Pattern Recognition - Summary

Friedrich-Alexander-Universität Erlangen-Nürnberg

Winter Term 2025/26

November 26, 2025

Contents

1 Optimization

1.1 Convexity

A function $f : \mathbb{R}^n \to \mathbb{R}$ is called **convex** if for all $x, y \in \mathbb{R}^n$ and for all $\theta \in [0, 1]$, the following condition holds:

Definition: Convexity & Concavity

A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if the domain $\text{dom}(f)$ of f is a convex set and if $\forall \mathbf{x}, \mathbf{y} \in \text{dom}(f)$, and θ with $0 \leq \theta \leq 1$, we have:

$$f(\theta \mathbf{x} + (1 - \theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta) f(\mathbf{y}) \tag{1.1}$$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is **concave** if $-f$ is convex.

This means that the line segment connecting any two points on the graph of the function lies **above or on the graph itself**. Convex functions have the property that any local minimum is also a global minimum, which is particularly useful in optimization problems.

1.2 Unconstrained Optimization

In unconstrained optimization, we aim to find the minimum (or maximum) of a function $f : \mathbb{R}^d \to \mathbb{R}$ without any restrictions on the variable values. Typically we assume that the function f is twice differentiable and convex. The unconstrained otpimization is the solution to the minimization problem

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) \tag{1.2}$$

Where $\vec{x}$ denotes the optimal point. For this family of functions, a necessary and sufficient condition for a minimum are the zero-crossings of the gradient:

$$\nabla f(\vec{x}^*) = 0 \tag{1.3}$$

Most of the time, we cannot find a closed-form solution to this equation. So we need to follow an iterative approach:

$$\begin{array}{ll} \text{initialization} & \mathbf{x}^{(0)} \\ \text{iteration step} & \mathbf{x}^{(k+1)} = g(\mathbf{x}^{(k)}) \end{array}$$

where $g : \mathbb{R}^d \to \mathbb{R}^d$ is the update function.

The iterations terminate, if

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon,$$

for some small threshold $\varepsilon > 0$. This means that there is not furher significant change in the variable values, indicating convergence to a solution. Some common iterative optimization methods will be discussed below.

1.2.1 Descent Methods

The basic idea of descent methods is to iteratively take small steps $\Delta\vec{x}^{(k)}$ in the direction of **steepest descent** (i.e., the negative gradient) to minimize the function value:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + t^{(k)} \Delta\vec{x}^{(k)} \tag{1.4}$$

where

$$\Delta\vec{x}^{(k)} \in \mathbb{R}^d: \quad \text{is the search direction in the k-th iteration} \tag{1.5}$$

$$t^{(k)} \in \mathbb{R}: \quad \text{denotes the step length in the k-th iteration} \tag{1.6}$$

and where we expect:

$$f(\vec{x}^{(k+1)}) < f(\vec{x}^{(k)}) \quad \text{except} \quad \vec{x}^{(k+1)} = \vec{x}^{(k)} = \vec{x}^* \tag{1.7}$$

Finding a suitable step size

It is important to choose an appropriate step size $t^{(k)}$ to ensure convergence and avoid overshooting the minimum. If t is too large, we might overshoot the minimum, while if it is too small, convergence can be very slow or get stuck in local minima. The procedure to find a suitable step

