

Support Vector Machines

Master ESA - University of Orléans

Christophe HURLIN

University of Orléans and IUF

Master Econométrie et Statistique Appliquée

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

Definition: Support Vector Machines

Definition: Support Vector Machines

Support Vector Machines (**SVM**), introduced by Vapnik (1995, 1998), are a set of supervised learning techniques designed to solve classification or regression problems.

References


Vapnik V.N. (1995). *The Nature of Statistical Learning Theory*. Springer.


Vapnik V.N. (1998). *Statistical Learning Theory*. John Wiley.

Maximum Margin Separators Hyperplan

Remark: SVMs are also called Maximum Margin Separators Hyperplan (MMSH).



Boser B.E, Guyon I.M., Vapnik V.N. (1992). A Training Algorithm for Optimal Margin Classifiers, *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.

SVM and SVR

Support Vector Methods

- For **classification** problems, the method is called SVM (**Support Vector Machine**).
- For **regression** problems, the method is called SVR (**Support Vector Regression**).
- In the following, we first present the classification case, and then extend the ideas to regression.

SVM Intuition

Support Vector Machines rely on two fundamental concepts:

1. The principle of the **maximum margin**, which seeks the separating hyperplane that maximizes the distance to the closest training points.
2. The use of a **kernel function**, which makes it possible to extend the method to non-linear decision boundaries by mapping the data into higher-dimensional spaces.

Maximum Margin

SVM Intuition: Maximum Margin

- The margin is the distance between the separating hyperplane and the closest training points, called support vectors.
- The optimal hyperplane is the one that maximizes this margin.
- The task is therefore to determine the separating hyperplane that achieves the largest possible margin, based on the training data.
- This leads to a quadratic optimization problem with linear constraints.

Kernel Trick

SVM Intuition: Kernel Trick

- When the data are not linearly separable, the idea is to map the inputs into a higher-dimensional feature space, where a linear separation may exist.
- The kernel trick allows this mapping to be performed implicitly: instead of computing the transformation explicitly, one directly evaluates a kernel function that measures similarity between data points.
- This makes it possible to replace the scalar product in the transformed space (potentially very expensive) by the evaluation of a simple kernel function in the original space.

Bibliography: Main References

References:


Cristianini, N., and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press.


Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer.


Suykens, J., and Vandewalle, J. (1999). Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3), 293–300.


Suykens, J., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J. (2002). *Least Squares Support Vector Machine*. World Scientific.


Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.


Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley.

Bibliography: Other Course Notes

Other Course Notes (Non-Exhaustive List)


Bousquet, O. *Introduction to Support Vector Machines*, CMA, École Polytechnique.


Clémençon, S. *Introduction to Learning Theory*, Télécom ParisTech.


Rakotomalala, R. *Support Vector Machine*, Université Lumière Lyon 2.


Rakotomamonjy, A. *Linear Large Margin Separators*, INSA Rouen.


Revel, A. *Support Vector Machines*, Université de La Rochelle.


Zhao, A. *Support Vector Machines*, Metis.

Bibliography: Applications in Risk Management

Selection of SVM/SVR Applications in Risk Management


Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., Vanthienen, J. (2003). Benchmarking State-of-the-Art Classification Algorithms for Credit Scoring. *Journal of the Operational Research Society*, 54, 627–635.


Loterman, G., Brown, I., Martens, D., Mues, C., and Baesens, B. (2012). Benchmarking Regression Algorithms for Loss Given Default Modeling. *International Journal of Forecasting*, 28(1), 161–170.


Tobback, E., Martens, D., Van Gestel, T., and Baesens, B. (2014). Forecasting Loss Given Default Models: Impact of Account Characteristics and the Macroeconomic State. *Journal of the Operational Research Society*, 65(3), 376–392.


Yao, X., Crook, J., and Andreeva, G. (2015). Support Vector Regression for Loss Given Default Modelling. *European Journal of Operational Research*, 240(2), 528–538.


Yao, X., Crook, J., and Andreeva, G. (2017). Enhancing Two-Stage Modelling Methodology for Loss Given Default with Support Vector Machines. *European Journal of Operational Research*, 263(2), 679–689.

Objectives of the Session

1. Understand the fundamental concepts of linear separability and margin maximization.
2. Derive the primal and dual formulations of the Support Vector Machine optimization problem.
3. Identify and interpret support vectors and their role in classification.
4. Apply the canonical hyperplane representation and compute optimal separating boundaries.
5. Implement hard margin and soft margin SVMs for linearly separable and non-separable data.
6. Understand the kernel trick.
7. Apply SVMs to real-world classification and regression problems in various domains.



Credit: iStock

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

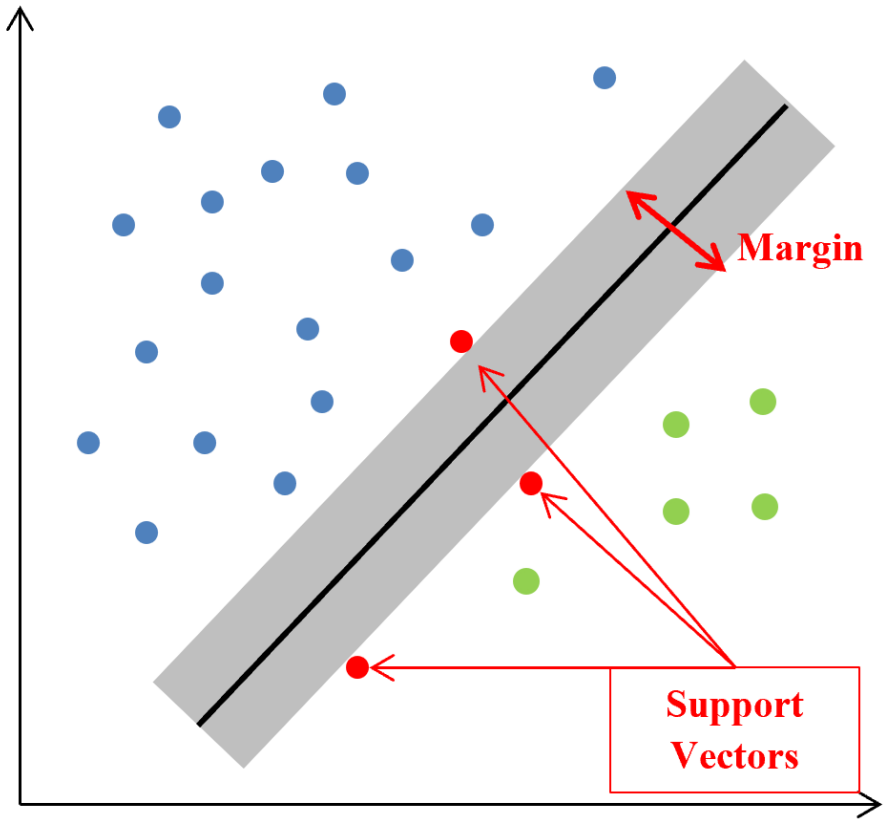
4. Soft Margin

5. Kernel Trick

6. SVM Variants

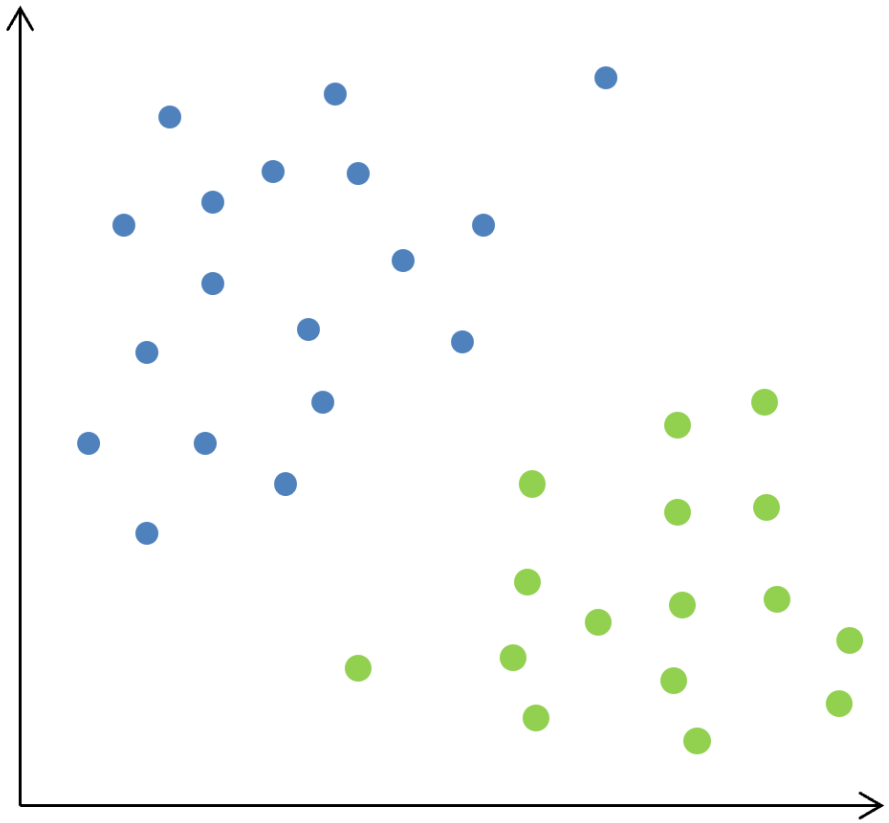
7. Appendix

SVM Intuition



- In SVM, the goal is to build a linear classifier able to separate the observations into different classes.
- To achieve this, we look for the optimal separating boundary, defined as the one that maximizes the margin.
- The margin is the distance between the separating boundary (hyperplane) and the closest observations. These closest observations are called support vectors.

SVM Intuition



- The problem consists in finding this optimal separating boundary from a training sample.
- The solution is to formulate the task as a quadratic optimization problem.
- A sample is said to be linearly separable if there exists a linear classifier that correctly classifies all the observations.

Notations

Notations

- Let (x, y) be a pair of random variables taking values in $\mathcal{X} \times \mathcal{Y}$.
- We first focus on the classification case with $\mathcal{Y} = \{-1, 1\}$.
- This can be extended to the case $card(\mathcal{Y}) = m > 2$, or to $\mathcal{Y} = \mathbb{R}$ (regression case).
- The variable X is a vector; with real-valued predictors we have $\mathcal{X} = \mathbb{R}^d$.
- The space $\mathcal{X}$ is equipped with an inner product.

Classifier and Decision Function

Definition: Classifier and Decision Function

We seek a **classifier** $\widehat{y} = g(x)$ such that $g : \mathcal{X} \to \{-1, 1\}$ minimizes the probability of a classification error:

$$\Pr\left(g(x) \neq y\right)$$

Instead of constructing g directly, one generally builds a **decision function** $h : \mathcal{X} \to \mathbb{R}$ and associates the classifier

$$g(x) = \operatorname{sgn}\big(h(x)\big).$$

Linear Classifier

Definition: Linear Classifier

A **linear classifier** (or perceptron) is a function of the form

$$g(x) = \text{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0, \end{cases}$$

where the decision function is linear, with

$$h(x) = \langle \omega, x \rangle + b,$$

of equivalently

$$h(x) = x'\omega + b = \sum_{j=1}^{d} \omega_j x_j + b,$$

where $\omega \in \mathbb{R}^d$, $b \in \mathbb{R}$, and $\langle \cdot, \cdot \rangle$ denotes the inner product.

Example of Linear Classifier in $\mathbb{R}^2$

Numerical Example: Linear Classifier in $\mathbb{R}^2$

Consider a training sample $\{(y_i, x_i)\}_{i=1}^n$ with two features ($\mathcal{X} = \mathbb{R}^2$).

We consider a **linear classifier** with $\omega = (1, 2)$ and $b = -1$:

$$g(x) = \text{sgn}\big(h(x)\big), \qquad \forall x = (x_1, x_2) \in \mathbb{R}^2$$

where

$$h(x) = \langle \omega, x \rangle + b = x_1 + 2x_2 - 1.$$

Each observation can then be classified. For instance:

$$\text{Example A: } (y, x_1, x_2) = (1, 1, -1) \quad \Rightarrow \quad h(x) = -2,\ g(x) = -1$$

$$\text{Example B: } (y, x_1, x_2) = (1, 2, 1) \quad \Rightarrow \quad h(x) = 3,\ g(x) = 1$$

Example of Linear Classifier in $\mathbb{R}^3$

Numerical Example: Linear Classifier in $\mathbb{R}^3$

Consider a training sample $\{(y_i, x_i)\}_{i=1}^n$ with two features ($\mathcal{X} = \mathbb{R}^2$).

We use a linear classifier with $\omega = (3, 2, 7)$ and $b = -2$:

$$g(x) = \text{sgn}\big(h(x)\big), \qquad \forall x = (x_1, x_2, x_3) \in \mathbb{R}^3$$

where

$$h(x) = \langle \omega, x \rangle + b = 3x_1 + 2x_2 + 7x_3 - 2.$$

Each observation can then be classified. For example:

$$\text{Example A: } (y, x_1, x_2, x_3) = (1, 0, 2, -4) \quad \Rightarrow \quad h(x) = -26, \ g(x) = -1$$

$$\text{Example B: } (y, x_1, x_2, x_3) = (1, 8, 1, 2) \quad \Rightarrow \quad h(x) = 38, \ g(x) = 1$$

Remarks

Remark: Because of the properties of the $\text{sgn}(\cdot)$ function, strictly speaking, the points lying exactly on the separating hyperplane are not classified into $\{-1, 1\}$:

$$g(x) = \text{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) > 0, \\ 0 & \text{if } h(x) = 0 \Leftrightarrow x \in H, \\ -1 & \text{if } h(x) < 0, \end{cases}$$

but in practice, they are usually assigned to one of the two half-spaces:

$$g(x) = \text{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0. \end{cases}$$

Reminder: for a continuous variable, the probability of being exactly at a single point is zero.

Separating Hyperplane

Definition: Separating Hyperplane

A **separating hyperplane** $\mathcal{H}$ divides the input space $\mathcal{X}$ into two half-spaces corresponding to the two classes of y. It is defined by the equation:

$$h(x) = 0.$$

Numerical Example of Separating Hyperplane in $\mathbb{R}^2$

Numerical Example: Separating Hyperplane in $\mathbb{R}^2$

Consider the case $\mathcal{X} = \mathbb{R}^2$ and a linear classifier $g(x)$ defined by $\omega = (1, 2)$ and $b = -1$:

$$h(x) = \langle \omega, x \rangle + b = x_1 + 2x_2 - 1.$$

$$g(x) = \text{sgn}(h(x)) = \begin{cases} 1 & \text{if } x_1 + 2x_2 - 1 \geq 0, \\ -1 & \text{if } x_1 + 2x_2 - 1 < 0. \end{cases}$$

Examples:

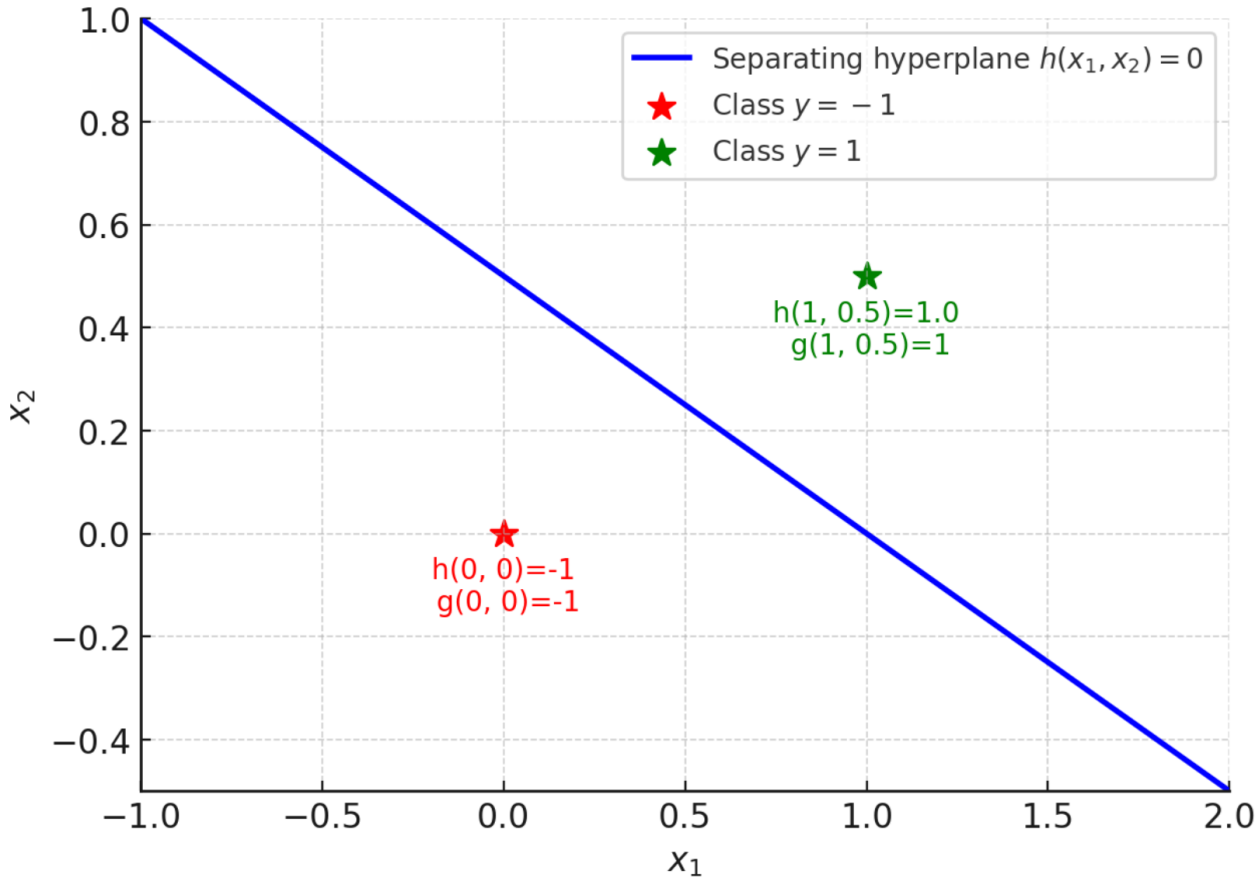
$$h(0, 0) = -1 \quad \Rightarrow \quad g(0, 0) = -1$$

$$h\left(1, \tfrac{1}{2}\right) = 1 \quad \Rightarrow \quad g\left(1, \tfrac{1}{2}\right) = 1$$

The affine hyperplane (here a line) separating $\mathbb{R}^2$ into two half-spaces corresponding to the two classifications is given by:

$$h(x) = x_1 + 2x_2 - 1 = 0, \quad \forall x = (x_1, x_2) \in \mathbb{R}^2.$$

Numerical Example of Separating Hyperplane in $\mathbb{R}^2$



Numerical Example of Separating Hyperplane in $\mathbb{R}^3$

Numerical Example: Separating Hyperplane in $\mathbb{R}^3$

Consider a linear classifier $g(x)$ defined by $\omega = (1, 2, -3)$ and $b = -2$:

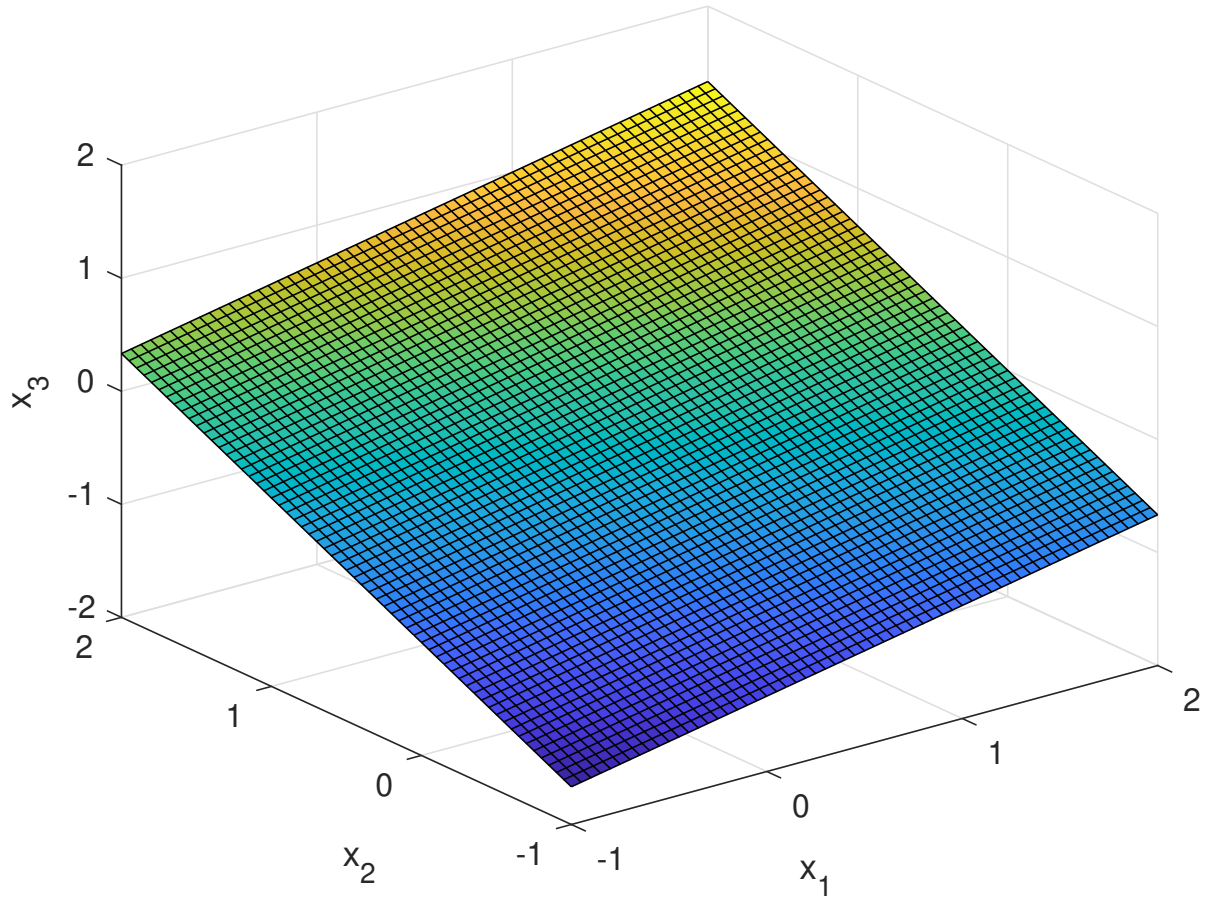
$$h(x) = \langle \omega, x \rangle + b = x_1 + 2x_2 - 3x_3 - 2.$$

The equation

$$h(x) = x_1 + 2x_2 - 3x_3 - 2 = 0, \qquad \forall x = (x_1, x_2, x_3) \in \mathbb{R}^3$$

defines a hyperplane that separates $\mathbb{R}^3$ into two half-spaces, each corresponding to a classification.

