*** Applied Machine Learning Fundamentals *** Support Vector Machines

Daniel Wehner

SAP SE

October 24, 2019

Agenda October 24, 2019

Section:

Introduction

What is a Support Vector Machine (SVM)?

- A support vector machine is a **binary classifier** *(Vapnik and Chervonenkis)*
  - The classes are denoted by $\{-1; +1\}$
  - Techniques for multi-class classification:
    - **OVR (One-vs-Rest)**
    - **OVO (One-vs-One)**
- An SVM finds the **best separating hyperplane**
- Why is it called *'machine'*?
  - It's no physical machine, it's a mathematical construct *(just as a 'Turing machine' is no real machine...)*
  - 'Machine' refers to 'Machine Learning'

What is the best separating hyperplane?

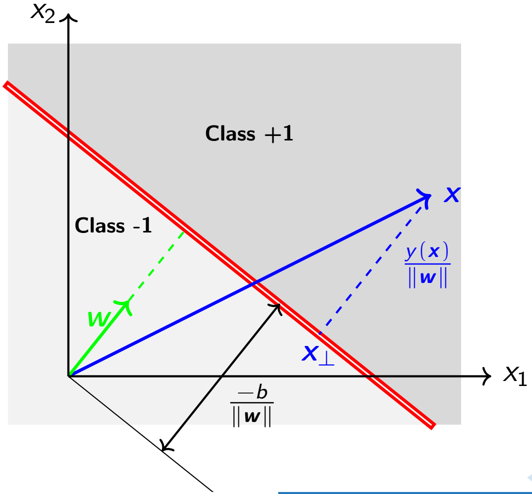
Discriminant Functions

- The simplest discriminant is a linear function of the form:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} + b = \sum_{j=1}^{m} w_j x_j + b = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + b \qquad (1)$$

- $\boldsymbol{w}$ is called the **weight vector** and b is the **bias**
- An input vector $\boldsymbol{x}$ is assigned class $\mathcal{C}_1$ if $y(\boldsymbol{x}) \geqslant 0$, class $\mathcal{C}_2$ otherwise
- The **decision boundary** is defined by the relation: $y(\boldsymbol{x}) = 0$
- The boundary is a $(D-1)$-dimensional hyperplane within the D-dimensional input space

Discriminant Functions (Ctd.)



Discriminant Functions (Ctd.)

- Consider two points $\mathbf{x}_A$ and $\mathbf{x}_B$ which lie on the decision surface
- Since $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$, we have $\mathbf{w}^\intercal(\mathbf{x}_A - \mathbf{x}_B) = 0$, hence $\mathbf{w}$ is **orthogonal** to every vector lying within the decision surface
- $\mathbf{w}$ determines the orientation of the decision surface
- Similarly, if $\mathbf{x}$ is a point on the decision surface, then $y(\mathbf{x}) = 0$ and the normal distance from the origin to the decision surface is given by:

$$\frac{\mathbf{w}^\intercal \mathbf{x} + b}{\|\mathbf{w}\|} = 0 \Leftrightarrow \frac{\mathbf{w}^\intercal \mathbf{x}}{\|\mathbf{w}\|} = -\frac{b}{\|\mathbf{w}\|} \tag{2}$$

- b controls the offset from the origin

Discriminant Functions (Ctd.)

- $y(\boldsymbol{x})$ gives a **signed measure** of the perp. distance of $\boldsymbol{x}$ to the boundary
- Consider an arbitrary point $\boldsymbol{x}$ and its orth. projection $\boldsymbol{x}_\perp$ onto the surface

$$\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \tag{3}$$

- Multiplying both sides by $\boldsymbol{w}^\intercal$ and adding b, and making use of:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\intercal\boldsymbol{x} + b \qquad \text{and} \qquad y(\boldsymbol{x}_\perp) = \boldsymbol{w}^\intercal\boldsymbol{x}_\perp + b = 0$$

