UNIVERSITÄT BONN

Support Vector Machines

Elena Trunz

February 17, 2025

Visual Computing Group, University of Bonn

Overview

Introduction

Classification

Given:

- Training data: example-label pairs: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
- Inputs $\mathbf{x}_i \in \mathbb{R}^m$
- Target labels $y_j \in Y = \{t_1, \ldots, t_K\}$, $K =$ number of classes

Find: Classifier (*hypothesis*) $f : \mathbb{R}^m \to Y$, with small generalization error.

Classification algorithm

Idea:

1. Represent input data in $\mathbb{R}^m$
2. Partition the space, such that only the examples with the same label are in the same partition

Binary classification

Only two classes: $Y = \{-1, +1\}$

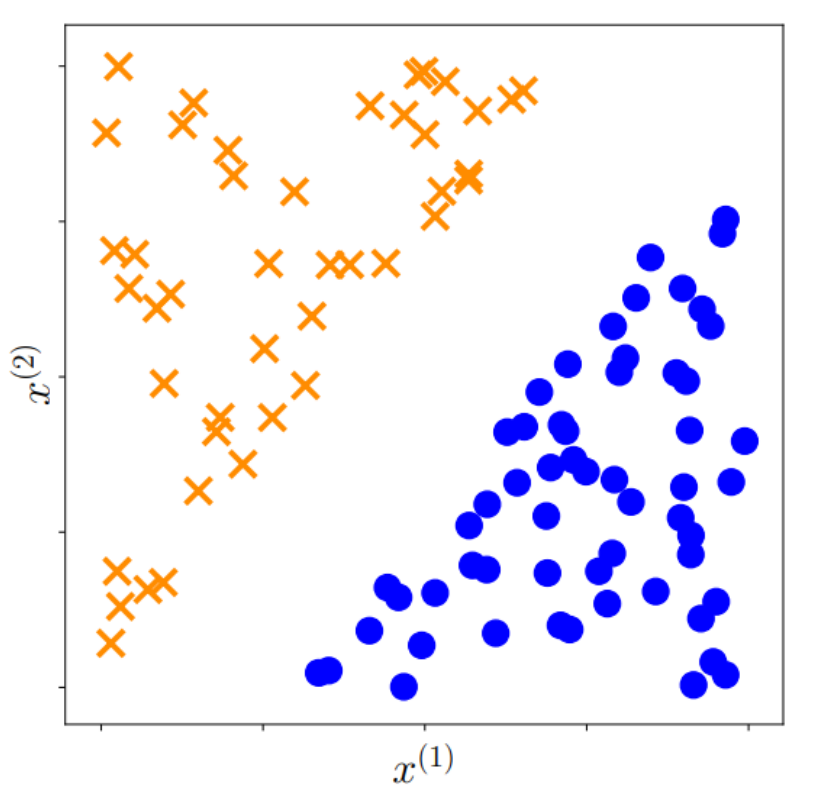
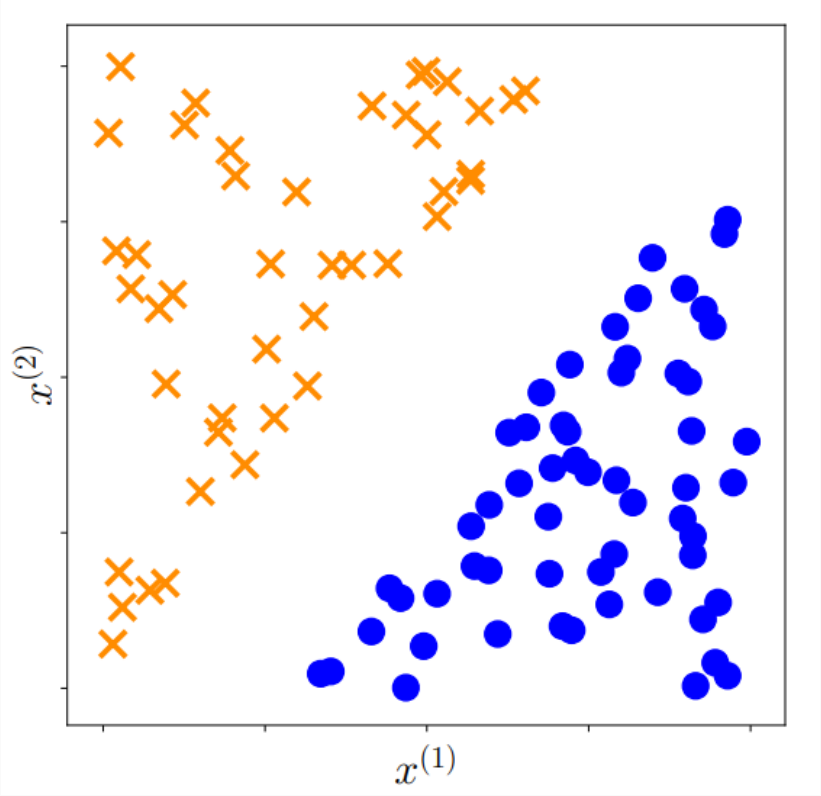


Image was taken from [DFO20]

What is the best classifier?



What about this data?

