Pattern Recognition - Summary

Friedrich-Alexander-Universität Erlangen-Nürnberg

Winter Term 2025/26

November 27, 2025

Contents

1 Optimization

Optimization is crucial for many solutions in pattern recognition (e.g., training classifiers). We distinguish between unconstrained and constrained optimization problems.

1.1 Convexity

A function $f : \mathbb{R}^n \to \mathbb{R}$ is called **convex** if for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ 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 $\text{dom}(f)$ is a convex set and if $\forall \vec{x}, \vec{y} \in \text{dom}(f)$, and θ with $0 \leq \theta \leq 1$, we have:

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

A function is **concave** if $-f$ is convex.

Implication: For convex functions, any local minimum is also a global minimum. This property is particularly useful because gradient-based methods won't get stuck in suboptimal local minima.

1.2 Unconstrained Optimization

Here, we aim to find the minimum of a function $f(\vec{x})$ without any restrictions on $\vec{x}$. Typically, we assume f is twice differentiable and convex.

$$\vec{x}^* = \text{argmin}_{\vec{x}} f(\vec{x}) \tag{1.2}$$

A necessary and sufficient condition for the minimum is the zero-crossing of the gradient:

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

Since a closed-form solution is often impossible, we use iterative approaches:

$$\begin{aligned} \text{initialization:} \quad & \vec{x}^{(0)} \\ \text{iteration step:} \quad & \vec{x}^{(k+1)} = \vec{x}^{(k)} + t^{(k)} \Delta\vec{x}^{(k)} \end{aligned}$$

where $\Delta\vec{x}^{(k)}$ is the **search direction** and $t^{(k)}$ is the **step size**.

1.2.1 Finding a Suitable Step Size (Line Search)

Choosing the correct step size $t^{(k)}$ is crucial:

- **Too small:** Convergence is extremely slow.
- **Too large:** The algorithm might overshoot the minimum or diverge.

Instead of finding the exact optimal t (which is computationally expensive), we use **inexact line search** methods like **Backtracking Line Search** (Armijo-Goldstein).

The goal is to find a step size t that, that puts us below the red line, ensuring that the function value decreases sufficiently (not just barely) relative to the step size.

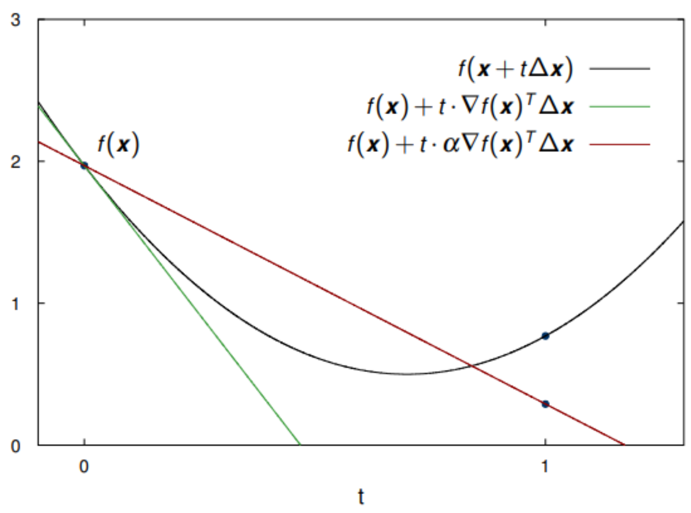

Figure 1: Backtracking line search

Armijo-Goldstein Condition (Backtracking)

We start with a large step size ($t = 1$) and iteratively reduce it ($t := \beta t$) until the function value decreases sufficiently. The condition is:

$$f(\vec{x} + t\Delta\vec{x}) \leq f(\vec{x}) + \alpha t \nabla f(\vec{x})^T \Delta\vec{x} \tag{1.4}$$

Where $\alpha \in (0, 0.5)$ defines the required "steepness" of the descent.

1.2.2 Gradient Descent

The most natural choice for the search direction is the direction of steepest descent, which is the negative gradient.

$$\Delta\vec{x}^{(k)} = -\nabla f(\vec{x}^{(k)}) \tag{1.5}$$

Algorithm: Gradient Descent

1. Set direction: $\Delta\vec{x}^{(k)} = -\nabla f(\vec{x}^{(k)})$
2. Line search (find optimal step size t):

$$t^{(k)} = \text{argmin}_{t\geq 0} f(\vec{x}^{(k)} + t\Delta\vec{x}^{(k)})$$

   (Usually approximated via **Backtracking Line Search** / **Armijo-Goldstein**).

