

19 The Kernel Trick: Turning Lines into Curves

19.1 From Features to Higher Dimensions: The Idea of Mapping

19.1.1 Why We Need Feature Mapping

In the previous chapter, we introduced the kernel trick used in Support Vector Machine (SVM). The core idea of this trick is to implicitly map the input data from its original low-dimensional space into a higher-dimensional feature space, where the data may become linearly separable. This transformation is achieved without explicitly computing the mapping, but instead through the use of a kernel function.

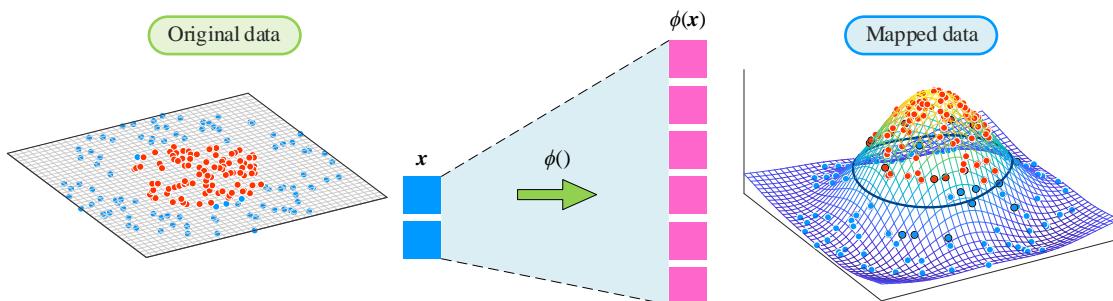
Commonly used kernel functions include the polynomial kernel, and radial basis function (RBF) kernel. The kernel trick greatly enhances the flexibility and performance of SVMs by allowing them to transform complex nonlinear classification (or regression) problems in the input space into linear problems in the transformed space. As a result, SVMs can capture nonlinear relationships and achieve better generalization on unseen data.

19.1.2 Visualizing the Mapping Function

First, let's understand the idea of a mapping function, also called a feature map. A mapping function, written as $\phi(\mathbf{x})$, takes the original data point \mathbf{x} and transforms it into a new point $\phi(\mathbf{x})$, in another, often higher-dimensional, space.

As shown in [Figure 1](#), this process increases the number of features, or dimensions, of the data. The key idea is that data which cannot be separated by a straight line (or hyperplane) in the original space may become linearly separable after being mapped to this new feature space.

In other words, the mapping function helps us “unfold” the data into a space where simple linear models, such as linear classifiers, can work effectively. This is the foundation of the kernel trick used in support vector machines and other algorithms that deal with nonlinear data.



[Figure 1](#). Illustration of the feature mapping principle

[Figure 2](#) shows data with a single feature, containing classes shown in red and blue. In [Figure 2 \(a\)](#), the original data has four red points on the left and four points on the right, while the nine blue points in the middle belong to another class. In this one-dimensional space (represented by x_1), the data points from the two classes cannot be separated by a single straight line.

However, if we apply a quadratic mapping—that is, we create a new feature such as x_1^2 —the data can be transformed into a two-dimensional feature space, as shown in Figure 2 (b). In this new space, the samples that were mixed together before can now be clearly separated by a straight line.

This example shows how feature mapping helps convert a nonlinear problem in the original space into a linear one in a higher-dimensional space, making classification much easier.

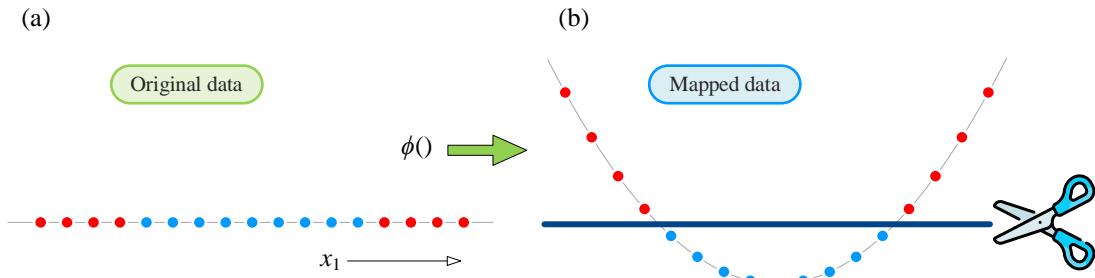


Figure 2. Illustration of how quadratic mapping makes nonlinearly separable data become linearly separable in a higher-dimensional space.

19.1.3 Decision Boundary

From the previous chapter, we know that the decision boundary of a Support Vector Machine can be expressed as a **hyperplane** in the feature space:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (1)$$

This equation defines the linear boundary that separates different classes in the original feature space.

After applying a **feature mapping**, the decision boundary becomes:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = 0 \quad (2)$$

Mapping function

The exact *shape* of the decision boundary depends on the mapping function $\phi(\mathbf{x})$.

Note that the vector \mathbf{w} in Equation (1) is different from that in Equation (2), because $\phi(\mathbf{x})$ changes the dimensionality of the data.

In general, the transformed feature vector $\phi(\mathbf{x})$ has many more features than the original input \mathbf{x} ; in extreme cases, it can even have infinitely many dimensions.

An important point is that we do **not** need to know the explicit form of $\phi(\mathbf{x})$. What truly matters is the **inner product** between mapped vectors, $(\phi(x_i)^\top \phi(x_j))$, because it captures the similarity between data points in the high-dimensional feature space.

With this understanding of feature mapping, we are ready to introduce the **kernel trick**, which allows us to compute these inner products efficiently without explicitly performing the mapping.

19.2 The Math Behind the Magic: Optimization in Kernel SVMs

19.2.1 Revisiting the SVM Optimization Problem

Kernel trick maps the input data into a high-dimensional feature space, transforming a nonlinear classification problem into a linear one. During this transformation, SVM still aims to find the optimal decision boundary—a hyperplane that best separates the data from different classes.

This process can be formulated as an optimization problem, where we seek to minimize a loss function while satisfying certain constraints. The problem can be solved by minimizing the Lagrangian function, a method that helps handle these constraints effectively.

Similar to the hard-margin SVM discussed in the previous chapter, the optimization problem for kernelized SVMs can be written as:

$$\begin{aligned} & \arg \min_{w,b} \frac{\|w\|^2}{2} \\ \text{subject to } & y^{(i)} \left(w \cdot \underbrace{\phi(x^{(i)})}_{\text{Mapping function}} + b \right) \geq 1, \quad i = 1, 2, 3, \dots, n \end{aligned} \tag{3}$$

19.2.2 The Lagrangian and Inner Products

To solve this constrained optimization problem, we construct the Lagrangian function:

$$L(w, b, \lambda) = \frac{\|w\|^2}{2} + \sum_{i=1}^n \lambda_i \left(1 - y^{(i)} (w \cdot \phi(x^{(i)}) + b) \right) \tag{4}$$

Taking the partial derivatives of $L(w, b, \lambda)$ with respect to w and b , and setting them to zero, we obtain:

$$\begin{cases} w = \sum_{i=1}^n \lambda_i y^{(i)} \phi(x^{(i)}) \\ \sum_{i=1}^n \lambda_i y^{(i)} = 0 \end{cases} \tag{5}$$

Thus the Lagrangian function can be rewritten as

$$\begin{aligned} L(w, b, \lambda) &= \frac{\|w\|^2}{2} + \sum_{i=1}^n \lambda_i \left(1 - y^{(i)} (w \cdot \phi(x^{(i)}) + b) \right) \\ &= \sum_{i=1}^n \lambda_i - \frac{\sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y^{(i)} y^{(j)} \overbrace{\phi(x^{(i)}) \cdot \phi(x^{(j)})}^{\text{Kernel function}}}{2} \end{aligned} \tag{6}$$

19.2.3 Introducing the Kernel Function

The inner product in the equation above can be replaced by a kernel function

$$\kappa(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \tag{7}$$

A kernel function performs a “vector-to-scalar” operation—it computes the similarity between two data points according to a specific rule, without needing to explicitly know the mapping function $\phi(\mathbf{x})$.

