Support Vector Machines: Theory and Practice

September 10, 2025

1 Introduction

Support Vector Machines (SVMs) are margin-based learners that find a decision boundary maximizing the margin between classes. With kernels, they represent complex non-linear decision boundaries while remaining convex to optimize.

2 Theory and Formulas

For linear, soft-margin SVM in primal form, given labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi} \geq 0} \quad \tfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \tag{1}$$

$$\text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \; i = 1, \ldots, n. \tag{2}$$

In the dual, data appear only via inner products; replacing them with kernels $K(\mathbf{x}, \mathbf{x}')$ provides non-linear SVMs. The decision function is

$$f(\mathbf{x}) = \sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b, \quad \hat{y} = \operatorname{sign} f(\mathbf{x}), \tag{3}$$

where SV are support vectors with non-zero multipliers α_i. RBF kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ is a common default. Hyperparameters C (slack penalty) and γ (kernel width) control regularization and boundary complexity.

3 Applications and Tips

- **Scaling:** always scale features; SVMs are sensitive to feature scales.
- **Hyperparameters:** tune C and γ (for RBF) via cross-validation; start with $C \in [0.1, 10]$, $\gamma \in [10^{-2}, 10]$ (log grid).
- **Class imbalance:** use `class_weight=balanced` to reweight classes.
- **Probability:** `SVC` supports probability with `probability=True` (adds calibration cost); otherwise use `decision_function`.
- **Multiclass:** `SVC` uses one-vs-one internally; `LinearSVC` uses one-vs-rest.

4 Python Practice

Run the script in this chapter directory to generate figures into `figures/`.

Listing 1: Generate SVM figures

```
python gen_svm_figures.py
```

Listing 2: gen_svm_figures.py

```
"""
Figure generator for the SVM chapter.

Generates illustrative figures and saves them into the chapter's 'figures/'
folder next to this script, regardless of current working directory.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn

Install (if needed):
  pip install numpy matplotlib scikit-learn

This script avoids newer or experimental APIs for broader compatibility.
"""
from __future__ import annotations

import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    from sklearn.datasets import make_moons, make_classification
    from sklearn.svm import SVC
except Exception:
    raise SystemExit(
        "Missing scikit-learn. Please install with: pip install scikit-learn"
    )


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD."""
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400)
```

```python
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).size)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k", s=20)
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")


def fig_svm_linear_vs_rbf(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=0)
    models = [
        (SVC(kernel="linear", C=1.0, random_state=0), "Linear kernel"),
        (SVC(kernel="rbf", C=1.0, gamma=1.0, random_state=0), "RBF kernel"),
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"SVM: {title}")
    fig.suptitle("SVM: Linear vs RBF kernel")
    out_path = os.path.join(out_dir, "svm_linear_vs_rbf.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_svm_C_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=1)
    models = [
        (SVC(kernel="rbf", C=0.3, gamma=1.0, random_state=1), "C=0.3 (more regularized)"),
        (SVC(kernel="rbf", C=100.0, gamma=1.0, random_state=1), "C=100 (less regularized)"),
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"SVM (RBF): {title}")
    fig.suptitle("Effect of C (soft-margin)")
    out_path = os.path.join(out_dir, "svm_C_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path
```

```python
def fig_svm_gamma_compare(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=2)
    models = [
        (SVC(kernel="rbf", C=3.0, gamma=0.2, random_state=2), "gamma=0.2 (smoother)"),
        (SVC(kernel="rbf", C=3.0, gamma=5.0, random_state=2), "gamma=5.0 (wiggly)")
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"SVM (RBF): {title}")
    fig.suptitle("Effect of gamma (RBF width)")
    out_path = os.path.join(out_dir, "svm_gamma_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_svm_margin_support_vectors(out_dir: str) -> str:
    # Linearly separable-like data for margin visualization
    X, y = make_classification(
        n_samples=200,
        n_features=2,
        n_redundant=0,
        n_informative=2,
        n_clusters_per_class=1,
        class_sep=2.0,
        random_state=3,
    )
    clf = SVC(kernel="linear", C=1e3, random_state=3)
    clf.fit(X, y)

    fig, ax = plt.subplots(figsize=(6.5, 4.8), dpi=160)
    _plot_decision_boundary(ax, clf, X, y, "Linear SVM with margin and SVs")

    # Plot the margin lines using w^T x + b = +/-1
    w = clf.coef_[0]
    b = clf.intercept_[0]
    # Create a grid line in x for margin lines
    x_vals = np.linspace(X[:, 0].min() - 0.5, X[:, 0].max() + 0.5, 200)
    # For y = -(w0*x + b - m)/w1 with m in {0, 1, -1}
    if abs(w[1]) > 1e-12:
        for m in [0.0, 1.0, -1.0]:
            y_vals = -(w[0] * x_vals + b - m) / w[1]
            style = "k-" if m == 0 else "k--"
            ax.plot(x_vals, y_vals, style, lw=1.2, alpha=0.9)

    # Highlight support vectors
    sv = clf.support_vectors_
    ax.scatter(sv[:, 0], sv[:, 1], s=80, facecolors="none", edgecolors="#000",
```

```
            linewidths=1.5, label="SV")
    ax.legend(loc="best")
    out_path = os.path.join(out_dir, "svm_margin_support_vectors.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_svm_decision_function(out_dir: str) -> str:
    np.random.seed(4)
    X, y = make_moons(n_samples=400, noise=0.25, random_state=4)
    clf = SVC(kernel="rbf", C=2.0, gamma=1.0, random_state=4)
    clf.fit(X, y)

    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 500), np.linspace(y_min, y_max, 500)
    )
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    fig, ax = plt.subplots(figsize=(6.5, 5.0), dpi=160)
    # Filled regions by sign
    ax.contourf(xx, yy, (Z > 0).astype(int), levels=2, cmap=ListedColormap(["#
        FFEEEE", "#EEEEFF"]), alpha=0.8)
    # Decision function contours for -1, 0, +1
    CS = ax.contour(xx, yy, Z, levels=[-1.0, 0.0, 1.0], colors=["k", "k", "k"
        ], linestyles=["--", "-", "--"], linewidths=1.2)
    ax.clabel(CS, inline=True, fontsize=8, fmt={-1.0: "-1", 0.0: "0", 1.0: "+1
        "})
    # Data points and SVs
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(["#E74C3C", "#3498DB
        "]), edgecolors="k", s=20)
    sv = clf.support_vectors_
    ax.scatter(sv[:, 0], sv[:, 1], s=80, facecolors="none", edgecolors="#000",
         linewidths=1.5, label="SV")
    ax.set_title("RBF SVM: decision function and margins")
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")
    ax.legend(loc="best")
    out_path = os.path.join(out_dir, "svm_decision_function.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_svm_linear_vs_rbf,
        fig_svm_C_compare,
        fig_svm_gamma_compare,
```

```
        fig_svm_margin_support_vectors,
        fig_svm_decision_function,
    ]
    print("Generating figures into:", os.path.abspath(out_dir))
    for gen in generators:
        try:
            p = gen(out_dir)
            print("Saved:", p)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)


if __name__ == "__main__":
    main()
```