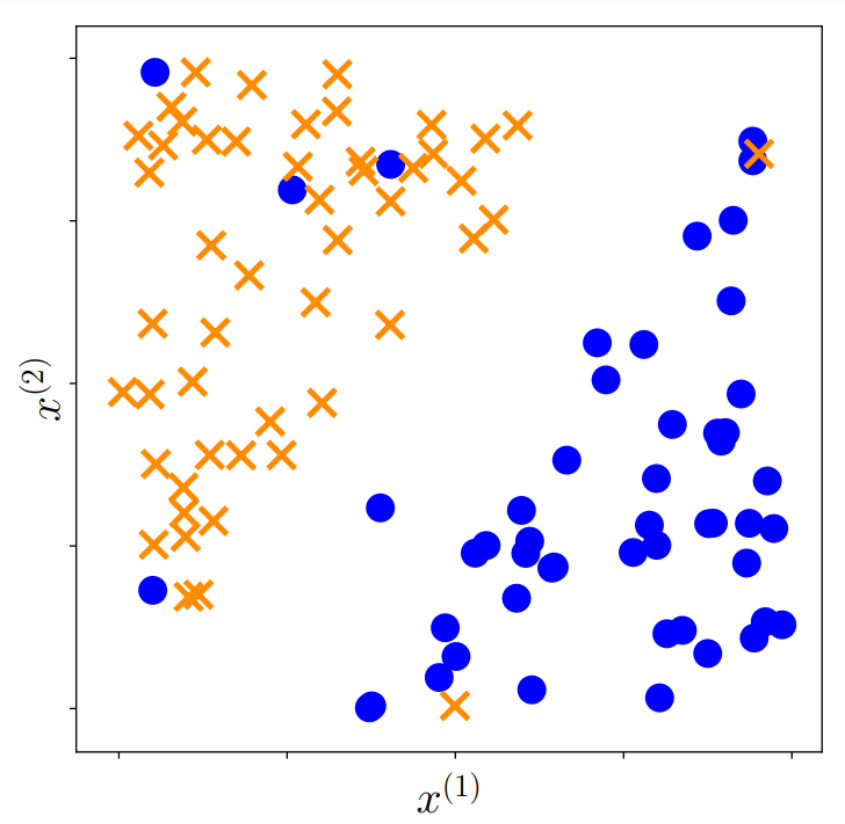
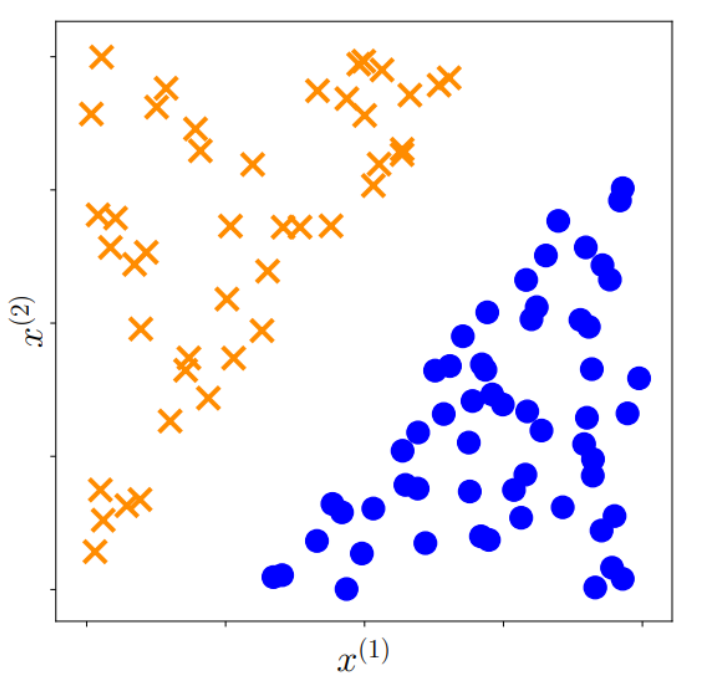


Image was taken from [DFO20]

Linearly Separable Data

Narrow down the search

What is the best classifier here?



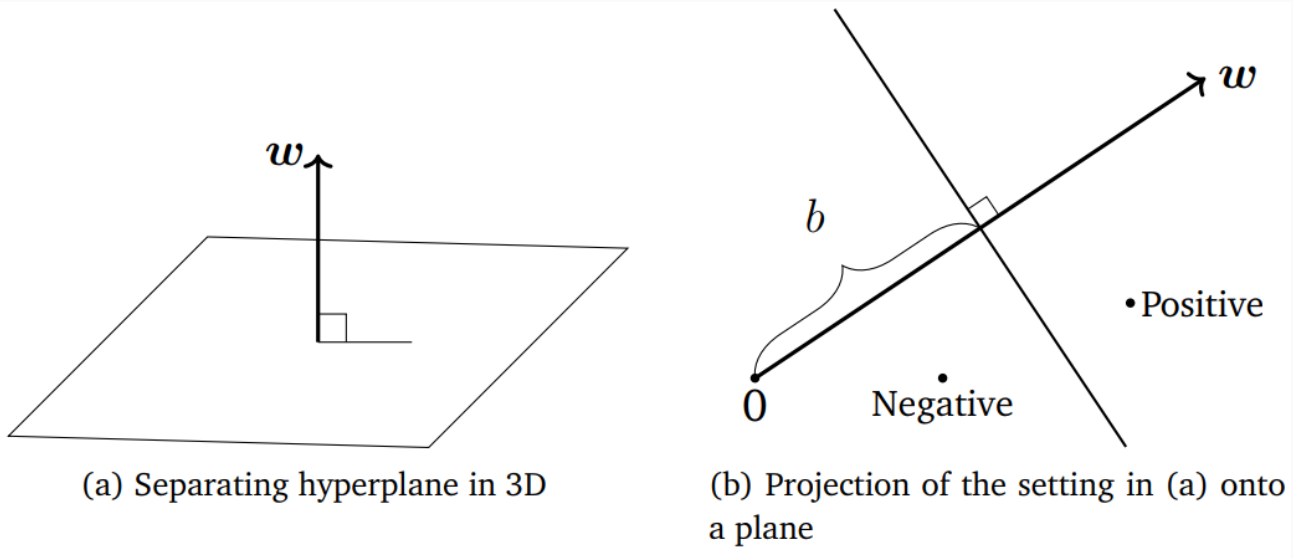
- Occam's razor principle: When presented with competing hypotheses about the same prediction, use the less complex one (fewer parameters, fewer assumptions etc.)
  $\rightarrow$ Use *linear classifiers* (*hyperplanes*) as a hypothesis set

