Kernel Methods

Machine Learning

Michael Wand

TA: Eric Alcaide

{michael.wand, eric.alcaide}@idsia.ch

Dalle Molle Institute for Artificial Intelligence Studies (IDSIA) USI - SUPSI

Fall Semester 2024

Introduction

- In the last part, we have talked about linear methods for classification and regression.
- The methods we have covered so far are all *parametric*: a model is computed from the training data, then the training data is discarded, and only the model is used for further calculations.
- There are, however, methods where at least a part of the training data is kept.
- Some of them are based on simple comparison of test samples and training samples (e.g. the kNN classifier).
- Kernel methods are more sophisticated examples of this category.
- Kernel methods are linear methods in a feature space. Yet, they offer a different view of both linear models, and the concept of features.
- We will cover in detail the best-known kernel algorithm, the Support Vector Machine.

Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
USI/SUPSI
IDSIA

Kernel Methods

Dual Representations

- Consider Linear Regression with training data $(\phi_n, t_n)_{n=1,\dots,N}$ with L_2 regularization, where $\phi_n = \phi(\mathbf{x}_n)$:

$$E(\mathbf{w}) = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{w}^T\phi_n - t_n)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

  (note that for easier calculation, this is the regularized squared error, not the *mean* squared error).
- The gradient of the right-hand side w.r.t. $\mathbf{w}$ is

$$\sum_{n=1}^{N}(\mathbf{w}^T\phi_n - t_n)\phi_n + \lambda\mathbf{w}$$

  and setting it to zero yields

$$\mathbf{w} = \sum_{n=1}^{N} \underbrace{-\frac{1}{\lambda}(\mathbf{w}^T\phi_n - t_n)}_{a_n}\phi_n$$

  with *scalars* a_n.

Dual Representations

- Using the design matrix Φ and writing the coefficients a_n as a vector $\mathbf{a}$, we get
$$\mathbf{w} = \Phi^T \mathbf{a}.$$
- Thus we see that not only the model is linear, but also that $\mathbf{w}$ is a linear combination of the (features of the) training data.
- Q: Do you think this is surprising?

Dual Representations

- Using the design matrix Φ and writing the coefficients a_n as a vector $\mathbf{a}$, we get
$$\mathbf{w} = \Phi^T \mathbf{a}.$$
- Thus we see that not only the model is linear, but also that $\mathbf{w}$ is a linear combination of the (features of the) training data.
- Q: Do you think this is surprising?
- The observation that $\mathbf{w}$ and the features Φ have such a simple relationship leads us to the idea of expressing the whole regression problem in terms of the training data points!
- Such a reformulation, called the dual representation, is possible for a variety of linear models.

Dual Representations

- Substituting $\mathbf{w} = \Phi^T \mathbf{a}$ into $E(\mathbf{w})$ yields the expression

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T \Phi\Phi^T \Phi\Phi^T \mathbf{a} - \mathbf{a}^T \Phi\Phi^T \mathcal{T} + \frac{1}{2}\mathcal{T}^T\mathcal{T} + \frac{\lambda}{2}\mathbf{a}^T\Phi\Phi^T\mathbf{a}$$

with the target vector $\mathcal{T} = (t_1, \ldots, t_N)^T$.

- We define the Kernel function

$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = \langle \phi(\mathbf{x}_n), \phi(\mathbf{x}_m) \rangle$$

and the Gram matrix

$$\mathbf{K} = \Phi\Phi^T = (k(\mathbf{x}_n, \mathbf{x}_m))_{n,m=1,\ldots,N},$$

which allows us to express the error in the dual representation

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T \mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T \mathbf{K}\mathcal{T} + \frac{1}{2}\mathcal{T}^T\mathcal{T} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}.$$

Solving in Dual Representation

- Setting the gradient of

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathcal{T} + \frac{1}{2}\mathcal{T}^T\mathcal{T} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}.$$

  w.r.t. **a** to zero, one obtains the following minimum of the error function:

$$\mathbf{a} = (\mathbf{K} + \lambda I_N)^{-1}\mathcal{T}.$$

  (I_N is the $N \times N$ unity matrix.)

- Plugging this solution into the original model, the prediction function is given by

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda I_N)^{-1}\mathcal{T}$$

  with $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}_n, \mathbf{x}))_{n=1,\dots,N}$.

The Role of the Kernel

- The solution is found by inverting an $N \times N$ matrix, whereas in the original formulation, we have inverted an $M \times M$ matrix.
- (Here M is the number of features, and N is the number of training data points.)
- So what is the advantage of the dual formulation?

The Role of the Kernel

- The solution is found by inverting an $N \times N$ matrix, whereas in the original formulation, we have inverted an $M \times M$ matrix.
- (Here M is the number of features, and N is the number of training data points.)
- So what is the advantage of the dual formulation?
- We never use the feature vector ϕ directly!
- The input data appears only inside the kernel function $k(x, y)$.
- This allows us to **implicitly** use very high-dimensional (possibly infinite-dimensional) feature spaces!

Features and Kernels

- Let us consider a well-known feature, namely *quadratic features*: For $\mathbf{x} = (x^{(1)}, \ldots, x^{(K)})$, let

$$\phi(\mathbf{x}) = (1, \sqrt{2}x^{(1)}, \ldots, \sqrt{2}x^{(K)},$$
$$(x^{(1)})^2, \ldots, (x^{(K)})^2, \sqrt{2}x^{(1)}x^{(2)}, \ldots, \sqrt{2}x^{(K-1)}x^{(K)}).$$

- For larger dimensionalities K, it could take a lot of computing power to compute a) the feature itself, b) the scalar product $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$ between two features?

