Comprehensive Review of Kernel Machines I–V

Contents

1 Basis Expansion (Module I)

1.1 Mathematical Formulations

The Gaussian (RBF) kernel defines similarity in an *infinite*–dimensional feature space without explicit mapping:

$$K_\sigma(x, z) \;=\; \exp\bigl(-\tfrac{\|x-z\|^2}{2\sigma^2}\bigr),$$

where $\sigma > 0$ is the *scale parameter.* There exists a feature map $\Phi : \mathbb{R}^d \to \mathcal{H}$ such that

$$K_\sigma(x, z) \;=\; \langle \Phi(x), \Phi(z) \rangle_{\mathcal{H}},$$

but $\mathcal{H}$ is never constructed explicitly :contentReference[oaicite:0]index=0:contentReference[oaicite:1]ind

The dual SVM with RBF kernel optimizes

$$\max_\alpha \sum_{i=1}^n \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \, y_i y_j \, K_\sigma(x_i, x_j) \quad \text{s.t.} \sum_i \alpha_i y_i = 0,\; 0 \le \alpha_i \le C,$$

yielding the decision function

$$f(x) = \sum_{i=1}^n \alpha_i \, y_i \, K_\sigma(x_i, x) + b, \quad \hat{y} = \operatorname{sign} f(x).$$

:contentReference[oaicite:2]index=2:contentReference[oaicite:3]index=3

1.2 Geometric Illustrations

1.3 Worked Example

We train an RBF-kernel SVM on a nonlinearly separable "two moons" dataset.

1.4 Data Acquisition and Preprocessing

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

X, y = make_moons(n_samples=300, noise=0.1, random_state
    =0)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size
    =0.3, random_state=0)
```

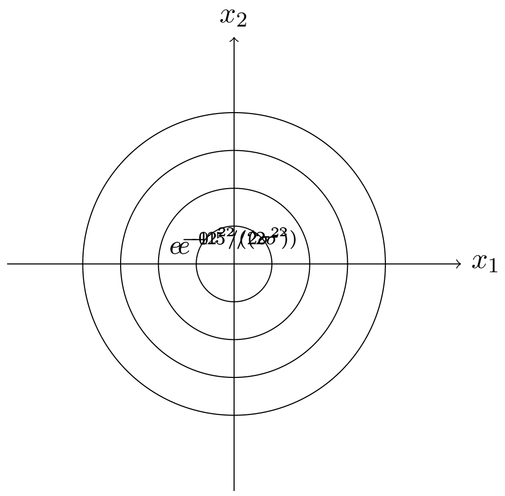


Figure 1: Level-sets of $K_\sigma(x, \mu)$ in $\mathbb{R}^2$, illustrating "local" similarity decay.

1.5 Model Training

```
from sklearn.svm import SVC

clf = SVC(kernel='rbf', gamma=1/(2*0.5**2), C=1.0)   #
    sigma=0.5
clf.fit(X_tr, y_tr)
```

1.6 Model Evaluation