Using this notation, the dual form of the Lagrangian becomes:

$$L(\boldsymbol{\lambda}) = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y^{(i)} y^{(j)} \underbrace{\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})}_{\text{Kernel}} \quad (8)$$

(8) can be written as:

$$L(\boldsymbol{\lambda}) = \sum_{i=1}^n \lambda_i - \frac{1}{2} \begin{bmatrix} \lambda_1 y^{(1)} \\ \lambda_2 y^{(2)} \\ \vdots \\ \lambda_n y^{(n)} \end{bmatrix}^T \underbrace{\begin{bmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(n)}) \\ \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(n)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(n)}) \end{bmatrix}}_{\text{Gram matrix}} \begin{bmatrix} \lambda_1 y^{(1)} \\ \lambda_2 y^{(2)} \\ \vdots \\ \lambda_n y^{(n)} \end{bmatrix} \quad (9)$$

Let the Gram matrix \mathbf{K} be defined as the matrix of all pairwise kernel values:

$$\mathbf{K} = \underbrace{\begin{bmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(n)}) \\ \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(n)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(2)}) & \cdots & \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(n)}) \end{bmatrix}}_{\text{Gram matrix}} \quad (10)$$

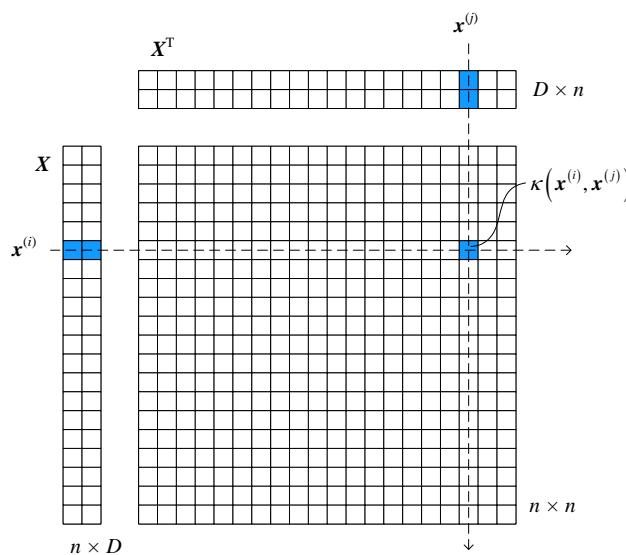


Figure 3. Gram matrix using a nonlinear kernel

19.2.4 From Dual Problem to Decision Function

The dual optimization problem can thus be written as:

$$\begin{aligned} \arg \max_{\lambda} & \sum_{i=1}^n \lambda_i - \frac{\sum_{j=1}^n \sum_{i=1}^n \lambda_i \lambda_j y^{(i)} y^{(j)} \overbrace{\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})}^{\text{Kernel}}}{2} \\ \text{subject to } & \begin{cases} \sum_{i=1}^n \lambda_i y^{(i)} = 0 \\ \lambda_i \geq 0, \quad i, j = 1, 2, 3, \dots, n \end{cases} \end{aligned} \quad (11)$$

Once the optimization problem is solved, the SVM decision boundary becomes:

$$f(\mathbf{x}) = \sum_{i=1}^n \left(\lambda_i y^{(i)} \underbrace{\kappa(\mathbf{x}^{(i)}, \mathbf{x})}_{\text{Kernel}} \right) + b = 0 \quad (12)$$

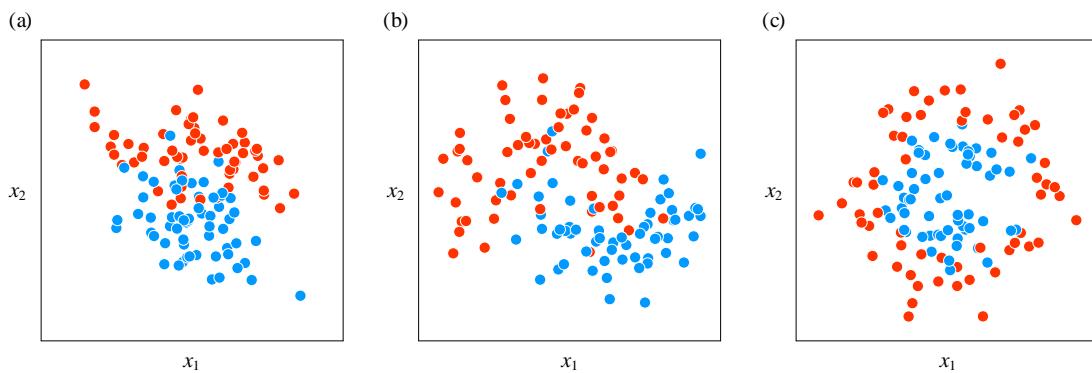
Here $\mathbf{x}^{(i)}$ are support vectors (training points that define the margin), and \mathbf{x} represents a new data point.

SVMs can use different types of kernels depending on the structure of the data:

- Linear kernel
- Polynomial kernel
- Gaussian kernel, also known as radial basis function (RBF) kernel
- Sigmoid kernel

Each kernel defines a different way to measure similarity between data points, resulting in different shapes of decision boundaries.

In [Figure 2](#), three datasets—linearly separable, crescent-shaped, and ring-shaped—are classified using these kernels. Comparing their decision boundaries helps illustrate the strengths and weaknesses of each kernel type.



[Figure 4](#). Three example datasets for SVM classification: linearly separable, crescent-shaped, and ring-shaped.

19.3 Linear Kernel: The Straightforward Baseline

19.3.1 Formula and Geometric Meaning

The linear kernel is the simplest and most commonly used kernel in SVM. Its mathematical form is:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (13)$$

Using a linear kernel, the SVM decision boundary can be expressed as:

$$f(\mathbf{x}) = \sum_{i=1}^n \underbrace{\lambda_i y^{(i)} \mathbf{x}^{(i)}}_{\text{Hyperplane}} + b = 0 \quad (14)$$

Here, $(x^{(i)}, y^{(i)})$ is the i -th training sample, and λ_i is the corresponding Lagrange multiplier obtained from the optimization process. Recall that \mathbf{x} is represented as a column vector of features.

19.3.2 Building the Decision Boundary from Support Vectors

We can define a function $f_i(\mathbf{x})$ to represent the contribution of each support vector:

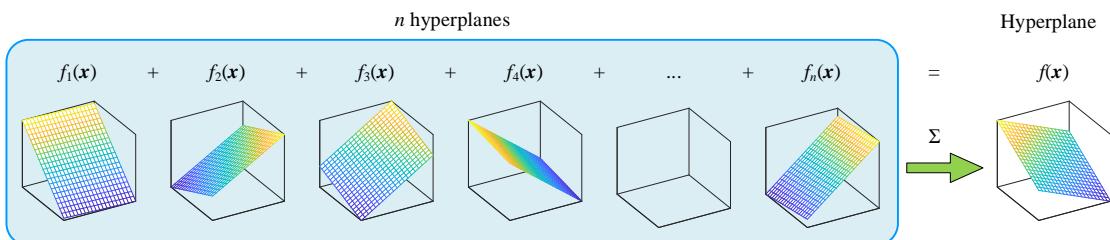
$$f_i(\mathbf{x}) = \underbrace{\lambda_i y^{(i)} \mathbf{x}^{(i)}}_{\text{Coefficients}} \quad (15)$$

Each λ_i acts like a weight for the corresponding training point.

