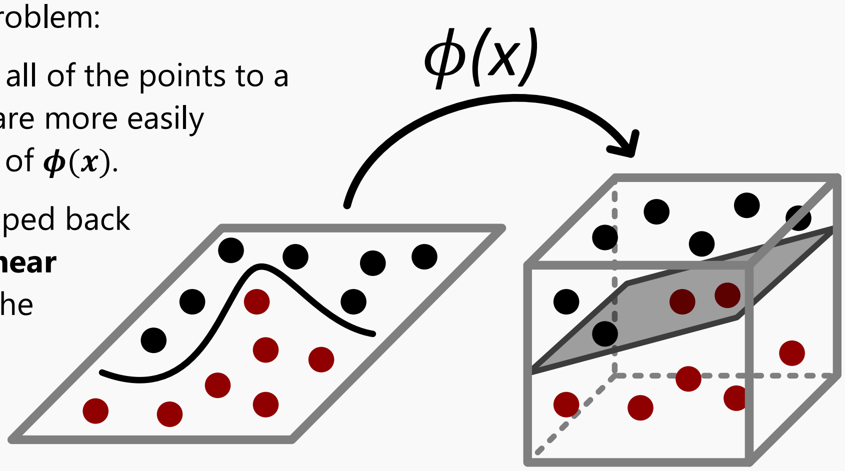
kernels for support vector machines (SVM)

Kernels allow for **Support Vector Machines** to convert from **linear models** to **nonlinear models** by utilizing a slightly different optimization problem:

The **Kernel "trick"** allows an **SVM** to map all of the points to a high dimensional space where the points are more easily separated; achieved through the mapping of $\boldsymbol{\phi}(\boldsymbol{x})$.



Then when the **decision boundary** is mapped back to the original dimensional space, a **nonlinear** boundary results that perfectly separates the data. The above illustration can be slightly misleading because the feature $\boldsymbol{\phi}(\boldsymbol{x})$ is almost never seen in the process.

An illustrative example of Kernels

Assuming a housing problem with the following features in the dataset:

$$x_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}, \dots] \rightarrow x_{i1} = \text{List Price}, x_{i2} = \text{Estimate}, x_{i3} = \text{Time on Market}, x_{i4} = \text{Avg Price}$$

For simplicity, the two features below will be of focus:

$$x_i = [x_{i1}, x_{i2}] \rightarrow x_{i1} = \text{House's List Price}, x_{i2} = \text{Zillow Estimate}$$

The **inner product** of features $[\boldsymbol{x_{i1}}, \boldsymbol{x_{i2}}]$ is calculated as a metric to understand **distance in space** (2D)

$$\textbf{Inner Product}\ (x_i, x_k) = \boldsymbol{x_i} \times \boldsymbol{x_k} = x_{i1}x_{k1} + x_{i2}x_{k2}$$

In attempts to translate the above **2 dimensional** example into a **3 dimensional** space, the points are mapped through the function $\boldsymbol{\phi}(\boldsymbol{x})$ as follows:

$$x_i = [x_{i1}, x_{i2}]$$

$$\boldsymbol{\phi}(\boldsymbol{x_i}) = \phi([x_{i1}, x_{i2}]) = [x_{i1}^2, x_{i2}^2, x_{i1}x_{i2}] \qquad (\textbf{2D} \rightarrow \textbf{3D}\ \text{for}\ \boldsymbol{i})$$

$$\boldsymbol{\phi}(\boldsymbol{x_k}) = \phi([x_{k1}, x_{k2}]) = [x_{k1}^2, x_{k2}^2, x_{k1}x_{k2}] \qquad (\textbf{2D} \rightarrow \textbf{3D}\ \text{for}\ \boldsymbol{k})$$

$\boldsymbol{x}$ is above being mapped from a **2D** to a **3D** space and the **inner product** can be computed in **3D**:

$$\textbf{Inner Product}\left(\boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_k})\right) = \boldsymbol{\phi}(\boldsymbol{x_i}) \times\ \boldsymbol{\phi}(\boldsymbol{x_k}) = x_{i1}^2, x_{i2}^2 + x_{k1}^2, x_{k2}^2 + x_{i1}x_{i2}x_{k1}x_{k2}$$

The above multiplies each dimension **component-wise** and computes the summation.

Therefore, one way to map to a high dimensional space with **SVM** is:

- Decide on the term $\boldsymbol{\phi}(\boldsymbol{x})$
- Replace all values of x_i in the optimization problem with $\boldsymbol{\phi}(\boldsymbol{x_i})$
- Solve for the new optimization problem

Alternate Kernel Application for SVM

There is a special property of the SVM optimization problem:

The problem only depends on $\boldsymbol{x_i}$ through the inner products

- The only terms involving $\boldsymbol{x_i}$ are $\textbf{Inner Product}\ (\boldsymbol{x_i}, \boldsymbol{x_k})$
- $\boldsymbol{x_i}$ will never occur alone or in other ways, they will only occur in inner products

The prior optimization problem derived for **SVM** is referred to as the **Primal Problem**. The alternate method introduced above is referred to as the **SVM Dual Form** of this problem:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_1 - \frac{1}{2} \sum_{i,\ k-1}^{m} \alpha_i \alpha_j y_i y_k \ \boldsymbol{x_i^T x_k} \ \longleftarrow \textbf{Inner Product}$$

$$\text{subject to } 0 \leq \alpha_j \leq C, \quad i = 1, \dots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

In the objective above, it is noted that the only occurrence of $\boldsymbol{x}$ is inside the **inner product**. Therefore, rather than determining what $\boldsymbol{\phi}$ is for the purpose of mapping the data to a high dimensional space, the **inner product** in the original space can be replaced with an **inner product** in a different space.