Numerical Example of Separating Hyperplane in $\mathbb{R}^3$



Confusion Matrix

Confusion Matrix

- Classifier $g(x) = \text{sgn}\big(h(x)\big)$
- If there is no error, then $g(x)y = 1$; otherwise $g(x)y = -1$

	$g(x) = -1$	$g(x) = 1$
$y = -1$	No error $g(x)y = 1$	Error $g(x)y = -1$
$y = 1$	Error $g(x)y = -1$	No error $g(x)y = 1$

Remark: the probability of a classification error is equal to

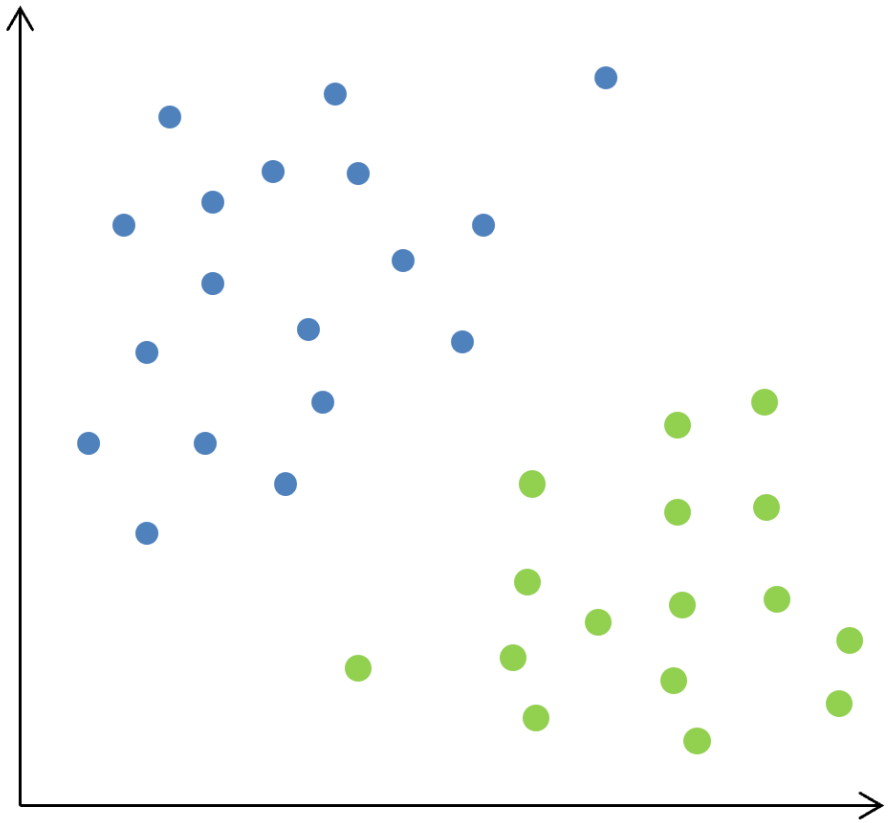
$$\Pr\big(g(x) \neq y\big) = \Pr\big(h(x)\, y \leq 0\big) = \Pr\big(g(x)\, y = -1\big).$$

Linearly Separable Sample

Definition: Linearly Separable Sample

A sample $S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (\mathcal{X} \times \mathcal{Y})$ is said to be **linearly separable** if there exists a linear classifier that correctly classifies all the observations in S.

SVM Intuition

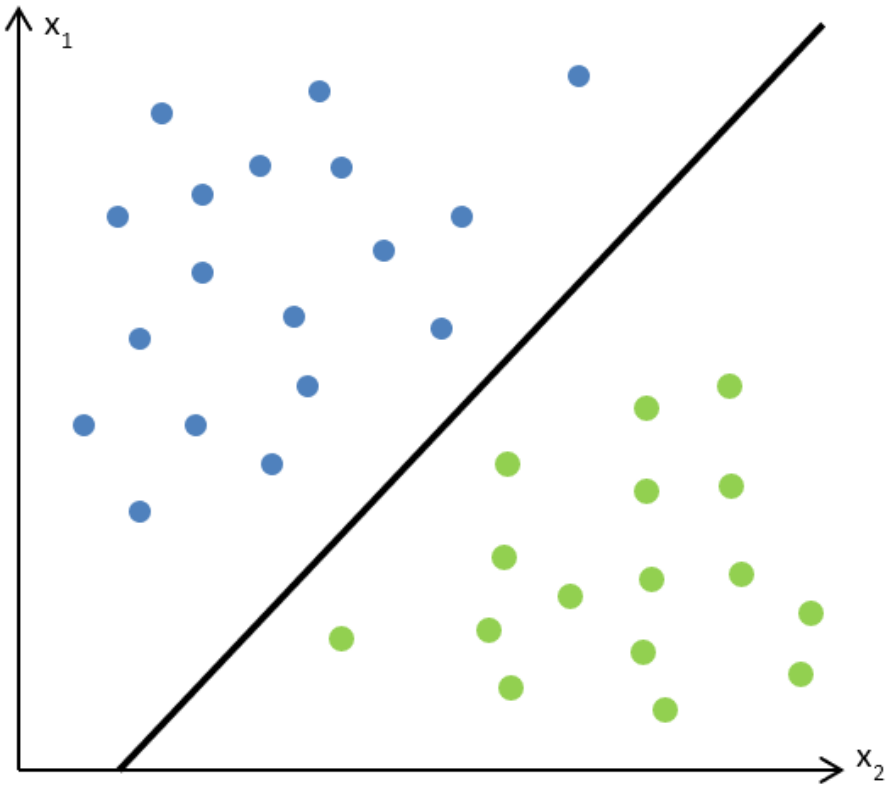


Consider $\mathcal{X} = \mathbb{R}^2$ with $x_i = (x_{i1}, x_{i2})$ and a sample $\{(x_i, y_i)\}$ for $i = 1, \ldots, n$.

Each triplet (x_{i1}, x_{i2}, y_i) is represented as a point in the plane (x_1, x_2).

When $y_i = 1$ (-1), the observation is represented by a blue (green) point.

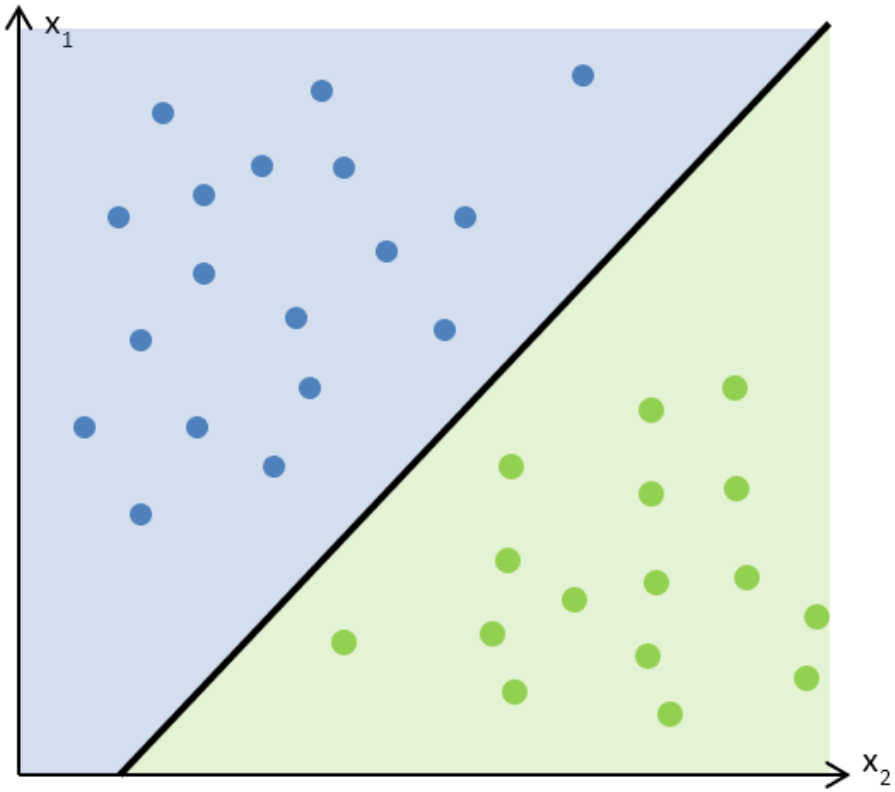
SVM Intuition



Let a linear classifier $g(x)$ be associated with an affine hyperplane:

$$h(x) = \langle \omega, x \rangle + b = 0.$$

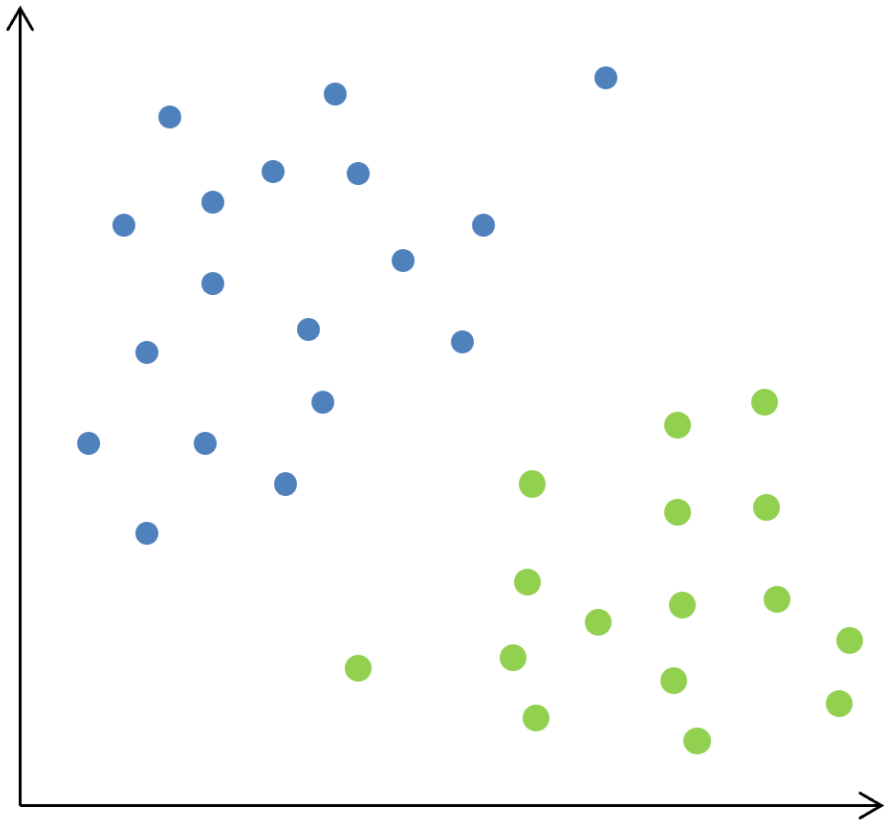
SVM Intuition



If all the observations with $y_i = 1$ lie in the blue region and all the observations with $y_i = -1$ lie in the green region, then there is no classification error.

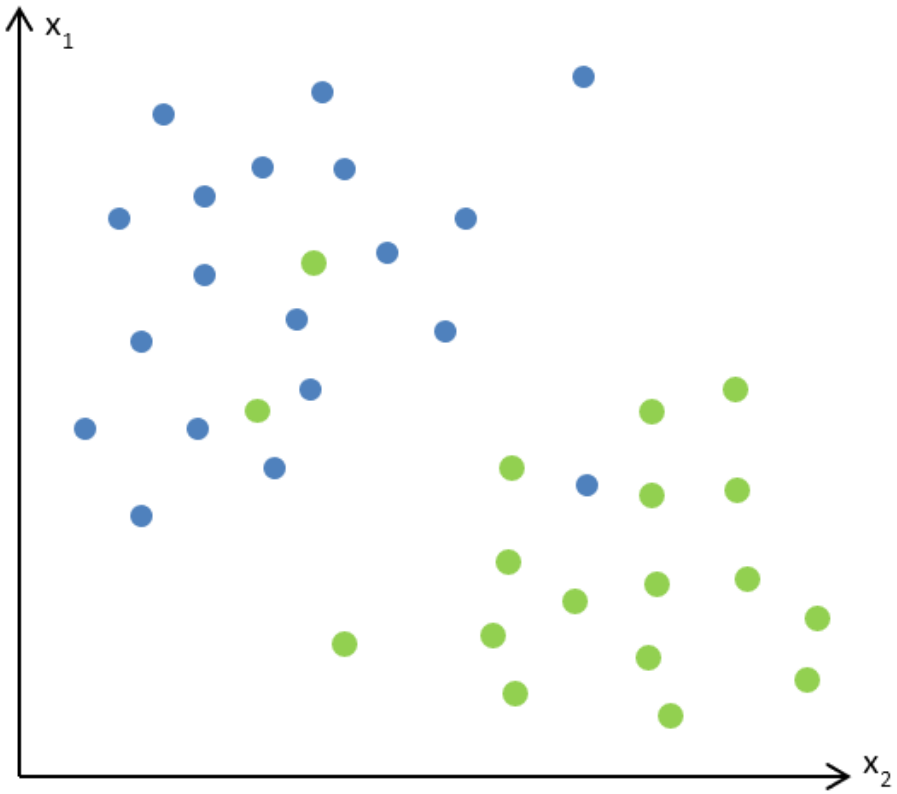
Therefore, a linear classifier exists that correctly classifies all the observations in the sample.

SVM Intuition



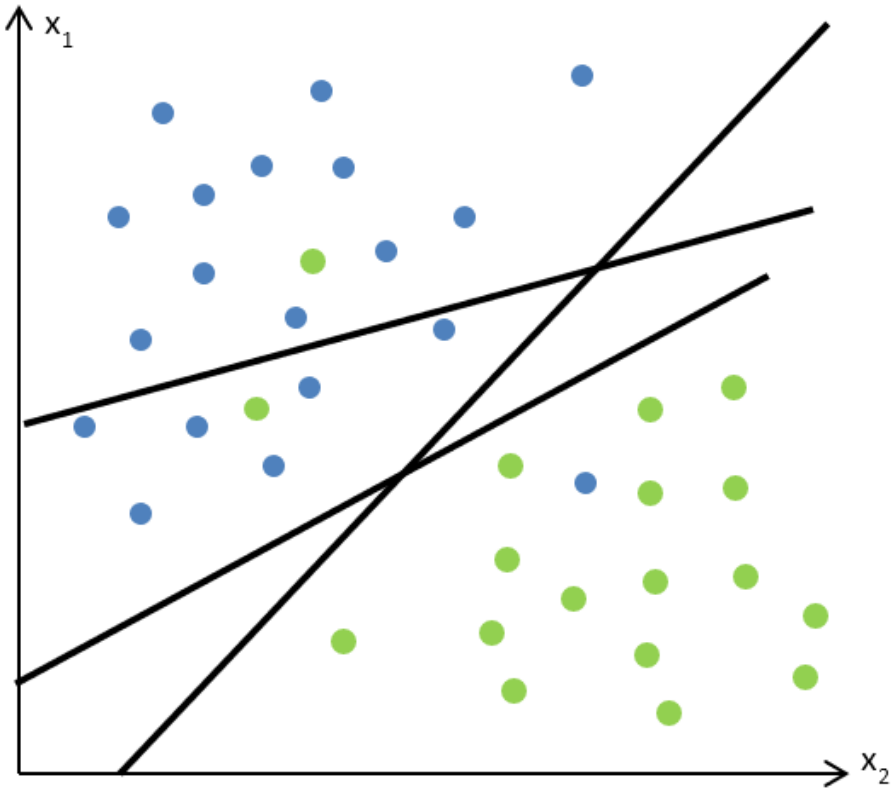
The sample is said to be linearly separable.

SVM Intuition



In contrast, this sample is not linearly separable.

SVM Intuition



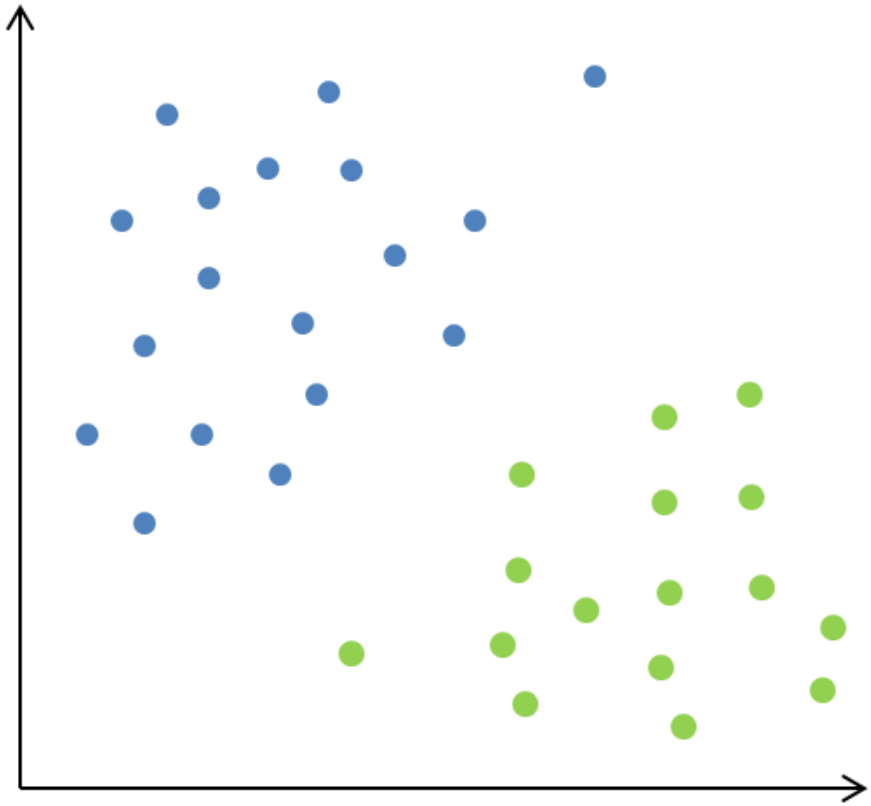
Indeed, there does not exist any linear classifier that can correctly classify all the observations.

SVM Intuition

Remark

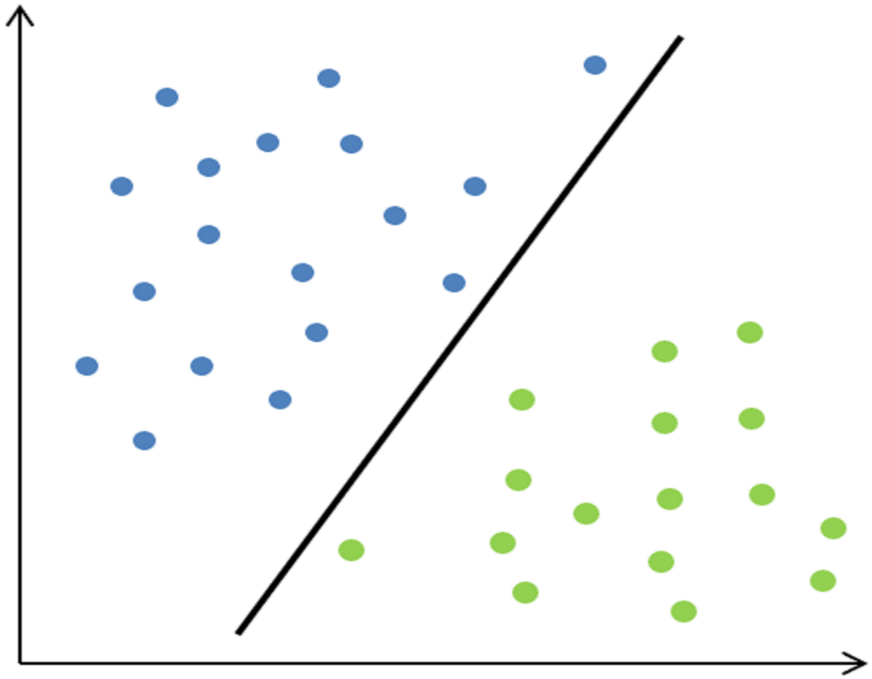
For a linearly separable sample, there may exist several linear classifiers (i.e., several pairs (ω, b)) that achieve the same learning performance (no classification error). Which one is then the optimal classifier?

SVM Intuition



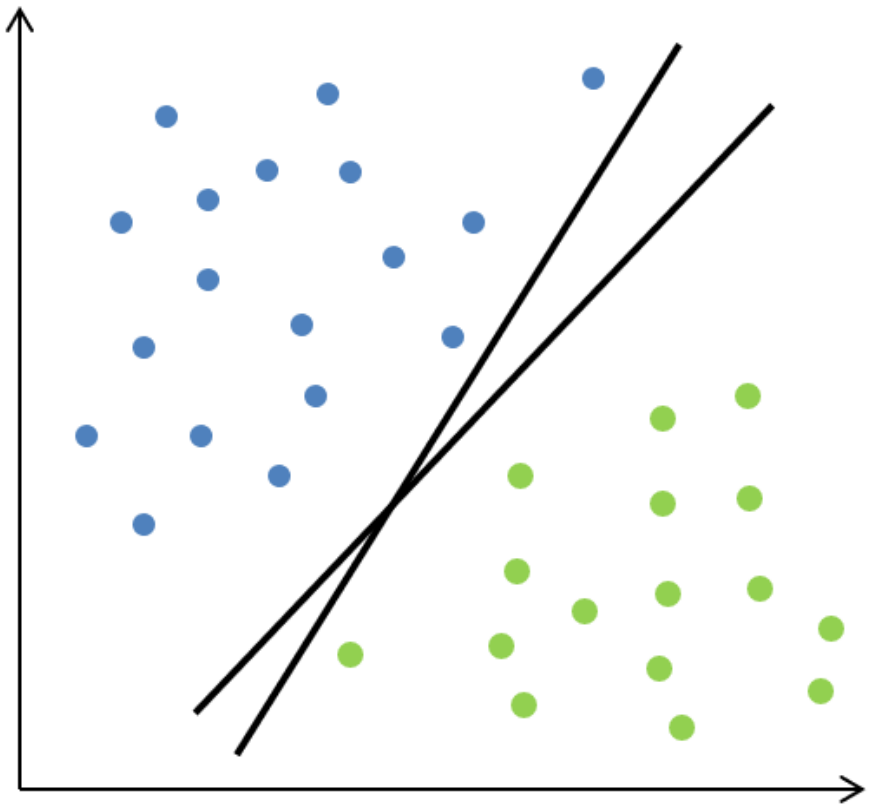
What is the **optimal** linear classifier for this sample?

SVM Intuition



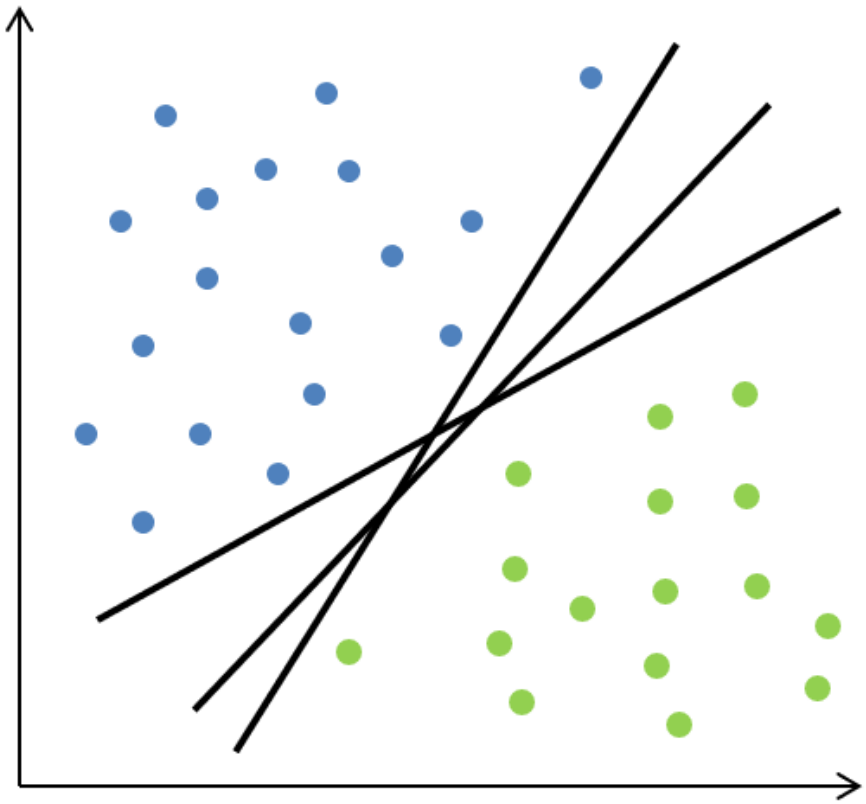
Is it this one?

SVM Intuition



Is it this one?

SVM Intuition



Is it this one?

