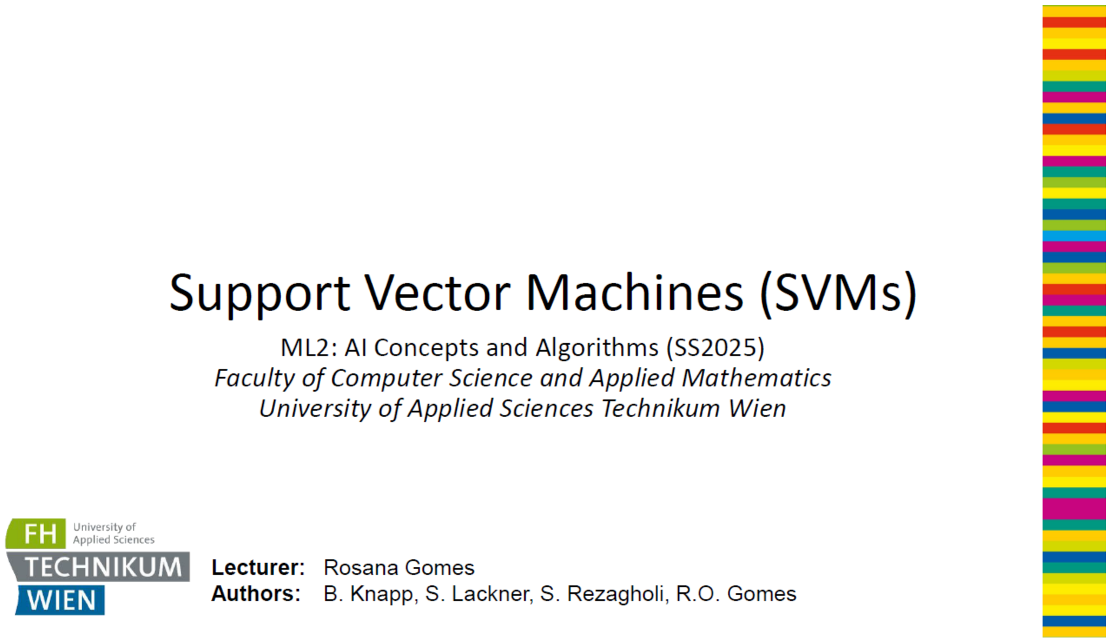
SVM

June 22, 2025



Support Vector Machines (SVMs)

ML2: AI Concepts and Algorithms (SS2025)

Faculty of Computer Science and Applied Mathematics

University of Applied Sciences Technikum Wien

Lecturer: Rosana Gomes

Authors: B. Knapp, S. Lackner, S. Rezagholi, R.O. Gomes

Good [morning/afternoon], everyone.

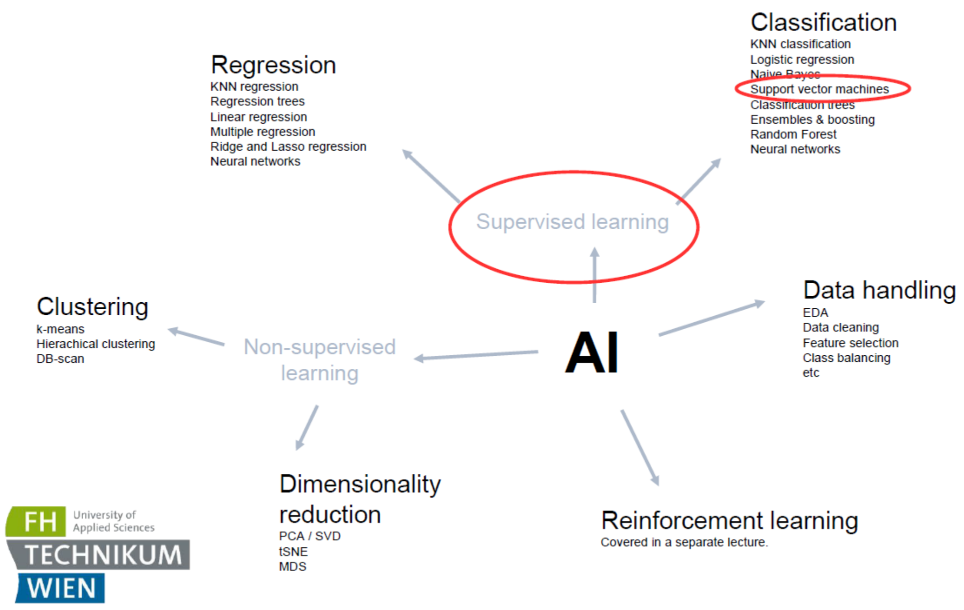
Welcome to today's presentation on **Support Vector Machines**, commonly known as **SVMs**.

This talk is part of the course **ML2: AI Concepts and Algorithms (SS2025)**, offered by the *Faculty of Computer Science and Applied Mathematics* at the **University of Applied Sciences Technikum Wien**.

The materials presented today were developed by a team of authors: **B. Knapp, S. Lackner, S. Rezagholi, and R.O. Gomes**, with **Rosana Gomes** as the lecturer for this module.

Let's begin our journey into one of the most powerful and elegant algorithms in machine learning: **Support Vector Machines**.

Please go ahead and show me the next slide.



On this slide, we see the **big picture of Artificial Intelligence (AI)** and where **Support Vector Machines (SVMs)** fit in.

The field of AI branches into several major subfields, such as:

- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**
- **Dimensionality Reduction**
- **Data Handling**

SVMs are part of the **Supervised Learning** branch, which is highlighted here in red. In supervised learning, we train models using labeled data. That means we have inputs and known outputs, and the goal is for the algorithm to learn how to map inputs to outputs.

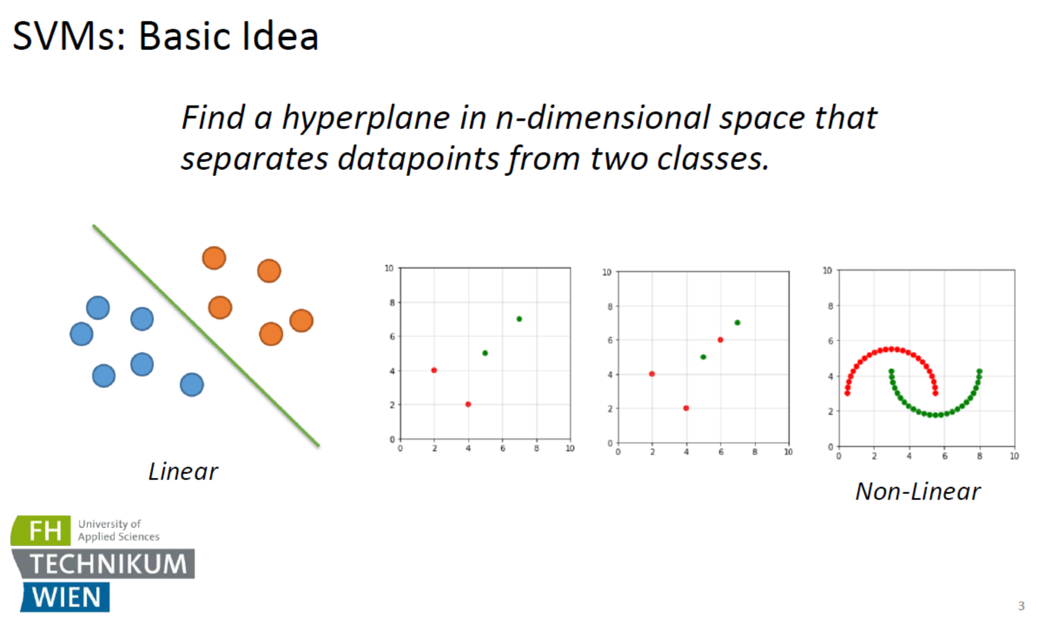
Within supervised learning, we find two major problem types:

- **Regression**, where the output is continuous.
- **Classification**, where the output is discrete.

SVMs belong to **Classification**, as shown on the right side, along with algorithms like KNN, logistic regression, and neural networks. In this context, SVMs are used to **classify data points into distinct categories**, often by finding the optimal decision boundary between them.

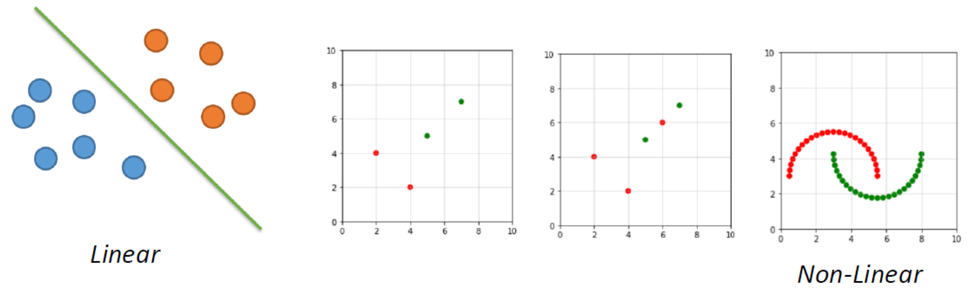
We'll now dive deeper into how SVMs work and why they are so powerful in separating data—even when it's not linearly separable.

Next slide, please.



SVMs: Basic Idea

Find a hyperplane in n-dimensional space that separates datapoints from two classes.



FH University of Applied Sciences TECHNIKUM WIEN

3

Let’s now explore the **basic idea** behind Support Vector Machines.

At the core, **SVMs aim to find a hyperplane in an n-dimensional space that best separates data points belonging to two different classes**.

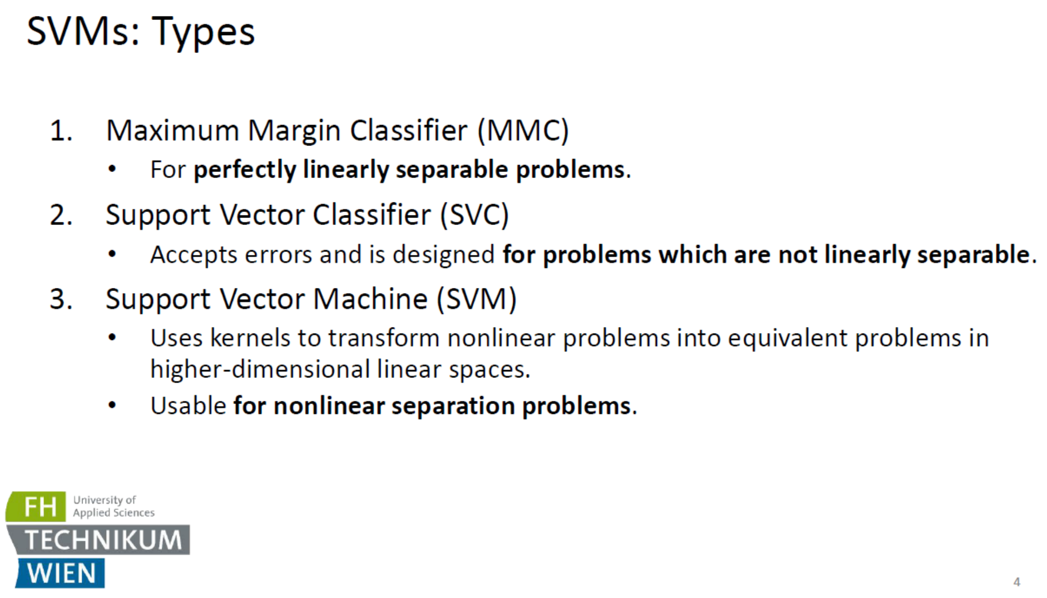
- On the **left**, you see a simple **linear separation**: two groups of points (orange and blue) can be separated with a **straight line**—or more generally, a *hyperplane* in higher dimensions. In this case, SVM would look for the **optimal hyperplane** that maximizes the margin between the two classes.
- The **middle plots** show examples where the separation is less obvious. When the classes aren’t cleanly separated, SVMs can still find a **soft margin** that tolerates some misclassifications but still aims for the best separation.
- On the **right**, we see a **non-linear case**, where a straight line—or even a flat hyperplane—cannot separate the classes. This is where SVMs show their power by applying the **kernel trick**, which maps data into a higher-dimensional space where a hyperplane *can* separate the classes.

