Kernels

Motivations

- Sometimes data is not linearly separable
- To make it linearly separable, we use a lifting trick where we apply a feature map $\phi(\vec{x})$ to the data

$$\phi(\vec{x}) = [x_1, x_2, x_1^2 + x_2^2]$$

Figure 1: Data without lifting.

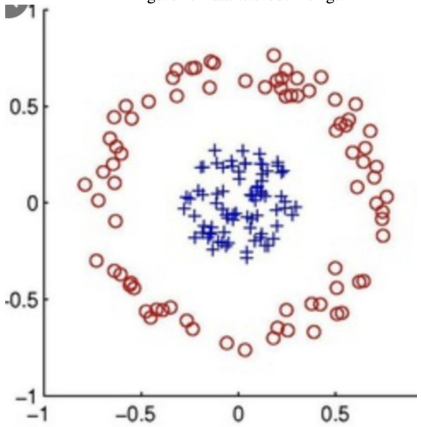
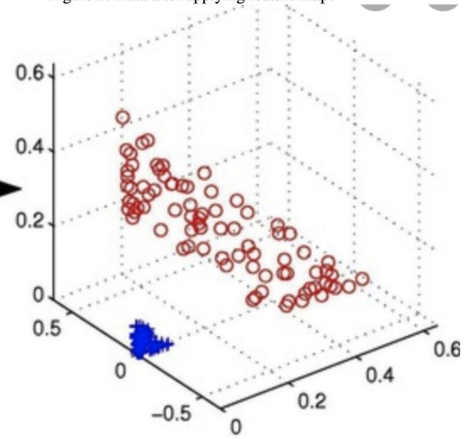


Figure 2: Data after applying feature map.



However, as we lift our data into even higher dimensions, computing the lifted features becomes very computationally expensive and infeasible. **Kernels** present a solution to this.

Review Ridge Regression

Given feature matrix $X \in \mathbb{R}^{n \times d}$ and vector $y \in \mathbb{R}^{n \times 1}$, to find the weights w such that

$$w^* = (X^T X + \lambda I)^{-1} X^T y$$

Sometimes it is advantageous to “lift” our features to a higher dimension d’ such that instead of the matrix X we now use $\Phi \in \mathbb{R}^{n \times d'}$ where d’ > d.

Problem

Computing takes $O(d'^2 n)$ and inverting takes $O(d'^3)$

Solution

We can instead express our solution like this:

$$w^* = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}y$$

Now we can use O(n^3) time to invert instead of O(‘d^3) to invert , $\Phi\Phi^T$ an n x n matrix

Derivation Part 1 $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$

$$(X^T X + \lambda I)\hat{w} = X^T y$$

$$X^T X\hat{w} + \lambda\hat{w} = X^T y$$

$$\lambda\hat{w} = X^T y - X^T X\hat{w}$$

$$\hat{w} = \frac{X^T y - X^T X\hat{w}}{\lambda}$$

$$\hat{w} = X^T \frac{y - X\hat{w}}{\lambda}$$

Derivation Part 2

$$\hat{w} = X^T \frac{y - X\hat{w}}{\lambda}$$

Let $$a = \frac{y - X\hat{w}}{\lambda}$$

$$w = X^T a$$

$$w = X^T a = X^T (XX^T + \lambda I)^{-1} y$$

$$\lambda a = y - X\hat{w} = y - XX^T a$$

$$y = \lambda a + XX^T a$$

$$y = (\lambda I + XX^T) a$$

$$a = (\lambda I + XX^T)^{-1} y$$

$$a = (XX^T + \lambda I)^{-1} y$$

What is a kernel function?

Here's what XX^T looked like in our earlier derivation

$$X = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \vdots & \\ — & x_n & — \end{bmatrix}$$

$$X^T = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_n \\ | & | & & | \end{bmatrix}$$

$$XX^T = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \vdots & \\ — & x_n & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ x_1^T & x_2^T & \dots & x_n^T \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \dots & \langle x_1, x_n \rangle \\ \langle x_1, x_n \rangle & \ddots & & \\ \vdots & & & \\ \langle x_n, x_1 \rangle & \dots & & \langle x_n, x_n \rangle \end{bmatrix}$$

We denote this Gram matrix as $K = XX^T$ and use a kernel function to define each inner product so that we don't need to know the data within X itself, making learning much more efficient

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_1, x_n) & \ddots & & \\ \vdots & & & \\ k(x_n, x_1) & \dots & & k(x_n, x_n) \end{bmatrix}$$

How can kernels be useful?