Hyperplane partition

Hyperplane: affine subspace of dimension $m - 1$: Set of points, such that

$$\mathcal{H} = \{\mathbf{x} | \mathbf{w} \cdot \mathbf{x} + b = 0\},$$

where $\mathbf{w} \in \mathbb{R}^m$ is a non-zero vector normal to the hyperplane and $b \in \mathbb{R}$ is a scalar (b is the intercept):



(a) Separating hyperplane in 3D

(b) Projection of the setting in (a) onto a plane

Image was taken from [DFO20]

Hyperplane partition: classification

Classifying a test example $\mathbf{x}_t$:

- Calculate $f(\mathbf{x}_t) := \mathbf{w} \cdot \mathbf{x}_t + b$
- Classify $\mathbf{x}_t$ as $+1$ if $f(\mathbf{x}_t) \geq 0$
- Classify $\mathbf{x}_t$ as -1 if $f(\mathbf{x}_t) < 0$

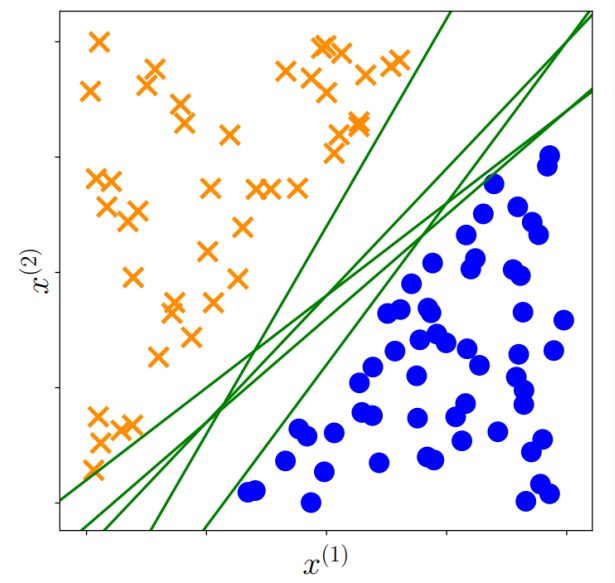
$\rightarrow$ We must find $\mathbf{w}$ and b, such that for all training points holds:

- $f(\mathbf{x}_n) \geq 0$ for points having $y_n = +1$
- $f(\mathbf{x}_n) < 0$ for points having $y_n = -1$

$\rightarrow$ $y_n f(\mathbf{x}_n) \geq 0$

Many candidate hyperplanes

Which hypothesis is the best one?



- We should try to find the classifier that will give the smallest generalization error!
- Idea: Our classifier should not "favor" any class

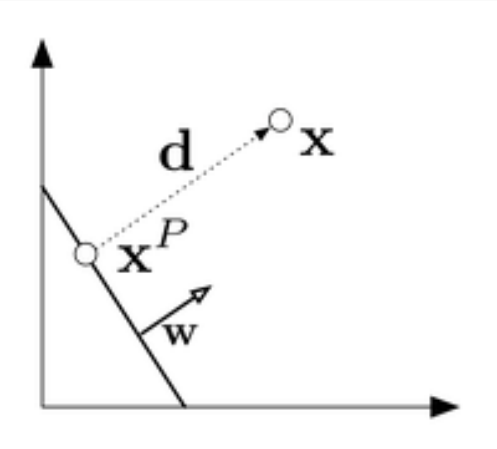
$\rightarrow$ Choose the hyperplane with equally large (maximum) distance to each class

Image was taken from [DFO20]

The margin

What is the distance between a hyperplane and a class?

It is called the *margin* and is defined as the perpendicular distance between the hyperplane and the closest of the data points



If $\mathbf{x}$ is the point closest to the hyperplane, then $\|\mathbf{d}\|_2$ is the margin.

It holds:

$$\|\mathbf{d}\|_2 = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|}$$

Image was taken from [DFO20]

Calculating the margin

The margin of $\mathcal{H}$ with respect to the dataset D can be calculated as follows:

$$\gamma(\mathbf{w}, b) = \min_{\mathbf{x} \in D} \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|}$$

Maximizing the margin

We are interested in hyperplanes for which all training points are correctly classified

$\rightarrow$ $y_n f(\mathbf{x}_n) \geq 0$ must hold for our margin!

Thus, the maximum margin solution is found by solving

$\underset{\mathbf{w},b}{\text{argmax}}\, \gamma(\mathbf{w}, b)$ or

$$\underset{\mathbf{w},b}{\text{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_{\mathbf{x} \in D} |\mathbf{w} \cdot \mathbf{x} + b| \right\}$$

subject to $y_n(\mathbf{w} \cdot \mathbf{x} + b) \geq 0$, for all $n = 1, \ldots, N$.

Maximizing the margin (2)

- Observation 1: Above optimizaiton problem is too complex to solve
- Observation 2: Rescaling $\mathbf{w} \rightarrow c\mathbf{w}$ and $b \rightarrow cb$ would not change the distance from the hyperplane to any point

$\rightarrow$ We can restrict ourselves to pairs $(\mathbf{w}, b)$ scaled such that $\min_{\mathbf{x} \in D} |\mathbf{w} \cdot \mathbf{x} + b| = 1$

We can combine this constraint with the previous ones.