In summary, SVMs are versatile:

- They work in both linear and non-linear settings.
- They focus on finding the *maximum margin separator*.
- And they are especially effective in high-dimensional spaces.

Let’s now see how this is achieved mathematically and geometrically.

Next slide, please.



SVMs: Types

1. Maximum Margin Classifier (MMC)
   - For **perfectly linearly separable problems**.
2. Support Vector Classifier (SVC)
   - Accepts errors and is designed **for problems which are not linearly separable**.
3. Support Vector Machine (SVM)
   - Uses kernels to transform nonlinear problems into equivalent problems in higher-dimensional linear spaces.
   - Usable **for nonlinear separation problems**.

FH University of Applied Sciences TECHNIKUM WIEN

4

Now let's look at the **types of Support Vector Machines**, each adapted to a specific kind of classification scenario:

0.0.1 1. Maximum Margin Classifier (MMC)

- This is the **simplest form** of SVM.
- It works **only when the data is perfectly linearly separable**, meaning there exists a straight line (or hyperplane) that can separate the classes with no errors.
- It finds the **maximum-margin hyperplane**, the one that is farthest from any data point.

0.0.2 2. Support Vector Classifier (SVC)

- Real-world data is **rarely perfectly separable**. That's where SVC comes in.
- It **allows for some misclassifications**—introducing the concept of a "soft margin."
- This model balances **maximizing the margin** and **minimizing classification errors**, by penalizing misclassified points.

0.0.3 3. Support Vector Machine (SVM)

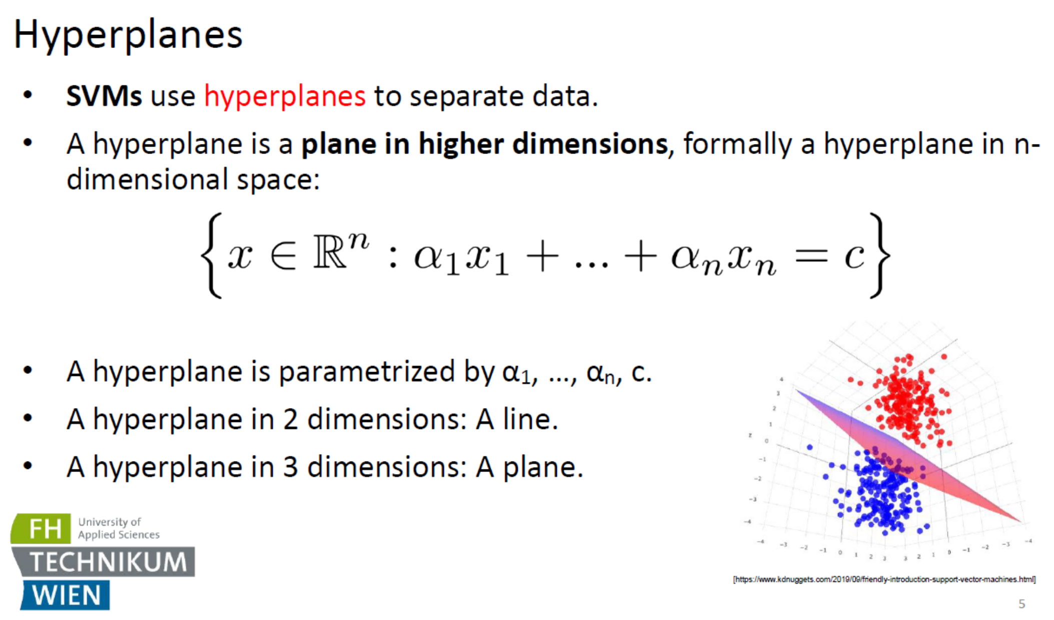
- This is the most **general and powerful form**.
- It handles **non-linear separations** by using the **kernel trick**: it transforms the input space into a higher-dimensional space where a linear separator can be found.
- This makes SVMs **very effective for complex datasets** where classes are intertwined in the original space.

In summary:

- **MMC** = hard-margin SVM, only for ideal cases.
- **SVC** = soft-margin SVM, practical for real-world noisy data.
- **SVM** = the full toolset, with kernel methods for non-linear problems.

Let's now look into the **geometry and mathematics** of how this separation is achieved.

Next slide, please.

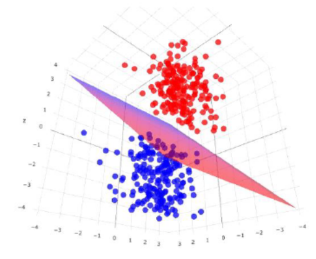


Hyperplanes

- **SVMs** use hyperplanes to separate data.
- A hyperplane is a **plane in higher dimensions**, formally a hyperplane in n-dimensional space:

$$\left\{x \in \mathbb{R}^n : \alpha_1 x_1 + \ldots + \alpha_n x_n = c\right\}$$

- A hyperplane is parametrized by $\alpha_1, \ldots, \alpha_n$, c.
- A hyperplane in 2 dimensions: A line.
- A hyperplane in 3 dimensions: A plane.



[https://www.kdnuggets.com/2019/09/friendly-introduction-support-vector-machines.html]

5

Let's now focus on the key geometric concept behind SVMs: the **hyperplane**.

0.0.4 What is a hyperplane?

- In the context of SVMs, a **hyperplane** is the decision boundary that **separates data points from different classes**.
- Mathematically, a hyperplane in an n-dimensional space is defined by the equation:

$$x \in \mathbb{R}^n : \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n = c$$

This equation defines a flat surface that splits the space into two halves.

0.0.5 Dimensional Examples:

- In **2D**, a hyperplane is just a **line**.
- In **3D**, it's a **flat plane**.
- In higher dimensions, it becomes harder to visualize, but the concept remains the same: a flat, (n—1)-dimensional surface that divides the n-dimensional space.

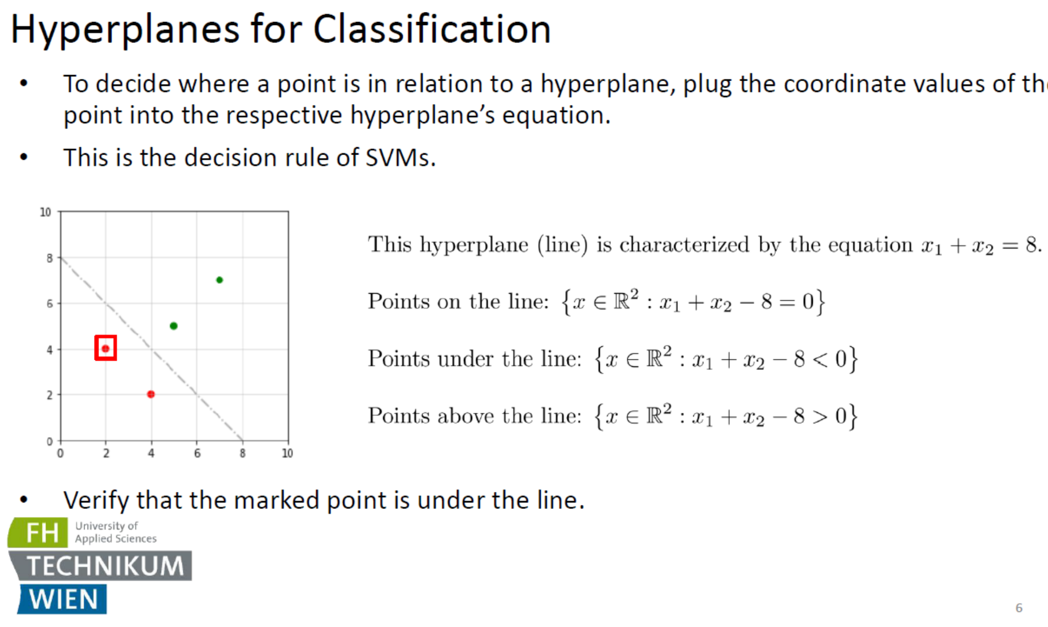
0.0.6 Role in SVMs:

- SVMs aim to **find the optimal hyperplane** that **maximizes the margin**—the distance between the hyperplane and the nearest data points from each class (the so-called *support vectors*).
- The image on the bottom right shows this idea in 3D, where a plane separates two clouds of red and blue points.

In essence, the **hyperplane is the SVM's decision boundary**, and all its power lies in finding the one that best separates the data—with the largest margin and, if necessary, in a transformed (kernel) space.

Next, we'll look at how this hyperplane maximizes the separation using the **margin**.

Next slide, please.



Hyperplanes for Classification

- To decide where a point is in relation to a hyperplane, plug the coordinate values of the point into the respective hyperplane's equation.
- This is the decision rule of SVMs.

This hyperplane (line) is characterized by the equation $x_1 + x_2 = 8$.

Points on the line: $\{x \in \mathbb{R}^2 : x_1 + x_2 - 8 = 0\}$

Points under the line: $\{x \in \mathbb{R}^2 : x_1 + x_2 - 8 < 0\}$

Points above the line: $\{x \in \mathbb{R}^2 : x_1 + x_2 - 8 > 0\}$

- Verify that the marked point is under the line.

FH University of Applied Sciences TECHNIKUM WIEN

6

Let's now see how hyperplanes are used for **classification decisions** in SVMs.

0.0.7 The Decision Rule

To determine on which side of the hyperplane a data point lies, we simply **plug the coordinates of that point into the hyperplane equation**.

In this example, the hyperplane (which is a line in 2D) is defined by:

$$x_1 + x_2 = 8$$

We can rewrite this as:

$$x_1 + x_2 - 8 = 0$$

This leads to the decision rule:

- **Points *on* the line** satisfy $x_1 + x_2 - 8 = 0$
- **Points *under* the line** satisfy $x_1 + x_2 - 8 < 0$
- **Points *above* the line** satisfy $x_1 + x_2 - 8 > 0$

0.0.8 Example: The Marked Point

Let's now verify the **red square-marked point**, located at (3, 4):

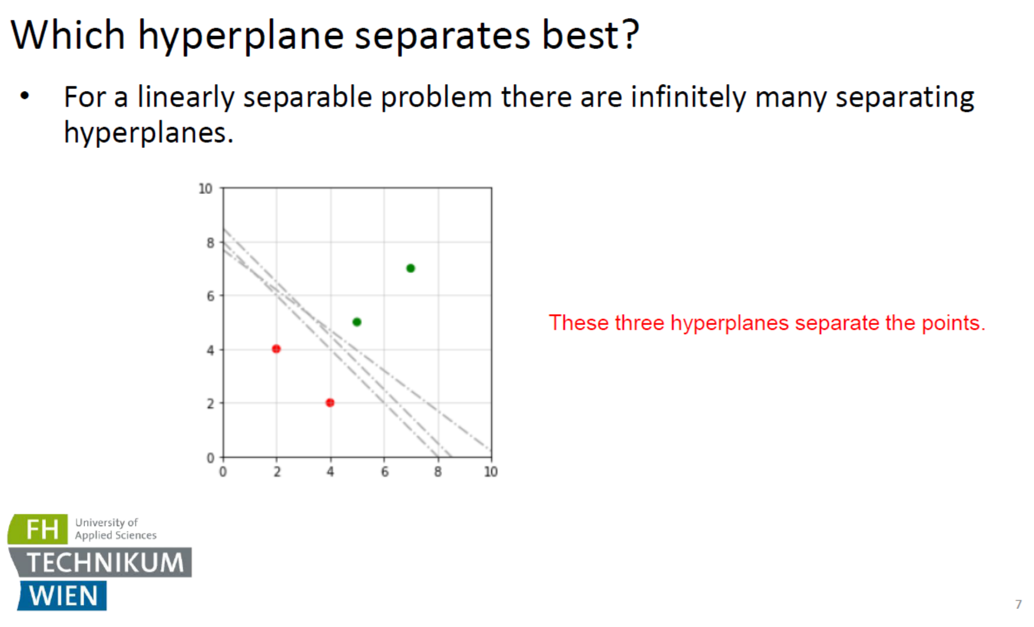
$$x_1 = 3, \quad x_2 = 4 \quad \Rightarrow \quad 3 + 4 - 8 = -1$$

Since $-1 < 0$, the point is **under the line**, which confirms the statement in the slide.

