Lecture 4: More on classification

谢丹
清华大学数学系

September 29, 2025

Classification Problem

- **Goal**: Assign input data $\mathbf{x}$ to one of K classes
- **Input**: Feature vector $\mathbf{x} \in \mathbb{R}^D$
- **Output**: Class label $y \in \{1, 2, \ldots, K\}$
- **Approaches**:
  - Discriminant functions
  - Generative models
  - Discriminative models (e.g., Logistic Regression)

Discriminant Functions

Definition

A function $f_k(\mathbf{x})$ for each class k that directly maps input $\mathbf{x}$ to class assignments:

$$y = \arg\max_k f_k(\mathbf{x})$$

Linear Discriminant:

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k$$

Non-linear Discriminant:

$$f_k(\mathbf{x}) = \phi(\mathbf{w}_k^T \mathbf{x} + b_k)$$

Key Property

Directly models decision boundaries without estimating probability distributions

Linear Discriminant Function Formulation I

Basic Form

For a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

where:

- $\mathbf{w}$: weight vector
- b: bias term
- $\mathbf{x}$: input feature vector

Classification Rule

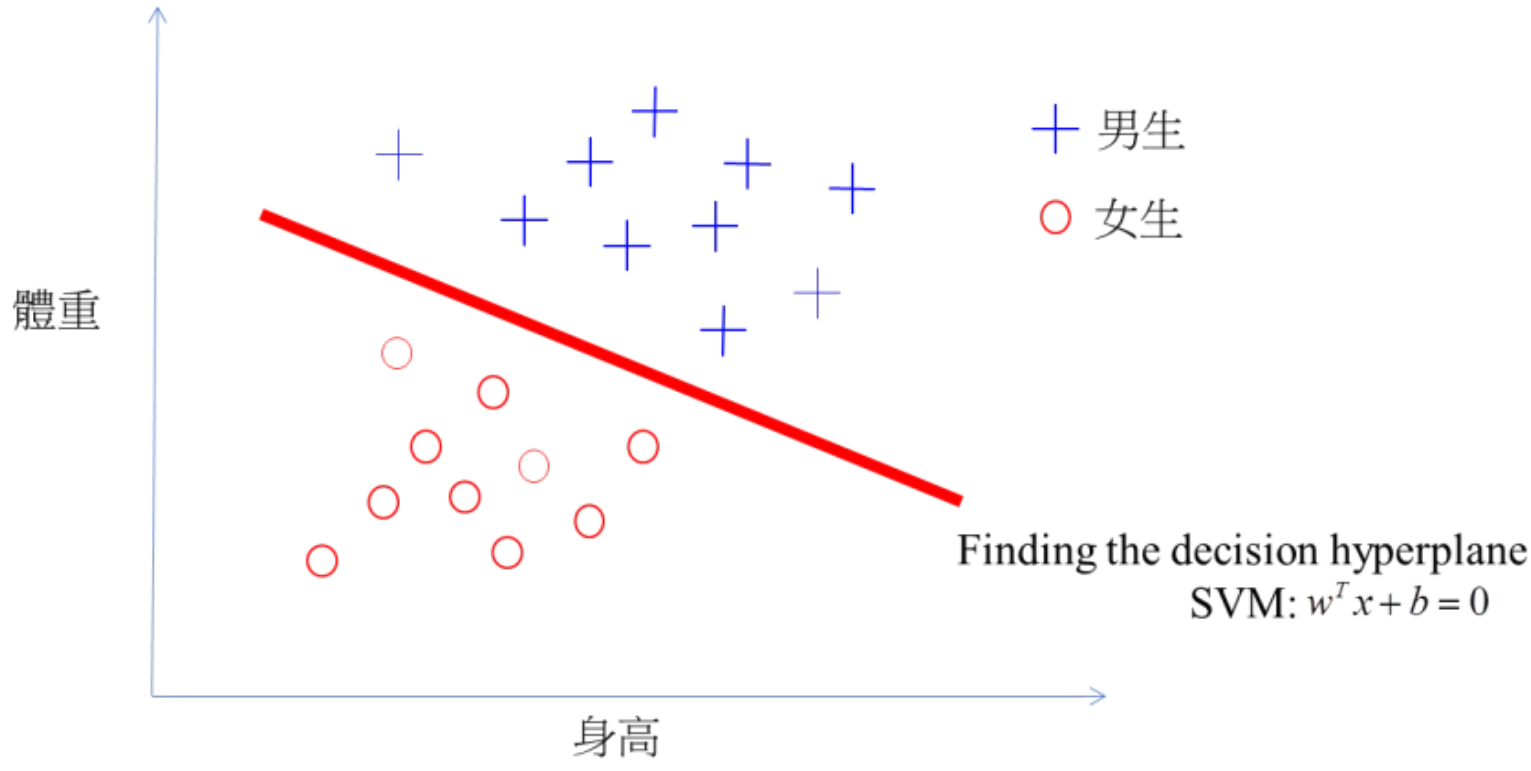
$$y = \begin{cases} +1 & \text{if } f(\mathbf{x}) \geq 0 \\ -1 & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

Linear Discriminant Function Formulation II

Multi-class Extension

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k, \quad y = \arg\max_k f_k(\mathbf{x})$$

Namely, the assignment for the class is given by the function with maximal value.