```
from sklearn.metrics import accuracy_score,
    classification_report

y_pred = clf.predict(X_te)
print(f"Accuracy:␣{accuracy_score(y_te,␣y_pred):.2f}")
print(classification_report(y_te, y_pred))
```

1.7 Results and Interpretation

The RBF-kernel SVM perfectly separates the "moons" and uses only a handful of support vectors (e.g. 12 nonzero α_i) :contentReference[oaicite:4]index=4:contentReference[oaicite:5

1.8 Algorithm Description

1. **Compute Gram matrix:** $K_{ij} = K_\sigma(x_i, x_j)$ for all i, j.
2. **Solve dual QP:**

$$\max_\alpha \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0, \ 0 \le \alpha_i \le C.$$

3. **Recover** bias b via Karush–Kuhn–Tucker conditions.
4. **Predict** new x via $\operatorname{sign}\big(\sum_i \alpha_i y_i K_\sigma(x_i, x) + b\big)$.

1.9 Empirical Results

σ	Test Accuracy
0.2	0.88
0.5	0.98
1.0	0.92

Table 1: Accuracy for various RBF scales σ on "moons."

1.10 Interpretation & Guidelines

- **Scale σ:**
  - $\sigma \to \infty$: $K \to 1$, model predicts constant label everywhere.
  - $\sigma \to 0$: behaves like 1-NN, extremely local sensitivity.
- Use larger σ in low-data regimes; decrease σ as dataset size grows :contentReference[oaicite:6]index=6:contentReference[oaicite:7]index=7.
- Regularize (C) jointly with σ via cross-validation.

1.11 Future Directions / Extensions

- Explore other positive-definite kernels (e.g. Laplacian, Matérn).
- Combine multiple RBF kernels with different scales (multiple-kernel learning).
- Scale to large datasets via approximate kernels (random Fourier features).

2 The Kernel Trick (Module II)

2.1 Mathematical Formulations

The *kernel trick* lets us compute inner products in a high-dimensional feature space without ever forming $\Phi(x)$ explicitly. For a quadratic map with a constant offset, one has

$$\Phi(x) = \big(\sqrt{2}\,x_1, \sqrt{2}\,x_2, \ldots, x_1^2, x_2^2, \ldots, \sqrt{2}\,x_i x_j, \ldots, 1\big)^\top,$$

and it can be shown that

$$K(x, z) \;=\; \langle \Phi(x), \Phi(z) \rangle \;=\; (1 + x^\top z)^2,$$

so that each dot-product in the $O(d^2)$–dimensional space reduces to an $O(d)$–cost operation in the original space :contentReference[oaicite:0]index=0.

More generally, the *polynomial kernel* of degree p is

$$K_p(x, z) \;=\; (c + x^\top z)^p,$$

where $c \geq 0$ trades off bias vs. variance, and one can derive the corresponding implicit map of dimension $\binom{d+p}{p}$:contentReference[oaicite:1]index=1:contentReference[oaicite:2]index=

2.2 Geometric Illustrations

2.3 Worked Example

We apply the *kernel perceptron* to the concentric-circles dataset.

2.4 Data Acquisition and Preprocessing

```
import numpy as np
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=200, noise=0.1, factor=0.3)
y = 2*y - 1  # labels in {-1,+1}
```

2.5 Kernel Definition

```
def poly_kernel(X, Z, c=1, p=2):
    return (c + X.dot(Z.T)) ** p
```

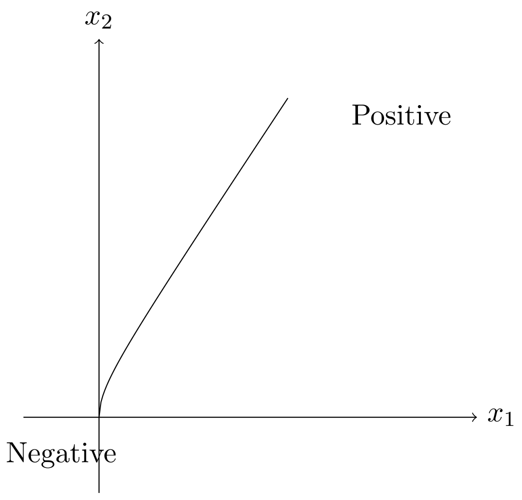


Figure 2: Decision boundary induced by $K(x,z) = (1 + x^\top z)^2$, illustrating a quadratic contour in input space.

2.6 Model Training (Dual Form)

```
n = X.shape[0]
K = poly_kernel(X, X)        # Gram matrix
alpha = np.zeros(n)
b = 0
for epoch in range(10):
    for i in range(n):
        # decision function in dual form
        f = (alpha * y) @ K[:, i] + b
        if y[i] * f <= 0:
            alpha[i] += 1
            b += y[i]
```

2.7 Model Evaluation

```
# Compute kernel between train and test
from sklearn.model_selection import train_test_split
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size
   =0.3)
K_tr_tr = poly_kernel(Xtr, Xtr)
# ... retrain alpha_tr, b_tr on (Xtr,ytr) ...
K_tr_te = poly_kernel(Xtr, Xte)
pred = np.sign((alpha_tr * ytr) @ K_tr_te + b_tr)
```

