# Kernel Machines

By Joseph Oluwasanya, DS4

1st November 2022

# Introduction

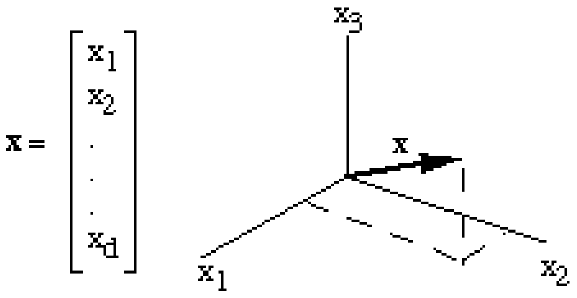
Kernel methods, or kernel machines refer to a class of learning algorithms which use kernel functions for pattern analysis.

Popularized with the development of the Support Vector Machine at AT&T Bell Labs by Vapnik et al. in the 90's.

# Concepts

For many algorithms for solving predictive modelling tasks, the data have to be transformed to a feature vector representation via feature maps defined by the user.

With Kernel machines, rather than a feature map, we must provide a kernel function instead. But what is a kernel function?



# What is a kernel function?

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ denote input data vectors of $n$ dimensions input space,

$\varphi(\boldsymbol{x})$ is a function mapping $\boldsymbol{x}$ and $\boldsymbol{y}$ from $n$-dimensional input space $\mathbb{R}^{\mathrm{n}}$ to an $m$-dimensional **inner product space** $\mathcal{V}$.

$$\varphi: \mathbb{R}^{\mathrm{n}} \rightarrow \mathcal{V}$$

# What is a kernel function? (Cont’d)

Kernel function, $K(x, y) = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathcal{V}}$ computes the inner product of $\varphi(\boldsymbol{x})$ and $\varphi(\boldsymbol{y})$.

$$K\colon\ \mathbb{R}^{\mathrm{n}} \times \mathbb{R}^{\mathrm{n}} \to \mathbb{R}$$

This gives us a similarity measure between the pair of inputs $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{\mathcal{X}}$

# Example

Let $x = (1, 2, 3), y = (4, 5, 6)$, and the function $\varphi(\boldsymbol{a}) = (a1a1, a1a2, a1a3, a2a1, a2a2, a2a3, a3a1, a3a2, a3a3)$.

So in this case,

$\varphi(\boldsymbol{x}) = (1, 2, 3, 2, 4, 6, 3, 6, 9)$ and
$\varphi(\boldsymbol{y}) = (16, 20, 2,4, 20, 25, 30, 24, 30, 36)$

# Example (Cont'd)

$\langle\varphi(\boldsymbol{x}), \varphi(\boldsymbol{y})\rangle = 1(16) + 2(20) + \ \ldots + 36(9) = 1024$

We then define the polynomial kernel $K(x, y) = (\langle x, y\rangle)^2$, and sub in $\boldsymbol{x}$, and $\boldsymbol{y}$ to get $K(x, y) = (4 + 10 + 18)^2 = 32^2 = 1024$

This is the same result, but is computationally cheaper than applying $\varphi$ computing the dot product.

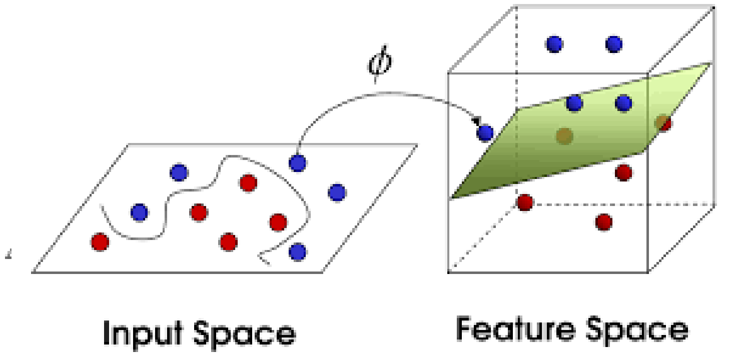
More on polynomial kernel here.

# Why do we even use $\varphi$?

As we saw earlier, $\varphi(\boldsymbol{x})$ is a function mapping data vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ from input space $\mathbb{R}^{\mathrm{n}}$ to an $m$-dimensional **inner product space** $\mathcal{V}$.

