

1 Non-linear Support Vector Machines – Kernels

Let's start by considering how we can use linear classifiers to make non-linear predictions. The easiest way is to first map all the examples $x \in \mathcal{R}^d$ into a different feature representation $\phi(x) \in \mathcal{R}^p$ where typically p is much larger than d . We would then simply use a linear classifier on the new (higher dimensional) feature vectors, pretending that they were the original input vectors. There are many ways to create such feature vectors. For example, we can build $\phi(x)$ by concatenating polynomial terms of the original coordinates. For example, in two dimensions, we could map $x = [x_1, x_2]^T$ to a five dimensional feature vector

$$\phi(x) = [x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T \quad (1)$$

We can then train a “linear” classifier (linear in the new ϕ -coordinates)

$$y = \text{sign}(\theta \cdot \phi(x) + \theta_0) \quad (2)$$

by mapping each training example to the corresponding feature vector. In other words, our training set is now $S_n^\phi = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \dots, n\}$. The resulting parameter estimates $\hat{\theta}, \hat{\theta}_0$ define a linear decision boundary in the ϕ -coordinates but a non-linear boundary in the original x -coordinates

$$\hat{\theta} \cdot \phi(x) + \hat{\theta}_0 \Leftrightarrow \hat{\theta}_1x_1 + \hat{\theta}_2x_2 + \hat{\theta}_3\sqrt{2}x_1x_2 + \hat{\theta}_4x_1^2 + \hat{\theta}_5x_2^2 + \hat{\theta}_0 = 0 \quad (3)$$

The non-linear boundary can represent, e.g., an ellipse in the original two dimensional space.

The main problem with the above procedure is that the feature vectors $\phi(x)$ can become quite high dimensional. For example, if we start with $x \in \mathcal{R}^d$, where $d = 1,000$, then compiling $\phi(x)$ by concatenating polynomial terms up to the 2^{nd} order would have dimension $d + d(d + 1)/2$ or about 500,000. Using higher order polynomial terms in such situations becomes quickly infeasible. However, it may still be possible to implicitly use such feature vectors. If training and prediction problems can be formulated only in terms of inner products between examples, then the relevant computation for us is $\phi(x) \cdot \phi(x')$. Depending on how we define $\phi(x)$, this computation can be carried out efficiently even if using $\phi(x)$ explicitly is not. For example, when $\phi(x) = [x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T$, we see that (check!)

$$\phi(x) \cdot \phi(x') = (x \cdot x') + (x \cdot x')^2 \quad (4)$$

$1000 + (1000(1000+1))/2$
 $\approx 500,000$

So the inner product is obtained easily from the original input vectors. One of the main advantages from considering the dual form of support vector machines is that it can be expressed entirely in terms of inner products. For example, in solving for the Lagrange multipliers $\hat{\alpha}_t$, the dual problem only requires us to evaluate inner products between the training examples, i.e., $(x^{(t)} \cdot x^{(t')})$. Similarly, the parameter $\hat{\theta}_0$ can be reconstructed from the margin constraints based on inner products. Finally, we can predict labels for new examples x with access only to the inner products between the training examples $x^{(t)}$ and the new points:

$$y = \text{sign}(\hat{\theta} \cdot x + \hat{\theta}_0) = \text{sign}\left(\sum_{t=1}^n \hat{\alpha}_t y^{(t)} (x^{(t)} \cdot x) + \hat{\theta}_0\right) \quad (5)$$

When we map examples into feature vectors, we replace $(x^{(t)} \cdot x^{(t')})$ with $\phi(x^{(t)}) \cdot \phi(x^{(t')})$. Instead of explicitly using feature vectors, we will focus on evaluating kernel functions $K(x^{(t)}, x^{(t')}) = \phi(x^{(t)}) \cdot \phi(x^{(t')})$.

2 Kernel Methods

We have discussed several linear prediction methods, including the perceptron algorithm, support vector machines, and linear (ridge) regression. All of these methods can be transformed into non-linear methods simply by mapping examples $x \in \mathcal{R}^d$ into feature vectors $\phi(x) \in \mathcal{R}^p$. Typically $p > d$ and $\phi(x)$ is constructed from x by appending polynomial (or other non-linear) terms involving the coordinates of x such as $x_i x_j, x_i^2$, and so on. The resulting predictors

$$\text{Perceptron :} \quad y = \text{sign}(\theta \cdot \phi(x) + \theta_0) \quad (6)$$

$$\text{SVM :} \quad y = \text{sign}(\theta \cdot \phi(x) + \theta_0) \quad (7)$$