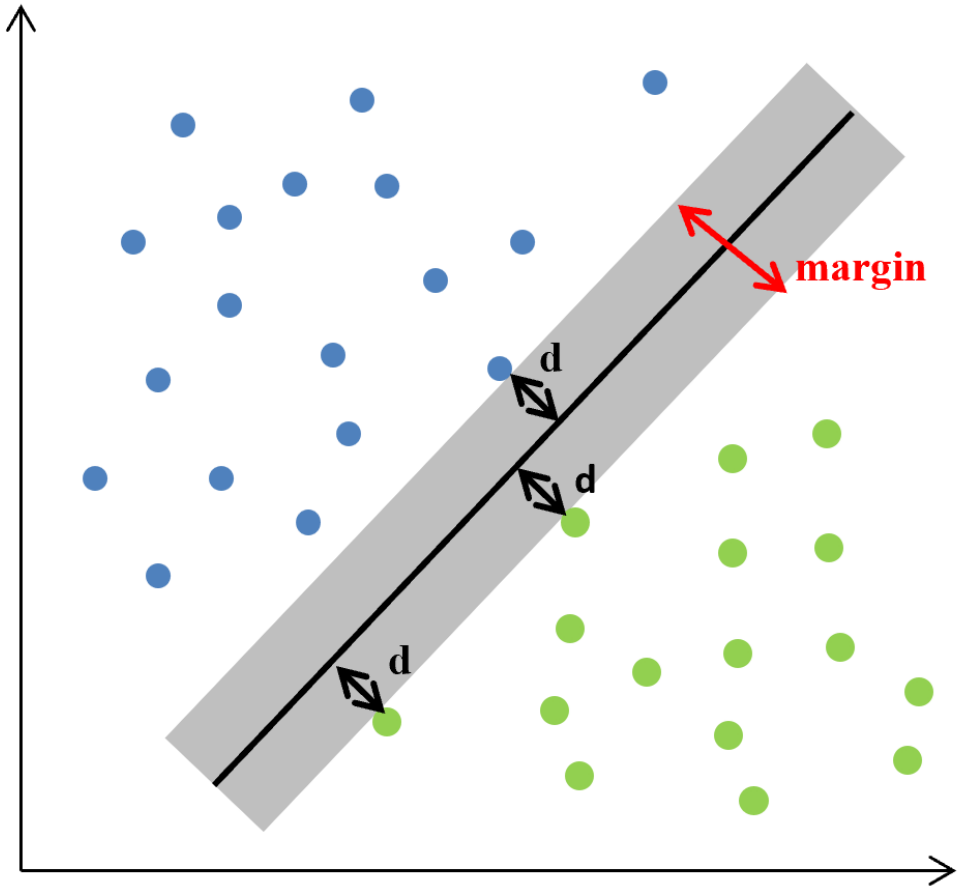
Margin: Definition

Definition: Margin

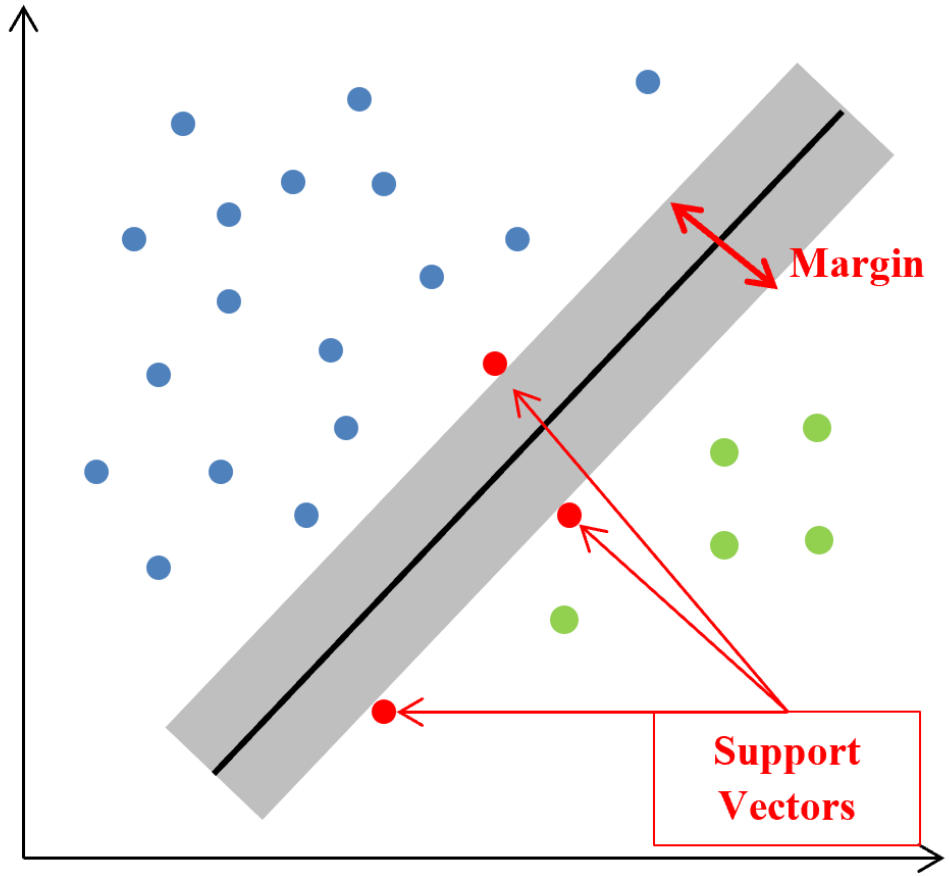
The **margin** M is defined as twice the distance d from the closest point to the hyperplane.

Idea of SVM: choose the hyperplane that correctly classifies the data and is as far as possible from all the observations (examples). This is the criterion of optimal margins

SVM Intuition



SVM Intuition

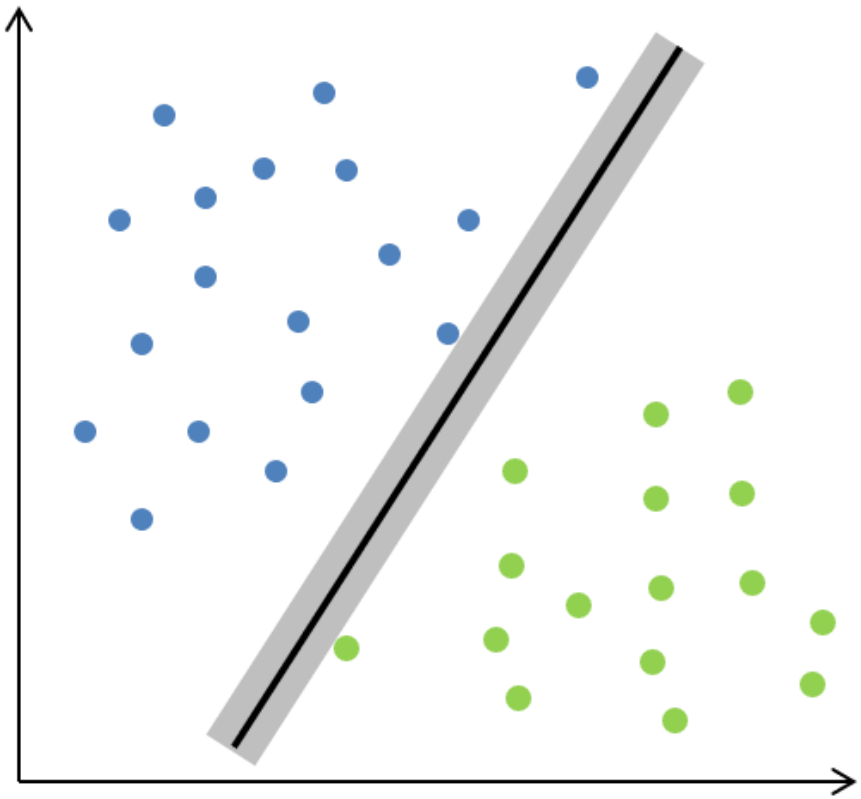


Support Vectors: Definition

Definition: Support Vectors

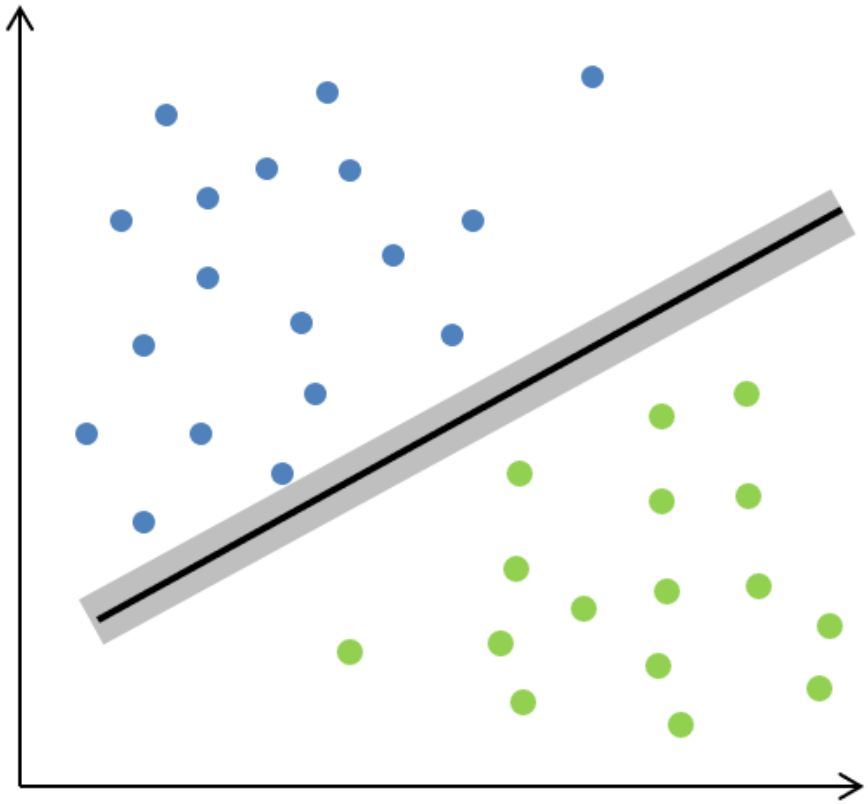
Support vectors are the observations located on the margin.

SVM Intuition



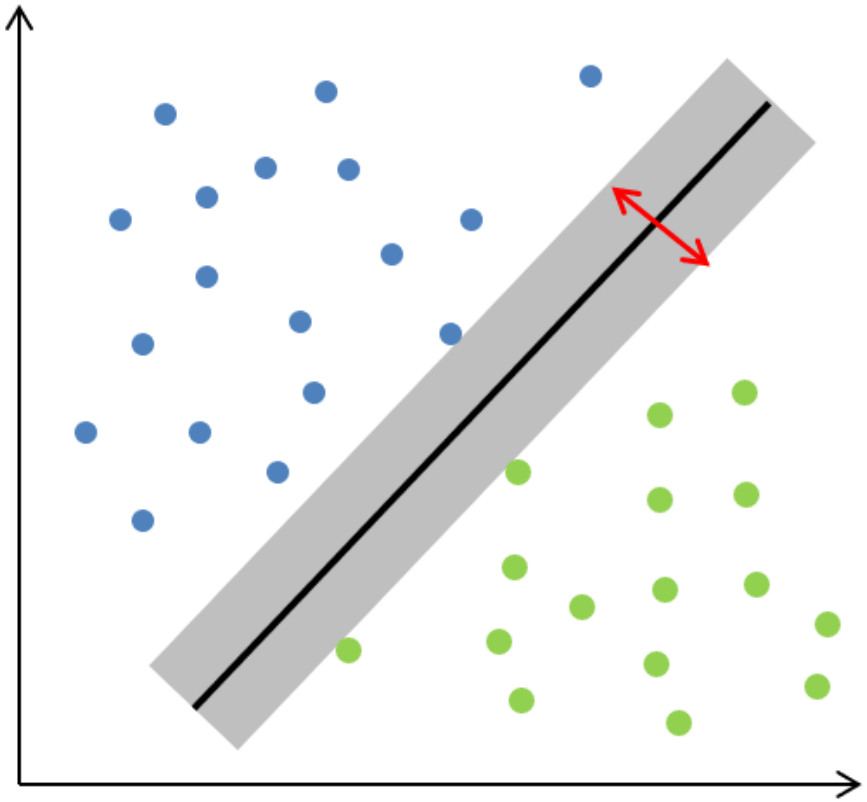
Returning to our example: what is the **optimal linear classifier**?

SVM Intuition



Returning to our example: what is the **optimal linear classifier**?

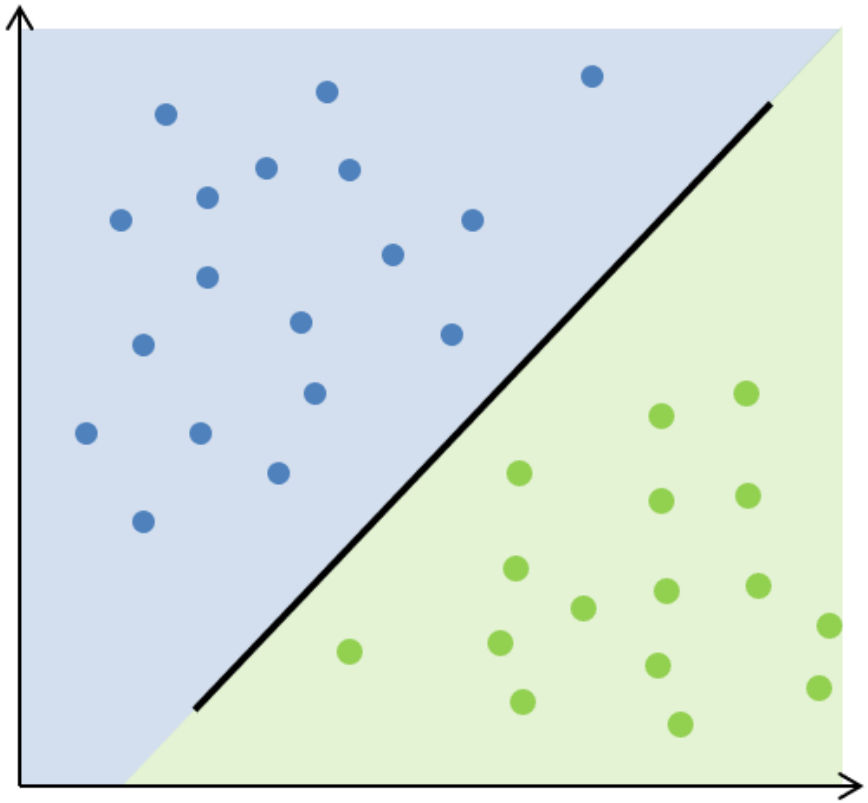
SVM Intuition



This is the classifier with the optimal margin, also called the **SVM**.

Support Vector Margin

SVM Intuition



Here is the classification associated with the hyperplane of optimal margin, i.e., the SVM.

Support Vector Machine: Definition

Definition: Support Vector Machine

The **Support Vector Machine (SVM)** is the linear classifier (ω^*, b^*) that perfectly classifies all the observations in the training sample and is associated with the largest margin.

SVM Intuition

Key Concepts

1. Linear classifier and decision function.
2. Separating hyperplane.
3. Linearly separable sample.
4. Margin.
5. Support vectors.
6. Support Vector Machine.

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

Reminders: Inner Product

Reminders

Let two vectors in $\mathbb{R}^d$: $r = (r_1, \ldots, r_d)'$ and $s = (s_1, \ldots, s_d)'$.

- **Inner product**

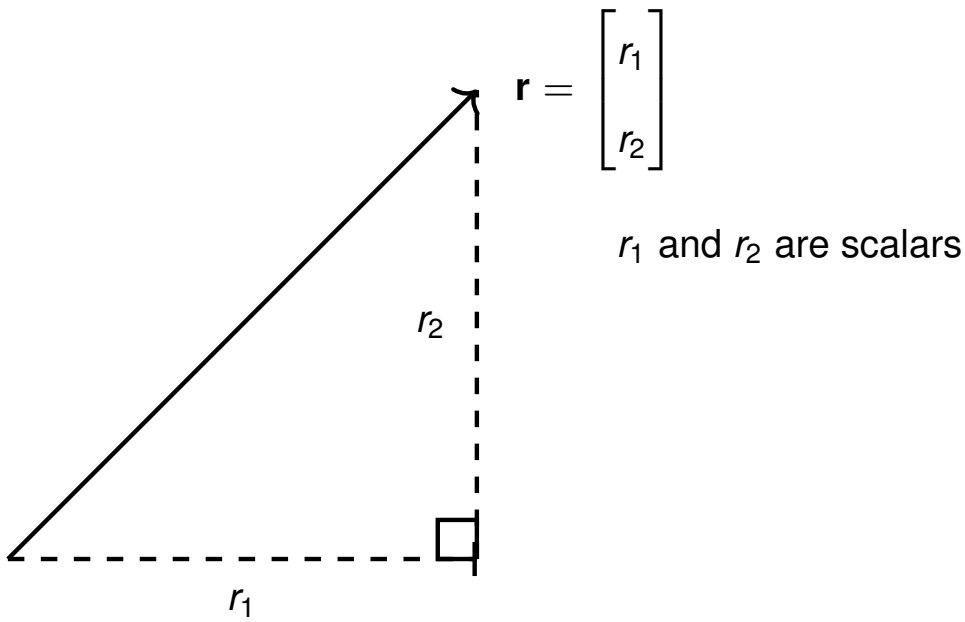
$$\forall (r, s) \in \mathbb{R}^d \times \mathbb{R}^d, \quad \langle r, s \rangle = \sum_{i=1}^{d} r_i s_i$$

- **Euclidean norm**

$$\forall r \in \mathbb{R}^d, \quad \|r\| = \sqrt{\langle r, r \rangle} = \sqrt{\sum_{i=1}^{d} r_i^2}$$

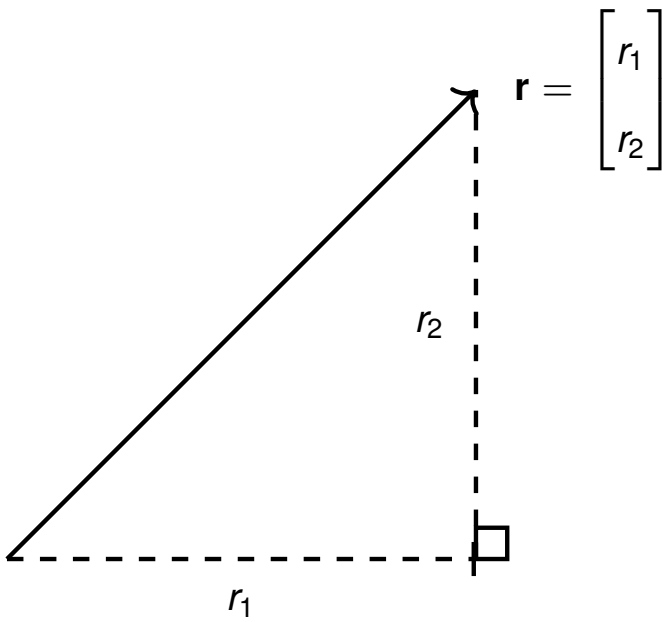
Reminders: Euclidean Norm

Euclidean Norm: Reminder (1/2)



Reminders: Euclidean Norm

Euclidean Norm: Reminder (2/2)



From the Pythagorean Theorem:

$$\|r\| = \sqrt{r_1^2 + r_2^2} = \sqrt{\langle r, r \rangle} = \sqrt{\sum_{i=1}^{2} r_i^2}$$

Reminders: Scalar projection and Vector Projection

Let two vectors in $\mathbb{R}^d$, $r = (r_1, \ldots, r_d)'$ and $s = (s_1, \ldots, s_d)'$.

- **Scalar projection** of vector s onto vector r:

$$\frac{\langle r, s \rangle}{\|r\|}$$

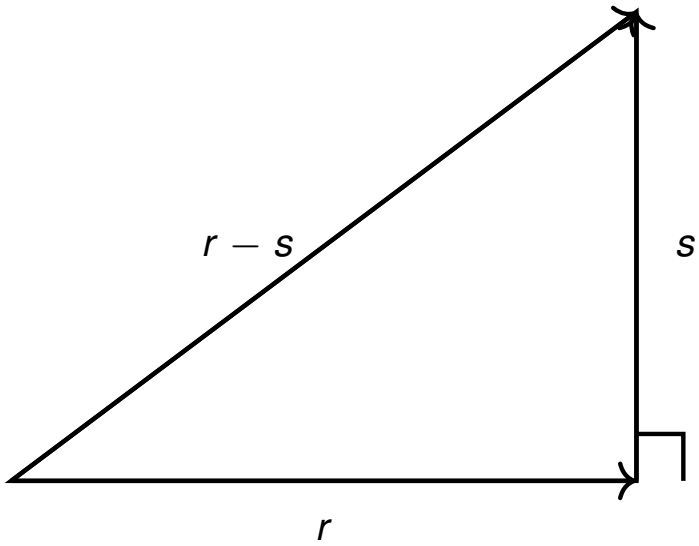
- **Vector projection** of vector s onto vector r:

$$\frac{\langle r, s \rangle}{\|r\|} \times \frac{r}{\|r\|}$$

Reminders: Orthogonality

- **Orthogonality**: r and s are orthogonal ($\perp$) if

$$\langle r, s \rangle = 0$$

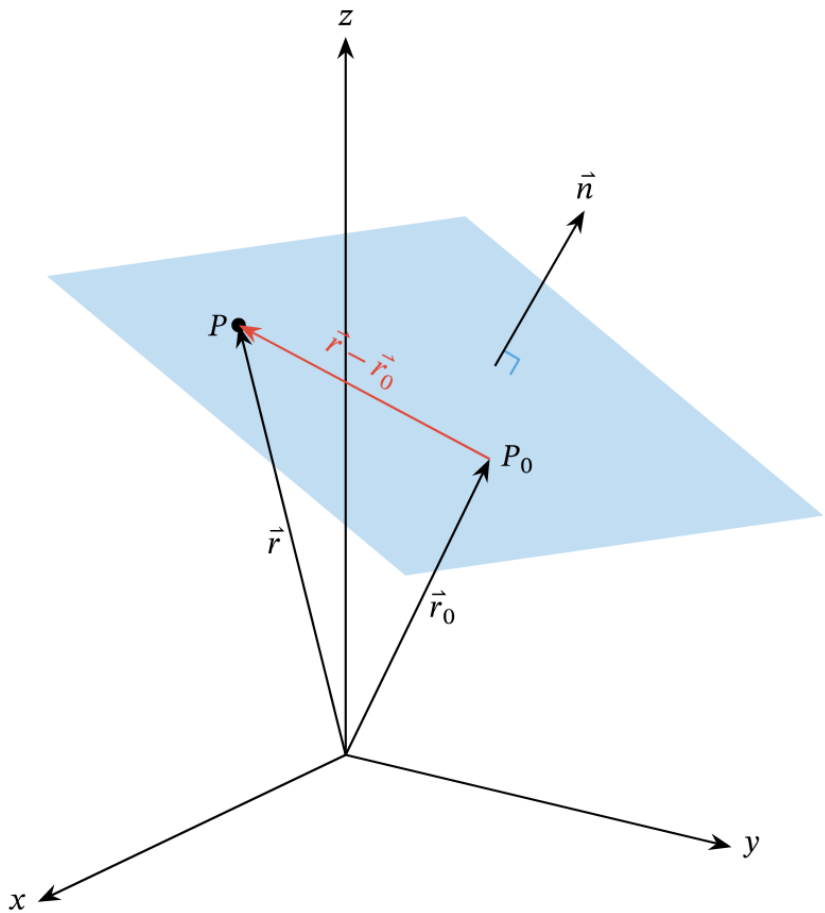


Reminder: Normal Vector

Reminders: Normal Line and Normal Vector

- The normal line to a plane at a given point is the line orthogonal to the plane at that point.
- Any direction vector of this line is called a normal vector to the surface at that point.
- By convention, the normal vector has a unit norm.