男生
女生
體重
身高
Finding the decision hyperplane
SVM: $w^T x + b = 0$

Method 1: Least Squares Approach I

Objective Function

Minimize sum of squared errors:

$$J(\mathbf{w}) = \sum_{i=1}^{N} (y_i - (\mathbf{w}^T \mathbf{x}_i + b))^2$$

Matrix Formulation

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Closed-form Solution

Method 1: Least Squares Approach II

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- Requires $\mathbf{X}^T\mathbf{X}$ to be invertible
- Sensitive to outliers
- Computationally efficient for small datasets

Method 2: Perceptron Loss Function I

The driving force behind perceptron learning

Definition

The perceptron uses a **hinge loss** function defined as:

$$L(\mathbf{w}) = \sum_{i \in M} -y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

where:

- M: set of misclassified samples (prediction is not the same as the true observed)
- $y_i \in \{-1, +1\}$: true label
- $\mathbf{w}$: weight vector
- b: bias term
- $\mathbf{x}_i$: input features

Method 2: Perceptron Loss Function II

The driving force behind perceptron learning

Key Properties

- **Convex**: Guarantees convergence
- **Piecewise linear**: Simple gradients
- **Zero for correct classifications**
- **Positive for misclassifications**

Gradient

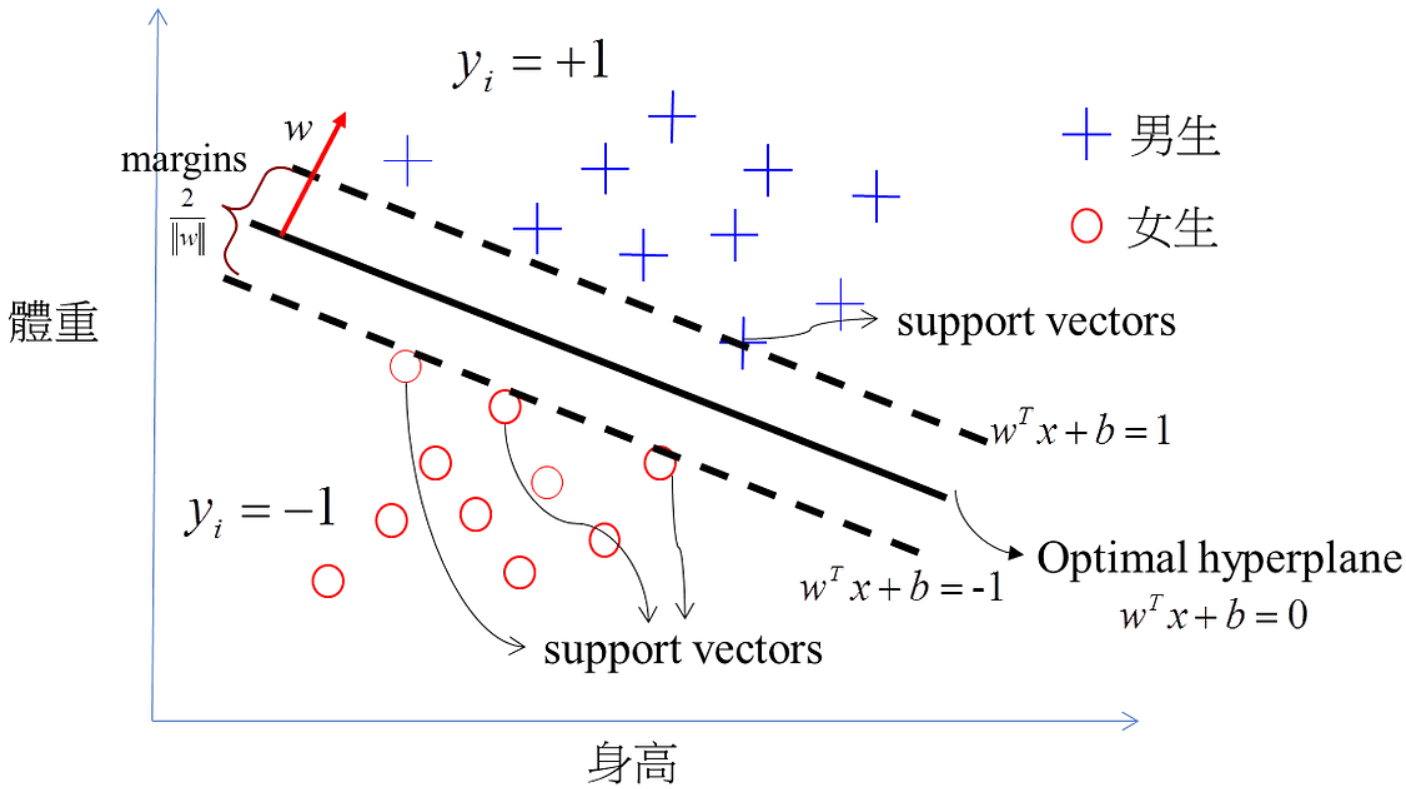
$$\nabla L(\mathbf{w}) = \sum_{i \in M} -y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = \sum_{i \in M} -y_i$$

Leads to the update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

Method 3: Support Vector Machines (SVM)



Transforming SVM into a Programming Problem I

From geometric intuition to optimization formulation

Original Geometric Problem

Maximize the margin: $\max \frac{2}{\|\mathbf{w}\|}$

Subject to: $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ for all i (linear separable)

Step 1: Equivalent Reformulation

Instead of maximizing $\frac{2}{\|\mathbf{w}\|}$, minimize $\|\mathbf{w}\|$:

$$\min \|\mathbf{w}\| \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{w}_i + b) \geq 1$$

Step 2: Convex Optimization Form

Transforming SVM into a Programming Problem II

From geometric intuition to optimization formulation

For computational convenience, use squared norm:

$$\min \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

- Convex objective function
- Linear constraints
- Quadratic Programming (QP) problem

Step 3: Primal QP Formulation

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

subject to:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n$$

Properties

- Convex objective
- Linear constraints
- Global optimum guaranteed

Transforming SVM into a Programming Problem III

From geometric intuition to optimization formulation

Step 4: Practical Implementation

- Use QP solvers (CVXOPT, MOSEK)
- Or specialized SVM libraries (LIBSVM, scikit-learn)
- Handle large datasets with optimization techniques

Soft-Margin SVM I

Soft-Margin SVM: Handling Noise and Overlap

Real data is rarely perfectly separable. We introduce slack variables ξ_i to allow misclassifications.