$$\text{Linear Regression :} \quad y = \theta \cdot \phi(x) + \theta_0 \quad (8)$$

differ from each other based on how they are trained in response to (expanded) training examples $S_n = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \dots, n\}$. In other words, the estimated parameters $\hat{\theta}$ and $\hat{\theta}_0$ will be different in the three cases even if they were all trained based on the same data. Note that, in the regression case, the responses $y^{(t)}$ are typically not binary labels. However, there's no problem applying the linear regression method even if the training labels are all ± 1 . The only issue is the mismatch between what the model/method assumes (real valued responses) and what the data look like (e.g., binary labels).

The problem with explicitly mapping examples to feature vectors $\phi(x)$ is, as mentioned above, that they tend to be high dimensional, making it difficult to estimate and use the corresponding prediction methods. However, we have already seen how support vector machines can be cast entirely in terms of inner products or kernels $K(x, x') = \phi(x) \cdot \phi(x')$. These inner products can be sometimes evaluated substantially more efficiently than the associated feature vectors $\phi(x)$. In fact, we often focus precisely on kernels that can be evaluated efficiently. In such cases, we can implicitly work with very high (or even infinite) dimensional feature vectors.

For example, the feature vectors that specify the *radial basis function kernel* (or *RBF kernel*):

$$K(x, x') = \phi(x) \cdot \phi(x') = \exp(-\|x - x'\| / 2) \quad (9)$$

are indeed infinite dimensional (see below). It is quite easy to specify powerful classifiers or regression methods via kernels. The only requirement is that the prediction method can be cast entirely in terms of inner products.

Properties of Kernel Functions

Both the kernel SVM and kernel perceptron can be run with any valid kernel function $K(x, x')$. A kernel function is valid if and only if there exists some feature mapping $\phi(x)$ such that $K(x, x') = \phi(x) \cdot \phi(x')$. We don't need to know what $\phi(x)$ is (necessarily), only that one exists. We can build many common kernel functions based only on the following four rules

1. $K(x, x') = 1$ is a kernel function.
2. Let $f : \mathcal{R}^d \rightarrow \mathcal{R}$ be any real valued function of x . Then, if $K(x, x')$ is a kernel function, then so is $\tilde{K}(x, x') = f(x)K(x, x')f(x')$
3. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then so is their sum. In other words, $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel.
4. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then so is their product $K(x, x') = K_1(x, x')K_2(x, x')$

To understand these composition rules, let's figure out how they relate to the underlying feature mappings. For example, a constant kernel $K(x, x') = 1$ simply corresponds to $\phi(x) = 1$ for all $x \in \mathcal{R}^d$. Similarly, if $\phi(x)$ is the feature mapping for kernel $K(x, x')$, then $\tilde{K}(x, x') = f(x)K(x, x')f(x')$ (rule 2) corresponds to $\tilde{\phi}(x) = f(x)\phi(x)$. Adding kernels means appending feature vectors. For example, let's say that $K_1(x, x')$ and $K_2(x, x')$ correspond to feature mappings $\phi^{(1)}(x)$ and $\phi^{(2)}(x)$, respectively. Then (see rule 3)

$$K(x, x') = \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix} \cdot \begin{bmatrix} \phi^{(1)}(x') \\ \phi^{(2)}(x') \end{bmatrix} \quad (10)$$

$$= \phi^{(1)}(x) \cdot \phi^{(1)}(x') + \phi^{(2)}(x) \cdot \phi^{(2)}(x') \quad (11)$$

$$= K_1(x, x') + K_2(x, x') \quad (12)$$

Can you figure out what the feature mapping is for $K(x, x')$ in rule 4, expressed in terms of the feature mappings for $K_1(x, x')$ and $K_2(x, x')$?

Many typical kernels can be constructed on the basis of these rules. For example, $K(x, x') = x \cdot x'$ is a kernel based on rules (1), (2), and (3). To see this, let $f_i(x) = x_i$ (i^{th} coordinate mapping), then

$$x \cdot x' = x_1x'_1 + \dots + x_dx'_d = f_1(x)1f_1(x') + \dots + f_d(x)1f_d(x') \quad (13)$$

where each term uses rules (1) and (2), and the addition follows from rule (3). Similarly, the 2nd order polynomial kernel

$$K(x, x') = (x \cdot x') + (x \cdot x')^2 \quad (14)$$

can be built from assuming that $(x \cdot x')$ is a kernel, using the product rule to realize the 2nd term, i.e., $(x \cdot x')^2 = (x \cdot x')(x \cdot x')$, and finally adding the two. More interestingly,