5 Result

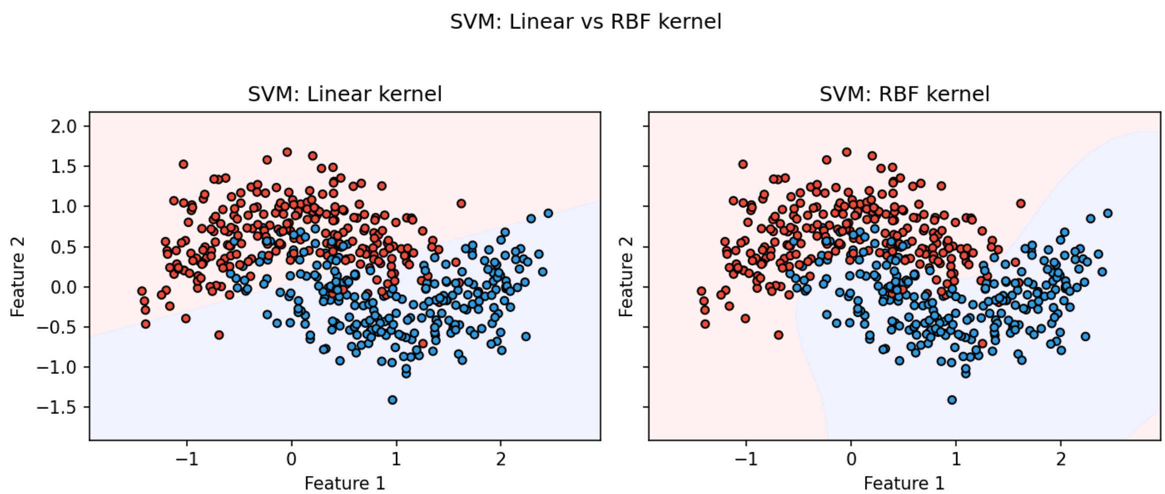

Figure 1: SVM decision boundaries: linear vs RBF kernel.