Summing these contributions produces the final decision function $f(\mathbf{x})$, which defines a hyperplane in the feature space. In other words, the linear kernel SVM decision boundary is formed by the superposition of n hyperplanes, as shown in [Figure 5](#):

$$f(\mathbf{x}) = \sum_{i=1}^n f_i(\mathbf{x}) + b = 0 \quad (16)$$

This shows that the decision boundary in a linear kernel SVM is always flat (a hyperplane) in the original feature space.



[Figure 5](#). Linear kernel SVM decision boundary constructed by the superposition of hyperplanes.

19.3.3 When It Works—and When It Fails

[Figure 6](#) shows the classification of linearly separable data using a linear kernel SVM. As expected, the decision boundary is a straight line that “almost” perfectly separates the two classes.

However, when the same SVM is applied to nonlinearly separable data, such as crescent-shaped ([Figure 7](#)) or ring-shaped ([Figure 8](#)) datasets, the linear kernel is insufficient. The decision boundary remains flat, and it cannot separate the classes correctly.

Even with a soft-margin SVM that allows some misclassification, linear kernels work well for linearly separable data but struggle with inherently nonlinear patterns.

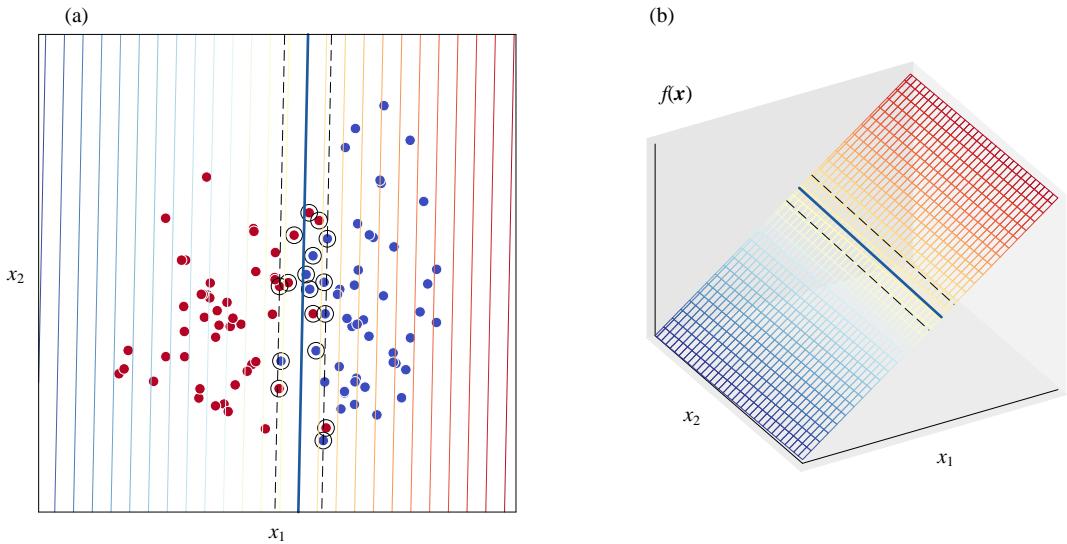


Figure 6. Linearly separable data classified by a linear kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

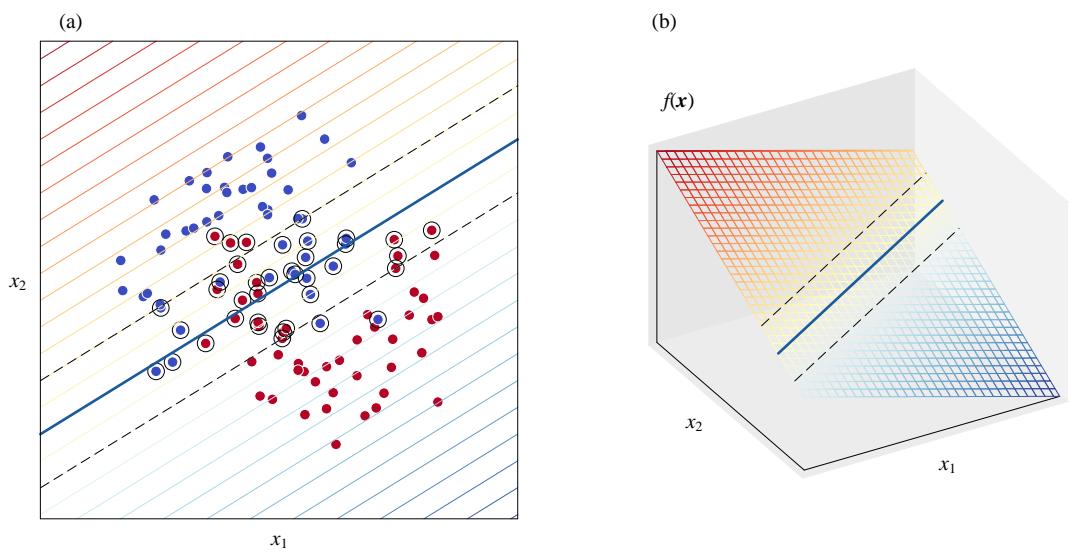


Figure 7. Crescent-shaped data classified by a linear kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

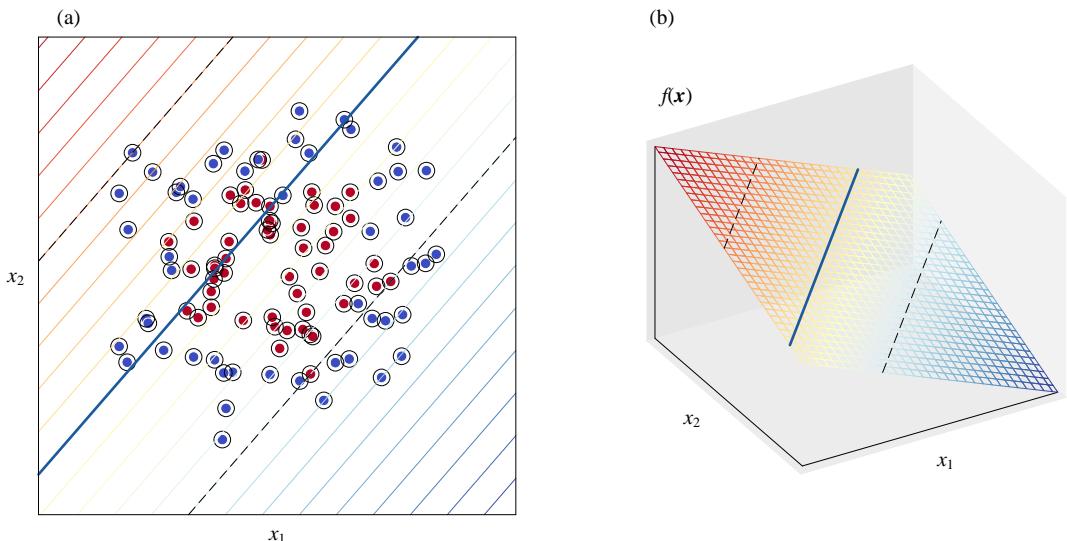


Figure 8. Ring-shaped data classified by a linear kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.4 Polynomial Kernel: Adding Curves to the Mix

19.4.1 The Polynomial Kernel Formula and Parameters

The **polynomial kernel** is a flexible kernel function that allows SVMs to model nonlinear decision boundaries by mapping the input data into a higher-dimensional space. Its general form is:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)^d \quad (17)$$

where γ is a scaling coefficient, r (and d) determines the constant term, d is the degree of the polynomial, which determines the complexity of the mapping.

