Nonparametrics and kernels

Daniel Lawson University of Bristol

Lecture 04.1.1 (v2.0.0)

Signposting

- We'll cover the following topics:
  - Nonparametric statistics - a birds eye view
  - Transforms - how to make good features
  - Density estimation
  - The Kernel Trick

Questions

- What is non-parametric statistics?
- Could we use a Fourier Transform in data science?
- How do we estimate density?
- Can we computationally compare in “infinite” dimensions?

Non-parametric statistics - overview

- Non-parametric statistics come in several flavours:
  1. Parameter-free hypothesis tests
  2. **Zero-parameter** representations which can be thought of as a **data transformation**.
     - examples include: Time-Frequency transforms, Kernel methods
  3. **Infinite-parameter** representations which can be thought of as generalisations of parametric models.
     - examples include: Hierarchical Dirichlet Process, the Stochastic Block Model for graphs
- We covered 1 in testing. We touch on 3 later. This lecture is about 2.
- Most methods are **parametric nonparametrics**: it is rare that a data transformation method isn't naturally thought of with a parameter!

Transforming data

- In previous practical problems we've used simple transforms to make the data easier to model:
  - log-transform
  - square-root/power transform
- Some data simplify greatly when transformed appropriately:
  - periodic data are simpler after taking a frequency transform
- Transformed data can be seen as feature augmentation, latent embedding, depending on use.
- Generally, the goal is to make the noise **additive** so that it **averages out**.

The Basis Expansion

- Most transforms we consider are designed to exactly reproduce the data.
- These are **basis expansions** and are typically invertible.
- They make good feature sets if they result in a **dimensionality reduction**;
  - that is, they lead to a useful approximation using only a few features.
- PCA is one example of this.
- There are many others. . .

Fourier transform

- The Fourier transform is written:

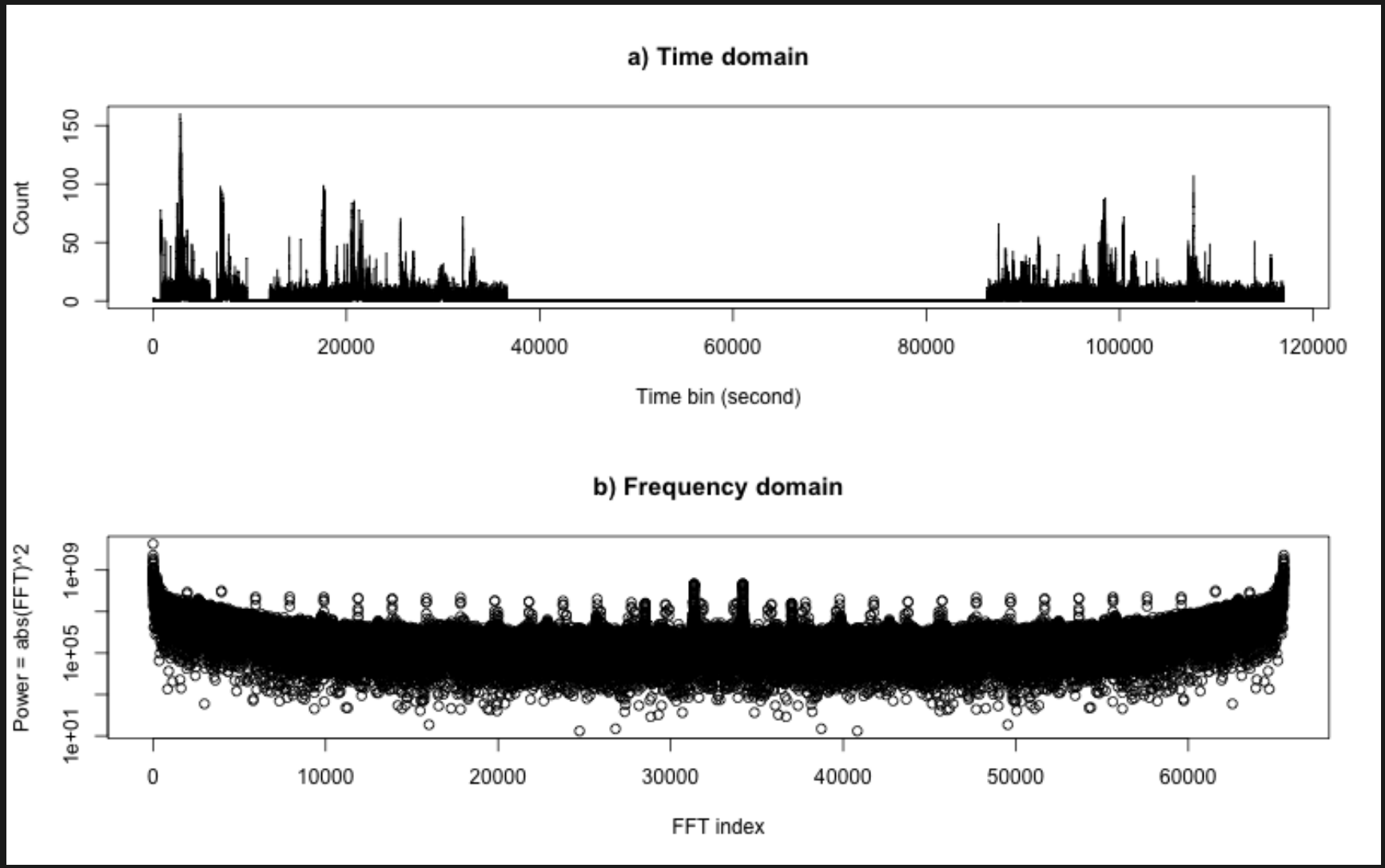
$$\hat{f}(\eta) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \eta} dx$$