$$K(x, x') = \exp(x \cdot x') = 1 + (x \cdot x') + \frac{1}{2!}(x \cdot x')^2 + \dots \quad (15)$$

is also a kernel by the same rules. But, since the expansion is an infinite sum, the resulting feature representation for $K(x, x')$ is infinite dimensional! This is also why the radial basis function kernel has an infinite dimensional feature representation. Specifically,

$$K(x, x') = \exp(-\|x - x'\|^2 / 2) \quad (16)$$

$$= \exp(-\|x\|^2 / 2) \exp(x \cdot x') \exp(-\|x'\|^2 / 2) \quad (17)$$

$$= f(x) \exp(x \cdot x') f(x') \quad (18)$$

where $f(x) = \exp(-\|x\|^2 / 2)$. The radial basis function kernel is special in many ways. For example, running the SVM with such a kernel function will always be able to return you a separable solution provided the training examples are all distinct.

Learning Objective

You need to know:

1. What is a kernel function and how it is used in the context of SVM.
2. What are the essential properties associated with a kernel function.
3. How to identify whether a function is a valid (or an invalid) kernel function.
4. What is the definition of a RBF kernel.

The Radial Basis Function Kernel

The **Radial basis function kernel**, also called the **RBF kernel**, or **Gaussian kernel**, is a kernel that is in the form of a radial basis function (more specifically, a Gaussian function). The RBF kernel is defined as

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp \left[-\gamma \|\mathbf{x} - \mathbf{x}'\|^2 \right]$$

where γ is a parameter that sets the “spread” of the kernel.

The RBF kernel as a projection into infinite dimensions

Recall a kernel is any function of the form:

$$K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

where ψ is a function that projects vectors \mathbf{x} into a new vector space. The kernel function computes the inner-product between two projected vectors.

As we prove below, the ψ function for an RBF kernel projects vectors into an infinite dimensional space. For Euclidean vectors, this space is an infinite dimensional Euclidean space.

That is, we prove that

$$\psi_{\text{RBF}} : \mathbb{R}^n \rightarrow \mathbb{R}^\infty$$

Proof:

Without loss of generality, let $\gamma = \frac{1}{2}$.

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp \left[-\frac{1}{2} \|\mathbf{x} - \mathbf{x}'\|^2 \right]$$

$$= \exp \left[-\frac{1}{2} \langle \mathbf{x} - \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle \right]$$

$$= \exp \left[-\frac{1}{2} (\langle \mathbf{x}, \mathbf{x} - \mathbf{x}' \rangle - \langle \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle) \right]$$

$$= \exp \left[-\frac{1}{2} (\langle \mathbf{x}, \mathbf{x} - \mathbf{x}' \rangle - \langle \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle) \right]$$

$$= \exp \left[-\frac{1}{2} (\langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{x}' \rangle - \langle \mathbf{x}', \mathbf{x} \rangle + \langle \mathbf{x}', \mathbf{x}' \rangle) \right]$$

$$= \exp \left[-\frac{1}{2} (\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\langle \mathbf{x}, \mathbf{x}' \rangle) \right]$$

$$= \exp \left[-\frac{1}{2} \|\mathbf{x}\|^2 - \frac{1}{2} \|\mathbf{x}'\|^2 \right] \exp \left[-\frac{1}{2} - 2\langle \mathbf{x}, \mathbf{x}' \rangle \right]$$

$$= C e^{\langle \mathbf{x}, \mathbf{x}' \rangle}$$

$$C := \exp \left[-\frac{1}{2} \|\mathbf{x}\|^2 - \frac{1}{2} \|\mathbf{x}'\|^2 \right] \text{ is a constant}$$

$$= C \sum_{n=0}^{\infty} \frac{\langle \mathbf{x}, \mathbf{x}' \rangle^n}{n!}$$

Taylor expansion of e^x

$$= C \sum_{n=0}^{\infty} \frac{K_{\text{poly}(n)}(\mathbf{x}, \mathbf{x}')}{n!}$$

We see that the RBF kernel is formed by taking an infinite sum over polynomial kernels.