Features and Kernels

- Let us consider a well-known feature, namely *quadratic features*: For $\mathbf{x} = (x^{(1)}, \ldots, x^{(K)})$, let

$$\phi(\mathbf{x}) = (1, \sqrt{2}x^{(1)}, \ldots, \sqrt{2}x^{(K)}, \\ (x^{(1)})^2, \ldots, (x^{(K)})^2, \sqrt{2}x^{(1)}x^{(2)}, \ldots, \sqrt{2}x^{(K-1)}x^{(K)}).$$

- For larger dimensionalities K, it could take a lot of computing power to compute a) the feature itself, b) the scalar product $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$ between two features?
- **No**, since the kernel function is given by

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = (\mathbf{x}^T \mathbf{y} + 1)^2 \qquad \text{(proof left as exercise)}$$

- The complex-looking kernel is thus actually very simple to compute! This is called the Kernel Trick.

Features and Kernels

- The feature space can even have an infinite number of dimensions!
- As an example, take the RBF (radial basis function) kernel

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

(with fixed σ).

- The exponential can be expanded as a power series, yielding a feature space of the form

$$\phi(\mathbf{x}) = \left[\exp\left(-\frac{1}{2}||\mathbf{x}||^2\right)\frac{x_1^{n_1}\cdot x_k^{n_k}}{\sqrt{(n_1!\cdot n_k!)}}\right]_{(n_1,\dots,n_k)\in\mathbb{N}^k}$$

which has an infinite number of dimensions.

- Still, we can use the kernel function to reason about linear regression (and other linear methods) in this infinite-dimensional space!

Features and Kernels

- The RBF kernel measures the closeness between two data points.
- This view can be extended to kernel functions in general:
  - As known from linear algebra, the scalar product measures the "alignment", i.e. the similarity, between two vectors.
  - Thus the kernel function likewise measures the alignment between data points!
- The feature which underlies a given kernel function determines what we consider "similar".
- Choosing a kernel is thus equivalent to choosing features, just that instead of searching for a good representation of the data, we search for a good representation of the *similarity* between data points!

Theory of Kernel Functions

- We aim to understand the properties of Kernel functions, and how to create new kernel functions.
- We have seen that the kernel function is defined as a scalar product in feature space:
$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$$
- Clearly, the kernel function is symmetric, thus the Gram matrix is also symmetric. One can also show that the Gram matrix $\mathbf{K} = (k(\mathbf{x}_n, \mathbf{x}_m))_{n,m=1,\dots,N}$ is positive-semidefinite, i.e. for all vectors $\mathbf{c} \neq 0$, $\mathbf{c}^T \mathbf{K} \mathbf{c} \geq 0$:
$$\begin{aligned} \mathbf{c}^T \mathbf{K} \mathbf{c} &= \sum_i \sum_j c_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle c_j \\ &= \langle \sum_i c_i \phi(\mathbf{x}_i), \sum_j c_j \phi(\mathbf{x}_j) \rangle = \langle y, y \rangle \geq 0 \end{aligned}$$
for $y = \sum_i c_i \phi(\mathbf{x}_i)$.

Kernel Functions

- It is also possible to show the opposite: If for a function $k(\mathbf{x}_n, \mathbf{x}_m)$, the Gram matrix is positive-semidefinite for any finite data set $(\mathbf{x}_n)_{n=1,\dots,N}$, one can write k as a scalar product in some (possibly infinite-dimensional) feature space:

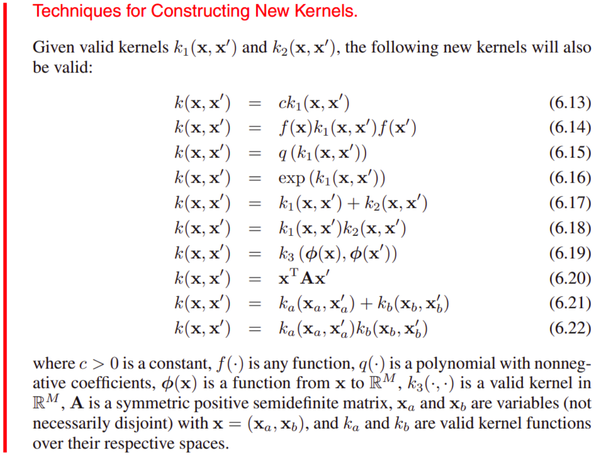
$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$$

  for some mapping ϕ.
- In this case, the function k is called a valid kernel.
- Note that this does *not* mean that $k(\mathbf{x}_n, \mathbf{x}_m)$ must always be positive!
- We do not cover the proof in this lecture, as a reference, consider Taylor & Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press 2004.

Constructing Kernels

- In practice, kernels can be constructed using a set of transformations, starting from the linear kernel $k_{\text{lin}}(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^T \mathbf{x}_m$.
- Thus, one avoids to describe the feature space explicitly.
- A number of standard kernels has been described in literature, with the polynomial kernel and the RBF kernel among the most common choices.
- It is also possible to devise specific, application-dependent kernels. Note that these kernels may map *arbitrary objects* into the implicit feature space!