  - The **Discrete Fourier Transform (DFT)** is used in practice as datasets typically have a minimum sampling rate δ.
  - It is usually computed using the **Fast Fourier Transform (FFT)**.
  - Consider using it for periodic data, or to look for periodicity.
- The **power** in any frequency i is proportional to $|\hat{f}(\eta_i)|^2$.
  - High power means this frequency is present in your data.
  - There are formal tests for "significance" of high power.

Fourier transform example

```
conndata_ts=data.frame(t=seq(min(conndata$ts),
            max(conndata$ts),by=1),x=0)
for(i in 1:dim(conndata)[1]){
    conndata_ts[ceiling(conndata[i,"ts"]-
                        conndata_ts[1,"t"]),"x"] =
            conndata_ts[ceiling(conndata[i,"ts"]-
                        conndata_ts[1,"t"]),"x"] + 1
}
# Not fast unless length(x)=2^k
myx=1:(2^16) # Largest valid choice
conndata_fft=fft(conndata_ts[myx,"x"])
```

Fourier transform example



Walsh-Hadamard transform

- The Walsh-Hadamard transform is a version of the Fourier Transform that is useful for **Binary data**.
- It is defined recursively via the **Hadamard Matrix**:

$$H_0 = 1,$$

$$H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}$$

- For N total bits, the whole matrix is of size $2^m \times 2^m = N \times N$.
- The transform is $\mathbf{w} = \mathbf{H}\mathbf{x}$.
- $\mathbf{w}$ can be computed efficiently with the **fast** Walsh-Hadamard transform in complexity $O(N \log(N))$.
- It was developed in encryption & signals processing but is useful to generate features in many contexts.