- Adding polynomials over original variables induces a richer space in which we can learn non-linear models.
- If we have a data matrix *X* with 2 features, we can lift to a higher dimension with this feature map

$$\phi(\vec{x}) = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 & \sqrt{2}x_1 & \sqrt{2}x_2 & 1 \end{bmatrix}$$

- We can define this corresponding kernel function

$$k(\phi(\vec{x}), \phi(\vec{z})) = \langle \phi(\vec{x}), \phi(\vec{z}) \rangle = (\phi(\vec{x})^T \phi(\vec{z}) + 1)^d$$

Reason

$$k(\phi(\vec{x}), \phi(\vec{z})) = \langle \phi(\vec{x}), \phi(\vec{z}) \rangle$$

$$= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 & \sqrt{2}x_1 & \sqrt{2}x_2 & 1 \end{bmatrix}^T \begin{bmatrix} z_1^2 & z_2^2 & \sqrt{2}z_1z_2 & \sqrt{2}z_1 & \sqrt{2}z_2 & 1 \end{bmatrix}$$

$$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1$$

$$= (x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2) + 2x_1z_1 + 2x_2z_2 + 1$$

$$= (x_1z_1 + x_2z_2)^2 + 2(x_1z_1 + x_2z_2) + 1$$

$$= (\vec{x}^T\vec{z})^2 + 2\vec{x}^T\vec{z} + 1$$

$$= (\vec{x}^T\vec{z} + 1)^2$$

By making our kernel function's complexity in terms of the original feature space and not the lifted feature space, we don't have to worry about what dimension we lift our data too.

What makes a kernel function usable and valid?

- The Gram Matrix must be **positive semi-definite and symmetric**
- A symmetric matrix $K = K^T$ is positive semi-definite if all of its eigenvalues are non-negative, i.e. $\lambda_i \geq 0$ for all eigenvalues λ_i of K. If a function satisfies the condition, then it expresses an inner product of two vectors

$$k(\vec{x}, \vec{z}) = \langle \phi(\vec{x}), \phi(\vec{v}) \rangle$$

where phi is some mapping of the original features.

Types of Kernels

1. Polynomial Kernel
2. RBF Kernel
3. Exponential Kernel

Polynomial Kernel

- A kernel approach to polynomial regression
- Speeds up learning process whenever the polynomial extends our data matrix from a tall matrix to a wide matrix, i.e more features than data points

$$k(\phi(\vec{x}), \phi(\vec{z})) = (\phi(\vec{x})^T \phi(\vec{z}) + 1)^d$$

Figure 1: Support vectors along with data points.

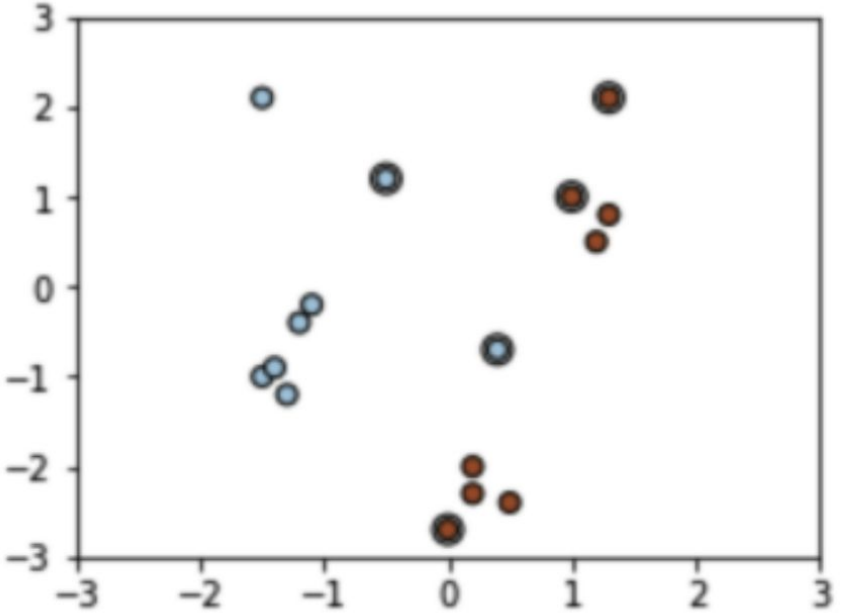


Figure 2: Decision boundary (solid line) and margins (dashed), Colored Mesh Faces