Constructing Kernels



Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

From Bishop, p.296

Kernel Methods: Summary and Outlook

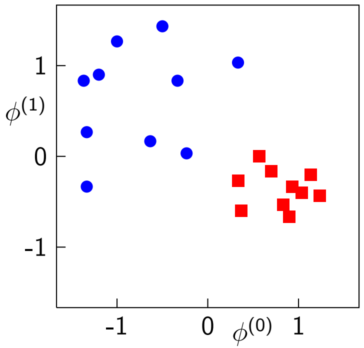
- We have seen that linear regression can be rewritten in *dual representation*, based only on the kernel function of the training data points.
- This reformulation allows to easily use high-dimensional or even infinite-dimensional feature spaces.
- Many linear methods can be *kernelized*, i.e. rewritten as kernel methods.
- However, this comes at the price of having to keep all training data points in memory. In particular, this can be a problem in the prediction phase.
- In the following section, we will get to know a specific kernel method for classification which avoids this problem by requiring only a small subset of the training data for classification: The Support Vector Machine (SVM).

Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
USI/SUPSI
IDSIA

Support Vector Machines

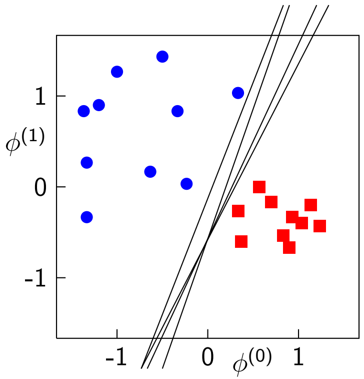
Introduction

- Remember the two-class problem from the last lecture?
  - ➔ I again removed the outlier
  - ➔ Perfect solution(s) exist, the classes are *linearly separable*.



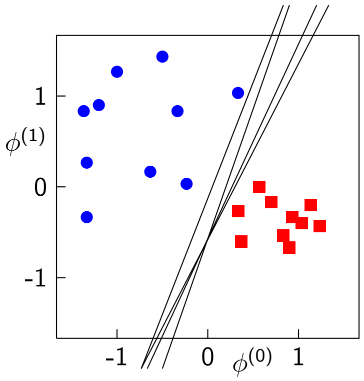
Introduction

- Remember the two-class problem from the last lecture?
  - I again removed the outlier
  - Perfect solution(s) exist, the classes are *linearly separable*.



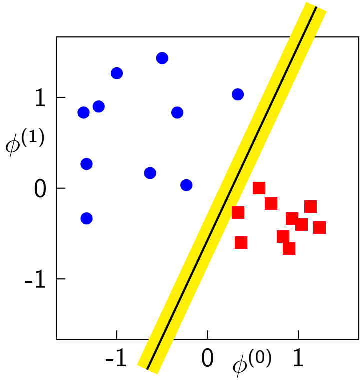
Introduction

- Remember the two-class problem from the last lecture?
  - → I again removed the outlier
  - → Perfect solution(s) exist, the classes are *linearly separable*.
- We do linear classification in feature space
  - → the classifier depends linearly on the features and the parameters
  - → the features themselves may be nonlinear.
- How can we choose an optimal hyperplane (here: a straight line) as decision boundary?



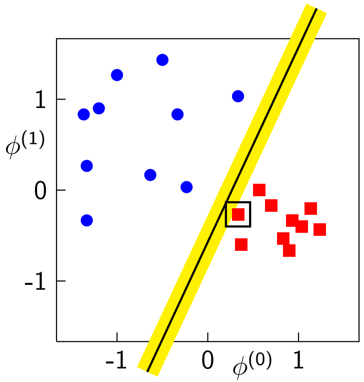
Definition of the Margin

- Assume that the classes are linearly separable.
- Define the *margin* of the linear classifier as twice the distance from the decision boundary to the nearest point (of either class).
- In the figure, the margin is indicated in yellow.



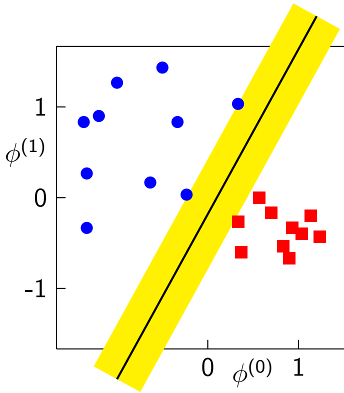
Definition of the Margin

- Assume that the classes are linearly separable.
- Define the *margin* of the linear classifier as twice the distance from the decision boundary to the nearest point (of either class).
- In the figure, the margin is indicated in yellow.



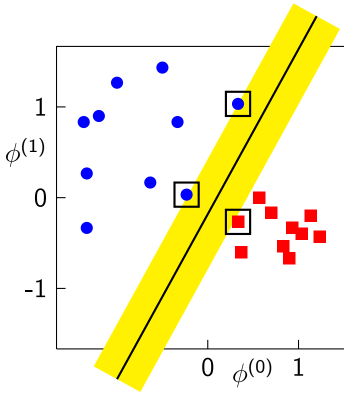
Maximum Margin

- Idea: We choose the decision boundary which *maximizes* the margin.



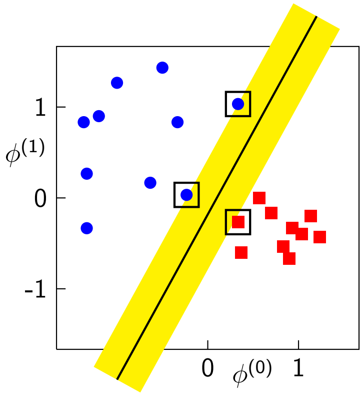
Maximum Margin

- Idea: We choose the decision boundary which *maximizes* the margin.
- Obviously, this means that we have at least one datapoint of each class which push against the margin.
- These data points are called the *support vectors*.