Linear kernels are a special case of polynomial kernels with ($d = 1$). When ($d = 2$), we have a quadratic kernel, and when ($d = 3$), it becomes a cubic kernel.

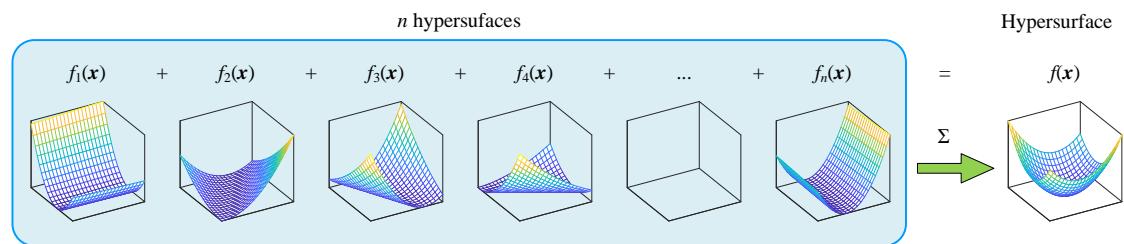
19.4.2 Quadratic Kernel: Parabolic Boundaries

Consider a quadratic kernel with $\gamma = 1$, $r = 0$, and $d = 2$:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j)^2 \quad (18)$$

Expanding the polynomial kernel gives a new feature mapping $\phi(\mathbf{x})$ that includes squared and cross-product terms. While the exact form of $\phi(\mathbf{x})$ is not unique, what matters is that it defines a mapping rule and allows us to compute the kernel inner products efficiently.

Just like in linear kernels, each training point contributes a weighted term $f_i(\mathbf{x})$, but in polynomial kernels, these terms form curved surfaces in the high-dimensional space, as illustrated in [Figure 9](#). The quadratic (second-degree) kernel can create decision boundaries shaped like valley surfaces, hyperbolic paraboloids, or elliptical paraboloids, depending on the data points.



[Figure 9](#). Illustration of how quadratic kernels produce hyper-curved decision surfaces.

19.4.3 Comparing Polynomial Kernels on Different Datasets

[Figure 10 ~ Figure 12](#) show the SVM classification results for linearly separable, crescent-shaped, and ring-shaped datasets using a quadratic kernel.

While quadratic kernels can capture some nonlinear patterns, their flexibility is limited. For more complex decision surfaces, **higher-degree polynomial kernels** are needed.

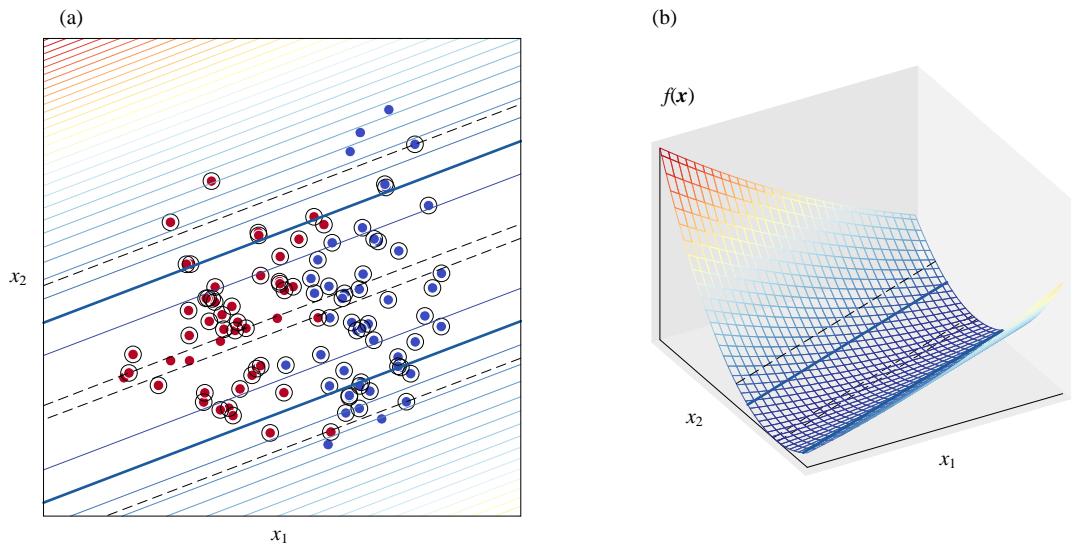


Figure 10. Linearly separable data classified by quadratic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

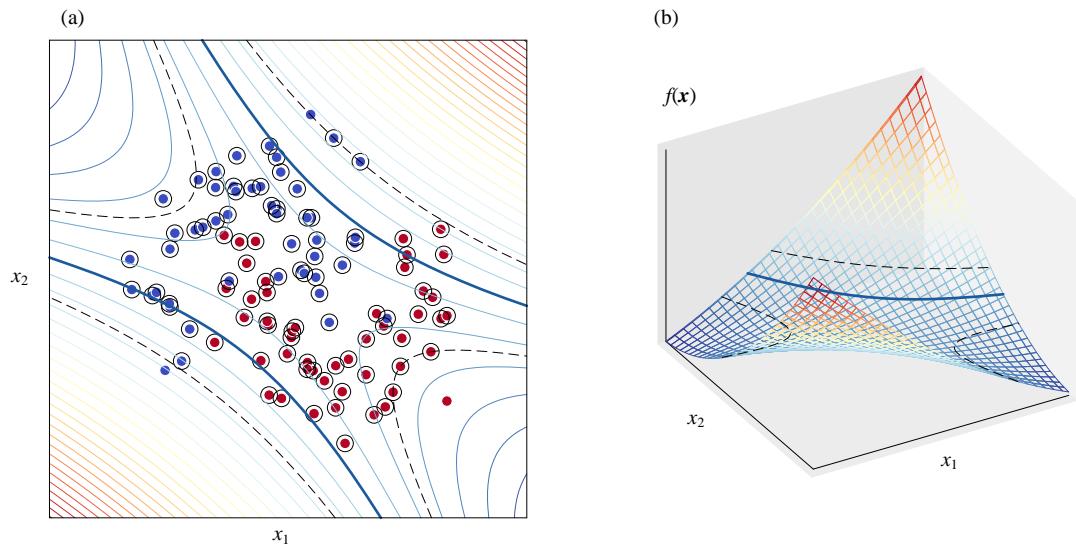


Figure 11. Crescent-shaped data classified by quadratic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