In this case all training points will have to satisfy the constraints: $y_n(\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1$:

Maximizing the margin (3)

Combining the margin maximization with the fact that the examples need to be on the correct side of the hyperplane gives us

$$\underset{\mathbf{w},b}{\text{argmax}} \frac{1}{\|\mathbf{w}\|},$$

subject to $y_n(\mathbf{w} \cdot \mathbf{x} + b) \geq 1$, for all $n = 1, \ldots, N$.

Instead of maximizing the reciprocal of the norm, we often minimize the squared norm:

$$\underset{\mathbf{w},b}{\text{argmin}} \|\mathbf{w}\|^2,$$

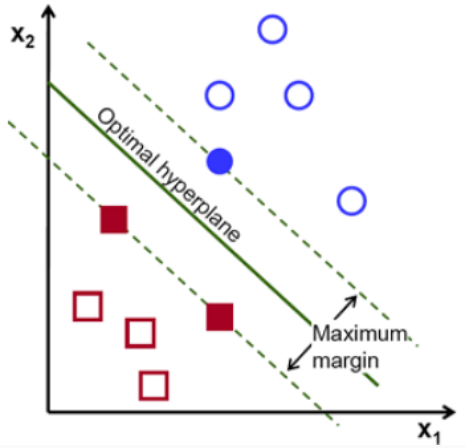
subject to $y_n(\mathbf{w} \cdot \mathbf{x} + b) \geq 1$, for all $n = 1, \ldots, N$.

$\rightarrow$ This optimization problem is known as the *hard margin SVM*.

Solving hard margin SVM

Hard margin SVM is a convex quadratic programming (QP) problem.

A variety of commercial and open-source solvers are available for solving convex QP problems.



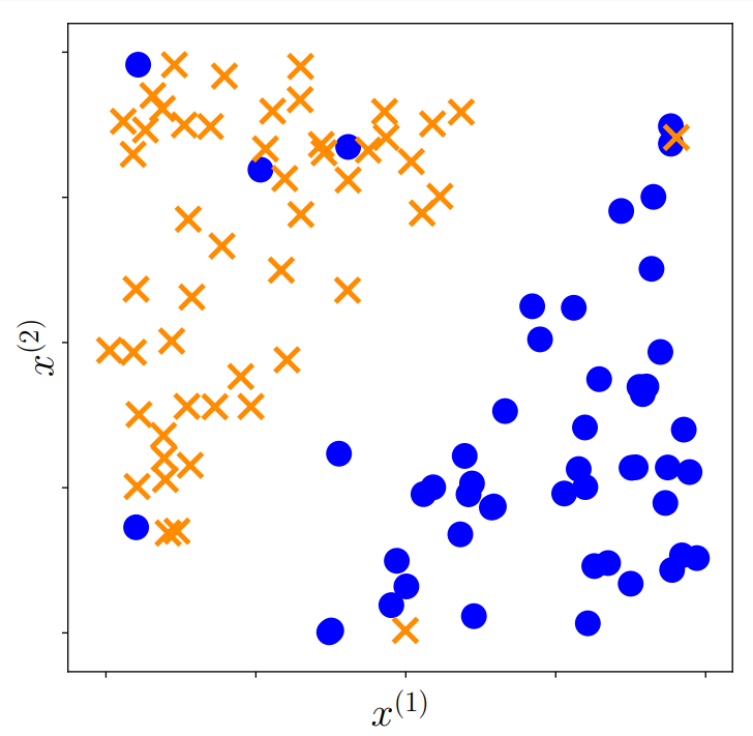
Maximum-margin hyperplane solution. Dashed lines are the marginal hyperplanes. The points on them are support vectors.

Support vectors satisfy $y_n(\mathbf{w} \cdot \mathbf{x}_n + b) = 1$.

Image was taken from [LDP19]

Soft-margin SVM

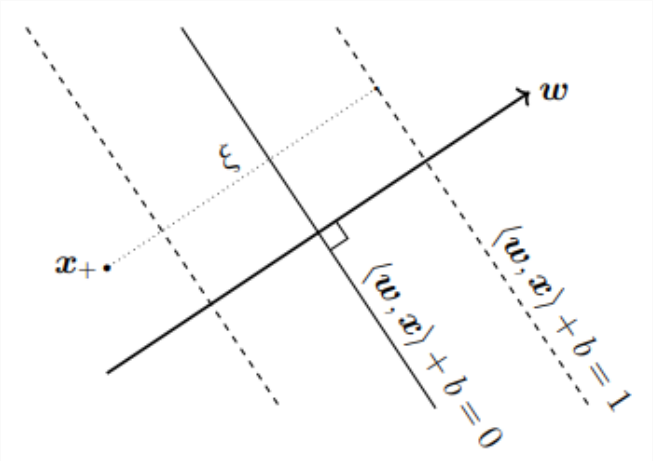
What hyperplane best separates this data?



Linear separation is possible, but with some errors.

Slack variable

To be able to deal with noise, we can soften the hard constraints of our optimization and allow for some slack



Slack variable ξ measures the distance of an example $\mathbf{x}_+$ when it is on the wrong side.

Image was taken from [DFO20]

Soft-margin SVM problem

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \xi_n,$$

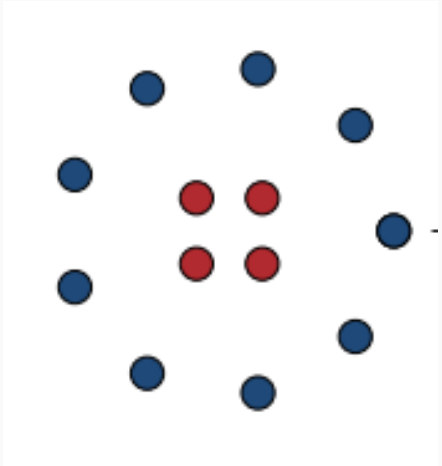
subject to $y_n(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_n$, for all $n = 1, \ldots, N$.

and $\xi_n \geq 0$, for all $n = 1, \ldots, N$.