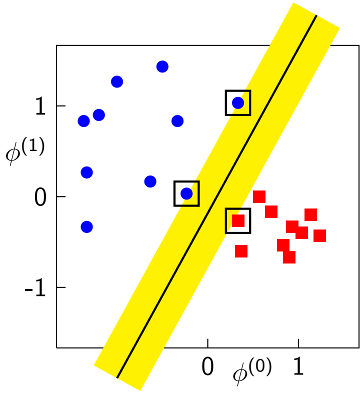
Maximum Margin

- Idea: We choose the decision boundary which *maximizes* the margin.
- Obviously, this means that we have at least one datapoint of each class which push against the margin.
- These data points are called the *support vectors*.
- This classifier is called the *Support Vector Machine*.



Maximum Margin

- Idea: We choose the decision boundary which *maximizes* the margin.
- Obviously, this means that we have at least one datapoint of each class which push against the margin.
- These data points are called the *support vectors*.
- This classifier is called the *Support Vector Machine*.
- There is always a unique solution.



Maximum Margin

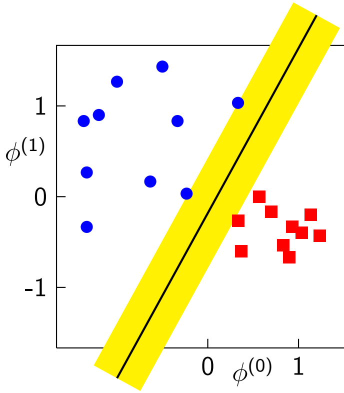
Why do we define our classifier this way?

- Intuitively this feels safest.
- If we have made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
- The model is easy since it is depends only on the support vector datapoints (this also means that we do not need to save the training data points after fitting).
- There is some theory (using VC dimension) which indicates that maximizing the margin is a good thing.
- Empirically it works very very well (compared to other classifiers which are linear in parameter space).

Based on Andrew Moore's SVM tutorial slides.

Mathematical Formulation

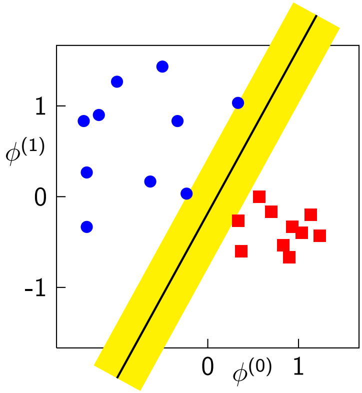
- Training samples: $\mathbf{x}_1, \ldots, \mathbf{x}_N$, from which we get features $\phi_1, \ldots, \phi_N$.
- $y(\phi) = \mathbf{w}^T\phi + w_0$ is the discriminant function; determine $\mathbf{w}$ and w_0.
- ● if $y(\phi) > 0$, 🟥 if $y(\phi) < 0$ (equality does not occur since the classes are linearly separable).



Mathematical Formulation

- Training samples: $\mathbf{x}_1, \ldots, \mathbf{x}_N$, from which we get features $\phi_1, \ldots, \phi_N$.
- $y(\phi) = \mathbf{w}^T\phi + w_0$ is the discriminant function; determine $\mathbf{w}$ and w_0.
- ● if $y(\phi) > 0$, 🟥 if $y(\phi) < 0$ (equality does not occur since the classes are linearly separable).

$\Rightarrow$ Since only the sign matters, the discriminant function can be scaled by an arbitrary constant.



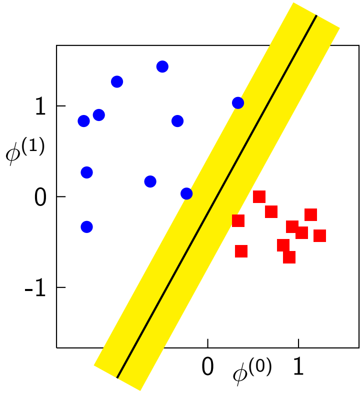
Mathematical Formulation

- Training samples: $\mathbf{x}_1, \ldots, \mathbf{x}_N$, from which we get features $\phi_1, \ldots, \phi_N$.
- $y(\phi) = \mathbf{w}^T \phi + w_0$ is the discriminant function; determine $\mathbf{w}$ and w_0.
- ● if $y(\phi) > 0$, 🟥 if $y(\phi) < 0$ (equality does not occur since the classes are linearly separable).

$\Rightarrow$ Since only the sign matters, the discriminant function can be scaled by an arbitrary constant.

$\Rightarrow$ We look for a *normalized* $y(\phi)$ such that for all data points, $|y(\phi)| = |\mathbf{w}^T \phi + w_0| \geq 1$, i.e.

- ● if $\mathbf{w}^T \phi + w_0 \geq 1$
- 🟥 if $\mathbf{w}^T \phi + w_0 \leq -1$.



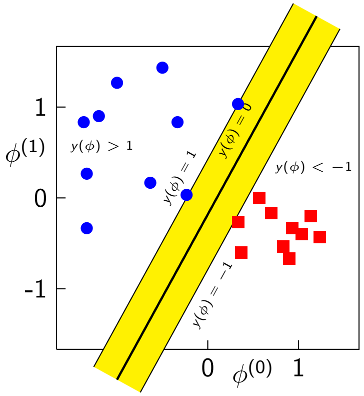
Mathematical Formulation

- Training samples: $\mathbf{x}_1, \ldots, \mathbf{x}_N$, from which we get features $\phi_1, \ldots, \phi_N$.
- $y(\phi) = \mathbf{w}^T\phi + w_0$ is the discriminant function; determine $\mathbf{w}$ and w_0.
- ● if $y(\phi) > 0$, 🟥 if $y(\phi) < 0$ (equality does not occur since the classes are linearly separable).