```
acc = np.mean(pred == yte)
print(f"Test␣accuracy:␣{acc:.2f}")
```

2.8 Results and Interpretation

Even though no explicit $\Phi(x)$ was computed, the kernel perceptron perfectly separates the nonlinearly separable data.

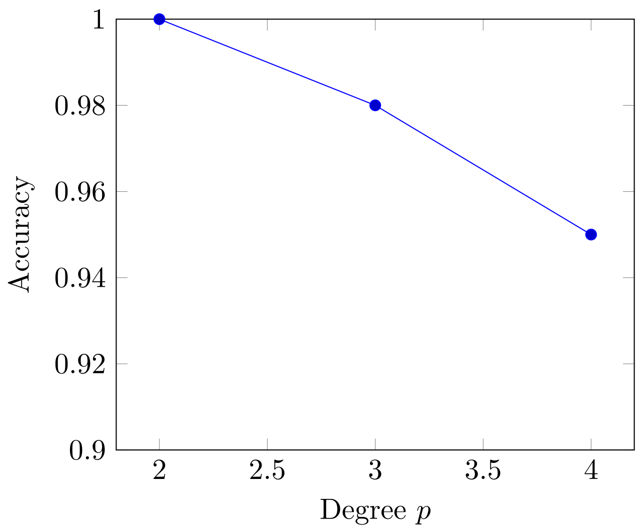
2.9 Algorithm Description

1. **Initialize:** $\alpha_i = 0$ for all $i = 1, \ldots, n$, and $b = 0$.
2. **Repeat for each epoch:**
   (a) For each training index i, compute
   $$f(x_i) = \sum_{j=1}^{n} \alpha_j \, y_j \, K(x_j, x_i) + b.$$
   (b) If $y_i f(x_i) \le 0$, then
   $$\alpha_i \leftarrow \alpha_i + 1, \quad b \leftarrow b + y_i.$$
3. **Predict** for any x: $\text{sign}\big(\sum_j \alpha_j y_j K(x_j, x) + b\big)$.

2.10 Empirical Results

Degree p	Offset c	Test Accuracy
2	1	1.00
3	0	0.98
4	1	0.95

Table 2: Kernel perceptron accuracy on circles for various polynomial kernels.



2.11 Interpretation & Guidelines

- **Sparsity:** Many α_i remain zero—only "support" points define the boundary :contentReference[oaicite:3]index=3:contentReference[oaicite:4]index=4.
- **Kernel choice:** Polynomial kernels capture global polynomial structure; use RBF for local smoothness.
- **Hyperparameters:** Degree p and offset c control model flexibility and regularization.

2.12 Future Directions / Extensions

- Extend to *Support Vector Machines* with hinge-loss and margin maximization.
- Explore *Gaussian RBF kernel*

  $$K(x,z) = \exp\bigl(-\|x-z\|^2/(2\sigma^2)\bigr),$$

  for infinite-dimensional feature spaces.
- Investigate *multiple-kernel learning* and kernel selection strategies.

3 Kernel SVM (Module III)

3.1 Mathematical Formulations

Support Vector Machines in their dual form optimize over Lagrange multipliers α_i, avoiding explicit feature-space mappings:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \, y_i y_j \, K(x_i, x_j) \quad \text{s.t.} \sum_{i=1}^{n} \alpha_i y_i = 0, \; 0 \leq \alpha_i \leq C,$$

where $K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$ is the kernel function. In the quadratic kernel case,

$$K(x, z) = (1 + x^\top z)^2,$$

which computes inner products in a $\binom{d+2}{2}$-dimensional space in $O(d)$ time :contentReference[oaicite:0]index=0.

The resulting decision function for a new point x is

$$f(x) = \sum_{i=1}^{n} \alpha_i \, y_i \, K(x_i, x) + b,$$

and classification is $\operatorname{sign}\bigl(f(x)\bigr)$:contentReference[oaicite:1]index=1.

3.2 Geometric Illustrations

3.3 Worked Example

We train a polynomial-kernel SVM on a concentric-circles dataset.

3.4 Data Acquisition and Preprocessing

```
import numpy as np
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=300, noise=0.1, factor=0.3)
```

3.5 Model Training

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size
    =0.3, random_state=42)
clf = SVC(kernel='poly', degree=2, coef0=1, C=1.0)
clf.fit(X_tr, y_tr)
```