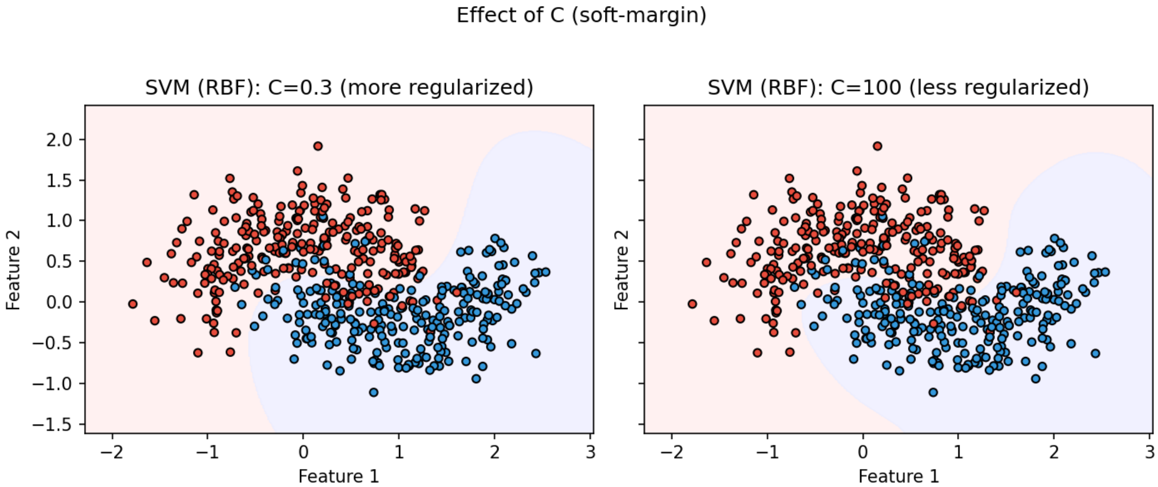

Figure 2: Effect of soft-margin parameter C (RBF kernel).

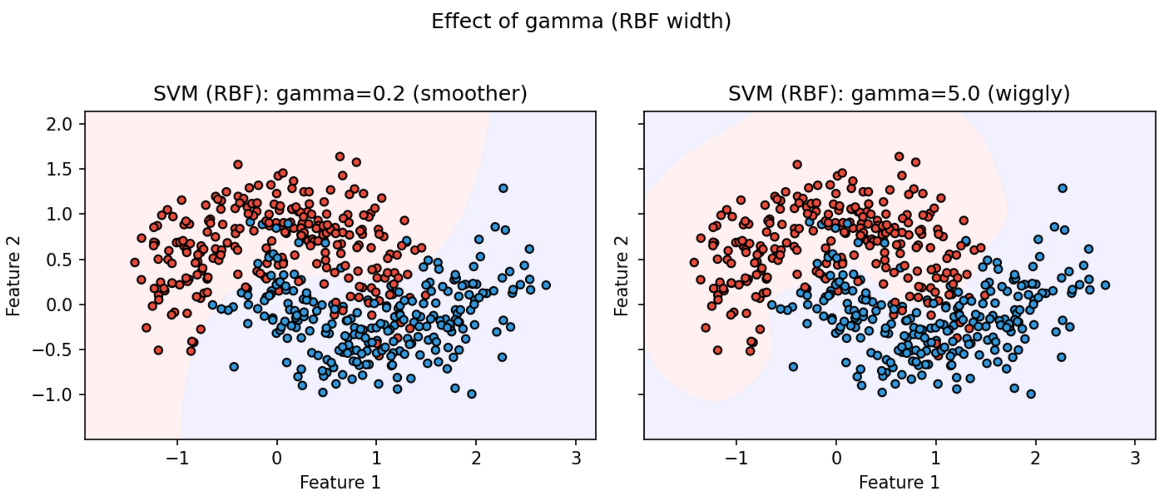

Figure 3: Effect of RBF gamma on boundary smoothness.