Normal Vector: Illustration



Source: Nagwa

Formalization of SVM

Notations

- Consider a **decision function** defined by
$$h(x) = \langle \omega, x \rangle + b$$
with $\omega \in \mathbb{R}^d$ and $b \in \mathbb{R}$.
- The equation $h(x) = 0$ defines the **separating hyperplane** H in $\mathbb{R}^d$.
- By definition of the separating hyperplane, $\forall x_0 \in H$:
$$h(x_0) = \langle \omega, x_0 \rangle + b = 0 \iff \langle \omega, x_0 \rangle = -b$$
- **Associated classifier**:
$$\forall x \in \mathcal{X}, \quad g(x) = \operatorname{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0. \end{cases}$$

Normal Vector: Definition

Definition: Normal Vector

The **normal vector** to the separating hyperplane H is defined as

$$\widetilde{\omega} = \frac{\omega}{\|\omega\|}$$

Exercise: Normal Vector

Exercise: Normal Vector in the Case $d = 2$

We consider a classification problem with two features $x = (x_1, x_2)' \in \mathcal{X} \subseteq \mathbb{R}^2$.
Let a linear classifier $g(x) = \text{sgn}(h(x))$ with

$$h(x) = \langle \omega, x \rangle + b = x_1 + 2x_2 - 1$$

and

$$\omega = (1, 2), \qquad b = -1.$$

Question: represent $\widetilde{\omega}$, the normal vector to the separating hyperplane H.

Exercise: Normal Vector

Solution: The equation defining the separating hyperplane H for $\omega = (1, 2)$ is

$$h(x) = x_1 + 2x_2 - 1 = 0, \qquad \forall x = (x_1, x_2) \in \mathbb{R}^2.$$

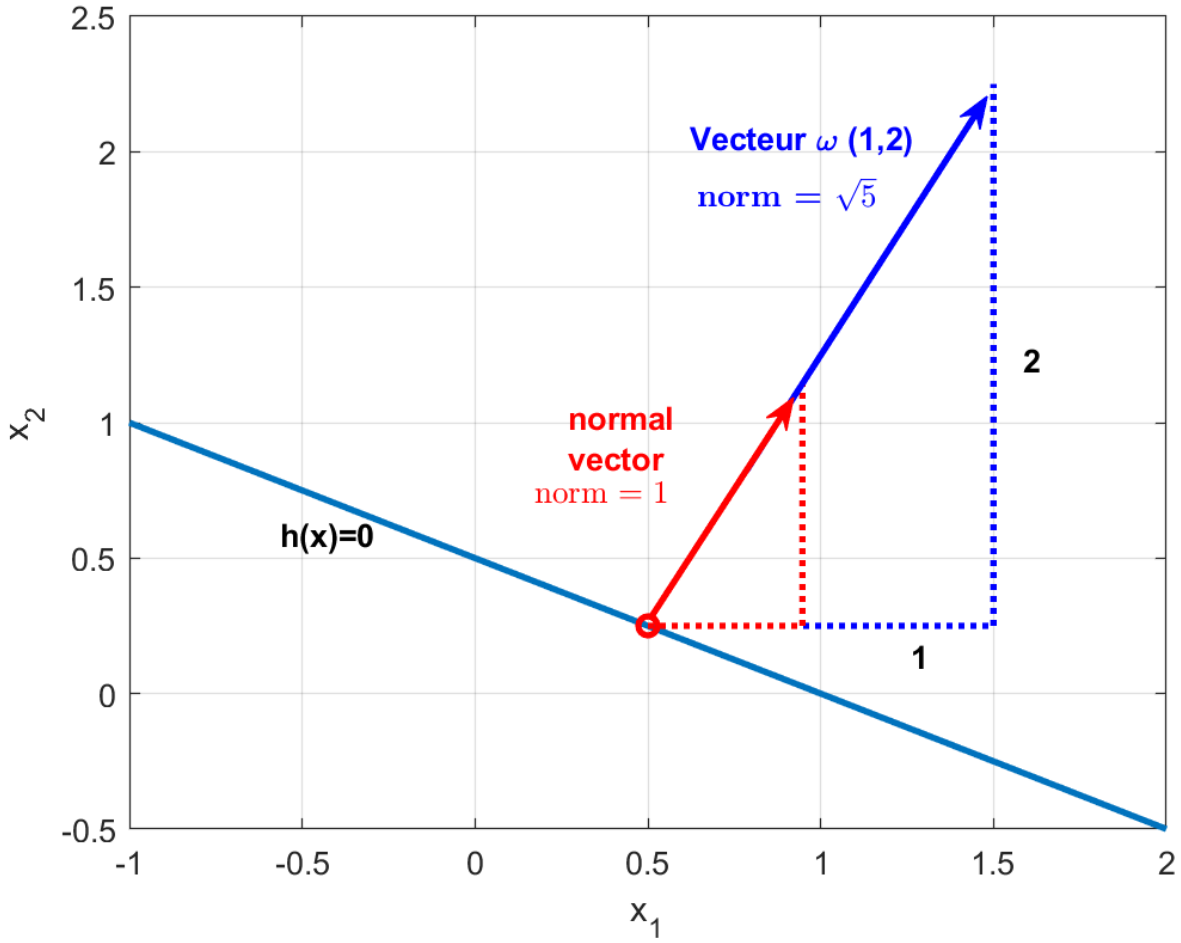
The normal vector to H is defined as

$$\widetilde{\omega} = \frac{\omega}{\|\omega\|} = \frac{1}{\sqrt{1+4}} \times (1, 2) = \left(\tfrac{1}{\sqrt{5}}, \tfrac{2}{\sqrt{5}}\right).$$

By definition,

$$\|\widetilde{\omega}\| = \sqrt{\left(\tfrac{1}{\sqrt{5}}\right)^2 + \left(\tfrac{2}{\sqrt{5}}\right)^2} = 1.$$

Exercise: Normal Vector



Distance from a Point to the Hyperplane

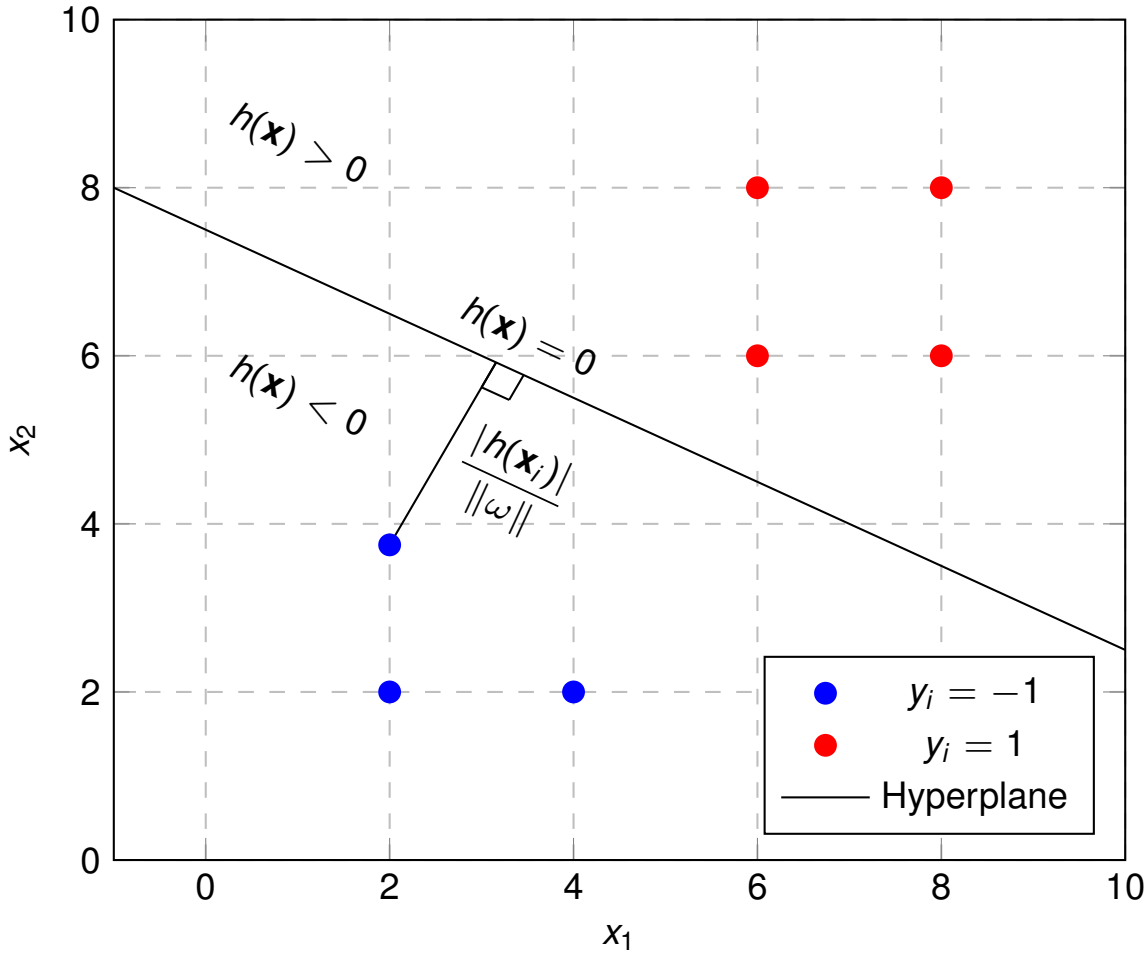
Definition: Distance from a Point to the Hyperplane

The **distance** from a point $x \in \mathbb{R}^d$ to the hyperplane H is

$$d(x, H) = \left|\langle \widetilde{\omega},\ x - x_0 \rangle\right| = \frac{\left|\langle \omega, x \rangle + b\right|}{\|\omega\|} = \frac{|h(x)|}{\|\omega\|},$$

where $x_0 \in H$ and $\widetilde{\omega}$ is the unit normal vector to the separating hyperplane H.

Distance from a Point to the Hyperplane



Distance from a Point to the Hyperplane

Proof:

$$
\begin{aligned}
d(x, H) &= \left|\langle \widetilde{\omega},\ x - x_0 \rangle\right| \\
&= \frac{1}{\|\omega\|} \left|\langle \omega,\ x - x_0 \rangle\right| \\
&= \frac{1}{\|\omega\|} \left|\langle \omega, x \rangle - \langle \omega, x_0 \rangle\right| \\
&= \frac{1}{\|\omega\|} \left|\langle \omega, x \rangle + b\right| \\
&= \frac{|h(x)|}{\|\omega\|}.
\end{aligned}
$$

Exercise: Distance to the Hyperplane

Exercise: Distance from a Point to the Hyperplane

Consider $\mathcal{X} = \mathbb{R}^2$. A linear classifier $g(x) = \text{sgn}(h(x))$ with

$$h(x) = \langle \omega, x \rangle + b, \qquad \omega = (1, 2), \ b = -1.$$

Question: what is the distance to the separating hyperplane for the following points?

$$A\left(1, \tfrac{3}{2}\right) \qquad B\left(-\tfrac{1}{2}, -\tfrac{1}{2}\right).$$

Exercise: Distance to the Hyperplane

Solution:

$$\omega = (1, 2) \quad b = -1 \qquad A(1, \tfrac{3}{2}) \qquad B(-\tfrac{1}{2}, -\tfrac{1}{2})$$

$$\|\omega\| = \sqrt{1^2 + 2^2} = \sqrt{5}.$$

$$d(A, H) = \frac{|h(x_A)|}{\|\omega\|} = \frac{|1 \times 1 + 2 \times (3/2) - 1|}{\sqrt{5}} = \frac{3}{\sqrt{5}}.$$

$$d(B, H) = \frac{|h(x_B)|}{\|\omega\|} = \frac{\left|1 \times (-\tfrac{1}{2}) + 2 \times (-\tfrac{1}{2}) - 1\right|}{\sqrt{5}} = \frac{2.5}{\sqrt{5}}.$$

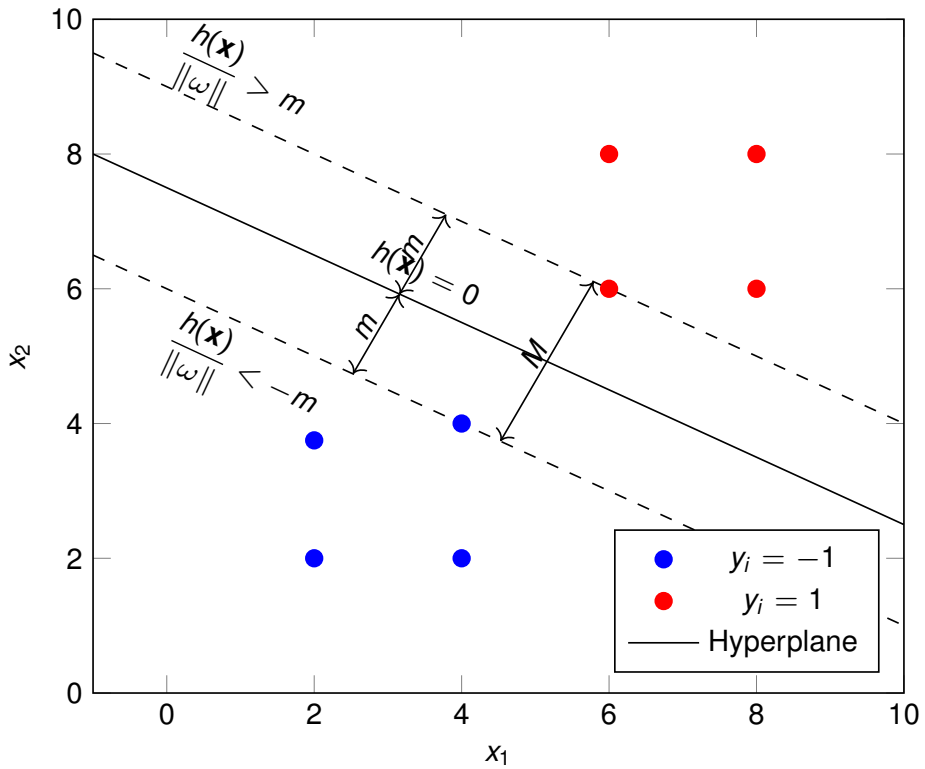
Formalization of the Margin

The goal of SVMs is to choose the separating hyperplane that:

- perfectly classifies all observations (in the linearly separable case),
- while maximizing the distance between the nearest observations from different classes, this distance is the margin, denoted M.
- The margin M equals twice the distance between the closest point(s) and the separating hyperplane (often called the support vectors), denoted m:

$$M = 2 \times m.$$

Formalization of the Margin



Thus, we have:

$$g(x_i) = \begin{cases} 1 & \text{if } \dfrac{h(\mathbf{x}_i)}{\|\omega\|} \geq m, \\ -1 & \text{if } \dfrac{h(\mathbf{x}_i)}{\|\omega\|} < -m. \end{cases}$$

To combine these two constraints into a single one, multiply by y_i:

$$y_i \frac{h(\mathbf{x}_i)}{\|\omega\|} \geq m \qquad \forall\, i = 1, \ldots, n.$$

Canonical Separating Hyperplane

Canonical Separating Hyperplane

This inequality holds for any $m > 0$. We can therefore set:

$$m = \frac{1}{\|\omega\|}.$$

We then obtain:

$$y_i\, h(x_i) \geq 1 \implies \min_{i=1,\ldots,n} |h(x_i)| = 1.$$

Definition: Canonical Separating Hyperplane

A separating hyperplane is said to be **canonical** with respect to the observations $\{x_1, \ldots, x_n\}$ if and only if

$$\min_{i=1,\ldots,n} |h(x_i)| = \min_i |\langle \omega, x_i \rangle + b| = 1.$$

By choosing $m = \frac{1}{\|\omega\|}$ we thus obtain a canonical separating hyperplane.

Normalization and Canonical Separating Hyperplane

Normalization and Canonical Separating Hyperplane

In practice, starting from a non-canonical separating hyperplane $h(x)$, we **normalize** the parameters ω and b so that the decision function $h(x)$ evaluated at a support vector equals $+1$ or -1.

$$\text{Choose } \gamma = \min_{i=1,\dots,n} |h(x_i)| > 0, \qquad (\omega', b') = \left(\frac{\omega}{\gamma}, \frac{b}{\gamma}\right), \qquad h'(x) = \langle \omega', x \rangle + b'.$$

$$\Rightarrow \quad \min_i |h'(x_i)| = 1, \quad \text{i.e., the hyperplane } h'(x) \text{ is canonical.}$$

Exercise: Canonical Separating Hyperplane

Exercise: Canonical Separating Hyperplane

Consider $\mathcal{X} = \mathbb{R}^2$ and a linear classifier $g(x) = \text{sgn}(\overline{h}(x))$, with

$$\overline{h}(x) = \langle \overline{\omega}, x \rangle + \overline{b}, \qquad x = (x_1, x_2)',$$

and

$$\overline{\omega} = (1, 2) \qquad \overline{b} = -1.$$

Assume that the vector $(1, 1)$ is a support vector.

Question: what is the decision function associated with the canonical hyperplane?

Exercise: Canonical Separating Hyperplane

Solution:

$$\overline{h}(x) = \langle \overline{\omega}, x \rangle + \overline{b} = x_1 + 2x_2 - 1, \qquad x = (x_1, x_2)',$$

For the support vector $x_{sv} = (1, 1)$, we have

$$\overline{h}(x_{sv}) = 1 + 2 - 1 = 2 \qquad \Rightarrow \qquad \overline{g}(x_{sv}) = \text{sgn}(2) = 1.$$

The canonical hyperplane is then defined by

$$h(x) = \tfrac{1}{2}\,\overline{h}(x) = 0,$$

or equivalently

$$h(x) = \langle \omega, x \rangle + b = 0.5\,x_1 + x_2 - 0.5 = 0,$$

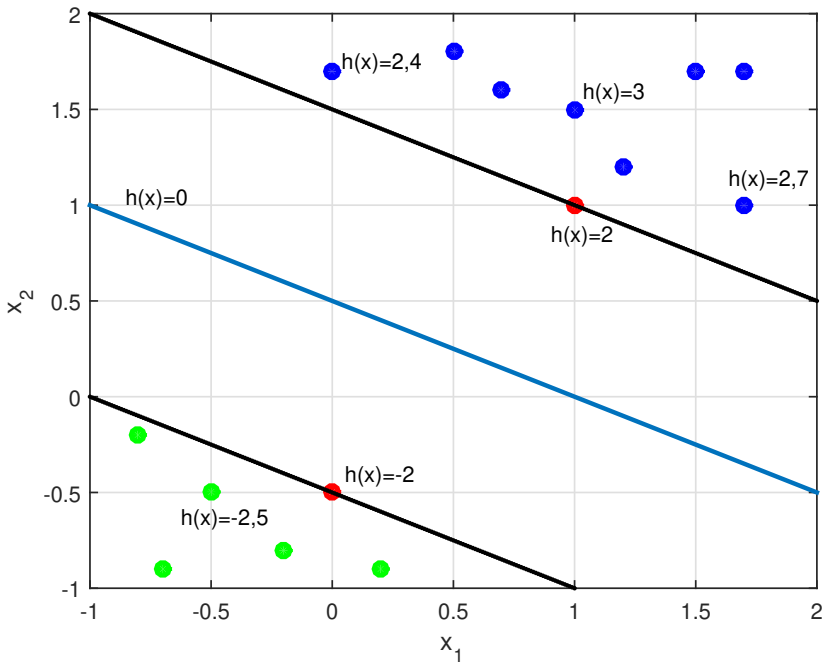
$$\omega = (0.5,\ 1) \qquad b = -0.5.$$

By definition,

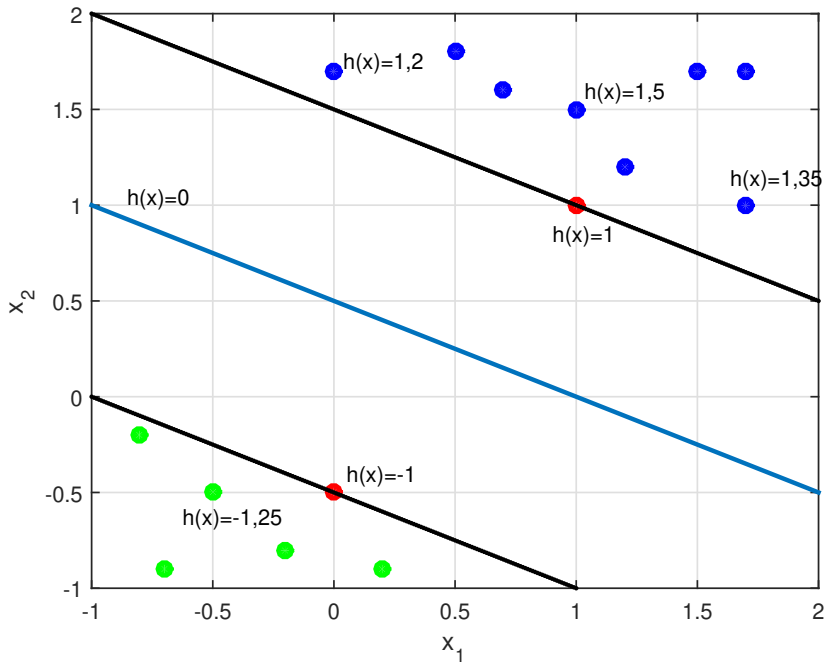
$$h(x_{sv}) = 0.5 + 1 - 0.5 = 1 \qquad \Rightarrow \qquad g(x_{sv}) = \text{sgn}(1) = 1.$$

Canonical Separating Hyperplane

$$\overline{h}(x) = x_1 + 2x_2 - 1$$



$$h(x) = 0.5\,x_1 + x_2 - 0.5$$



Support Vectors: Definition

Definition: Support Vectors

The **support vectors** are the observations $x_i \in S$ that satisfy

$$h(x_i) = \pm 1 \quad \forall i \in S$$

or equivalently

$$y_i \, h(x_i) = 1 \quad \forall i \in S.$$

where $h(x) = \big(\langle \omega, x \rangle + b\big)$ is a canonical hyperplane,

Perfect Classification

Perfect Classification

For a canonical hyperplane, the conditions for a **perfect classification** (on the training sample) are:

$$h(x_i) = \langle \omega, x_i \rangle + b \geq 1 \quad \text{if } y_i = 1$$

$$h(x_i) = \langle \omega, x_i \rangle + b \leq -1 \quad \text{if } y_i = -1$$

for all $i = 1, \ldots, n$. These can be written compactly as:

$$y_i \, h(x_i) \geq 1 \quad \forall i = 1, \ldots, n.$$

Formal Definition of the Margin

Definition: Margin

The **margin** $M(\omega, b)$ is twice the distance between a support vector x_{sv} and the canonical separating hyperplane H.

$$M(\omega, b) = 2 \times d(x_{sv}, H) = 2 \times \frac{|h(x_{sv})|}{\|\omega\|} = \frac{2}{\|\omega\|},$$

since by definition $h(x_{sv}) = \pm 1$.