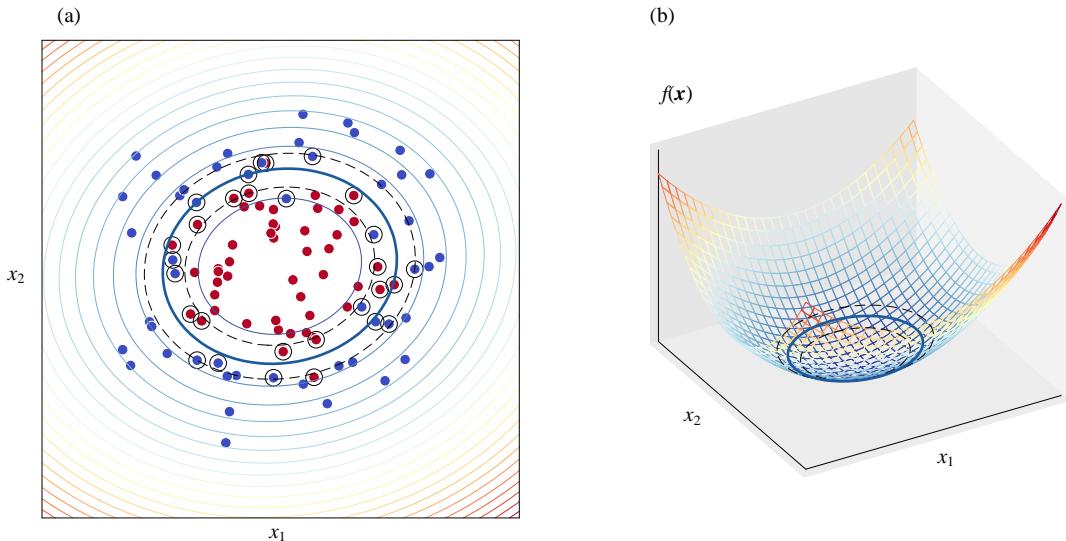


Figure 12. Ring-shaped data classified by quadratic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.4.4 Cubic Kernel: Flexibility Increased

The cubic (third-degree) kernel further increases the flexibility of the decision boundary:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)^3 \quad (19)$$

Figure 13 ~ Figure 15 show how cubic kernels improve classification on linearly separable, crescent, and ring-shaped datasets compared to quadratic kernels.

Higher-degree polynomial kernels can model increasingly complex surfaces, allowing the SVM to fit more complicated data patterns.

However, there is a trade-off: higher-degree kernels may cause overfitting. Overfitting happens when the model fits the training data too closely, capturing noise instead of underlying patterns, which reduces performance on new, unseen data. Techniques such as regularization or limiting the polynomial degree are often used to prevent overfitting and maintain good generalization.

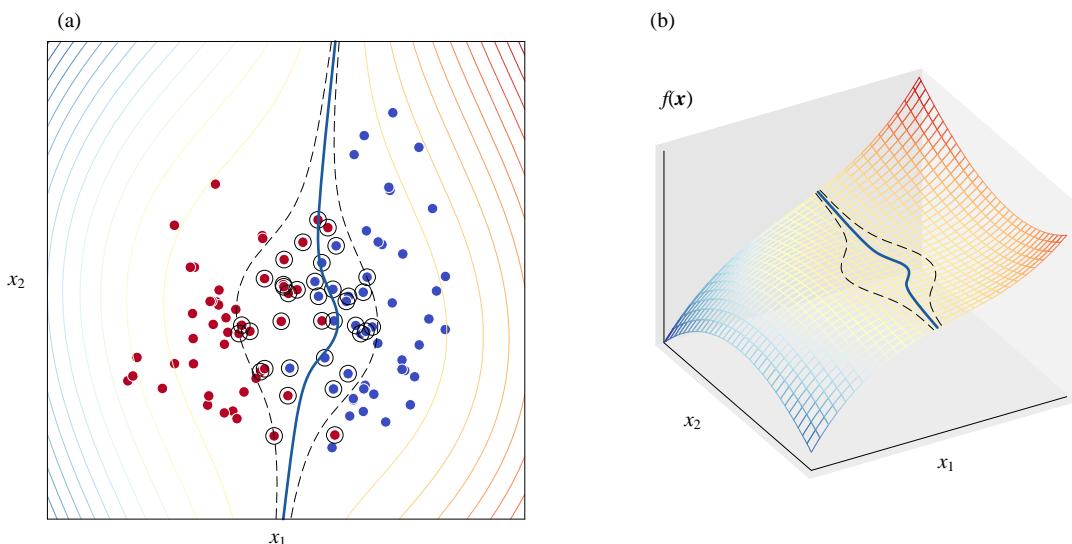


Figure 13. Linearly separable data classified by cubic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

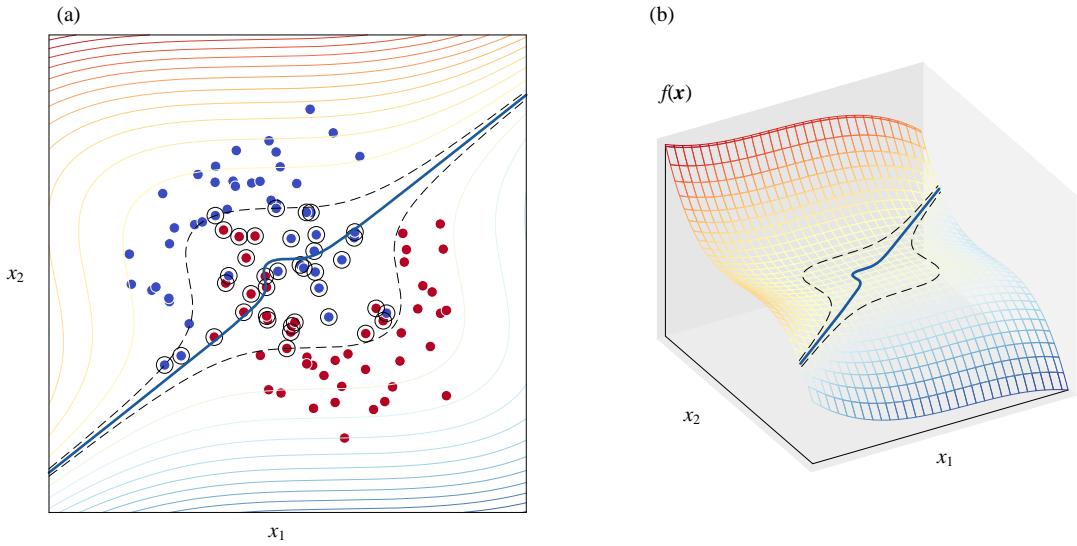


Figure 14. Crescent-shaped data classified by cubic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

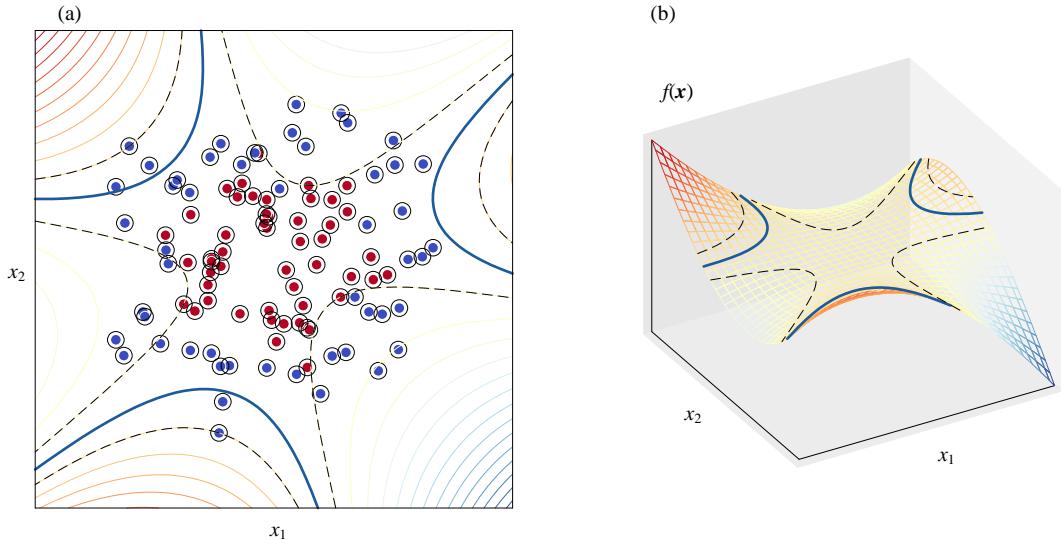


Figure 15. Ring-shaped data classified by cubic kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.5 Gaussian (RBF) Kernel: Infinite Dimensions Made Simple