- We get:

$$r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|} \tag{4}$$

Linear Separability

- We have n input vectors $\boldsymbol{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}\}$
- With corresponding target values $t^{(1)}, t^{(2)}, \ldots, t^{(n)}$ where $t^{(i)} \in \{-1, +1\}$
- New data points are classified according to the sign of $y(\boldsymbol{x})$: $sign(y(\boldsymbol{x}))$

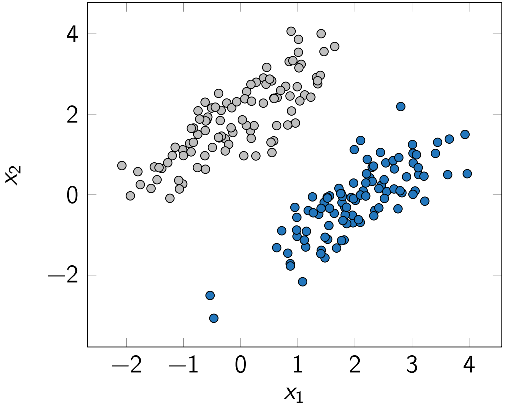
A data set is **linearly separable** in feature space, if $\exists(\boldsymbol{w}, b)$ such that

$$y(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b > 0 \qquad \forall \boldsymbol{x}^{(i)} \text{ with } t^{(i)} = +1 \tag{5}$$

$$y(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b < 0 \qquad \text{otherwise } (t^{(i)} = -1) \tag{6}$$

This can also be written as: $t^{(i)} y(\boldsymbol{x}^{(i)}) > 0 \ \forall i$

Example Data Set (linearly separable)



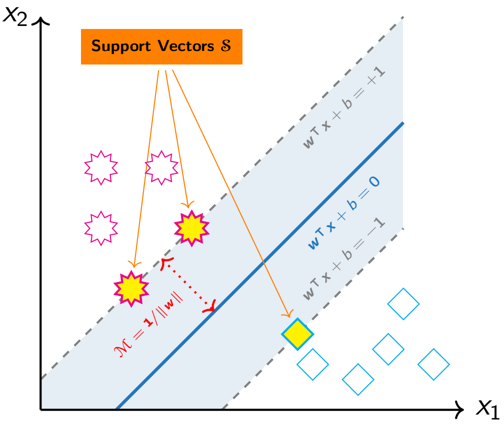
- This data set is **linearly separable** (you can find a straight line to separate the two classes)
- The possible number of hyperplanes is infinite...
- **Which hyperplane should be chosen?**

Maximum Margin Classifiers

- An SVM is a so-called **maximum margin classifier**
- It maximizes the margin $\mathcal{M}$

$$\max_{\boldsymbol{w},b} \mathcal{M}$$

- The larger $\mathcal{M}$ the less likely are false predictions
- Only the **support vectors** determine the hyperplane



Maximum Margin Classifiers (Ctd.)

- Recall the perpendicular distance of a point $\boldsymbol{x}$ to the hyperplane:

$$\frac{|y(\boldsymbol{x})|}{\|\boldsymbol{w}\|} \tag{7}$$

- Furthermore, we are only interested in solutions for which all data points are correctly classified, i. e. $t^{(i)} y(\boldsymbol{x}^{(i)}) > 0 \ \forall i$, thus the distance is given by:

$$\frac{t^{(i)} y(\boldsymbol{x}^{(i)})}{\|\boldsymbol{w}\|} = \frac{t^{(i)} (\boldsymbol{w}^{\intercal} \boldsymbol{x}^{(i)} + b)}{\|\boldsymbol{w}\|} \tag{8}$$

Maximum Margin Classifiers (Ctd.)