This rule is the **core of classification with SVMs**: the sign of the equation tells us on which side of the decision boundary the point lies, and therefore which class it belongs to.

In the next slide, we'll explore how to find not just any hyperplane, but the **optimal one** that offers the **maximum margin** between classes.

Please proceed to the next slide.



Which hyperplane separates best?

- For a linearly separable problem there are infinitely many separating hyperplanes.

These three hyperplanes separate the points.

FH University of Applied Sciences TECHNIKUM WIEN

7

Now that we know what a hyperplane is, a natural question arises:

Which hyperplane should we choose?

0.0.9 Many Hyperplanes Can Separate the Data

As this slide shows, when a dataset is **linearly separable**, there are **infinitely many hyperplanes** that can separate the two classes.

In the figure, we see **three dashed lines**, each of which separates the green and red points correctly. However, they are not all equally good.

0.0.10 Why Not Just Any Hyperplane?

Choosing a hyperplane that's too close to one of the classes may result in poor **generalization**—the model might work on the training set but fail to classify new data correctly.

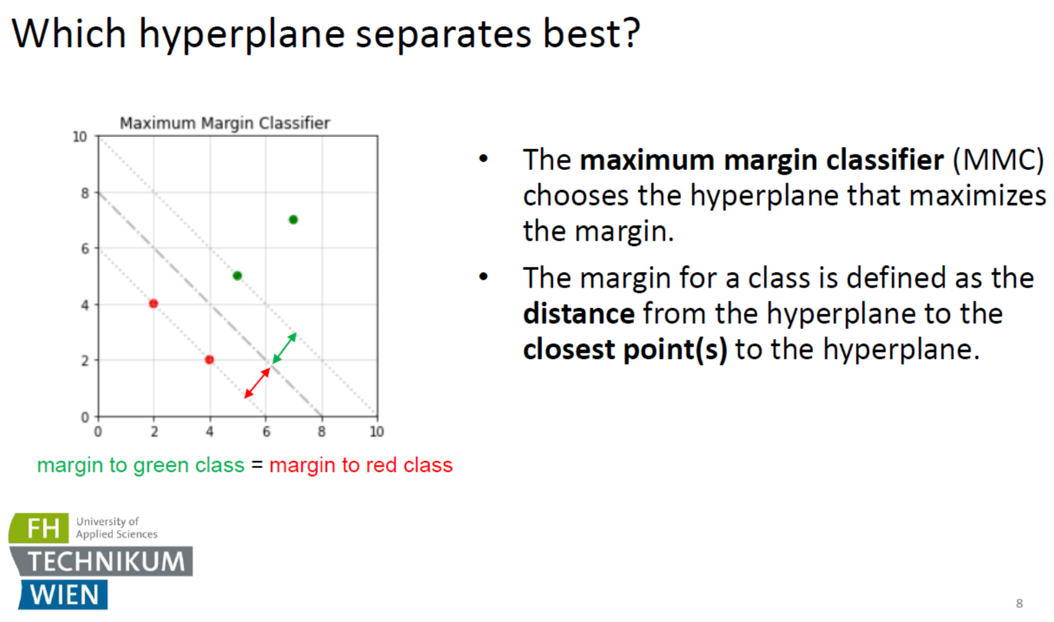
That's why we don't just want **any** hyperplane—we want the **best** one.

0.0.11 The Goal of SVMs

SVMs aim to **select the hyperplane that maximizes the margin**—that is, the one that lies as far away as possible from the nearest points of both classes. This maximized margin helps the model **generalize better** to unseen data.

This brings us to the key geometric concept behind SVMs: the **margin** and the **support vectors**.

Let's move to the next slide to see how that works.


Maximum Margin Classifier
0
2
4
6
8
10

Which hyperplane separates best?

- The **maximum margin classifier** (MMC) chooses the hyperplane that maximizes the margin.
- The margin for a class is defined as the **distance** from the hyperplane to the **closest point(s)** to the hyperplane.

margin to green class = margin to red class

8

Here, we answer the key question:

Which hyperplane is best?

0.0.12 Maximum Margin Classifier (MMC)

Among all possible separating hyperplanes, the **Maximum Margin Classifier** chooses the one that **maximizes the margin**.

- The **margin** is the distance from the hyperplane to the **closest data points** from either class.
- These closest points are called **support vectors**, and they are the only points that matter in determining the hyperplane.
- In this example, the margin is shown symmetrically with green and red arrows—from the decision boundary to the nearest green and red points.

0.0.13 Why Maximize the Margin?

- A larger margin generally leads to **better generalization** on unseen data.
- It gives the model more **tolerance to noise** and avoids overfitting to specific training examples.

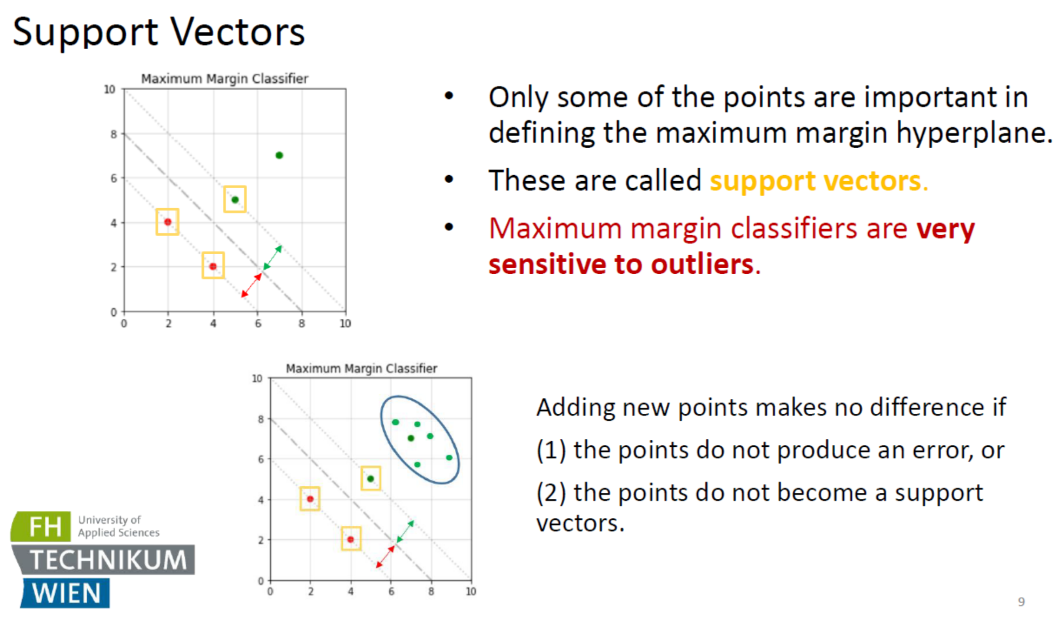
0.0.14 Visual Summary

- The solid black line is the **optimal hyperplane**.
- The two dashed lines represent the **margin boundaries**.
- The red and green arrows show that the margin to each class is **equal**—which is a key characteristic of the maximum margin solution.

In essence, **SVMs don't just look for any separation—they look for the most confident separation possible**.

Next, we'll formalize how this idea is expressed as an optimization problem.

Please proceed to the next slide.



Support Vectors

- Only some of the points are important in defining the maximum margin hyperplane.
- These are called support vectors.
- Maximum margin classifiers are **very sensitive to outliers**.

Adding new points makes no difference if

(1) the points do not produce an error, or

(2) the points do not become a support vectors.

FH University of Applied Sciences TECHNIKUM WIEN

9

Let’s now talk about the **key players in an SVM model**: the **Support Vectors**.

0.0.15 What are Support Vectors?

- When fitting the maximum margin hyperplane, **not all data points matter**.
- **Only the points closest to the margin** are critical—these are called **support vectors**.
- In the upper plot, these support vectors are highlighted with yellow squares.

They lie **on the margin boundaries**, and if you removed or changed any of them, the optimal hyperplane would **change**.

0.0.16 Why are Support Vectors Important?

- They **define the margin**.
- The model’s decision boundary is **completely determined** by these few points.
- Points **far from the margin** don’t influence the hyperplane at all.

0.0.17 Sensitivity to Outliers

- Because SVMs rely so heavily on support vectors, they can be **very sensitive to outliers**.
- A single mislabeled or noisy point **near the margin** can drastically shift the boundary.

In the **lower plot**, additional green points are added—but because they’re far from the decision boundary and don’t affect the margin, they are **ignored** by the model.

0.0.18 Conclusion

Adding more data points doesn't change the SVM result **unless**:

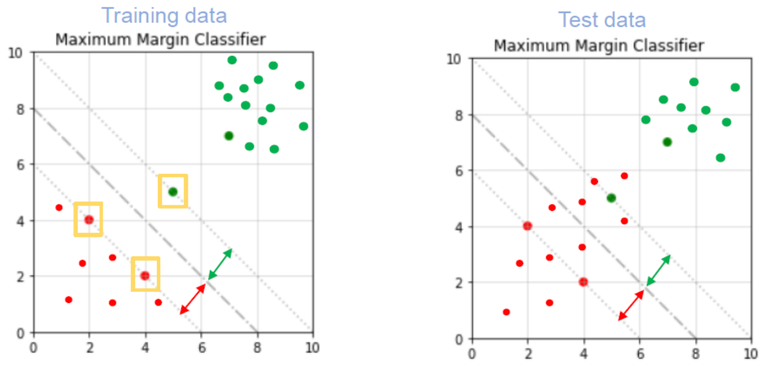
1. The new point **introduces a classification error**, or
2. It becomes a **new support vector** by lying on or near the margin.

In the next slide, we'll formalize this with the **optimization problem** that SVMs solve.

Next slide, please.

Support Vectors

One outlier can have large influence.



FH University of Applied Sciences TECHNIKUM WIEN

This slide shows a **critical weakness** of the **Maximum Margin Classifier**: its **sensitivity to outliers**.

0.0.19 Left Plot: Training Data with One Outlier

- We see a training dataset where one **green point** lies far from the cluster—marked as a support vector.
- This **single outlier** pulls the decision boundary **downward**, affecting the placement of the margin and the hyperplane.
- The result: the model prioritizes separating this **anomalous point** rather than following the natural distribution of the data.

0.0.20 Right Plot: Test Data

- When tested on new data, we observe the **consequences** of the distorted boundary.
- The **decision line** is now poorly positioned—classifying some test points incorrectly.
- **Generalization suffers**, even though training accuracy might look perfect.

0.0.21 Key Insight

> One outlier—if it becomes a support vector—can have a **disproportionate influence** on the model.

This is a major limitation of **hard-margin SVMs**, motivating the need for **soft-margin classifiers**, which **allow some flexibility** by tolerating small classification errors.

Next, we'll formally define how we find the optimal hyperplane using an optimization problem—with and without such flexibility.