$\Rightarrow$ Since only the sign matters, the discriminant function can be scaled by an arbitrary constant.

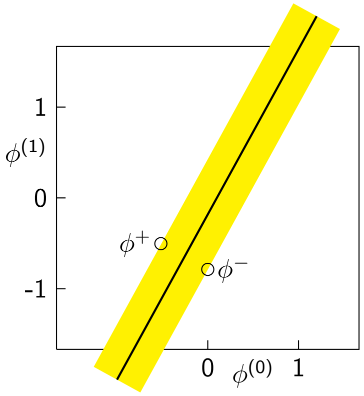
$\Rightarrow$ We look for a *normalized* $y(\phi)$ such that for all data points, $|y(\phi)| = |\mathbf{w}^T\phi + w_0| \geq 1$, i.e.

- ➔ ● if $\mathbf{w}^T\phi + w_0 \geq 1$
- ➔ 🟥 if $\mathbf{w}^T\phi + w_0 \leq -1$.



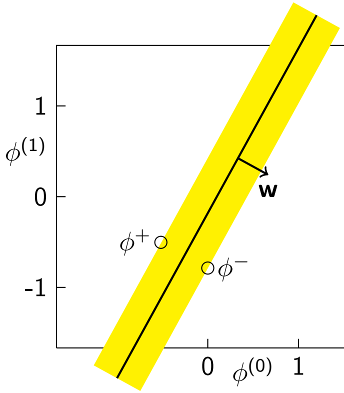
Mathematical Formulation

- We need an expression for the margin.
- Let ϕ^+ be a point (not necessarily a training data sample) on the "plus" margin, and ϕ^- the *closest* point to ϕ^+ on the "minus" margin.
- Then $\phi^+ = \phi^- + \lambda\mathbf{w}$ for a scalar λ. (Why?)



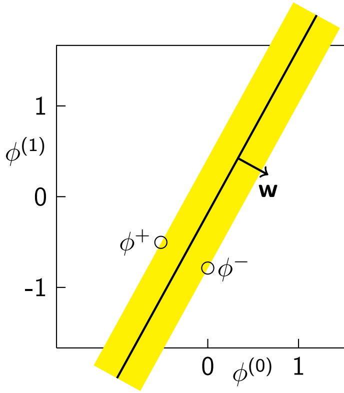
Mathematical Formulation

- We need an expression for the margin.
- Let ϕ^+ be a point (not necessarily a training data sample) on the "plus" margin, and ϕ^- the *closest* point to ϕ^+ on the "minus" margin.
- Then $\phi^+ = \phi^- + \lambda \mathbf{w}$ for a scalar λ, since one finds the closest point on a plane by orthogonal projection, and $\mathbf{w}$ is orthogonal to the decision boundary and thus also to the margin planes.



Mathematical Formulation

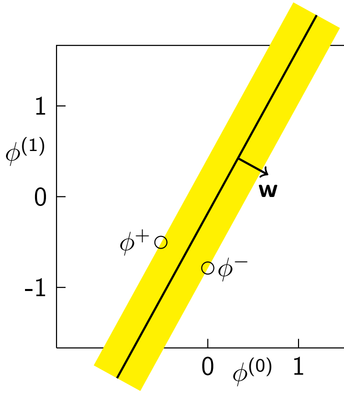
- We need an expression for the margin.
- Let ϕ^+ be a point (not necessarily a training data sample) on the "plus" margin, and ϕ^- the *closest* point to ϕ^+ on the "minus" margin.
- Then $\phi^+ = \phi^- + \lambda \mathbf{w}$ for a scalar λ, since one finds the closest point on a plane by orthogonal projection, and $\mathbf{w}$ is orthogonal to the decision boundary and thus also to the margin planes.
- The margin width is thus $||\lambda \mathbf{w}||$.



Mathematical Formulation

- From
  - $\mathbf{w}^T\phi^+ + w_0 = 1$
  - $\mathbf{w}^T\phi^- + w_0 = -1$
  - $\phi^+ = \phi^- + \lambda\mathbf{w}$

  easily follows $\lambda = \frac{2}{\mathbf{w}^T\mathbf{w}}$.

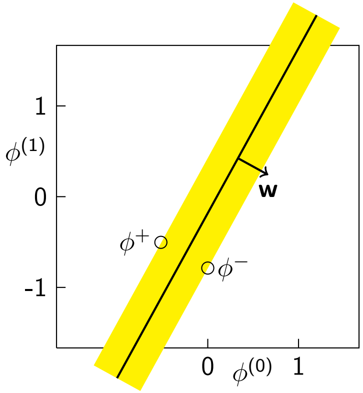


Mathematical Formulation

- From
  - $\mathbf{w}^T\phi^+ + w_0 = 1$
  - $\mathbf{w}^T\phi^- + w_0 = -1$
  - $\phi^+ = \phi^- + \lambda\mathbf{w}$

  easily follows $\lambda = \frac{2}{\mathbf{w}^T\mathbf{w}}$.
- Since the margin width is given by $M = ||\lambda\mathbf{w}||$, we finally obtain

$$M = ||\lambda\mathbf{w}|| = \lambda||\mathbf{w}|| = \lambda\sqrt{\mathbf{w}^T\mathbf{w}} = \frac{2\sqrt{\mathbf{w}^T\mathbf{w}}}{\mathbf{w}^T\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}.$$



Mathematical Formulation

Given a guess of $\mathbf{w}$ and w_0, we can

- check whether the resulting hyperplane separates the classes
- compute the margin width.

But how do we search through all possible solutions to find the best one (i.e. the one with maximal margin width)?

