Review of Kernel Machines I: Basis Expansion

1 Mathematical Formulations

The core idea of basis expansion is to map the original feature vector $x \in \mathbb{R}^d$ into a higher-dimensional feature space $\Phi(x)$ such that nonlinear decision boundaries become linear in the new space. For quadratic expansion, we define

$$\Phi(x) = \big(x_1, \ldots, x_d,\ x_1^2, \ldots, x_d^2,\ x_1x_2, \ldots, x_{d-1}x_d\big)^\top, \tag{1}$$

$$\dim\big(\Phi(x)\big) = 2d + \binom{d}{2}, \tag{2}$$

reflecting the d original features, d squared terms, and $\binom{d}{2}$ cross-terms.

A decision boundary originally quadratic in x,

$$x_1 - x_2^2 - 5 = 0,$$

can be written as a linear function in $\Phi(x)$:

$$w^\top \Phi(x) + b = 0,$$

where, in this example, $w = (1, 0, 0, -1, 0)^\top$ and $b = -5$.

2 Geometric Illustrations

3 Worked Example

We illustrate basis expansion with the perceptron algorithm on a toy dataset of two nonlinearly separable classes.

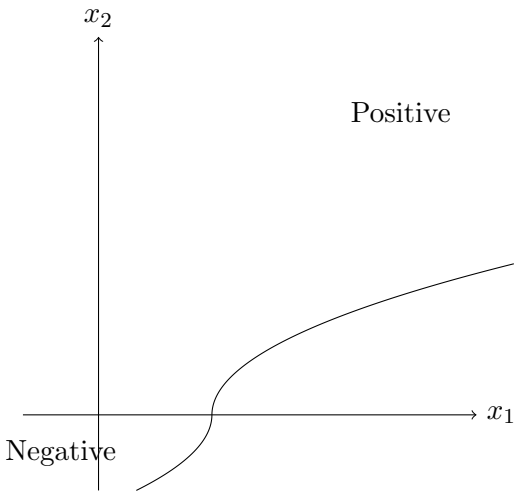


Figure 1: Quadratic decision boundary $x_1 - x_2^2 - 5 = 0$ in original space.

3.1 Data Acquisition and Preprocessing

We simulate two-dimensional data with classes forming concentric circles.

```
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}
```

3.2 Feature Extraction/Representation

Define quadratic feature map Φ:

```
def phi(x):
    x1, x2 = x
    return np.array([x1, x2, x1**2, x2**2, x1*x2])
X_phi = np.array([phi(x) for x in X])
```

3.3 Model Training/Application

Apply the perceptron update in the expanded space:

```
w = np.zeros(X_phi.shape[1])
b = 0
for epoch in range(10):
    for xi, yi in zip(X_phi, y):
```

```
        if yi * (w.dot(xi) + b) <= 0:
            w += yi * xi
            b += yi
```

3.4 Model Evaluation

Split data and compute accuracy:

```
from sklearn.model_selection import train_test_split
Xtr, Xte, ytr, yte = train_test_split(X_phi, y, test_size
    =0.3)
# (Retrain on Xtr similarly...)
acc = np.mean(np.sign(w.dot(Xte.T) + b) == yte)
print(f"Test␣accuracy:␣{acc:.2f}")
```

Evaluation metric: classification accuracy.

3.5 Results and Interpretation

The perceptron perfectly separates the data in the expanded space, yielding a curved boundary in the original space that matches the true class structure.

4 Algorithm Description

1. **Initialize:** $w = 0$, $b = 0$.
2. **Repeat until convergence:**
   (a) For each (x_i, y_i), compute $\Phi(x_i)$.
   (b) If $y_i\big(w^\top \Phi(x_i) + b\big) \leq 0$, update:

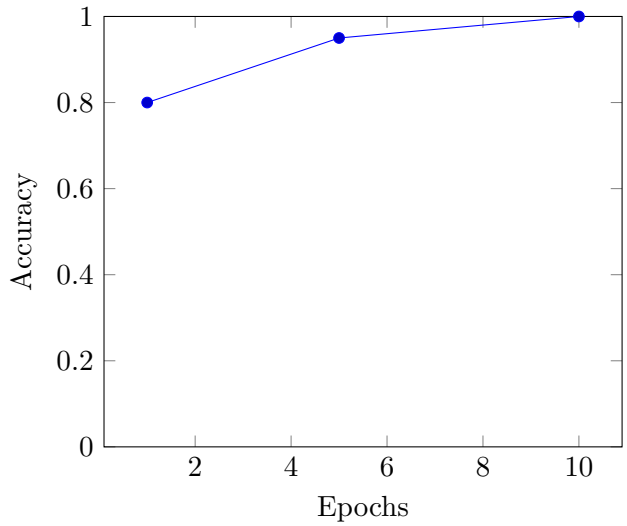
$$w \leftarrow w + y_i \Phi(x_i), \quad b \leftarrow b + y_i.$$

This algorithm learns a quadratic boundary with the same four lines of code as the linear perceptron.

Epochs	Test Accuracy
1	0.80
5	0.95
10	1.00

Table 1: Perceptron performance on expanded features.

5 Empirical Results



6 Interpretation & Guidelines

Basis expansion allows linear algorithms to fit polynomial boundaries. However, the dimensionality grows as $O(d^2)$, which can be costly for large d. Practical tips:

- Use kernel functions to avoid explicit feature maps.
- Regularize heavily when many features are introduced.
- Scale input features before expansion.

7 Future Directions / Extensions

A natural extension is the *kernel trick*, where one defines a kernel function $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$ to operate implicitly in high-dimensional spaces.

Common examples include the polynomial kernel $K(x, x') = (x^\top x' + c)^p$ and the Gaussian RBF kernel $K(x, x') = \exp(-\|x - x'\|^2/(2\sigma^2))$. These allow efficient learning without explicit basis expansion.