Walsh-Hadamard matrices

$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Walsh-Hadamard matrices

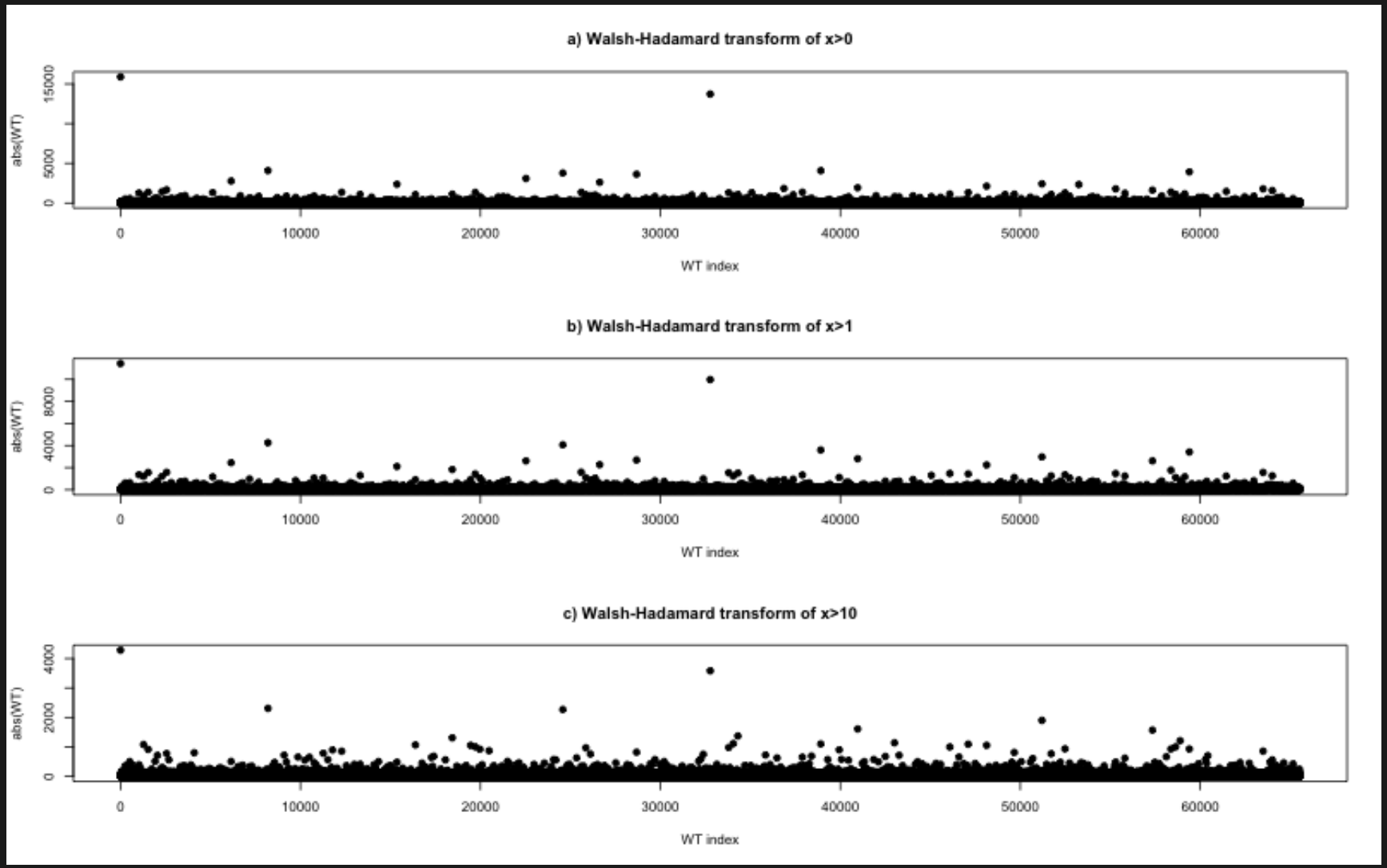
$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H_2 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Walsh-Hadamard transform examples

- Examples:
  - 00000... -> 00000...
  - 11111... -> +0000...
  - 01010... -> +-000...
  - 10101... -> ++000...
  - 00010001... -> ++++000....
- i.e. the i-th bit is **activated by periodicity** of length i
- The details are sensitive to the "phase", i.e. exactly where in the sequence the periodicity lies.

Walsh-Hadamard transform example



Other transforms

- Other transforms can be useful. For example:
  - Wavelets (time and space decomposition)
  - Laplace transform
  - Sine/ Cosine transforms
  - Hankel transform (radial basis function)
  - Polynomials
  - ... etc
- All you need is a **basis function** and you have a **transform**.