- The margin is given by the perp. distance to the closest data point $\boldsymbol{x}^{(i)}$
- We wish to optimize the parameters $\boldsymbol{w}$ and b to maximize this distance
- We have to solve:

$$\boldsymbol{w}^*, b^* = \arg\max_{\boldsymbol{w},b} \overbrace{\left\{ \frac{1}{\|\boldsymbol{w}\|} \overbrace{\min_i [t^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b)]}^{\text{'closest data points'}} \right\}}^{\text{'maximize distance to closest data points'}} \tag{9}$$

- Note that $1/\|\boldsymbol{w}\|$ does not depend on i
- A direct solution of this optimization would be very complex $\Rightarrow$ rewrite!

Maximum Margin Classifiers (Ctd.)

- We note that rescaling $\boldsymbol{w}$ and b by a factor κ **does not change the distance** to the decision boundary
- Therefore, for the points that are closest to the surface, we can set:

$$t^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) = 1 \tag{10}$$

- In this case, all data points $\boldsymbol{x}^{(i)}$ satisfy the constraint:

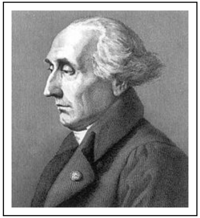
$$t^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) \geqslant 1 \qquad i = 1, 2, \dots, n \tag{11}$$

- It is sufficient to solve: $\arg\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|^2$ (1/2 for later mathematical convenience)

Maximum Margin Classifiers (Ctd.)

$$\boldsymbol{w}^*, b^* = \arg\min_{\boldsymbol{w}, b} \frac{1}{2} \|\boldsymbol{w}\|^2 \tag{12}$$

- This is a **quadratic optimization (QP)** problem
- A global optimum exists due to **convexity**
- How to solve such problems?



Lagrangian Optimization

(named after Joseph-Louis Lagrange)

Lagrangian Optimization: A simple Example

- Lagrangian optimization is optimization **subject to constraints**
- **Example:**

$$\overbrace{f(x_1, x_2) = 1 - x_1^2 - x_2^2}^{\text{function to optimize } f(\boldsymbol{x})} \quad \text{s. t.} \quad \overbrace{g(x_1, x_2) = x_1 + x_2 - 1 = 0}^{\text{linear constraint } g(\boldsymbol{x}) = 0} \tag{13}$$

- To find a solution we have to formulate the **Lagrangian equation**:

  General form: $\mathcal{L}(\boldsymbol{x}, \alpha) = f(\boldsymbol{x}) + \alpha g(\boldsymbol{x})$

$$\mathcal{L}(\boldsymbol{x}, \alpha) = 1 - x_1^2 - x_2^2 + \alpha(x_1 + x_2 - 1) \tag{14}$$

Lagrangian Optimization: A simple Example (Ctd.)

$$\mathcal{L}(\boldsymbol{x}, \alpha) = 1 - x_1^2 - x_2^2 + \alpha(x_1 + x_2 - 1)$$

We determine the partial derivatives w. r. t. x_1, x_2 and α and set them to zero:

$$\frac{\partial \mathcal{L}(\boldsymbol{x}, \alpha)}{\partial x_1} = -2x_1 + \alpha \overset{!}{=} 0 \tag{15}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{x}, \alpha)}{\partial x_2} = -2x_2 + \alpha \overset{!}{=} 0 \tag{16}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{x}, \alpha)}{\partial \alpha} = x_1 + x_2 - 1 \overset{!}{=} 0 \tag{17}$$

Lagrangian Optimization: A simple Example (Ctd.)

- Solving the first two equations for x_1 and x_2, respectively, we get:

$$x_1 = {}^{1}\!/_{2}\alpha \tag{18}$$
$$x_2 = {}^{1}\!/_{2}\alpha \tag{19}$$

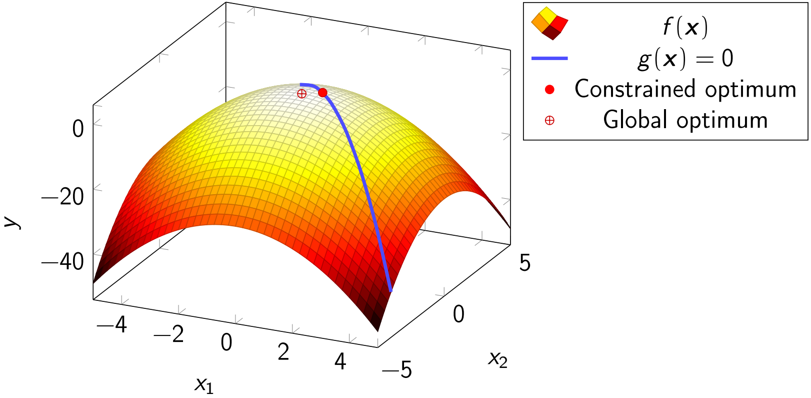
- Substitution into the third equation $x_1 + x_2 - 1 = 0$:

$$\widetilde{\mathcal{L}}(\alpha) = {}^{1}\!/_{2}\alpha + {}^{1}\!/_{2}\alpha - 1 = 0 \Leftrightarrow \alpha = 1 \tag{20}$$

- Finally, we get:

$$x_1 = {}^{1}\!/_{2} \qquad x_2 = {}^{1}\!/_{2}$$

Lagrangian Optimization: A simple Example (Ctd.)



SVM Parameter Optimization

We have to solve the Lagrangian:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \overbrace{1/2\|\boldsymbol{w}\|^2}^{f(\boldsymbol{x})} - \sum_{i=1}^{n} \alpha_i [\overbrace{t^{(i)} \cdot (\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) - 1}^{g(\boldsymbol{x})=0}] \tag{21}$$

- $\boldsymbol{\alpha}$ is a vector of **Lagrangian multipliers**
- There is **one constraint per data point!**
- The Lagrangian multipliers will be non-zero for all support vectors

SVM Parameter Optimization (Ctd.)

We have to compute the partial derivatives w. r. t. $\boldsymbol{w}$ and b and set them to zero:

linear combination of input!

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i t^{(i)} \boldsymbol{x}^{(i)} \stackrel{!}{=} 0 \Rightarrow \boxed{\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i t^{(i)} \boldsymbol{x}^{(i)}} \tag{22}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i t^{(i)} \stackrel{!}{=} 0 \Rightarrow \boxed{\sum_{i=1}^{n} \alpha_i t^{(i)} = 0} \tag{23}$$

SVM Parameter Optimization (Ctd.)

As a next step the partial derivatives are substituted in into $\mathcal{L}$:

$$\widetilde{\mathcal{L}}(\boldsymbol{\alpha}) = \frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i t^{(i)} \boldsymbol{x}^{(i)}\right)\left(\sum_{j=1}^{n} \alpha_j t^{(j)} \boldsymbol{x}^{(j)}\right) - \left(\sum_{i=1}^{n} \alpha_i t^{(i)} \boldsymbol{x}^{(i)}\right)\left(\sum_{j=1}^{n} \alpha_j t^{(j)} \boldsymbol{x}^{(j)}\right) - \sum_{i=1}^{n} \alpha_i t^{(i)} b + \sum_{i=1}^{n} \alpha_i \tag{24}$$

$$= \underbrace{\boxed{\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j t^{(i)} t^{(j)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle}}_{\text{Wolfe dual}} \quad \text{s. t. } \alpha_i \geqslant 0 \ \forall i \text{ and } \sum_{i=1}^{n} \alpha_i t^{(i)} = 0 \tag{25}$$

SVM Parameter Optimization (Ctd.)