Please continue to the next slide.

Let's now formalize the problem that the **Maximum Margin Classifier (MMC)** solves.

0.0.22 Objective: Find the Optimal Hyperplane

To calculate the maximum margin hyperplane, we need to:

1. **Encode the target classes** as:

$$y_i \in \{-1, +1\}$$

2. Use the **hyperplane equation**:

$$\alpha_1 x_1 + \cdots + \alpha_n x_n - c = 0$$

   This equation separates the space into:

   - > 0 for class $+1$
   - < 0 for class -1

0.0.23 Interpretation of Variables

- The x**-vector** refers to a data point.
- The α**-vector** determines the **orientation** of the hyperplane.
- The c parameter shifts the hyperplane without changing its direction (i.e., it controls the **bias**).

The key requirement is:

$$y_i \left(b + \alpha_1 x_{i1} + \cdots + \alpha_p x_{ip} \right) > 0 \quad \text{for all training points } i$$

This expression ensures that **all points are classified correctly**, with a positive margin.

0.0.24 What Does the MMC Do?

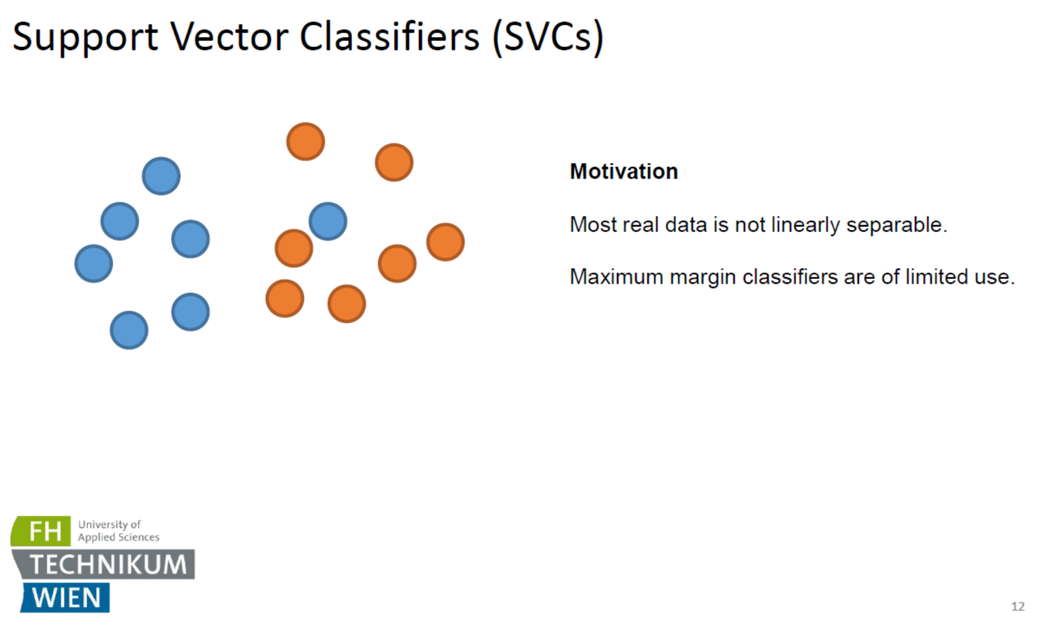
It **maximizes the margin**—the distance to the closest point—**while satisfying this inequality for every observation**.

This becomes an optimization problem:

- **Objective**: maximize margin (or minimize norm of α).
- **Constraint**: each point lies on the correct side of the hyperplane with at least margin 1.

Next, we'll see how this optimization problem is formulated more precisely—with constraints and a cost function.

Please continue to the next slide.



Support Vector Classifiers (SVCs)

Motivation

Most real data is not linearly separable.

Maximum margin classifiers are of limited use.

FH University of Applied Sciences TECHNIKUM WIEN

12

With this slide, we are introduced to the next step in the evolution of SVMs:

> **Support Vector Classifiers (SVCs)**

0.0.25 Motivation: Real-World Data Isn't Perfect

As shown in the diagram:

- The blue and orange classes are **not linearly separable**—some points are overlapping.
- **No straight line** can divide the two groups **without errors**.

This reflects the **real world**, where:

- Data is **noisy**,
- There may be **labeling errors**,
- And classes often **overlap**.

0.0.26 Limitation of the Maximum Margin Classifier (MMC)

- MMC **requires perfect separation**.
- But in most cases, **this isn't feasible**—so we need a more flexible approach.

0.0.27 What SVC Brings

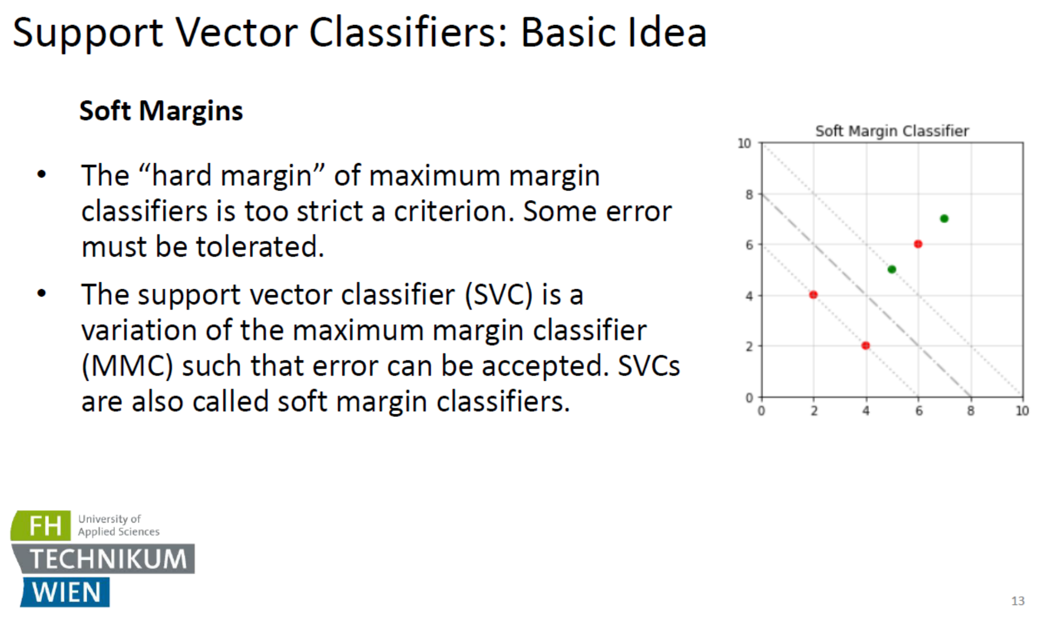
Support Vector Classifiers introduce the idea of a **"soft margin"**, where:

- Some points **can be on the wrong side** of the margin or even misclassified.
- These violations are allowed but **penalized** in the optimization.

This balance between **margin maximization** and **error tolerance** allows the classifier to be **robust to outliers** and **more useful in practice**.

In the next slide, we'll see **how this is done mathematically** by adjusting the optimization problem to include **slack variables**.

Next slide, please.



Support Vector Classifiers: Basic Idea

Soft Margins

- The "hard margin" of maximum margin classifiers is too strict a criterion. Some error must be tolerated.
- The support vector classifier (SVC) is a variation of the maximum margin classifier (MMC) such that error can be accepted. SVCs are also called soft margin classifiers.

Soft Margin Classifier
0
2
4
6
8
10

FH University of Applied Sciences TECHNIKUM WIEN

13

Now we explore the **core idea** behind **Support Vector Classifiers (SVCs)**:

> The use of **Soft Margins**.

0.0.28 Why Soft Margins?

- The **hard margin** used by the Maximum Margin Classifier (MMC) is **too strict**.

- It assumes the data is **perfectly separable** with no errors—an unrealistic expectation in most real-world scenarios.
- **Some error must be tolerated**, especially when:
  - Classes overlap
  - There is noise
  - Or outliers exist

0.0.29 What is a Soft Margin?

- A **soft margin** allows **some points to lie within the margin** or even on the **wrong side of the hyperplane**.
- These violations are allowed, but the model **penalizes them**.
- The **trade-off** is controlled by a regularization parameter C (to be introduced later).

0.0.30 The Plot: Soft Margin in Action

In the visual:

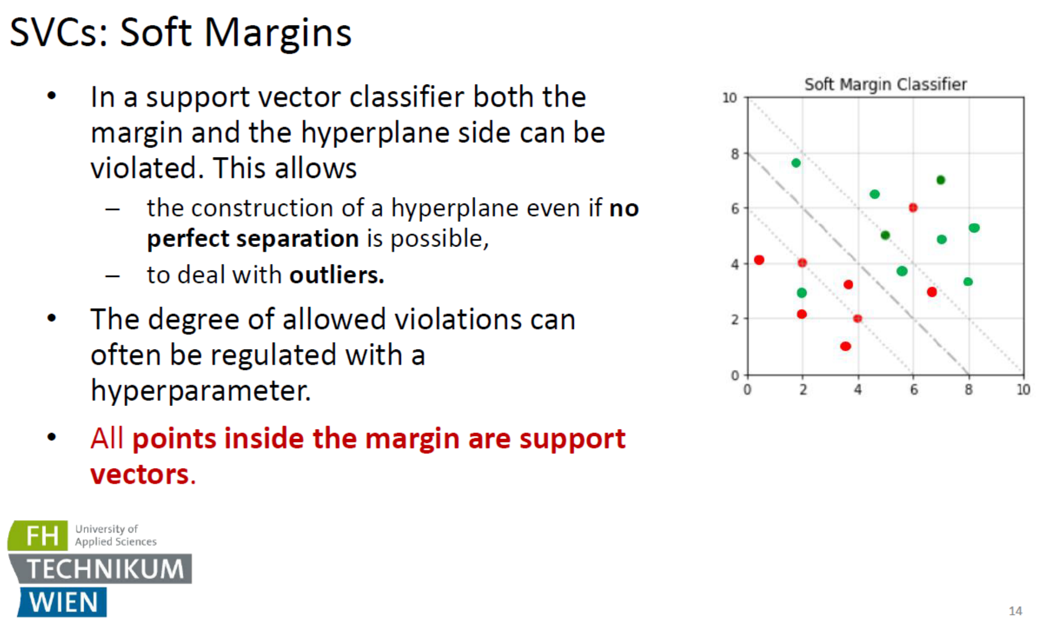
- Some **red and green points** are inside the margin or misclassified.
- Still, the **gray dashed line** finds the best compromise between **maximum margin** and **minimal classification error**.

0.0.31 Conclusion

Support Vector Classifiers (SVCs), also called **soft margin classifiers**, strike a **balance between flexibility and separation**—making them suitable for **practical, noisy, real-world datasets**.

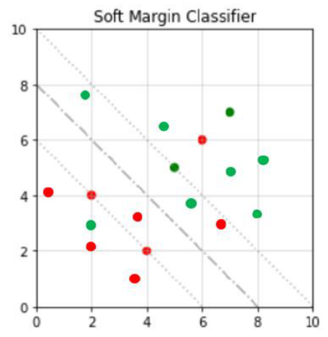
In the next slide, we'll define the **optimization problem** that SVC solves and how it introduces **slack variables** to allow those controlled violations.

Please proceed.



SVCs: Soft Margins

- In a support vector classifier both the margin and the hyperplane side can be violated. This allows
  - the construction of a hyperplane even if **no perfect separation** is possible,
  - to deal with **outliers.**
- The degree of allowed violations can often be regulated with a hyperparameter.
- All **points inside the margin are support vectors.**



FH University of Applied Sciences TECHNIKUM WIEN

14

This slide deepens our understanding of **Soft Margins** in Support Vector Classifiers (SVCs).