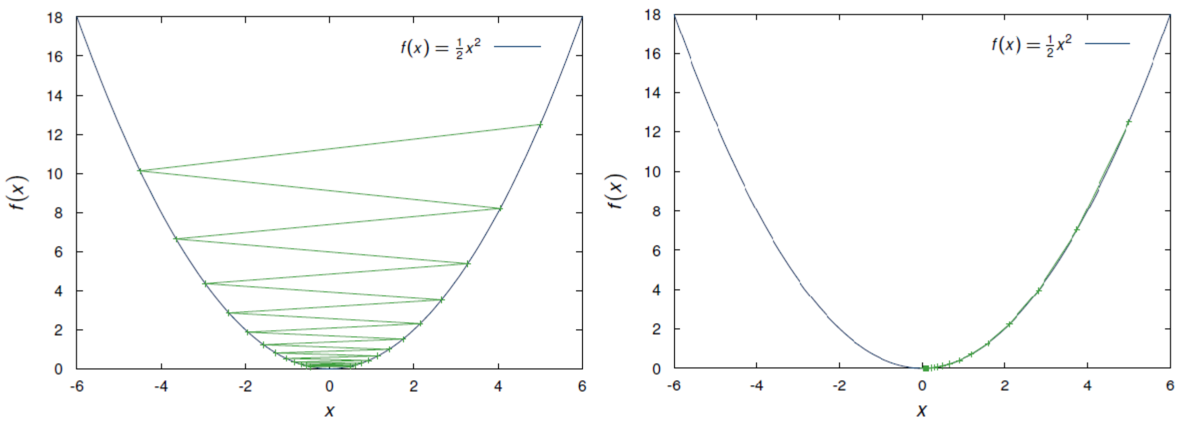

Figure 1: Learning Rate Illustration

size is called **line search**. You can basically see this as a function $F(\vec{x}^{(k)}) + t^{(k)}\Delta\vec{x}^{(k)}$ where $\Delta\vec{x}^{(k)}$ and $\vec{x}^{(k)}$ are fixed and we only vary $t^{(k)}$ to find the minimum of this 1D function. A com-

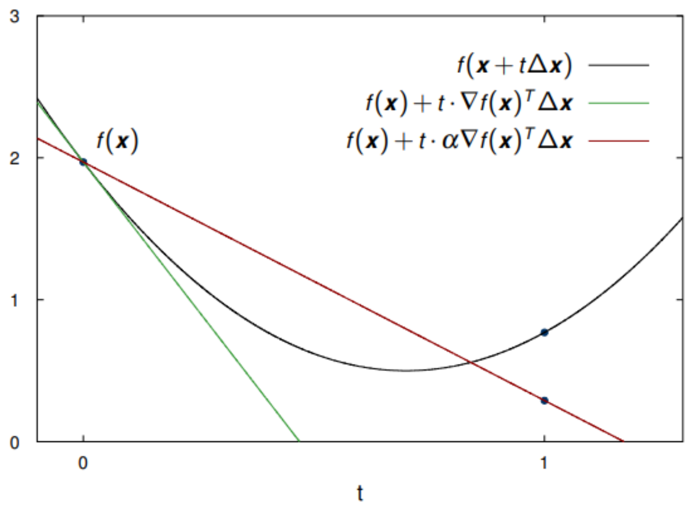

Figure 2: Backtracking line search

mon method for line search is **backtracking line search (Armijo-Goldstein Algorithm)**, which starts with an initial step size and iteratively reduces it until a sufficient decrease in the function value is observed.

1.2.2 Gradient Descent

A natutral choice for the search direction is the negative gradient of the function at the current point:

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)}) \tag{1.8}$$

You can use either a fixed step size or perform a line search to determine the optimal step size at each iteration.

```
Algorithm:
Input: function $f$, initial estimate $\mathbf{x}^{(0)}$
initialize: $k := 0$
repeat
    1. Set descent direction: $\Delta\mathbf{x}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$
    2. Line search (1-D optimization):
           $t^{(k)} = \arg\min_{t \geq 0} f(\mathbf{x}^{(k)} + t \cdot \Delta\mathbf{x}^{(k)})$
    3. Update:
           $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)} \Delta\mathbf{x}^{(k)}$
    4. $k := k + 1$
until $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2 < \varepsilon$
Output: $\mathbf{x}^{(k)}$
```

1.2.3 Normalized Steepest Descent

In normalilzed steepest descent, the search direction is given by the following optimization problem:

$$\Delta\vec{x} = \arg\min_{u} \nabla f(\vec{x})^T \vec{u} \quad \text{s.t.} \quad \|\vec{u}\| = 1 \tag{1.9}$$

