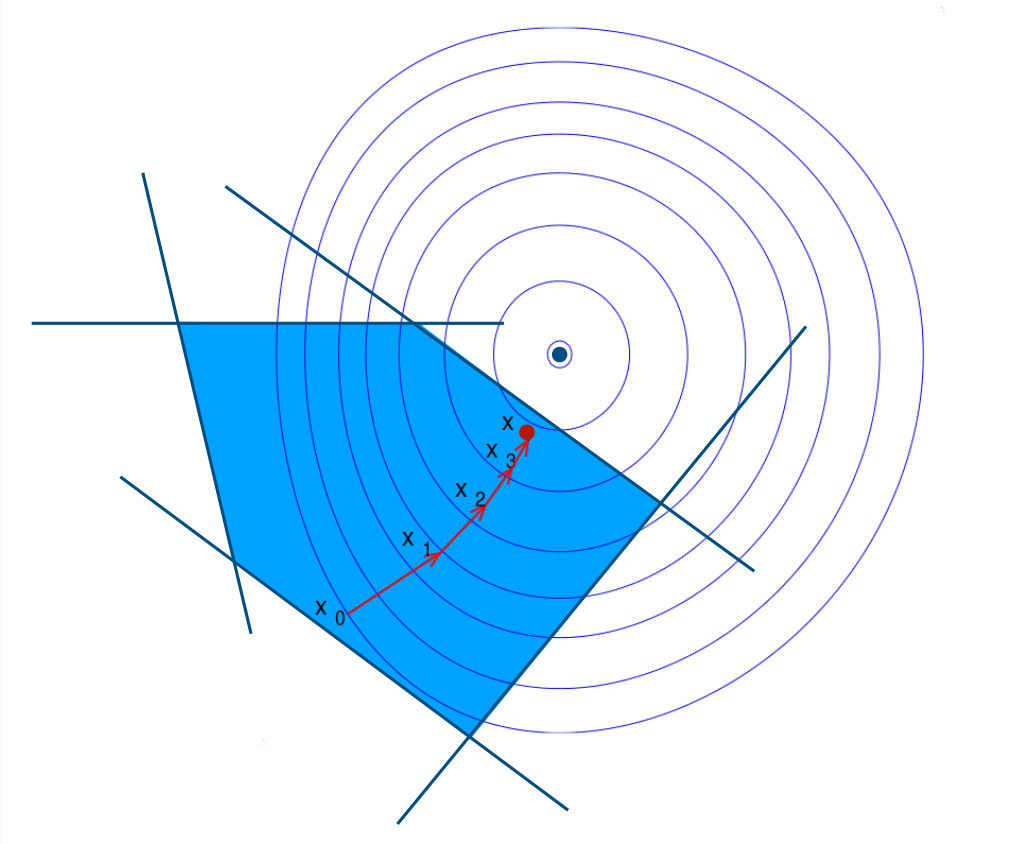
⋆ Objective function is quadratic, i.e., very easy!





Standard SVM Formulation

- ⋆ Constraints are linear, and define a convex feasible region.
- ⋆ Therefore, the COP is a *convex optimization problem*, which is the "easiest type of optimization problem for a machine".



Hard-Margin SVM Summary

- **Prepare the Data:** $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{N}$ such that $x^{(\ell)} \in \mathbb{R}^d$ and $y^{(\ell)} \in \{\pm 1\}$.
- **Training:** Find w, b by solving

$$\begin{aligned} &\min_{w,b} \ \frac{1}{2} \|w\|^2 \\ &\text{s.t. } y^{(\ell)} \left(w^T x^{(\ell)} + b \right) \geq 1 \ \forall \ell \end{aligned}$$

- **Test:** Classify/predict new data using the following decision function

$$f(x) = \text{sign}\left(w^T x + b \right)$$

Why Support Vector Machine?

- It can be shown that the parameter w of the maximum margin hyperplane can be written as a linear combination of the data pointss

$$w = \sum_{\ell} \alpha_\ell \, y^{(\ell)} \, x^{(\ell)} \, .$$

- Thus, also the decision function admits an expansion

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_{\ell} \alpha_\ell \, y^{(\ell)} \, \langle x^{(\ell)}, x \rangle + b\right) \, .$$

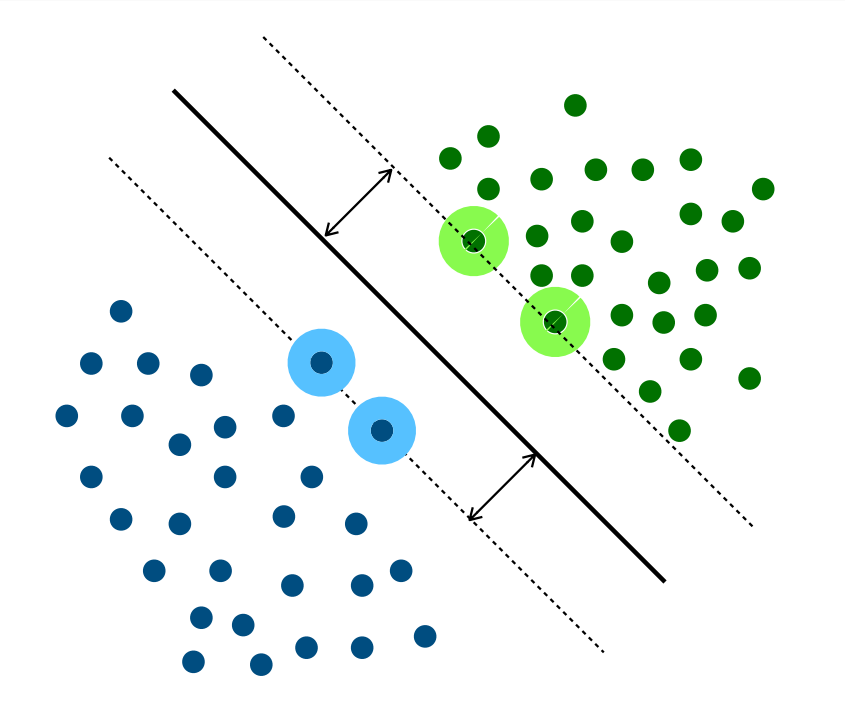
  where $\langle a, b \rangle$ denotes the dot product $a^T b$.

- Often, only a subset of training points has $\alpha_\ell \neq 0$. They are called the support vectors. Only these data points are relevant to take decisions or make predictions.

Why Support Vector Machine?

Geometrically, the support vectors are the training points nearest the boundary, i.e., the points determining the margin.

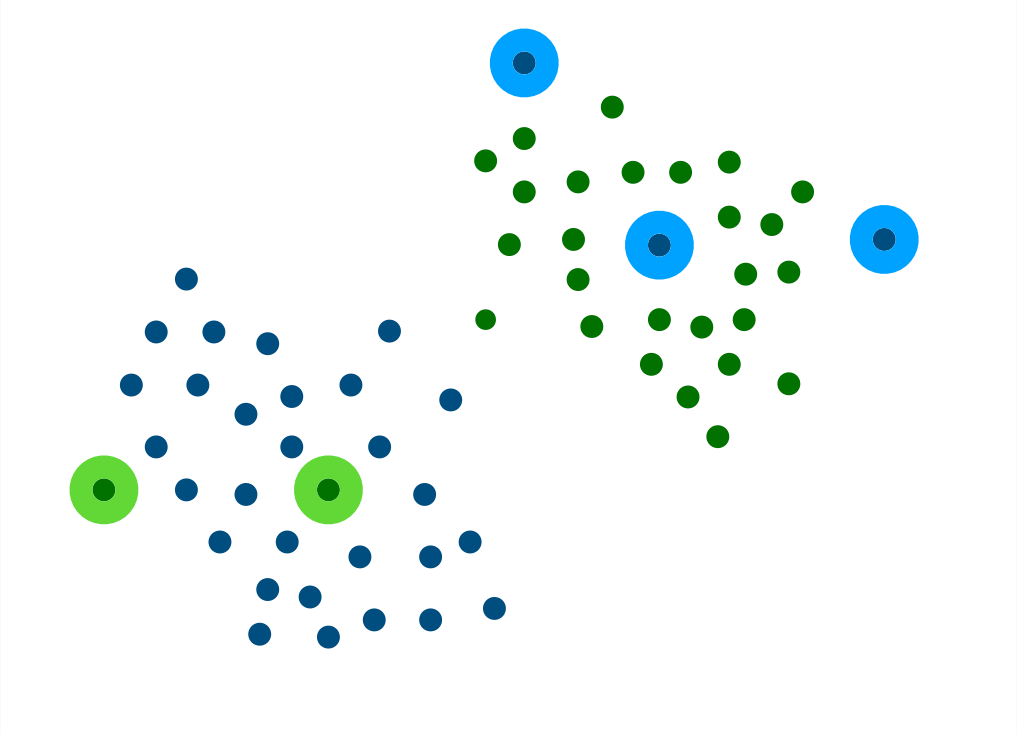
Any other point in the dataset is irrelevant to compute the classifier.



Soft Margins

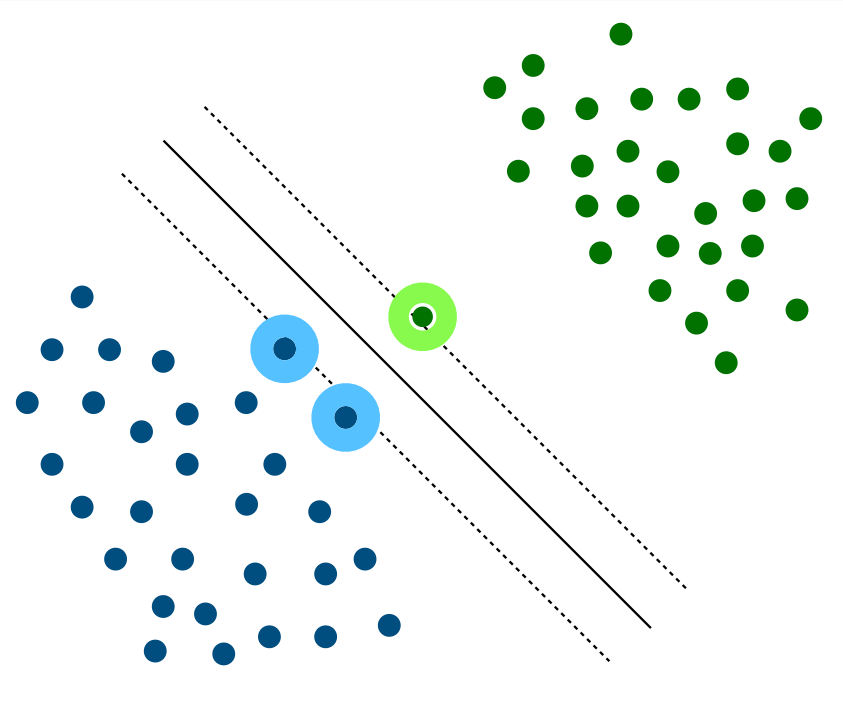
Limitations of Hard Margins

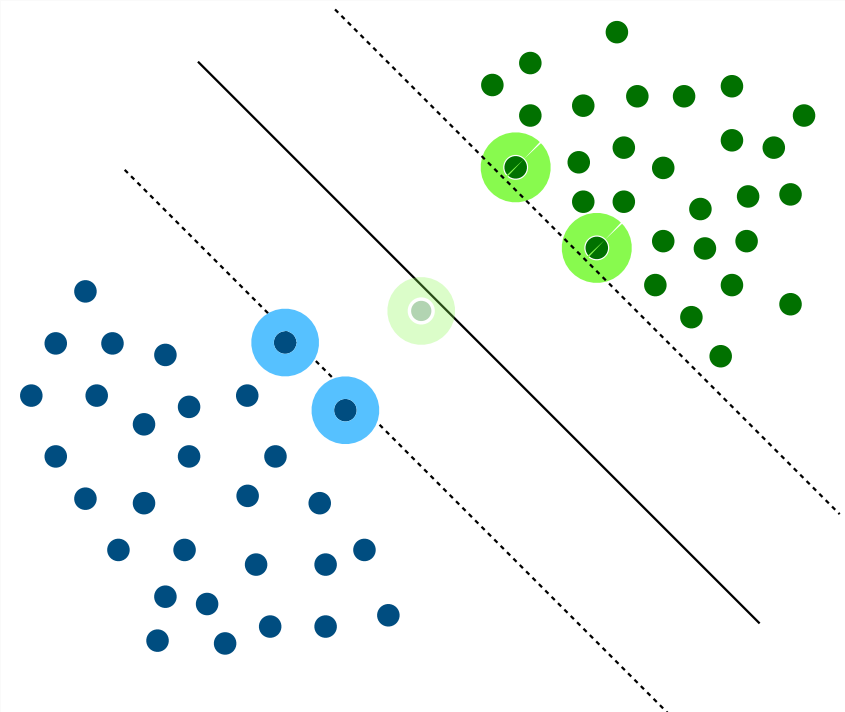
1. Our previous SVM works only for linearly separable data.



Limitations of Hard Margins

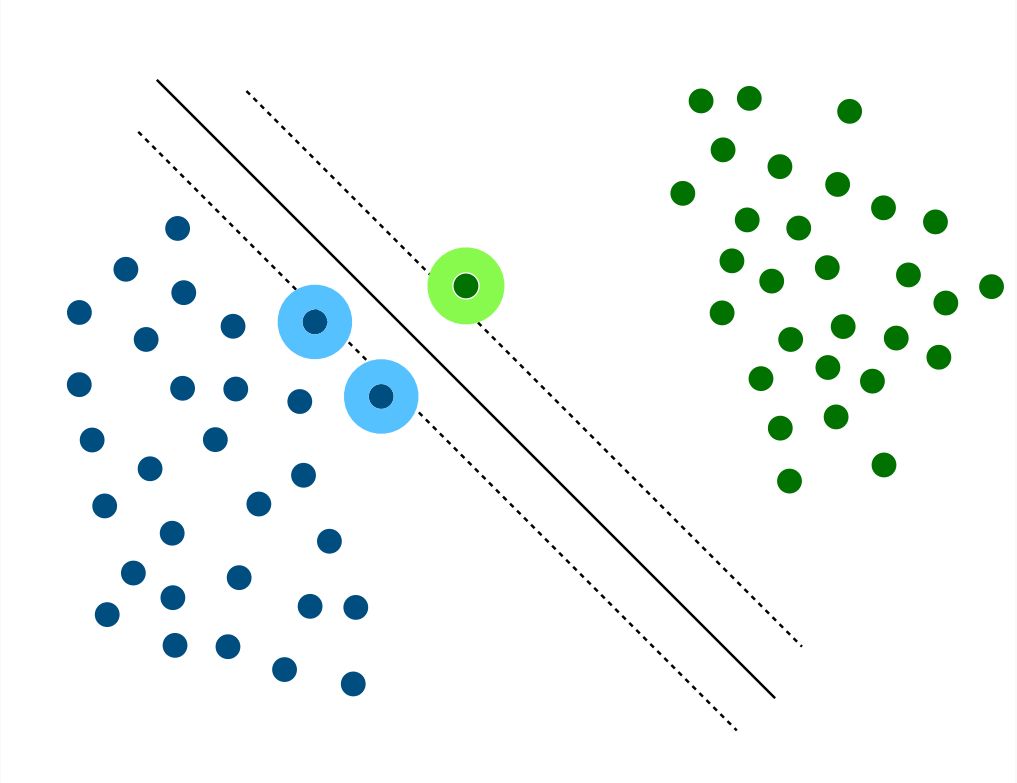
2. It is not really robust to changes in the training data, which suggests a risk of overfitting.