- Once we know $\boldsymbol{\alpha}$, we can determine b by noting that any support vector satisfies $t^{(i)} y(\mathbf{x}^{(i)}) = 1$: ($\mathcal{S} \equiv$ indices of support vectors)

$$t^{(i)} \left(\sum_{j \in \mathcal{S}} \alpha_j t^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle + b \right) = 1 \tag{26}$$

- Average over all support vectors to compute b: ($n_{\mathcal{S}} \equiv$ number of support vectors)

$$b = \frac{1}{n_{\mathcal{S}}} \sum_{i \in \mathcal{S}} \left(t^{(i)} - \sum_{j \in \mathcal{S}} \alpha_j t^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \right) \tag{27}$$

Updated Decision Rule

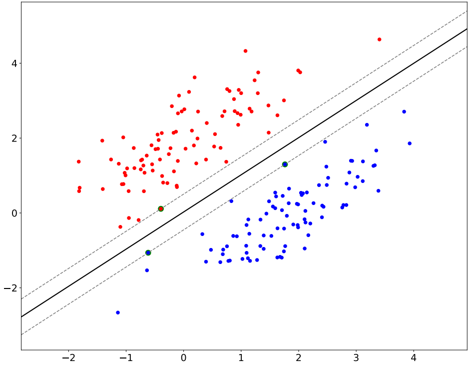
- Given our derivations, we can rewrite the SVM decision rule as follows:

$$y(\boldsymbol{x}) = sign\left(\sum_{i \in \mathcal{S}} \alpha_i t^{(i)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x} \rangle + b\right) \tag{28}$$

- $\boldsymbol{x}$ is an unknown instance for which the class label is not known

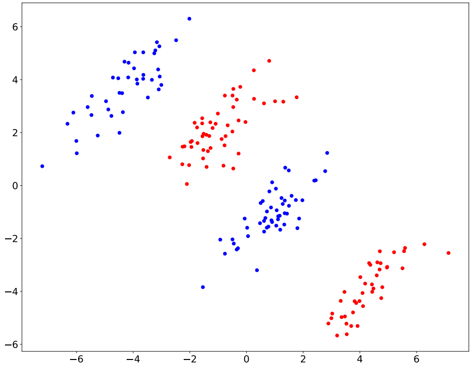
Since all α_i will be zero for non-support vectors, the decision for a class depends on the support vectors only! This makes predictions fast, even for large data sets. **The number of support vectors can also be used as an evaluation criterion.**

Linear SVM: Example



Non-Linear SVMs / Non-Linear Separability

- So far we have assumed **linear separability** of the data
- **What if the data is not linearly separable?**
  (which will be the case in practice...)
- We cannot find a straight line...
- $\overset{\textit{Remedy}}{\Longrightarrow}$ Feature maps, Kernels

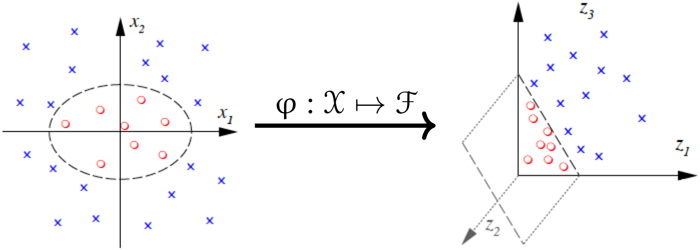


Feature Mapping

The mapping function φ maps from input space $\mathcal{X}$ to feature space $\mathcal{F}$:

$\mathcal{X}$

Input space



$\mathcal{F}$

Feature space

$$\varphi(x_1, x_2) \mapsto \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) = (z_1, z_2, z_3)$$

Feature Mapping (Ctd.)

- A feature map **explicitly** transforms the data to a higher dimension where classification becomes easier
- Computing the feature map can – from a computational point of view – become very expensive
- And how do you know how many dimensions to add? What transformations should be used?
- A more tractable solution is required $\Rightarrow$ Kernels

What is a Kernel?

- A kernel can be considered a **similarity function**
- Many algorithms have been *'kernelized'*, e. g. *Kernel PCA*, *Kernel SVM*
- Think of it as projecting the data in a higher dimensional space to make it linearly separable

A kernel allows the SVM to operate in a **high-dimensional, implicit feature space** without ever computing the coordinates of the data in that space, but rather by simply computing the **inner products** between the images of **all pairs of data** in the feature space. $\Rightarrow$ **Kernel trick** [Wikipedia]

What is a Kernel? (Ctd.)