19.5.1 Formula and Intuition

Gaussian kernel, also radial basis function (RBF) kernel, is widely used because it can handle non-linear data by implicitly mapping it to a higher-dimensional space. Its mathematical form is:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad (20)$$

Here, $\gamma > 0$ is a key parameter that controls the "spread" or "width" of the Gaussian function.

Intuitively, a small γ produces a wide, smooth surface, while a large γ produces a narrow, sharp surface, as shown in Figure 16. This affects how the SVM separates data points: smaller γ leads to smoother decision boundaries, while larger γ can fit more complex, detailed boundaries.

One reason the Gaussian kernel is powerful is that it corresponds to an infinite-dimensional mapping. If we attempt to explicitly expand the Gaussian function using a Taylor series, we get an infinite sum of polynomial terms.

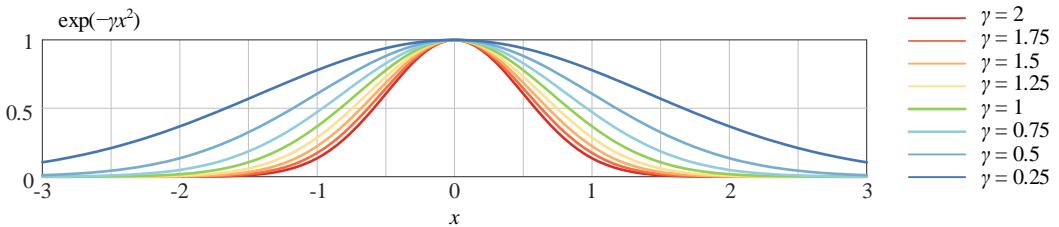


Figure 16. The parameter γ controls the width of the Gaussian kernel surface, with larger γ producing a narrower peak and smaller γ producing a wider, flatter surface.

19.5.2 Gaussian Hills and Decision Surfaces

Figure 17 shows the decision surface of an SVM using a Gaussian (RBF) kernel. Each support vector creates a smooth, hill-shaped influence on the surface, and the overall decision boundary is formed by combining these hills.

The peaks are centered at the support vectors, and the width of each hill is controlled by the parameter γ : larger γ produces narrower, sharper hills, while smaller γ produces wider, flatter hills. This creates a flexible surface that can adapt to complex, nonlinear patterns in the data, allowing the SVM to separate classes that are not linearly separable.

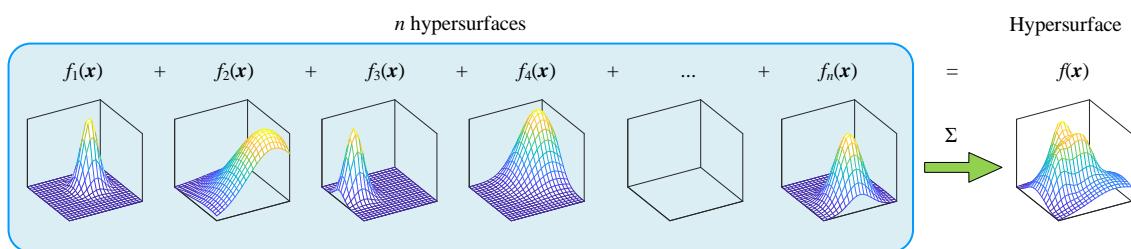


Figure 17. Gaussian kernel SVM decision surface forms a “Gaussian hill” structure, with each support vector contributing a smooth, peak-shaped influence. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.5.3 When RBF Kernels Shine

Gaussian kernel SVMs can create highly flexible decision boundaries, allowing them to classify data that is not linearly separable. For example:

- Linear separable data: produces a smooth boundary similar to a linear SVM (Figure 18).
- Crescent-shaped data: captures the curved structure accurately (Figure 19).
- Ring-shaped data: correctly separates inner and outer regions (Figure 20).

Compared with polynomial kernels of degree 2 or 3, Gaussian kernel SVMs typically achieve more accurate and versatile classifications because of their ability to adapt to complex, nonlinear structures in the data.

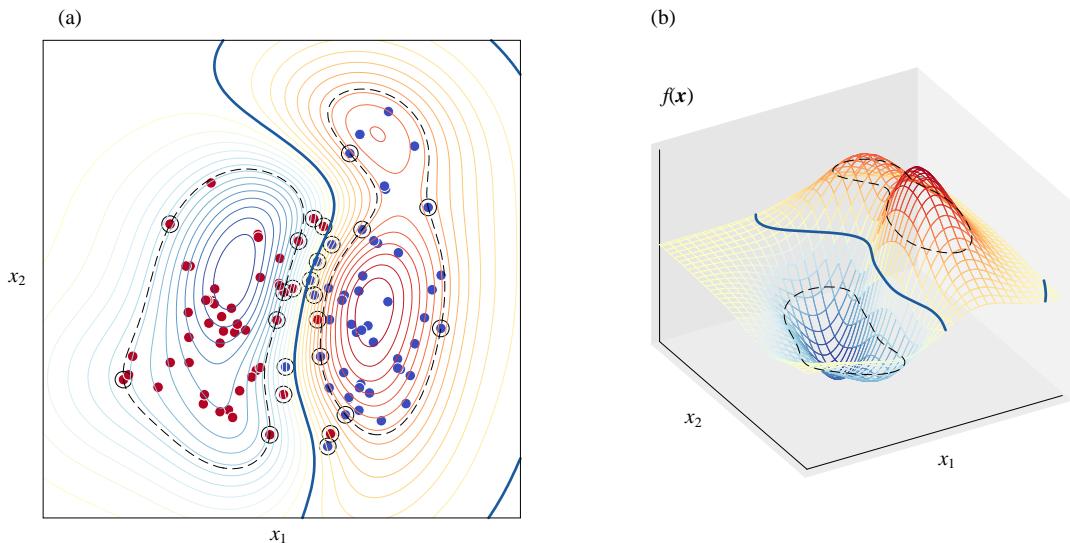


Figure 18. Linearly separable data classified by RBF kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