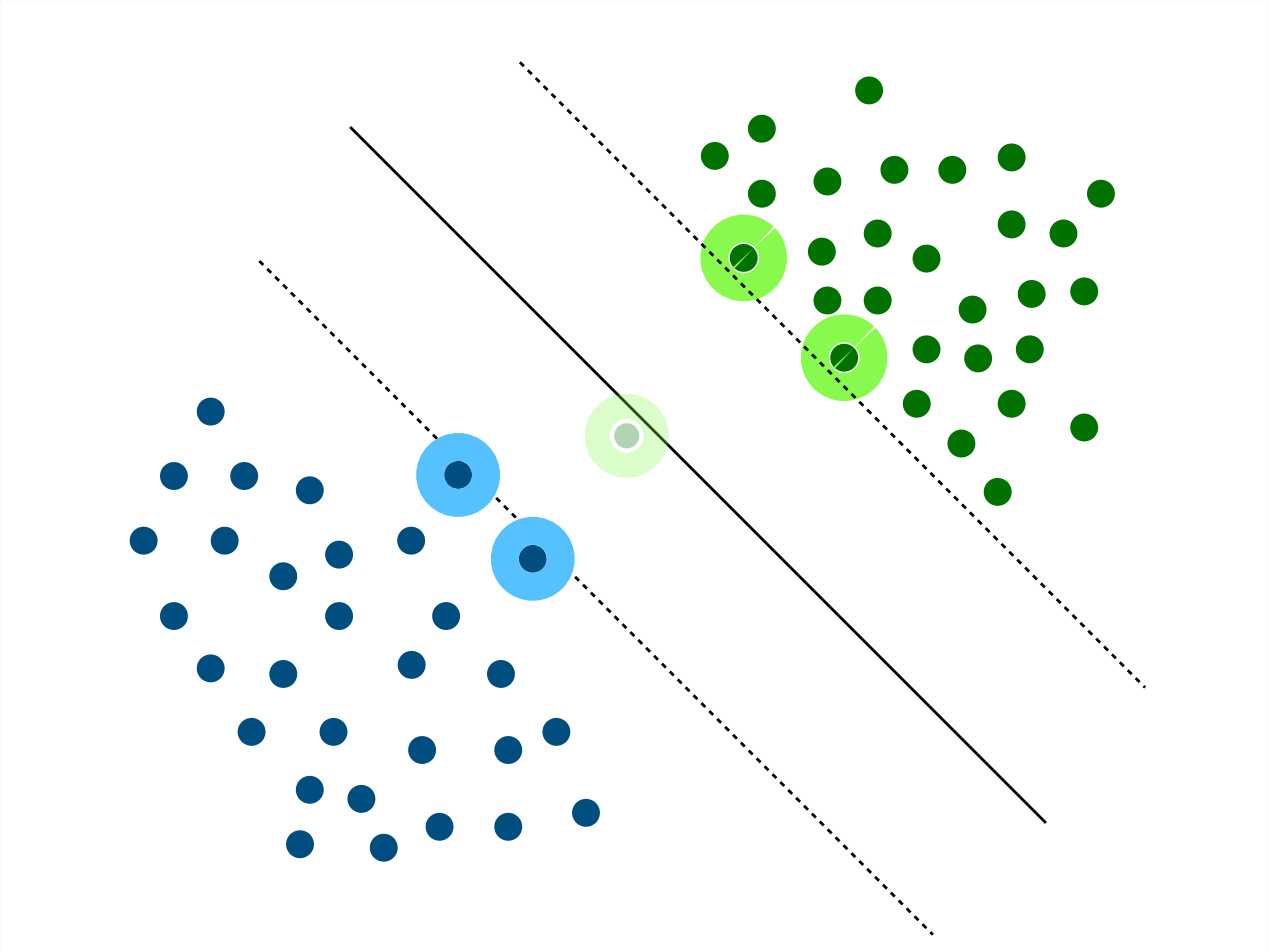
Limitations of Hard Margins

3. The classifier is determined by the support vectors only. It is not necessarily representative of the "general shape" of the class.



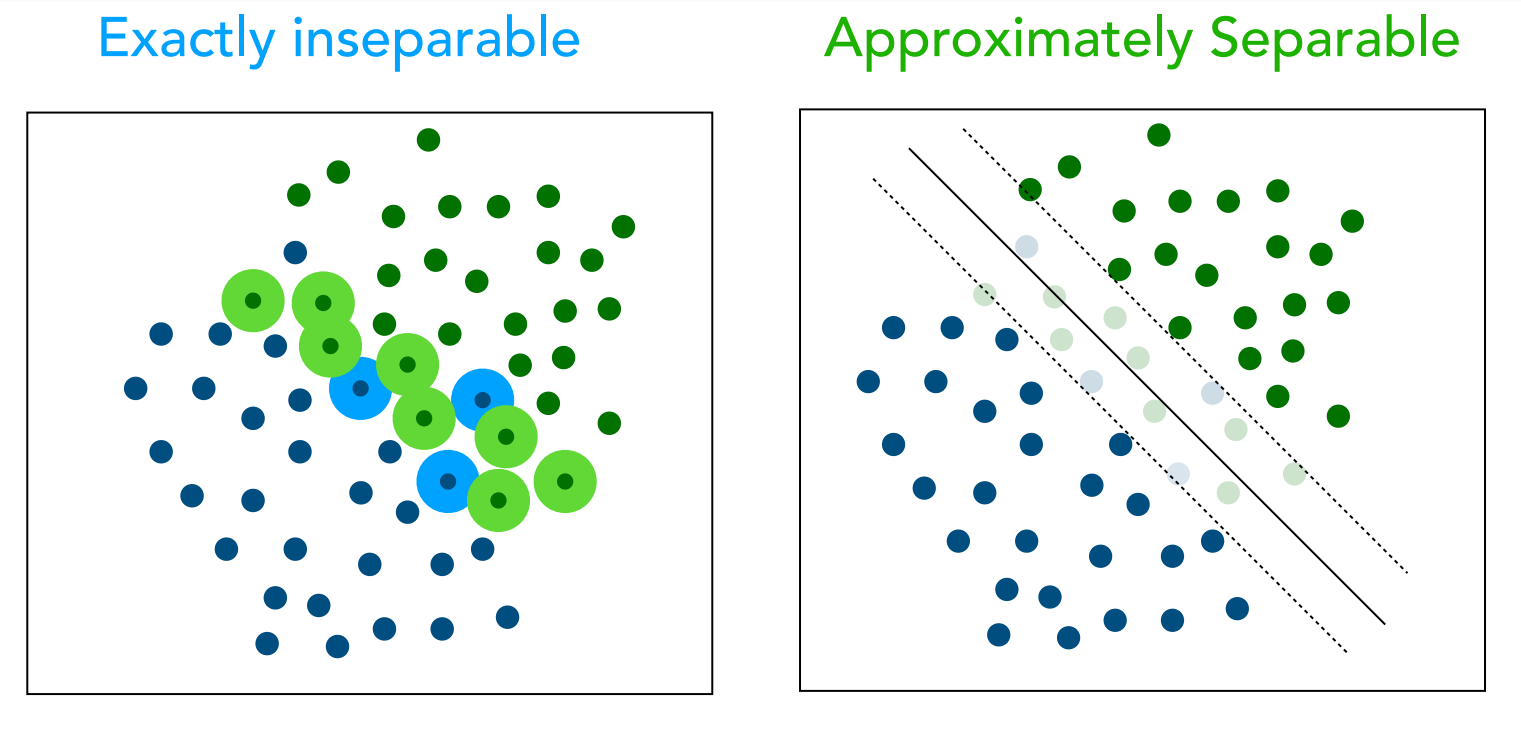
Soft Margins

Idea: Allow a part of the training data to be ignored in the computation of the margin.



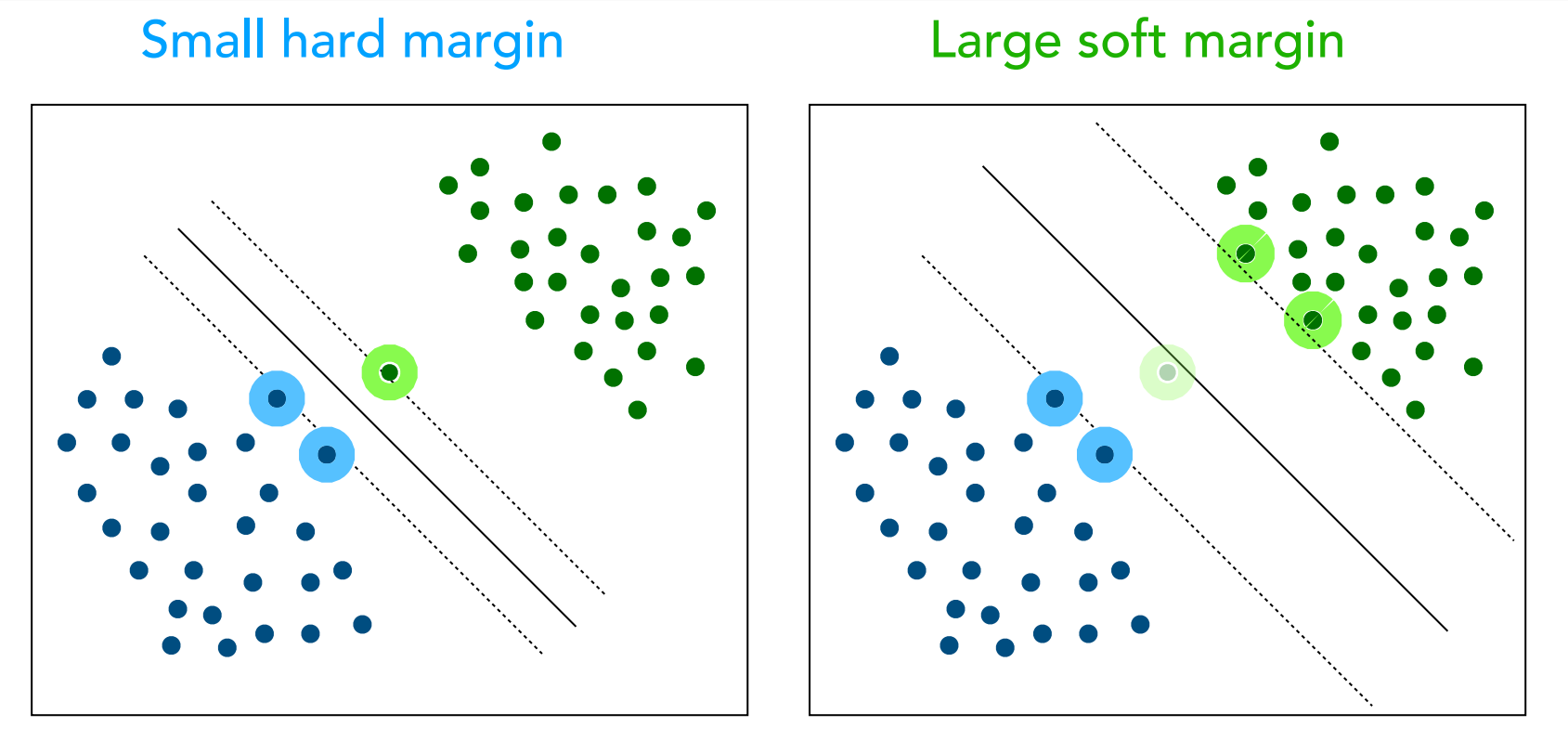
Opportunities of Soft Margins

1. Ignoring some points allows to handle (slightly) inseparable sproblems.



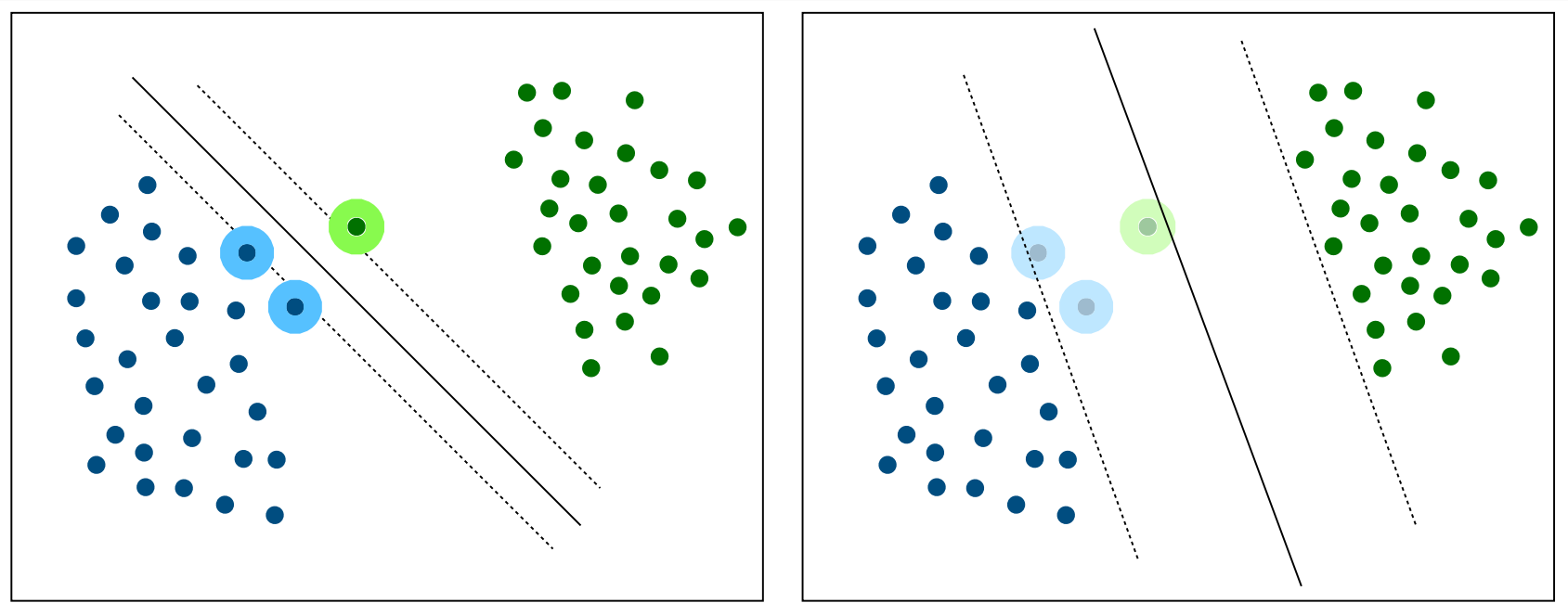
Opportunities of Soft Margins

2. Ignoring some points may allow to obtain approximate, but larger, margins.



Opportunities of Soft Margins

3. Ignoring some points may allow to follow better the geometry of the dataset.



Opportunities of Soft Margins

Large soft margins are intuitively better than small hard margins. Again, this intuition can be demonstrated to be right! Let

- **Train Error**(ρ) be the fraction of points that has to be "ignored" to achieve a margin ρ.
- **Test Error**(ρ) be the probability that the soft hyperplane misclassifies a future (novel) data point.