3. Update: $\vec{x}^{(k+1)} = \vec{x}^{(k)} + t^{(k)}\Delta\vec{x}^{(k)}$
4. Repeat until convergence ($\|\vec{x}^{(k)} - \vec{x}^{(k-1)}\| < \epsilon$).

1.2.3 Normalized Steepest Descent (General Norms)

Ideally, we want the direction that gives the largest decrease in the linear approximation of f. This depends on the chosen norm $\|\cdot\|$.

$$\Delta\vec{x} = \text{argmin}_{\vec{u}}\{\nabla f(\vec{x})^T\vec{u} \mid \|\vec{u}\| = 1\} \tag{1.6}$$

- **L_2-Norm:** Direction is $-\nabla f(\vec{x})$ (Standard Gradient Descent). Unit ball is a sphere.

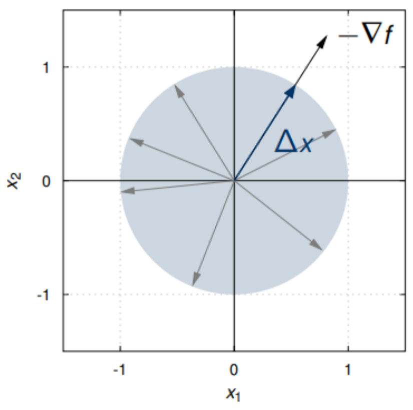

Figure 2: Unit ball in L_2 norm

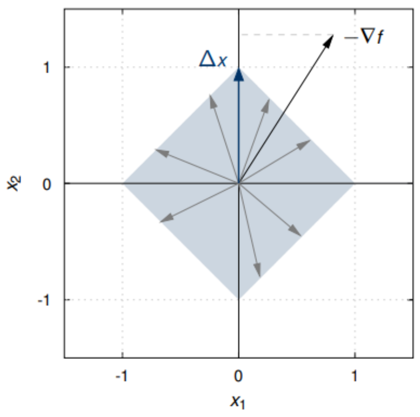

Figure 3: Unit ball in L_1 norm

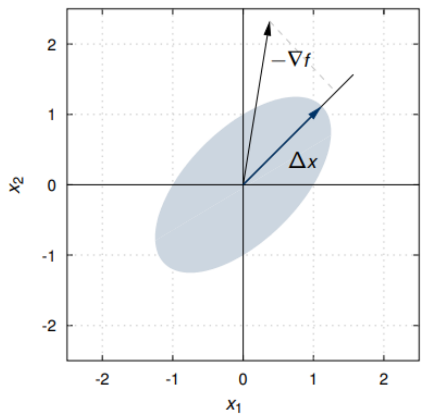

Figure 4: Unit ball in L_P norm

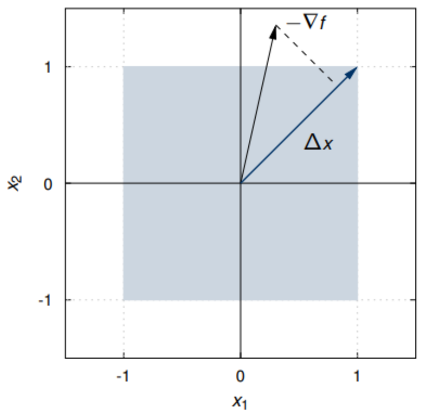

Figure 5: Unit ball in L_∞ norm

- L_1**-Norm:** Direction is along the coordinate axis with the largest absolute gradient value (Coordinate Descent). Unit ball is a diamond.
- L_∞**-Norm:** Direction points towards the corners of the unit hypercube (e.g., vector with entries ± 1). Unit ball is a square/cube.
- L_P**-Norm (Quadratic Norm):** Defined by a positive definite matrix $\mathbf{P}$ as $\|\vec{u}\|_{\mathbf{P}} = \sqrt{\vec{u}^T \mathbf{P} \vec{u}}$.

$$\Delta \vec{x} = -\mathbf{P}^{-1} \nabla f(\vec{x}) \tag{1.7}$$

This transforms the space to make the contours spherical before taking the gradient step (similar to whitening in LDA). The update direction aligns with the largest principal component in the transformed space.