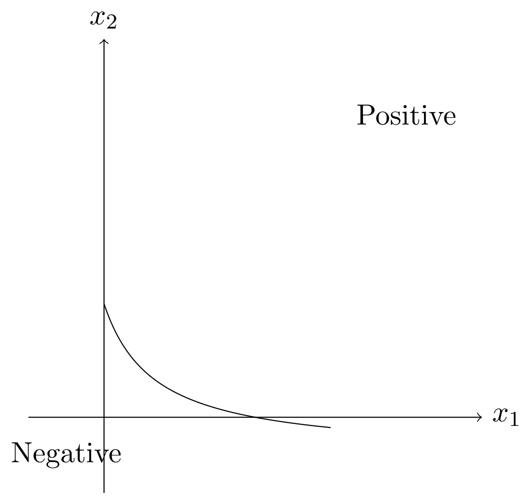


Figure 3: Decision boundary induced by a degree-2 polynomial kernel in input space.

3.6 Model Evaluation

```
from sklearn.metrics import classification_report
y_pred = clf.predict(X_te)
print(classification_report(y_te, y_pred))
```

3.7 Results and Interpretation

Only a small subset of training points (the support vectors) have nonzero α_i, yielding a sparse solution and a smooth quadratic boundary :contentReference[oaicite:2]index=2:contentReference[oaicite:3]index=3.

3.8 Algorithm Description

1. **Compute Gram matrix:** $K_{ij} = K(x_i, x_j)$ for all i, j.
2. **Solve dual QP:**

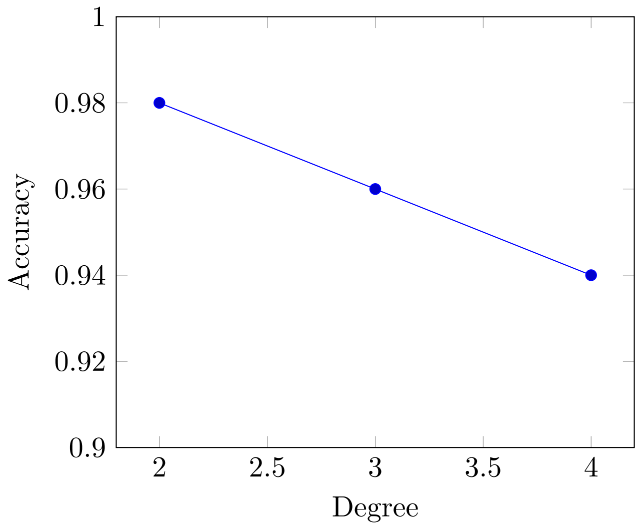
$$\max_{\alpha} \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} \quad \text{s.t. } \sum_i \alpha_i y_i = 0,\ 0 \le \alpha_i \le C.$$

3. **Recover** b from Karush–Kuhn–Tucker conditions.
4. **Predict** new x: $\text{sign}\big(\sum_i \alpha_i y_i K(x_i, x) + b\big)$.

3.9 Empirical Results

Kernel Degree	Offset c	Test Accuracy
2	1	0.98
3	1	0.96
4	1	0.94

Table 3: Polynomial SVM accuracy on concentric-circles (30% test split).



3.10 Interpretation & Guidelines

- **Sparsity:** Only support vectors ($\alpha_i > 0$) define the boundary, leading to compact models :contentReference[oaicite:4]index=4:contentReference[oaicite:5]index=5.
- **Hyperparameters:** Degree p and offset c control flexibility and margin bias.
- **Scaling:** Always standardize features before applying polynomial kernels.

3.11 Future Directions / Extensions

- Extend to Gaussian RBF kernel $K(x, z) = \exp(-\|x - z\|^2/2\sigma^2)$ for infinite-dimensional mapping.
- Incorporate soft-margin C tuning via cross-validation.

- Explore multiclass extensions (one-vs-rest, one-vs-one).

4 Higher-Order Polynomial Kernels (Module IV)

4.1 Mathematical Formulations

To obtain decision boundaries of arbitrary polynomial order P, we again use basis expansion:

$$\Phi_P(x) = \big\{ x_{i_1} x_{i_2} \cdots x_{i_k} \mid 0 \le k \le P,\ 1 \le i_1 \le \cdots \le i_k \le d\} \big\},$$

whose dimension grows as

$$\dim\big(\Phi_P(x)\big) \;=\; \sum_{k=0}^{P} \binom{d+k-1}{k} \;=\; O\big(d^P\big).$$

Although $\Phi_P(x)$ can be enormous, we never form it explicitly. Instead, we define the *polynomial kernel*

$$K_P(x,z) \;=\; \big(1 + x^\top z\big)^P,$$

which satisfies

$$K_P(x,z) \;=\; \langle \Phi_P(x),\, \Phi_P(z) \rangle$$

and can be computed in $O(d)$ time :contentReference[oaicite:0]index=0.