Theorem*

$$\textbf{Test Error}(\rho) \leq \textbf{Train Error}(\rho) + \text{cte} \cdot \sqrt{\frac{\frac{R^2}{\rho^2} \ln m + \ln \frac{1}{\delta}}{n}},$$

* Zhang, T. (2002). *Covering Number Bounds of Certain Regularized Linear Function Classes*. Journal of Machine Learning Research, 2, 527-550.

Soft Margins & Test Error

Note that now there is a tradeoff between the fraction of points you ignore and the size of the obtained margin.

$$\textbf{Test Error}(\rho) \leq \textbf{Train Error}(\rho) + \text{cte} \cdot \sqrt{\frac{\frac{R^2}{\rho^2} \ln m + \ln \frac{1}{\delta}}{n}},$$

- "Ignoring" points increases the first term of the bound but can reduce the second (because it allows to find a larger margin ρ).
- One needs to carefully balance between these two terms in order to obtain the classifier with the best prediction error.

Soft-Margin SVM

These new ideas can be easily incorporated into the optimization problem.

- The new formulation takes the form

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C \cdot \sum\nolimits_{\ell} \xi_\ell \\ \text{subject to } & y^{(\ell)}(w^T x^{(\ell)} + b) \geq 1 - \xi_\ell \ \forall \ell \\ & \xi_\ell \geq 0 \ \forall \ell \,, \end{aligned} \tag{7}$$

  and it is known as *soft-margin SVM* (more specifically, C-SVM).
- The new variables ξ_ℓ are called "slack variables". They are introduced to allow violations in the original margin constraints.

Interpretation of the New Constraints

The constraint involving the example $x^{(\ell)}$ is:

$$y^{(\ell)}(w^T x^{(\ell)} + b) \geq 1 - \xi_\ell$$

- If $\xi_\ell = 0$, the example $x^{(\ell)}$ satisfies the hard-margin constraint.
- If $\xi_\ell > 0$, the example $x^{(\ell)}$ violates the hard-margin constraint but it is correctly classified. Its is partially ignored by the machine.
- If $\xi_\ell > 1$, the example $x^{(\ell)}$ violates the hard-margin constraint and is also misclassified by the hyperplane. Its is totally ignored by the machine.

Regularization Parameter C

In the new formulation,

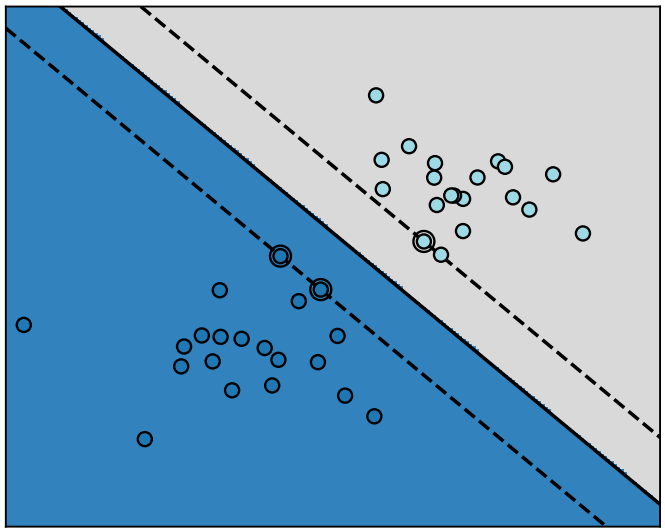
$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + \boldsymbol{C} \cdot \sum_\ell \xi_\ell \\ \text{subject to } & y^{(\ell)}(w^T x^{(\ell)} + b) \geq 1 - \xi_\ell \ \forall \ell \\ & \xi_\ell \geq 0 \ \forall \ell \,, \end{aligned} \tag{8}$$

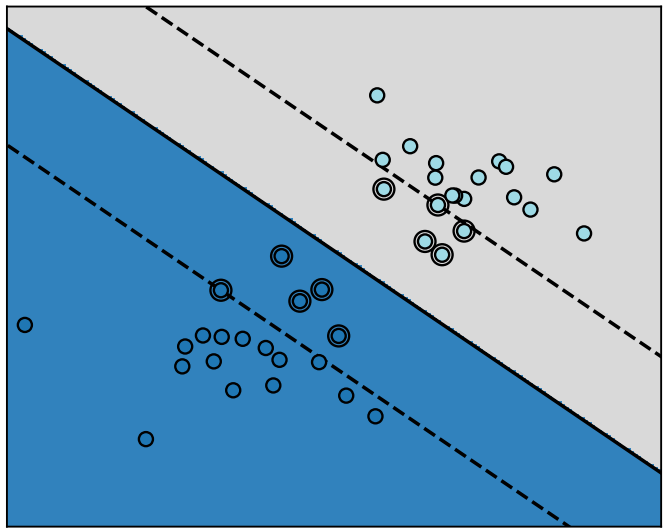
parameter C is known as the *regularization parameter*.

This parameter allows to tradeoff between maximizing the margin (first term in the objective) and satisfying the constraints (second term in the objective).

Role of Parameter C

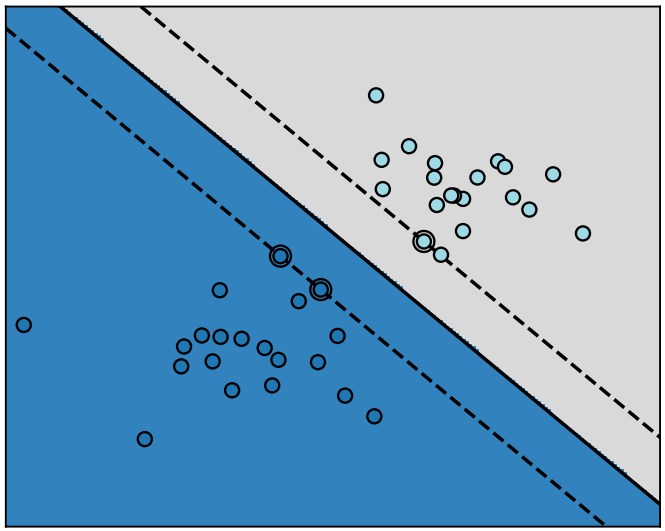
A small value of C allow many points to violate the margin constraint. It focus more on *margin maximization*. Many points may be misclassified by the model. Below: a real example using *scikit-learn*.

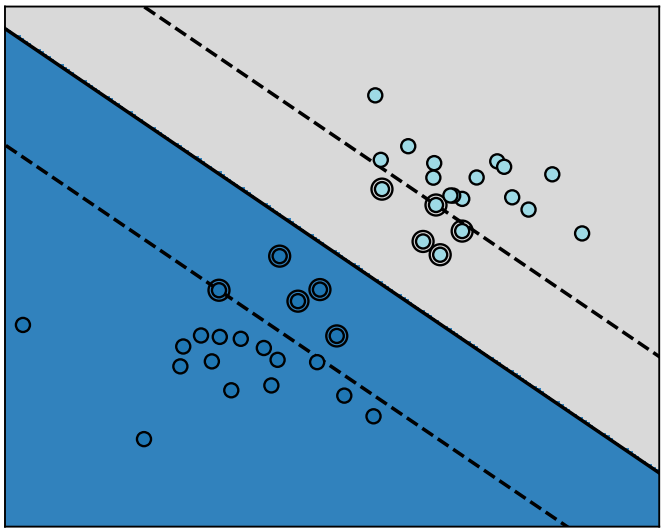




Role of Parameter C

A larger value of C allow less points to violate the margin constraint. It focus more on the *training error*. It is less likely that the model misclassifies training examples. Below: a real example using *scikit-learn*.





Soft-Margin $\succ$ Hard-Margin

Note that using $C = \infty$ (or a very big value) reduces the soft-margin SVM to the hard-margin case.

Indeed, with $C = \infty$ it is so costly to violate the margin constraints that you obtain a solution where $\xi_\ell = 0 \, \forall \ell$.

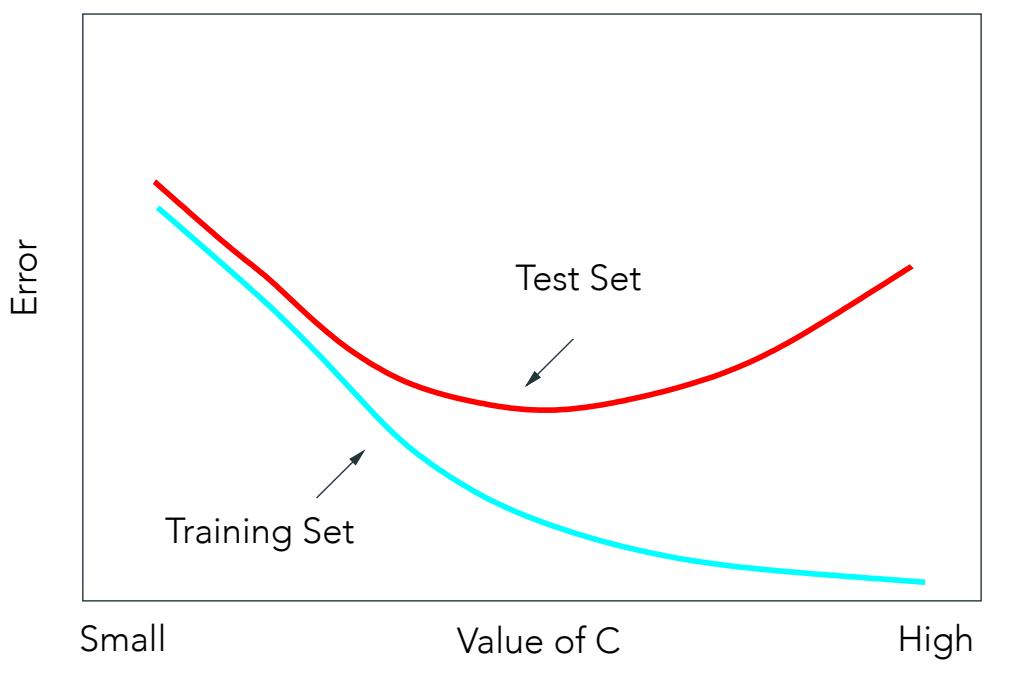
$$\begin{aligned} \min_{w,b,\xi} \; & \frac{1}{2} \|w\|^2 + \boldsymbol{C} \cdot \sum\nolimits_\ell \xi_\ell \\ \text{subject to } & y^{(\ell)}(w^T x^{(\ell)} + b) \geq 1 - \xi_\ell \, \forall \ell \\ & \xi_\ell \geq 0 \, \forall \ell \, , \end{aligned} \tag{9}$$

Thus, *it makes no sense to choose the value of C to minimize the training error*. $C = \infty$ is always the optimal choice for minimizing the training error.

Choosing the Value of C

Choosing the value of C allows to reduce the *prediction/test error*.

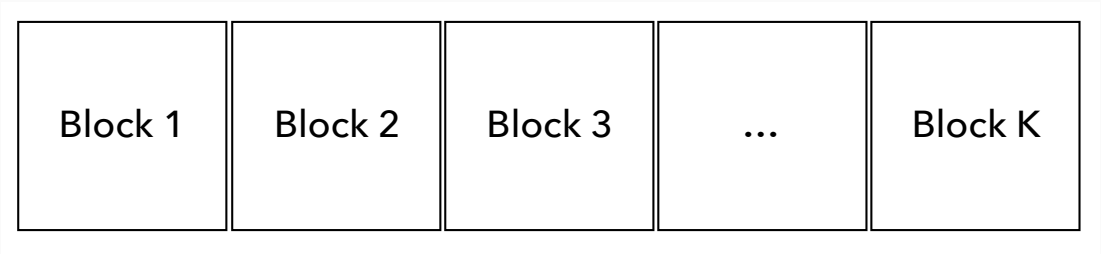
- If C is too small, you observe overfitting (small training error and high test error).
- If C is too large, you observe underfitting (high training error and high test error).



Choosing the Value of C

To determine if some value of C is the optimal, you need to estimate the test prediction/test error of the SVM.

Typically, you use a validation set or K-fold cross-validation (we will apply this in the lab).



Warning Regarding Libraries

In some libraries, the objective

$$\min \frac{1}{2} \|w\|^2 + C \cdot \sum_\ell \xi_\ell$$

is re-written as

$$\min \sum_\ell \xi_\ell + C \cdot \|w\|^2$$

In that case, the interpretation of C is the opposite!