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i \end{aligned}$$

Parameter C controls the trade-off between a large margin and classifying points correctly.

Multi-class Training Strategies

One-vs-Rest (OvR)

- Train K binary classifiers
- Each separates one class from all others
- Final: $\arg\max_k f_k(\mathbf{x})$

One-vs-One (OvO)

- Train $\frac{K(K-1)}{2}$ classifiers
- Each separates one pair of classes
- Final: majority voting

Generative Models

Bayesian Approach

Model the joint distribution $p(\mathbf{x}, y)$ using:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

- **Class prior**: $p(y)$ - probability of each class
- **Class-conditional density**: $p(\mathbf{x}|y)$ - distribution of features given class
- **Posterior**: $p(y|\mathbf{x})$ - probability of class given features

Examples

- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Naive Bayes classifiers

Linear Discriminant Analysis (LDA)

Assumptions

- Gaussian class-conditional densities:
  $p(\mathbf{x}|y = k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$
- Shared covariance matrix across classes
- Equal covariance for all classes

$$\log p(y = k|\mathbf{x}) \propto -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) + \log \pi_k$$

Decision Boundary

Linear in $\mathbf{x}$ due to shared covariance matrix:

$$(\mathbf{x} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) = (\mathbf{x} - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_2)$$

Quadratic Discriminant Analysis (QDA)

Assumptions

- Gaussian class-conditional densities
- Class-specific covariance matrices
- $p(\mathbf{x}|y = k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

$$\log p(y = k|\mathbf{x}) \propto -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log \pi_k$$

Decision Boundary

Quadratic in $\mathbf{x}$ due to different covariance matrices:

$$(\mathbf{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)+\log|\boldsymbol{\Sigma}_1| = (\mathbf{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)+\log|\boldsymbol{\Sigma}_2|$$

What is Naive Bayes?

Definition

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with a strong (naive) independence assumption between features.

- **Simple** yet powerful
- **Fast** training and prediction
- **Probabilistic** outputs
- Works well with high-dimensional data

Why "Naive"?

The Naive Assumption

Features are conditionally independent given the class label:

$$P(X_1, X_2, \ldots, X_d|Y) = P(X_1|Y) \cdot P(X_2|Y) \cdots P(X_d|Y)$$

Real-world Example

- Classifying emails as spam/ham
- Words like "free" and "money" may be correlated
- Naive Bayes assumes they're independent given "spam"
- Surprisingly, this often works well in practice!

Naive Bayes Probability Model

Complete Formula

For features $X_1, X_2, \ldots, X_d$ and class Y:

$$P(Y|X_1, \ldots, X_d) = \frac{P(Y) \prod_{j=1}^{d} P(X_j|Y)}{P(X_1, \ldots, X_d)}$$

Classification Rule

We predict the class with highest probability:

$$\hat{y} = \arg\max_y P(y) \prod_{j=1}^{d} P(x_j|y)$$

- $P(X_1, \ldots, X_d)$ is constant for all classes
- We can ignore it for comparison

Gaussian Naive Bayes

For Continuous Features

Assumes features follow normal distribution:

$$P(X_j|Y = y_k) = \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} \exp\left(-\frac{(x_j - \mu_{jk})^2}{2\sigma_{jk}^2}\right)$$

Parameter Estimation

- $\mu_{jk} = \frac{1}{n_k} \sum_{i:y_i=y_k} x_j^{(i)}$
- $\sigma_{jk}^2 = \frac{1}{n_k} \sum_{i:y_i=y_k} (x_j^{(i)} - \mu_{jk})^2$
- n_k: number of samples in class y_k

Multinomial Naive Bayes

For Discrete Counts

Commonly used for text classification:

$$P(X_j|Y = y_k) = \frac{\text{count}(X_j, Y = y_k) + \alpha}{\sum_{l=1}^{d} \text{count}(X_l, Y = y_k) + \alpha d}$$

- α: Smoothing parameter
- Prevents zero probabilities
- Laplace smoothing when $\alpha = 1$

Example

Word counts in documents:

Word	Spam Count
free	150
money	120
...	...

