CHAPTER 5

Support Vector Machines (SVMs)

Chapter 5 introduces **Support Vector Machines (SVMs)** as powerful models for classification, regression, and even outlier detection, especially strong on small–medium, complex datasets. The central idea is to find a decision boundary that maximizes the margin between classes, relying only on a subset of "support vectors" that lie on the edge of this margin and fully determine the boundary.

Linear SVM classification

For linearly separable data, a linear SVM chooses the hyperplane that leaves the **widest possible margin** ("street") between positive and negative classes, rather than just any separating line. Only points on the margin (support vectors) influence the solution; moving non–support vectors does not change the boundary. Hard-margin SVM strictly forbids margin violations, but is sensitive to nonseparable data and outliers, so **soft-margin SVM** relaxes this via a hyperparameter **C** that trades off margin width versus margin violations (smaller C → wider margin, more violations; larger C → narrower margin, fewer violations). In practice, inputs must be **scaled** (e.g., StandardScaler), because SVMs are sensitive to feature scales, otherwise the margin becomes skewed along large-scale features.

Nonlinear classification and kernels

To handle nonlinear data, SVMs either expand features explicitly (e.g., **polynomial features** with PolynomialFeatures + LinearSVC) or use the **kernel trick** via SVC. A **polynomial kernel** (e.g., degree 3 or 5 with kernel="poly", parameters degree, coef0, C) implicitly corresponds to a polynomial feature expansion without actually constructing all polynomial terms. The **Gaussian RBF kernel**

$$K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \parallel \mathbf{a} - \mathbf{b} \parallel^2\right)$$

adds similarity-based features, where **γ** controls each point's "influence radius" (small γ → smooth boundary, risk underfitting; large γ → very wiggly boundary, risk overfitting). In practice you tune **C** and **γ** (e.g., with grid search); increasing either generally makes the model more complex. Other kernels (e.g., sigmoid, string kernels) exist but linear and RBF cover most use cases; a common rule of thumb is to try a **linear SVM** first (often LinearSVC), then RBF if linear is insufficient.

SVM regression

SVMs can perform **regression** by inverting the usual objective: instead of maximizing the margin between classes, **SVR** tries to fit as many points as possible inside an ε-wide tube around a prediction function, while limiting violations outside the tube. The **ε** parameter sets tube width; C again trades off flatness vs tolerance to errors (larger C → fit data more tightly). Linear regression SVMs use LinearSVR, while nonlinear SVR uses SVR with kernels like "rbf" or "poly" and inherits the same computational scaling as SVC.

Computational aspects and when to use which class

LinearSVC (liblinear) trains linear SVMs with time roughly $O(m \cdot n)$ where m is instances and n features, and is well suited for large, high-dimensional problems (no kernel trick). SVC (libsvm) supports kernels but typically scales between $O(m^2 \cdot n)$ and $O(m^3 \cdot n)$, so it is ideal for complex but small/medium datasets, especially with sparse features. SGDClassifier with loss="hinge" approximates a linear SVM using online/stochastic learning, useful for streaming data or huge datasets that do not fit in memory. A similar relationship holds for regression: LinearSVR scales linearly, while SVR is kernelized and becomes slow as m grows.

Under the hood (high level)

Conceptually, a linear SVM finds weights **w** and bias **b** that maximize margin (minimize $\frac{1}{2} \parallel w \parallel^2$) while satisfying classification constraints; soft margin introduces slack variables and C to allow violations. The optimization can be posed as a **quadratic programming (QP)** problem in either **primal** or **dual** form; the **dual form** exposes dot products between data points, enabling the kernel trick by replacing $\mathbf{x_i}^\top \mathbf{x_j}$ with a kernel $K(\mathbf{x_i}, \mathbf{x_j})$. For online linear SVMs, an SGD-style objective with **hinge loss**

$$J(w,b) = \frac{1}{2} \parallel w \parallel^2 + C \sum_i \max(0, 1 - t_i(w^\top x_i + b))$$

is minimized iteratively, where $t_i \in \{-1, +1\}$ are class labels.