Support Vectors in the Soft-Margin Case

- In the soft-margin case, you still have the linear expansion in terms of the support vectors

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_{\ell} \alpha_\ell \, y^{(\ell)} \, \langle x^{(\ell)}, x \rangle + b\right) .$$

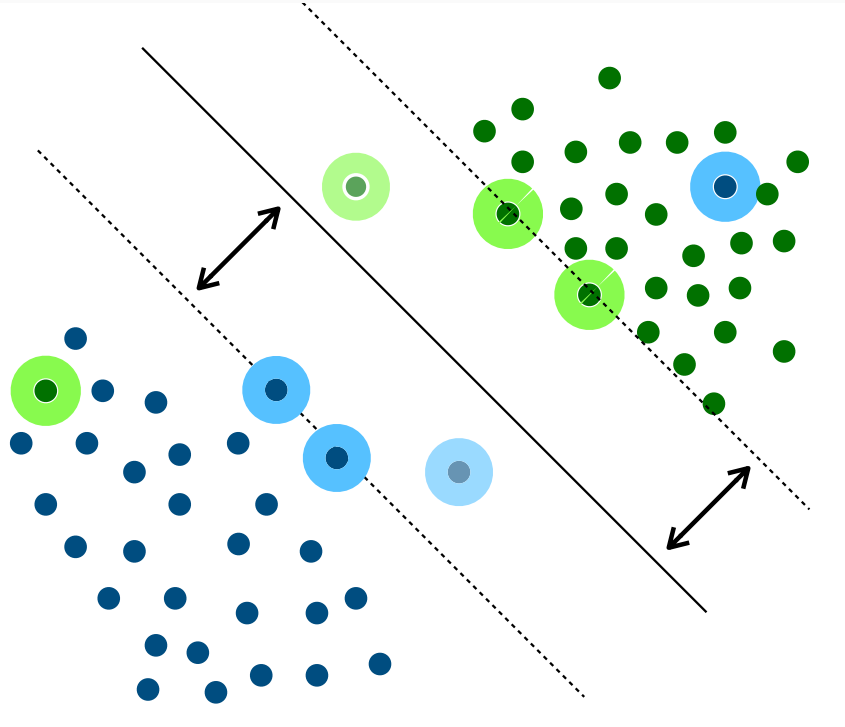
  where $\langle a, b \rangle$ denotes the dot product $a^T b$.

- Only the support vectors are relevant to take decisions or make predictions.

Support Vectors in the Soft-Margin Case

Geometrically, the soft-margin case create two types of support vector:

- The "hard" support vectors for which $w^T x^{(\ell)} + b = 1$.
- The "soft" support vectors for which $w^T x^{(\ell)} + b = 1 - \xi_i$ with $\xi_i \neq 0$.



Soft-Margin SVM Summary

- **Prepare your Data:** $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{N}$ such that $x^{(\ell)} \in \mathbb{R}^d$ and $y^{(\ell)} \in \{\pm 1\}$.
- **Training:** For a fixed value of C, find w, b by solving

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + \boldsymbol{C} \cdot \sum_{\ell} \xi_\ell \\ \text{subject to } & y^{(\ell)}(w^T x^{(\ell)} + b) \geq 1 - \xi_\ell \ \forall \ell \\ & \xi_\ell \geq 0 \ \forall \ell\,, \end{aligned}$$

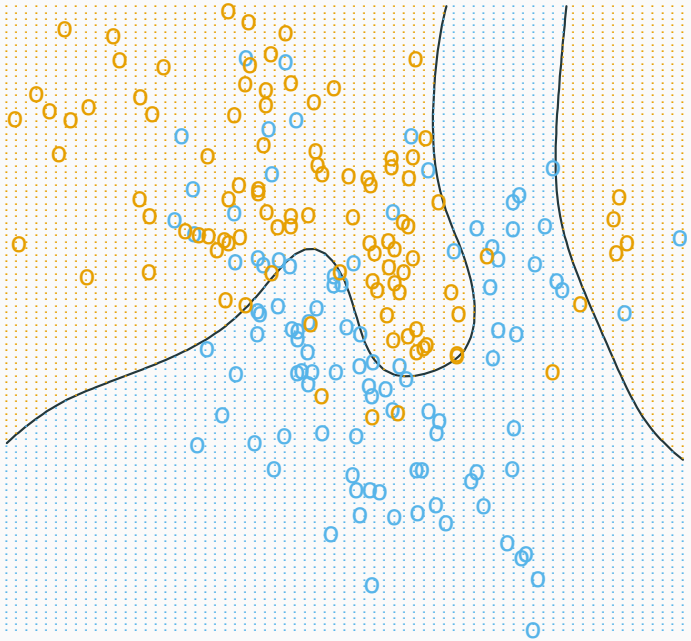
- **Model Selection:** Choose the best value of C using K-fold cross-validation or another model selection technique.
- **Test:** Classify/predict new data as usual

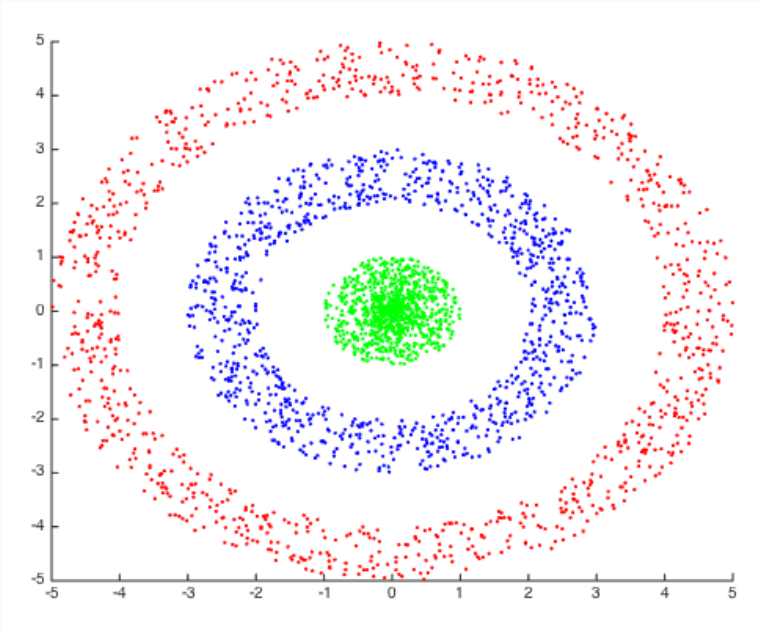
$$f(x) = \text{sign}\left(w^T x + b\right)$$

Kernels

Non Linearly Separable Problems

- The previous SVMs separate the classes using a linear function.
- These models work well for linearly separable problems or "almost" linearly separable problems.
- What about non-linear problems?

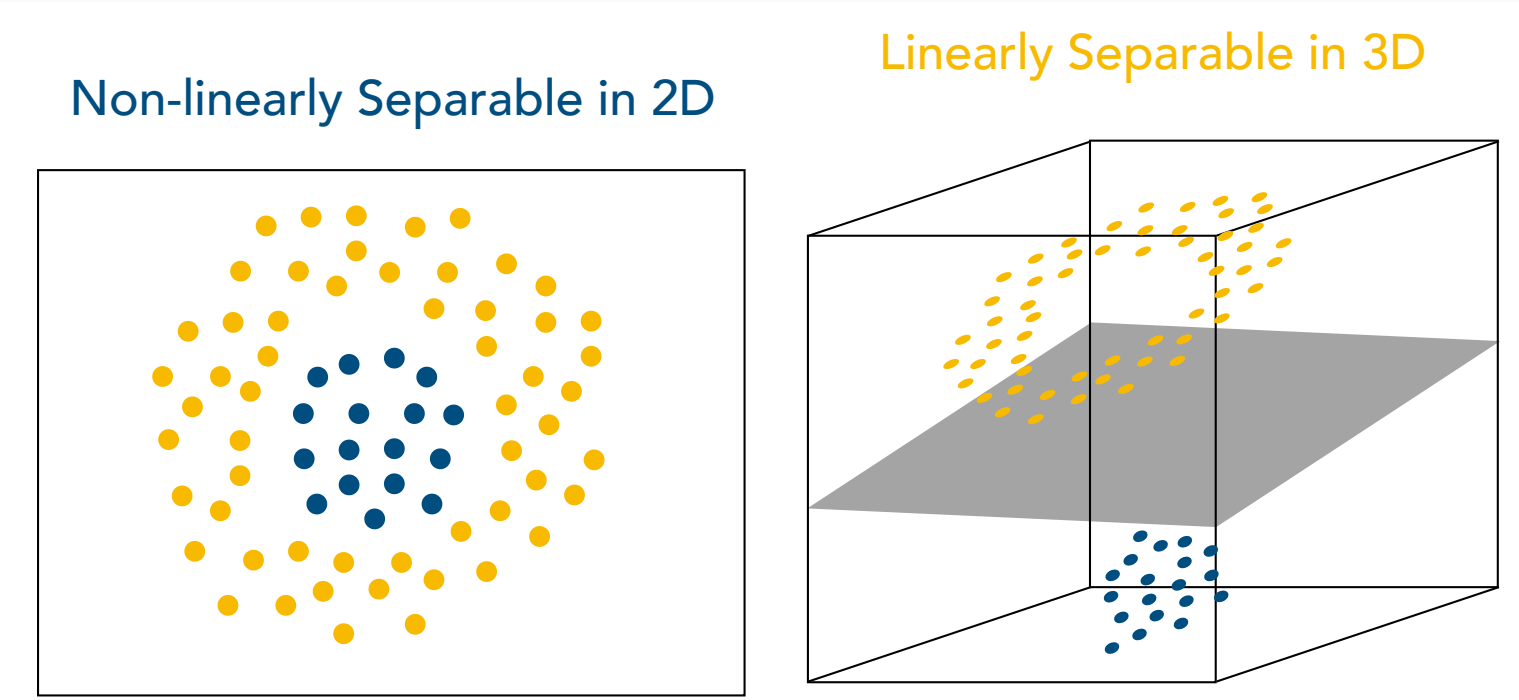




Representation Trick

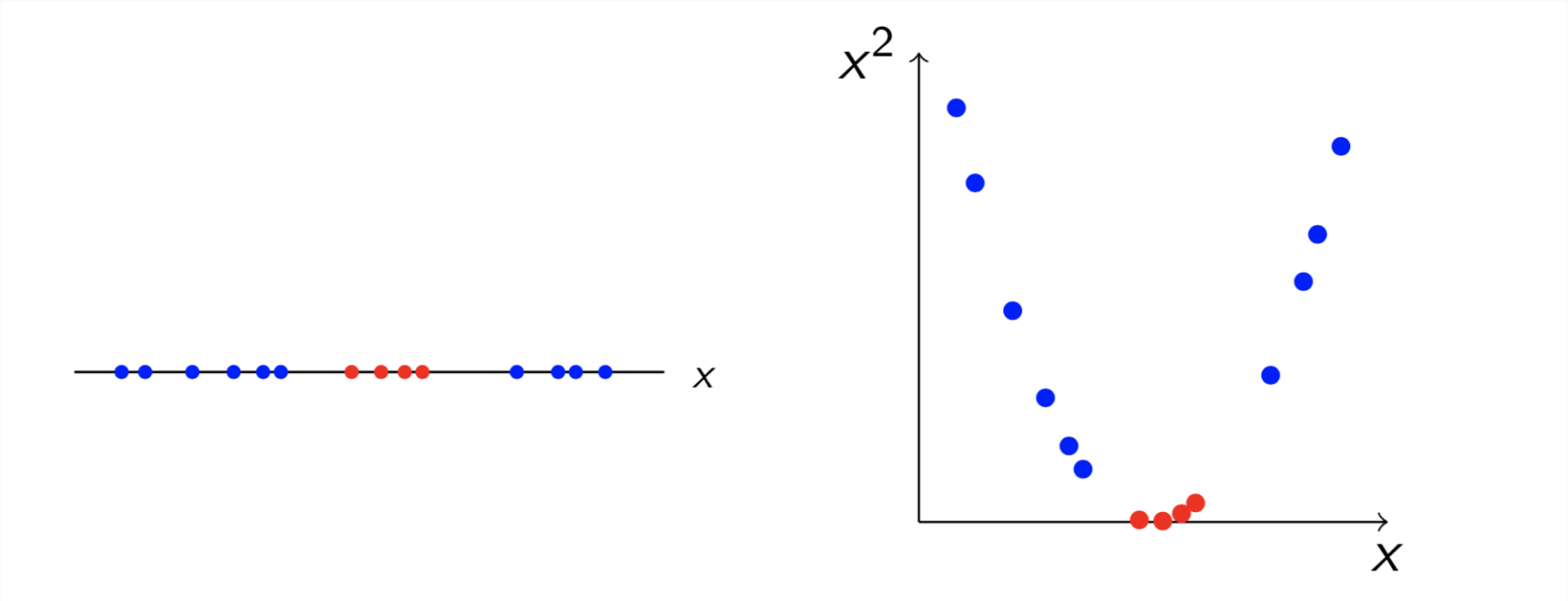
Idea: A problem that looks non-linearly separable can become linearly separable if you *change the representation of your data*, i.e., the set of attributes used to describe the input patterns.

Often, adding new attributes (new axes to the feature space) can make your dataset linearly separable.



Representation Trick

Idea: Often, adding new attributes (new axes to the feature space) can make your dataset linearly separable.



Cover's Theorem

This intuition can be demonstrated to be right.

Let D the number of attributes used to represent x.

