Pattern Recognition - Summary

Friedrich-Alexander-Universität Erlangen-Nürnberg

Winter Term 2025/26

November 24, 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 $\mathrm{dom}(f)$ of f is a convex set and if $\forall \mathbf{x}, \mathbf{y} \in \mathrm{dom}(f)$, 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:

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

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.

1.3 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:

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

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

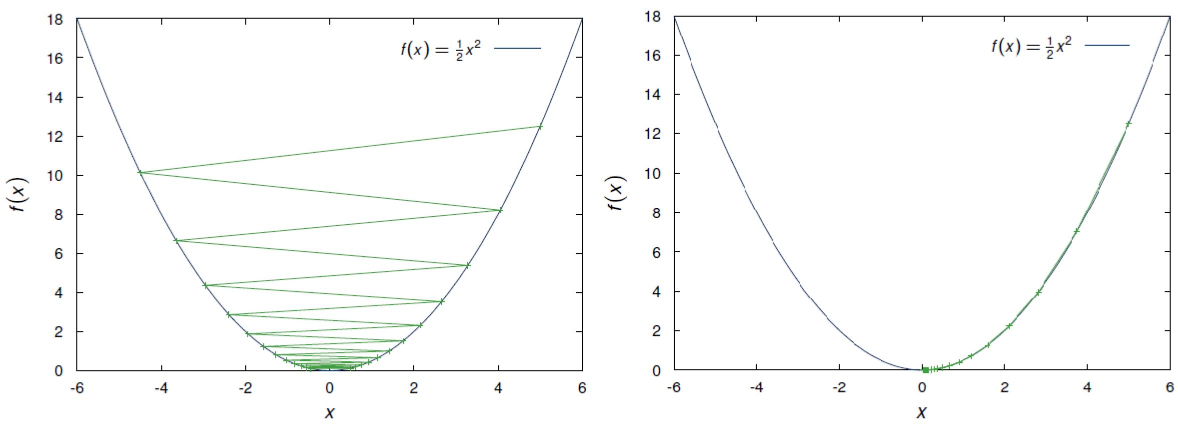

Figure 1: Learning Rate Illustration

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-

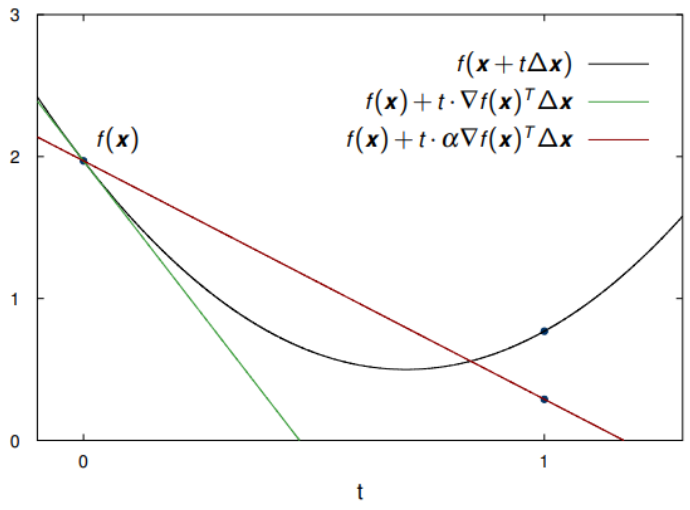

Figure 2: Backtracking line search

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.

1.3.1 Gradient Descent

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

1.3.2 Newtons Method

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

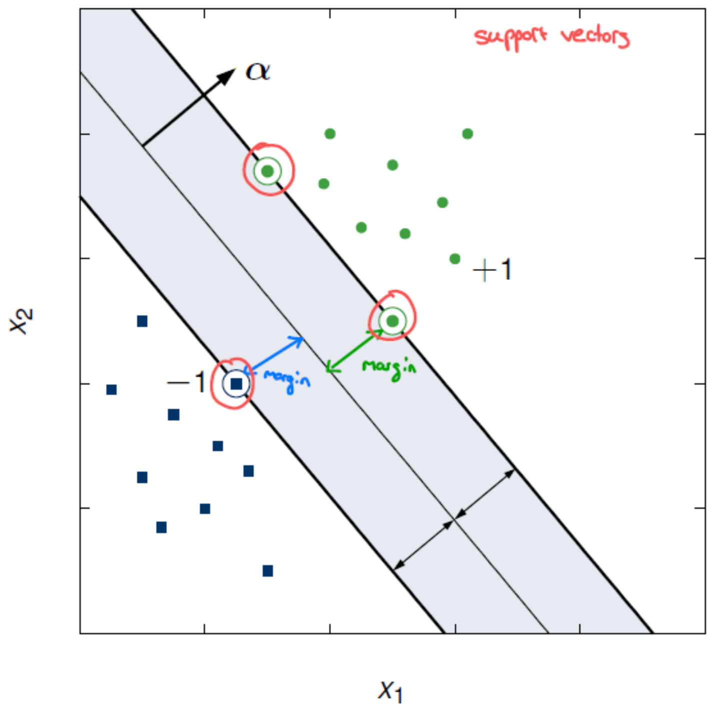

Figure 3: Hard margin SVM

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:

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

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

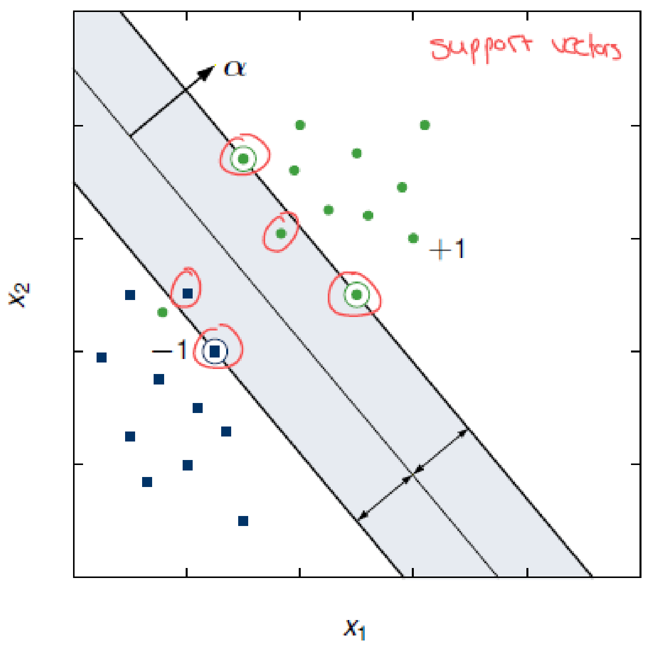

Figure 4: Soft margin SVM

- $\xi_i > 1$: 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.