- The explicit computation of a feature map $\varphi(\boldsymbol{x})$ is avoided...
- ...by replacing the dot product with the kernel $\mathcal{K}$:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') \Leftrightarrow \varphi(\boldsymbol{x})^\intercal \varphi(\boldsymbol{x}') \tag{29}$$

- Instead of mapping features explicitly, we calculate the **Gram matrix** $\boldsymbol{K} \in \mathbb{R}^{N \times N}$, where:

$$K_{ij} = \mathcal{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \tag{30}$$

Well-known Kernels

- Linear kernel

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\intercal}\boldsymbol{x}' \tag{31}$$

- Polynomial kernel

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\intercal}\boldsymbol{x}' + c)^{p} \tag{32}$$

- Radial-Basis-Function (RBF) kernel

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left\{-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right\} = \exp\{-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2\} \tag{33}$$

Power of Kernels

- Suppose $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^m$ with $m = 2$
- Polynomial feature mapping ($c = 0$):

$$\varphi(\boldsymbol{x}) = [x_1 x_1, x_1 x_2, x_2 x_1, x_2 x_2]^\mathsf{T} \qquad \varphi(\boldsymbol{x}') = [x_1' x_1', x_1' x_2', x_2' x_1', x_2' x_2']^\mathsf{T} \tag{34}$$

- Using a polynomial kernel:

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\mathsf{T} \boldsymbol{x}')^2 = \left(\sum_{i=1}^{m} x_i x_i' \right) \left(\sum_{j=1}^{m} x_j x_j' \right) = \sum_{i=1}^{m} \sum_{j=1}^{m} (x_i x_j)(x_i' x_j') = \varphi(\boldsymbol{x})^\mathsf{T} \varphi(\boldsymbol{x}') \tag{35}$$

We need $\mathcal{O}(n^2)$ to compute $\varphi(x)$ and $\varphi(\boldsymbol{x}')$ but only $\mathcal{O}(n)$ to compute $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$

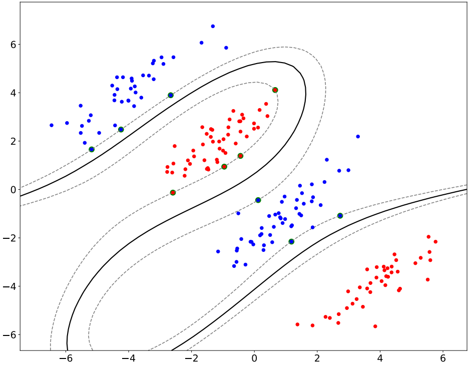
Incorporating the Kernel Function

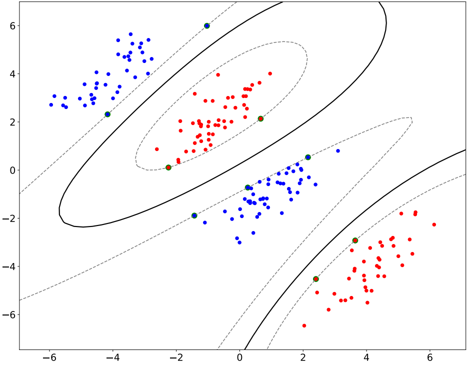
- The kernel function $\mathcal{K}$ replaces each occurrence of $\boldsymbol{x}^\intercal \boldsymbol{x}'$
- **Example:**

$$\widetilde{\mathcal{L}} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t^{(i)} t^{(j)} \mathcal{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \tag{36}$$

$$y(\boldsymbol{x}) = sign \left(\sum_{i \in \mathcal{S}} \alpha_i t^{(i)} \mathcal{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}) + b \right) \tag{37}$$

Polynomial Kernel vs. RBF Kernel





Mercer's Condition

- A kernel is valid, if it fulfills Mercer's condition
- This is the case, if for all **square-integrable functions** $g(\boldsymbol{x})$...

$$\int_{-\infty}^{\infty} |g(\boldsymbol{x})|^2 \, \mathrm{d}x < \infty \tag{38}$$

- ...it holds:

$$\int \int g(\boldsymbol{x}) \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') g(\boldsymbol{x}') \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{x}' \geqslant 0 \tag{39}$$

Mercer's Condition (Ctd.)