This means that we compute the inner product of the gradient and a unit vector $\vec{u}$ (project the gradient onto $\vec{u}$) and choose the direction with the steepest descent.

```
Algorithm:
Input: function $f$, initial estimate $\mathbf{x}^{(0)}$, norm $\|\cdot\|$
initialize: $k := 0$
repeat
    1. Compute highest descent direction:
           $\Delta\mathbf{x}^{(k)} = \arg\min_{\mathbf{u}} \{\nabla f(\mathbf{x}^{(k)})^T \mathbf{u}; \|\mathbf{u}\| = 1\}$
    2. Line search (1-D optimization):
           $t^{(k)} = \arg\min_{t \geq 0} f(\mathbf{x}^{(k)} + t \cdot \Delta\mathbf{x}^{(k)})$
    3. Update:
           $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)} \Delta\mathbf{x}^{(k)}$
    4. $k := k + 1$
```

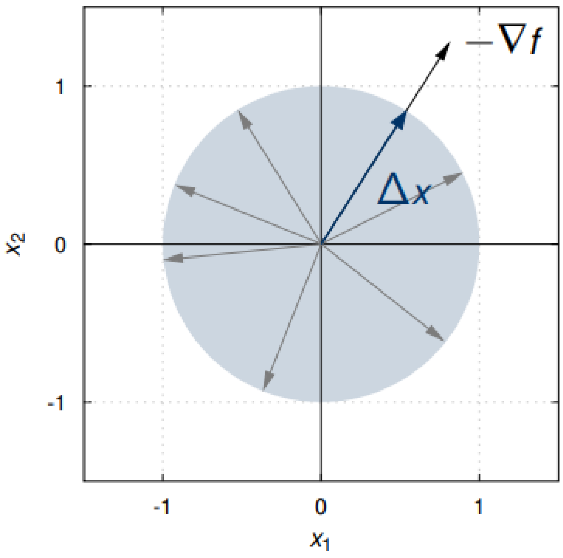

Figure 3: Unit ball in L2 norm

until $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \varepsilon$
Output: $\mathbf{x}^{(k)}$ If you choose the L2-norm, the unit ball is a circle (2D) or a sphere (3D). In this case, the steepest descent direction is simply the negative gradient direction. Therefore we get the same update rule as in gradient descent.

If you choose the L1-norm, the unit ball is a diamond (2D) or an octahedron (3D). In this case,