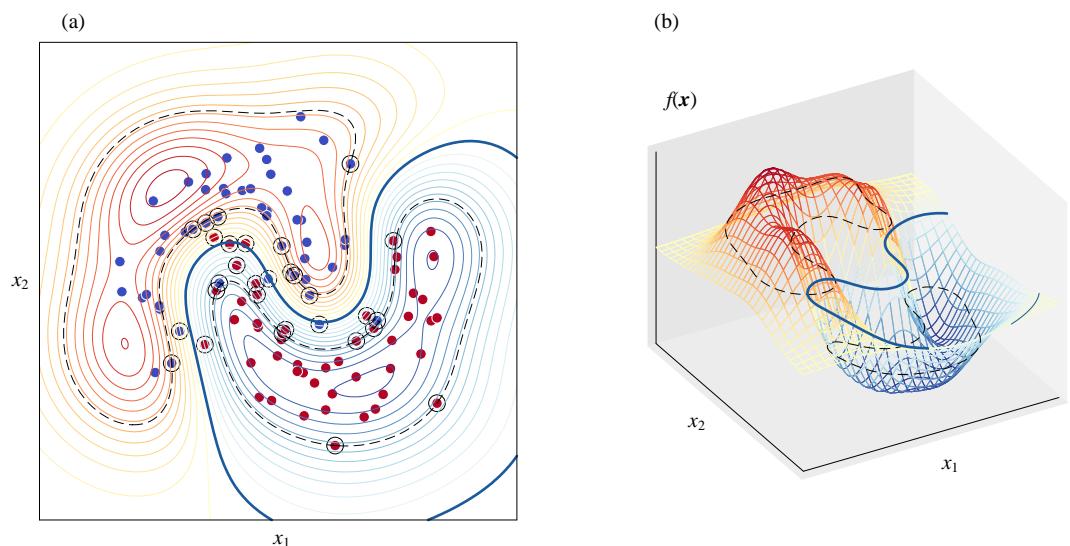


Figure 19. Crescent-shaped data classified by RBF kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

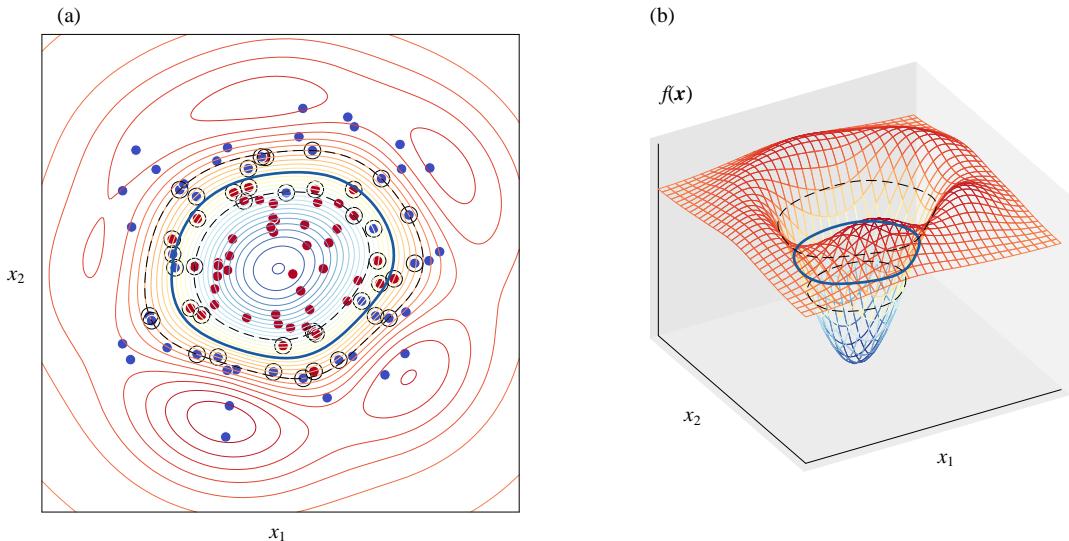


Figure 20. Ring-shaped data classified by RBF kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.6 Sigmoid Kernel: Learning with S-Shaped Curves

19.6.1 Mathematical Definition and Parameters

The sigmoid kernel, also called the S-shaped kernel, is defined as:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r) \quad (21)$$

Here, $\tanh()$ is the hyperbolic tangent function, which produces an S-shaped curve.

The parameters γ and r control the shape of this curve: γ adjusts the slope or “steepness” of the S, while r shifts the curve along the input axis.

Intuitively, a larger γ makes the curve steeper, which makes the SVM decision boundary more sensitive to small differences between data points, as shown in Figure 21.

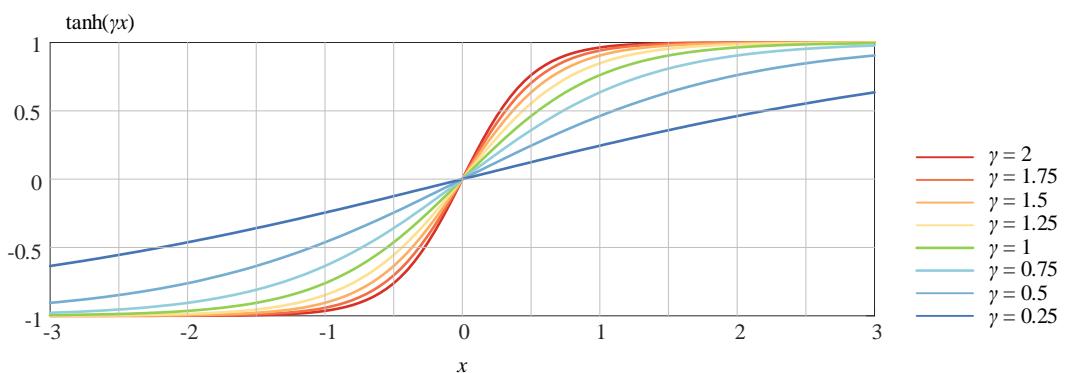


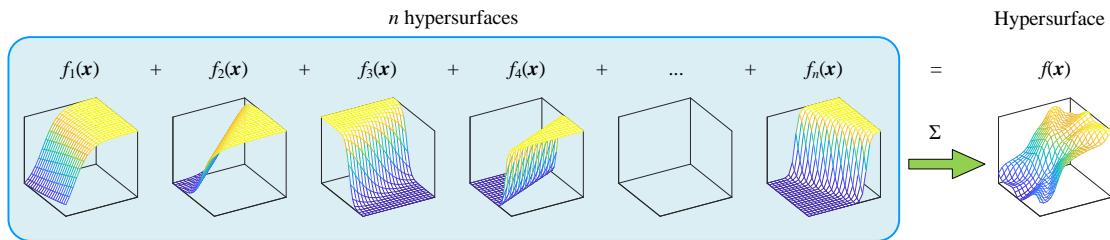
Figure 21. γ controls the steepness of the hyperbolic tangent curve, with larger γ producing a sharper S-shape and smaller γ producing a gentler slope. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.6.2 Intuitive Picture: Combining S-Shaped Surfaces

The decision function of an SVM with a sigmoid kernel is similar to other kernels:

$$f(\mathbf{x}) = \sum_{i=1}^n (\lambda_i y^{(i)} \tanh(\gamma \mathbf{x}^{(i)} \cdot \mathbf{x} + r)) + b = 0 \quad (22)$$