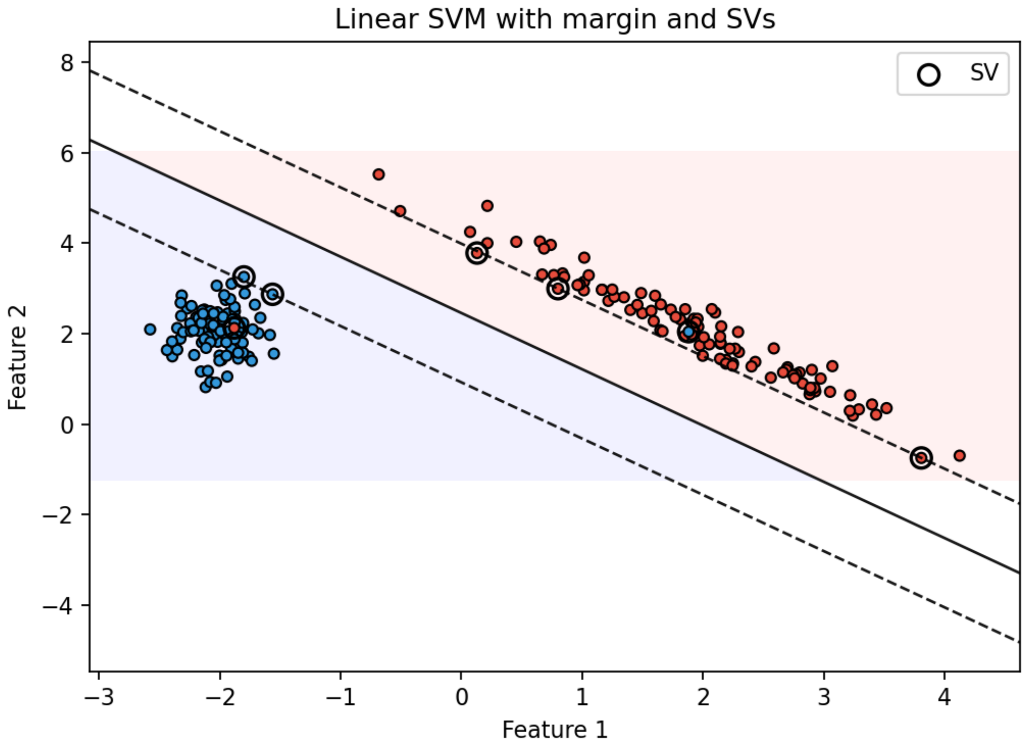

Figure 4: Linear SVM: margin lines and highlighted support vectors.

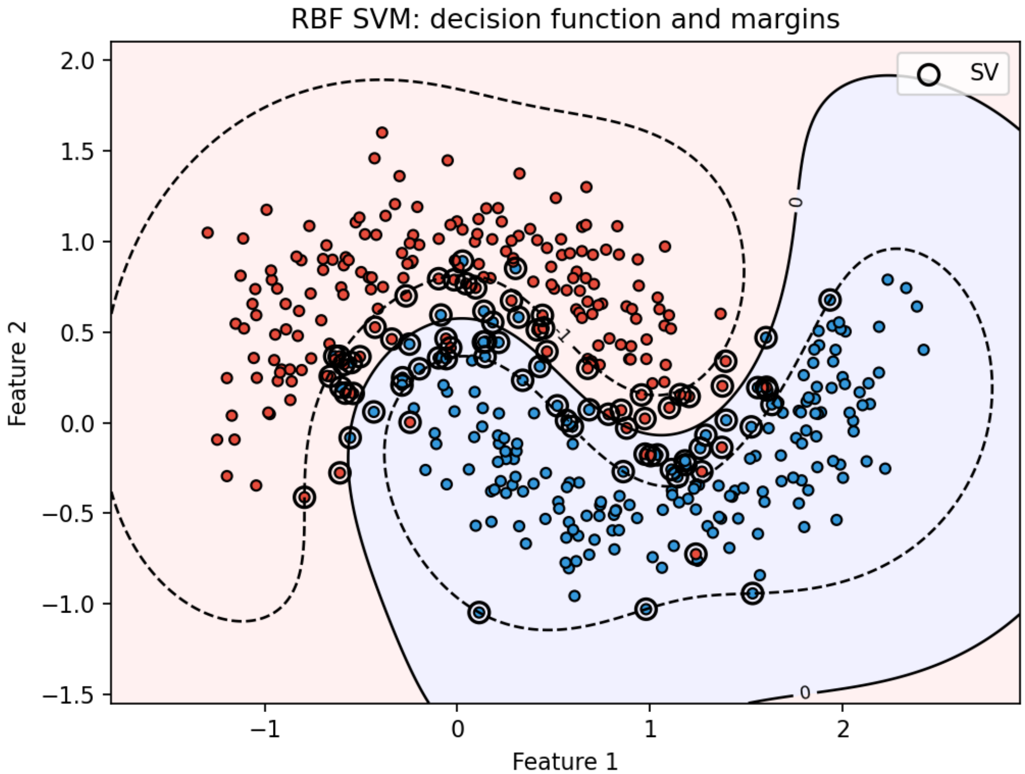

Figure 5: RBF SVM decision function: contours at -1, 0, +1.

6 Summary

SVMs maximize margins for robust classification and extend to non-linear problems via kernels. With proper scaling and tuning of C and kernel parameters, they deliver strong performance across many domains.