Nonparametrics and kernels (Part 3, The Kernel Trick)

Daniel Lawson University of Bristol

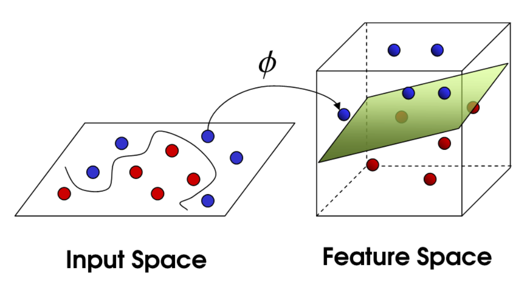
Lecture 04.1.3 (v1.0.1)

Signposting

- This is part 3 of Lecture 4.1, which is split into:
  - 4.1.1 covers Transforms
  - 4.1.2 covers Density estimation
  - 4.1.3 covers the Kernel Trick.

The Kernel trick - a Motivation

- What if there is a nonlinearity in the data?
- **Solution**: map the data into a higher dimensional space in which the relationship is (approximately) linear



The Kernel Trick

- **Problem**: High dimensional spaces are hard to work with and computationally costly
- **Solution**: Make the space **implicit**: all computation is done using a **Kernel** that uses a map $\phi : X \to \mathbb{R}^n$ for data in the original space $x, y \in X$:

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

- Kernels are any function that can be expressed as an **inner product**..

Kernel example

- **Input space** $X \subseteq \mathbb{R}^2$ with the **map**:

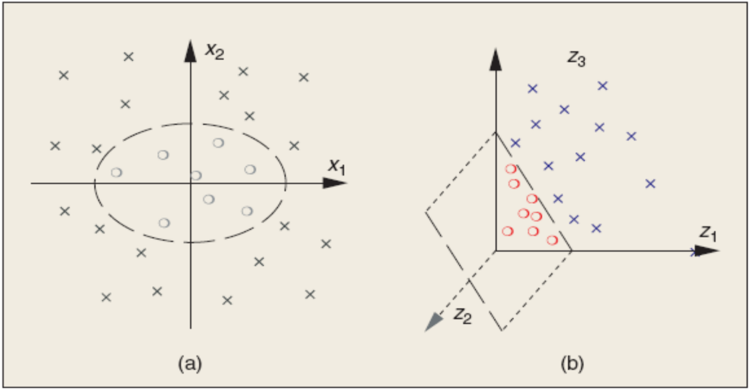
$$\phi : X = (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2) \in \mathbb{R}^3$$

- i.e. the **second moments**. Then:

$$\begin{aligned}
\langle \phi(x), \phi(y) \rangle = & \quad \langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (y_1^2, y_2^2, \sqrt{2}y_1y_2) \rangle & (1) \\
= & \quad (x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2) & (2) \\
= & \quad (x_1y_1 + x_2y_2)^2 = \langle \mathbf{x}, \mathbf{y} \rangle, & (3)
\end{aligned}$$

- i.e. the (squared) **dot product**.

Kernel examples[1]



1. Effect of the map $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. (a) Input space $\mathcal{X}$ and (b) feature space $\mathcal{H}$.

[1] Dave Krebs' class

Kernel properties

- Kernel spaces are **closed** under many operations.
- Being **closed** under f means that if x is in the space, $f(x)$ is also in the space.
- The operations are:
  1. Addition: $K(x, y) = K_1(x, y) + K_2(x, y)$
  2. Multiplication of a scalar: $K(x, y) = \alpha K_1(x, y)$
  3. Kernel Product: $K(x, y) = K_1(x, y)K_2(x, y)$
  4. Functional Product: $K(x, y) = f(x)f(y)$
  5. Kernel of a Kernel: $K(x, y) = K_3(\phi(x), \phi(y))$
  6. Matrix operation: $K(x, y) = x^T By$
- It is therefore possible to make **modular kernels**.

Gram Matrix

- The **Gram matrix** is used by many methods exploiting the Kernel Trick:

$$\mathbf{K} \equiv (k(x_i, x_j))_{ij}\,, \qquad \forall i, j$$

- This is a pre-computation: we compute the kernel between all pairs once, at the beginning, from which all subsequent computations follow.
- As long as the Gram matrices are positive semi-definite for all training sets. You can do the theory, or just check...
- The resulting space is called a **Reproducing Kernel Hilbert Space** (RKHS).
- It provides several important properties[2] and underpins many applications...

---

[2] Hofmann, Schoelkopf, & Smola (2008) "Kernel Methods in Machine Learning" (Ann. Stat.)

Important applications (later)

- Support Vector Machines
- Kernel Regression
- Kernel models on graphs (random walk, etc)
- Causal inference (Markov graphs)
- Kernel PCA

Kernel PCA

- For illustration we'll consider **kernel PCA**. Map $x_i \in \mathbb{R}^d$ to an arbitrary feature space $\phi(x_i) \in \mathbb{R}^n$ using the **Gram Matrix**:

$$K(x, y) = \phi(x)^T \phi(y)$$

- For which we'll consider the **eigenvector equation** for $v \in \mathbb{R}^n$:

$$Cv = \lambda v$$

- with the usual properties for the mean $\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) = 0$ and covariance $C = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \phi(x_i)^T$.