Optimal Canonical Hyperplane

Optimal Canonical Hyperplane

A canonical hyperplane $h(x) = \langle \omega^*, x \rangle + b^* = 0$ is **optimal** with respect to the observations $\{(x_i, y_i)\}_{i=1}^n$ if it maximizes the margin $M(\omega, b)$ and perfectly classifies all observations. It satisfies:

$$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^d,\ b \in \mathbb{R}}{\arg\max} \frac{2}{\|\omega\|}$$

$$\text{subject to} \quad y_i\, h(x_i) \geq 1 \quad \forall\, i = 1, \ldots, n.$$

Optimal Canonical Hyperplane

Finding a hyperplane that maximizes the margin $M(\omega)$ is equivalent to finding parameters (ω^*, b^*) such that:

$$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\max} \; M(\omega) = \frac{2}{\|\omega\|}$$
$$\text{subject to} \quad y_i \, h(x_i) \geq 1 \quad \forall \, i = 1, \ldots, n.$$

This is equivalent to:

$$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \; \frac{1}{2} \|\omega\|^2$$
$$\text{subject to} \quad y_i \, h(x_i) \geq 1 \quad \forall \, i = 1, \ldots, n.$$

SVM Primal Program

Definition: SVM Primal Program

The SVM **primal program** is:

$$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \frac{1}{2} \|\omega\|^2$$

$$\text{subject to} \quad y_i \, h(x_i) \geq 1 \quad \forall \, i = 1, \ldots, n,$$

with

$$h(x) = \langle \omega, x \rangle + b, \qquad \forall \, x \in \mathcal{X}.$$

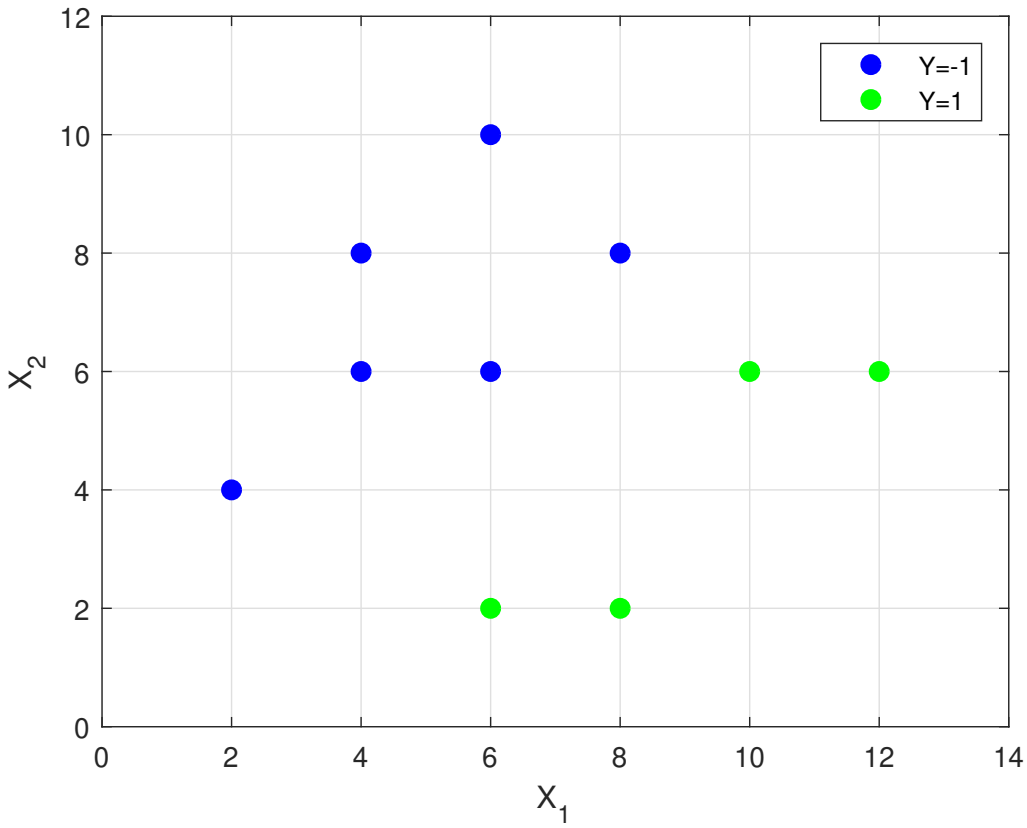
Exercise: SVM Primal Program

Exercise: SVM Primal Program

Let $x = (x_1, x_2)' \in \mathbb{R}^2$ and a training sample of size $n = 10$ with the following data. **Task:** using a quadratic programming solver, solve the SVM primal program and determine the equation of the optimal canonical separating hyperplane.

x_1	x_2	y
4	8	-1
2	4	-1
4	6	-1
6	6	-1
8	8	-1
6	10	-1
12	6	1
10	6	1
8	2	1
6	2	1

Exercise: SVM Primal Program



Solution: SVM Primal Program

Solution

$$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^2,\, b \in \mathbb{R}}{\arg\min} \; \frac{1}{2} \|\omega\|^2$$

$$\text{subject to} \;\; y_i \, h(x_i) \geq 1 \quad \forall \, i = 1, \ldots, n$$

Write this program in matrix form to use a quadratic programming solver in Python:

$$\theta = \begin{pmatrix} \omega_1 & \omega_2 & b \end{pmatrix}'$$

$$\frac{1}{2} \|\omega\|^2 = \frac{1}{2} (\omega_1^2 + \omega_2^2) = \frac{1}{2} \begin{pmatrix} \omega_1 & \omega_2 & b \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ b \end{pmatrix} = \frac{1}{2} \theta^\top P \, \theta,$$

Link to Google Colab: Click here

Solution: SVM Primal Program

Solution

We obtain

$$\omega^* = \left(\frac{1}{2}, -\frac{1}{2}\right)' \qquad b^* = -1.$$

The decision function is therefore

$$h(x_i) \;=\; \langle \omega^*, x_i \rangle + b^* \;=\; \tfrac{1}{2}\, x_{1i} - \tfrac{1}{2}\, x_{2i} - 1 \qquad \forall\, x_i \in \mathbb{R}^2.$$

The equation of the optimal canonical separating hyperplane is

$$h(x_i) = 0 \iff x_{2i} = x_{1i} - 2\,.$$

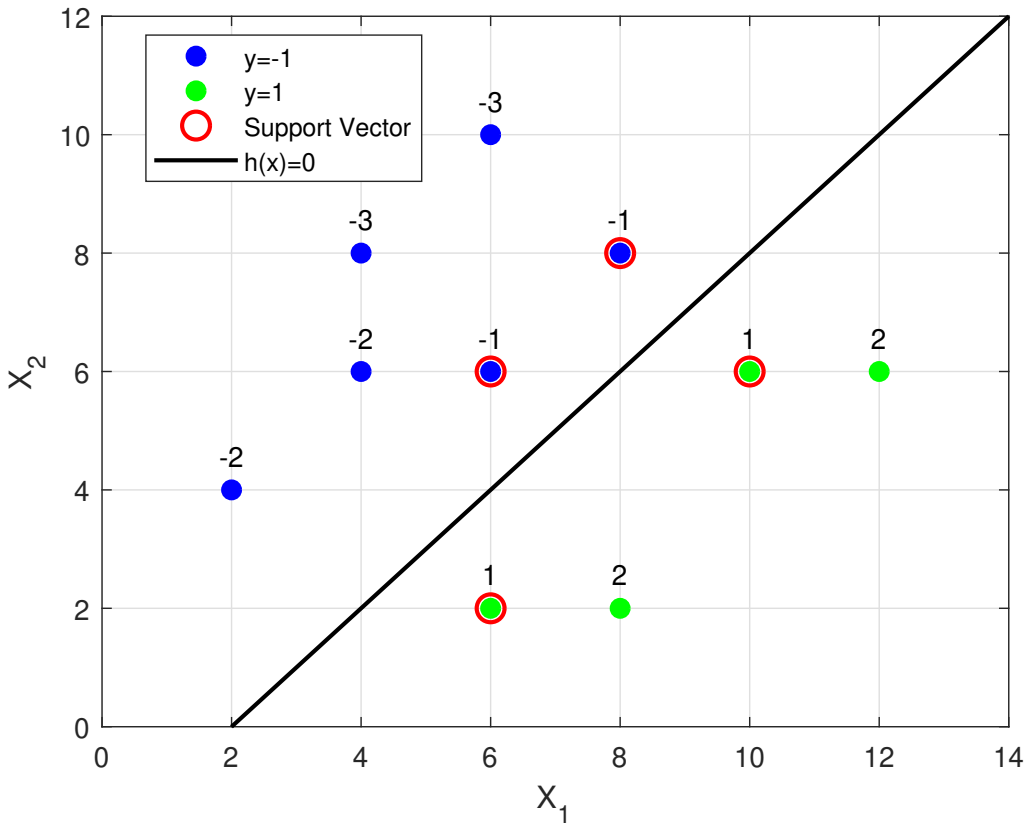
Solution: SVM Primal Program

Solution

$$h(x_i) = \frac{1}{2} x_{1i} - \frac{1}{2} x_{2i} - 1 \qquad h(x_i)\, y_i \geq 1$$

x_{1i}	x_{2i}	y_i	$h(x_i)$	$y_i h(x_i)$	Status
4	8	-1	-3	3	—
2	4	-1	-2	2	—
4	6	-1	-2	2	—
6	6	-1	-1	1	support vector
8	8	-1	-1	1	support vector
6	10	-1	-3	3	—
12	6	1	2	2	—
10	6	1	1	1	support vector
8	2	1	2	2	—
6	2	1	1	1	support vector

Solution: SVM Primal Program



SVM Implementation with `scikit-learn`

SVC

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False,
random_state=None)
```

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer or other Kernel Approximation.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

To learn how to tune SVC's hyperparameters, see the following example: Nested versus non-nested cross-validation

Read more in the User Guide.

Source: scikit-learn documentation

Example: SVM Implementation

Example: SVM Implementation

```
# Hard-margin SVM
clf = SVC(kernel="linear", C=1e20)
clf.fit(X, y)

# Coefficients and intercept
w = clf.coef_
b = clf.intercept_

# Support vectors
sv=clf.support_vectors_
```

Google Colab: Click here.

Primal vs. Dual Optimization Problems

Primal vs. Dual Optimization Programs

$$\begin{aligned} \min_{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}} \quad & \frac{1}{2} \|\omega\|^2 \\ \text{subject to} \quad & y_i\, h(x_i) \geq 1 \qquad \forall\, i = 1, \ldots, n \end{aligned}$$

- Since this optimization problem is convex, solving the primal or the dual is equivalent.
- The dual problem introduces Lagrange multipliers associated with the n constraints (one per example/observation).

Lagrangian

Lagrangian

The **Lagrangian** associated with the SVM program is

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{n} \lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b \big) - 1 \Big],$$

with $\lambda = (\lambda_1, \ldots, \lambda_n)'$ the vector of multipliers satisfying, for all $i = 1, \ldots, n$,

$$\lambda_i \geq 0$$

$$\lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b \big) - 1 \Big] = 0.$$

Remarks

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{n} \lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b\big) - 1 \Big]$$

$$\lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b\big) - 1 \Big] = 0 \qquad \text{with } \lambda_i \geq 0$$

1. The Lagrangian must be *minimized* with respect to ω and b, and *maximized* with respect to λ (saddle point).
2. If the constraint $\big[y_i(\langle \omega, x_i \rangle + b) - 1 \big] = 0$ is active, then $\lambda_i > 0$ and x_i is a support vector.
3. If $\big[y_i(\langle \omega, x_i \rangle + b) - 1 \big] > 0$, then $\lambda_i = 0$ and the point x_i, lying beyond the margin, is correctly classified. In practice, many λ_i are zero (sparsity).
4. Denote $x_i = (x_{i1}, \ldots, x_{id})'$, $\forall i$ and $\omega = (\omega_1, \ldots, \omega_d)'$, the Lagrangian can be written as:

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\omega^\top \omega - \sum_{i=1}^{n} \lambda_i y_i \, x_i^\top \omega - b \sum_{i=1}^{n} \lambda_i y_i + \sum_{i=1}^{n} \lambda_i .$$

Karush-Kuhn-Tucker Conditions

Karush-Kuhn-Tucker (KKT) Conditions

The **Karush-Kuhn-Tucker conditions** for the SVM primal are:

$$\frac{\partial \mathcal{L}(\omega, b, \lambda)}{\partial \omega} = \omega - \sum_{i=1}^{n} \lambda_i y_i x_i = 0$$

$$\frac{\partial \mathcal{L}(\omega, b, \lambda)}{\partial b} = -\sum_{i=1}^{n} \lambda_i y_i = 0$$

$$\lambda_i \left(y_i \, h(x_i) - 1 \right) = \lambda_i \left[y_i \left(\langle \omega, x_i \rangle + b \right) - 1 \right] = 0$$

$$\lambda_i \geq 0, \qquad \forall \, i = 1, \ldots, n.$$

Note: Introducing the KKT conditions yields an optimization that depends only on the multipliers:

SVM Dual Problem

Definition: SVM Dual Problem

The SVM program can be written in **dual form** as:

$$\lambda^* = \underset{\lambda_1,\ldots,\lambda_n}{\arg\max} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{subject to} \quad \lambda_i \geq 0 \quad \forall\, i = 1, \ldots, n$$

$$\text{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0.$$

Proof: see appendix.

Exercise: SVM Dual Program

Exercise: SVM Dual Program

We consider the same training sample (see table).

Task: using a quadratic programming solver, solve the SVM *dual* program and determine the equation of the optimal canonical separating hyperplane.

Google Colab: Click here.

Obs.	$x_{1,i}$	$x_{2,i}$	y_i
1	4	8	-1
2	2	4	-1
3	4	6	-1
4	6	6	-1
5	8	8	-1
6	6	10	-1
7	12	6	1
8	10	6	1
9	8	2	1
10	6	2	1

Solution: SVM Dual Program

$$\lambda^* = \arg\max_{\lambda_1,\ldots,\lambda_n} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{s.t. } \lambda_i \geq 0, \qquad \sum_{i=1}^{n} \lambda_i y_i = 0$$

x_{1i}	x_{2i}	y_i	Statut	λ_i
4	8	-1	—	0
2	4	-1	—	0
4	6	-1	—	0
6	6	-1	support vector	0,1528
8	8	-1	support vector	0,0972
6	10	-1	—	0
12	6	1	—	0
10	6	1	support vector	0,1736
8	2	1	—	0
6	2	1	support vector	0,0764

Solution: SVM Dual Program

Solution: The coefficient vector is

$$\omega^* = \sum_{i \in S} \lambda_i^* y_i x_i = 0.1528(-1)\begin{pmatrix} 6 \\ 6 \end{pmatrix} + 0.0972(-1)\begin{pmatrix} 8 \\ 8 \end{pmatrix} + 0.1736(+1)\begin{pmatrix} 10 \\ 6 \end{pmatrix} + 0.0764(+1)\begin{pmatrix} 6 \\ 2 \end{pmatrix}.$$

$$\Rightarrow \quad \omega^* = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}.$$

The bias satisfies, for any support vector x_i,

$$b^* = y_i - \langle \omega^*, x_i \rangle, \qquad \forall i \in S.$$

Using $x = (6, 6)'$ with $y = -1$:

$$b^* = -1 - \left(\frac{1}{2} \cdot 6 - \frac{1}{2} \cdot 6 \right) = -1.$$

(One gets the same b^* using any of the support vectors.)

Solution: SVM Dual Program

Solution: For $x = (x_1, x_2)' \in \mathbb{R}^2$, the decision function is

$$h(x) = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^* \;=\; \langle \omega^*, x \rangle + b^*.$$

With the dual solution,

$$\omega^* = \begin{pmatrix} \frac{1}{2} \\ \\ -\frac{1}{2} \end{pmatrix}, \qquad b^* = -1.$$

Hence the optimal canonical separating hyperplane is

$$h(x_j) = \tfrac{1}{2} x_{1j} - \tfrac{1}{2} x_{2j} - 1 = 0,$$

or equivalently

$$x_{2j} = x_{1j} - 2.$$

Example: Dual Coefficients

Example: Dual Coefficients

```python
# Hard-margin SVM
clf = SVC(kernel="linear", C=1e20)
clf.fit(X, y)

# Dual coefficients (alpha_i * y_i for each support vector)
print("dual_coef_:", clf.dual_coef_)

# Absolute values give the alpha_i
alphas = np.abs(clf.dual_coef_[0])
print("alphas:", alphas)
```

Google Colab: Click here.

Optimal Decision Function

Optimal SVM Solution

1. Once the optimal multipliers λ^* are obtained, the coefficient vector is

$$\omega^* = \sum_{i=1}^{n} \lambda_i^* y_i x_i = \sum_{i \in S} \lambda_i^* y_i x_i,$$

where S denotes the set of support vectors (since $\lambda_i^* = 0$ for $i \notin S$).

2. To determine the constant b^*, take any support vector $i \in S$ such that

$$y_i \left(\langle \omega^*, x_i \rangle + b^* \right) = 1.$$

Then

$$b^* = \frac{1}{y_i} - \langle \omega^*, x_i \rangle, \qquad \forall\, i \in S.$$

Since $y_i \in \{-1, 1\}$, this can also be written as

$$b^* = y_i - \langle \omega^*, x_i \rangle, \qquad \forall\, i \in S.$$

Optimal Decision Function

Definition: Optimal Decision Function

For any $x \in \mathcal{X}$, the **optimal decision function** is

$$h(x) \;=\; \langle \omega^*, x \rangle + b^* \;=\; \sum_{i \in S} \lambda_i^* \, y_i \, \langle x_i, x \rangle \;+\; b^*,$$

where S denotes the set of support vectors. The associated classifier is

$$g(x) \;=\; \operatorname{sgn}\big(h(x)\big) \;=\; \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0. \end{cases}$$

Main advantage of the SVM: Only the support vectors (weighted by λ_i) determine the classification of all observations in both the training and test samples.

Exercise: SVM and Prediction

Exercise: SVM and Prediction

We consider the same training sample as before. We want to predict the class for the following test observations:

$$x_A = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$$

$$x_B = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$$

Task: Using only the Lagrange multipliers λ_i of the support vectors (see previous table), determine the predictions y_A and y_B.

Link to Google Colab: Click here

Solution: SVM and Prediction

x_1	x_2	y	$h(x)$	$y\,h(x)$	Status	λ
6	6	-1	-1	1	support vector	0.1528
8	8	-1	-1	1	support vector	0.0972
10	6	1	1	1	support vector	0.1736
6	2	1	1	1	support vector	0.0764

Solution: Using only the support vectors, the decision function is

$$\widehat{h}(x) = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^*.$$

This gives

$$\begin{aligned}\widehat{h}(x) = &-0.1528\,(6x_1 + 6x_2) - 0.0972\,(8x_1 + 8x_2) \\ &+ 0.1736\,(10x_1 + 6x_2) + 0.0764\,(6x_1 + 2x_2) - 1\end{aligned}$$

or equivalently

$$\widehat{h}(x) = \frac{1}{2}x_1 - \frac{1}{2}x_2 - 1.$$

Solution: SVM and Prediction

Solution: the decision function (from the dual and primal forms) is

$$\widehat{h}(x) = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^* = \tfrac{1}{2} x_1 - \tfrac{1}{2} x_2 - 1, \qquad b^* = -1.$$

with

$$x_A = \begin{pmatrix} 6 \\ 8 \end{pmatrix} \quad \text{and} \quad x_B = \begin{pmatrix} 10 \\ 2 \end{pmatrix},$$

we obtain

$$\widehat{h}(x_A) = \tfrac{1}{2} \cdot 6 - \tfrac{1}{2} \cdot 8 - 1 = -2.$$

$$\widehat{h}(x_B) = \tfrac{1}{2} \cdot 10 - \tfrac{1}{2} \cdot 2 - 1 = 3.$$

Predictions:

$$\widehat{y}_A = \operatorname{sgn}\left(\widehat{h}(x_A)\right) = \operatorname{sgn}(-2) = -1, \qquad \widehat{y}_B = \operatorname{sgn}\left(\widehat{h}(x_B)\right) = \operatorname{sgn}(3) = 1.$$

Example: SVM Predictions

Example: SVM Predictions

```
# Hard-margin SVM
clf = SVC(kernel="linear", C=1e20)
clf.fit(X, y)