Teorema[1]

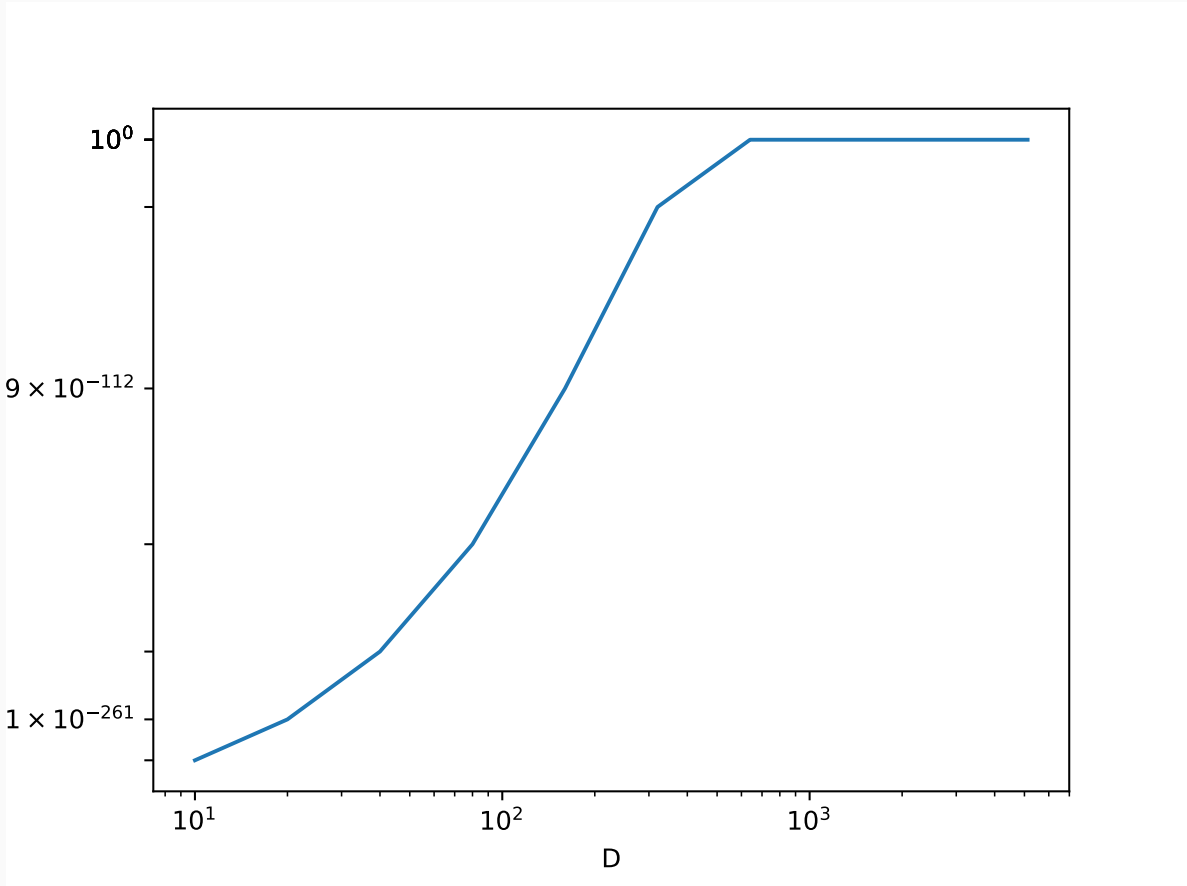
The probability that a dataset of n points is linearly separable is given by

$$P(n, D) = \frac{1}{2^{n-1}} \sum_{k=0}^{D-1} \binom{n-1}{k}.$$

[1] T. Cover (1965). *Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition*. IEEE Transactions on Electronic Computers. Page 330.

Cover's Theorem

• Cover's Theorem for $n = 1000$ points. In the horizontal axis, we have the dimensionality D of the representation. In the vertical axis the probability of the dataset being linearly separable.



SVM in the New Space

So, instead of learning the hyperplane with the original data,

- We can first change the representation, introducing a high-dimensional transformation $\phi : \mathbb{X} \to \mathbb{Z}$.
- We do not build the SVM in $\mathbb{X}$ but in the new feature space $\mathbb{Z}$

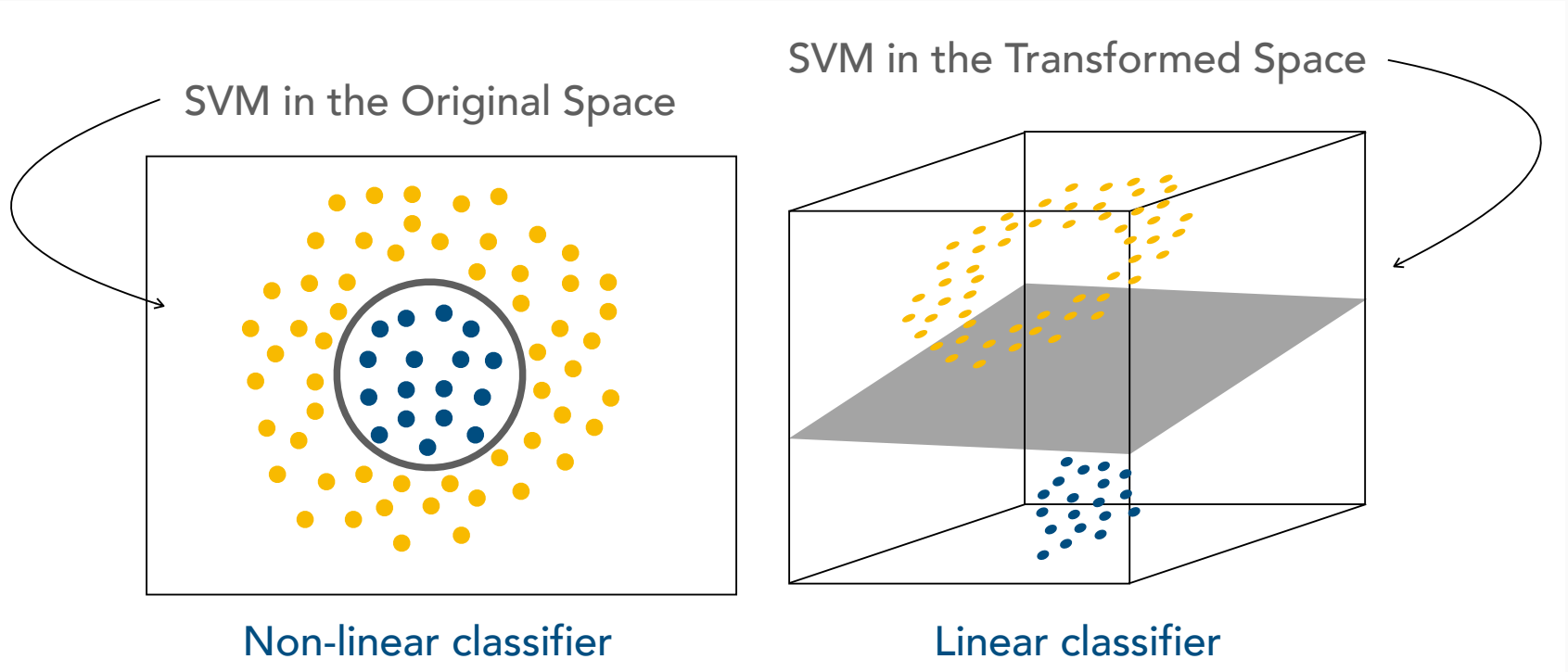
$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C \cdot \sum_\ell \xi_\ell \\ \text{s.t. } & y^{(\ell)}(w^T \boldsymbol{\phi}(\boldsymbol{x}^{(\ell)}) + b) \geq 1 - \xi_\ell \, \forall \ell \\ & \xi_\ell \geq 0 \, \forall \ell \end{aligned}$$

- We classify/predict new data by first transforming the input pattern

$$f(x) = \text{sign}\left(w^T \boldsymbol{\phi}(\boldsymbol{x}) + b\right)$$

SVM in the Original Space

Using a non-linear transformation $\phi(x)$, your classifier is linear in the new space, but is non-linear in the original space.



Kernel Trick

Limitations of the Representation Trick:

- How to choose/design the new representation?
- How to handle the computational complexity (time & memory) arising from the use of some many attributes.

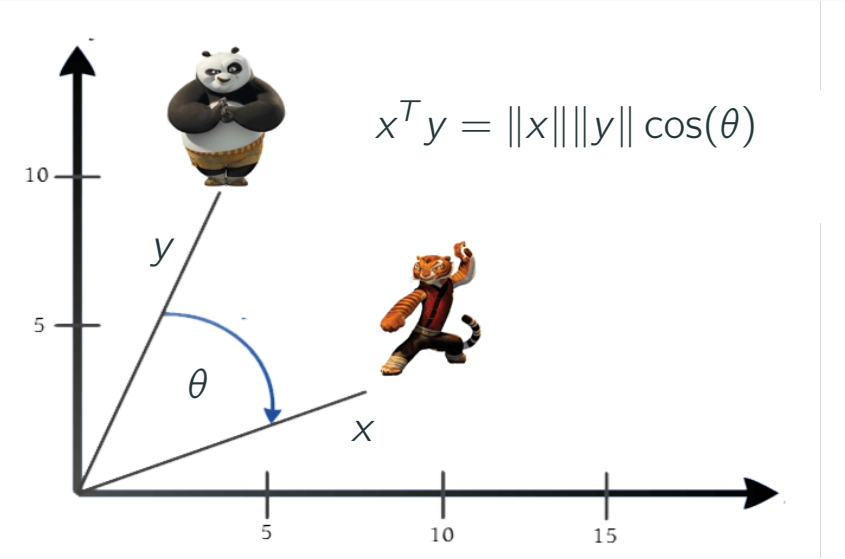
Alternative: Instead of changing the representation you can change the geometry of your computations.

- For instance, your decision function is linear because you use dot products to compare the input pattern with the support vectors:

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_{\ell} \alpha_\ell \, y^{(\ell)} \, \langle \boldsymbol{x}^{(\ell)}, \boldsymbol{x} \rangle + b\right) .$$

Kernel Trick

The dot product used in the decision function can be understood as a similarity function used to compare data.



Kernel Trick

What if we change the dot products $\langle x^{(\ell)}, x \rangle$ in

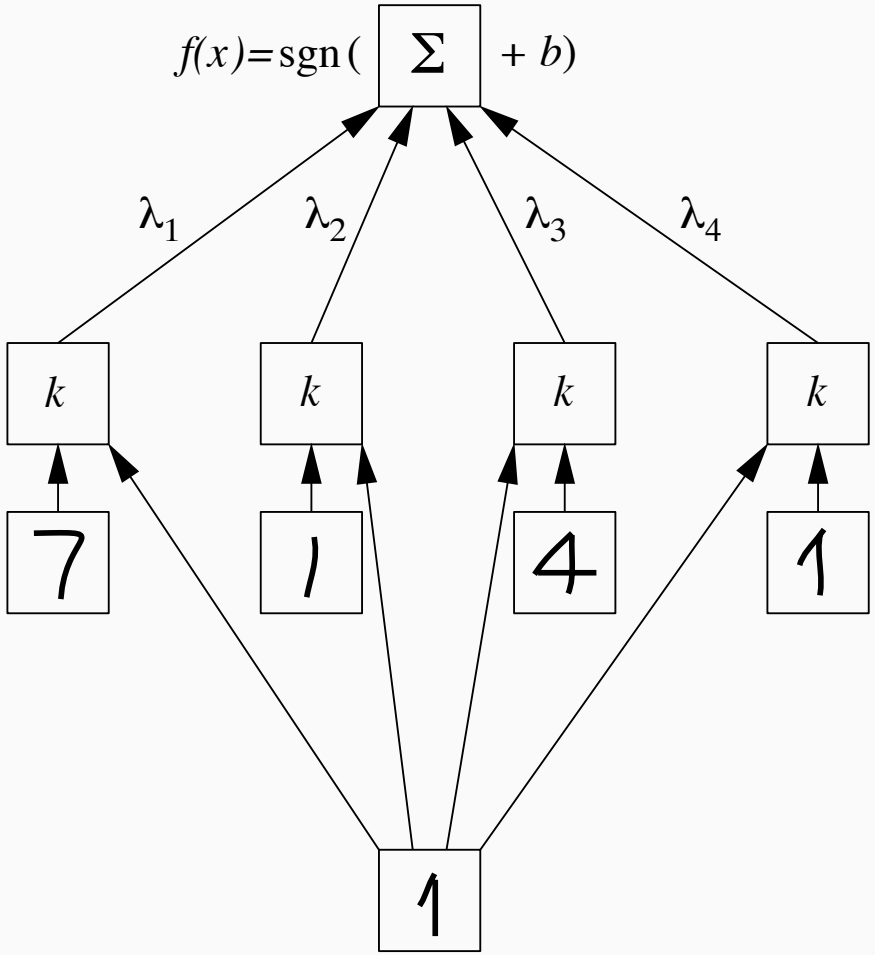
$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_\ell \alpha_\ell \, y^{(\ell)} \, \langle \boldsymbol{x^{(\ell)}, x} \rangle + b\right) .$$

by some other operation $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, e.g. a non-linear one?

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_\ell \alpha_\ell \, y^{(\ell)} \, \boldsymbol{k(x^{(\ell)}, x)} + b\right) .$$

Intuitively, we are modifying the way by which our classifier compares the pattern to predict with the support vectors.

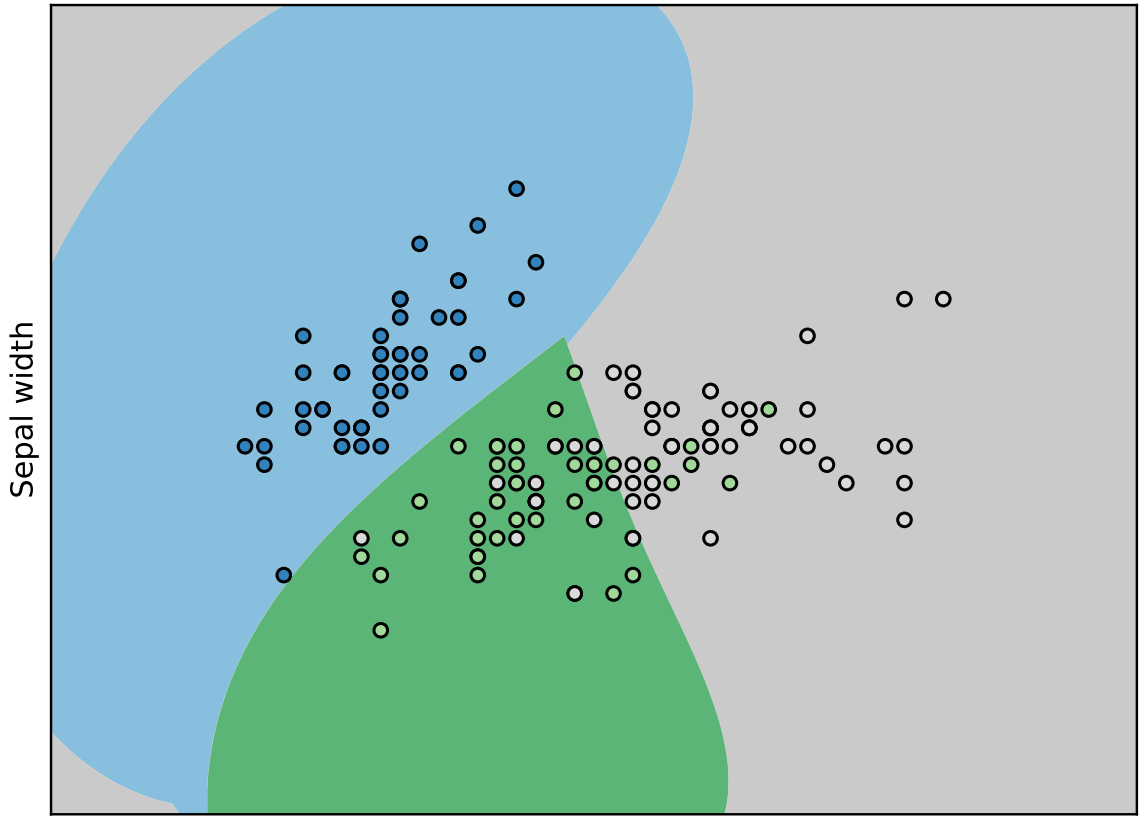
Kernel Trick Intuition



Clearly, after the substitution, **the classifier is no longer linear** ...