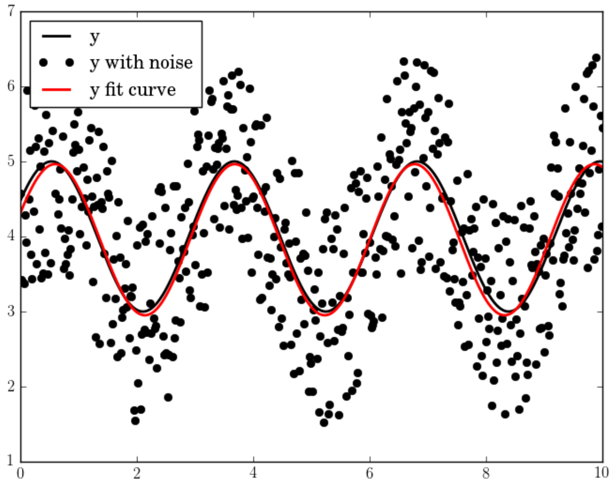
But why do we want to do this?

In the context of classification, kernel machines map data to higher dimensions in **attempt to make the target classes linearly separable**, and define a non-linear decision boundary in $\mathbb{R}^{\mathrm{n}}$.



# Why do we even use $\varphi$? (Cont'd)

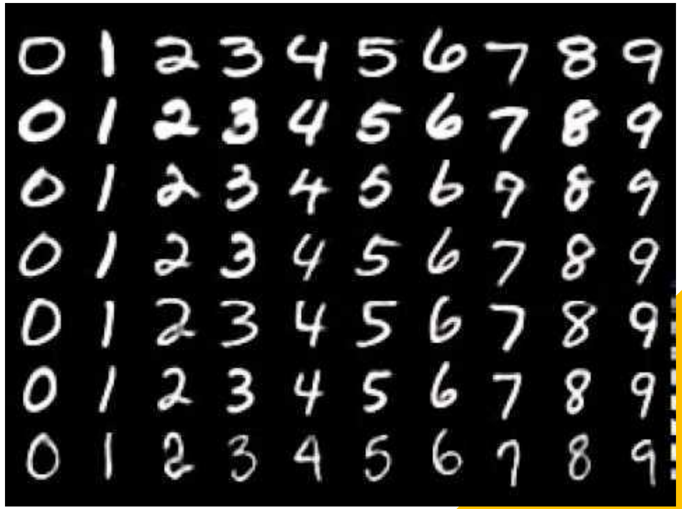
There is plenty of data in our world which can be modelled most effectively using non-linear functions, so a good choice of $\varphi$ will allow us to find a good non-linear fit/decision boundary, hence improving the performance of our model.



# Why do we even use $\varphi$? (Cont'd)

In the 90's, the support vector machine, a popular kernel machine algorithm, was found to be competitive with neural networks on tasks such as handwritten recognition.

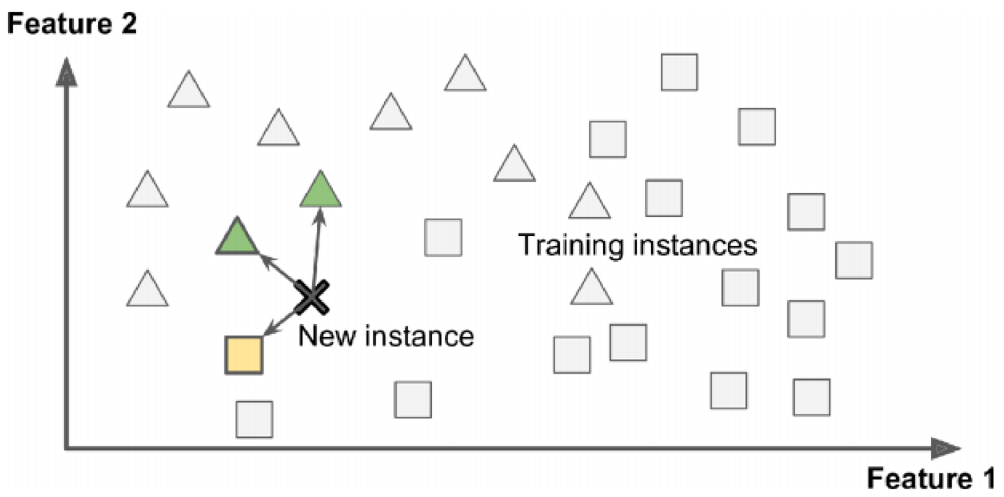
Today, it is still commonly used for classification and regression tasks.



# Instance-based learners

Kernel methods are **Instance-based learners**.

Rather than learning some fixed set of parameters corresponding to the input features (like a linear model), they compare new observations with those seen in training, which are stored in memory.



