Support Vector Machines and k-Nearest Neighbors

Based on
NYU Courant Mehryar Mohri's *Foundations of Machine Learning*
UoLiverpool Dom Richards's *COMP336/COMP529 Big Data Analytics Lecture Note*
written

1 Support Vector Machines (SVM)

1.1 Strict Definition of Linear Separability

Consider a training dataset

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \quad \mathbf{x}_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\}.$$

We say that this dataset is *linearly separable* in $\mathbb{R}^d$ if there exists a hyperplane parameterized by $\mathbf{w} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$ such that for every $i \in \{1, \ldots, n\}$,

$$y_i\big(\mathbf{w}^\top \mathbf{x}_i + b\big) > 0.$$

More strictly, if there exists some positive margin $\gamma > 0$ such that for all i,

$$y_i\big(\mathbf{w}^\top \mathbf{x}_i + b\big) \geq \gamma,$$

then γ is called the *geometric margin*. Under linear separability, we aim to find a hyperplane that separates the data with the largest geometric margin.

1.2 Hard-Margin Linear SVM Optimization

In the ideal case of linear separability, the SVM solves the following **hard-margin** problem:

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

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n.$$

Since the geometric margin is $\frac{2}{\|\mathbf{w}\|}$, minimizing $\frac{1}{2}\|\mathbf{w}\|^2$ is equivalent to maximizing that margin.

1.2.1 Lagrange Multipliers and the Dual Problem

To solve this constrained problem, we form the Lagrangian:

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

where $\alpha_i \geq 0$ are Lagrange multipliers. Taking partial derivatives with respect to $\mathbf{w}$ and b and setting them to zero, we obtain

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i,$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Plugging these back into the Lagrangian, we get the **dual problem**:

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

This is a **quadratic programming (QP)** problem that can be solved efficiently with standard optimization methods.

1.2.2 KKT Conditions

The **Karush-Kuhn-Tucker (KKT) conditions** are necessary for optimality of the solution $(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*)$. They include:

1. *Primal feasibility*:
$$y_i\big(\mathbf{w}^\top \mathbf{x}_i + b\big) - 1 \;\; \geq \;\; 0, \quad \forall\, i;$$

2. *Dual feasibility*:
$$\alpha_i \;\; \geq \;\; 0, \quad \forall\, i;$$

3. *Complementary slackness*:
$$\alpha_i \, \Big[\, y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \, \Big] = 0, \quad \forall\, i;$$

4. *Stationarity*:
$$\mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

From the complementary slackness condition, if $\alpha_i > 0$, then $y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 = 0$. Such points are called **support vectors**. If $\alpha_i = 0$, then the point is not necessarily on the margin boundary.

1.3 Soft Margin and Regularization

Real-world data are often not perfectly linearly separable. To allow misclassification, one introduces slack variables $\xi_i \geq 0$ and a regularization parameter $C > 0$. The **soft-margin** SVM solves:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

Here, C controls the trade-off between maximizing the margin and penalizing classification errors.

1.4 Kernel Functions and the Kernel Trick

1.4.1 Motivation: High-Dimensional Mapping

When data are not linearly separable in $\mathbb{R}^d$, one can construct a mapping

$$\phi : \mathbb{R}^d \ \rightarrow \ \mathcal{H},$$

mapping each $\mathbf{x} \in \mathbb{R}^d$ into a (potentially high- or infinite-dimensional) Hilbert space $\mathcal{H}$. A linear separation in $\mathcal{H}$ may be easier to achieve. The SVM then seeks a hyperplane

$$\mathbf{w}^\top \phi(\mathbf{x}) + b = 0$$

in $\mathcal{H}$. Computing $\phi(\mathbf{x})$ explicitly, however, can be extremely expensive or infeasible in high dimensions.

1.4.2 Kernel Function

If there exists a kernel function

$$k(\mathbf{x}_i, \mathbf{x}_j) \ = \ \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j),$$

then the SVM *dual formulation* only needs values of $k(\mathbf{x}_i, \mathbf{x}_j)$ instead of explicit $\phi(\cdot)$. This is the famous **kernel trick**, avoiding direct computation of ϕ.

1.4.3 Common Kernel Functions

Linear Kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j.$$

The simplest kernel, corresponding to $\phi(\mathbf{x}) = \mathbf{x}$. Often used for linear or high-dimensional sparse data.

Polynomial Kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \big(\mathbf{x}_i^\top \mathbf{x}_j + c\big)^p,$$

where $c \geq 0$ and p is the polynomial degree. Higher-degree polynomials can capture more complex boundaries. Kernel trick mitigates the curse of dimensionality by avoiding explicit feature expansion.

Gaussian (RBF) Kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\Big).$$

Corresponds to an infinite-dimensional mapping, often yields smooth and local decision boundaries, very popular in practice.

Sigmoid Kernel (Neural Network Kernel)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh\big(\alpha\, \mathbf{x}_i^\top \mathbf{x}_j + c\big).$$

Similar to a two-layer neural network activation. Not guaranteed to be a Mercer kernel for all parameter settings.