As proven previously, recall that the sum of two kernels

$$K_c(\mathbf{x}, \mathbf{x}') := K_a(\mathbf{x}, \mathbf{x}') + K_b(\mathbf{x}, \mathbf{x}')$$

implies that the ψ_c function is defined so that it forms vectors of the form

$$\psi_c(\mathbf{x}) := (\psi_a(\mathbf{x}), \psi_b(\mathbf{x}))$$

That is, the vector $\psi_c(\mathbf{x})$ is a tuple where the first element of the tuple is the vector $\psi_a(\mathbf{x})$ and the second element is $\psi_b(\mathbf{x})$. The inner-product on the vector space of ψ_c is defined as

$$\langle \psi_c(\mathbf{x}), \psi_c(\mathbf{x}') \rangle := \langle \psi_a(\mathbf{x}), \psi_a(\mathbf{x}') \rangle + \langle \psi_b(\mathbf{x}), \psi_b(\mathbf{x}') \rangle$$

For Euclidean vector spaces, this means that $\psi_c(\mathbf{x})$ is the vector formed by appending the elements of $\psi_b(\mathbf{x})$ onto the $\psi_a(\mathbf{x})$ and that

$$\begin{aligned} \langle \psi_c(\mathbf{x}), \psi_c(\mathbf{x}') \rangle &:= \sum_i^{\text{dimension}(a)} \psi_{a,i}(\mathbf{x}) \psi_{a,i}(\mathbf{x}') + \sum_j^{\text{dimension}(b)} \psi_{b,j}(\mathbf{x}) \psi_{b,j}(\mathbf{x}') \\ &= \sum_i^{\text{dimension}(a)+\text{dimension}(b)} \psi_{c,i}(\mathbf{x}) \psi_{c,i}(\mathbf{x}') \end{aligned}$$

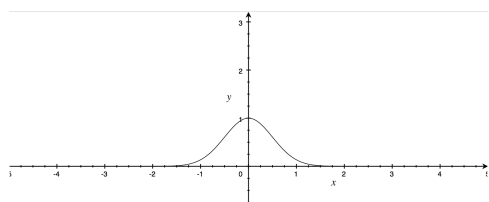
Since the RBF is an infinite sum over such appendages of vectors, we see that the projections is into a vector space with infinite dimension.

□

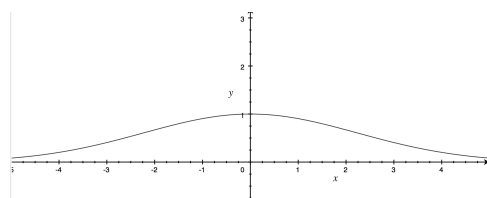
The γ parameter

Recall a kernel expresses a measure of similarity between vectors. The RBF kernel represents this similarity as a decaying function of the distance between the vectors (i.e. the squared-norm of their distance). That is, if the two vectors are close together then, $\|\mathbf{x} - \mathbf{x}'\|$ will be small. Then, so long as $\gamma > 0$, it follows that $-\gamma \|\mathbf{x} - \mathbf{x}'\|^2$ will be larger. Thus, closer vectors have a larger RBF kernel value than farther vectors. This function is of the form of a bell-shaped curve.

The γ parameter sets the width of the bell-shaped curve. The larger the value of γ the narrower will be the bell. Small values of γ yield wide bells. This is illustrated in Figure 1.



(a)



(b)

Figure 1: (a) Large γ . (b) Small γ .

W4 Recap

$$L(\vec{\theta}, \theta_0, \xi, \alpha, \beta) \max_{\substack{\alpha \geq 0 \\ \beta \geq 0}} \min_{\substack{\vec{\theta}, \theta_0, \xi}} \frac{1}{2} \|\theta\|^2 + \sum_{t=1}^n \alpha_t (1 - \underbrace{\xi_t - y^{(t)}(\vec{\theta} \cdot \vec{x}^{(t)} + \theta_0)}_f) + C \underbrace{\sum_{t=1}^n \xi_t}_0 + \sum_{t=1}^n \beta_t (-\xi_t)$$

symmetric because vector multiplication is commutative

\downarrow
 ξ becomes non-iv
 $-\xi$ becomes +v
 \downarrow
 $+ve$, spoil goal of min error.

$$\min \frac{1}{2} \alpha^T \begin{bmatrix} y^{(1)} y^{(1)} \vec{x}^{(1)} \vec{x}^{(1)} & y^{(2)} y^{(1)} \vec{x}^{(2)} \vec{x}^{(1)} & \dots & y^{(n)} y^{(1)} \vec{x}^{(n)} \vec{x}^{(1)} \\ y^{(1)} y^{(2)} \vec{x}^{(1)} \vec{x}^{(2)} & y^{(2)} y^{(2)} \vec{x}^{(2)} \vec{x}^{(2)} & \dots & y^{(n)} y^{(2)} \vec{x}^{(n)} \vec{x}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ y^{(1)} y^{(n)} \vec{x}^{(1)} \vec{x}^{(n)} & y^{(2)} y^{(n)} \vec{x}^{(2)} \vec{x}^{(n)} & \dots & y^{(n)} y^{(n)} \vec{x}^{(n)} \vec{x}^{(n)} \end{bmatrix} \alpha$$