Bernoulli Naive Bayes

For Binary Features

Models presence/absence of features:

$$P(X_j|Y = y_k) = P(j|y_k)^{x_j}(1 - P(j|y_k))^{1-x_j}$$

Application

- $x_j = 1$ if feature j is present
- $x_j = 0$ if feature j is absent
- Useful for document classification with binary word presence

Training Algorithm

Step 1: Estimate Priors

$$P(Y = y_k) = \frac{\text{number of samples in class } y_k}{\text{total samples}}$$

Step 2: Estimate Likelihoods

- Gaussian: Compute mean and variance for each feature per class
- Multinomial: Compute frequency counts for each feature per class
- Bernoulli: Compute probability of feature presence per class

Prediction Algorithm

Step 1: Compute Class Probabilities

For each class y_k:

$$P(y_k|\mathbf{x}) \propto P(y_k) \prod_{j=1}^{d} P(x_j|y_k)$$

Step 2: Handle Numerical Issues

Use log probabilities to avoid underflow:

$$\log P(y_k|\mathbf{x}) = \log P(y_k) + \sum_{j=1}^{d} \log P(x_j|y_k)$$

Step 3: Make Prediction

$$\hat{y} = \arg\max_{y_k} \log P(y_k|\mathbf{x})$$

Advantages and Limitations

Advantages

- Fast training and prediction
- Works well with high dimensions
- Handles continuous and discrete data
- Provides probability estimates
- Robust to irrelevant features

Limitations

- Strong independence assumption
  item Can be outperformed by more complex models
- Zero-frequency problem (needs smoothing)
- Not ideal for complex feature interactions

When to Use Naive Bayes

- Text classification (spam detection, sentiment analysis)
- Recommendation systems
- Multi-class prediction
- When training data is limited

Discriminate model: Logistic Regression

Discriminative Approach

Directly model posterior probability $p(y|\mathbf{x})$ without modeling $p(\mathbf{x}|y)$

Binary case:

$$p(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Multiclass case (Softmax):

$$p(y=k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T\mathbf{x} + b_k)}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^T\mathbf{x} + b_j)}$$

Advantage

Makes fewer assumptions about data distribution compared to generative models

Maximum Likelihood Estimation

Maximize log-likelihood of training data:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} [y_i \log p(y_i|\mathbf{x}_i, \mathbf{w}) + (1 - y_i) \log(1 - p(y_i|\mathbf{x}_i, \mathbf{w}))]$$

Gradient for Binary Case

$$\nabla_{\mathbf{w}} \mathcal{L} = \sum_{i=1}^{N} (y_i - p(y_i = 1|\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

Optimization Methods

- Gradient Descent
- Newton-Raphson method
- Stochastic Gradient Descent

Comparison of Approaches

Method	Type	Boundary	Assur
Discriminant Functions	Non-probabilistic	Flexible	N
LDA	Generative	Linear	Gaussian, sha
QDA	Generative	Quadratic	Gaussian, diffe
Logistic Regression	Discriminative	Linear	Linear decis

Table: Comparison of classification methods

- **Generative**: Better with small datasets, can generate samples, handles missing data
- **Discriminative**: Often better performance with large datasets, focuses on decision boundary

When to Use Each Method I

Discriminant Functions

- When probabilistic interpretation is not needed
- When computational efficiency is critical
- For simple, interpretable models

Generative Models (LDA/QDA)

- When dataset is small
- When you want to generate new samples
- When features follow approximately Gaussian distribution
- When you need to handle missing data

Logistic Regression

When to Use Each Method II

- For large datasets
- When you want well-calibrated probabilities
- When Gaussian assumptions are violated
- As a baseline for more complex models

Addressing Non-Linearity

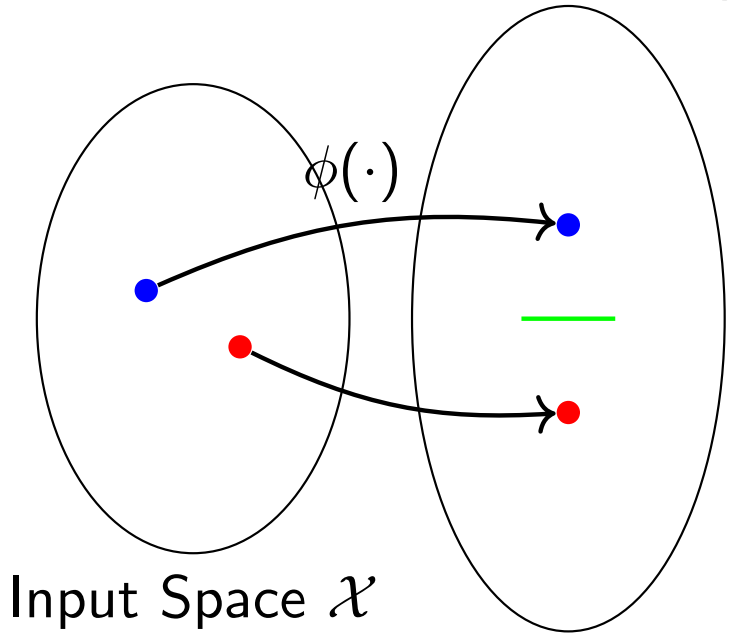
We can address non-linear effects through two primary approaches:

1. **Explicit Feature Transformation**
   - Transform the input features using a set of **basis functions** ($\phi(\mathbf{x})$).
   - The target variable may be transformed via a **link function** ($g(\mu)$).
   - The model remains linear in the parameters: $\mathbb{E}[y] = \mathbf{w}^T \phi(\mathbf{x})$.
2. **Implicit Representation Learning**
   - The mean is a **complex, non-linear function** of the inputs and parameters.
   - This function is **learned directly** by a model like a neural network.

Next, we will explore a powerful alternative: methods based on **kernels** and, more generally, **Gaussian Processes**.