Density estimation overview

- Density estimation is an extremely hard problem because it maps **discrete data** into a **continuous estimator**.
- This is only conceptually well founded if we are willing to make (strong) **assumptions** regarding smoothness.
- There is no way to perform an entirely data-driven analysis!
- Many popular methods are very bad if, for example, the smoothness varies by location.

Kernel density estimation (KDE)

- Let $\{\vec{x}_i\}_{i=1}^N$ be a dataset on some space (for simplicity taken as $\mathbb{R}^d$).
- Then the Kernel K provides the density estimate for **any point** $\vec{y}$ as:

$$f_{\mathbf{H}}(\vec{y}) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}}\left(\vec{y} - \vec{x}_i\right),$$

  where $\mathbf{H}$ is a matrix of bandwidths.
- In other words, its a **sum of independent contributions** from each datapoint.
- It can be written:

$$K_{\mathbf{H}}\left(\vec{y} - \vec{x}_i\right) = \frac{1}{\det(\mathbf{H})} K\left(\mathbf{H}^{-1}(\vec{y} - \vec{x}_i)\right)$$

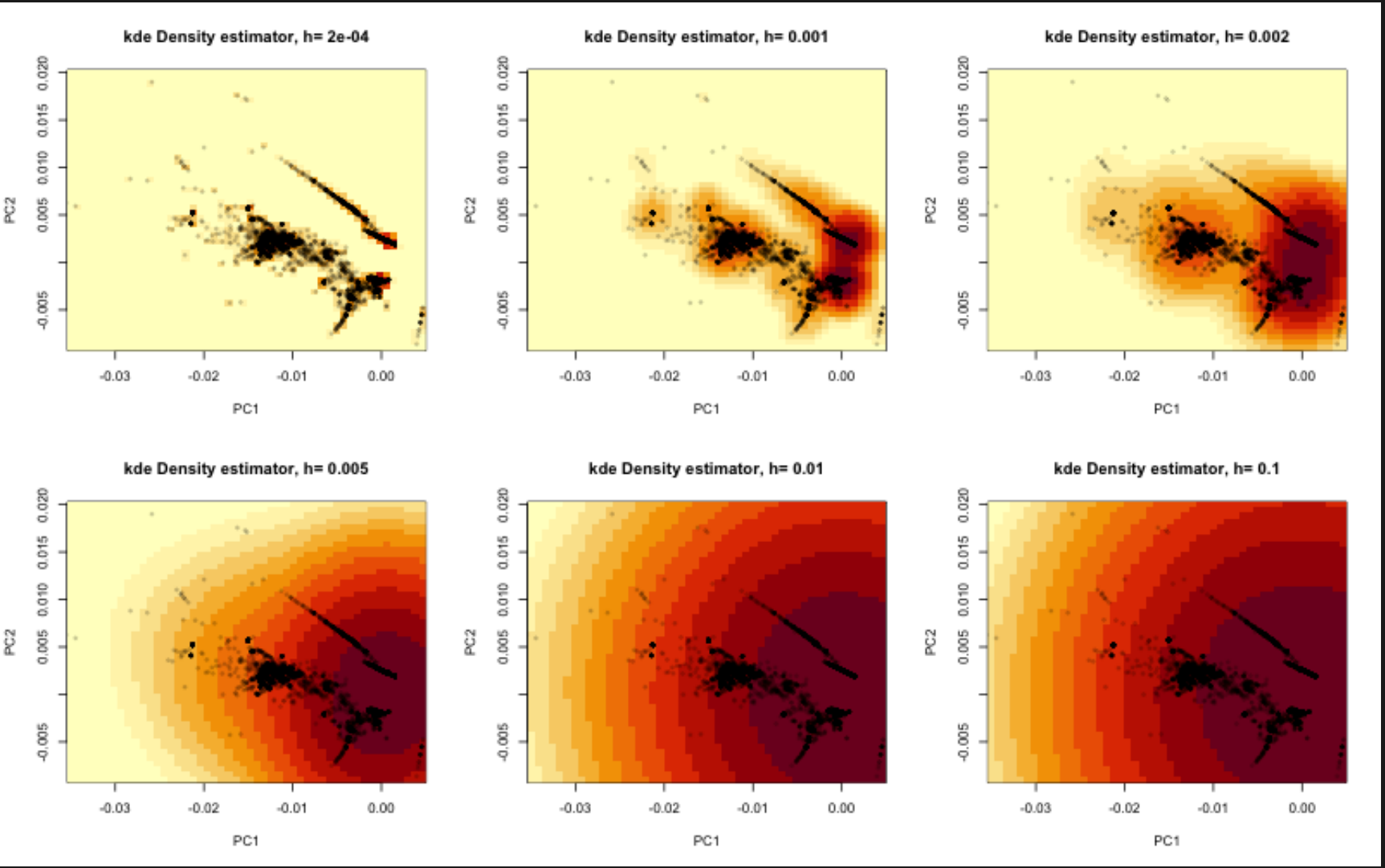
KDE in 1d

- In 1D:

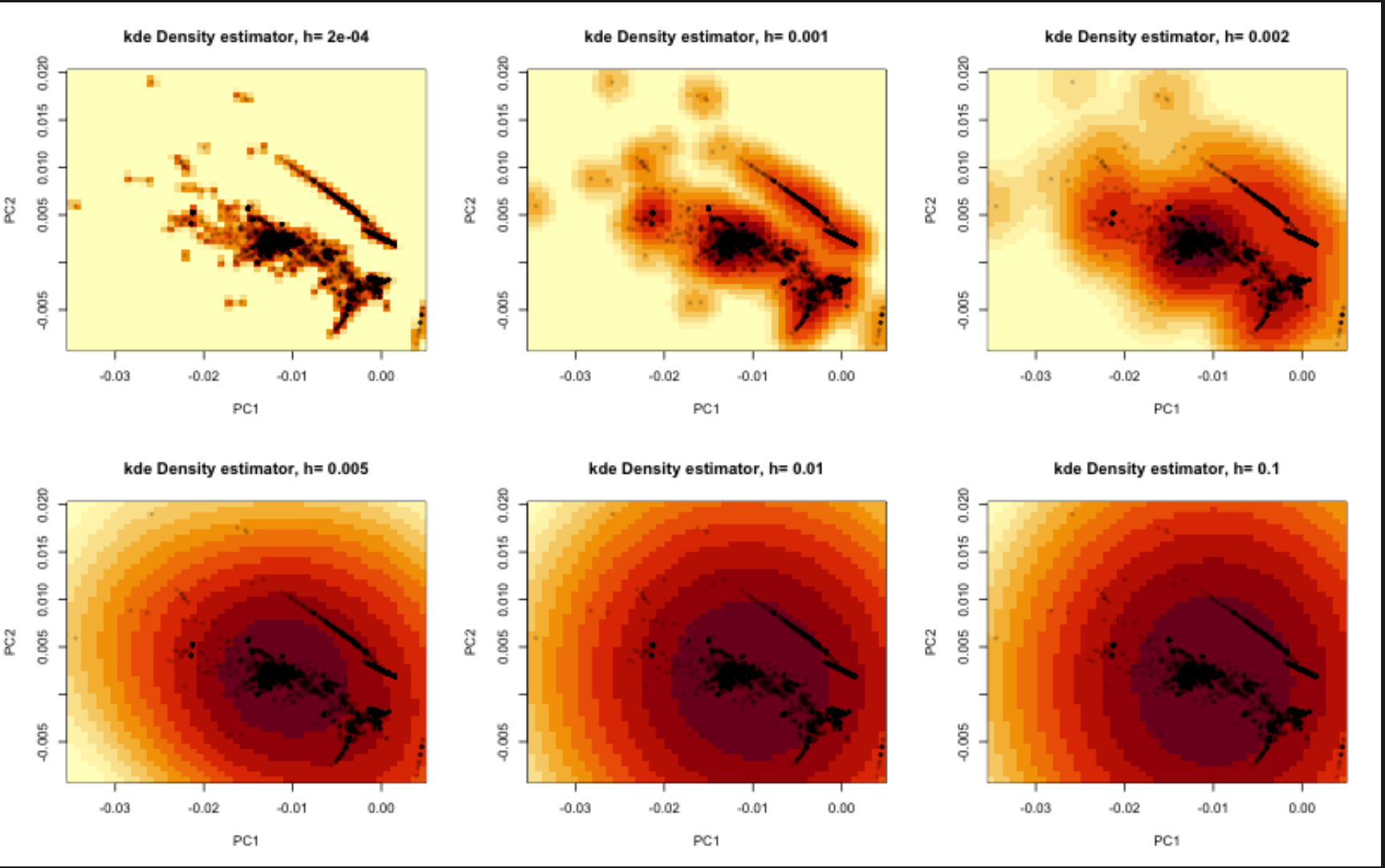
$$f_h(\vec{y}) = \frac{1}{N} \sum_{i=1}^{N} K\left(\frac{\vec{y} - \vec{x}_i}{h}\right)$$