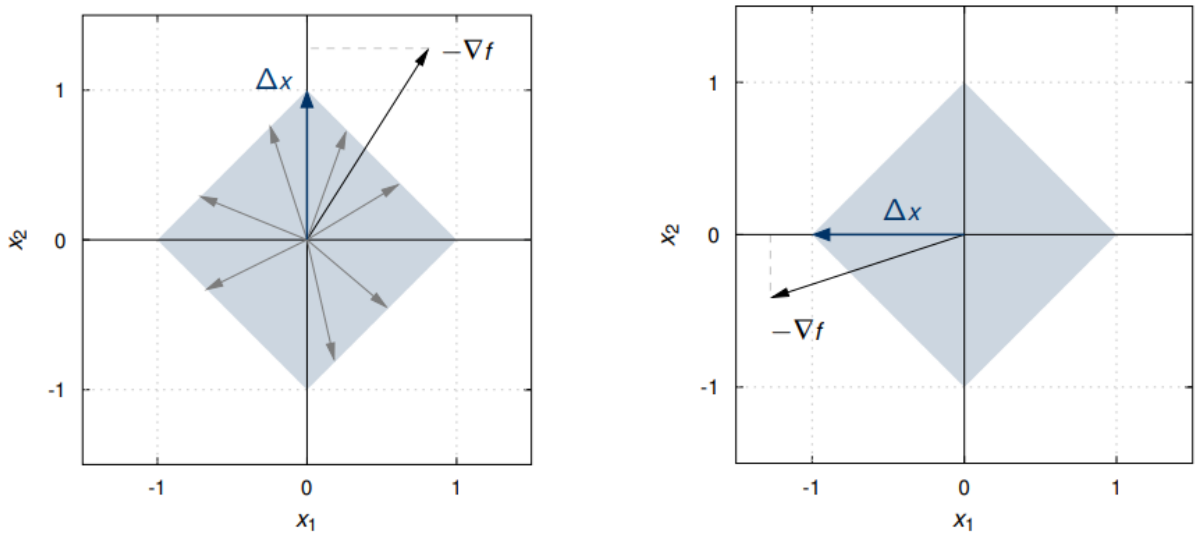

Figure 4: Unit ball in L1 norm

the steepest descent direction corresponds to the coordinate axis clostest to the negative gradient direction. This leads to a coordinate descent method, where we optimize along one coordinate axis in each iteration:

$$\begin{aligned}\Delta\mathbf{x} &= \arg\min_{\mathbf{u}} \left\{\nabla f(\mathbf{x})^T\mathbf{u}\,;\ \|\mathbf{u}\|_1 = 1\right\}\\ &= -\operatorname{sgn}\left(\frac{\partial}{\partial x_i} f(\mathbf{x})\right)\mathbf{e}_i\end{aligned}$$

where e_i is the unit vector along the coordinate axis with the highest absolute gradient value.

If you choose the L_∞ norm, the unit ball is a square (2D) or a cube (3D). In this case, the

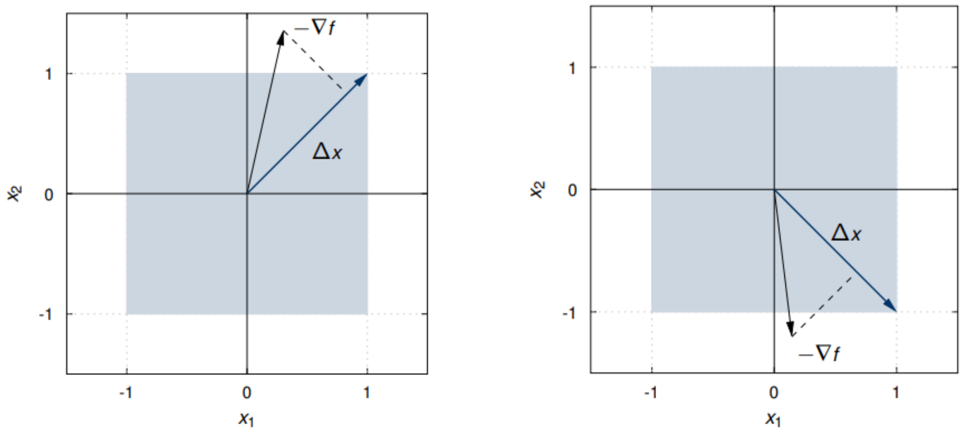

Figure 5: Unit ball in L-Infinity norm

direction of steepest descent is always aligned withe the diagonal of the unit cube (0°).

If you choose the L_P norm, the unit ball is a rounded square (2D) or a rounded cube (3D). Using

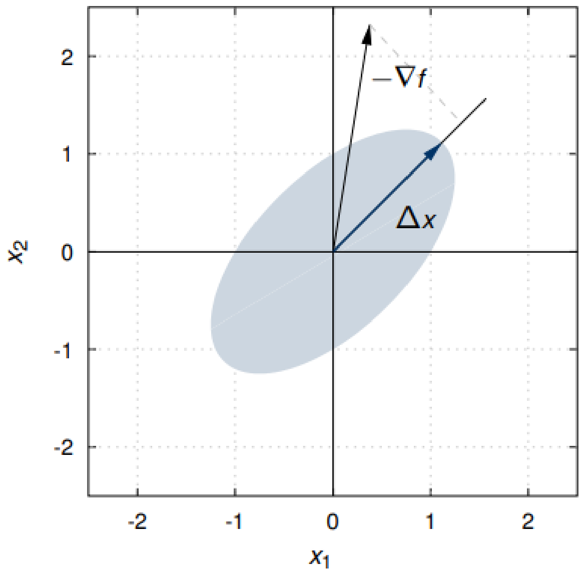

Figure 6: Unit ball in L-P norm

a quadratic norm defined by a positive definite matrix $\mathbf{P}$ ($\|\vec{u}\|_{\mathbf{P}} = \sqrt{\vec{u}^T\mathbf{P}\vec{u}}$), the steepest descent direction is:

$$\Delta\vec{x} = -\mathbf{P}^{-1}\nabla f(\vec{x}) \tag{1.10}$$

Connection to LDA: Just like in LDA, where we transform the feature space to make the class distributions spherical (whitening) using the covariance matrix, this optimization step uses $\mathbf{P}^{-1}$ to account for the geometry of the function.

- If $\mathbf{P} = \mathbf{I}$: Standard Gradient Descent (assumes spherical geometry).
- If $\mathbf{P} = \nabla^2 f(\vec{x})$ (Hessian): Newton's Method (accounts for local curvature).

1.2.4 Newton's Method

Newton's method uses second-order information (the Hessian matrix) to find the search direction. It uses the second-order Taylor polynomial to approximate the function $f(x)$ at the current point $x^{(k)}$ and finds the minimum of this approximation. The minimum is the new point $x^{(k+1)}$. The

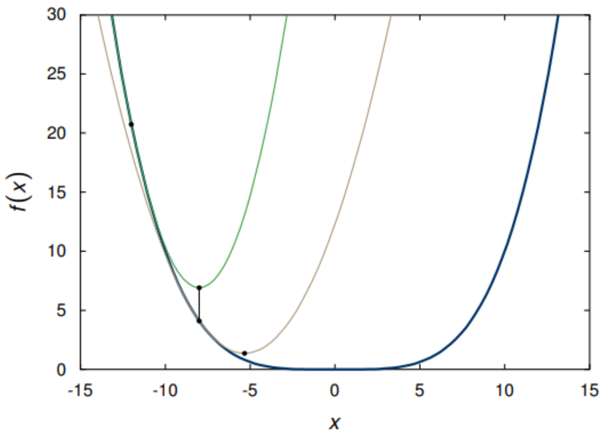

Figure 7: Newtons Method using 2nd order Taylor approximation

second-order Taylor approximation of $f(\vec{x})$ is given by:

$$f(\vec{x} + \Delta\vec{x}) \approx f(\vec{x}) + \nabla f(\vec{x})^T \Delta\vec{x} + \frac{1}{2}\Delta\vec{x}^T()\nabla^2 f(\vec{x}))\Delta\vec{x} \tag{1.11}$$

To find the minimum of this approximation, we set the gradient with respect to $\Delta\vec{x}$ to zero (works because the function is convex) and resolve for $\Delta\vec{x}$:

$$\Delta\vec{x} = -\underbrace{(\nabla^2 f(\vec{x}))^{-1}}_{\text{inverse of the Hessian}} \nabla f(\vec{x}) \tag{1.12}$$

Note: Have a look at the steepest descent method using the $\mathbf{L_P}$-norm. When P is the Hessian matrix ($\mathbf{P} = \nabla^2 f(\vec{x})$), we get the same search direction as in Newton's method.

1.3 Constrained Optimization

We consider the primal optimization problem:

$$\begin{aligned} \text{minimize} \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, m \\ & h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, p \end{aligned}$$

Note: $f_0(\vec{x})$ is not required to be convex!