Kernel PCA continued

- Eigenvectors are **linear combinations** of the features: $v = \sum_{i=1}^{n} \alpha_i \phi(x_i)$.
- It turns out that kernel PCA requires only solving the **regular eigenvector problem** for the eigenvalues α_i of a Kernel matrix $\tilde{K}$:

$$\tilde{K}\alpha_i = \lambda_i \alpha_i$$

Because the feature space may not be mean centred, $\tilde{K} \neq K$ in general but is simply related:

$$\tilde{K} = K - 2\mathbf{1}_{1/n}K + \mathbf{1}_{1/n}K\mathbf{1}_{1/n}$$

- where $\mathbf{1}_{1/n}$ is a vector of length n with elements $1/n$.

Kernel PCA example

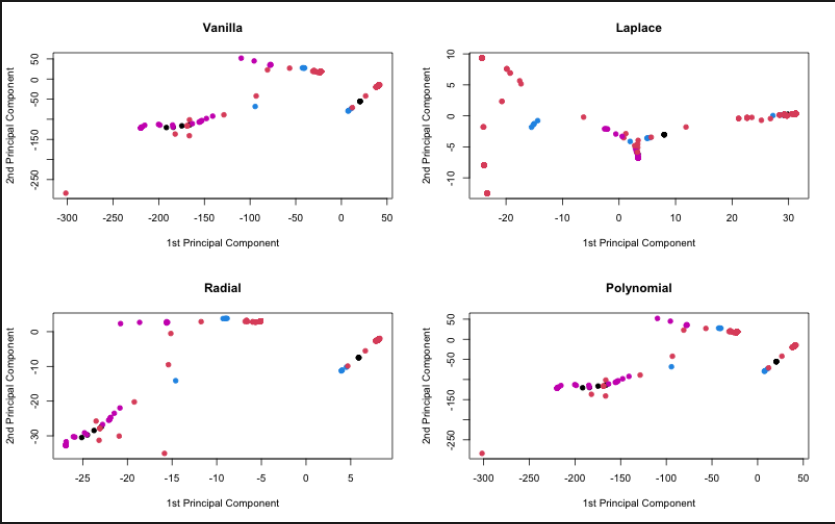
- See [3].

```
library("kernlab")
kpcvanilla=kpca(~.,data=testdata_sample,
    kernel="vanilladot",kpar=list(),features=4)
kpc=kpca(~.,data=testdata_sample,
    kernel="rbfdot",kpar=list(sigma=0.02),features=4)
kpclaplace=kpca(~.,data=testdata_sample,
    kernel="laplacedot",kpar=list(),features=10)
kpcpoly=kpca(~.,data=testdata_sample,
    kernel="polydot",kpar=list(),features=10)

plot(kpc@eig) # Plot eigenvalues
```

[3] Hofmann, Schoelkopf, & Smola (2008) "Kernel Methods in Machine Learning" (Ann. Stat.)

Kernel PCA example



Example Kernels:

- Linear Kernel: $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y} + c$
  - The regular dot product.
- Gaussian Kernel: $k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-|\mathbf{x}-\mathbf{y}|^2}{2\sigma^2}\right) + c$
  - Very susceptible to outliers due to the "narrow tails"
- Exponential Kernel: $k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-|\mathbf{x}-\mathbf{y}|}{2\sigma^2}\right) + c$
  - Also called the radial kernel
  - Related to the Laplacian kernel
- Power Kernel: $k(\mathbf{x}, \mathbf{y}) = -|\mathbf{x} - \mathbf{y}|^p$
  - conditionally positive definite, so needs extra care
- Log Kernel: $k(\mathbf{x}, \mathbf{y}) = -\log(|\mathbf{x} - \mathbf{y}|+1)$
  - conditionally positive definite, so needs extra care
- Histogram Intersection Kernel
- . . . and so on!

Thoughts on kernels

- The choice of Kernel is a **parameter**
- Which may itself **contain additional parameters**, e.g. bandwidths
- How to estimate? Evaluating performance requires calculating the whole N^2 matrix so it will be slow to iterate!
- Machine Learning thrives on usage cases where these decisions are either **relatively unimportant** or **determined by the method**.
- As we've seen, **adaptive** kernels such as nearest neighbour density estimation may be more robust than **parametric** kernels. Similar guidance holds here.

Reflection

- What is the benefit of the Kernel Trick? What is the cost?
- How would you apply it in practice?
- By the end of the course, you should:
  - Be able to perform basic computations with the 'Kernel Trick'
  - Be able to reason at a high level about the advantages and disadvantages of deploying the kernel trick for a particular cyber security example

Signposting

- In 4.2 we give some thought to the concept of **outliers** and **missing data**.
- References:
  - For the Kernel Trick Dave Krebs' Intro to Kernels
  - For the Kernel PCA: Rita Osadchi's Kernel PCA notes
  - Hofmann, Schoelkopf, & Smola (2008) "Kernel Methods in Machine Learning" (Ann. Stat.)
  - Schoelkopf B., A. Smola, K.-R. Mueller (1998) "Nonlinear component analysis as a kernel eigenvalue problem".