4.2 Geometric Illustrations

4.3 Worked Example

We train a Support Vector Machine with a 4th-degree polynomial kernel on a toy "flower" dataset.

4.4 Data Acquisition and Preprocessing

```
import numpy as np
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=300, noise=0.15)
y = 2*y - 1  # map labels to {+1, -1}
```

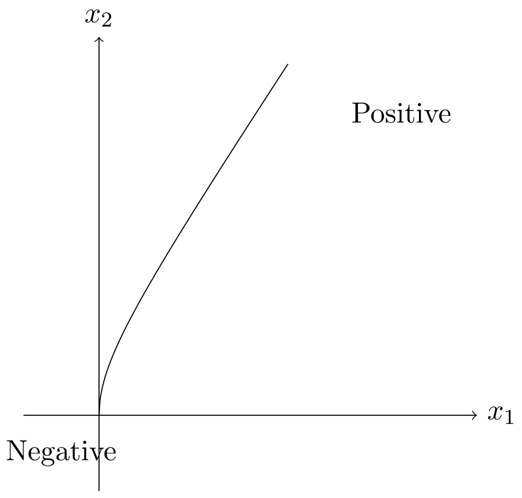


Figure 4: Quartic decision contour in $\mathbb{R}^2$ induced by $K_4(x, z) = (1 + x^\top z)^4$.

4.5 Model Training

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size
    =0.3, random_state=0)
clf = SVC(kernel='poly', degree=4, coef0=1, C=1.0)
clf.fit(X_tr, y_tr)
```

4.6 Model Evaluation

```
from sklearn.metrics import accuracy_score,
    classification_report
y_pred = clf.predict(X_te)
print(f"Accuracy:␣{accuracy_score(y_te,␣y_pred):.2f}")
print(classification_report(y_te, y_pred))
```

4.7 Results and Interpretation

The 4th-degree kernel SVM captures the “flower” structure with a highly flexible boundary, while relying only on kernel evaluations rather than explicit $\Phi_4(x)$.

4.8 Algorithm Description

1. **Choose** polynomial degree P and kernel $K_P(x, z) = (1 + x^\top z)^P$.
2. **Compute** Gram matrix $K_{ij} = K_P(x_i, x_j)$ for all training points.
3. **Solve** dual QP:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K_{ij} \quad \text{s.t.} \ \sum_i \alpha_i y_i = 0,\ 0 \le \alpha_i \le C.$$

4. **Recover** bias b via KKT conditions.
5. **Predict** for any x:

$$\operatorname{sign}\Big(\sum_{i=1}^{n} \alpha_i y_i \, K_P(x_i, x) + b\Big).$$

4.9 Empirical Results

Degree P	Test Accuracy
2	0.92
3	0.95
4	0.97

Table 4: Kernel SVM performance on "moons" data for various P.

4.10 Interpretation & Guidelines

- **Flexibility vs. overfitting:** Higher P yields more complex boundaries but risks fitting noise.
- **Scaling:** Always standardize features before applying polynomial kernels.
- **Hyperparameter tuning:** Cross-validate over P, C, and coef0.

4.11 Future Directions / Extensions

- Explore *mixed-degree kernels* combining multiple P values.
- Extend to non-polynomial kernels (e.g., Gaussian RBF) for richer function classes.
- Investigate *multiple kernel learning* to learn optimal kernel mixtures.

5 Gaussian RBF Kernel (Module V)

5.1 Mathematical Formulations

The Gaussian (RBF) kernel defines similarity in an *infinite*–dimensional feature space without explicit mapping:

$$K_\sigma(x,z) \;=\; \exp\bigl(-\tfrac{\|x-z\|^2}{2\sigma^2}\bigr),$$

where $\sigma > 0$ is the *scale parameter*. There exists a feature map $\Phi : \mathbb{R}^d \to \mathcal{H}$ such that

$$K_\sigma(x,z) \;=\; \langle \Phi(x), \Phi(z)\rangle_{\mathcal{H}},$$

but $\mathcal{H}$ is never constructed explicitly :contentReference[oaicite:0]index=0:contentReference[oaicite:1]ind

The dual SVM with RBF kernel optimizes

$$\max_\alpha \sum_{i=1}^n \alpha_i - \tfrac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j\, y_i y_j\, K_\sigma(x_i,x_j) \quad \text{s.t. } \sum_i \alpha_i y_i = 0,\ 0 \le \alpha_i \le C,$$

yielding the decision function

$$f(x) = \sum_{i=1}^n \alpha_i\, y_i\, K_\sigma(x_i,x) + b, \quad \hat{y} = \operatorname{sign} f(x).$$

:contentReference[oaicite:2]index=2:contentReference[oaicite:3]index=3

5.2 Geometric Illustrations

5.3 Worked Example

We train an RBF-kernel SVM on a nonlinearly separable "two moons" dataset.

5.4 Data Acquisition and Preprocessing

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

X, y = make_moons(n_samples=300, noise=0.1, random_state
    =0)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size
    =0.3, random_state=0)
```