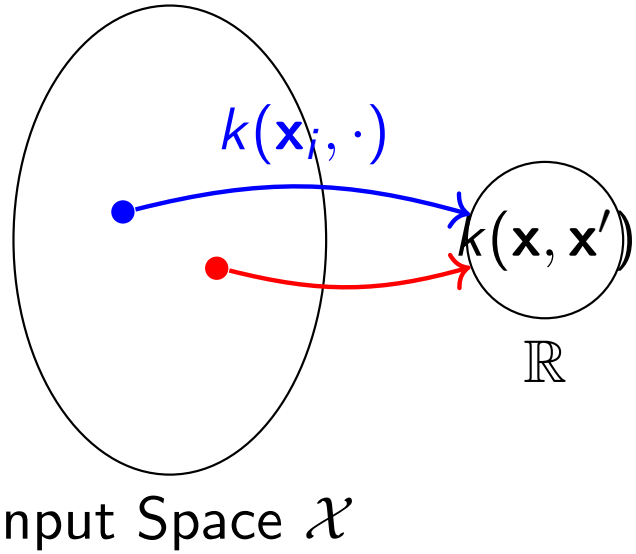
The Feature Map ϕ and The Kernel Trick

Explicit Feature Mapping



Compute $\phi(\mathbf{x})$, then $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$
Can be computationally expensive.

The Kernel Trick



Skip the transformation! Compute the inner product directly in the input space via the kernel function.

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

The Kernel Trick Defined

If an algorithm can be formulated solely in terms of inner products, we can make it non-linear by replacing every $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with $k(\mathbf{x}_i, \mathbf{x}_j)$.

Common Kernel Functions I

Linear Kernel

The simplest kernel, no mapping. Equivalent to standard dot product.

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

Polynomial Kernel

Creates polynomial feature maps of degree d. Learns polynomial decision boundaries.

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

Radial Basis Function (RBF) / Gaussian Kernel

Common Kernel Functions II

The most popular kernel. Implicitly maps data to an infinite-dimensional feature space. Highly flexible.

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

γ controls the influence of a single training example (smoothness of the boundary).

Why Does It Work? Mercer's Theorem

Mercer's Condition

For a function k to be a valid kernel, it must be:

1. **Symmetric:** $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
2. **Positive Semi-Definite:** The kernel matrix $\mathbf{K}$ (aka Gram matrix), where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, must be PSD for any set of inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.

Theorem (Mercer's Theorem)

Any function k satisfying Mercer's condition corresponds to an inner product in some feature space.

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

Practical Implication: We don't need to know ϕ! We can just choose any valid kernel function k and be guaranteed that it corresponds to some complex feature space.

Applications and Summary

Kernelized Algorithms

Many classic algorithms have kernelized versions:

- **Support Vector Machines (SVM):** The classic application. Kernel SVM is incredibly powerful.
- **Kernel Ridge Regression:** For non-linear regression.
- **Gaussian Processes:** A full Bayesian approach using kernels.

Summary: The Power of Kernels

- **Efficiency:** Work in high-dimensional spaces without the computational cost.
- **Flexibility:** Model complex, non-linear relationships.
- **Generality:** Apply ML to non-vectorial data (e.g., graphs, sequences) by designing a kernel that measures similarity.

Kernels separate the task of designing the feature space from the task of learning.

Kernel Ridge Regression I

Problem Formulation

Ridge regression minimizes the penalized least squares objective:

$$J(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2$$

where:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the design matrix
- $\mathbf{y} \in \mathbb{R}^n$ is the target vector
- $\mathbf{w} \in \mathbb{R}^d$ is the weight vector
- $\lambda \geq 0$ is the regularization parameter

Derivation

Kernel Ridge Regression II

$$\begin{aligned}\nabla_{\mathbf{w}} J(\mathbf{w}) &= -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} \\ 0 &= -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w} + \lambda\mathbf{w} \\ \mathbf{X}^\top\mathbf{y} &= (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\mathbf{w}\end{aligned}$$

Closed-Form Solution

$$\boxed{\hat{\mathbf{w}}_{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}}$$

- The matrix $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ is always invertible for $\lambda > 0$
- Regularization improves numerical stability compared to OLS
- Solution reduces to OLS when $\lambda = 0$

Ridge Regression: Dual Formulation I

Primal Problem

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{w} \in \mathbb{R}^d$, $\lambda > 0$.

Representer Theorem Insight

The solution can be expressed as:

$$\mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ are dual variables.

Dual Formulation Derivation

Ridge Regression: Dual Formulation II

Substitute $\mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha}$ into the primal:

$$\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \|\mathbf{y} - \mathbf{X}\mathbf{X}^\top \boldsymbol{\alpha}\|^2 = \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2$$
$$\|\mathbf{w}\|^2 = \boldsymbol{\alpha}^\top \mathbf{X}\mathbf{X}^\top \boldsymbol{\alpha} = \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ is the Gram matrix.

Dual Problem

Ridge Regression: Dual Formulation III

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \frac{\lambda}{2}\boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}$$

Solution:

$$\boxed{\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}}$$

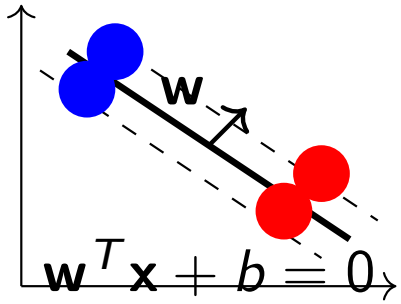
Prediction for new point $\mathbf{x}$:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} = \boldsymbol{\alpha}^\top \mathbf{X}\mathbf{x} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^\top \mathbf{x}$$

Kernel trick: replace the trivial Kernel by a different Kernel function $K(x, y)$.