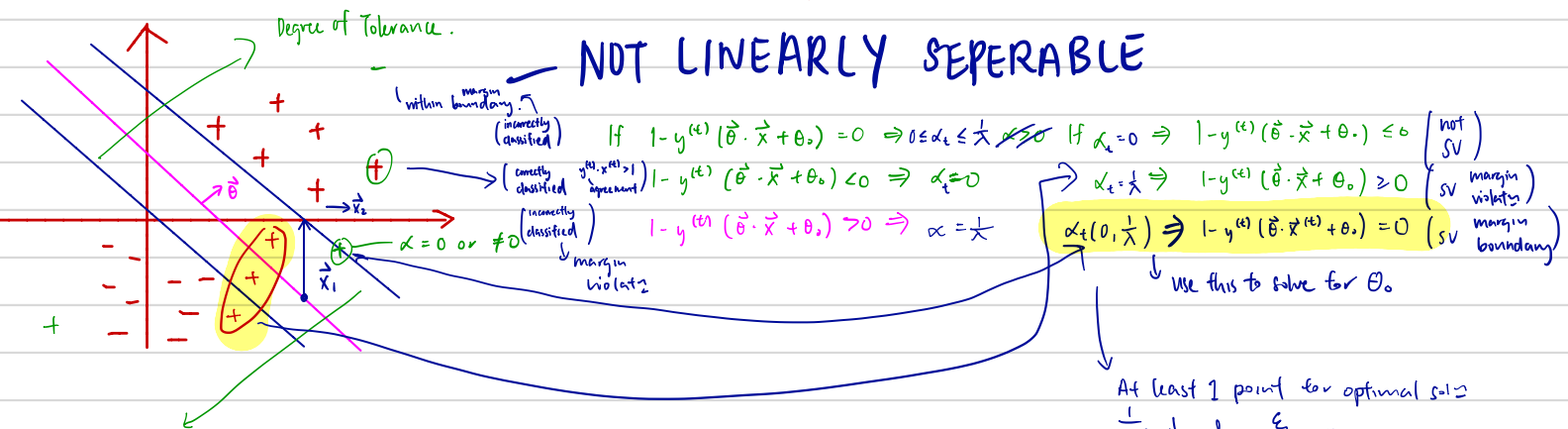
$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

↓ Pick any pt for support vector.

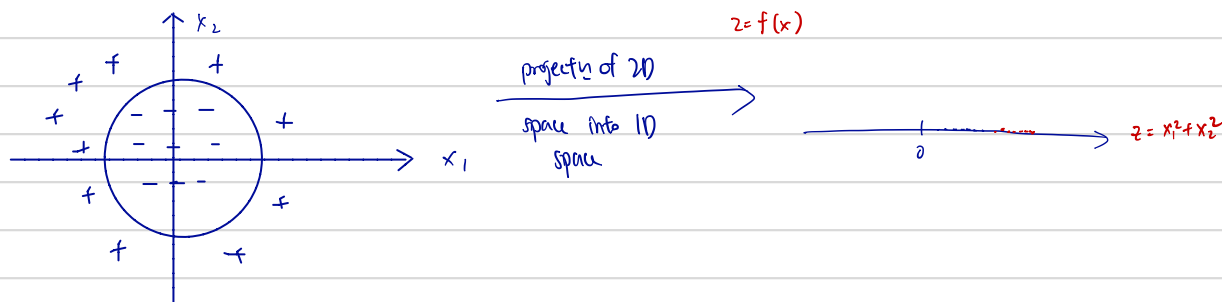
subject to : $\alpha_t \geq 0$
for $t = 1, \dots, n$