Mathematical Formulation

Given a guess of $\mathbf{w}$ and w_0, we can

- check whether the resulting hyperplane separates the classes
- compute the margin width.

But how do we search through all possible solutions to find the best one (i.e. the one with maximal margin width)?

The problem can be formulated as a **Quadratic programming** problem.

Quadratic Programming

A general quadratic programming problem is as follows:

- Minimize[1] (for fixed Q and $\mathbf{c}$)

$$\arg\min_{x} \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$$

- subject to the constraints

$$\begin{aligned} A_{\text{eq}}\mathbf{x} &= b_{\text{eq}} \\ A_{\text{ineq}}\mathbf{x} &\leq b_{\text{ineq}}, \end{aligned}$$

  where the two sets of linear equality and inequality constraints are written in matrix form.

- Such problems can be solved very efficiently and reliably with specialized algorithms (which we do not cover here).

[1] alternatively maximize

Margin Maximization by Quadratic Programming

- Remember the discriminant function with normalized $\mathbf{w}$:
  - ➔ ● if $\mathbf{w}^T\phi + w_0 \geq 1$; 🟥 if $\mathbf{w}^T\phi + w_0 \leq -1$.

  and the margin width
  - ➔ $M = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$.

Margin Maximization by Quadratic Programming

- Remember the discriminant function with normalized $\mathbf{w}$:
  - ● if $\mathbf{w}^T\phi + w_0 \geq 1$; ■ if $\mathbf{w}^T\phi + w_0 \leq -1$.

  and the margin width
  - $M = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$.
- From this, we get the setup for the Quadratic Programming problem:
  - Compute $\arg\min_{\mathbf{w}} \mathbf{w}^T\mathbf{w}$ (this amounts to maximizing M)

  under the set of constraints
  - $\mathbf{w}^T\phi + w_0 \leq -1$ for *all* samples ϕ of class ■
  - $\mathbf{w}^T\phi + w_0 \geq +1$ for *all* samples ϕ of class ● .

Margin Maximization by Quadratic Programming

- Remember the discriminant function with normalized $\mathbf{w}$:
  - ➔ ● if $\mathbf{w}^T\phi + w_0 \geq 1$; 🟥 if $\mathbf{w}^T\phi + w_0 \leq -1$.

  and the margin width
  - ➔ $M = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$.
- From this, we get the setup for the Quadratic Programming problem:
  - ➔ Compute $\arg\min_{\mathbf{w}} \mathbf{w}^T\mathbf{w}$ (this amounts to maximizing M)

  under the set of constraints
  - ➔ $\mathbf{w}^T\phi + w_0 \leq -1$ for *all* samples ϕ of class 🟥
  - ➔ $\mathbf{w}^T\phi + w_0 \geq +1$ for *all* samples ϕ of class ● .
- Define numerical targets $t_1, \ldots, t_N \in \{-1, +1\}$ for each sample $\phi_1, \ldots, \phi_N$ (🟥 $\equiv -1$, ● $\equiv +1$).

Margin Maximization by Quadratic Programming

- Remember the discriminant function with normalized $\mathbf{w}$:
  - ● if $\mathbf{w}^T\phi + w_0 \geq 1$; 🟥 if $\mathbf{w}^T\phi + w_0 \leq -1$.

  and the margin width
  - $M = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$.
- From this, we get the setup for the Quadratic Programming problem:
  - Compute $\arg\min_{\mathbf{w}} \mathbf{w}^T\mathbf{w}$ (this amounts to maximizing M)

  under the set of constraints
  - $\mathbf{w}^T\phi + w_0 \leq -1$ for *all* samples ϕ of class 🟥
  - $\mathbf{w}^T\phi + w_0 \geq +1$ for *all* samples ϕ of class ● .
- Define numerical targets $t_1, \ldots, t_N \in \{-1, +1\}$ for each sample $\phi_1, \ldots, \phi_N$ (🟥 $\equiv -1$, ● $\equiv +1$).
- Now we can write the constraints in a unified way
  - $t_n(\mathbf{w}^T\phi_n + w_0) \geq 1, n = 1, \ldots, N$

  and let a standard QP solver compute the solution.

Excurse: Lagrange Multipliers

- For the next step, we will briefly review a mathematical tool for constrained optimization, the *Lagrange Multipliers* and their generalization by Karush, Kuhn, and Tucker.
- For a simple start, assume we need to optimize a (two-dimensional) function as follows: Find

$$\arg\max_{x,y} f(x,y) \qquad \text{subject to} \qquad g(x,y) = 0$$

Excurse: Lagrange Multipliers

- Clearly, we cannot directly maximize f by setting the derivative to zero, since the resulting maximum (A) does not (generally) fulfill the constraint.
- At the constrained maximum B, the gradient of f is *not* zero.

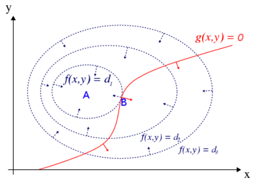


Image source: Wikipedia, Lagrange Multiplier, modified

- Of course, we could try to rewrite the constraint

$$g(x,y) = 0 \Leftrightarrow y = \tilde{g}(x)$$

- Then, eliminate y from f and solve the unconstrained problem

$$\arg\max_x f(x, \tilde{g}(x))$$

- But this is ugly, complicated, and often not even possible!