Kernel support vector machine

Goal: Find hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ that separates the classes ($y_i \in \{-1, +1\}$) with max margin.



The distance from a point to the hyperplane is:

$$\frac{|\mathbf{w}^T\mathbf{x}_i + b|}{\|\mathbf{w}\|} = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

We want to maximize the margin $M = \frac{2}{\|\mathbf{w}\|}$. This leads to the optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i \end{aligned}$$

This is a Quadratic Programming (QP) problem.

Soft-Margin SVM and The Kernel Trick I

Soft-Margin SVM: Handling Noise and Overlap

Real data is rarely perfectly separable. We introduce slack variables ξ_i to allow misclassifications.

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i \end{aligned}$$

Parameter C controls the trade-off between a large margin and classifying points correctly.

Soft Margin SVM: Dual Formulation I

Primal Problem (Soft Margin SVM)

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n \\ & \xi_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

where ξ_i are slack variables and $C > 0$ is the regularization parameter.

Lagrangian Function

Soft Margin SVM: Dual Formulation II

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i$$

with $\alpha_i \geq 0$, $\beta_i \geq 0$ as Lagrange multipliers.

Dual Problem

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boxed{\mathbf{x}_i^\top \mathbf{x}_j} \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \end{aligned}$$

Soft Margin SVM: Dual Formulation III

Kernel trick: replace the trivial Kernel by a different Kernel function $K(x, y)$.

Beyond Parametric Models

Parametric models (e.g., Linear Regression):

$$y = \mathbf{w}^\top \phi(\mathbf{x}) + \epsilon$$

Learn parameters $\mathbf{w}$. Limited flexibility.

Non-parametric models (e.g., Gaussian Processes):

- Don't learn a fixed set of parameters $\mathbf{w}$.
- Instead, define a probability distribution over possible functions $f(\mathbf{x})$.
- The complexity grows with the amount of data.

The Core Idea of a Gaussian Process

Definition

A Gaussian Process is a collection of random variables, any finite number of which have a consistent joint Gaussian distribution.

Analogy:

- A Gaussian distribution: distribution over vectors.
- A Gaussian process: distribution over functions.

A GP is completely specified by its:

- Mean function: $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$
- Covariance function (kernel):
  $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$

We write: $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$.

The Heart of the GP: The Kernel

The kernel $k(\mathbf{x}, \mathbf{x}')$ defines the covariance between the function values $f(\mathbf{x})$ and $f(\mathbf{x}')$. It encodes our prior assumptions about the

function's properties.

The Prior

We assume a prior over functions. Often we set the mean function to zero: $m(\mathbf{x}) = 0$.

$$f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$$

For any finite set of points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the function values $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^\top$ have a multivariate Gaussian prior:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$$

where $\mathbf{K}$ is the $N \times N$ kernel matrix with entries $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

Noisy Observations

We observe noisy data:

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$$

The joint distribution of the observed targets $\mathbf{y}$ and the latent function values $\mathbf{f}$ is:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 I & \mathbf{K} \\ \mathbf{K} & \mathbf{K} \end{bmatrix}\right)$$

The Posterior Predictive Distribution

For a new test point $\mathbf{x}_*$, we want the predictive distribution $p(f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*)$.

Key Formulas

The posterior predictive distribution is Gaussian:

$$p(f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) \sim \mathcal{N}(\bar{f}_*, \mathbb{V}[f_*])$$

Predictive Mean:

$$\bar{f}_* = \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{y}$$

Predictive Variance:

$$\mathbb{V}[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{k}_*$$

where $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_N)]^\top$.

Applications of Gaussian Processes

Regression

- Small datasets (¡10,000 points)
- Where uncertainty matters
- E.g., calibration, sensor data

Bayesian Optimization

- Optimizing expensive black-box functions
- GPs guide the search for the optimum

Geostatistics (Kriging)

- Interpolating spatial data
- The original application of GPs

State Space Models

- GPs can be used as components in more complex models

Summary

- Gaussian Processes provide a non-parametric, Bayesian approach to regression.
- They define a distribution over functions.
- The kernel encodes prior knowledge about the function's properties.
- They provide full predictive distributions (mean + uncertainty).
- The main limitation is computational complexity $\mathcal{O}(N^3)$.
- They are the method of choice for many problems where data is scarce and uncertainty quantification is crucial.

The Optimization Challenge in ML

- Machine learning often involves **constrained optimization**
- We want to minimize loss **while satisfying constraints**
- Examples:
  - SVMs: Maximize margin while classifying correctly
  - Regularization: Minimize error while keeping weights small

The Fundamental Problem

How do we efficiently solve constrained optimization problems?

The Primal Problem Formulation

General Form of Constrained Optimization

Minimize: $f(\mathbf{w})$
Subject to:

$$g_i(\mathbf{w}) \leq 0, \quad i = 1, \ldots, k \quad \text{(Inequality constraints)}$$
$$h_j(\mathbf{w}) = 0, \quad j = 1, \ldots, m \quad \text{(Equality constraints)}$$

- $\mathbf{w}$: Model parameters (weights)
- $f(\mathbf{w})$: Objective/loss function
- $g_i(\mathbf{w})$: Inequality constraints
- $h_j(\mathbf{w})$: Equality constraints

The Lagrangian Approach

Key Idea

Transform constrained problem into unconstrained problem by introducing **Lagrange multipliers**