0.0.32 What Does "Soft Margin" Mean in Practice?

In SVCs:

- The **margin boundaries** can be **violated**—points can lie **inside the margin** or even **on the wrong side** of the hyperplane.
- This flexibility allows the classifier to:
  - **Construct a hyperplane** even when **perfect separation** is impossible.
  - **Tolerate outliers**, rather than letting them distort the model entirely.

0.0.33 Controlling the Tolerance

- The **amount of allowed violation** is controlled by a **hyperparameter**, usually denoted as C.
- A **high** C tries to avoid violations → closer to hard margin behavior.
- A **low** C allows more violations → softer margin, more flexibility.

0.0.34 Support Vectors in Soft Margins

All points inside the margin are support vectors.

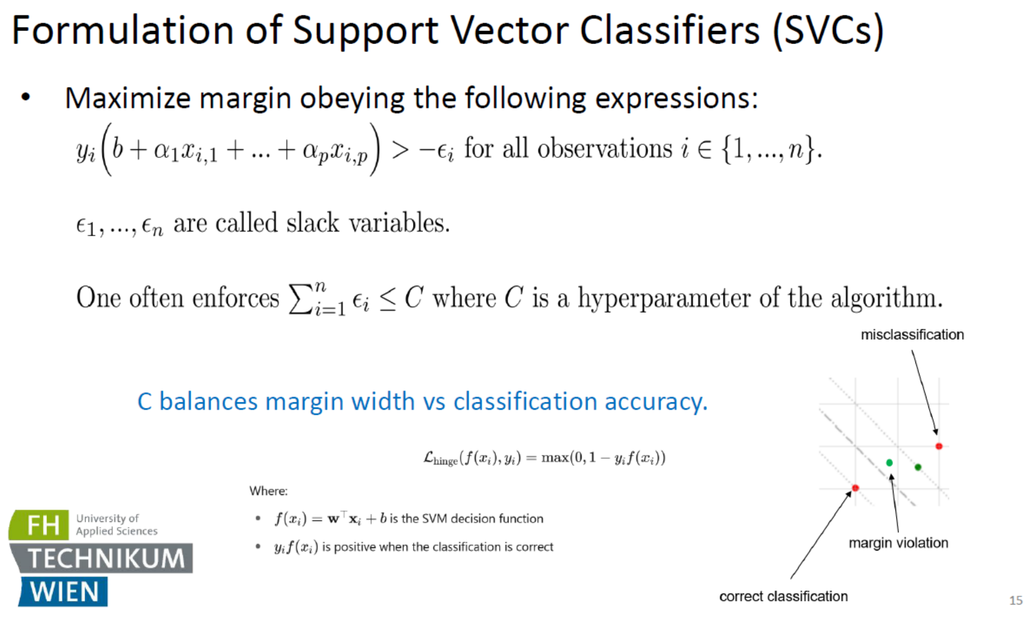
- That includes:
  - Points **on the margin**,

  - Points **between the margin and the hyperplane**, and
  - Even points **on the wrong side of the hyperplane** (misclassified).

This is visible in the plot on the right: many red and green points lie within or across the margin boundaries, but they all influence the decision boundary.

In the next slide, we will formalize this with the **slack variables** and the modified **optimization problem** that SVC solves.

Please go ahead with the next slide.



Formulation of Support Vector Classifiers (SVCs)

- Maximize margin obeying the following expressions:
  $y_i\Big(b + \alpha_1 x_{i,1} + \ldots + \alpha_p x_{i,p}\Big) > -\epsilon_i$ for all observations $i \in \{1, \ldots, n\}$.

  $\epsilon_1, \ldots, \epsilon_n$ are called slack variables.

  One often enforces $\sum_{i=1}^{n} \epsilon_i \leq C$ where C is a hyperparameter of the algorithm.

C balances margin width vs classification accuracy.

$$\mathcal{L}_{\text{hinge}}(f(x_i), y_i) = \max(0, 1 - y_i f(x_i))$$

Where:

- $f(x_i) = \mathbf{w}^\top \mathbf{x}_i + b$ is the SVM decision function
- $y_i f(x_i)$ is positive when the classification is correct

FH University of Applied Sciences TECHNIKUM WIEN

15

Now we formalize the **optimization problem** behind Support Vector Classifiers (SVCs), which implement **soft margins**.

0.0.35 Objective

We aim to:

> **Maximize the margin**, while allowing **some violations**, and **penalizing** them.

This leads to the modified constraint:

$$y_i \left(b + \alpha_1 x_{i1} + \cdots + \alpha_p x_{ip}\right) > -\epsilon_i \quad \text{for all } i = 1, \ldots, n$$

Here:

- $y_i \in \{-1, +1\}$ is the class label
- $\epsilon_i \geq 0$ is a **slack variable** for point i

0.0.36 Slack Variables ϵ_i

- $\epsilon_i = 0$: point lies correctly outside the margin
- $0 < \epsilon_i < 1$: point is **inside** the margin but still correctly classified
- $\epsilon_i > 1$: point is **misclassified**

0.0.37 Controlling the Trade-off

To control the total amount of allowed error, we constrain:

$$\sum_{i=1}^{n} \epsilon_i \leq C$$

- C is a **hyperparameter**
- A **larger** C punishes errors more harshly → **smaller margin, fewer violations**
- A **smaller** C is more forgiving → **larger margin, more violations**

This **balances margin width with classification accuracy**, as highlighted in blue.

0.0.38 Loss Function: Hinge Loss

The SVM uses the **hinge loss** to measure how much each data point violates the margin:

$$\mathcal{L}_{\text{hinge}}(f(x_i), y_i) = \max(0, 1 - y_i f(x_i))$$

Where:

- $f(x_i) = w^\top x_i + b$
- If a point is correctly classified and outside the margin → zero loss
- Otherwise → positive penalty

0.0.39 Illustration

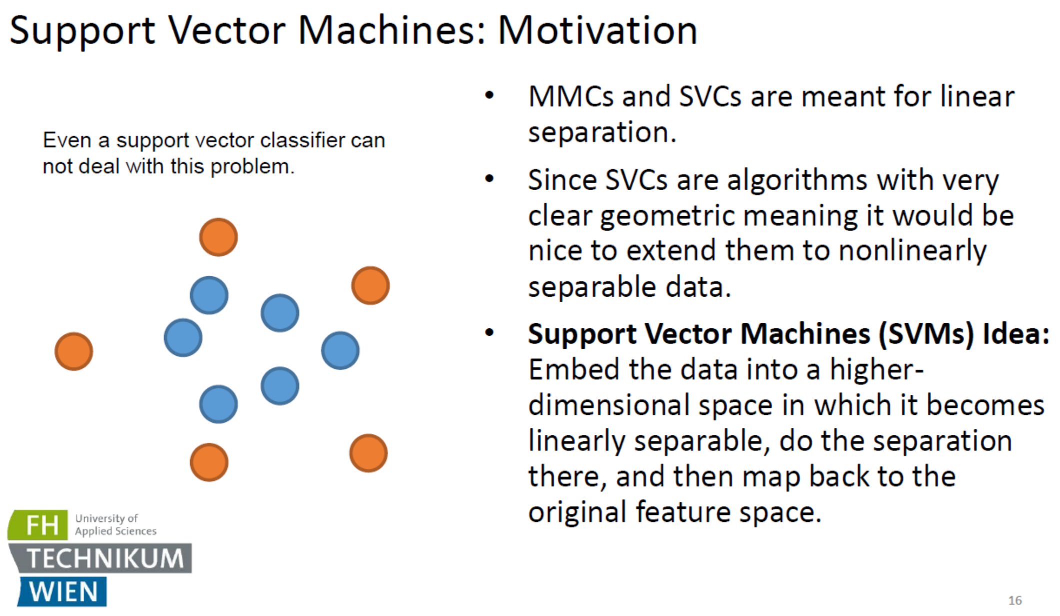
The image shows:

- A **correctly classified** point outside the margin
- A **margin violation**
- A **misclassification**

These cases correspond to different values of ϵ_i.

In the next section, we'll explore how SVMs can handle **non-linearly separable data** using the **kernel trick**.

Next slide, please.



Support Vector Machines: Motivation

Even a support vector classifier can not deal with this problem.

- MMCs and SVCs are meant for linear separation.
- Since SVCs are algorithms with very clear geometric meaning it would be nice to extend them to nonlinearly separable data.
- **Support Vector Machines (SVMs) Idea:** Embed the data into a higher-dimensional space in which it becomes linearly separable, do the separation there, and then map back to the original feature space.

FH University of Applied Sciences TECHNIKUM WIEN

16

We now arrive at the **motivation behind Support Vector Machines (SVMs)**.

0.0.40 The Problem: Non-linear Separation

The figure shows a classic example:

- Blue points are surrounded by orange points.
- **No straight line or soft-margin SVC** can separate these classes effectively.

Even the most flexible **Support Vector Classifier** cannot solve this case in its original feature space.

0.0.41 The Insight Behind SVMs

To handle such cases, **Support Vector Machines** propose a powerful idea:

> **Map the data into a higher-dimensional space** where the classes **become linearly separable**.

Then:

1. Perform linear separation **in that higher space**.
2. Map the result **back to the original space**.

0.0.42 Why Does This Work?

In higher dimensions:

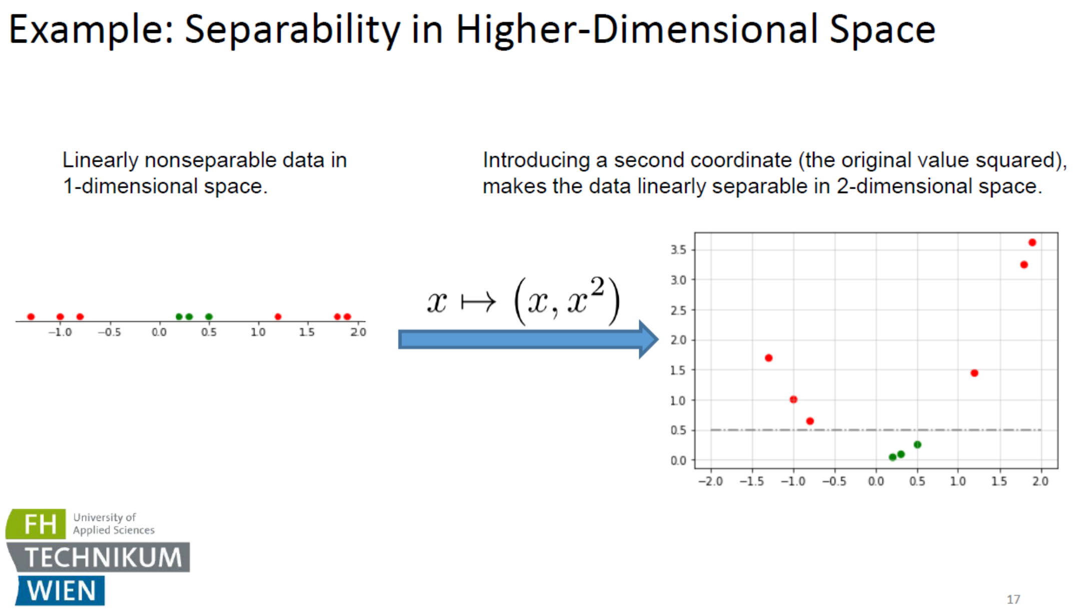
- A complex boundary in 2D can be **transformed into a linear boundary** in 3D or beyond.
- We don't need to manually compute this mapping—the **kernel trick** (explained next) does it efficiently.

0.0.43 Analogy

Imagine drawing a circle around the blue cluster. That's not possible with a line in 2D—but if you lift the center region into a third dimension (like a hill), then a plane can separate the classes.

This marks the transition from SVCs to full **Support Vector Machines**, capable of solving non-linear problems using kernels.