Excurse: Lagrange Multipliers

A better *intuitive* solution works like this:

- Walk along the constraint line $g(x, y) = 0$, looking for a maximum
- A maximum is reached when we just "touch" a contour line of f (a line where the value of f is constant)
- At a touching point, the tangents of the contour line and the constraint line must be parallel
- Since the gradient is orthogonal to the tangent, this means that the gradients of f and g must be parallel at such touching points

$\Rightarrow$ Necessary condition: $\nabla f + \lambda \nabla g = 0$ for some λ.

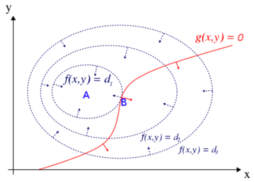


Image source: Wikipedia, Lagrange Multiplier, modified

Skipped some details (e.g. differentiability requirements, what happens when f or g has a plateau, how to check whether we really have a maximum, multiple variables and constraints, etc.). Never mind.

Excurse: Lagrange Multipliers

Task: Maximize[2] $f(\mathbf{x})$ (for a vector $\mathbf{x}$), subject to M constraints $g_m(\mathbf{x}) = 0, m = 1, \dots, M$.
Solution (without proof):

- Introduce the *Lagrangian*

$$L(\mathbf{x}, \lambda_1, \dots, \lambda_M) = f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \dots + \lambda_M g_M(\mathbf{x})$$

- Solve (can also be done numerically)

$$\nabla_{(\mathbf{x}, \lambda_1, \dots, \lambda_M)} L(\mathbf{x}, \lambda_1, \dots, \lambda_M) = 0$$

  (note that the derivatives w.r.t. λ just yield the constraints)

- The resulting points (usually finitely many) are candidates for extrema. Check which ones suit your needs[3].

[2]or minimize

[3]there is a formal criterion, using a variant of the usual Hessian matrix, but even mathematicians say that's "more trouble than it's worth"

Karush-Kuhn-Tucker Conditions

- The idea can also be used when we have *inequality constraints* of the form $g(\mathbf{x}) \geq 0$.
- Consider a single constraint. Two cases are possible:
  - A constrained extremum $\mathbf{x}_E$ lies *within* the region described by $g(\mathbf{x}) \geq 0$, i.e. $g(\mathbf{x}_\mathbf{E}) > 0$. Then the constraint is *inactive*, and since a neighborhood of $\mathbf{x}_E$ is contained in the constraint zone $g(\mathbf{x}) > 0$, $\mathbf{x}_E$ must be a "true" extremum, the constraint is said to be *inactive*, and we have $\lambda = 0$ and $\nabla f(x_E) = 0$.
  - A constrained extremum $\mathbf{x}_E$ lies *on the border* of the region described by $g(\mathbf{x}) \geq 0$, i.e. $g(\mathbf{x}_\mathbf{E}) = 0$. This is the case we had before, the constraint is *active*.
- We again have the condition on the Lagrangian

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) = 0$$

here with a multiplier $\lambda \geq 0$ (the sign accounts for the inequality constraint).

Karush-Kuhn-Tucker Conditions

- In either case, $\lambda \cdot g(\mathbf{x}) = 0$ (for inactive constraints: $\lambda = 0$, for active constraints: $g(\mathbf{x}) = 0$).
- Thus we have to optimize the Lagrangian $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$ under the constraints

$$\begin{aligned} g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned}$$

- You will see that the last constraint ($\lambda g(\mathbf{x}) = 0$) plays a very important role for SVMs!
- Optimization is usually done numerically.

Dual Representation of the SVM

Let's go back to the SVM and use Lagrange multipliers to compute a solution!

- We have to minimize $\mathbf{w}^T\mathbf{w}$ with the constraints $t_n(\mathbf{w}^T\phi_n + w_0) \geq 1, n = 1, \ldots, N$.
- Introduce N Lagrange multipliers $\mathbf{a} = (a_n)_n, n = 1, \ldots, N$ with $a_n \geq 0$, to get the Lagrangian

$$L(\mathbf{w}, w_0, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n(t_n(\mathbf{w}^T\phi_n + w_0) - 1)$$

- Setting the derivatives of L to zero yields the conditions

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi_n \qquad \text{and} \qquad 0 = \sum_{n=1}^{N} a_n t_n.$$

- Use these equations to simplify L by eliminating $\mathbf{w}$ and w_0.

Dual Representation of the SVM

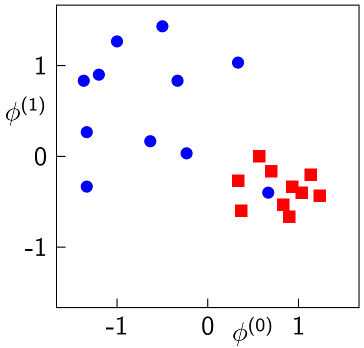
- We obtain the famous *dual representation* of the maximum margin problem

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m),$$

  where $k(\mathbf{x}_n, \mathbf{x}_m) = \phi_n^T \phi_m$ is the kernel function.
- $\tilde{L}$ needs to be optimized over $\mathbf{a}$ with the constraints $a_n \geq 0, n = 1, \ldots, N$, and $0 = \sum_{n=1}^{N} a_n t_n$.
- We need to optimize as many coefficients as there are training data points.
- This can again be solved by quadratic programming.
- Fundamental advantage of this formulation: the data samples only appear in the kernel function!