- Suppose $\mathcal{K}$ is a valid kernel and let $\boldsymbol{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$ be given
- For any vector $\boldsymbol{z} \in \mathbb{R}^n$:

$$\boldsymbol{z}^\mathsf{T}\boldsymbol{K}\boldsymbol{z} = \sum_i \sum_j z_i K_{ij} z_j = \sum_i \sum_j z_i \varphi(\boldsymbol{x}^{(i)})^\mathsf{T} \varphi(\boldsymbol{x}^{(j)}) z_j \tag{40}$$

$$= \sum_i \sum_j z_i \sum_k (\varphi(\boldsymbol{x}^{(i)}))_k (\varphi(\boldsymbol{x}^{(j)}))_k z_j = \sum_k \sum_i \sum_j z_i (\varphi(\boldsymbol{x}^{(i)}))_k (\varphi(\boldsymbol{x}^{(j)}))_k z_j \tag{41}$$

$$= \sum_k \left(\sum_i z_i \varphi(\boldsymbol{x}^{(i)})_k \right)^2 \geqslant 0 \Longrightarrow \boldsymbol{K} \geqslant 0 \tag{42}$$

- $\boldsymbol{K} \geqslant 0$ means that matrix $\boldsymbol{K}$ must be positive semi-definite (psd)

Mercer's Condition (Ctd.)

Mercer's Theorem:
$\mathcal{K}$ is a valid kernel, iff for any set of n training examples $\boldsymbol{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$ kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is **positive semi-definite**. The kernel is then called **Mercer kernel**.

- This entails: $K_{ij} \geqslant 0 \; \forall i, j$
- **Example**:
  - $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) = -1 \neq \varphi(\boldsymbol{x})^{\intercal} \varphi(\boldsymbol{x})$
  - $\mathcal{K}$ cannot be a valid kernel

Constructing new Kernels

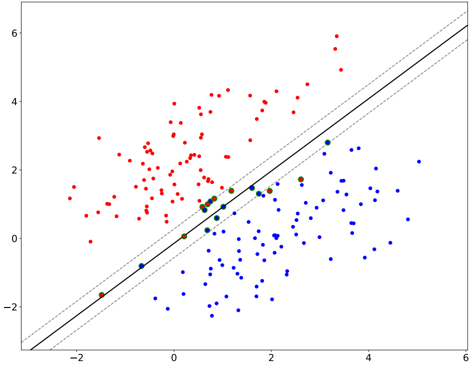
- It is not always easy to check if Mercer's condition is satisfied, but it is possible to construct **new kernels out of known ones**
- If $\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')$ and $\mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}')$ are valid kernels, so are:

  - $c \cdot \mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}')$
  - $\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}') + \mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}')$
  - $\mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}') \cdot \mathcal{K}_2(\boldsymbol{x}, \boldsymbol{x}')$
  - $f(\boldsymbol{x}) \cdot \mathcal{K}_1(\boldsymbol{x}, \boldsymbol{x}') \cdot f(\boldsymbol{x}')$
  - etc.