Let's move on to the next slide to understand how **kernels** make this transformation possible without explicitly computing higher-dimensional coordinates.



Example: Separability in Higher-Dimensional Space

Linearly nonseparable data in 1-dimensional space.

Introducing a second coordinate (the original value squared), makes the data linearly separable in 2-dimensional space.

$x \mapsto (x, x^2)$

This slide beautifully illustrates **how data that is not linearly separable in a low-dimensional space can become separable in a higher-dimensional space**.

0.0.44 1D Non-separable Data

On the left:

- We have a **1D feature space**.

- Green points are **surrounded** by red ones.
- It is **impossible** to draw a straight line to separate the classes.

0.0.45 Lifting to 2D via Feature Mapping

To fix this, we apply a **feature transformation**:

$$x \mapsto (x, x^2)$$

This **adds a new dimension** based on the square of the original value.

0.0.46 2D Space: Now Linearly Separable

On the right:

- We plot the transformed points in **2D space**.
- Red points have large x^2, while green points cluster near zero.
- Now, we can draw a **horizontal line** that clearly separates the classes.

0.0.47 The Big Idea

This is the **core of SVMs**:

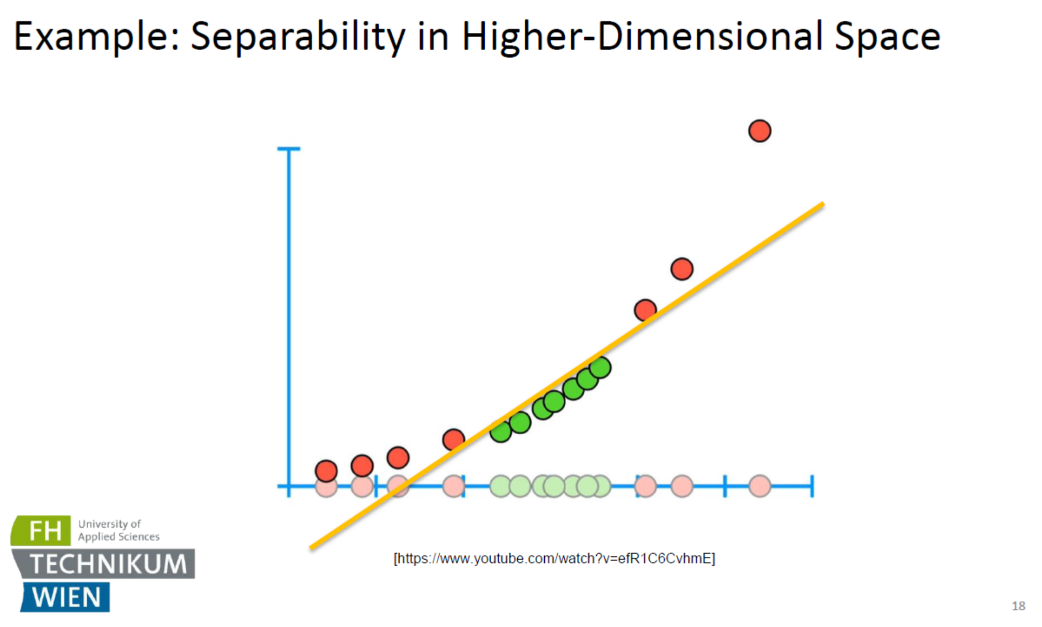
> Transform data to a **higher-dimensional space** where it becomes **linearly separable**, then apply a standard linear classifier.

0.0.48 But Isn't That Computationally Expensive?

Yes—**explicitly computing these transformations** can be very costly. **Fortunately**, SVMs use a **kernel function** to do this **implicitly** and efficiently.

That's exactly what we'll see on the next slide: the **kernel trick**.

Please continue.



Example: Separability in Higher-Dimensional Space

[https://www.youtube.com/watch?v=efR1C6CvhmE]

FH University of Applied Sciences TECHNIKUM WIEN

18

This slide gives another intuitive example of **separability in higher-dimensional space**, visually reinforcing the **power of feature transformation**.

0.0.49 Bottom: The Original Space

- Points (green and red) lie on a **1D line**.
- They are **not linearly separable**—the green class is surrounded by red.
- A **linear classifier would fail** to distinguish them in this space.

0.0.50 Left Axis: Lifting to Higher Dimensions

- Each point is lifted **vertically** to a second axis, based on some transformation—most likely something like $x \mapsto x^2$ or another non-linear function.
- Red points now appear **higher** than green points.

0.0.51 Top Right: Separation Achieved

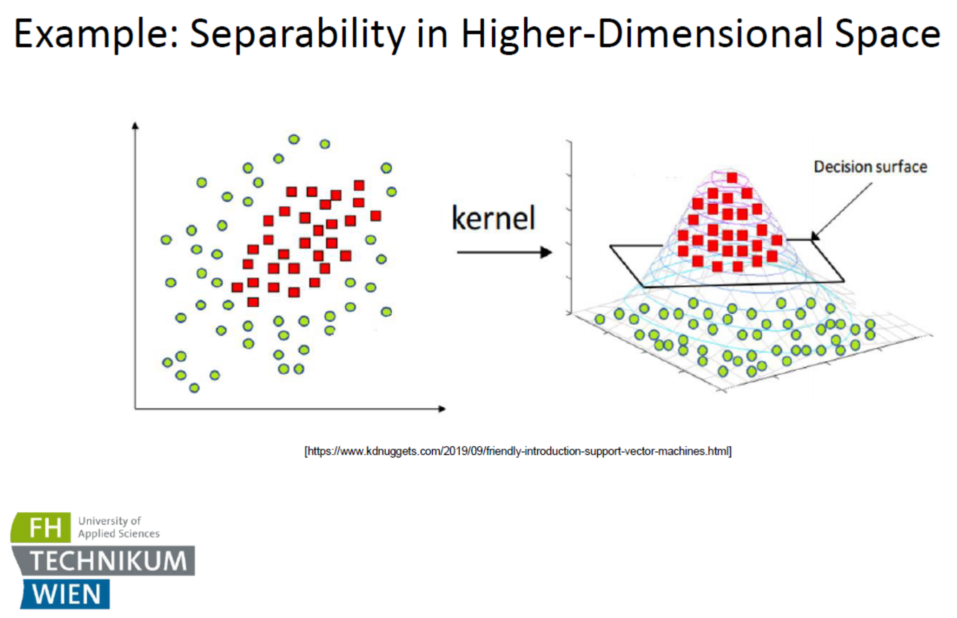
- In this new space, a **straight yellow line** can now perfectly separate the classes.
- This is the essence of what Support Vector Machines do with the **kernel trick**:
  - **Project data into a higher-dimensional space**.
  - **Find a linear separator** there.
  - **Avoid explicit transformation** via kernels.

0.0.52 Key Insight

What was **non-linearly separable** in the original space becomes **linearly separable** in the transformed space.

On the next slide, we'll explain **how kernels let us work in this higher space without explicitly computing the transformation**.

Please continue.



Example: Separability in Higher-Dimensional Space

[https://www.kdnuggets.com/2019/09/friendly-introduction-support-vector-machines.html]

This slide gives us a visual explanation of **what the kernel trick does** in Support Vector Machines (SVMs).

0.0.53 Left: The Problem

In the original 2D feature space:

- We have **green circles** and **red squares**.
- The two classes are **not linearly separable**—no straight line can cleanly separate them.

0.0.54 The Kernel Trick

Instead of manually computing a higher-dimensional representation, we apply a **kernel function** that:

- **Implicitly maps** the original 2D points into a higher-dimensional space,
- Where a **linear separation becomes possible**.

This is shown on the **right**, where:

- Red squares are "lifted" onto a hill,
- Green circles remain on a lower surface,
- And a **flat plane**—the decision surface—can now separate them perfectly.

0.0.55 Why is This Powerful?

- We **don't compute** the high-dimensional coordinates explicitly.
- The **kernel function** computes the **dot product in that higher space**, efficiently.
- This enables SVMs to handle **very complex, non-linear boundaries** with high performance.

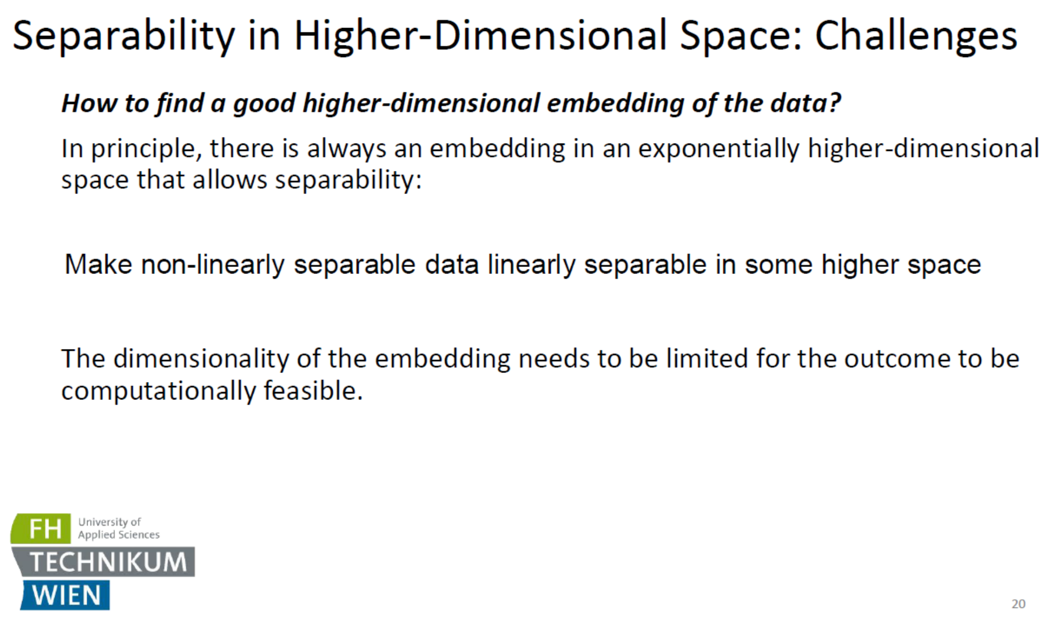
0.0.56 Common Kernels:

- **Linear kernel**: No mapping, just standard SVC
- **Polynomial kernel**: Maps into polynomial feature space
- **RBF (Gaussian) kernel**: Maps into infinite-dimensional space
- **Sigmoid kernel**: Similar to neural networks

Each kernel function enables the model to fit a **different type of non-linearity**.

Next, we'll see an example of a **non-linear decision boundary** produced by using a kernel-based SVM.

Please continue.



Separability in Higher-Dimensional Space: Challenges

How to find a good higher-dimensional embedding of the data?

In principle, there is always an embedding in an exponentially higher-dimensional space that allows separability:

Make non-linearly separable data linearly separable in some higher space

The dimensionality of the embedding needs to be limited for the outcome to be computationally feasible.

FH University of Applied Sciences TECHNIKUM WIEN

20

This slide addresses the **core challenge** in kernel-based SVMs:

> **How to find a good higher-dimensional embedding of the data?**

0.0.57 Theoretical Insight

- In theory, **any non-linearly separable dataset** can be made **linearly separable** by embedding it into a **higher-dimensional space**.
- This is known as **Cover's Theorem**: the probability of linear separability increases with the dimensionality of the feature space.