Wherever the formulation states to compute the $\textbf{Inner Product}\ (\boldsymbol{x_i}, \boldsymbol{x_k})$, it will be replaced with the $\textbf{Inner Product}\ \big(\boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_k})\big)$; There computation of $\boldsymbol{\phi}(\boldsymbol{x_i})$ is thus never required.

The question arises: How to compute the **inner products** in the new space without computing $\boldsymbol{\phi}$'s?

Using the previous example to illustrate the solution to the question:

$$x_i = [x_{i1}, x_{i2}]$$

$$\textbf{Inner Product}\ (x_i, x_k) = \boldsymbol{x_i} \times \boldsymbol{x_k} = x_{i1}x_{k1} + x_{i2}x_{k2}$$

$$\textbf{Inner Product}\ \big(\boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_k})\big) = \boldsymbol{\phi}(\boldsymbol{x_i}) \times \ \boldsymbol{\phi}(\boldsymbol{x_k}) = x_{i1}^2, x_{i2}^2 + x_{k1}^2, x_{k2}^2 + x_{i1}x_{i2}x_{k1}x_{k2}$$

$$\textbf{Inner Product}\ (x_i, x_k) \Longrightarrow \textbf{Inner Product}\ \big(\boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_k})\big)$$

It does not matter if $\boldsymbol{\phi}$ is infinite dimensional or impossible to calculate, as long as the **Kernel** can be calculated between two points, the optimization formula can be run.

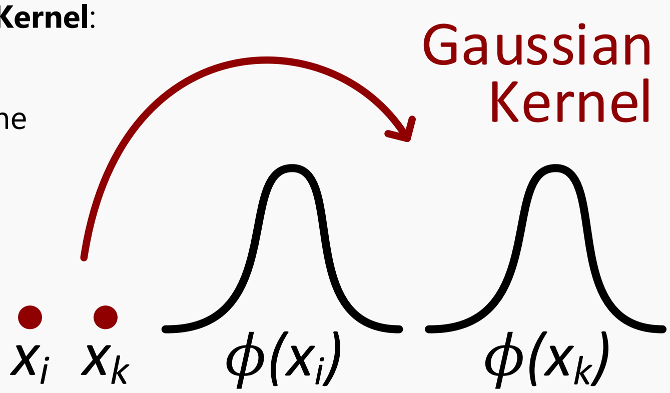
$$\textbf{Inner Product}\ (x_i, x_k) \Longrightarrow \textbf{Inner Product}\ \big(\boldsymbol{\phi}(\boldsymbol{x_i}), \boldsymbol{\phi}(\boldsymbol{x_k})\big) = \boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_k})$$

A particularly useful Kernel in practice is the **Gaussian Kernel**:

In this case, phi $\boldsymbol{\phi}$ cannot be effectively calculated on a computer due to its infinite dimensionality. However, the **inner product** can be calculated instead:

$$\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_k}) = \exp\left(-\frac{\|x_i - x_k\|_2^2}{2\sigma^2}\right), \text{where } \boldsymbol{\sigma} \text{ is selected}$$

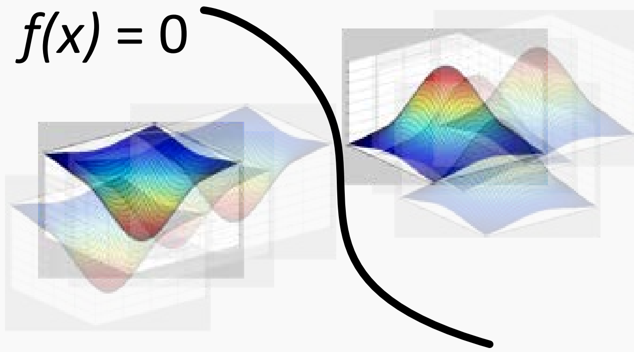
Therefore **Support Vector Machine** is now able to be performed in an **infinite dimensional space**.



When using a **Gaussian Kernel**, also referred to as a **Radial Basis Function (RBF) Kernel**, a **normalized**, **high-dimensional distribution** is essentially being applied to each datapoint. They are subsequently weights and summed. The distributions can be both **positive (+)** and **negative (-)** depending on the solution to the optimization problem. When a prediction is computed for a new observation in the dataset, a weighted summation of the normalized distributions is computed:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i) + b$$

→ the **weights** $\boldsymbol{\alpha_i}$ are determined by the solution to the SVM optimization problem



The **Decision Boundary** is simply constructed where the function $\boldsymbol{f(x) = 0}$ as illustrated →

The examples above illustrated how **Support Vector Machines** are able to compute **complex nonlinear decision boundaries.**

It is important to note that if the **bandwidth parameter** is too small, the model can **overfit** data.

The **bandwidth parameter** is the **standard deviation** of the **Gaussian Distribution**.

Common Kernels for Support Vector Machines

- Linear Kernel (synonymous to no kernel and using the regular inner product)
- Polynomial (the degree of polynomial is chosen and will overfit the data if too large)
- Gaussian Kernel (the bandwidth is chosen and will overfit the data if too large)

Advantages of Support Vector Machines

- Easily handles nonlinearities (with Kernels)
- Based on quadratic programing with specialized solvers (like coordinate descent with dual-adjustments to variables at a time)
- Reproducible results regardless of user or runs (theoretically)
- Easily handles imbalanced data by reweighting the points

Disadvantages of Support Vector Machines

- The solvers can sometimes be slow
- Does not naturally scale to larger datasets
- The performance is not as reliable in practice (even after adjusting the kernel parameters)
- Support Vector Machine models are uninterpretable

- The choice of Kernel can be difficult