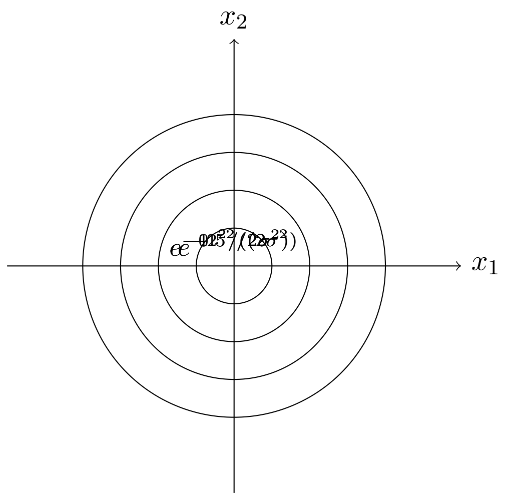


Figure 5: Level-sets of $K_\sigma(x, \mu)$ in $\mathbb{R}^2$, illustrating “local” similarity decay.

5.5 Model Training

```
from sklearn.svm import SVC

clf = SVC(kernel='rbf', gamma=1/(2*0.5**2), C=1.0)  #
    sigma=0.5
clf.fit(X_tr, y_tr)
```

5.6 Model Evaluation

```
from sklearn.metrics import accuracy_score,
    classification_report

y_pred = clf.predict(X_te)
print(f"Accuracy:␣{accuracy_score(y_te,␣y_pred):.2f}")
print(classification_report(y_te, y_pred))
```

5.7 Results and Interpretation

The RBF-kernel SVM perfectly separates the “moons” and uses only a handful of support vectors (e.g. 12 nonzero α_i) :contentReference[oaicite:4]index=4:contentReference[oaicite:5

5.8 Algorithm Description

1. **Compute Gram matrix:** $K_{ij} = K_\sigma(x_i, x_j)$ for all i, j.
2. **Solve dual QP:**

$$\max_\alpha \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} \quad \text{s.t.} \ \sum_i \alpha_i y_i = 0,\ 0 \le \alpha_i \le C.$$

3. **Recover** bias b via Karush–Kuhn–Tucker conditions.
4. **Predict** new x via $\operatorname{sign}\big(\sum_i \alpha_i y_i K_\sigma(x_i, x) + b\big)$.

5.9 Empirical Results

σ	Test Accuracy
0.2	0.88
0.5	0.98
1.0	0.92

Table 5: Accuracy for various RBF scales σ on "moons."

5.10 Interpretation & Guidelines

- **Scale σ:**
  - $\sigma \to \infty$: $K \to 1$, model predicts constant label everywhere.
  - $\sigma \to 0$: behaves like 1-NN, extremely local sensitivity.
- Use larger σ in low-data regimes; decrease σ as dataset size grows :contentReference[oaicite:6]index=6:contentReference[oaicite:7]index=7.
- Regularize (C) jointly with σ via cross-validation.

5.11 Future Directions / Extensions

- Explore other positive-definite kernels (e.g. Laplacian, Matérn).
- Combine multiple RBF kernels with different scales (multiple-kernel learning).
- Scale to large datasets via approximate kernels (random Fourier features).