Hence the strategy:

> **Make non-linearly separable data linearly separable in some higher space**

0.0.58 The Practical Problem

- The **higher the dimension**, the **more computational resources** are needed.
- Embedding data into a very high-dimensional (or infinite-dimensional) space may:
  - Require excessive memory
  - Lead to overfitting
  - Increase computation time
  - Introduce numerical instability

0.0.59 The Kernel Trick: Solution to This Dilemma

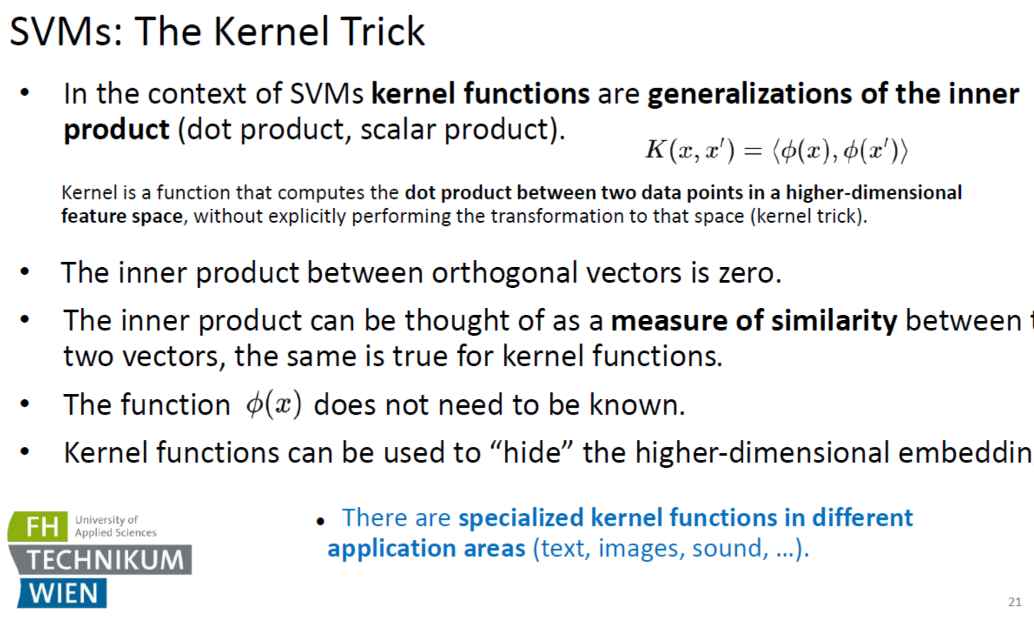
- Instead of explicitly embedding the data into a high-dimensional space,
- We use **kernel functions** to **implicitly compute** the dot product in that space,
- Thus **avoiding the curse of dimensionality**, while retaining the power of non-linear separation.

The challenge is to **select an appropriate kernel** that balances:

- **Expressive power** (ability to separate complex data)
- **Computational feasibility** (efficiency and generalization)

In the final slides, we will likely see **examples of decision boundaries** using different kernels or a summary of what we've covered.

Please go on to the next slide.



SVMs: The Kernel Trick

- In the context of SVMs **kernel functions** are **generalizations of the inner product** (dot product, scalar product). $K(x, x') = \langle \phi(x), \phi(x') \rangle$

  Kernel is a function that computes the **dot product between two data points in a higher-dimensional feature space**, without explicitly performing the transformation to that space (kernel trick).

- The inner product between orthogonal vectors is zero.
- The inner product can be thought of as a **measure of similarity** between the two vectors, the same is true for kernel functions.
- The function $\phi(x)$ does not need to be known.
- Kernel functions can be used to "hide" the higher-dimensional embedding.

FH University of Applied Sciences TECHNIKUM WIEN

- There are **specialized kernel functions in different application areas** (text, images, sound, …).

21

Now we come to the formal definition of the **kernel trick**, which is at the heart of how Support Vector Machines perform non-linear classification.

0.0.60 What Is a Kernel Function?

A **kernel** is a function that **generalizes the dot product** (also called the inner product) to **higher-dimensional feature spaces**.

Mathematically:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

- x, x' are inputs in the original space
- $\phi(x)$ is the mapping to a higher-dimensional space
- The **kernel computes this dot product without needing to know** $\phi(x)$

This is what we mean by the **kernel trick**.

0.0.61 Why Is This Powerful?

- We can **compute similarities** between points **as if they were in a higher-dimensional space**, without explicitly mapping them.
- This saves computation and **avoids memory explosion**.
- It enables **complex, curved decision boundaries** in the original input space.

0.0.62 Conceptual Benefits

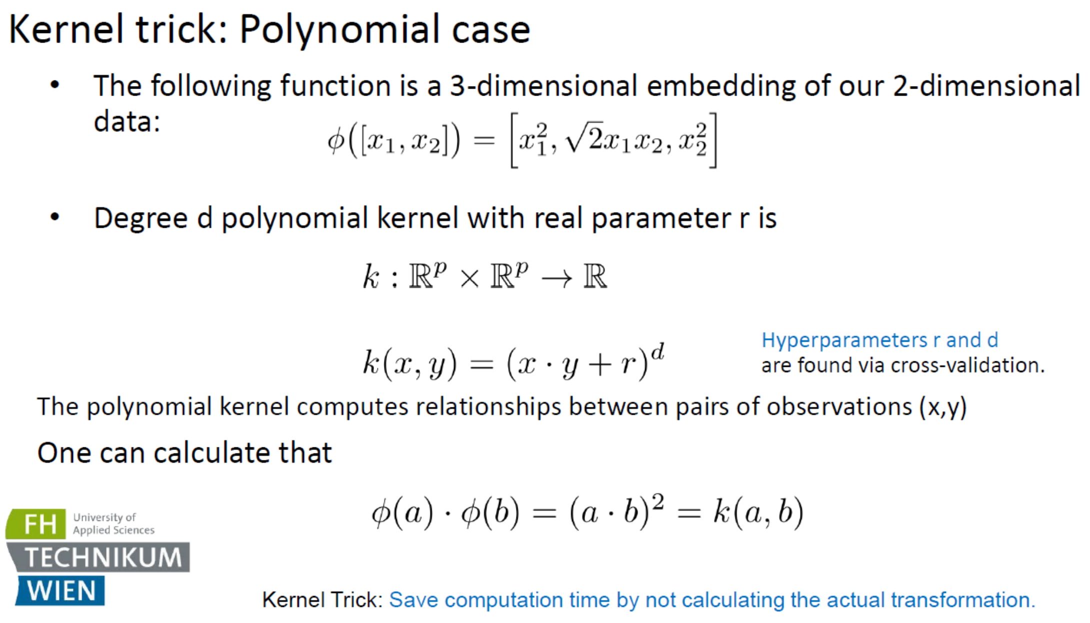
- **Inner products** measure **similarity**—so do **kernel functions**.
- **Orthogonality** (dot product = 0) still implies no correlation.
- The function $\phi(x)$ itself is **not needed or known**—only $K(x, x')$ is used.

0.0.63 Practical Implications

- Kernel functions allow us to **"hide" the complexity** of higher-dimensional embeddings.
- There are **specialized kernels** for different data types:
  - **Text:** string kernels
  - **Images:** histogram intersection kernels
  - **Graphs:** diffusion kernels
  - **Sound/Audio:** wavelet kernels

In the next slides, we'll likely see **concrete examples of kernel functions** and the resulting **decision boundaries** in SVMs.

Please continue.



Kernel trick: Polynomial case

- The following function is a 3-dimensional embedding of our 2-dimensional data: $\phi([x_1, x_2]) = \left[x_1^2, \sqrt{2}x_1x_2, x_2^2\right]$
- Degree d polynomial kernel with real parameter r is

$$k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$$

$$k(x, y) = (x \cdot y + r)^d$$

Hyperparameters r and d are found via cross-validation.

The polynomial kernel computes relationships between pairs of observations (x,y)

One can calculate that

$$\phi(a) \cdot \phi(b) = (a \cdot b)^2 = k(a, b)$$

FH University of Applied Sciences TECHNIKUM WIEN

Kernel Trick: Save computation time by not calculating the actual transformation.

This slide provides a **concrete example** of the **kernel trick**, specifically the **polynomial kernel** case.

0.0.64 Explicit Mapping: 2D to 3D

We start with a 2D input $[x_1, x_2]$ and define a **non-linear mapping** to 3D:

$$\phi([x_1, x_2]) = \left[x_1^2, \sqrt{2}x_1x_2, x_2^2\right]$$

This is a **feature expansion**: it adds interaction and squared terms.

But as always, we don't want to compute this mapping directly.

0.0.65 Polynomial Kernel Definition

Instead, we define a **kernel function**:

$$k(x, y) = (x \cdot y + r)^d$$

Where:

- $x \cdot y$ is the dot product
- r is a **constant (bias)** that makes the kernel more flexible
- d is the **degree of the polynomial**

This kernel implicitly computes the **dot product in a higher-dimensional space**, corresponding to a polynomial expansion.

0.0.66 Why This Works: Kernel Trick in Action

You can verify:

$$\phi(a) \cdot \phi(b) = (a \cdot b)^2 = k(a, b)$$

This tells us:

- Even though we **never explicitly calculate** $\phi(a)$ or $\phi(b)$,
- The **polynomial kernel gives us the same result** as their inner product in that space.

Computation saved Higher-dimensional geometry captured No feature explosion

---

0.0.67 Tuning the Kernel

- The parameters r and d are **hyperparameters**.
- We select them using **cross-validation** to balance **model complexity** and **generalization**.

In the next slide, we'll likely see **how different kernels affect the decision boundary** in practice.

Please continue.

This final slide provides a powerful **visual example of the kernel trick in action** using a **Gaussian kernel** (also known as the RBF kernel).

0.0.68 What You're Seeing

- The data consists of **two classes**: Red $\times$ and Blue $\times$
- The classifier is an **SVM with a Gaussian kernel**.
- The **contour lines** represent values of the **decision function** $y(x)$.
- The **thick black contour** is the **decision boundary**: where the classifier is undecided (i.e., $y(x) = 0$).

0.0.69 Key Elements

- **Support Vectors** are highlighted with **green circles**.
  - These are the critical points that **define the decision boundary**.
- The Gaussian kernel allows the decision function to be **non-linear**, **curved**, and even **disconnected**.
- The margin boundaries (though not explicitly labeled) lie along curves where $y(x) = \pm 1$.

0.0.70 The Power of the Kernel

- In the original input space, this problem is **not linearly separable**.
- The Gaussian kernel maps it into an **infinite-dimensional feature space**.
- Yet the SVM finds a **clear and elegant boundary** that separates the two classes with **maximum margin** in that space.

0.0.71 Why Use a Hard Margin Here?

This example assumes the data is **perfectly separable** in the transformed space—hence, a **hard margin** is used. In practice, we often still use **soft margins** to tolerate noise and improve generalization.

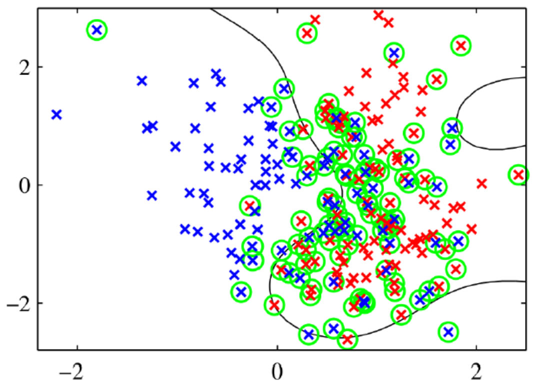
This concludes the conceptual journey of Support Vector Machines:

- From **linear classifiers**,
- To **soft margins**,
- To **kernel methods**,
- And finally to **non-linear classification** in complex spaces.

Would you like a closing slide summary or a full recap in one paragraph?

Kernel Trick & Soft Margin: Example

Illustration of the ν-SVM applied to a nonseparable data set in two dimensions. The support vectors are indicated by circles.



[Bishop (2006): Pattern Recognition and Machine Learning, Figure 7.4]

This final example beautifully illustrates the combined power of the **kernel trick** and the **soft margin** in Support Vector Machines.