Crucial Connection: If we set $\mathbf{P} = \nabla^2 f(\vec{x})$ (the Hessian), we effectively perform steepest descent in the local curvature norm, which yields **Newton's Method**.

1.2.4 Newton's Method

Newton's method finds the search direction by approximating the function $f(\vec{x})$ locally using a **second-order Taylor polynomial**:

$$f(\vec{x} + \Delta \vec{x}) \approx f(\vec{x}) + \nabla f(\vec{x})^T \Delta \vec{x} + \frac{1}{2} \Delta \vec{x}^T \nabla^2 f(\vec{x}) \Delta \vec{x} \tag{1.8}$$

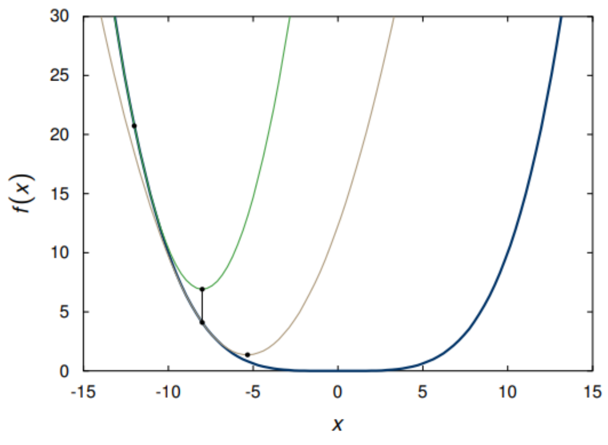

Figure 6: Newtons Method using 2nd order Taylor approximation

We find the optimal step $\Delta\vec{x}$ by setting the derivative of this approximation to zero:

$$\begin{aligned} &\nabla_{\Delta\vec{x}}(\dots) = \nabla f(\vec{x}) + \nabla^2 f(\vec{x})\Delta\vec{x} = 0 \\ \Rightarrow \quad &\Delta\vec{x} = -(\nabla^2 f(\vec{x}))^{-1}\nabla f(\vec{x}) \end{aligned}$$

Key Difference: While Gradient Descent assumes the local geometry is a sphere (linear approximation), Newton's Method accounts for the local **curvature** (ellipsoidal geometry) given by the Hessian $\nabla^2 f(\vec{x})$.

1.3 Constrained Optimization

We consider the primal optimization problem with constraints:

$$\begin{aligned} \text{minimize} \quad & f_0(\vec{x}) \\ \text{subject to} \quad & f_i(\vec{x}) \le 0, \quad i = 1, \dots, m \\ & h_i(\vec{x}) = 0, \quad i = 1, \dots, p \end{aligned}$$

Note: f_0 is not required to be convex yet.

1.3.1 The Lagrangian & Dual Function

To handle constraints, we define the **Lagrangian** L:

$$L(\vec{x}, \vec{\lambda}, \vec{\nu}) = f_0(\vec{x}) + \sum_{i=1}^{m} \lambda_i f_i(\vec{x}) + \sum_{i=1}^{p} \nu_i h_i(\vec{x}) \tag{1.9}$$

We formulate the **Lagrange dual function** $g(\vec{\lambda}, \vec{\nu})$ by minimizing L over $\vec{x}$:

$$g(\vec{\lambda}, \vec{\nu}) = \inf_{\vec{x}} L(\vec{x}, \vec{\lambda}, \vec{\nu}) \tag{1.10}$$

Properties of the Dual Function

- $g(\vec{\lambda}, \vec{\nu})$ is **always concave**, even if the primal problem is not convex.
- It provides a **lower bound** on the optimal primal value p^*: $g(\vec{\lambda}, \vec{\nu}) \le p^*$.

This leads to the **Dual Problem**: Find the best lower bound by maximizing g.

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

1.3.2 Strong Duality & Slater's Condition

- **Weak Duality:** $d^* \leq p^*$ (Always true). The difference $p^* - d^*$ is the *duality gap*.
- **Strong Duality:** $d^* = p^*$ (Gap is zero). This allows solving the dual problem to find the primal solution.