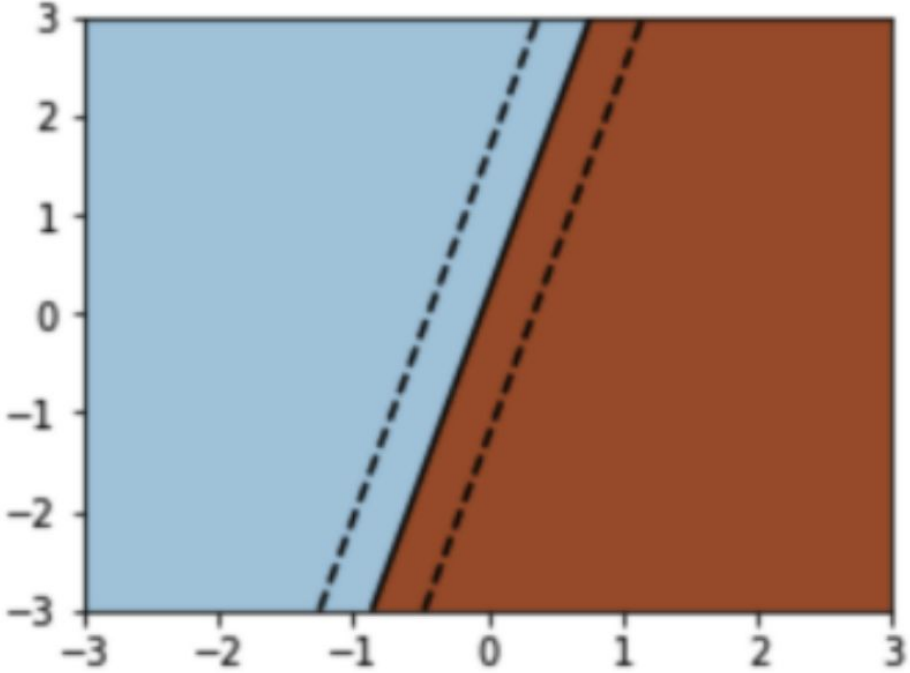


Figure 3: Soft-Margin SVM normally plotted with both plots overlaid.

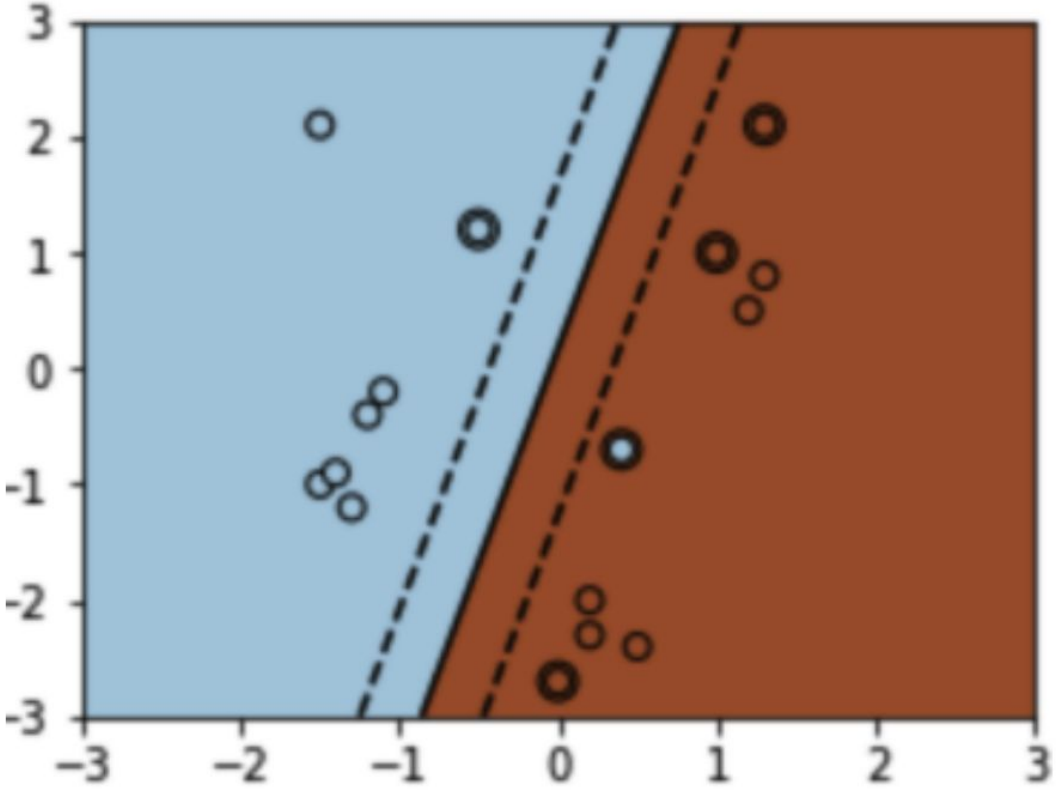
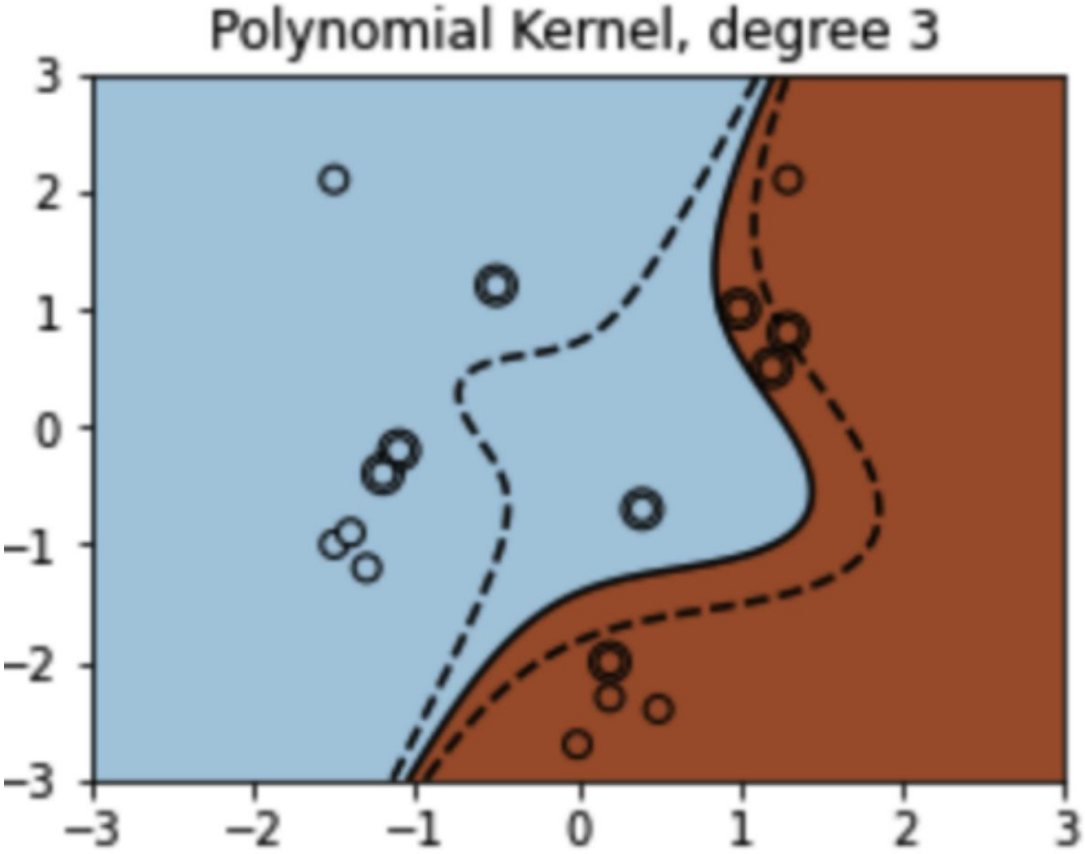