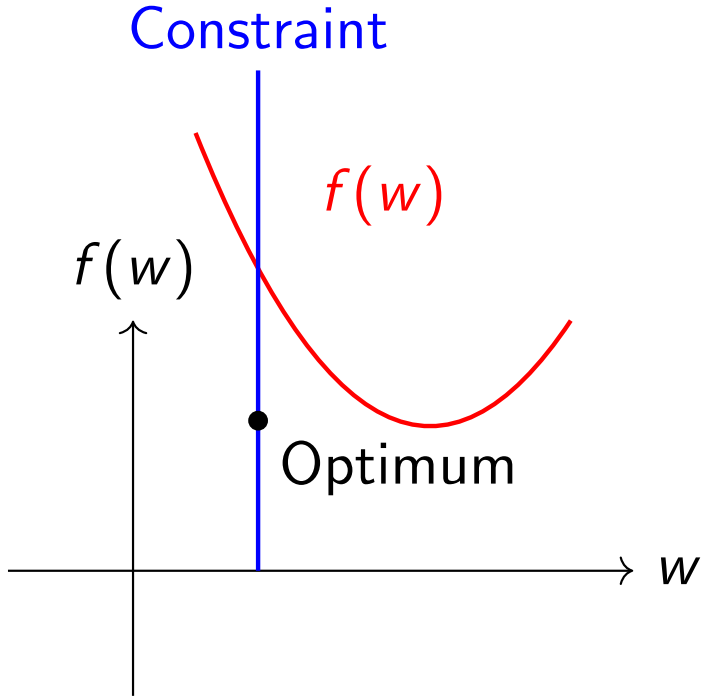
Lagrange Multipliers:

- $\alpha_i \geq 0$ for $g_i(\mathbf{w}) \leq 0$
- β_j for $h_j(\mathbf{w}) = 0$

Lagrangian Function

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{j=1}^{m} \beta_j h_j(\mathbf{w})$$

Intuition Behind Lagrange Multipliers



- Constraints define feasible region
- Lagrange multipliers act as **"prices"** for constraint violation
- Balance between objective and constraint satisfaction

From Primal to Dual

Lagrangian Dual Function

$$\mathcal{G}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Dual Problem

Maximize: $\mathcal{G}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
Subject to: $\alpha_i \geq 0, \quad i = 1, \ldots, k$

- We switched from **minimization** to **maximization**
- Constraints become much **simpler**
- Often easier to solve!

Duality Theorems

Weak Duality

For any feasible $\mathbf{w}$ and $\boldsymbol{\alpha} \geq 0$:

$$\mathcal{G}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq f(\mathbf{w})$$

The dual always provides a **lower bound** on the primal.

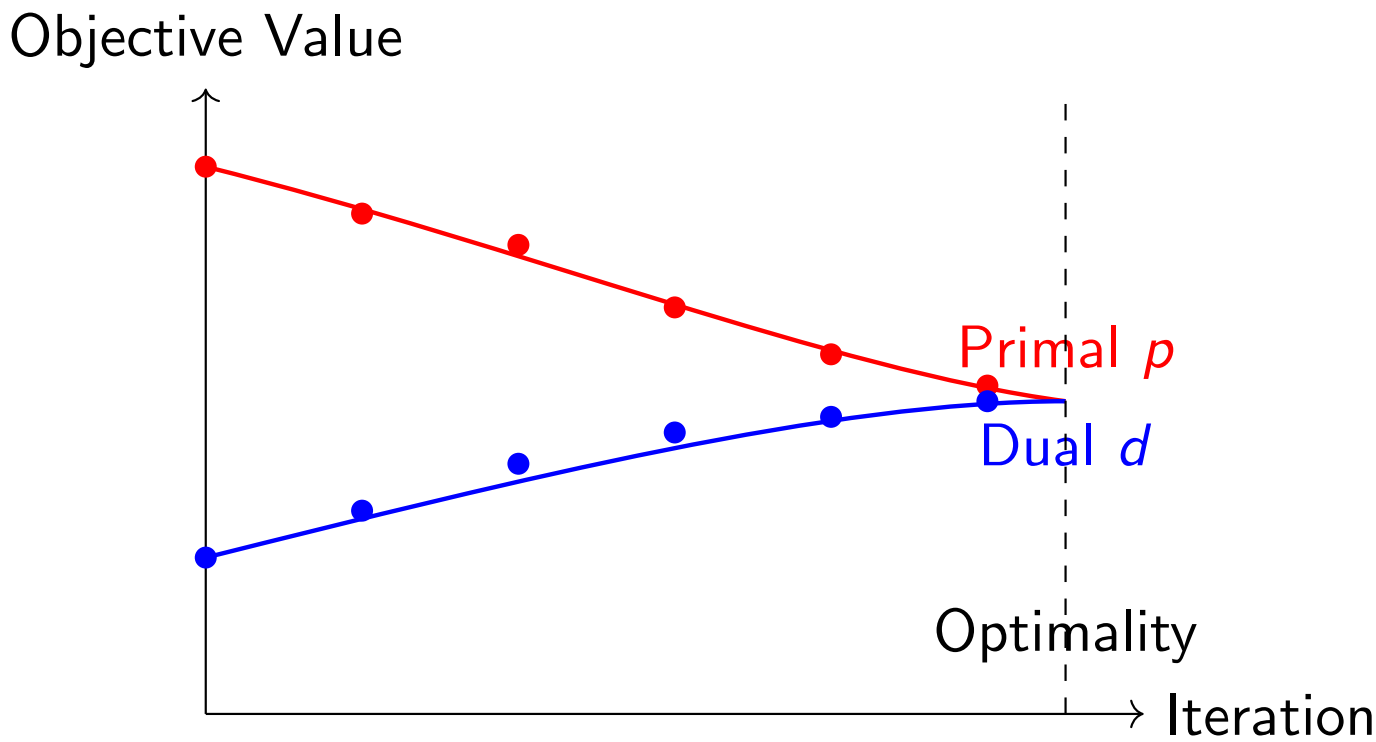
Strong Duality

Under certain conditions (convexity + Slater's condition):

$$d^* = p^*$$

where p^* is optimal primal value and d^* is optimal dual value.