# Test dataset
X_test = np.array([[6, 8], [10, 2]])

# Predictions
y_pred = clf.predict(X_test)
```

Google Colab: Click here.

SVM Primal and Dual Programs

SVM Primal Program	SVM Dual Program
	$\lambda^* = \underset{\lambda_1,..,\lambda_n}{\arg\max} \sum_{i=1}^n \lambda_i$
$(\omega^*, b^*) = \underset{\omega \in \mathbb{R}^d, b \in \mathbb{R}}{\arg\min} \frac{1}{2} \|\omega\|^2$	$-\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$
sc : $y_i \, h(x_i) \geq 1 \quad \forall i = 1, \ldots, n$	sc : $\lambda_i \geq 0$
	sc : $\sum_{i=1}^n \lambda_i y_i = 0$
$h(x) = \langle \omega^*, x \rangle + b^*$	$\omega^* = \sum_{i \in S} \lambda_i^* y_i x_i$
	$b^* = y_i - \langle \omega^*, x_i \rangle , \quad \forall i \in S$
	$h(x) = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^*$

Gram matrix

The term $\langle x_i, x_j \rangle$ is the entry in the i-th row and j-th column of the **Gram matrix** (which also appears as the Hessian in the dual):

$$\text{Gram matrix} = \underset{n \times n}{\Omega} = \begin{pmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \cdots & \cdots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \langle x_n, x_1 \rangle & \cdots & \cdots & \cdots & \langle x_n, x_n \rangle \end{pmatrix}.$$

Remark: If $X = (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$ collects the $x_i^\top$ as rows, then the Gram matrix can be written as

$$\Omega = X X^\top.$$

Exercise: Gram Matrix

Exercise: Gram Matrix

We consider the same training sample. **Task:** determine the Gram matrix.

Obs.	$x_{1,i}$	$x_{2,i}$	y_i
1	4	8	-1
2	2	4	-1
3	4	6	-1
4	6	6	-1
5	8	8	-1
6	6	10	-1
7	12	6	1
8	10	6	1
9	8	2	1
10	6	2	1

Solution: Gram Matrix

Solution: The Gram matrix associated with the linear kernel is

$$\underset{(n\times n)}{\Omega} = \begin{pmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \cdots & \cdots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \langle x_n, x_1 \rangle & \cdots & \cdots & \cdots & \langle x_n, x_n \rangle \end{pmatrix}.$$

Here, with $n = 10$ we obtain

$$\underset{(10\times 10)}{\Omega} = \begin{pmatrix} 80 & 40 & 64 & 72 & 96 & 104 & 96 & 88 & 48 & 40 \\ 40 & 20 & 32 & 36 & 48 & 52 & 48 & 44 & 24 & 20 \\ 64 & 32 & 52 & 60 & 80 & 84 & 84 & 76 & 44 & 36 \\ 72 & 36 & 60 & 72 & 96 & 96 & 108 & 96 & 60 & 48 \\ 96 & 48 & 80 & 96 & 128 & 128 & 144 & 128 & 80 & 64 \\ 104 & 52 & 84 & 96 & 128 & 136 & 132 & 120 & 68 & 56 \\ 96 & 48 & 84 & 108 & 144 & 132 & 180 & 156 & 108 & 84 \\ 88 & 44 & 76 & 96 & 128 & 120 & 156 & 136 & 92 & 72 \\ 48 & 24 & 44 & 60 & 80 & 68 & 108 & 92 & 68 & 52 \\ 40 & 20 & 36 & 48 & 64 & 56 & 84 & 72 & 52 & 40 \end{pmatrix}.$$

Importance of Standardisation in SVM

Definition: Standardisation

The **standardised** feature x_j is defined as

$$z_j = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j is the sample mean and σ_j the sample standard deviation of feature x_j.

Why is standardisation important for SVMs?

- SVMs rely on inner products and distances between vectors.
- Features with larger scales dominate the margin calculation.
- Standardisation ensures that all features contribute equally to the separating hyperplane.

Formalization of the Support Vector Machine

Key Concepts

1. Margin
2. Support vectors
3. Canonical hyperplane
4. Optimal hyperplane
5. SVM Primal program
6. Lagrangian and KKT conditions
7. SVM Dual program
8. Gram matrix

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

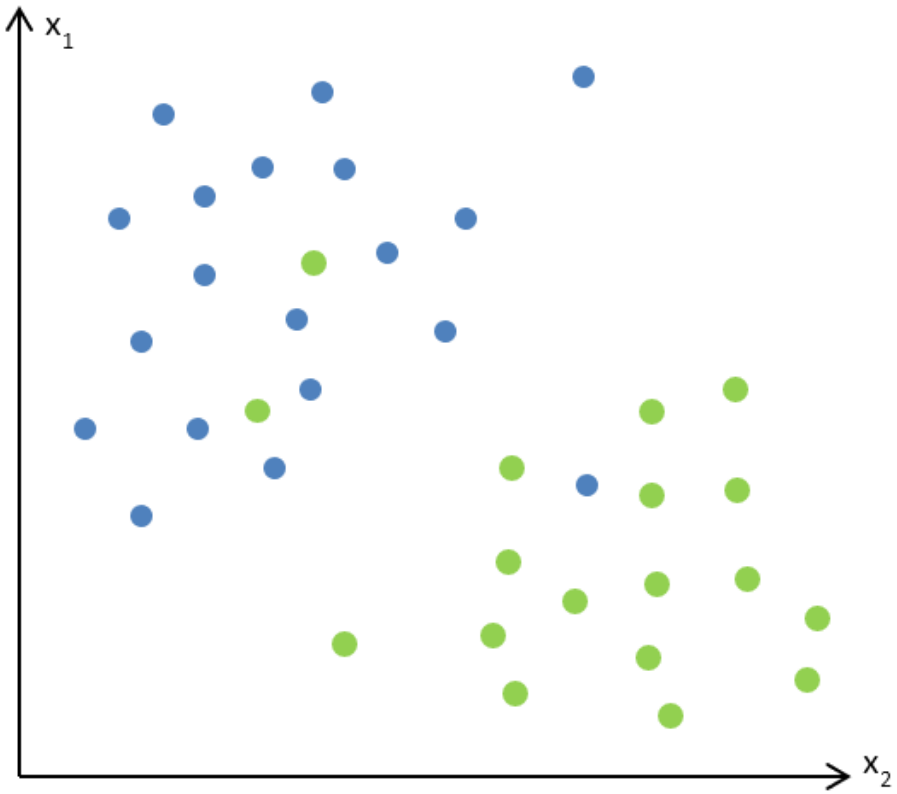
A Sample Not Linearly Separable

We now assume that the training sample is not linearly separable.

Two cases:

1. The sample is almost linearly separable: the "optimal" separation is linear, but some observations cannot be correctly classified.
2. The sample is not linearly separable: the "optimal" separation is non-linear.

A Sample Almost Linearly Separable



A sample that is almost linearly separable.

Slack Variables

Definition: Slack Variables

In the case of an **almost linearly separable** sample, we introduce n relaxation variables for the classification constraints $y_i\, h(x_i) \geq 1$.

These variables, denoted $\xi = (\xi_1, \ldots, \xi_n)'$, are called **slack variables** and satisfy:

$$y_i \big(\langle \omega, x_i \rangle + b \big) \;\geq\; 1 - \xi_i, \qquad \xi_i \geq 0.$$

Soft Margin

Definition: Soft Margin

A **soft margin** SVM allows classification constraints to be relaxed through the introduction of **slack variables** $\xi_i \geq 0$:

$$y_i\big(\langle \omega, x_i \rangle + b\big) \;\geq\; 1 - \xi_i, \qquad i = 1, \ldots, n.$$

The slack variables measure the extent of violation of the margin constraints.



Cortes, C. and V. Vapnik (1995), Support-Vector Networks, *Machine Learning*, 20.

Interpretation of the Slack Variables

Soft Margin and the Slack Variables

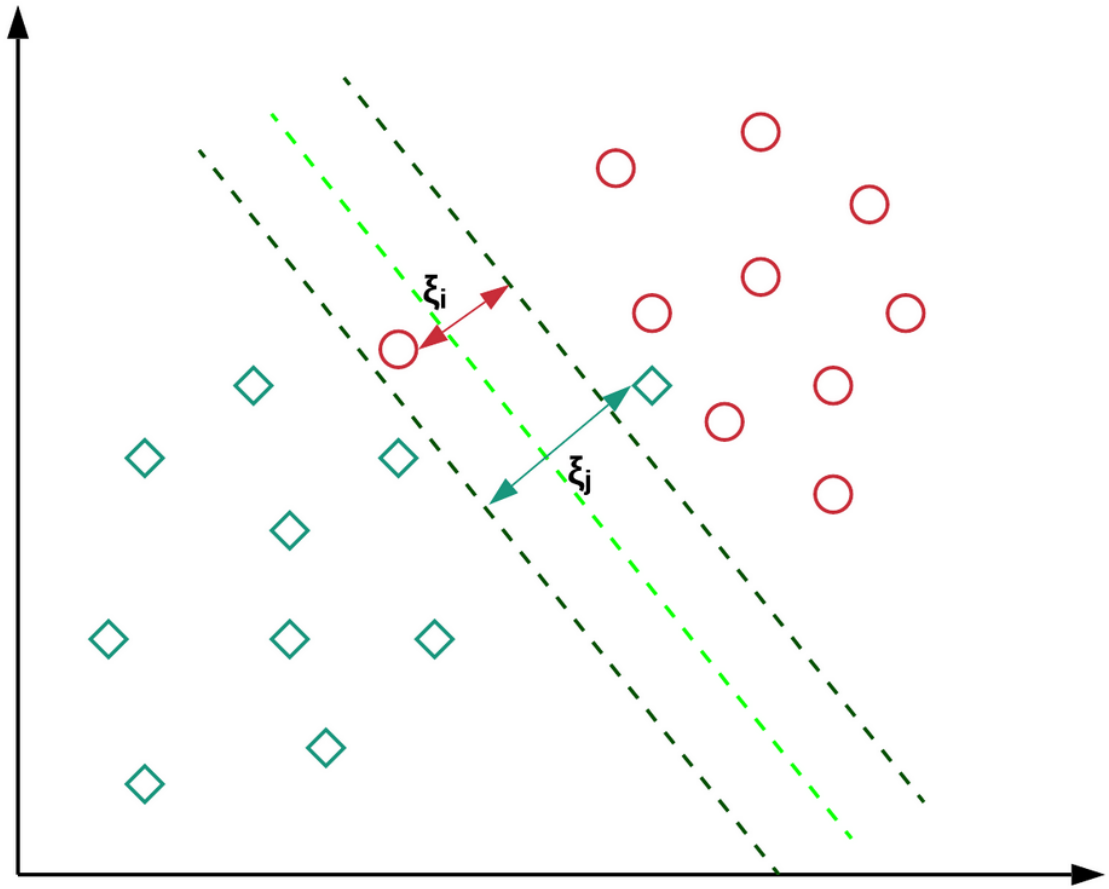
$$y_i\big(\langle \omega, x_i\rangle + b\big) \;\geq\; 1-\xi_i, \qquad \xi_i \geq 0.$$

Three configurations can occur depending on the value of ξ_i:

1. If $\xi_i = 0$, then $y_i h(x_i) \geq 1$: the observation (x_i, y_i) is correctly classified and outside the margin.
2. If $\xi_i \geq 1$, then the observation (x_i, y_i) is misclassified and lies on the wrong side of the separating hyperplane.
3. If $0 < \xi_i < 1$, then (x_i, y_i) is correctly classified but lies within the margin, i.e. at a distance from the separating hyperplane smaller than half of the margin:

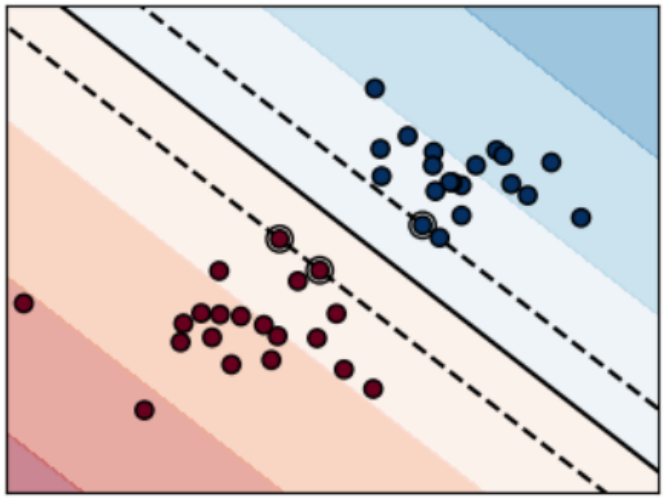
$$d(x_i, H) < \frac{1}{\|\omega\|}.$$

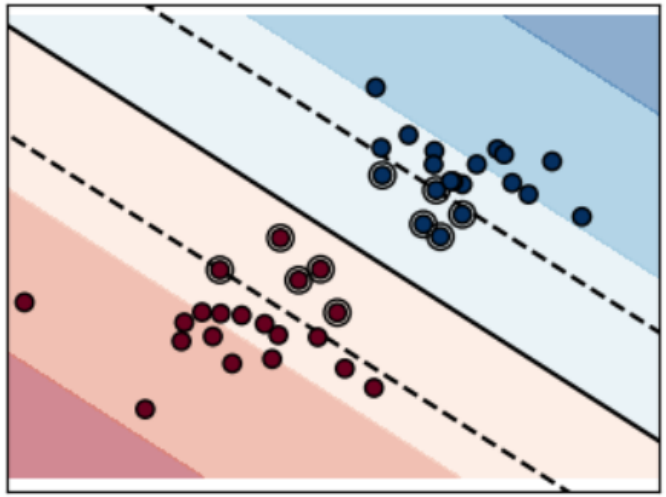
Soft Margin: Illustration



Source: https://towardsdatascience.com

Soft Margin: Illustration





Source: scikit-learn documentation

Primal Program with Soft Margin

Definition: Primal Program with Soft Margin

The **primal SVM problem in the non-separable case (with slack variables)** is defined as:

$$
\begin{aligned}
(\omega^*, b^*, \xi^*) &= \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \ \frac{1}{2}\|\omega\|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.} &: y_i\big(\langle \omega, x_i \rangle + b\big) \geq 1 - \xi_i, \qquad \forall i = 1, \ldots, n \\
\text{s.t.} &: \xi_i \geq 0.
\end{aligned}
$$

where $C > 0$ denotes the **penalty parameter** (or cost parameter).

Interpretation

Interpretation

$$
\begin{aligned}
(\omega^*, b^*, \xi^*) \quad &= \quad \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \quad \underbrace{\tfrac{1}{2}\|\omega\|^2}_{\text{margin}} + \underbrace{C}_{\text{cost parameter}} \underbrace{\sum_{i=1}^{n} \xi_i}_{\text{classification errors}} \\
\text{s.t.} \quad &: \quad y_i\big(\langle \omega, x_i \rangle + b\big) \geq 1 - \xi_i, \qquad \forall i = 1, \ldots, n \\
\text{s.t.} \quad &: \quad \xi_i \geq 0.
\end{aligned}
$$

Penalty Parameter

Role of the Penalty Parameter C

The penalty parameter C controls the trade-off between the margin size and the training error rate.

- If C is small, misclassification errors are weakly penalized and the focus is on maximizing the margin. This may lead to underfitting.
- If C is large, the focus is on avoiding misclassification at the cost of a smaller margin. This may lead to overfitting.

Penalty Parameter in `scikit-learn`

Connection with scikit-learn

In `scikit-learn`, the penalty parameter is also denoted C in `SVC` (Support Vector Classifier): it controls the balance between maximizing the margin and minimizing classification errors.

- Small C: larger margin, tolerance to errors, possible underfitting.
- Large C: smaller margin, less tolerance, possible overfitting.

The appropriate value of C should be determined by **cross-validation**.

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. For an intuitive visualization of the effects of scaling the regularization parameter C, see Scaling the regularization parameter for SVCs.

Example: Soft Margin

Example: SVM Soft Margin

```
# Without CV (fixed C)
clf_fixed = SVC(kernel="linear", C=1.0)
clf_fixed.fit(X_train, y_train)

# With CV (grid search for best C)
C_grid = np.logspace(-3, 3, 13)
grid = GridSearchCV(
    SVC(kernel="linear"),
    param_grid={"C": C_grid},
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
grid.fit(X_train, y_train)
```

Google Colab: Click here.

Dual Problem with Soft Margin

Definition: Dual Problem with Soft Margin

The **dual program** of the soft-margin SVM is

$$\lambda^* = \underset{\lambda_1,\ldots,\lambda_n}{\arg\max} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{s.t.} \quad 0 \leq \lambda_i \leq C \quad (i = 1, \ldots, n), \qquad \sum_{i=1}^{n} \lambda_i y_i = 0,$$

where $C > 0$ is the penalty (cost) parameter.

Remark: The program is identical to the separable case: the only difference is the upper bound on the Lagrange multipliers λ_i.

Proof: see appendix.

Interpretation of the Dual Coefficients

Configurations According to the Dual Coefficients λ_i

Three configurations can occur depending on the value of λ_i:

1. If $\lambda_i = 0$, then
$$y_i\big(\langle \omega, x_i \rangle + b\big) > 1, \qquad \mu_i = C > 0, \qquad \xi_i = 0.$$
The observation (x_i, y_i) is correctly classified.

2. If $0 < \lambda_i < C$, then
$$y_i\big(\langle \omega, x_i \rangle + b\big) = 1, \qquad \mu_i = C - \lambda_i > 0, \qquad \xi_i = 0.$$
The observation (x_i, y_i) is a support vector.

3. If $\lambda_i = C$, then
$$y_i\big(\langle \omega, x_i \rangle + b\big) = 1 - \xi_i, \qquad \mu_i = 0, \qquad \xi_i > 0.$$
The observation (x_i, y_i) may lie correctly classified, but within the margin or misclassified, i.e., on wrong side of the hyperplane.

Soft Margin: Primal and Dual Programs

Soft-Margin Primal Program	Soft-Margin Dual Program
$(\omega^*, b^*, \xi^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \ \frac{1}{2}\|\omega\|^2 + C \sum_{i=1}^n \xi_i$	$\lambda^* = \underset{\lambda_1, \ldots, \lambda_n}{\arg\max} \ \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$
s.t. $y_i h(x_i) \geq 1 - \xi_i, \quad i = 1, \ldots, n$	s.t. $0 \leq \lambda_i \leq C, \quad i = 1, \ldots, n$
s.t. $\xi_i \geq 0, \quad i = 1, \ldots, n$	s.t. $\sum_{i=1}^n \lambda_i y_i = 0$
$h(x) = \langle \omega^*, x \rangle + b^*$	$\omega^* = \sum_{i \in S} \lambda_i^* y_i x_i$
—	$b^* = y_i - \langle \omega^*, x_i \rangle, \quad \forall i \in S$
—	$h(x) = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^*$

Penalty Parameter and Cross-Validation

Penalty Parameter and Cross-Validation

The performance of SVMs is **highly sensitive** to the choice of the penalty parameter C.

An "optimal" hyperparameter C must therefore be selected with care.

A standard approach is to rely on Cross-Validation (CV) methods, such as hold-out or k-fold validation.

Exercise: Soft-Margin SVM with Cross-Validation

Exercise: Soft-Margin SVM with Cross-Validation

Consider the **Breast Cancer Wisconsin** dataset from scikit-learn (malignant vs. benign). The goal is to choose the optimal parameter C to avoid overfitting.

- Split the data into training 70% and test 30% with stratification.
- Standardize the data (fit on training, transform train and test).
- Fit a baseline linear SVM with $C = 1.0$ on the training set.
- Evaluate on the *test set*: Accuracy, Precision, Recall, F1-score.
- Use `GridSearchCV` to tune $C \in \{10^{-4}, \ldots, 10^{4}\}$ with 5-fold CV and scoring = F1; refit the best model on the full training set.
- Re-evaluate on the *test set* the tuned model and compare with the baseline (report at least Accuracy and F1).

Note. The cross-validated score used to select C is computed on *training folds* only; the final comparison must use the *held-out test set*.

Google Colab: Click here.

Exercise: Python Code

```
# Data split
X, y = load_breast_cancer(return_X_y=True)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.3, stratify=
    y, random_state=42)

# Standardization (fit on training, transform train and test)
scaler = StandardScaler()
X_tr_s = scaler.fit_transform(X_tr)
X_te_s = scaler.transform(X_te)

# Train WITHOUT CV (fixed C)
clf_fixed = SVC(kernel="linear", C=1.0, random_state=42)
clf_fixed.fit(X_tr_s, y_tr)

# Train WITH CV to select C
C_grid = np.logspace(-4, 4, 17)  # 1e-4 ... 1e4 (all positive)
grid = GridSearchCV(
    SVC(kernel="linear", random_state=42),
    param_grid={"C": C_grid},
    cv=5,
    scoring="f1",
    n_jobs=-1,
    refit=True
)
grid.fit(X_tr_s, y_tr)
```

Exercise: Python Output

```
No CV (C=1.0) (EVALUATED ON TEST SET)
  Accuracy  : 0.982
  Precision : 0.981
  Recall    : 0.991
  F1-score  : 0.986
  Confusion matrix:
 [[ 62   2]
 [  1 106]]

With CV (best C = 0.01) (EVALUATED ON TEST SET)
  Accuracy  : 0.959
  Precision : 0.946
  Recall    : 0.991
  F1-score  : 0.968
  Confusion matrix:
 [[ 58   6]
 [  1 106]]

