Review of Kernel Machines II: The Kernel Trick

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

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

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

3 Worked Example

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

3.1 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}
```

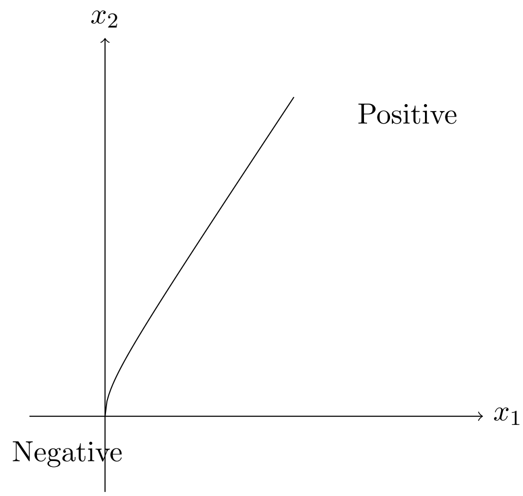


Figure 1: Decision boundary induced by $K(x, z) = (1 + x^{\top} z)^2$, illustrating a quadratic contour in input space.

3.2 Kernel Definition

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

3.3 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]
```

3.4 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}")
```

3.5 Results and Interpretation

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

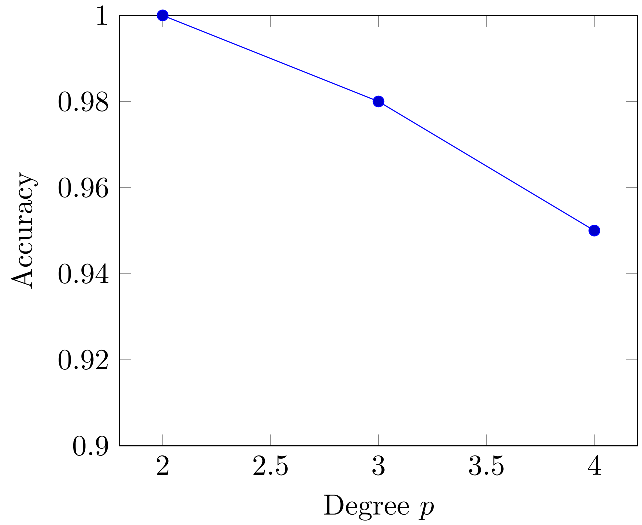
4 Algorithm Description

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 $y_i f(x_i) \leq 0$, then
   $$\alpha_i \leftarrow \alpha_i + 1, \quad b \leftarrow b + y_i.$$
3. **Predict** for any x: $\operatorname{sign}\big(\sum_j \alpha_j y_j K(x_j, x) + b\big)$.

5 Empirical Results

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

Table 1: Kernel perceptron accuracy on circles for various polynomial kernels.



6 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.

7 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.