Overlapping Class Distributions

- What if the classes are *not* linearly separable in feature space?
- Introduce *slack variables* $\xi_n \geq 0$ for each training sample
  - ➔ $\xi_n = 0$ if the sample is correctly classified *and* outside the margin (i.e. $t_n(\mathbf{w}^T\phi_n + w_0) \geq 1$)
  - ➔ $\xi_n = |t_n - y(\phi_n)|$ otherwise
- Note that $\xi_n > 0$ does *not* mean the data point is misclassified (it could be correctly classified, but violate the margin)



Overlapping Class Distributions

- The classification constraints now become $t_n y(\phi_n) = t_n(\mathbf{w}^T \phi_n + w_0) \geq 1 - \xi_n$ with $\xi_n \geq 0$.
- Compute

$$\arg\min_{\mathbf{w}} \left(\frac{1}{2} ||\mathbf{w}||^2 + C \sum_{n=1}^{N} \xi_n \right)$$

where the parameter C is the trade-off between margin width and margin/classification errors.

- The dual representation of the problem is as before

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m),$$

with new constraints $0 \leq a_n \leq C$, $n = 1, \ldots, N$, and $0 = \sum_{n=1}^{N} a_n t_n$.

Prediction

- If we used the original, direct SVM formulation, we would obtain optimized $\mathbf{w}$ and w_0.
- Prediction is as usual: $y(\phi) = \mathbf{w}^T\phi + w_0$, with the optimal values for $\mathbf{w}$ and w_0.
- If we use the dual representation, we have optimal parameters $\mathbf{a}$.
- The prediction can likewise be formulated using the kernel function (without proof):

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x_n}) + w_0$$

with

$$w_0 = \frac{1}{|\mathcal{S}|} \sum_{n\in\mathcal{S}} \left(t_n - \sum_{m\in\mathcal{S}} a_m t_m k(\mathbf{x_m}, \mathbf{x_n}) \right)$$

where $\mathcal{S}$ contains the indices of the support vectors.

Summary so far

We have

- defined a criterion for finding a linear classifier (in feature space): *maximum margin*
- outlined how to formulate this criterion
- derived the *dual* formulation, where the features only appear in the kernel function
- have described how to set up the task to be solved by standard *quadratic programming* algorithms
- seen how to trade-off in the case that a perfect solution is impossible.

Further reading: Bishop, *Pattern Recognition and Machine Learning*, Chapter 7 (contains many more details on mathematical derivation, dealing with multiple classes, obtaining probability estimates, etc.)
Now we consider properties of the resulting classifier, particularly regarding efficient computation!

Role of Sparsity

- Even with the kernel trick, the prediction

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x_n}) + w_0.$$

still requires computing a sum over all training data points?

Role of Sparsity

- Even with the kernel trick, the prediction

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x_n}) + w_0.$$

  still requires computing a sum over all training data points?
- **No**, because all a_n which do *not* belong to support vectors are *zero*!

⇒ Multiplication needs only to be done for support vectors.

Role of Sparsity

- Even with the kernel trick, the prediction

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x_n}) + w_0.$$

  still requires computing a sum over all training data points?
- **No**, because all a_n which do *not* belong to support vectors are *zero*!

$\Rightarrow$ Multiplication needs only to be done for support vectors.

- Why? The a_n are Lagrange multipliers for the inequality constraints $t_n y(\phi_n) \geq 1 - \xi_n$, and they are nonzero only for *active* constraints
- Constraint n is active $\Leftrightarrow$ x_n is a Support Vector.

Role of Sparsity

- Even with the kernel trick, the prediction

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x_n}) + w_0.$$

  still requires computing a sum over all training data points?
- **No**, because all a_n which do *not* belong to support vectors are *zero*!

$\Rightarrow$ Multiplication needs only to be done for support vectors.

- Why? The a_n are Lagrange multipliers for the inequality constraints $t_n y(\phi_n) \geq 1 - \xi_n$, and they are nonzero only for *active* constraints
- Constraint n is active $\Leftrightarrow$ x_n is a Support Vector.

Thus the whole classifier depends only on the support vectors: The sparsity property of SVMs. This can also be considered a form of simplicity regularization.

SVM Example

Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

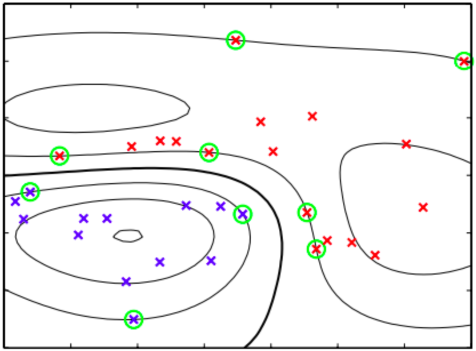


Image: Bishop, figure 7.2

Summary: SVM

- We have seen that the SVM is based on two fundamental concepts: *Kernel trick* (efficient dealing with nonlinearities) and *Sparsity* (data efficiency during testing).
- Thus, while the SVM is a classifier which is linear in feature space, the concept is different from the ordinary linear classification which we covered before:
  - ➔ we retain a (small) subset of the training samples (the support vectors) even during testing
  - ➔ we do not explicitly compute $\mathbf{w}$ (and neither the features).
- The specific properties of the dual formulation of the SVM make this possible.
- A variety of related algorithms exist, allowing to use the Support Vector framework e.g. for regression, or for probabilistic outputs, etc.

Conclusion

- In this lecture, we have taken the idea of linear models and pushed it forward quite a bit.
- We arrived at a reformulation of linear models which is quite different from the standard parametric model: Instead of using a parameter vector, prediction is performed on the basis of the training data points, which enter the calculation only via the kernel function.
- This allows us to easily reason about very high-dimensional, implicitly given feature spaces.
- The key ingredient for all kernel methods is the kernel function, which should reflect similarity between data points.
- We have covered a very important example of a kernel method, namely the SVM classifier.
- SVM keywords: max margin, kernel trick, sparsity