Kernel Trick Effect

For instance, using the function $k(x, x') = \exp(-0.1 \|x - x'\|^2)$ we obtain classification boundaries of the type below



Admissible Kernels

Do we have a valid hyperplane classifier after the substitution?

A function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is an **admissible kernel** if there exist a transformation $\phi : \mathbb{X} \to \mathbb{Z}$ such that

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \ \forall x, x' \in \mathbb{X}.$$

In that case, computing

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_\ell \alpha_\ell \, y^{(\ell)} \, \boldsymbol{k(x^{(\ell)}, x)} + b\right),$$

is the same that computing

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_\ell \alpha_\ell \, y^{(\ell)} \, \boldsymbol{\langle \phi(x^{(\ell)}), \phi(x) \rangle} + b\right).$$

Admissible Kernels

Rephrasing: **If k is an admissible kernel, using the decision function**

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum_\ell \alpha_\ell \, y^{(\ell)} \, \boldsymbol{k(x^{(\ell)}, x)} + b\right) .$$

is equivalent to (1) transform the data using $\phi : \mathbb{X} \to \mathbb{Z}$ and (2) build an ordinary/linear SVM in $\mathbb{Z}$.

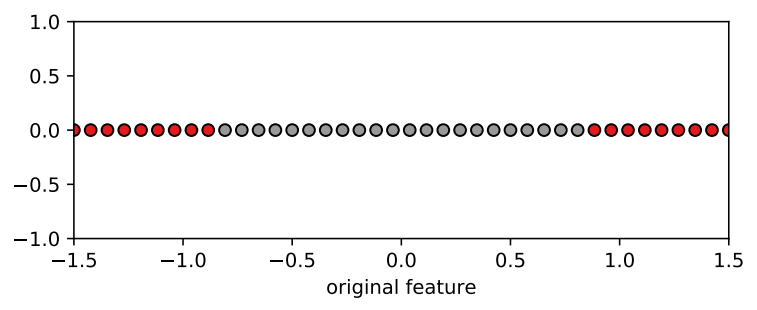
Often the transformation corresponding to a valid kernel is quite high-dimensional, as we wanted.

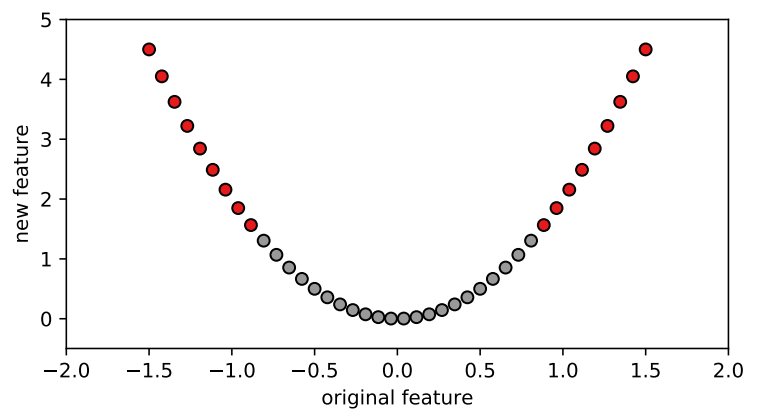
However we do not need to explicitly compute and handle the transformation.

Example: Polynomial Features

The polynomial transformation of degree p is an old approach to build non-linear models.

It consists in creating new features by using the powers of the original features up to degree p.





Example: Polynomial Features

- For $x \in \mathbb{R}$, and $p = 2$

$$\phi(x) = \left(1,\ x,\ x^2\right)^T \quad \text{(2 features)}$$

- For $x \in \mathbb{R}$, and $p = 3$

$$\phi(x) = \left(1,\ x,\ x^2, x^3\right)^T \quad \text{(3 features)}$$

- For $x \in \mathbb{R}^2$ and $p = 2$

$$\phi(x) = \left(1,\ x_1,\ x_2,\ x_1^2,\ x_1 \cdot x_2,\ x_2^2\right)^T \quad (\approx p^2 = 4 \text{ features})$$

- For $x \in \mathbb{R}^d$ and arbitrary p the new representation has $\approx p^d$ features. (e.g. $d = 100$ and $p = 5$ lead to 10 billion attributes).
- **Problem:** Computing this map for large p or d is prohibitively expensive. In addition you need to store it!

Example: Polynomial Kernel

- What if we use a polynomial kernel?

$$k(x, x') = (\langle x, x' \rangle + c)^p$$

- For instance, for $d = 2$, $p = 2$ and $c = 1$,

$$k(x, x') = x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2\, x_1 x'_1 x_2 x'_2 + 2\, x_1 x'_1 + 2\, x_2 x'_2 + 1\,.$$

- It is not hard to see that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ with

$$\phi(x) = \left(x_1^2,\, x_2^2,\, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2,\, 1\right)^T\,.$$

- **Conclusion:** The polynomial kernel automatically generates polynomial features without the need to explicitly compute them.

Example: Poly Kernel versus Poly Transformation

- In general, computing the polynomial kernel

$$k(x, x') = (\langle x, x' \rangle + c)^p$$

  costs only $\approx d$ floating point operations (flops).
- The transformation has $\approx p^d$ features. Thus, computing $\phi(x)^T \phi(x')$ costs $\approx p^d$ flops.
- **For $d = 100$ and $p = 5$, computing the kernel costs ≈ 100 flops, but computing the transformation costs ≈ 10 billion flops.**
- In addition, you do not have to store $\phi(x)$ in memory.

How to Choose the Kernel?

- Custom kernel (domain knowledge) or general purpose kernel.
- To be technically consistent, k has to be an admissible/valid kernel.
- Many theorems give conditions these kernels should verify. E.g.

Theorem (Mercer)

A kernel is admissible if and only if it is positive definite, that is, for any dataset $\{x^{(\ell)}\}_{\ell=1}^{n}$ the $n \times n$ matrix $\boldsymbol{K}$ defined as $\boldsymbol{K}_{ij} = k(x^{(i)}, x^{(j)})$ is positive semi-definite.

$$\boldsymbol{K} = \begin{bmatrix} k(x^{(1)}, x^{(1)}) & k(x^{(1)}, x^{(2)}) & \dots & k(x^{(1)}, x^{(n)}) \\ k(x^{(2)}, x^{(1)}) & k(x^{(2)}, x^{(2)}) & \dots & k(x^{(2)}, x^{(n)}) \\ \dots & & & \\ k(x^{(n)}, x^{(1)}) & k(x^{(n)}, x^{(2)}) & \dots & k(x^{(n)}, x^{(n)}) \end{bmatrix} \succeq 0$$

Custom Kernels

- Kernels can be defined for an specific application, treating them as similarity functions.
- Example: String kernels for Text Classification and Bioinformatics.

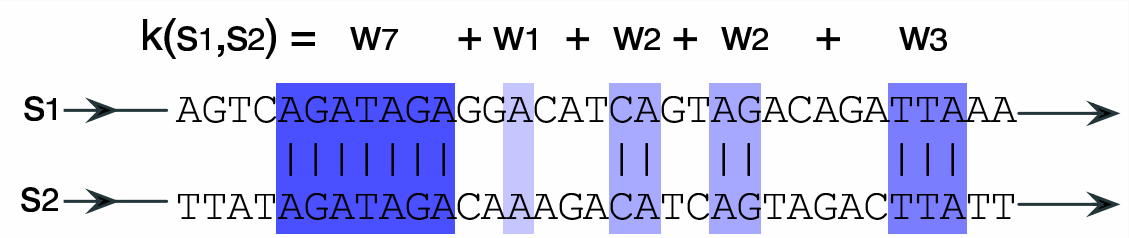


Figure 4.1: Given two sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ of equal length, the kernel consists of a weighted sum to which each match in the sequences makes a contribution w_B depending on its length B, where longer matches contribute more significantly.

General-Purpose Kernels

Assuming data represented as vectors ($\mathbb{X} \subset \mathbb{R}^d$).

- The *linear kernel* $k(x, x') = x^T x'$.
- The *polynomial kernel*, with parameters γ (scale), c_0 and p

$$k(x, x') = (\gamma \cdot x^T x' + c_0)^p \,.$$

- The *RBF (or Gaussian) kernel*, with parameter γ (scale)

$$k(x, x') = \exp(-\gamma \, \|x - x'\|^2) \,.$$

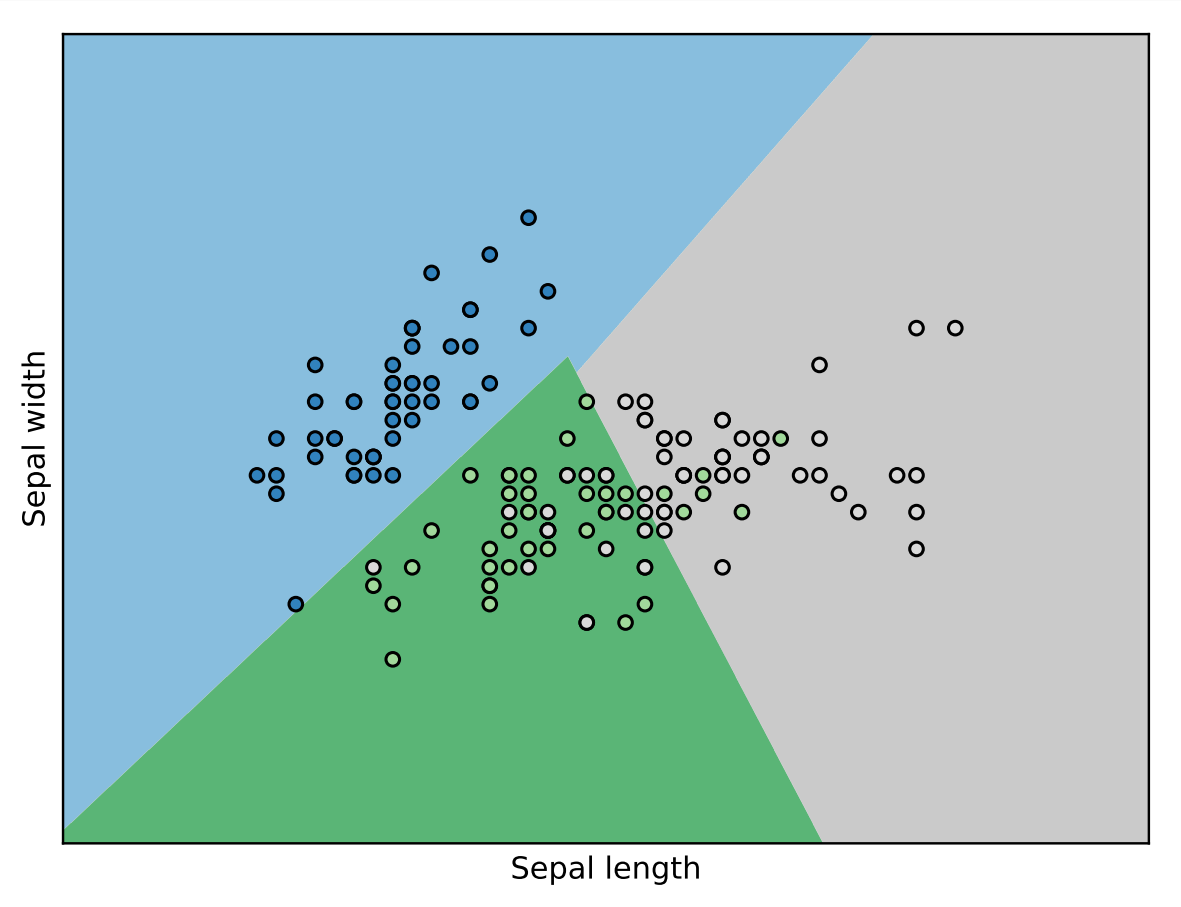
- The *Laplacian kernel*, with parameter γ (scale)

$$k(x, x') = \exp(-\gamma \, \|x - x'\|_{l1}) \,.$$

where $\|x - x'\|_{l1} = \sum_j |x_j - x'_j|$.

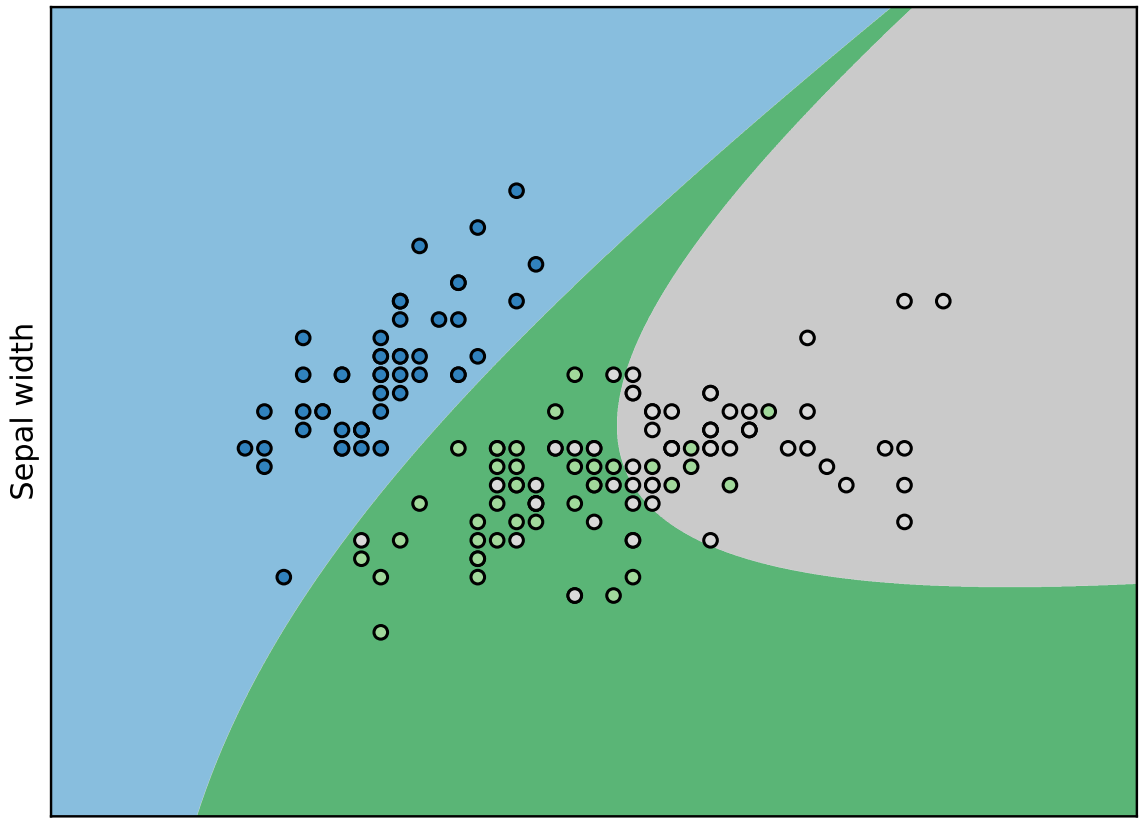
Effect of the Kernel

Example of an SVM implemented with a *linear kernel* on the Iris dataset (first two attributes).