The parameter C is a hyperparameter that controls the regularization.

Non-Linearly Separable Data

What hyperplane best separates this data?

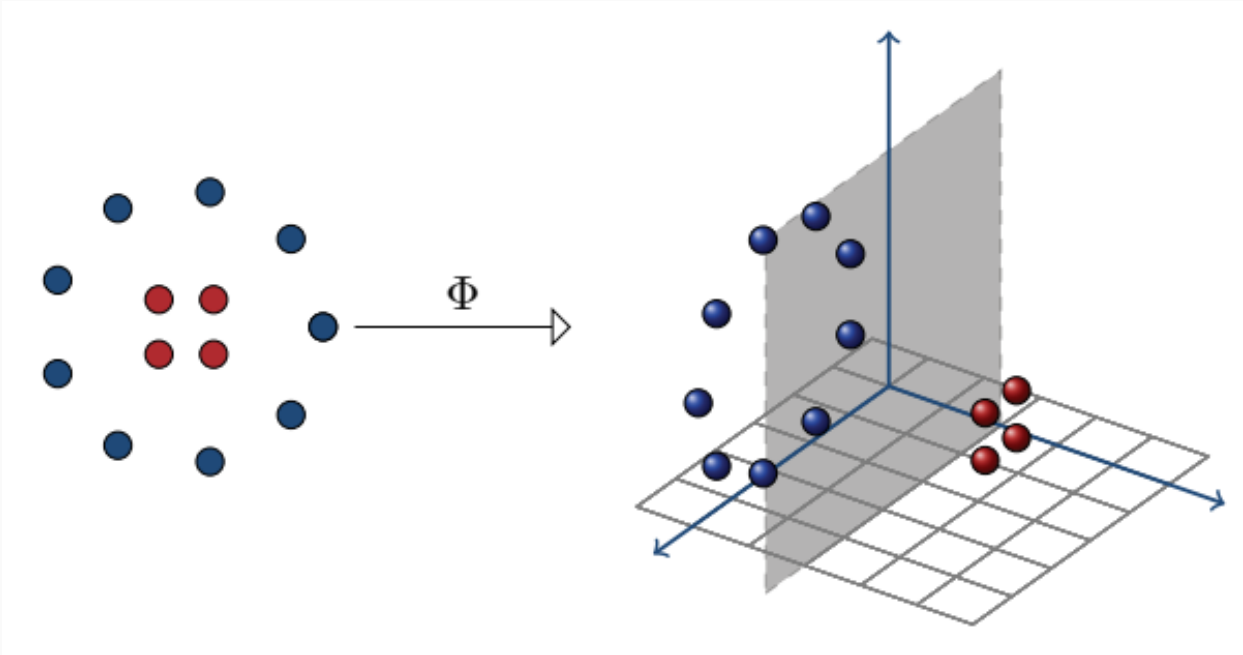


Linear separation is not possible!

Image was taken from [MT18]

Transformation for separation

While the data may not be linear separable in the input space, it may be in a feature space, after a fancy transormation ϕ:



Left: the input space. Right: the feature space.

Image was taken from [MT18]

Non-linear SVM

Idea: instead of tweaking the definition of SVM to accomodate non-linear decision boundaries, gain linearly separation by mapping the input data to a higher dimensional feature space, where the classes are linearly separable:

- Apply transform $\phi : \mathbb{R}^m \to \mathbb{R}^D$ on training data

$$\mathbf{x} = (x_1, \ldots, x_m)^\top \mapsto \phi(\mathbf{x}) = (\phi_1, \ldots, \phi_D)$$

  where $D > m$.

- Train an SVM on the transformed data:

$$\{(\phi(\mathbf{x})_1, y_1), \ldots, (\phi(\mathbf{x})_N, y_N)\}$$

Optimization problem

Using the transformations we now need to maximize the following dual Lagrangian:

$$\widetilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

Problem:

- Dimension D of the feature space can be very large in practice
- Determining the hyperplane solution requires multiple inner product computations in this high dimensional feature space, which can be very costly

$\rightarrow$ Use the kernel trick!

The Kernel Trick

Inner products

Recall: Training an SVM involves maximizing the following function:

$$\widetilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

We need to compute the inner products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ in the feature space.

We are not really interested in the quantities $\phi(\mathbf{x}_i)$ as such!

$\rightarrow$ Instead of explicitly defining a non-linear feature map ϕ and computing the resulting inner product, we define a similarity function $K(\mathbf{x}_i, \mathbf{x}_j)$ between $\mathbf{x}_i$ and $\mathbf{x}_j$, which implicitly defines ϕ.

Kernel function

Given a transformation $\phi : \mathbb{R}^m \to \mathbb{R}^D$ from input space $\mathbb{R}^m$ to feature space $\mathbb{R}^D$, the function $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ defined by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j), \quad \mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^m$$

is called the *kernel function* of ϕ.

Generally: kernel function may refer to any function $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ that measures the similarity of vectors in $\mathbb{R}^m$ without explicitly defining a transform ϕ.

Kernel function for optimization

For a choice of kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ we train an SVM by maximizing

$$\widetilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Computing $K(\mathbf{x}_i, \mathbf{x}_j)$ can be done without computing the mappings $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$.