Summary (TEST set):
  No CV  -> Acc 0.982, F1 0.986
  With CV-> Acc 0.959, F1 0.968
```

Note: Cross-validation combats overfitting by selecting a much smaller C (wider margin, lower model complexity), which reduces variance, even if, as in this split, it slightly sacrifices test accuracy.

Soft Margin

Key Concepts

1. Soft margin
2. Non-separable sample
3. Slack variables
4. Penalty parameter C

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

Higher-Dimensional Space

General Idea of the Kernel Trick

Most classification problems involve **non-linear separations**. However, the data space can always be embedded in a higher-dimensional space in which the data may become linearly separable.

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated (Cover (1965)).



Cover, T.M. (1965), Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition, *IEEE Transactions on Electronic Computers*, 14(3), 326–334.

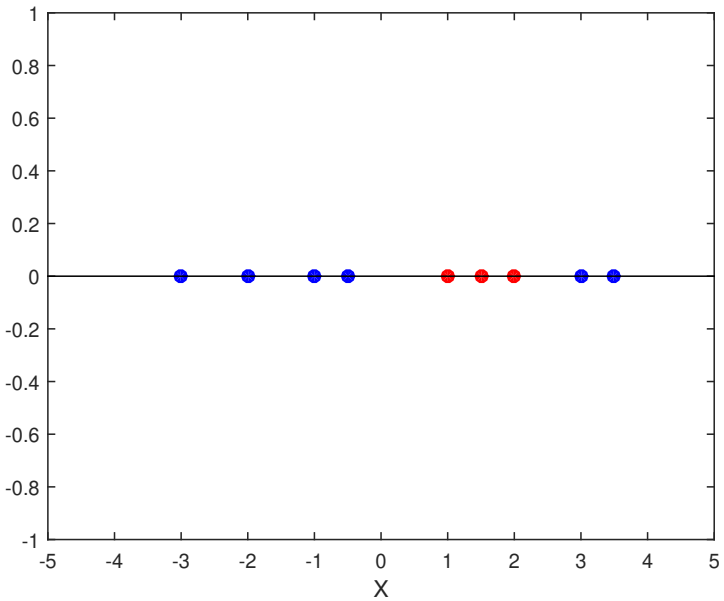
Higher-Dimensional Space

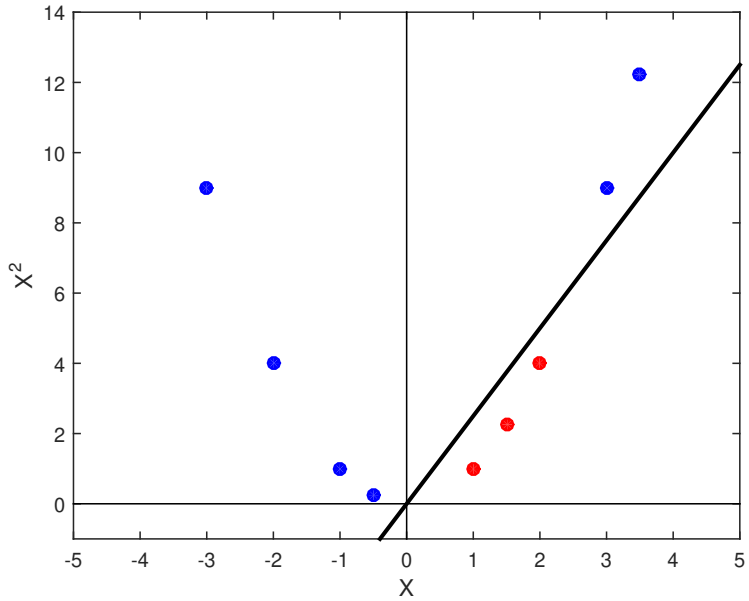
Non-linear Sample

A training sample that is **not linearly separable** in the input space...

Higher-dimensional Mapping

...can become **linearly separable** in a higher-dimensional feature space.





Kernel Trick: General Principle

Kernel Trick

The resolution of SVMs relies on the scalar product $\langle x_i, x_j \rangle$ between input vectors. If the training data are mapped into a **higher-dimensional space** via the transformation $\Phi(x)$, this Hilbert space is associated with the **inner product**

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle,$$

where $K(x_i, x_j)$ is called the **kernel function**.

To implement an SVM, only the kernel function is required, without the explicit computation of the transformation $\Phi(\cdot)$: this is the essence of the **kernel trick**.

Feature Space

Definition: Feature Space

Instead of searching for a separating hyperplane in the input space $\mathcal{X}$ (here $\mathbb{R}^d$), one first maps the data into an **intermediate representation space (feature space)** of higher dimension:

$$\Phi : \mathcal{X} \to \mathcal{F}, \qquad x \mapsto \Phi(x).$$

Example

Let $x = (x_1, x_2) \in \mathbb{R}^2$. Consider the mapping

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3, \qquad (x_1, x_2) \mapsto \Phi(x) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2).$$

Kernel Trick

Illustration

Consider the mapping

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3, \qquad (x_1, x_2) \mapsto \Phi(x) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right).$$

Instead of manipulating the original two variables x_1, x_2, one must now handle three transformed variables x_1^2, $\sqrt{2}x_1x_2$, x_2^2, which increases computational and storage costs.

Kernel Trick

In this higher-dimensional feature space, only the **inner product** is required:

$$\langle \Phi(x_i), \Phi(x_j) \rangle = x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2.$$

This simplifies to

$$\langle \Phi(x_i), \Phi(x_j) \rangle = \left(x_{i1}x_{j1} + x_{i2}x_{j2}\right)^2 = \left(\langle x_i, x_j \rangle\right)^2.$$

Hence, the inner product in the feature space can be computed without explicitly evaluating $\Phi(x)$, using the **kernel function**

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle = \left(\langle x_i, x_j \rangle\right)^2.$$

Numerical Example: Kernel Trick

Numerical Example: Kernel Trick

Let $x = (x_1, x_2) \in \mathbb{R}^2$. Consider the mapping

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^6, \qquad (x_1, x_2) \mapsto \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right).$$

Question: Determine the kernel function associated with this transformation.

Solution: For two vectors $u = (u_1, u_2)'$ and $v = (v_1, v_2)'$, we have

$$\Phi(u) = \left(1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1 u_2\right),$$

$$\Phi(v) = \left(1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1 v_2\right).$$

The corresponding kernel is

$$K(u, v) = \langle \Phi(u), \Phi(v) \rangle = 1 + 2u_1 v_1 + 2u_2 v_2 + u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 u_2 v_2,$$

which simplifies to

$$K(u, v) = \left(1 + u_1 v_1 + u_2 v_2\right)^2 = \left(1 + \langle u, v \rangle\right)^2.$$

Numerical Example: Kernel Trick

Numerical Example: Kernel Trick

Consider two vectors $u = (1, 3\sqrt{2})'$ and $v = (1, 2\sqrt{2})'$, and the mapping

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^6, \qquad (x_1, x_2) \mapsto \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right).$$

Question: Show that the kernel function

$$K(u, v) = \left(1 + \langle u, v \rangle\right)^2$$

is associated with the transformation $\Phi(x)$ for these two vectors.

Numerical Example: Kernel Trick

Solution via Feature Mapping:

Using the transformation

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^6, \qquad (x_1, x_2) \mapsto \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right),$$

we obtain for $u = (1, 3\sqrt{2})'$ and $v = (1, 2\sqrt{2})'$:

$$\Phi(u) = (1, \sqrt{2}, 6, 1, 18, 6), \qquad \Phi(v) = (1, \sqrt{2}, 4, 1, 8, 4).$$

Therefore,

$$\langle \Phi(u), \Phi(v) \rangle = 1 \times 1 + \sqrt{2} \times \sqrt{2} + 6 \times 4 + 1 \times 1 + 18 \times 8 + 6 \times 4 = 196.$$

Numerical Example: Kernel Trick

Solution via Kernel Function:

Using the kernel

$$K(u, v) = (1 + \langle u, v \rangle)^2,$$

we compute

$$\langle u, v \rangle = 1 \times 1 + 3\sqrt{2} \times 2\sqrt{2} = 13,$$

which gives

$$K(u, v) = (1 + 13)^2 = 196.$$

Hence,

$$K(u, v) = \langle \Phi(u), \Phi(v) \rangle,$$

which confirms the equivalence between the explicit mapping and the kernel function.

Kernel Function: Definition

Definition: Kernel Function

A **kernel function** represents the inner product associated with the feature space. For a kernel $K(x_i, x_j)$, there exists a Hilbert space $\mathcal{F}$ and a mapping $\Phi(\cdot)$ such that

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle \iff \Phi : \mathbb{R}^d \to \mathbb{R}^m, \quad m > d, \quad x \mapsto \Phi(x).$$

In practice, the **kernel trick** consists of selecting an appropriate kernel function $K(\cdot, \cdot)$ without explicitly characterizing the space $\mathcal{F}$ or the mapping $\Phi(\cdot)$.

Procedure

Procedure

1. **Transform the space** of the input data into a higher-dimensional feature space (possibly infinite-dimensional).
2. In this space, it is more likely that a linear separation exists.
3. Determine the **maximum-margin canonical hyperplane** (SVM), with or without slack variables (soft margin).

Primal Program with Feature Mapping

Definition: Primal Program with Feature Mapping

The **primal SVM problem with soft margin and feature mapping** is defined as:

$$
\begin{aligned}
(\omega^*, b^*, \xi^*) \quad &= \quad \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \quad \frac{1}{2}\|\omega\|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.} \quad &: \quad y_i \Big(\langle \omega, \Phi(x_i) \rangle + b \Big) \geq 1 - \xi_i, \qquad \forall i = 1, \ldots, n \\
\text{s.t.} \quad &: \quad \xi_i \geq 0.
\end{aligned}
$$

where $C > 0$ is the penalty parameter and $\Phi : \mathcal{X} \to \mathcal{F}$ is the feature mapping.

Difference between Primal and Dual Problems

Difference between Primal and Dual Problems

The primal program requires the explicit use (and knowledge) of the mapping $\Phi(x)$. By contrast, the dual program only involves the kernel function, which is the essence of the kernel trick.

$$
\begin{aligned}
(\omega^*, b^*, \xi^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \quad & \tfrac{1}{2}\|\omega\|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.} \quad & y_i\big(\langle \omega, \Phi(x_i) \rangle + b\big) \geq 1 - \xi_i, \qquad i = 1, \ldots, n, \\
& \text{s.t.} \quad \xi_i \geq 0.
\end{aligned}
$$

Dual Problem with Feature Mapping

Definition: Dual Problem with Feature Mapping

The **dual SVM problem with soft margin and kernel trick** is defined as

$$\lambda^* = \underset{\lambda_1,\ldots,\lambda_n}{\arg\max} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \; K(x_i, x_j)$$

$$\text{s.t.} \quad 0 \leq \lambda_i \leq C, \qquad \sum_{i=1}^{n} \lambda_i y_i = 0.$$

Here $C > 0$ is the penalty parameter, and $K(x_i, x_j)$ denotes the kernel function associated with the (unknown) mapping $\Phi : \mathcal{X} \to \mathcal{F}$.

Optimal Decision Function

Optimal Decision Function

For any point $x \in \mathcal{X}$, the **optimal decision function** is

$$h(x) = \langle \omega^*, \Phi(x) \rangle + b^* = \sum_{i \in S} \lambda_i^* y_i K(x_i, x) + b^*,$$

where S denotes the set of support vectors and slack variables.

The associated classifier is

$$g(x) = \operatorname{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0. \end{cases}$$

Note: the constant b^* is defined as:

$$b^* = y_j - \sum_{i \in S} \lambda_i^* y_i K(x_i, x_j), \qquad \forall j \in S,$$

Kernel Trick: Primal and Dual Programs

Primal Program with Feature Mapping	Dual Program with Kernel Trick
$(\omega^*, b^*, \xi^*) = \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^n}{\arg\min} \ \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^n \xi_i$	$\lambda^* = \underset{\lambda_1, \ldots, \lambda_n}{\arg\max} \ \sum_{i=1}^n \lambda_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j K(x_i, x_j)$
s.t. $y_i h(\Phi(x_i)) \geq 1 - \xi_i, \quad i = 1, \ldots, n$	s.t. $0 \leq \lambda_i \leq C, \quad i = 1, \ldots, n$
s.t. $\xi_i \geq 0, \quad i = 1, \ldots, n$	s.t. $\sum_{i=1}^n \lambda_i y_i = 0$
$h(x) = \langle \omega^*, \Phi(x) \rangle + b^*$	$h(x) = \sum_{i \in S} \lambda_i^* y_i K(x_i, x) + b^*$
—	$b^* = y_j - \sum_{i \in S} \lambda_i^* y_i K(x_i, x_j), \quad \forall j \in S$
—	$\omega^* = \sum_{i \in S} \lambda_i^* y_i \Phi(x_i)$

scikit-learn Documentation for SVC

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by $\text{sign}(w^T \phi(x) + b)$ is correct for most samples.

SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$
$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, \ldots, n$$

Intuitively, we're trying to maximize the margin (by minimizing $||w||^2 = w^T w$), while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i(w^T \phi(x_i) + b)$ would be ≥ 1 for all samples, which indicates a perfect prediction. But problems are usually not always perfectly separable with a hyperplane, so we allow some samples to be at a distance ζ_i from their correct margin boundary. The penalty term `C` controls the strength of this penalty, and as a result, acts as an inverse regularization parameter (see note below).

Source: scikit-learn support

scikit-learn Documentation for SVC

The dual problem to the primal is

$$\begin{aligned} \min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to } y^T\alpha = 0 \\ 0 \leq \alpha_i \leq C, i = 1, \ldots, n \end{aligned}$$

where e is the vector of all ones, and Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. The terms α_i are called the dual coefficients, and they are upper-bounded by C. This dual representation highlights the fact that training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ: see kernel trick.

Source: scikit-learn support

`scikit-learn` Documentation for `SVC`

Once the optimization problem is solved, the output of decision_function for a given sample x becomes:

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b,$$

and the predicted class corresponds to its sign. We only need to sum over the support vectors (i.e. the samples that lie within the margin) because the dual coefficients α_i are zero for the other samples.

These parameters can be accessed through the attributes `dual_coef_` which holds the product $y_i \alpha_i$, `support_vectors_` which holds the support vectors, and `intercept_` which holds the independent term b.

Source: scikit-learn support

Mercer's Conditions

Definition: Mercer's Conditions

A continuous, symmetric, and positive function $K(\cdot, \cdot)$ is a **kernel function** if, for all possible $x_i \in \mathcal{X}$, the Gram matrix

$$\left(K(x_i, x_j)\right)_{i,j}$$

is symmetric and positive semi-definite.

In this case, there exists a Hilbert space $\mathcal{F}$ and a mapping Φ such that

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle.$$

Gram Matrix

Gram Matrix and Kernel

The **Gram matrix** associated with the kernel $K(\cdot, \cdot)$ is

$$\Omega = \begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \cdots & K(x_n, x_n) \end{pmatrix}_{n \times n}.$$

By Mercer's conditions, this matrix must be symmetric and positive semi-definite.

Common Kernels

Common Kernels

Linear Kernel (retrieves the case $\mathcal{F} = \mathcal{X}$):

$$K(x_i, x_j) = \langle x_i, x_j \rangle.$$

Polynomial Kernel of degree p (hyperparameters: θ_0, p):

$$K(x_i, x_j) = \left(\theta_0 + \langle x_i, x_j \rangle\right)^p.$$

Radial Basis Function (RBF) or Gaussian Kernel (hyperparameter: σ):

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right).$$

Sigmoid Kernel (two-layer perceptron) (hyperparameters: θ_1, θ_2):

$$K(x_i, x_j) = \tanh\left(\theta_1 \langle x_i, x_j \rangle + \theta_2\right),$$

where $\tanh(\cdot)$ is the hyperbolic tangent function.

Construction of Kernel Functions

Construction of Kernel Functions

Beyond the standard kernels, new kernel functions **can be constructed** using the following properties.

Properties: If $K_1(x, y)$ and $K_2(x, y)$ are kernel functions, and $\alpha \in \mathbb{R}^+$, then the following functions are also valid kernels:

$$K(x, y) = K_1(x, y) + K_2(x, y),$$

$$K(x, y) = \alpha\, K_1(x, y),$$

$$K(x, y) = \langle K_1(x, y), K_2(x, y) \rangle,$$

$$K(x, y) = xAy^\top,$$

where A is a symmetric positive semi-definite matrix.

Example: SVM with non-linear Kernel

Example: SVM with Non-Linear Kernels

```
# Nonlinear SVMs
svm_rbf  = SVC(kernel="rbf",  C=1.0, gamma=0.2, random_state=0)
svm_rbf.fit(X_tr_s, y_tr)

svm_poly = SVC(kernel="poly", degree=3, gamma="scale", coef0
    =1.0, C=1.0, random_state=0)
svm_poly.fit(X_tr_s, y_tr)
```

Google Colab: Click here.

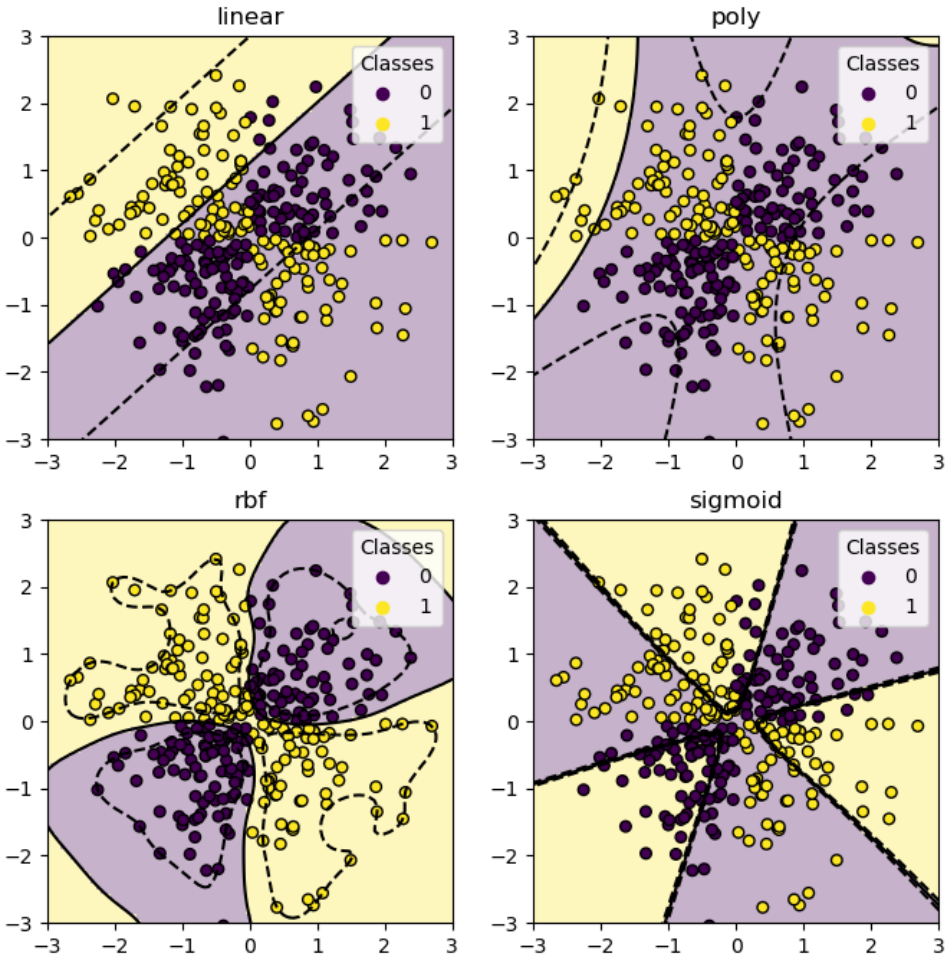
Kernels Available with SVC

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`. For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.

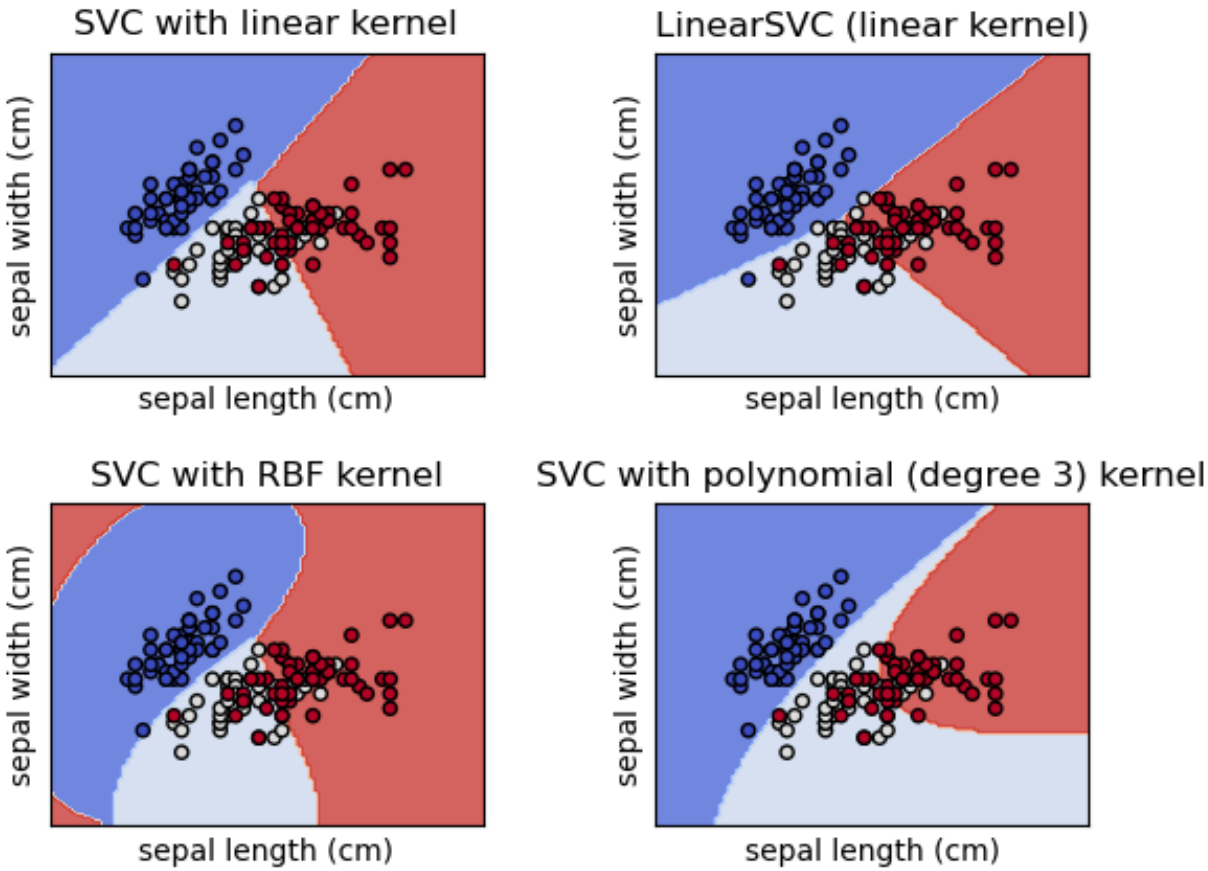
Source: scikit-learn documentation

Comparison of Kernels: Illustration



A classical example of a dataset which is not linearly separable is the XOR dataset. Source: scikit-learn support

Comparison of Kernels: Illustration



Comparison of different linear SVM classifiers on a 2D projection of the iris dataset. Source: scikit-learn support

SVM Procedure with Kernel Functions

SVM Procedure with Kernel Functions

The SVM procedure with a feature space transformation proceeds as follows:

1. Selection of the kernel function.
2. Selection of hyperparameters and the penalty parameter C using training data via cross-validation.
3. Evaluation of predictive performance on a test dataset.

Caution: The predictive performance of SVMs is **highly sensitive** to the choice of kernel, the value of the penalty parameter C, and the hyperparameters of the kernel function.

Exercise: Credit Scoring with Linear and Nonlinear SVM

Exercise: Credit Scoring with Linear and Nonlinear SVM

Consider the credit scoring dataset of borrower and loan characteristics stored in `scoring_data.xlsx`.

- Define the target $y =$ `Default` and the features X.
- Standardize all predictors using the training data statistics (zero mean, unit variance), and apply the same transformation to the test data.
- Split the data into training 70% and test 30% with stratification.
- Train support vector machines with three kernels: linear (baseline), radial basis function (RBF), and polynomial.
- Select hyperparameters by k-fold cross validation on the training set only: $C \in \{10^{-3}, \ldots, 10^{3}\}$; for RBF also $\gamma \in \{10^{-3}, \ldots, 10^{1}\}$; for polynomial also $\deg \in \{2, 3\}$, $\gamma \in \{10^{-3}, \ldots, 1\}$, and $\theta_0 \in \{0, 1\}$.
- Refit each best model on the full training set and evaluate on the held out test set using Accuracy, Precision, Recall, F1 score, and the confusion matrix; compare the three kernels.

Google Colab: Click here.

Exercise: Python Output

```
Best hyperparameters (CV on train) and TEST metrics:
Kernel    C gamma degree coef0 CV F1 (mean) Test Acc Test Prec Test Rec Test F1
   RBF  100   0.1      -     -         0.337    0.770     0.367    0.321   0.343
  Poly   10     1      3     1         0.408    0.740     0.323    0.357   0.339
Linear 0.01     -      -     -         0.187    0.810     0.429    0.054   0.095

Best parameter sets by kernel:
  RBF: C=100, gamma=0.1
  Poly: C=10, gamma=1, degree=3, coef0=1
  Linear: C=0.01
```

Note: the RBF SVM ($C = 100, \gamma = 0.1$) gives the best test $F1$ (0.343) and higher recall, while the linear SVM posts higher accuracy (0.810) by missing most defaults (recall 0.054), showing why $F1$/recall, not accuracy, should guide model choice under class imbalance.

Kernel Trick

Key Concepts

1. Non-linearly separable sample
2. Feature space (intermediate representation)
3. Kernel function
4. Primal and dual programs with kernels
5. Mercer's conditions
6. Common kernel functions

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

SVM Variants

We consider four **SVM variants**:

1. Probabilities and SVM decision scores
2. Multi-class SVM
3. Support Vector Regression (SVR)
4. Least Squares SVM (LS-SVM)

SVM and Decision Scores

Two main questions:

- How can we compute class membership **probabilities**?
- How can the raw **SVM output** be transformed into probabilities?

For any observation $x \in \mathcal{X}$ (training, test, or new sample), the SVM output is a classification:

$$\widehat{y} = g(x) = \text{sgn}\big(h(x)\big) = \begin{cases} 1 & \text{if } h(x) \geq 0, \\ -1 & \text{if } h(x) < 0. \end{cases}$$

with the optimal decision function

$$h(x) = \langle \omega^*, x \rangle + b^* = \sum_{i \in S} \lambda_i^* y_i \langle x_i, x \rangle + b^*,$$

where S denotes the set of support vectors.

Platt Scaling

Definition: Platt Scaling

Platt's method (Platt scaling) is a post-processing technique that transforms the output of a classification model into a probability distribution over the classes.

Implementation:

- Platt's method consists of using a parametric or non-parametric function to map the values of $h(x)$ into the interval $[0, 1]$.
- Any cumulative distribution function (CDF) can be used for this mapping.

Plat Scaling

Examples

- **Logistic distribution (simple sigmoid):**

$$\Pr(y_i = 1 \mid x) = \frac{1}{1 + \exp\big(- h(x_)\big)}, \qquad \forall i.$$

- **Logit model:** estimate by MLE the parameters (θ_1, θ_2) such that

$$\Pr(y_i = 1 \mid x) = \frac{1}{1 + \exp\big(- (\theta_1 + \theta_2 h(x_i))\big)}, \qquad \forall i.$$

SVC Probabilities with `scikit-learn`

probability : *bool, default=False*

Whether to enable probability estimates. This must be enabled prior to calling `fit`, will slow down that method as it internally uses 5-fold cross-validation, and `predict_proba` may be inconsistent with `predict`. Read more in the User Guide.

1.4.1.2. Scores and probabilities

The `decision_function` method of `SVC` and `NuSVC` gives per-class scores for each sample (or a single score per sample in the binary case). When the constructor option `probability` is set to `True`, class membership probability estimates (from the methods `predict_proba` and `predict_log_proba`) are enabled. In the binary case, the probabilities are calibrated using Platt scaling [9]: logistic regression on the SVM's scores, fit by an additional cross-validation on the training data. In the multiclass case, this is extended as per [10].

Source: scikit-learn documentation

Example: SVC Decision score and Probabilities

Example: SVC Decision score and Probabilities

```
# Train SVC with probability=True
svc = SVC(probability=True, random_state=42)
svc.fit(X_train, y_train)

# Print probabilities
probas = svc.predict_proba(X_test)
print("Probabilities:")
print(probas)

```

SVM Variants

We consider four **SVM variants**:

1. Probabilities and SVM decision scores
2. Multi-class SVM
3. Support Vector Regression (SVR)
4. Least Squares SVM (LS-SVM)

Multi-class SVM

Multi-class SVM

- SVMs can be adapted to handle multi-class classification problems.
- The discrete outcome variable y has k categories, with $y \in \{m_1, \ldots, m_k\}$.
- Two main approaches exist:
  1. One-vs-Rest (OvR) approach.
  2. One-vs-One (OvO) or pairwise approach.

One-vs-Rest Approach

One-vs-Rest (OvR) Approach

- The idea is to transform the k-class problem into k binary classifiers.
- Construct k binary models for dichotomous outcomes y_j:

$$y_j = \begin{cases} 1 & \text{if } y = m_j, \\ -1 & \text{otherwise.} \end{cases}$$

- We obtain k decision functions $h_j(x)$. The predicted class corresponds to the one with the highest score:

$$\widehat{y} = m_c \quad \text{with } c = \underset{j=1,\ldots,k}{\arg\max}\ h_j(x).$$

One-vs-One Approach

One-vs-One (OvO) Approach

- Build $k(k-1)/2$ binary SVM models, one for each pair of classes. For example:

$$y_{ij} = \begin{cases} 1 & \text{if } y = m_i, \\ -1 & \text{if } y = m_j. \end{cases}$$

- This gives $k(k-1)/2$ decision functions $h_{ij}(x)$. Classification is determined by majority vote (or another voting rule).
- Let $D_j(x)$ denote the number of votes for class m_j. The final prediction is

$$\widehat{y} = m_c \quad \text{with } c = \underset{j=1,\ldots,k}{\arg\max}\, D_j(x).$$

- The vote count is computed as

$$D_j(x) = \sum_{i \neq j,\ i=1}^{k} \operatorname{sgn}\big(h_{ij}(x)\big).$$

Example: Multi-Class SVM

Example: Multi-Class SVM