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = 0$$

$$\min \frac{\lambda}{2} \|\vec{\theta}\|^2 + \sum_{t=1}^n \xi_t \quad \text{subject to : } \begin{cases} \xi_t \geq 0 & \text{or} & -\xi_t \leq 0 \\ 1 - \xi_t - y^{(t)} (\vec{\theta} \cdot \vec{x}^{(t)} + \theta_0) \leq 0 \end{cases}$$



$$\xi_t = \max(0, 1 - y^{(t)} (\vec{\theta} \cdot \vec{x}^{(t)} + \theta_0))$$



$$\min_{\alpha \geq 0} \frac{1}{2} \sum_{t=1}^n \sum_{t'=1}^n \alpha_t \alpha_{t'} y^{(t)} y^{(t')} \frac{1}{\bar{x}^{(t)}} \cdot \frac{1}{\bar{x}^{(t')}} - \sum_{t=1}^n \alpha_t$$

$$1, \quad \frac{1}{\lambda} \geq \alpha_c \geq 0 \quad \text{for } |c| = 1, \dots, n$$

$$\sum_{t=1}^s \alpha_c y^{(t)} = 0$$

symmetric because vector multiplication is commutative

$$\min_{\alpha} \frac{1}{2} \alpha^T \begin{bmatrix} y^{(1)} y^{(1)} k(\vec{x}^{(1)}, \vec{x}^{(1)}) & y^{(2)} y^{(1)} k(\vec{x}^{(2)}, \vec{x}^{(1)}) & \dots & y^{(n)} y^{(1)} k(\vec{x}^{(n)}, \vec{x}^{(1)}) \\ y^{(1)} y^{(2)} k(\vec{x}^{(1)}, \vec{x}^{(2)}) & y^{(2)} y^{(2)} k(\vec{x}^{(2)}, \vec{x}^{(2)}) & \dots & y^{(n)} y^{(2)} k(\vec{x}^{(n)}, \vec{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ y^{(1)} y^{(n)} k(\vec{x}^{(1)}, \vec{x}^{(n)}) & y^{(2)} y^{(n)} k(\vec{x}^{(2)}, \vec{x}^{(n)}) & \dots & y^{(n)} y^{(n)} k(\vec{x}^{(n)}, \vec{x}^{(n)}) \end{bmatrix} \alpha$$

Properties of a kernel

- ① $k(\vec{x}, \vec{x}') = 1$ is kernel, $f(\vec{x}) = 1$
- ② if $k(\vec{x}, \vec{x}')$ is kernel, $g(\vec{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ i.e. $g(\vec{x}) \cdot k(\vec{x}, \vec{x}') \cdot g(\vec{x}')$
 $f(\vec{x}) \cdot f(\vec{x}')$

③ The sum of 2 kernels is a kernel

④ The product of 2 kernels is a kernel \rightarrow Go Home Ps.

Proof

$$\begin{aligned} \textcircled{3} \quad k_1(\vec{x}, \vec{x}') + k_2(\vec{x}, \vec{x}') \\ &= f_1(\vec{x}) \cdot f_1(\vec{x}') + f_2(\vec{x}) \cdot f_2(\vec{x}') \\ &= \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \end{bmatrix} \begin{bmatrix} f_1(\vec{x}') \\ f_2(\vec{x}') \end{bmatrix} \\ &= f(\vec{x}) \cdot f(\vec{x}') \end{aligned}$$

\nearrow can be -ve
 $k(\vec{x}, \vec{x}') = -1$ is / not a kernel
 If $x = x'$, $f(\vec{x}) \cdot f(\vec{x}) \neq -1$
 $\hookrightarrow 0 \in \|f(\vec{x})\|^2 \rightarrow$ not satisfied

$k(\vec{x}, \vec{x}') = 0 \rightarrow$ kernel from part 2.
 $k(\vec{x}, \vec{x}') = 10$

RBF Kernel — Radial Basis Function (Proof)

$$\begin{aligned} k(\vec{x}, \vec{x}') &= \exp(-\|\vec{x} - \vec{x}'\|^2 / 2) \quad \nearrow \text{use property 2 to proof.} \\ &= \exp\left(-\frac{\|\vec{x}\|^2}{2}\right) \exp(\vec{x} \cdot \vec{x}') \cdot \exp\left(-\frac{\|\vec{x}'\|^2}{2}\right) \end{aligned}$$

$$\begin{aligned} \exp(\vec{x} \cdot \vec{x}') \text{, use Taylor Expansion, } e^x &= \sum_{n=0}^{\infty} \frac{1}{n!} x^n \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} (\vec{x} \cdot \vec{x}')^n \rightarrow \text{kernel (Property 4)} \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \cdot (\vec{x} \cdot \vec{x}')^n \cdot \frac{1}{n!} \quad (\text{Property 2}) \\ &\quad \downarrow (\text{Property 3.}) \end{aligned}$$