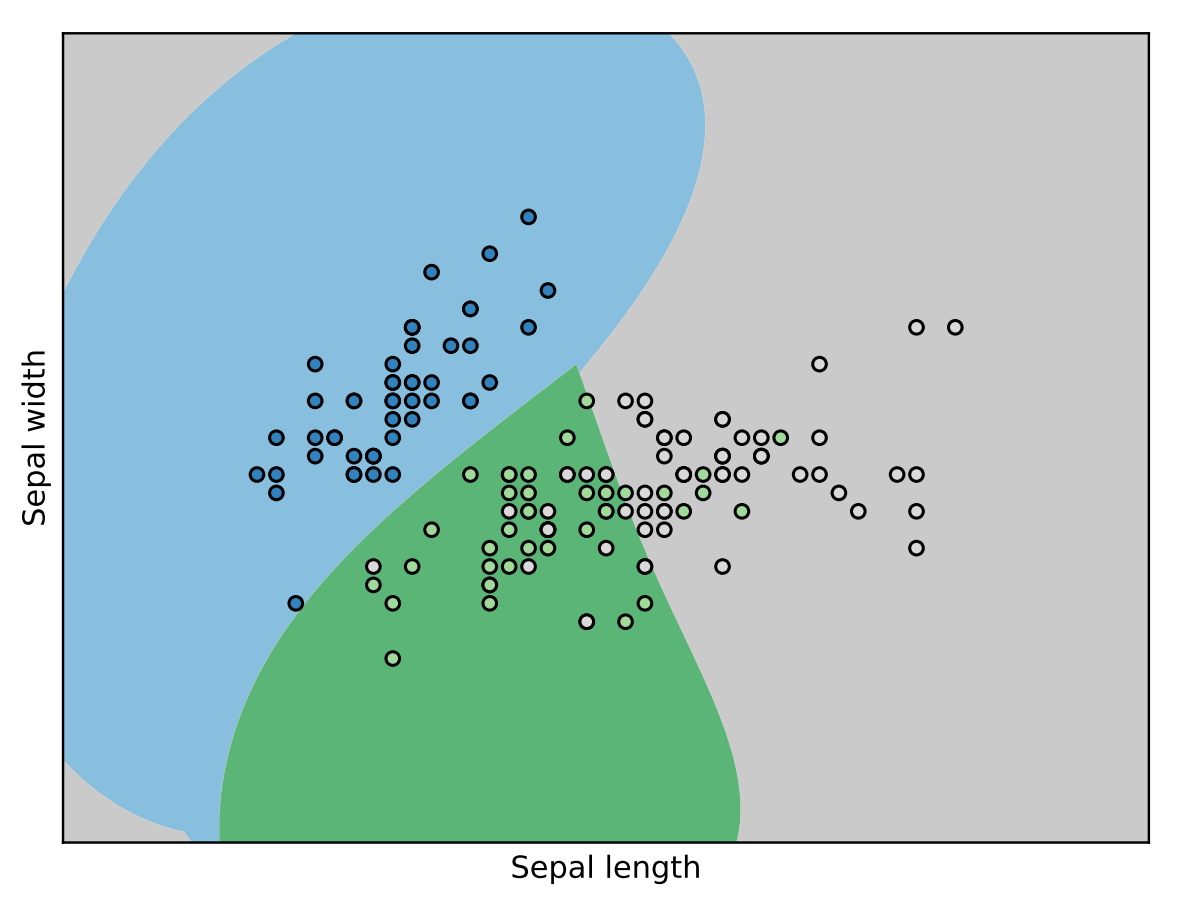
Effect of the Kernel

Example of an SVM implemented with a *polynomial kernel* ($p = 3, c = 1, \gamma = 0.5$) on the same dataset.



Effect of the Kernel

Example of an SVM implemented with a *gaussian kernel* ($\gamma = 0.5$) on the same dataset.



More Kernels

- The *sigmoid kernel* with parameters γ (scale) and c_0

$$k(x, x') = \tanh(\gamma \cdot x^T x' + c_0)\,,$$

  is not an admissible kernel but is sometimes used.
- Kernels can be combined using simple operations (sum, element-wise product, etc).
- Kernels can be also learnt from data.

Training with Kernels

We have focused on making the SVM non-linear by modifying the prediction function

$$f(x) = \text{sign}\left(\sum_{\ell} \alpha_\ell^* y^{(\ell)} \underbrace{k(x^{(\ell)}, x)}_{\langle \phi(x^{(\ell)}), \phi(x) \rangle} + b\right)$$

However, you also need to train the model consistenty!

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C \cdot \sum\nolimits_{\ell} \xi_\ell \\ \text{s.t. } & y^{(\ell)}(w^T \phi(x^{(\ell)}) + b) \geq 1 - \xi_\ell \, \forall \ell \\ & \xi_\ell \geq 0 \, \forall \ell \end{aligned}$$

How do you put kernels here??

Training with Kernels

Fortunately, math helps. It can be shown that solving the SVM problem

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C \cdot \sum_\ell \xi_\ell \\ \text{s.t. } & y^{(\ell)}(w^T \phi(x^{(\ell)}) + b) \geq 1 - \xi_\ell \, \forall \ell \\ & \xi_\ell \geq 0 \, \forall \ell \end{aligned}$$

is equivalent to solve an alternative problem, known as the *dual*

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \ & \sum_{i=1}^{n} \boldsymbol{\alpha}_\ell - \frac{1}{2} \sum_{i,j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^{(i)} y^{(j)} \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \\ \text{s.t. } & \sum_i \boldsymbol{\alpha}_i y^{(i)} = 0, \ 0 \leq \boldsymbol{\alpha}_i \leq C \, \forall i \, . \end{aligned} \tag{10}$$

The original (primal) optimization variables w, b can be retrieved from new (dual) optimization variables $\boldsymbol{\alpha}$.

Primal-Dual Equations

If you solve the dual, you can recover w and b using the following equations

$$w = \sum_{\ell} \alpha_\ell \, y^{(\ell)} x^{(\ell)} \qquad (11)$$
$$b = |B|^{-1} \sum_{\ell : B} \left(y^{(\ell)} - w^T x^{(\ell)} \right) ,$$

with $B : \{\ell : 0 < \alpha_\ell < C\}$.

Summary: SVMs with Kernels

- Kernelization: Choose a valid kernel function k.
- Model Selection & Training: Choose a value of C and solve

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_\ell - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)})$$
$$\text{s.t.} \quad \sum\nolimits_i \alpha_i y^{(i)} = 0, \; 0 \leq \alpha_i \leq C \, \forall i \, ,$$

  and get b from the primal-dual equations.

- Testing: Classify/predict new data as

$$f(x) = \text{sign}\left(w^T x + b\right) = \text{sign}\left(\sum\nolimits_\ell \alpha_\ell \, y^{(\ell)} k(x^{(\ell)}, x) + b\right)$$

Practical Issues

Data Preparation

- To obtain good results in practice, you need to scale your numerical attributes. Two methods often work well:
  - Feature-wise normalization: Each attribute is centered to have 0 mean and standard deviation 1.
  - Instance-wise normalization: Each training vector is scaled to have (Euclidean) norm 1.

	age	color	food	height	score	state
Jane	30	blue	Steak	165	4.6	NY
Nick	2	green	Lamb	70	8.3	TX
Aaron	12	red	Mango	120	9.0	FL
Penelope	4	white	Apple	80	3.3	AL
Dean	32	gray	Cheese	180	1.8	AK
Christina	33	black	Melon	172	9.5	TX
Cornelia	69	red	Beans	150	2.2	TX

Data Preparation

- If your training set is $S = \{(x^{(\ell)}, y^{(\ell)})\}_{\ell=1}^{n}$, with $x^{(\ell)} \in \mathbb{R}^d$, the mean and variance of each attribute across the dataset are:

$$\mu_i = \frac{1}{n}\sum_{\ell} x_i^{(\ell)}, \quad \sigma_i^2 = \frac{1}{n-1}\sum_{\ell}(x_i^{(\ell)} - \mu_i)^2 .$$

- Feature-wise normalization means applying the following transformation:

$$x_i^{(\ell)} \leftarrow \frac{(x_i^{(\ell)} - \mu_i)}{\sigma_i} \quad \begin{matrix} \forall i = 1, \ldots, d \\ \forall \ell = 1, \ldots, n . \end{matrix}$$

- Instance-wise normalization instead means applying the following transformation:

$$x^{(\ell)} \leftarrow \frac{x^{(\ell)}}{\|x^{(\ell)}\|} = \frac{x^{(\ell)}}{\sqrt{\sum_i (x_i^{(\ell)})^2}} \; \forall \ell = 1, \ldots, n .$$

Data Preparation

- As many other machine learning methods, SVMs do not provide "native support" for categorical attributes.
- However, the following approach works well in practice
  - If an attribute is ordinal (e.g. Bad, Average, Good), treat it as a numerical attribute, assigning consecutive integers to each value and preserving the order (e.g. Bad $= 0$, Average $= 1$, Good $= 2$).
  - If an attribute is nominal (e.g. Black, Blue, Green) use one-hot-vectors of dimensionality equal to the number of different values.

Black	1	0	0
Blue	0	1	0
Green	0	0	1

SVM Hyper-parameters

- SVM performance can heavily depend on the choice of its hyper-parameters:
  - Regularization parameter C, controlling the tradeoff between training error and margin maximization.
  - The kernel function k and its parameters (e.g. γ,p,c_0, etc).
- In contrast to the SVM parameters (w, b), hyper-parameters cannot be selected from the training set.
  - Optimal C would be $\approx \infty$ ($\Rightarrow$ training error $= 0$).
  - Degenerate kernel functions are optimal in the training set (e.g. RBF with $\gamma \approx \infty$) gives always training error $= 0$).
- **Hyper-parameter values are chosen to improve the prediction ability of the classifier.** Therefore, you need to measure in some way this prediction ability.

Model Selection versus Model Assessment

Two goals often confused

- **Model selection:** Estimating the prediction error of different models to choose among them. For example
  - SVM + gaussian kernel or SVM + linear kernel?
  - Linear SVM with $C = 1$ or Linear SVM with $C = 100$?
- **Model assessment:** Estimating the prediction error of the model you have already chosen.
- Model assessment has to be done *after* model selection.
- The error you estimated for a model during model selection is often an over-optimistic estimate of its real prediction error. The reason is that your model has been optimized for the selection metric.

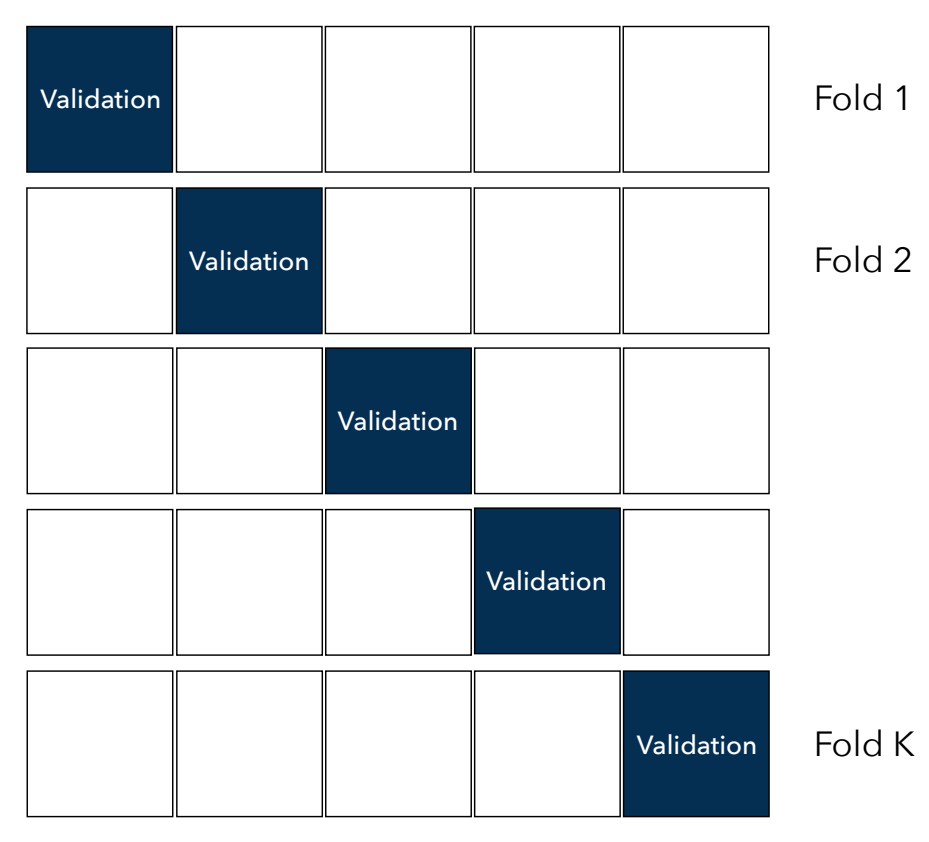
Training, Validation and Test Sets

- The *training set* is used to fit the SVM parameters.
- The *validation set* is used for model selection, e.g., choose C and the kernel function parameters.
- The *test set* is used for assessment of the final model (SVM with chosen C and kernel function).