This way of training a SVM in feature space without explicitly computing the mappings ϕ is called the *kernel trick*.

Example

Define $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ by

$$\phi((x_1, x_2)^\top) = (x_1^2, x_2^2, \sqrt{2}x_1^2x_2^2, \sqrt{2}x_1^2, \sqrt{2}x_2^2, 1)^\top$$

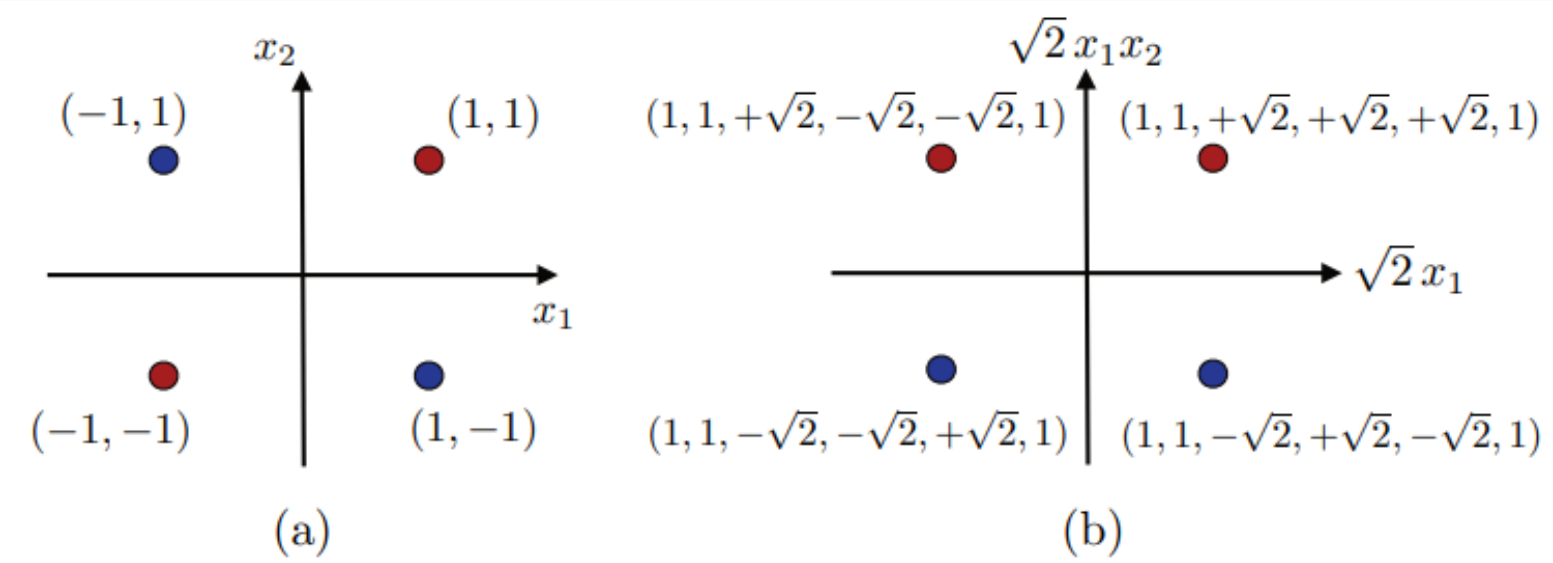
The inner product in the feature space is

$$\phi((x_1, x_2)^\top) \cdot \phi((x_1^{'}, x_2^{'})^\top) = (x_1x_1^{'} + x_2x_2^{'} + 1)^2$$

Thus, we can directly define a kernel function $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ by

$$K(\mathbf{x}_1, \mathbf{x}_2) = (x_1x_1^{'} + x_2x_2^{'} + 1)^2$$

Example: XOR function



(a) XOR Problem linearly non-separable in the input space. (b) Points are mapped to the six-dimensional space defined by second degree polynomial (here projection of these points on the two-dimensional space defined by their third and fourth coordinates). The problem becomes separable by the hyperplane $x_1x_2 = 0$.

Image was taken from [MT18]

Useful kernel functions

- Polynomial kernel:

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^d$$

  where d denotes the degree of the polynomial and is a hyperparameter.

- Gaussian kernel:

$$K(\mathbf{x}_1, \mathbf{x}_2) = exp\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}\right)$$

  where σ is a hyperparameter.

- Sigmoid kernels:

$$K(\mathbf{x}_1, \mathbf{x}_2) = tanh(a(\mathbf{x}_1 \cdot \mathbf{x}_2) + b)$$

  where a and b are hyperparameters.