Visualizing Duality



- Primal decreases toward optimum
- Dual increases toward optimum
- At optimum: $p^* = d^*$ (strong duality)

Support Vector Machines (SVMs)

SVM Primal Problem

Minimize: $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$

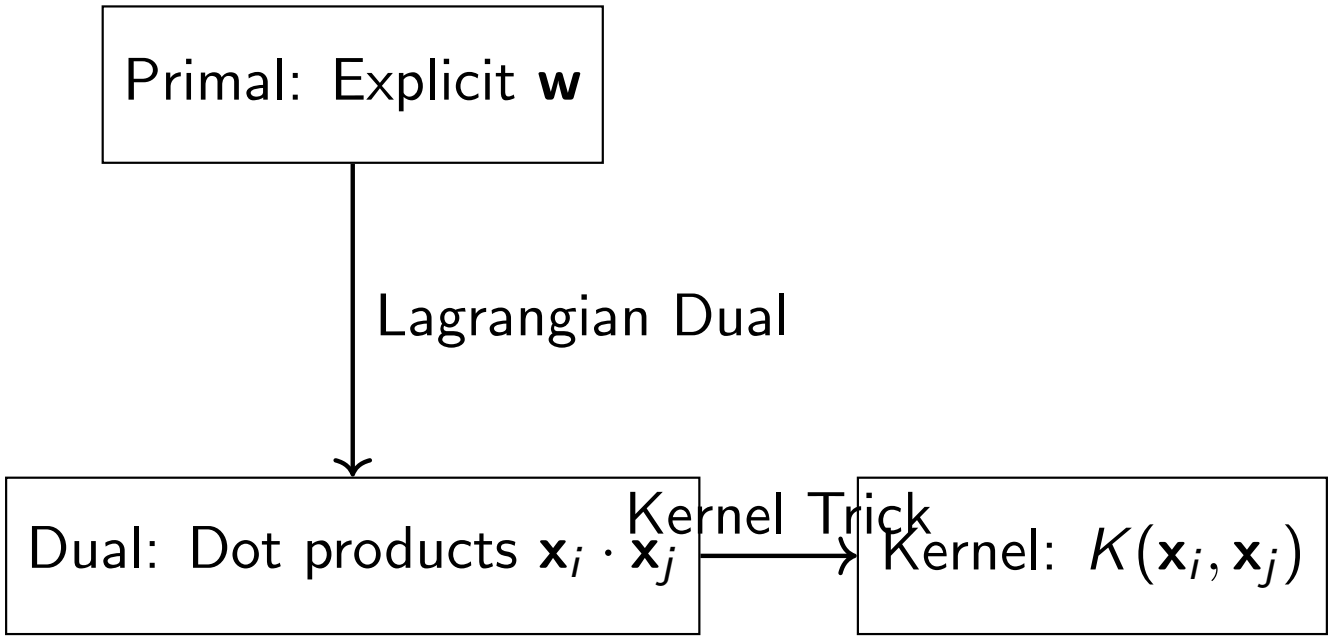
Subject to: $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0$

SVM Dual Problem

Maximize: $\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j$

Subject to: $0 \leq \alpha_i \leq C,\ \sum_{i=1}^{n}\alpha_i y_i = 0$

The Kernel Trick



- Dual formulation reveals dot products
- Enables **kernel trick**: Replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $K(\mathbf{x}_i, \mathbf{x}_j)$
- Allows non-linear decision boundaries

Other ML Applications

Regularization

- L1/L2 regularization
- Elastic net
- Constrained optimization perspective

Neural Networks

- Lagrange multipliers for constraints
- Adversarial training

Graphical Models

- Inference as optimization
- Variational methods

Fairness Constraints

- Demographic parity
- Equalized odds
- Enforced via constraints

Advantages of Dual Formulation

- **Simpler constraints**: Often just bound constraints on α_i
- **Kernel trick**: Enables non-linear models
- **Theoretical insights**: Reveals problem structure (e.g., support vectors)
- **Numerical stability**: Often better conditioned
- **Feature space interpretation**: Works in high-dimensional spaces implicitly

Limitations and Considerations

Computational Issues

- Number of variables = number of constraints
- Can be large for big datasets item May need specialized solvers

Theoretical Limitations

- Strong duality not always guaranteed
- Duality gap may exist

Practical Challenges

- Recovering primal solution from dual
- Interpreting dual variables
- Implementation complexity

When to Use Dual?

- Constraints are complex
- Kernel methods needed
- Problem has special structure

Summary

- Lagrangian dual transforms **constrained** $\rightarrow$ **unconstrained** problems
- Provides **lower bounds** via weak duality
- Under nice conditions: **strong duality** ($p^* = d^*$)
- Crucial for **kernel methods** (SVMs)
- Reveals **problem structure** and insights
- Enables powerful ML algorithms

Key Takeaway

The dual perspective often reveals structure and enables methods that would be impossible in the primal formulation.