To optimize a function with constraints we have to use Lagrange-Multipliers. The Lagrangian L is defined as:

$$L(\vec{x}, \vec{\lambda}, \vec{\nu}) = f_0(\vec{x}) + \underbrace{\sum_{i=1}^{m} \lambda_i f_i(\vec{x})}_{\text{sum over all inequality constraints}} + \underbrace{\sum_{i=1}^{p} \nu_i h_i(\vec{x})}_{\text{sum over all equality constraints}} \tag{1.13}$$

We add the constraints and weight them with multipliers λ, ν (Langrange multipliers or simply dual variables). The optimization problem can then be reformulated using the **Lagrange dual function**:

$$\begin{aligned} g(\vec{\lambda}, \vec{\nu}) &= \inf_{\vec{x}} L(\vec{x}, \vec{\lambda}, \vec{\nu}) \\ &= \inf_{\vec{x}} \left(f_0(\vec{x}) + \sum_{i=1}^{m} \lambda_i f_i(\vec{x}) + \sum_{i=1}^{p} \nu_i h_i(\vec{x}) \right) \end{aligned}$$

Definition: Infimum (inf)

The **infimum** is the greatest lower bound of a set. Unlike the minimum, it exists even for open sets (e.g., for the set $\{x \in \mathbb{R} \mid x > 0\}$, the minimum does not exist, but the infimum is 0).

Important Properties:

- The Lagrange dual function is always **concave**, even if the original primal problem is not convex!
- **Pointwise Affine:** The dual function is a pointwise affine function in the dual variables.
  - *Explanation:* If we fix $\vec{x}$, the Lagrangian $L(\vec{x}, \vec{\lambda}, \vec{\nu})$ becomes just a linear equation in terms of $\vec{\lambda}$ and $\vec{\nu}$ (like $a + b\lambda + c\nu$). It describes a flat plane.
- It provides a **lower bound** on the optimal value p^* of the primal problem. For any $\vec{\lambda} \succeq 0$ and any $\vec{\nu}$, the following holds:
$$g(\vec{\lambda}, \vec{\nu}) \leq p^* \tag{1.14}$$

This leads us to the **Lagrange Dual Problem**, where we try to find the best (highest) lower bound:

$$\begin{aligned} \text{maximize} \quad & g(\vec{\lambda}, \vec{\nu}) \\ \text{subject to} \quad & \vec{\lambda} \succeq 0 \end{aligned}$$

Optimal Duality Gap

Let p^* be the optimal value of the primal problem and d^* the optimal value of the Lagrange dual problem.

- The difference $p^* - d^*$ is the **optimal duality gap**.
- If $p^* = d^*$, the duality gap is zero. In this case we talk about **strong duality**.
- If $p^* > d^*$, we have **weak duality**.

The duality gap is always zero when **Slater's Condition** is fulfilled:

Theorem: Slater's Condition

For a **convex** optimization problem, strong duality ($p^* = d^*$) holds if there exists a strictly feasible point $\mathbf{x}$ such that:

$$f_i(\mathbf{x}) < 0 \quad \forall i = 1, \ldots, m \quad \text{and} \quad A\mathbf{x} = \mathbf{b} \tag{1.15}$$

Intuition: To guarantee strong duality, we need at least one "strictly feasible" point.

- It must lie **strictly inside** the inequality constraints ($f_i(\mathbf{x}) < 0$), meaning it does not touch these boundaries.
- At the same time, it must **exactly satisfy** the linear equality constraints ($A\mathbf{x} = \mathbf{b}$).

In simple terms: You need a point that walks exactly on the required path ($A\mathbf{x} = \mathbf{b}$) without touching the walls of the feasible region ($f_i(\mathbf{x}) < 0$).

2 Support Vector Machines (SVM)

Just like other classifiers (Neural Nets, Nearest Neighbor, etc.), the goal of SVMs is to draw a linear line (decision boundary) to separate classes. But instead of drawing any sufficient line to separate the classes, SVMs aim to find a unique decision boundary that **maximizes the margin (distance)** between each class. The solution to this problem is unique and depends only on the features that are close to the decision boundary.