Dilemma: A big validation set or small one?

Cross-Validation

The gold standard for estimating the prediction error of a model.



Cross-Validation (CV) for Model Selection

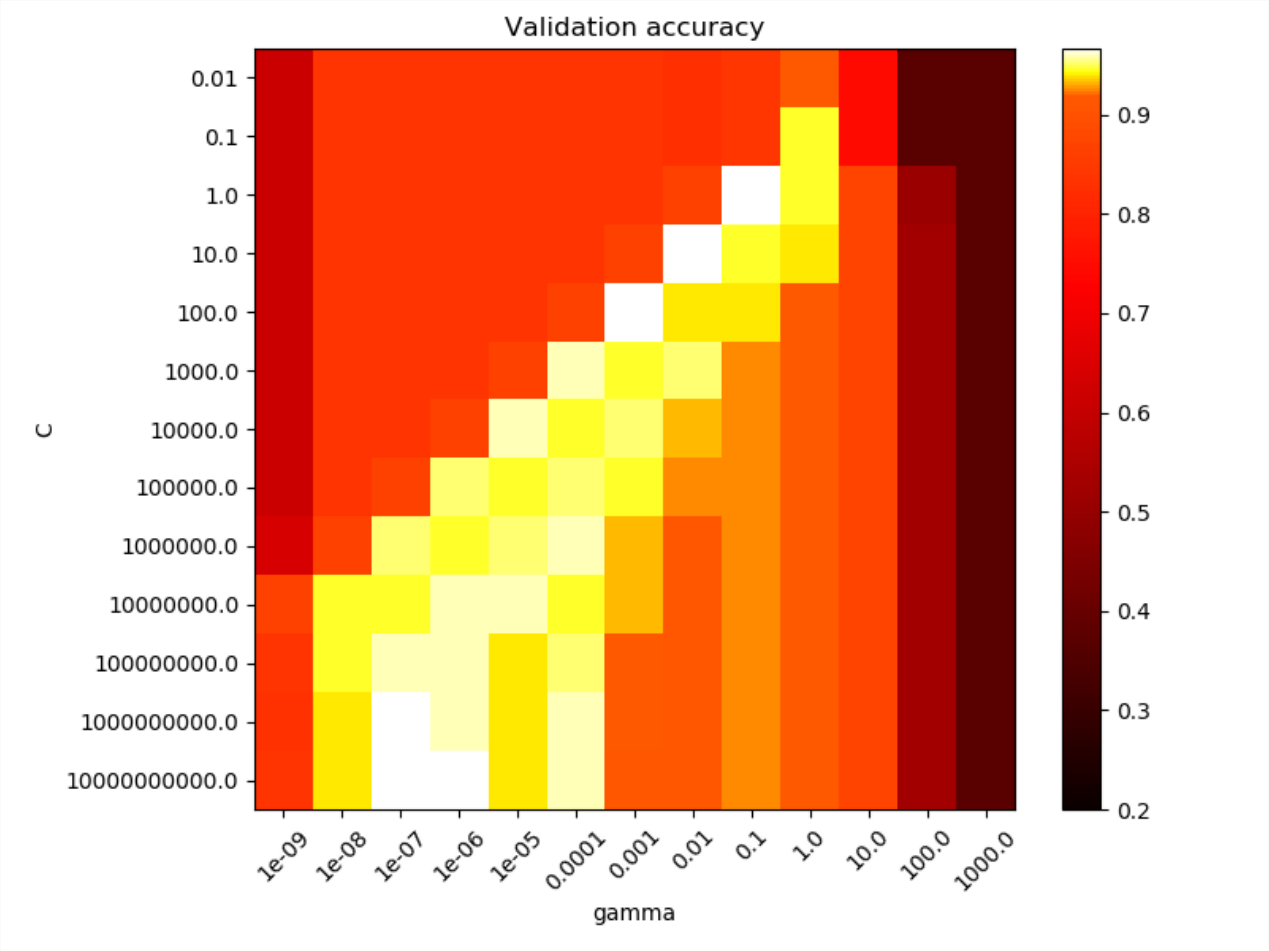
- At each round of CV, one block is used as a validation set to estimate the error of the model and the other blocks are used for training the model.
- At the end, the average of the errors observed among the different folds is used to compare among models.
- In general, the CV error is not a good estimate of the test error of the model you have chosen. Your selected model may "overfit" the cross-validation procedure.
- The most simple approach is to reserve a test set for model assessment and use cross-validation for model selection.

Grid Search

- To choose among different values for a hyper-parameters you need a list of candidate values.
- For the value of C and γ (in the RBF kernel) a logarithmic grid of the form $\{2^{-2}, 2^{-1}, 2^0, 2^1, 2^2\}$ usually works well.
- If you have more than one parameter, the typical approach is to explore, exhaustively, all the possible combinations of candidate values for each parameter.
- The latter method is called grid search and is used within many machine learning models.

Grid Search

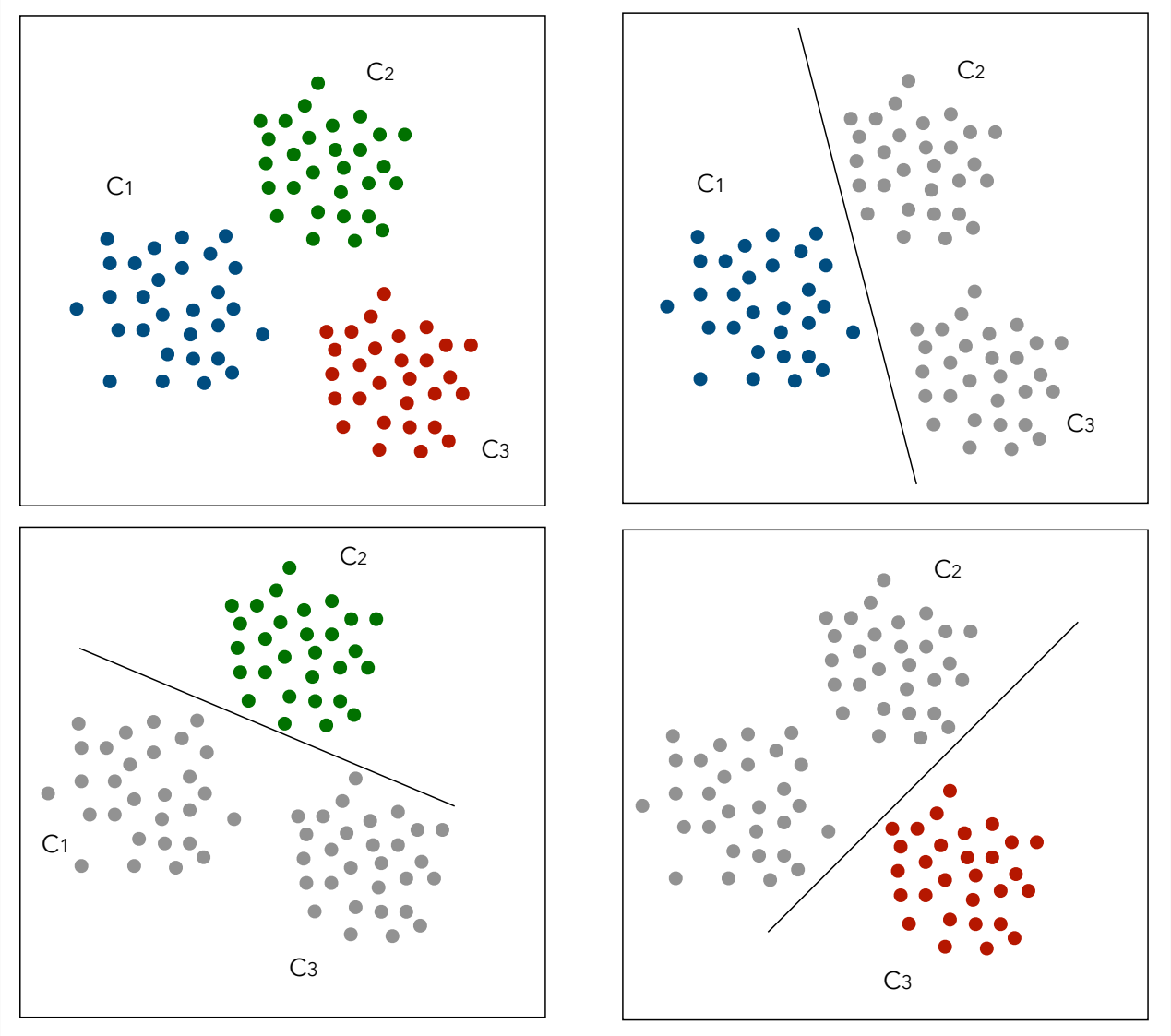
A heat-map depicting the results of a grid search for determining the best value of C and γ in an SVM with RBF kernel.



Multiple Classes

- SVMs are (by default) binary classifiers. They can be extended to multi-category problems in two different ways:
  - *Direct Methods:* Reformulating the optimization problem to consider all the classes simultaneously.
  - *Ensemble Methods:* Train several binary classifiers and then combine their decisions.
- Ensemble methods remain the most widely used method in practice. In particular, there two popular schemes.
  - One versus The Rest (OVR)
  - One versus One (OVO)

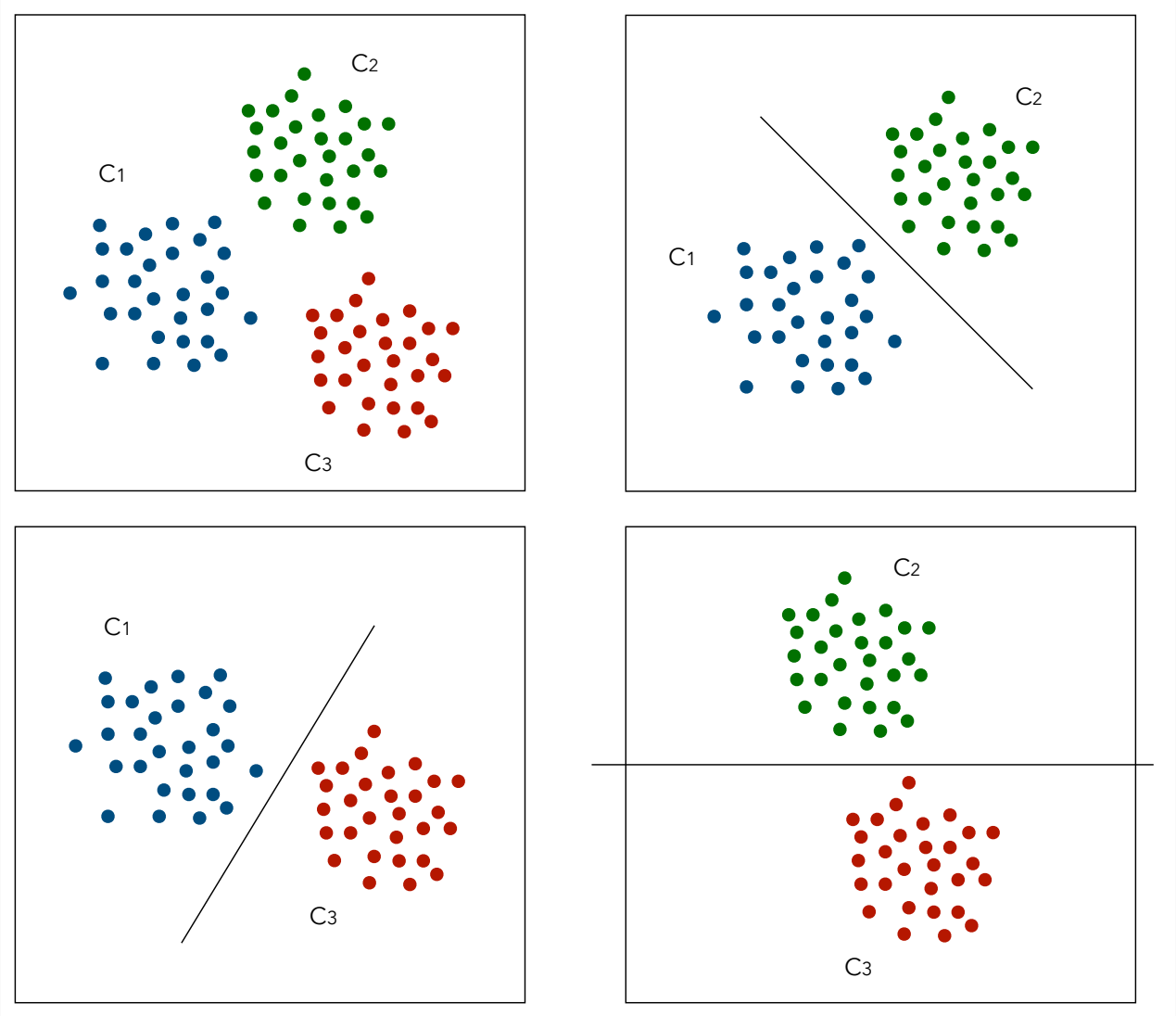
One versus the Rest (OVR) Scheme



One versus the Rest (OVR) Scheme

- If you have K classes, you train K binary classifiers
  - Each classifier is trained to determine if x is more likely to belong to class i $(+1)$ or any other class.
- The decision rule is built as follows:
  - The input pattern is first classified by all the binary classifiers.
  - Each classifier returns the confidence with which the pattern is identified as an example of class i (negative or small value if the pattern is rejected).
  - For SVMs, the "confidence" is proportional to the (signed) distance of the point to the hyperplane $f(x) = (w^T x + b)$.
  - The class with the largest confidence i is finally predicted.

One versus One (OVO) Scheme



One versus One (OVO) Scheme

- If you have K classes, you train $K(K-1)/2$ binary classifiers
  - Each classifier receives examples corresponding to a different pair of classes (j, k), $j < k$ and is trained to determine if an input pattern is more likely to belong to class j $(+1)$ or class k (-1).
- The decision rule is often built using majority voting:
  - The input pattern is first classified by all the binary classifiers.
  - One then counts the "votes" for each class, i.e., number of times a class i has been preferred by a binary classifier.
  - The class with more "votes" is finally predicted.

Referencias

- B. Scholkopf and A. Smola. *Learning with Kernels*, 2001.
- J.Friedman et al. *The Elements of Statistical Learning*, 2009.
- S. Raschka. *Python Machine Learning*, 2015.