Strong duality is guaranteed if **Slater's Condition** holds:

Theorem: Slater's Condition

For a **convex** optimization problem, strong duality holds if there exists a strictly feasible point $\vec{x}$ such that:

$$f_i(\vec{x}) < 0 \quad \forall i = 1, \ldots, m \quad \text{and} \quad A\vec{x} = \vec{b} \tag{1.11}$$

Refinement: If constraints are affine (linear), touching the boundary ($f_i(\vec{x}) \leq 0$) is allowed. Only non-linear constraints require strict inequality.

1.3.3 Karush-Kuhn-Tucker (KKT) Conditions

If strong duality holds, any optimal pair $(\vec{x}^*, \vec{\lambda}^*, \vec{\nu}^*)$ **must** satisfy the KKT conditions:

1. **Primal Feasibility:**
$$f_i(\vec{x}^*) \leq 0, \quad h_i(\vec{x}^*) = 0$$
2. **Dual Feasibility:**
$$\vec{\lambda}^* \succeq 0$$
3. **Complementary Slackness:** (Crucial for SVMs!)
$$\lambda_i^* \cdot f_i(\vec{x}^*) = 0$$
Meaning: Either a constraint is active ($f_i(\vec{x}) = 0$) or its multiplier is zero ($\lambda_i = 0$).
4. **Stationarity:** Gradient of Lagrangian is zero.
$$\nabla f_0(\vec{x}^*) + \sum \lambda_i^* \nabla f_i(\vec{x}^*) + \sum \nu_i^* \nabla h_i(\vec{x}^*) = 0$$

Conclusion: For convex problems with strong duality, KKT conditions are necessary and sufficient for optimality. Finding a point that satisfies them means we found the global optimum.

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

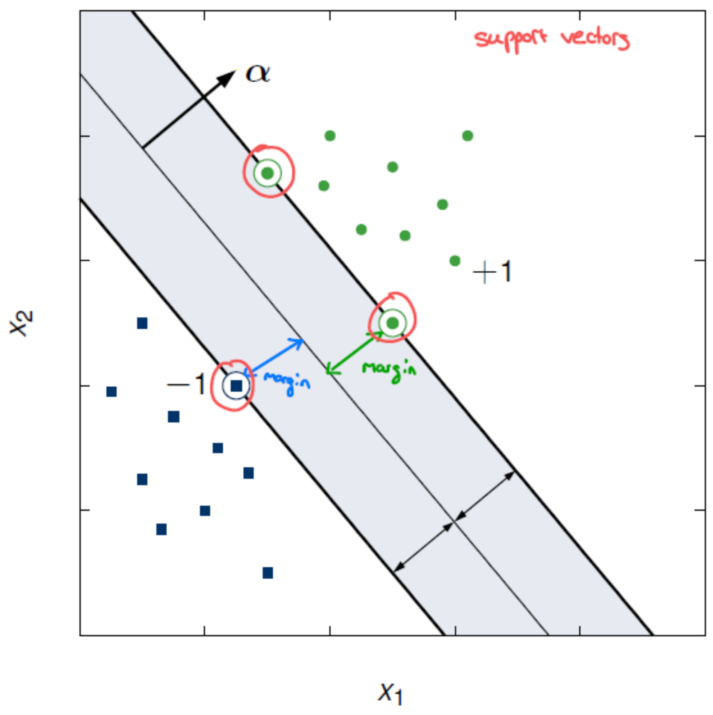

Figure 7: Hard margin SVM

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 (**Hinge Loss**):