2.1 Hard Margin Problems

The hard margin SVM needs linearly seperable classes. Lets assume there is an affine function

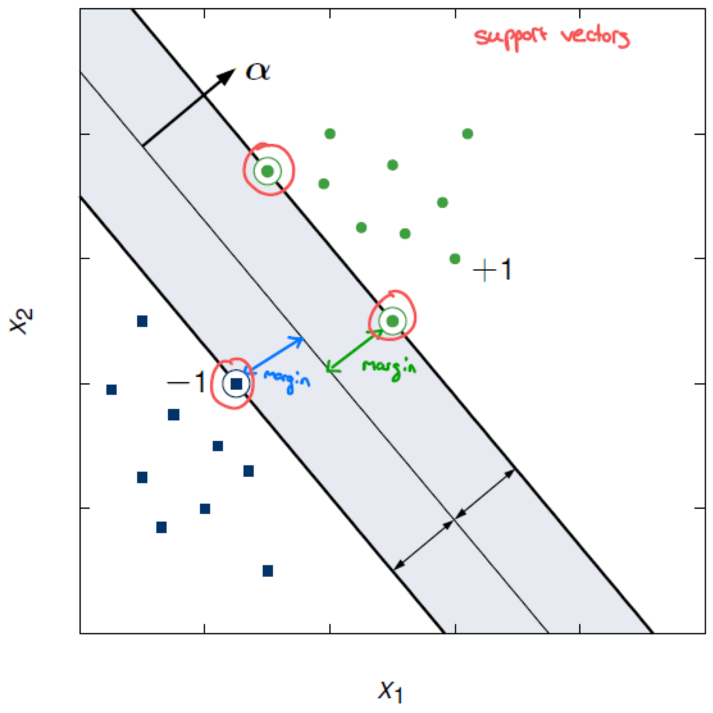

Figure 8: Hard margin SVM

defined as:

$$f(x) = \vec{\alpha}^T x + \alpha_0 \tag{2.1}$$

Where $\vec{\alpha}$ is the normal vector to the decision boundary and α_0 is some sort of bias. For any point x on the decision boundary, it holds that $f(x) = 0$.
There are three major points we need to think about to end up with a nice optimization problem:

1. Introduce margin constraints:
We need to ensure that all points are classified correctly and therefore lie outside the margin. Therefore, we introduce the following constraints:

- For points of class $+1$: $f(x) \geq 1$
- For points of class -1: $f(x) \leq -1$

This means that for a sample point x_i with the label $y_i = +1$, it holds that $f(x_i) \geq 1$. Similarly, for a sample point x_j with the label $y_j = -1$, it holds that $f(x_j) \leq -1$. These two constraints can be combined into one single constraint ($y \in \{+1, -1\}$):

$$y_i f(x_i) - 1 = y_i(\vec{\alpha}^T x_i + \alpha_0) - 1 \geq 0 \quad \forall i \tag{2.2}$$

2. Define the margin:
The margin is defined as the distance between the decision boundary and the closest points from either class. To compute the margin width, we take a sample from each class that lies exactly on the margin (i.e., satisfies $y_i f(x_i) - 1 = 0 \quad \forall i$) and subtract them from each other. When we project the resulting vector onto tho normalized normal vector of the hyperplane, we get the margin width:

$$\text{width} = \frac{\vec{\alpha}}{\|\vec{\alpha}\|_2} \cdot (\vec{x}_{y=+1} - \vec{x}_{y=-1}) \tag{2.3}$$

Now we multiply this out:

$$\text{width} = \frac{1}{\|\vec{\alpha}\|_2}(\vec{\alpha}^T \vec{x}_{y=+1} - \vec{\alpha}^T \vec{x}_{y=-1}) \tag{2.4}$$

We know from our margin constraints defined in step 1 that for support vectors (points on the margin), the inequality becomes an equality:

- For the positive support vector $\vec{x}_{y=+1}$:

$$\vec{\alpha}^T \vec{x}_{y=+1} + \alpha_0 = 1 \quad \Rightarrow \quad \vec{\alpha}^T \vec{x}_{y=+1} = 1 - \alpha_0$$