Once a kernel is chosen, the dual problem becomes

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \, y_i y_j \, k(\mathbf{x}_i, \mathbf{x}_j),$$

leading to the decision function

$$f(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{i=1}^{n} \alpha_i y_i \, k(\mathbf{x}, \mathbf{x}_i) + b\Big).$$

1.5 Reproducing Kernel Hilbert Space (RKHS)

1.5.1 Definition and Reproducing Property

Consider a Hilbert space $\mathcal{H}$ endowed with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. If there is a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that

$$k(\mathbf{x}, \mathbf{y}) = \big\langle k(\mathbf{x}, \cdot),\ k(\mathbf{y}, \cdot) \big\rangle_{\mathcal{H}}$$

and for every $\mathbf{x} \in \mathcal{X}$ and $f \in \mathcal{H}$,

$$f(\mathbf{x}) = \big\langle f,\ k(\mathbf{x}, \cdot) \big\rangle_{\mathcal{H}},$$

then $\mathcal{H}$ is called a **Reproducing Kernel Hilbert Space (RKHS)**, and k is its **reproducing kernel**.

Here,

$$k(\mathbf{x}, \cdot)$$

is a function in $\mathcal{H}$ mapping from $\mathcal{X}$ to $\mathbb{R}$, and the kernel itself can be seen as the inner product of these functions. This is known as the **reproducing property**.

1.5.2 Motivation for Introducing RKHS in SVMs

- **Unified theoretical framework:** RKHS provides a rigorous mathematical foundation for kernel methods. The idea of "replace inner products by a kernel function" becomes systematically justified.
- **Representer Theorem:** In many kernel-based learning methods (including SVM with squared-norm regularization), the optimal solution $f^* \in \mathcal{H}$ can be written as

  $$f^*(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \, k(\mathbf{x}_i, \mathbf{x}).$$

  Even if $\mathcal{H}$ is infinite-dimensional, the minimizer is a finite linear combination of kernels evaluated at the training points.
- **Geometric interpretation:** In an RKHS, $\|f\|_{\mathcal{H}}$ can be seen as a measure of function complexity. Minimizing $\|f\|_{\mathcal{H}}^2$ plus a loss term corresponds to balancing model complexity and empirical error, closely aligning with the margin-maximizing principle of the SVM.

1.5.3 Notation and Formulas

In the above:

- $\mathcal{X}$ is the input space, often $\mathbb{R}^d$.
- $\mathcal{H}$ is a (possibly infinite-dimensional) Hilbert space whose elements can be thought of as functions.
- $k(\cdot, \cdot)$ is the **reproducing kernel**, so for any $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$k(\mathbf{x}, \mathbf{y}) = \langle\, k(\mathbf{x}, \cdot),\ k(\mathbf{y}, \cdot)\rangle_{\mathcal{H}}.$$

- $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in $\mathcal{H}$.
- $f(\mathbf{x})$ is given by

$$f(\mathbf{x}) = \langle f,\ k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}}.$$

This shows why, in SVMs, we can rely on kernel functions instead of explicit feature maps, and why the dual solution is "sparse" in terms of kernel evaluations at training points.

2 k-Nearest Neighbors (kNN)

2.1 Algorithmic Steps

Given a training set

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^n, \quad y_i \in \{-1, +1\},$$

the **kNN** classification procedure is:

1. **Distance metric:** Commonly the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$, or other metrics (Manhattan, Chebyshev, etc.) depending on the task.
2. **Find the k neighbors:** For a query point $\mathbf{x}$, compute distances to every training sample, choose the k points closest to $\mathbf{x}$.
3. **Vote or weighted vote:** For classification, use majority vote or a weighted scheme (e.g., distance-based weighting) among the labels of those k neighbors.

2.2 A Mathematical Viewpoint

kNN is a **non-parametric method**: it does not explicitly fit parameters to the training data but relies on local data distribution at prediction time. We can view kNN as estimating the posterior probability $P(Y|\mathbf{x})$ by counting in a local neighborhood:

$$\hat{P}(y \mid \mathbf{x}) \approx \frac{1}{k} \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})} \mathbf{1}_{\{y_i = y\}},$$

where $\mathcal{N}_k(\mathbf{x})$ is the set of the k nearest neighbors to $\mathbf{x}$, and $\mathbf{1}_{\{\cdot\}}$ is the indicator function.

2.3 Distance Metrics and Weighting

In practice, the choice of distance metric can significantly affect performance. Apart from Euclidean distance, one might choose Manhattan, Chebyshev, or custom metrics. Features with different scales may require normalization. Voting can be weighted by inverse distance:

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i) + \varepsilon},$$

where ε is a small constant to avoid division by zero.

2.4 Choosing k

One typically selects the best k via **cross-validation** or similar methods. Too small k may overfit; too large k may underfit, losing the local decision boundary advantage.