Figure 4: SVM with a polynomial kernel.



RBF Kernel

- Expresses a feature space with essentially infinite dimensions.
- Great when you don’t have prior information about your data.
- Best for approximating smooth functions

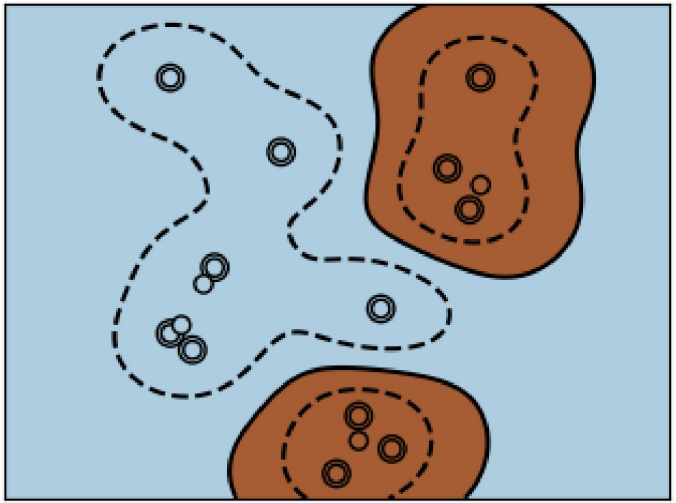
$$k(\vec{x}, \vec{z}) = exp(-\frac{||\vec{x} - \vec{z}||^2}{2\sigma})$$

$$k(\vec{x}, \vec{z}) = exp(-\gamma||\vec{x} - \vec{z}||^2)$$

RBF Kernel in Sci-Kit Learn

```
clf = svm.SVC(kernel='rbf', gamma=2)
clf.fit(X, Y)
```

Figure 5: Decision boundary of RBF kernel



Exponential Kernel

- Similar to the RBF kernel, but instead of using the L2 Norm squared it uses the L1 Norm.

$$k(\vec{x}, \vec{z}) = exp(-\gamma|\vec{x} - \vec{z}|_1)$$

```
gamma = 0.2
def exponential_kernel(x,z):
    return np.exp(- np.linalg.norm(x - z, ord=1) / gamma)
clf = svm.SVC(kernel=exponential_kernel)
clf.fit(X, Y)
```