Example: SVM with Gaussian kernel

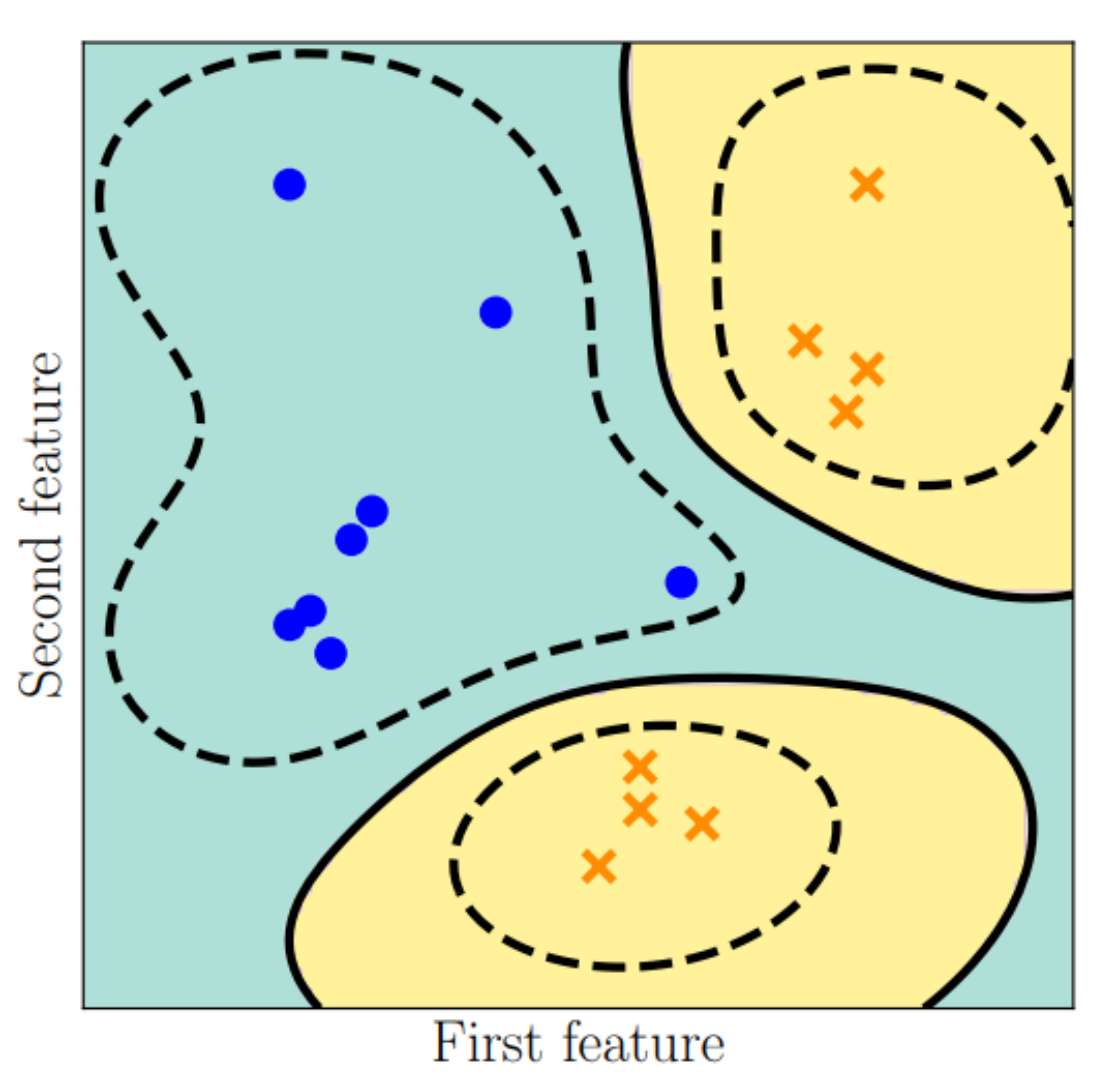


Image was taken from [DFO20]