```
1 # Train multiclass SVC
2 svc = SVC(kernel='rbf', C=1.0, decision_function_shape='ovr',
      probability=True)
3 svc.fit(X_train, y_train)
4
```

decision_function_shape : *{'ovo', 'ovr'}, default='ovr'*

Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one ('ovo') is always used as a multi-class strategy to train models; an ovr matrix is only constructed from the ovo matrix. The parameter is ignored for binary classification.

Source: scikit-learn documentation

SVM Variants

We consider four **SVM variants**:

1. Probabilities and SVM decision scores
2. Multi-class SVM
3. Support Vector Regression (SVR)
4. Least Squares SVM (LS-SVM)

Support Vector Regression

We now consider the **regression setting**.

- **Data:** $\{(x_i, y_i)\}_{i=1,\dots,n}$ with $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.
- **Predictor:** for $\beta \in \mathbb{R}^d$ and $b \in \mathbb{R}$,
$$h(x) = x^\top \beta + b.$$
- **Norm:**
$$\|h\| = \|\beta\| = \sqrt{\beta^\top \beta}.$$
- **Criterion:** least squares,
$$\sum_{i=1}^{n} \left(y_i - h(x_i)\right)^2.$$

Support Vector Regression

Definition: Support Vector Regression

The idea of **Support Vector Regression (SVR)** is to impose a constraint on the L_2-norm of the weights:

$$\begin{aligned} (\widehat{\beta}, \widehat{b}) &= \underset{\beta \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} \left(y_i - h(x_i)\right)^2 \\ \text{s.t.} \quad &: \quad \|h\| \leq \lambda, \end{aligned}$$

where λ is a regularization parameter.

SVR and Ridge Regression

SVR and Ridge Regression

Support Vector Regression (SVR) can equivalently be expressed as a **ridge regression** problem:

$$(\widehat{\beta}, \widehat{b}) = \underset{\beta \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} \left(y_i - h(x_i)\right)^2 + C\|h\|^2,$$

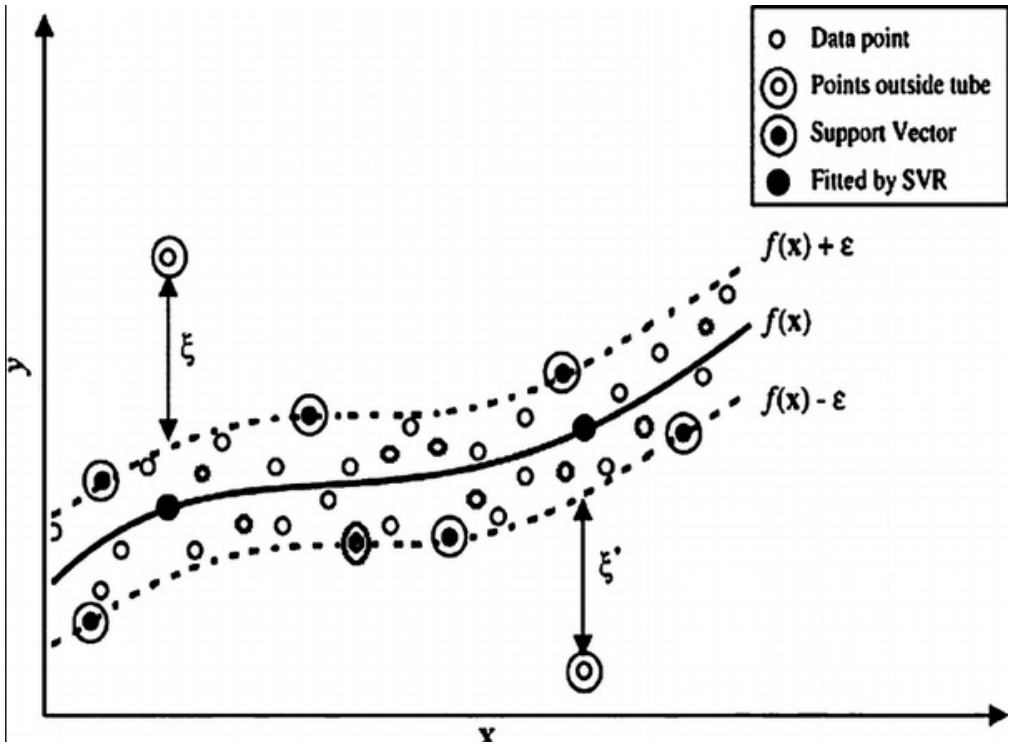
where $C > 0$ is the penalty parameter.

The solution takes the form

$$\widehat{\beta} = (X^\top X + C I_d)^{-1} X^\top Y,$$

with $X = (x_1, \ldots, x_n)^\top$ and $Y = (y_1, \ldots, y_n)^\top$.

7.3. Support Vector Regression (SVR)



Source: www.quora.com

Support Vector Regression

- We seek a **predictor** of the form

$$h(x) = \sum_{i=1}^{n} \alpha_i \, K(x_i, x),$$

where $K(x_i, x)$ is a kernel function.

- Let $\Omega = \big(K(x_i, x_j)\big)_{i,j}$ be the Gram matrix. We define the kernel-induced norm as

$$\|h\|_K^2 = \alpha^\top \Omega \alpha,$$

with $\alpha = (\alpha_1, \ldots, \alpha_n)^\top$.

SVR in Dual Form

Definition: SVR in Dual Form

The **dual problem** of the Support Vector Regression (SVR) can be written as

$$\widehat{\alpha} = \underset{\alpha \in \mathbb{R}^n}{\arg\min} \sum_{i=1}^{n} \left(y_i - \Omega\alpha\right)^2 + C\,\alpha^\top \Omega\alpha,$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)^\top$ and $\Omega = \left(K(x_i, x_j)\right)_{i,j}$ is the Gram matrix associated with the kernel $K(\cdot, \cdot)$. The solution is

$$\widehat{\alpha} = \left(\Omega + C I_n\right)^{-1} Y,$$

where $Y = (y_1, \ldots, y_n)^\top$.

SVR Implementation with `scikit-learn`

1.4.7.2. SVR

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, and a vector $y \in \mathbb{R}^n$ ε-SVR solves the following primal problem:

$$\begin{aligned} \min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ \text{subject to } y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ \zeta_i, \zeta_i^* \geq 0, i = 1, \ldots, n \end{aligned}$$

Here, we are penalizing samples whose prediction is at least ε away from their true target. These samples penalize the objective by ζ_i or ζ_i^*, depending on whether their predictions lie above or below the ε tube.

The dual problem is

$$\begin{aligned} \min_{\alpha,\alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon e^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*) \\ \text{subject to } e^T (\alpha - \alpha^*) = 0 \\ 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \ldots, n \end{aligned}$$

where e is the vector of all ones, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

Source: scikit-learn documentation

Support Vector Regression with `scikit-learn`

SVR

```
class sklearn.svm.SVR(*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001,
C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)
```

Epsilon-Support Vector Regression.

The free parameters in the model are C and epsilon.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to datasets with more than a couple of 10000 samples. For large datasets consider using `LinearSVR` or `SGDRegressor` instead, possibly after a `Nystroem` transformer or other Kernel Approximation.

Read more in the User Guide.

Source: scikit-learn documentation

Example: Support Vector Regression

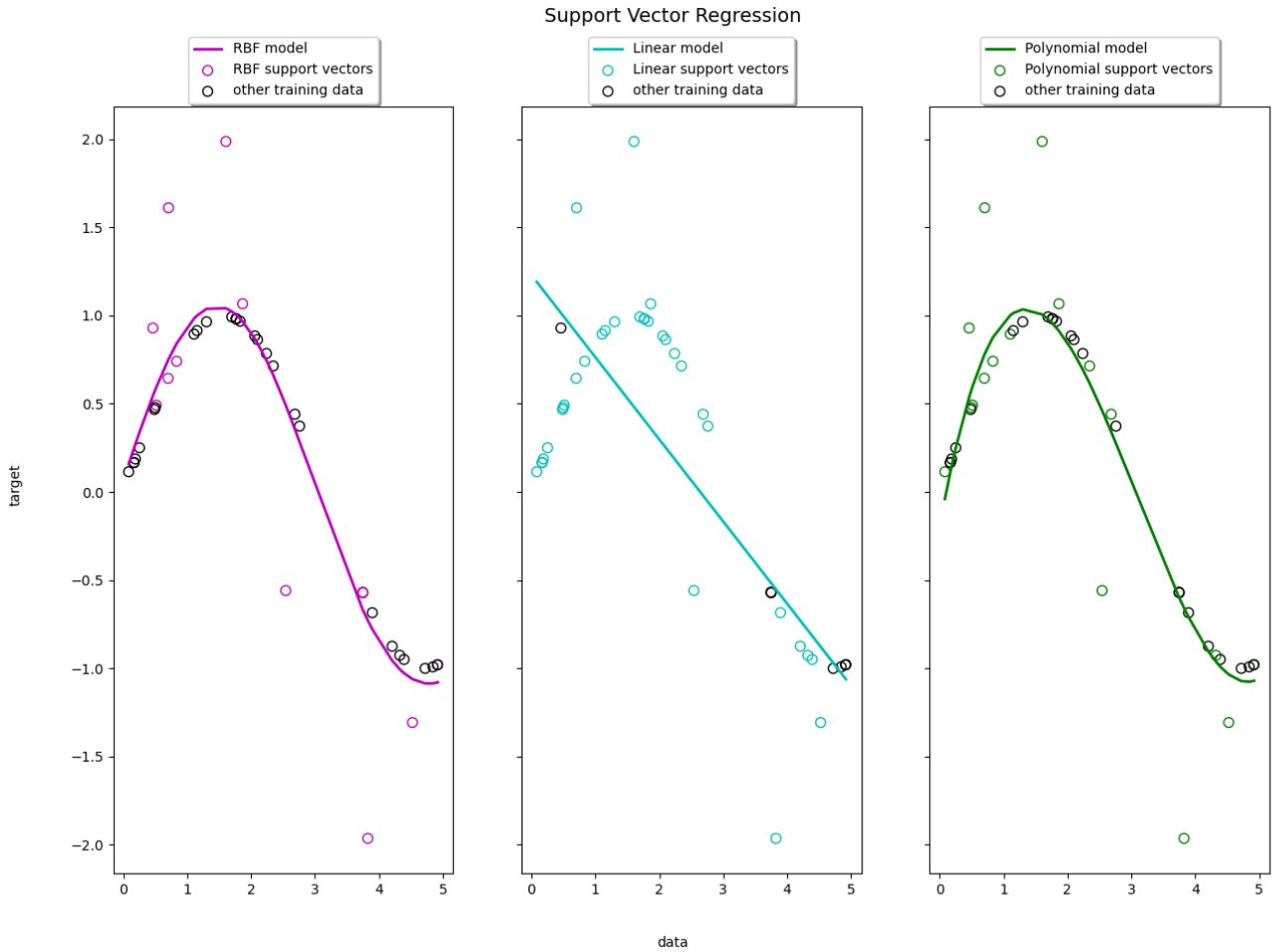
Example: Support Vector Regression

```
# Create and train SVR
svr = SVR(kernel='rbf',     # RBF kernel
          C=100,            # Regularization parameter
          epsilon=0.1,      # Epsilon-tube
          gamma='scale')    # RBF kernel parameter

svr.fit(X_train, y_train)

# Predictions
y_pred = svr.predict(X_test)
```

Comparison of Kernels: Illustration



Toy example of 1D regression using linear, polynomial and RBF kernels. Source: scikit-learn documentation

SVM Variants

We consider four **SVM variants**:

1. Probabilities and SVM decision scores
2. Multi-class SVM
3. Support Vector Regression (SVR)
4. Least Squares SVM (LS-SVM)

Least Squares Support Vector Machine

LS-SVM: General Idea

- The LS-SVM and its regression counterpart LS-SVR were introduced by Suykens and Vandewalle (1999) and Suykens et al. (2002).
- The method is computationally efficient: instead of solving a quadratic programming problem, it reduces to solving a linear system of equations.


Suykens, J. and Vandewalle, J. (1999). Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3), 293–300.


Suykens, J., Van Gestel, T., De Brabanter, J., De Moor, B. and Vandewalle, J. (2002). *Least Squares Support Vector Machine*. Singapore: World Scientific.

Least Squares Support Vector Machine

Definition: LS-SVM

The **primal problem of LS-SVM** is defined as:

$$
\begin{aligned}
(\omega^*, b^*) \quad &= \quad \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \quad \frac{C_1}{2} \|\omega\|^2 \; + \; \frac{C_2}{2} \sum_{i=1}^{n} \xi_i^2 \\
\text{s.t.} \quad &: \quad y_i \big(\langle \omega, \Phi(x_i) \rangle + b \big) \; \geq \; 1 - \xi_i, \quad i = 1, \ldots, n, \\
\text{s.t.} \quad &: \quad \xi_i \; \geq \; 0.
\end{aligned}
$$

where $(C_1, C_2) \in \mathbb{R}^+ \times \mathbb{R}^+$ are penalty parameters. This specification can be interpreted as a regression with a binary dependent variable $y \in \{-1, 1\}$.

Least Squares Support Vector Machine

Proof: Using the fact that $y^2 = 1$, the LS-SVM program

$$\begin{aligned}
(\omega^*, b^*) &= \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \ \frac{C_1}{2}\|\omega\|^2 + \frac{C_2}{2}\sum_{i=1}^{n}\xi_i^2 \\
\text{s.t.} &: \ y_i\big(\langle \omega, \Phi(x_i)\rangle + b\big) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \\
\text{s.t.} &: \ \xi_i \geq 0.
\end{aligned}$$

can be rewritten as:

$$\begin{aligned}
(\omega^*, b^*) &= \underset{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}}{\arg\min} \ \frac{C_1}{2}\|\omega\|^2 + \frac{C_2}{2}\sum_{i=1}^{n}\Big[y_i - \big(\langle \omega, \Phi(x_i)\rangle + b\big)\Big]^2 \\
\text{s.t.} &: \ y_i\big(\langle \omega, \Phi(x_i)\rangle + b\big) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \\
\text{s.t.} &: \ \xi_i \geq 0.
\end{aligned}$$

Solution of LS-SVM

Solution of LS-SVM

From the first-order conditions, solving LS-SVM reduces to solving the linear system:

$$\begin{pmatrix} 0 & e_n^\top \\ e_n & \Omega + \gamma^{-1} I_n \end{pmatrix} \begin{pmatrix} \omega^* \\ b^* \end{pmatrix} = \begin{pmatrix} 0 \\ Y \end{pmatrix}$$

where e_n is the unit vector, I_n is the $n \times n$ identity matrix, Ω is the Gram matrix, $Y = (y_1, \ldots, y_n)^\top$, and $\gamma = C_2/C_1$.

Main advantage: LS-SVM replaces quadratic programming by a linear system.

Applications of LS-SVR in Risk Management

Several studies have shown the strong performance of LS-SVR for modeling LGD (Loss Given Default):



Loterman, G., Brown, I., Martens, D., Mues, C. and Baesens, B. (2012). Benchmarking Regression Algorithms for Loss Given Default Modeling. *International Journal of Forecasting*, 28(1), 161–170.



Nazemi, A., Fatemi Pour, F., Heidenreich, K. and Fabozzi, F. J. (2017). Fuzzy Decision Fusion Approach for Loss-Given-Default Modeling. *European Journal of Operational Research*, 262(2), 780–791.



Yao, X., Crook, J. and Andreeva, G. (2015). Support Vector Regression for Loss Given Default Modelling. *European Journal of Operational Research*, 240(2), 528–538.

SVM Variants

Key Concepts

1. SVM Decision Scores & Probabilities
2. Platt scaling
3. One-vs-Rest (OvR)
4. One-vs-One (OvO)
5. Support Vector Regression (SVR)
6. Ridge regression connection

Outline

1. Introduction

2. SVM Intuition

3. Formalization of the Support Vector Machine

4. Soft Margin

5. Kernel Trick

6. SVM Variants

7. Appendix

Appendix A: Derivation of the SVM Dual Program

Appendix A

Derivation of the SVM Dual Program

Appendix A: Lagrangian

Lagrangian

The **Lagrangian** associated with the SVM program is

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{n} \lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b \big) - 1 \Big],$$

with $\lambda = (\lambda_1, \ldots, \lambda_n)'$ the vector of multipliers satisfying, for all $i = 1, \ldots, n$,

$$\lambda_i \geq 0$$

$$\lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b \big) - 1 \Big] = 0.$$

Appendix A: Karush-Kuhn-Tucker Conditions

Karush-Kuhn-Tucker (KKT) Conditions

The **Karush-Kuhn-Tucker conditions** for the SVM primal are:

$$\frac{\partial \mathcal{L}(\omega, b, \lambda)}{\partial \omega} = \omega - \sum_{i=1}^{n} \lambda_i y_i x_i = 0$$

$$\frac{\partial \mathcal{L}(\omega, b, \lambda)}{\partial b} = -\sum_{i=1}^{n} \lambda_i y_i = 0$$

$$\lambda_i \left(y_i \, h(x_i) - 1 \right) = \lambda_i \left[y_i \left(\langle \omega, x_i \rangle + b \right) - 1 \right] = 0$$

$$\lambda_i \geq 0, \qquad \forall \, i = 1, \ldots, n.$$

Appendix A: Derivation of the SVM Dual Program

Reminder

For a column vector $\omega \in \mathbb{R}^d$:

$$\frac{\partial (\omega^\top \omega)}{\partial \omega} = \frac{\partial}{\partial \omega}\left(\sum_{j=1}^{d} \omega_j^2\right) = \begin{pmatrix} \frac{\partial}{\partial \omega_1} \sum_{j=1}^{d} \omega_j^2 \\ \vdots \\ \frac{\partial}{\partial \omega_d} \sum_{j=1}^{d} \omega_j^2 \end{pmatrix} = 2 \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_d \end{pmatrix} = 2\,\omega.$$

Similarly, for a column vector $x_i \in \mathbb{R}^d$:

$$\frac{\partial (x_i^\top \omega)}{\partial \omega} = \frac{\partial}{\partial \omega}\left(\sum_{j=1}^{d} x_{i,j}\,\omega_j\right) = \begin{pmatrix} \frac{\partial}{\partial \omega_1} \sum_{j=1}^{d} x_{i,j}\,\omega_j \\ \vdots \\ \frac{\partial}{\partial \omega_d} \sum_{j=1}^{d} x_{i,j}\,\omega_j \end{pmatrix} = \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,d} \end{pmatrix} = x_i.$$

Appendix A: Derivation of the SVM Dual Program

Introducing the KKT conditions yields an optimization that depends only on the multipliers:

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\langle \omega, \omega \rangle - \sum_{i=1}^{n} \lambda_i y_i \langle \omega, x_i \rangle - b \sum_{i=1}^{n} \lambda_i y_i + \sum_{i=1}^{n} \lambda_i.$$

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\langle \omega, \omega \rangle - \sum_{i=1}^{n} \lambda_i y_i \langle \omega, x_i \rangle + \sum_{i=1}^{n} \lambda_i, \qquad \text{since } \sum_{i=1}^{n} \lambda_i y_i = 0.$$

Appendix A: Derivation of the SVM Dual Program

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2}\langle \omega, \omega \rangle - \sum_{i=1}^{n} \lambda_i y_i \langle \omega, x_i \rangle + \sum_{i=1}^{n} \lambda_i.$$

Moreover, since $\omega = \sum_{i=1}^{n} \lambda_i y_i x_i$, we obtain:

$$\mathcal{L}(\omega, b, \lambda) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle - \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=1}^{n} \lambda_i.$$

$$\mathcal{L}(\omega, b, \lambda) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=1}^{n} \lambda_i.$$

Appendix A: Derivation of the SVM Dual Program

Indeed:

$$\sum_{i=1}^{n} \lambda_i y_i \langle \omega, x_i \rangle = \sum_{i=1}^{n} \lambda_i y_i \left\langle \sum_{j=1}^{n} \lambda_j y_j x_j, \ x_i \right\rangle$$

$$= \sum_{i=1}^{n} \lambda_i y_i \left[\langle \lambda_1 y_1 x_1, x_i \rangle + \cdots + \langle \lambda_n y_n x_n, x_i \rangle \right]$$

$$= \sum_{i=1}^{n} \lambda_i y_i \left[\lambda_1 y_1 \langle x_1, x_i \rangle + \cdots + \lambda_n y_n \langle x_n, x_i \rangle \right]$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_j, x_i \rangle .$$

Appendix A: Derivation of the SVM Dual Program

We also have:

$$\frac{1}{2}\langle \omega, \omega \rangle = \frac{1}{2}\left\langle \sum_{i=1}^{n} \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j \right\rangle$$

$$= \frac{1}{2}\left\langle \lambda_1 y_1 x_1 + \lambda_2 y_2 x_2 + \cdots + \lambda_n y_n x_n, \sum_{j=1}^{n} \lambda_j y_j x_j \right\rangle$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left\langle \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j \right\rangle$$

$$= \frac{1}{2}\sum_{i=1}^{n} \lambda_i y_i \left\langle x_i, \sum_{j=1}^{n} \lambda_j y_j x_j \right\rangle$$

$$= \frac{1}{2}\sum_{i=1}^{n} \lambda_i y_i \left[\sum_{j=1}^{n} \lambda_j y_j \langle x_j, x_i \rangle\right]$$

$$\Rightarrow \quad \frac{1}{2}\langle \omega, \omega \rangle = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_j, x_i \rangle.$$

Appendix A: SVM Dual Problem

Definition: SVM Dual Problem

The SVM program can be written in **dual form** as:

$$\lambda^* = \underset{\lambda_1,\ldots,\lambda_n}{\arg\max} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{subject to} \quad \lambda_i \geq 0 \quad \forall\, i = 1, \ldots, n$$

$$\text{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0.$$

Appendix A: Derivation of the SVM Dual Program with Soft Margin

Appendix B

Derivation of the SVM Dual Program with Soft Margin

Appendix B: Lagrangian Formulation

Definition: Lagrangian Formulation (Non-separable Case)

For the soft-margin SVM, the **Lagrangian** is

$$\mathcal{L}(\omega, b, \xi, \lambda, \mu) = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\lambda_i\Big[\, y_i\big(\langle\omega, x_i\rangle + b\big) - \big(1-\xi_i\big)\,\Big] - \sum_{i=1}^{n}\mu_i\,\xi_i,$$

with multipliers $\lambda = (\lambda_1, \ldots, \lambda_n)'$ and $\mu = (\mu_1, \ldots, \mu_n)'$ satisfying, for all $i = 1, \ldots, n$,

$$\lambda_i \geq 0, \qquad \mu_i \geq 0, \qquad \xi_i \geq 0,$$

and the complementary slackness conditions

$$\lambda_i\Big[\, y_i\big(\langle\omega, x_i\rangle + b\big) - \big(1-\xi_i\big)\,\Big] = 0, \qquad \mu_i\,\xi_i = 0.$$

Appendix B: Karush–Kuhn–Tucker Conditions

Karush–Kuhn–Tucker Conditions

The **Karush–Kuhn–Tucker (KKT)** conditions for the soft-margin SVM primal are:

$$\frac{\partial \mathcal{L}}{\partial \omega} = \omega - \sum_{i=1}^{n} \lambda_i y_i x_i = 0, \qquad \frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \lambda_i y_i = 0,$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \mu_i - \lambda_i = 0, \qquad i = 1, \ldots, n,$$

$$\lambda_i \Big[y_i \big(\langle \omega, x_i \rangle + b \big) - \big(1 - \xi_i \big) \Big] = 0, \qquad \mu_i \, \xi_i = 0,$$

$$\lambda_i \geq 0, \qquad \mu_i \geq 0, \qquad \xi_i \geq 0, \qquad i = 1, \ldots, n.$$

Appendix B: Dual Problem with Soft Margin

Definition: Dual Problem with Soft Margin

The **dual program** of the soft-margin SVM is

$$\lambda^* = \underset{\lambda_1,\ldots,\lambda_n}{\arg\max} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$\text{s.t.} \quad 0 \leq \lambda_i \leq C \quad (i = 1, \ldots, n), \qquad \sum_{i=1}^{n} \lambda_i y_i = 0,$$

where $C > 0$ is the penalty (cost) parameter.

End of Session

Christophe Hurlin (University of Orléans and IUF)