# Instance-based learners (Cont'd)

Kernel methods "remember" the $i$-th training example $(\boldsymbol{x_i}, y_i)$ and learn a corresponding weight $w_i$.

Prediction of unlabelled inputs is treated by the application of a **similarity function** $K$ which is the kernel, between the unlabelled instance $\boldsymbol{x}$' and each training input $\boldsymbol{x_i}$.

# Instance-based learners (Cont'd)

For example, a kernelized binary classifier typically computes a weighted sum of similarities.

$$\hat{y} = sgn \sum_{i=1}^{n} w_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}')$$

Where $\hat{y} \in \{-1, +1\}$ is the predicted label for unlabelled input $\boldsymbol{x}$' whose hidden true label $y$ is of interest.

$k$: $\mathbb{R}^{\mathrm{n}} \times \mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{R}$ is the kernel function that measures similarity between any pair of inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{\mathrm{n}}$

# Instance-based learners (Cont’d)

The kernelized binary classifier we just covered is called the “kernel perceptron”.

more information can be found here

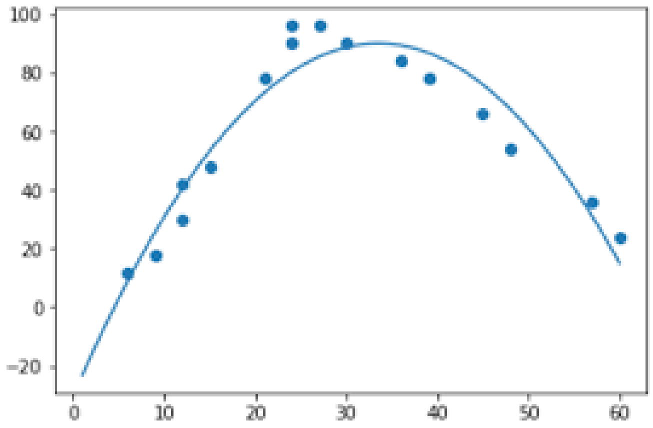
# The Kernel Trick (Motivation)

We've seen In order to compute non-linear functions or decision boundaries, linear learning algorithms require us to transform input data to higher dimensions.

E.g., linear regression model

$$\beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon = \hat{y}$$

Requires input vector $(x)$ to be mapped to $(x, x^2)$ for the 2nd order fit.



# The Kernel Trick (Motivation)

In the context of kernel machines, function $\varphi$ maps the feature vector to a higher dimensional inner product space.

This can be very demanding to compute, and in the end we compute the inner product to return 1-dimension.

# The Kernel Trick

The kernel trick **avoids computing the explicit mapping** of the input data to the inner product space $\mathcal{V}$, instead skipping this step and returning a scalar which is equivalent to computing the inner product in $\mathcal{V}$.

If $k: \mathbb{R}^{\mathrm{n}} \times \mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{R}$ is a kernel function, and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{\mathrm{n}}$, and $k$ can be written as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathcal{V}}$$

Then we can just compute k, which is cheaper than computing the RHS

# The Kernel Trick (Cont’d)

I used the word “if” in the last slide when explaining the kernel trick, but it was been proven mathematically, by James Mercer,  that any positive-definite kernel can be expressed as a dot product in a high-dimensional space. This is **Mercer’s Theorem.**

# The Kernel Trick (Cont'd)

If you have a function $\varphi$ such that $\langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathcal{V}}$ is a valid inner product in $\mathcal{V}$, then there is a kernel function that exists that achieve the result of the inner product for cheaper. Alternatively, if you have a positive definite kernel, you can deconstruct its implicit basis function $\varphi$.

Proof of Mercer's theorem can be found here

# Questions?

# Practical

The practical grouped with these slides will take you through how to implement support vector machines (SVM), a popular kernel machine algorithm. We use SVM to identify patients with malignant tumors.

See "*practical.ipynb*"

The original notebook was created Kaggle user *Fares Sayah, and can be found* *here*

# Additional reading

Helpful material for further learning about kernel machines. A lot of information in the slides originated from here: https://en.wikipedia.org/wiki/Kernel_method

Lili Jiang, example of kernel function (slide 5) : https://www.quora.com/What-are-kernels-in-machine-learning-and-SVM-and-why-do-we-need-them/answer/Lili-Jiang?srid=oOgT

Comprehensive explanation of the kernel trick, and Mercer's theorem. https://gregorygundersen.com/blog/2019/12/10/kernel-trick/