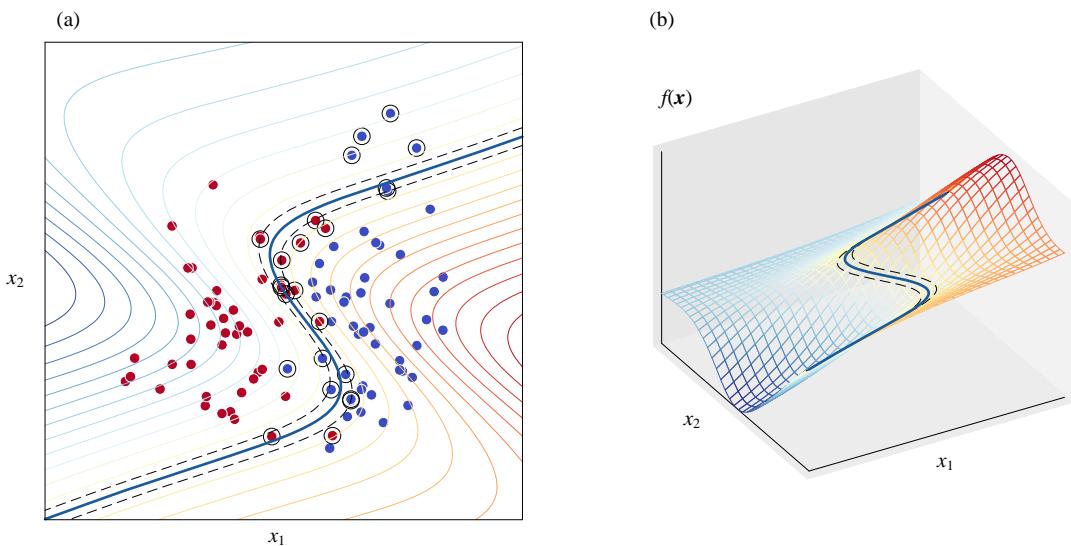
As shown in [Figure 22](#), each support vector contributes a small S-shaped surface to the overall decision surface. By combining these S-shaped surfaces, the SVM can form complex decision boundaries that adapt to the underlying data distribution.



[Figure 22](#). The overall sigmoid kernel surface is formed by combining multiple S-shaped surfaces contributed by the support vectors.

19.6.3 Comparing Sigmoid Kernels on Different Datasets

As shown in [Figure 23](#) ~ [Figure 25](#), similar to Gaussian kernels, sigmoid kernels are flexible and capable of constructing complex decision surfaces, allowing SVMs to capture subtle patterns in data.



[Figure 23](#). Linearly separable data classified by Sigmoid kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

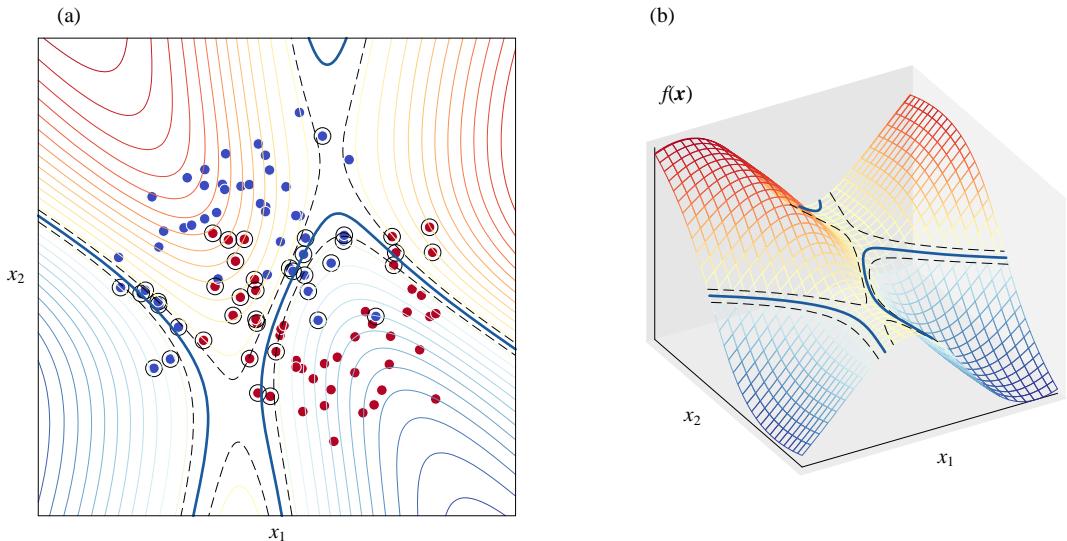


Figure 24. Crescent-shaped data classified by Sigmoid kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

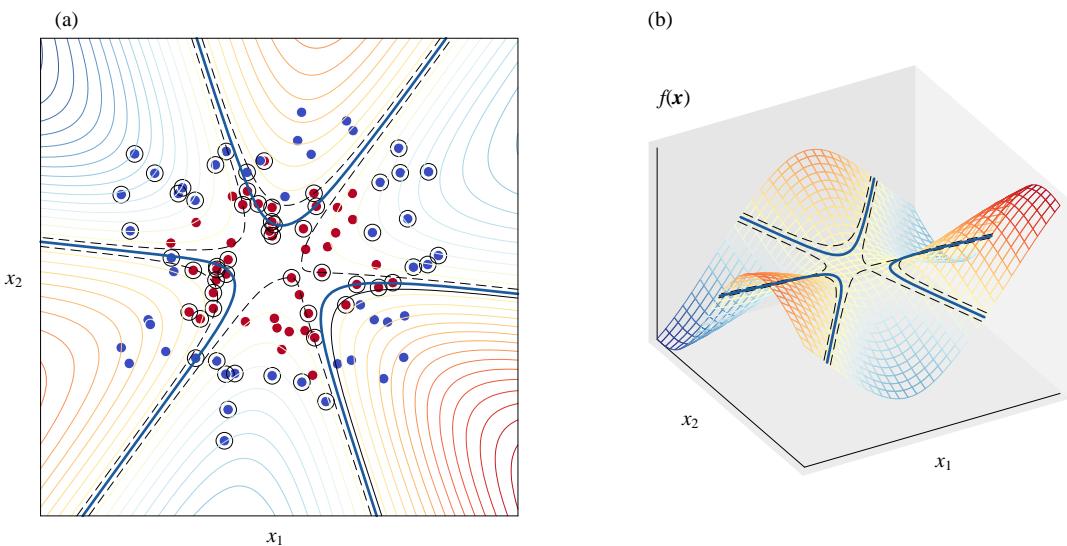


Figure 25. Ring-shaped data classified by Sigmoid kernel SVM. Figures generated by Ch19_01_Kernel_trick_in_SVM.ipynb.

19.7 Conclusion

This chapter introduced kernel functions in Support Vector Machines (SVMs), which allow SVMs to handle nonlinear data by implicitly mapping it into a higher-dimensional feature space. The kernel trick computes inner products in this space without explicitly performing the mapping.

The linear kernel produces flat hyperplane decision boundaries, suitable for linearly separable data but ineffective for nonlinear patterns like crescent or ring-shaped datasets. Polynomial kernels extend this by mapping data using polynomial functions, with higher degrees capturing more complex curves, though too high a degree may cause overfitting.

The Gaussian (RBF) kernel is highly flexible, with each support vector contributing a smooth, hill-shaped influence. The parameter γ controls the width of these hills: smaller γ creates wide, smooth surfaces, while larger γ creates narrow, sharp peaks. This allows SVMs to adapt to complex, nonlinear patterns.

The sigmoid kernel produces S-shaped surfaces using the hyperbolic tangent function. Parameters γ and r control the steepness and shift of the curve, and multiple S-shaped surfaces combine to form flexible decision boundaries.

Overall, kernel choice and parameter tuning determine the SVM's ability to separate classes. Linear kernels are simple, polynomial kernels capture moderate nonlinearity, and Gaussian and sigmoid kernels handle complex, nonlinear data effectively.