Example: SVM with polynomial (degree 2) kernel

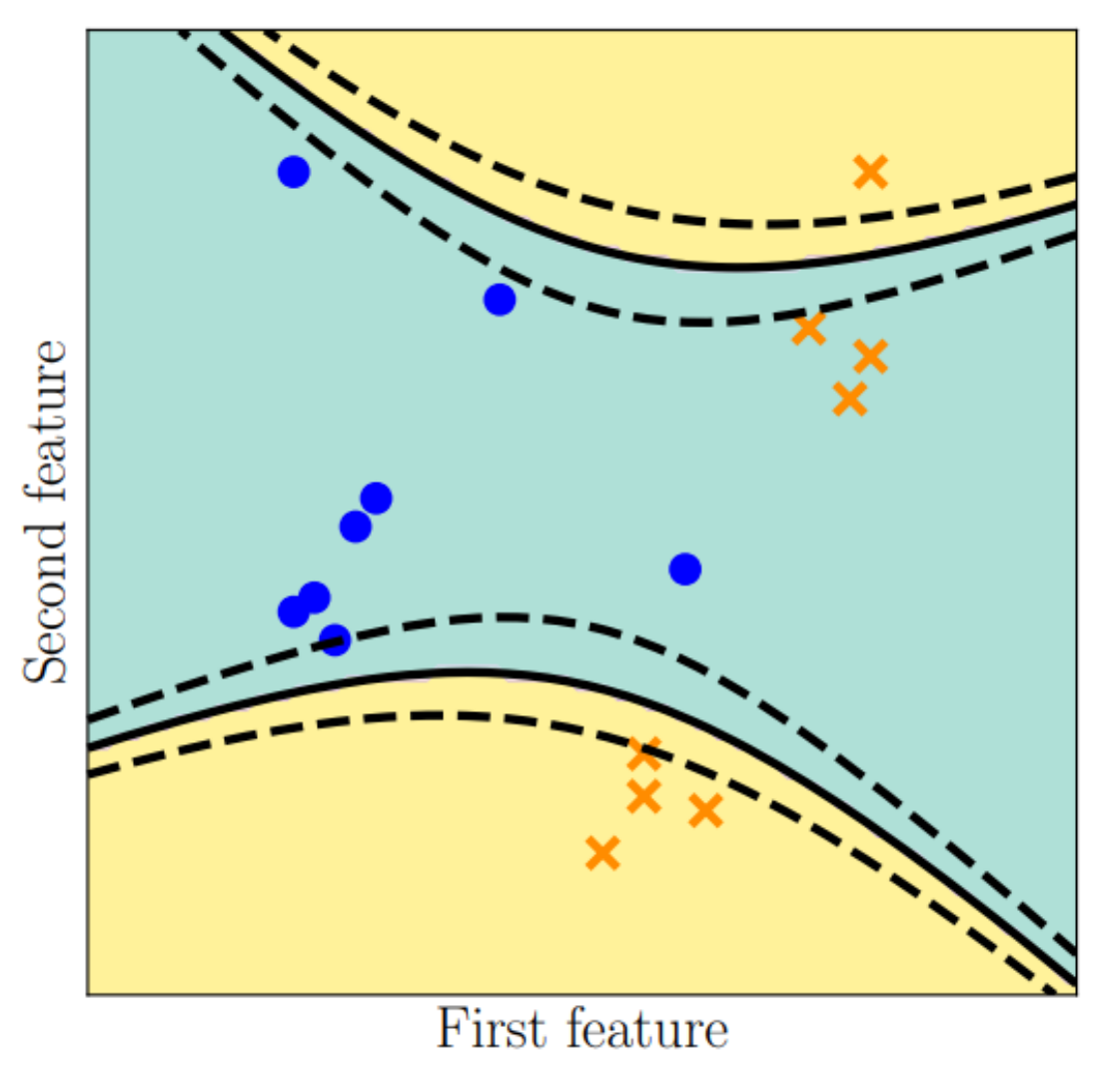


Image was taken from [DFO20]

Example: SVM with polynomial (degree 3) kernel

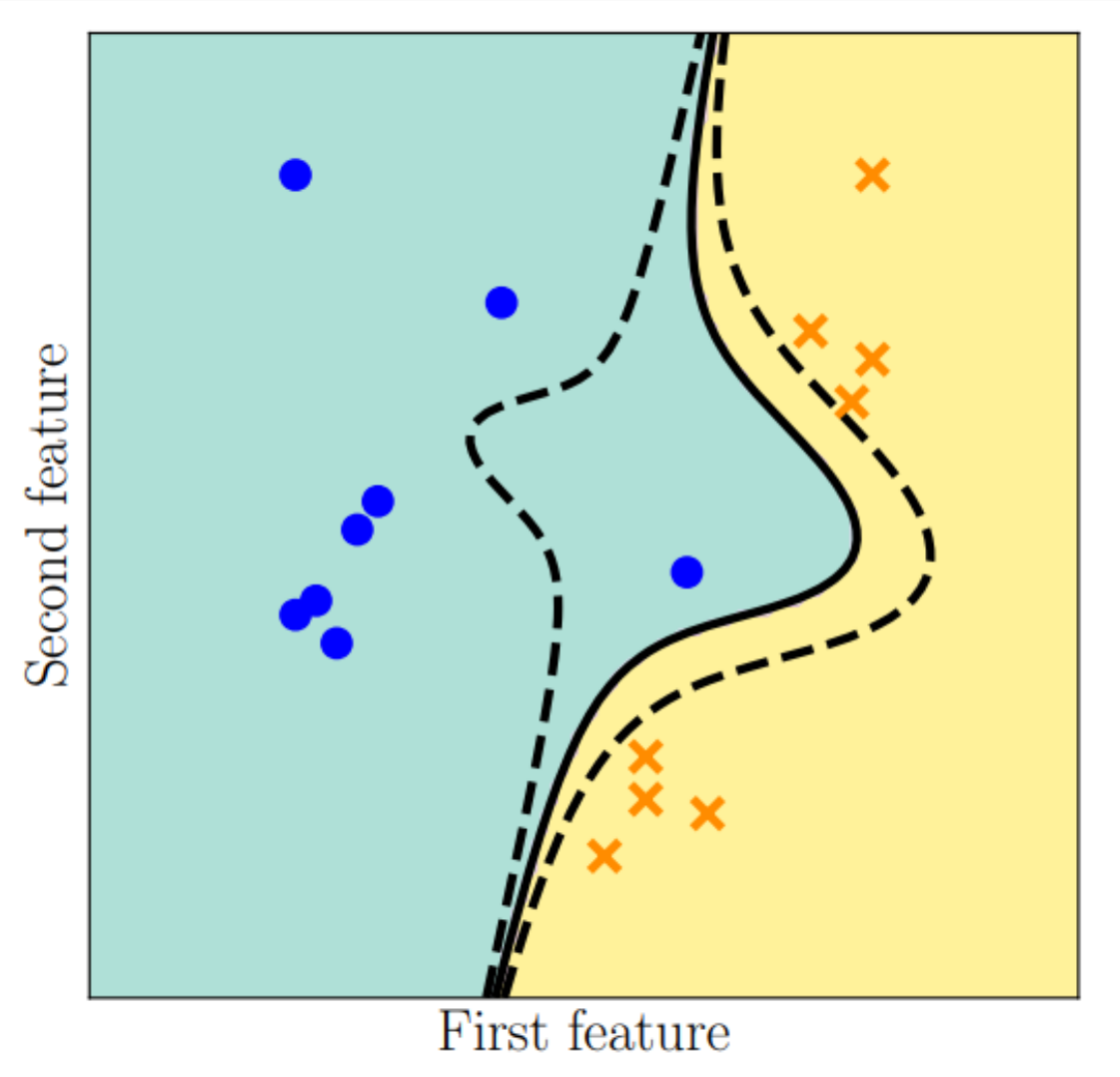


Image was taken from [DFO20]

Choosing suitable kernel functions

Kernel function is a hyperparameter.

The choice depends on a problem at hand.

Try several common ones and choose the one with the best results on the validation set.

If in doubt - use a Gaussian!

References

[DFO20] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[LDP19] R LakshmanNaika, R. Dinesh, and S Prabhanjan. "Handwritten Electric Circuit Diagram Recognition: An Approach Based on Finite State Machine." In: *International Journal of Machine Learning and Computing* (2019).

[MT18] Afshin Rostamizadeh Mehryar Mohri and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2018.