- For the negative support vector $\vec{x}_{y=-1}$:

$$\vec{\alpha}^T \vec{x}_{y=-1} + \alpha_0 = -1 \quad \Rightarrow \quad \vec{\alpha}^T \vec{x}_{y=-1} = -1 - \alpha_0$$

Substituting these expressions back into the width equation:

$$\begin{aligned} \text{width} &= \frac{1}{\|\vec{\alpha}\|_2}((1 - \alpha_0) - (-1 - \alpha_0)) \quad &(2.5) \\ &= \frac{1}{\|\vec{\alpha}\|_2}(1 - \alpha_0 + 1 + \alpha_0) \quad &(2.6) \\ &= \frac{2}{\|\vec{\alpha}\|_2} \quad &(2.7) \end{aligned}$$

3. Minimize the norm:
Since we want to **maximize** the margin width $\frac{2}{\|\vec{\alpha}\|_2}$, this is mathematically equivalent to **minimizing** the length of the normal vector $\|\vec{\alpha}\|_2$. For mathematical convenience (to make derivatives easier later), we minimize the squared norm:

Primal Optimization Problem (Hard Margin)

$$\text{minimize} \quad \frac{1}{2}\|\vec{\alpha}\|_2^2 \quad \text{subject to} \quad y_i(\vec{\alpha}^T x_i + \alpha_0) \geq 1 \quad \forall i$$

2.2 Soft Margin Problems

In real world applications, data is often not perfectly linearly separable. The Soft Margin SVM relaxes the hard margin constraints by allowing some points to violate the margin or even be misclassified.

Therefore, we introduce slack variables $\xi_i \geq 0$ for each training sample x_i, which measure the degree of misclassification:

- $\xi_i = 0$: point is correctly classified and outside or on the margin
- $0 < \xi_i \leq 1$: point is inside the margin but still correctly classified

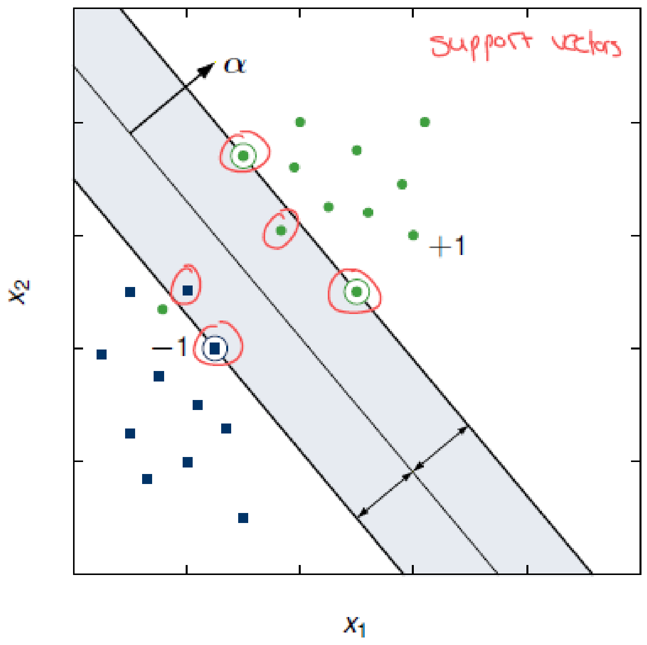

Figure 9: Soft margin SVM

- $\xi_i > 1$: point is misclassified

The margin constraints are now relaxed to:

$$y_i(\vec{\alpha}^T x_i + \alpha_0) \geq 1 - \xi_i \quad \forall i \tag{2.8}$$

The primal optimization problem becomes:

Primal Optimization Problem (Soft Margin)

$$\text{minimize} \quad \frac{1}{2}\|\vec{\alpha}\|_2^2 + \mu \sum_{i=1}^{n} \xi_i \quad \text{subject to} \quad -(y_i(\vec{\alpha}^T x_i + \alpha_0) - 1 + \xi_i) \leq 0, \quad -\xi_i \leq 0 \quad \forall i$$

where $\mu > 0$ is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error.