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

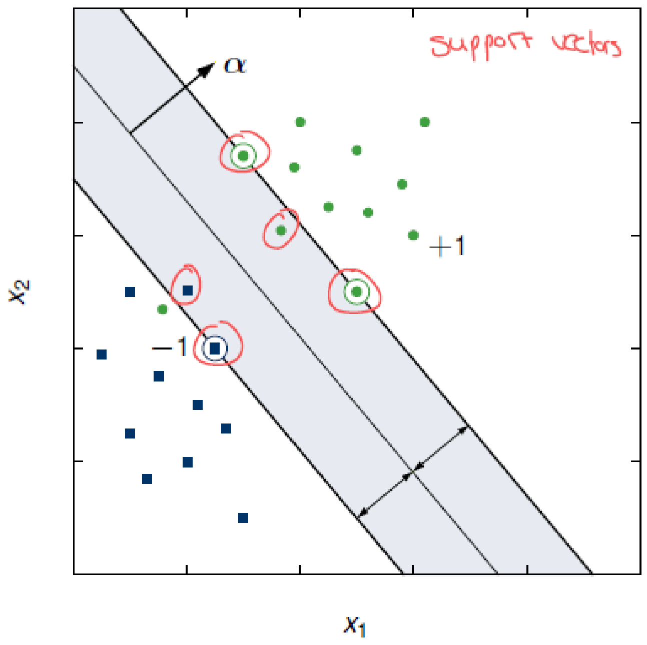

Figure 8: Soft margin SVM

- $\xi_i > 1$: 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.

2.3 Dual Representation

The primal problem minimizes with respect to $\vec{\alpha}$ and α_0. However, solving the **Lagrange Dual Problem** is often more powerful. It reveals that the solution depends *only* on the inner products of the data points, which is the key to the Kernel Trick.

2.3.1 Derivation from the Lagrangian

For simplicity, we derive the dual only for the hard margin case here. In the soft margin case, the derivation is similar but includes additional terms for the slack variables. We start with the

Lagrangian of the hard margin problem (using multipliers $\lambda_i \geq 0$):

$$L(\vec{\alpha}, \alpha_0, \vec{\lambda}) = \underbrace{\frac{1}{2}\|\vec{\alpha}\|_2^2}_{\text{primal problem } f_0(x)} \underbrace{- \sum_{i=1}^{m} \lambda_i [y_i(\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1]}_{\text{constraints } f_i(x)} \tag{2.9}$$

To find the dual, we minimize L with respect to the primal variables $\vec{\alpha}$ and α_0 (setting gradients to zero):

$$\nabla_{\vec{\alpha}} L = \vec{\alpha} - \sum_{i=1}^{m} \lambda_i y_i \vec{x}_i = 0 \quad \Rightarrow \quad \tilde{\alpha} = \sum_{\mathbf{i=1}}^{\mathbf{m}} \lambda_{\mathbf{i}} \mathbf{y_i} \tilde{\mathbf{x}}_{\mathbf{i}} \tag{2.10}$$

$$\frac{\partial L}{\partial \alpha_0} = - \sum_{i=1}^{m} \lambda_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{m} \lambda_i y_i = 0 \tag{2.11}$$

Substituting (2.10) back into the Lagrangian eliminates $\vec{\alpha}$ and yields the **Dual Objective Function**, which depends only on $\vec{\lambda}$:

Dual Optimization Problem

$$\text{maximize} \quad \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j (\vec{x}_i^T \vec{x}_j)$$

Subject to:

$$\lambda_i \geq 0 \quad \forall i, \quad \text{and} \quad \sum_{i=1}^{m} \lambda_i y_i = 0$$

2.3.2 From Dual Variables to the Decision Boundary

Once we have solved the Dual Problem and found the optimal Lagrange multipliers λ_i^*, we need to reconstruct the decision boundary. But which λ_i are actually relevant?

This is answered by the **Complementary Slackness** condition (KKT condition 3):

$$\lambda_i \cdot \underbrace{(y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1)}_{f_i(\vec{\alpha}, \alpha_0)} = 0 \tag{2.12}$$

Since the product must be zero, we have two cases for every data point i:

- **Case 1:** $\lambda_i = 0$. The point is correctly classified and lies somewhere "safe" (not on the margin). These points effectively vanish from our solution.
- **Case 2:** $\lambda_i > 0$. Then the bracket term *must* be zero:

$$y_i(\vec{\alpha}^T \vec{x}_i + \alpha_0) = 1 \tag{2.13}$$

  These points lie exactly on the margin boundaries. We call them **Support Vectors**.