0.0.72 What We See

- A **non-separable dataset** in two dimensions:
  - Red × and Blue × mark the two classes.
  - The data is noisy and heavily overlapping.
- A **non-linear SVM** is applied using:
  - A **kernel function** to allow for curved decision boundaries.
  - A **soft margin** to tolerate misclassifications and overlapping points.

0.0.73 Support Vectors and Boundary

- **Green circles** indicate the **support vectors**—points that lie on or inside the margin and influence the decision boundary.
- The **black curves** represent the **decision boundary** and margin borders, shaped by the kernel function in the transformed space.

0.0.74 What Is ν-SVM?

This is a variant of SVM that:

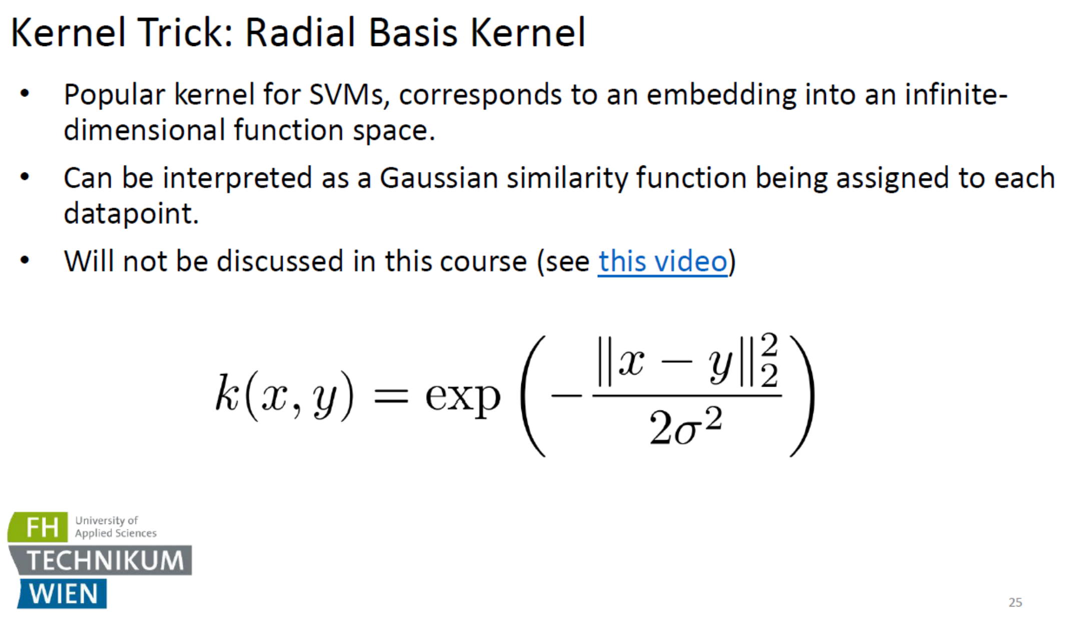
- Controls the number of support vectors and margin violations **via a single parameter** $\nu \in (0, 1]$,
- ν replaces the role of C (soft-margin trade-off) in classical SVMs,
- Often more interpretable and easier to tune in some settings.

0.0.75 Key Takeaways

- Even in **complex, noisy, overlapping data**, kernel SVMs can learn **powerful, flexible boundaries**.
- The **soft margin** prevents overfitting and improves generalization.
- **Only the support vectors matter**—they define the model.

Would you like now:

- A **complete wrap-up summary** of SVMs in one slide or paragraph?
- Or a **short explanation** of how to implement this in Python using scikit-learn?



Kernel Trick: Radial Basis Kernel

- Popular kernel for SVMs, corresponds to an embedding into an infinite-dimensional function space.
- Can be interpreted as a Gaussian similarity function being assigned to each datapoint.
- Will not be discussed in this course (see this video)

$$k(x, y) = \exp\left(-\frac{\|x - y\|_2^2}{2\sigma^2}\right)$$

FH University of Applied Sciences TECHNIKUM WIEN

25

This final slide introduces the **Radial Basis Function (RBF) Kernel**, one of the most powerful and widely used kernels in SVMs.

0.0.76 What is the RBF Kernel?

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

- It measures the **similarity** between two points x and y.

- Based on the **Euclidean distance**: the farther apart the points, the smaller the value.
- Controlled by σ (the kernel width or spread):
  - Small σ: sharp, localized influence
  - Large σ: smoother, more global influence

0.0.77 Why is it so powerful?

- The RBF kernel corresponds to a mapping into an **infinite-dimensional space**.
- Yet, we compute the kernel **without ever knowing the mapping**—thanks to the **kernel trick**.
- It is **extremely flexible** and can model **complex non-linear boundaries**.

0.0.78 Interpretation

- Can be seen as **placing a Gaussian bump** over each training point.
- Class boundaries are shaped by **overlapping these Gaussians**.
- The model becomes a **weighted sum of localized decisions**.

0.0.79 Course Note

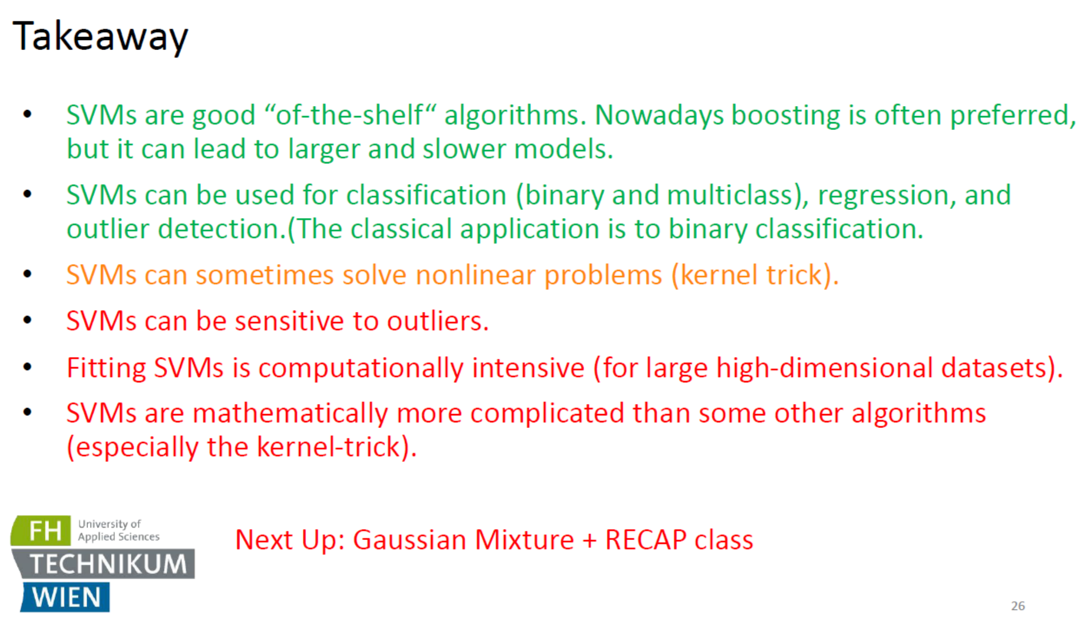
Although this course does **not cover RBF kernels in depth**, the linked video provides further exploration.

Still, this kernel is **very often the best default choice** for SVMs in practice, especially when the data is not linearly separable.

If you like, I can now provide:

- A **recap of the full presentation** in bullet or paragraph format,
- A **Python implementation summary** using `sklearn`,
- Or help preparing **your speaker notes or handout** for this talk.

Just tell me what you need next!



Takeaway

- SVMs are good “of-the-shelf“ algorithms. Nowadays boosting is often preferred, but it can lead to larger and slower models.
- SVMs can be used for classification (binary and multiclass), regression, and outlier detection.(The classical application is to binary classification.
- SVMs can sometimes solve nonlinear problems (kernel trick).
- SVMs can be sensitive to outliers.
- Fitting SVMs is computationally intensive (for large high-dimensional datasets).
- SVMs are mathematically more complicated than some other algorithms (especially the kernel-trick).

FH University of Applied Sciences TECHNIKUM WIEN

Next Up: Gaussian Mixture + RECAP class

26

Here is a clear and concise **wrap-up of the full Support Vector Machines (SVMs) presentation**, based on your final takeaway slide:

0.1 SVMs – Key Takeaways

Strengths

- SVMs are reliable, off-the-shelf models especially for **medium-sized structured data**.
- Useful for **classification** (binary and multiclass), **regression**, and **outlier detection**.
- Through the **kernel trick**, they can solve **non-linear problems** by implicitly working in higher-dimensional spaces.

Limitations

- **Sensitive to outliers**, especially in hard-margin settings.
- **Computationally expensive** on large or high-dimensional datasets.
- Require careful **hyperparameter tuning** (e.g., kernel type, C, σ, ν).
- **Mathematically intensive**, especially with non-linear kernels.

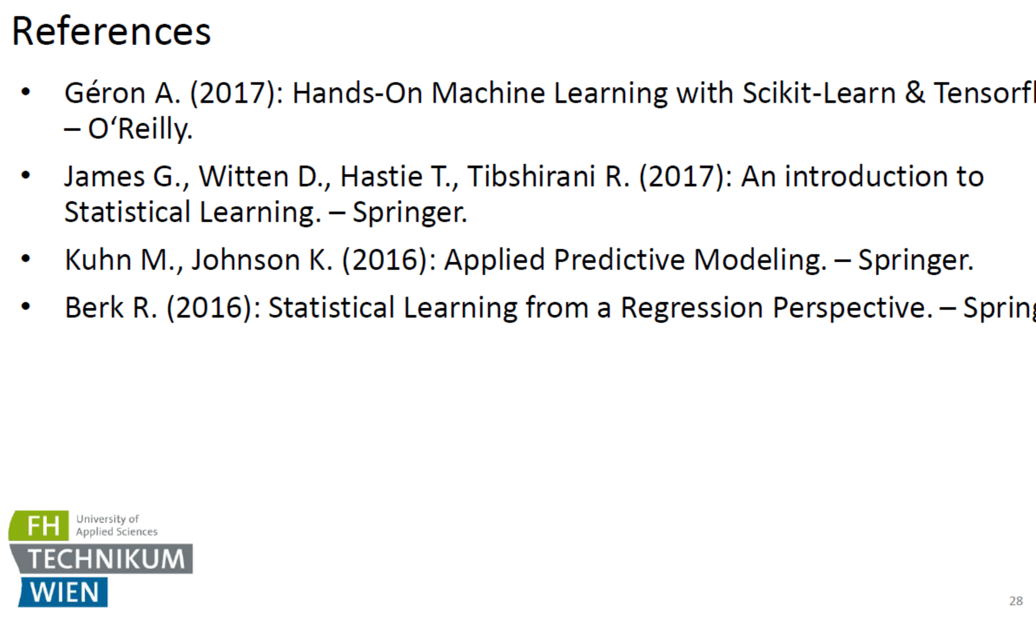
When to use SVMs

- When you need **strong performance** on a **well-prepared, non-huge dataset**.
- When you want a **clear geometric interpretation**.
- When boosting or deep learning might be overkill.

What's next?

You'll now continue with **Gaussian Mixture Models (GMMs)** and a **recap class**, transitioning from large-margin classifiers to **generative probabilistic models**.

Would you like me to generate speaker notes, a summary handout, or a Python demo notebook to support this topic?



References

- Géron A. (2017): Hands-On Machine Learning with Scikit-Learn & Tensorflow. – O‘Reilly.
- James G., Witten D., Hastie T., Tibshirani R. (2017): An introduction to Statistical Learning. – Springer.
- Kuhn M., Johnson K. (2016): Applied Predictive Modeling. – Springer.
- Berk R. (2016): Statistical Learning from a Regression Perspective. – Springer.