Overlapping Distributions

- We assumed linearly separable data
  $\Rightarrow$ **SVM gives exact solution**
- **But:** The classes may overlap
  $\Rightarrow$ **Exact separation leads to poor generalization**
- **Soft-margin SVM:** Allow some data points to be misclassified
- To this end, a penalty is introduced:
  - Misclassifications are penalized
  - This penalty increases linearly with the distance from the decision boundary
- This is done using **slack variables**

Overlapping Distributions: Example



Slack Variables

- The slack is denoted by ξ_i (where $\xi_i \geqslant 0; i = 1, \ldots, n$), one per data point
- **Different cases**:

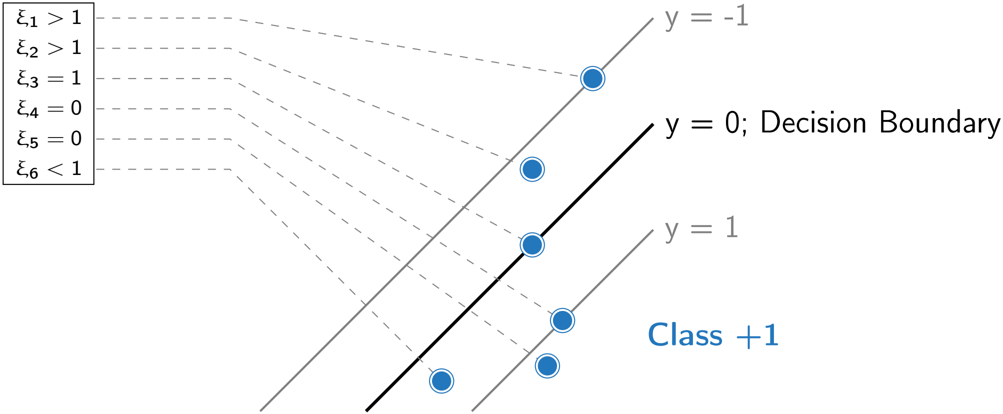
$$\begin{array}{ll} \xi_i = 0 & \text{if } \boldsymbol{x}^{(i)} \text{ is on or inside the correct margin boundary} \\ 0 < \xi_i < 1 & \text{if } \boldsymbol{x}^{(i)} \text{ lies inside the margin, but on the correct side} \\ \xi_i = 1 & \text{if } \boldsymbol{x}^{(i)} \text{ is on the decision boundary} \\ \xi_i > 1 & \text{if } \boldsymbol{x}^{(i)} \text{ lies on the wrong side of the decision boundary (misclassification)} \end{array}$$

- The classification constraints are replaced with:

$$t^{(i)} y(\boldsymbol{x}^{(i)}) \geqslant 1 - \xi_i \qquad i = 1, \ldots, n \tag{43}$$

- We get a soft-margin classifier

Slack Variables (Ctd.)



Soft SVM Parameter Optimization

- We want to maximize the margin, while softly **penalizing points which lie on the wrong side of the boundary**:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{s. t.} \quad t^{(i)} y(\boldsymbol{x}^{(i)}) \geqslant 1 - \xi_i \text{ and } \xi_i \geqslant 0 \ \forall i \tag{44}$$

- $C > 0$ controls the *'degree of softness'*, the larger C the more we penalize
- The Lagrangian function:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \{t^{(i)} y(\boldsymbol{x}^{(i)}) - 1 + \xi_i\} - \sum_{i=1}^{n} \mu_i \xi_i \tag{45}$$

Soft SVM Parameter Optimization (Ctd.)

- It turns out that the dual objective function looks exactly the same:

$$\widetilde{\mathcal{L}}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t^{(i)} t^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \tag{46}$$

- But the constraints differ slightly:

$$1) \quad \sum_{i=1}^{n} \alpha_i t^{(i)} = 0 \tag{47}$$

$$2) \quad \boxed{0 \leqslant \alpha_i \leqslant C \qquad i = 1, \ldots, n} \tag{48}$$

- Constraint 2) is called **boxed constraint**

Section:

Wrap-Up

Summary

Lecture Overview

Unit I: Machine Learning Introduction

Self-Test Questions

Recommended Literature and further Reading

Thank you very much for the attention!

Topic: *** Applied Machine Learning Fundamentals *** Support Vector Machines
Date: October 24, 2019

Contact:

Daniel Wehner (D062271)
SAP SE
daniel.wehner@sap.com

Do you have any questions?