Conclusion 1 (Sparsity): The weight vector $\vec{\alpha}$ (see 2.3.1) depends *only* on the support vectors (where $\lambda_i > 0$):

$$\vec{\alpha} = \sum_{i=1}^{m} \lambda_i y_i \vec{x}_i = \sum_{i \in SV} \lambda_i y_i \vec{x}_i \tag{2.14}$$

Conclusion 2 (The Decision Rule): We can now write the classification function for a new point $\vec{u}$ purely in terms of training samples and their inner products:

$$f(\vec{u}) = \vec{\alpha}^T \vec{u} + \alpha_0 = \underbrace{\left(\sum_{i=1}^{m} \lambda_i y_i \vec{x}_i\right)^T}_{\vec{\alpha}^T} \vec{u} + \alpha_0 = \sum_{i=1}^{m} \lambda_i y_i (\vec{x}_i^T \vec{u}) + \alpha_0 \tag{2.15}$$

This form is crucial because it allows us to apply the **Kernel Trick** in the next section.

2.4 Feature Transform & The Kernel Trick

Both hard and soft margin SVMs can only generate a **linear decision boundary**. This has serious limitations:

- Non-linearly separable data cannot be classified.
- Noisy data can cause problems.
- The formulation deals with vectorial data only.

2.4.1 Mapping to a Higher Dimensional Space

To solve this, we map the data into a richer, higher-dimensional feature space using a non-linear transformation Φ:

$$\Phi : \mathbb{R}^d \to \mathbb{R}^D \quad (D \gg d) \tag{2.16}$$

such that the features $\Phi(\vec{x}_i)$ become linearly separable in that new space.

Example (Quadratic Decision Boundary): Assume a 2D decision boundary defined by a quadratic function:

$$f(\vec{x}) = \alpha_0 + \alpha_1 x_1^2 + \alpha_2 x_2^2 + \alpha_3 x_1 x_2 + \alpha_4 x_1 + \alpha_5 x_2$$

This is non-linear in the original space $\vec{x} = (x_1, x_2)^T$. However, we can define a transformation $\Phi(\vec{x})$:

$$\Phi(\vec{x}) = (1, x_1^2, x_2^2, x_1 x_2, x_1, x_2)^T \tag{2.17}$$

Now, the decision function becomes linear in the transformed space: $f(\vec{x}) = \vec{\alpha}^T \Phi(\vec{x})$.

2.4.2 Applying the Transform to the Dual Problem

Because we reformulated the SVM into its **Dual Representation**, we don't need to explicitly calculate the weight vector $\vec{\alpha}$ in the high-dimensional space. The decision boundary can be rewritten using inner products of the transformed features:

$$f(\vec{x}) = \sum_{i=1}^{m} \lambda_i y_i \langle \Phi(\vec{x}_i), \Phi(\vec{x}) \rangle + \alpha_0 \tag{2.18}$$

The optimization problem changes accordingly:

Dual Optimization with Feature Transform

$$\text{maximize} \quad -\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle + \sum_i \lambda_i$$

subject to $\quad \vec{\lambda} \succeq 0, \quad \sum \lambda_i y_i = 0$

Now we have a linear decision boundary in a higher dimensional space!

2.4.3 Kernel Functions

Computing the explicit transformation $\Phi(\vec{x})$ can be computationally very expensive (or even impossible for infinite dimensions). However, since the algorithm **only depends on the inner product**, we can abstract this using a **Kernel Function**:

$$k(\vec{x}, \vec{x}') = \langle \Phi(\vec{x}), \Phi(\vec{x}') \rangle \tag{2.19}$$

Using kernel functions, we avoid modeling the transformation $\Phi(\vec{x})$ explicitly but still get the same result.

Typical Kernel Functions:

- **Linear:** $k(\vec{x}, \vec{x}') = \langle \vec{x}, \vec{x}' \rangle$
- **Polynomial:** $k(\vec{x}, \vec{x}') = (\langle \vec{x}, \vec{x}' \rangle + 1)^d$ (The example above corresponds to $d = 2$)
-
- **Radial Basis Function (RBF) / Gaussian:**

$$k(\vec{x}, \vec{x}') = \exp\left(-\frac{\|\vec{x} - \vec{x}'\|^2}{2\sigma^2}\right) \tag{2.20}$$

- **Sigmoid:** $k(\vec{x}, \vec{x}') = \tanh(a\vec{x}^T\vec{x}' + b)$