  - Its common to use a Normal kernel $K(x) = \text{Normal}(x; \mu = 0, \sigma = 1)$.
  - h can be chosen by minimising the "Mean Integrated Square Error"...
  - which theoretically suggests a functional form $h \propto N^{-1/5}$.
- Most density tools in packages use a reasonable default (which also depends on dimension).
  - This is appropriate for statistical inference of the density estimate at an unspecified point x.
- In practice the **"right" bandwidth** is a function of the **question**, so defaults might work poorly.
  - For **EDA**, we often want a **smaller bandwidth** to reveal potential data features

KDE Example


kde Density estimator, h= 2e-04
kde Density estimator, h= 0.001
kde Density estimator, h= 0.002
kde Density estimator, h= 0.005
kde Density estimator, h= 0.01
kde Density estimator, h= 0.1
PC1
PC2
-0.03
-0.02
-0.01
0.00
-0.005
0.005
0.010
0.015
0.020

KDE with unique points



KDE kernels

- Some **important multivariate kernels**:
  - Spheroid Gaussian ($\mathbf{H}$ and Σ are diagonal)
  - Rectangular ($\mathbf{H}$ is diagonal, Uniform kernel)
  - Product Gaussian ($\mathbf{H}$ off-diagonals are products, Σ is diagonal)
- $\mathbf{H}$ is a parameter. It can be estimated by Cross-Validation but it is high dimensional so this is hard.

Applications of KDE

- Kernel density estimates are considered important in many applications, including:
  - Smoothing
  - Clustering
  - Topological Data Analysis
  - Level set estimation
  - Feature Extraction
  - ... etc!

K-Nearest neighbours

- Measuring neighbourhoods is a very important component of many applications.
- A fast way to do this is by computing k-Nearest neighbours (k-NN) for each point.
- Note the requirement for a **distance measure** (metric or otherwise).
- Algorithms to do this are called **nearest neighbour search**:
  - **Linear algorithms**: Check all distances for all points. $O(N^2)$ to compute the structure.
  - **Space partitioning**: KD-trees etc partition the space. $O(N \log(N))$ but are less good in high dimensions...
  - **Approximate methods**: there are many great methods for this problem, which are often nearly perfect and much faster. **Locality Sensitive Hashing** is popular.

k-NN density estimation

- A **Density estimate** using **k-NN**:

$$\hat{p}_{kNN}(x) = \frac{k}{N} \cdot \frac{1}{V_d R_k^d(x)}$$

- where:
  - d is the dimension of the space,
  - k is the number of neighbours,
  - N is the sample size,
  - $R_k^d(x)$ is the "radius", i.e. the distance to the k-th closest neighbour of x, and
  - V_d is the volume of a unit ball:

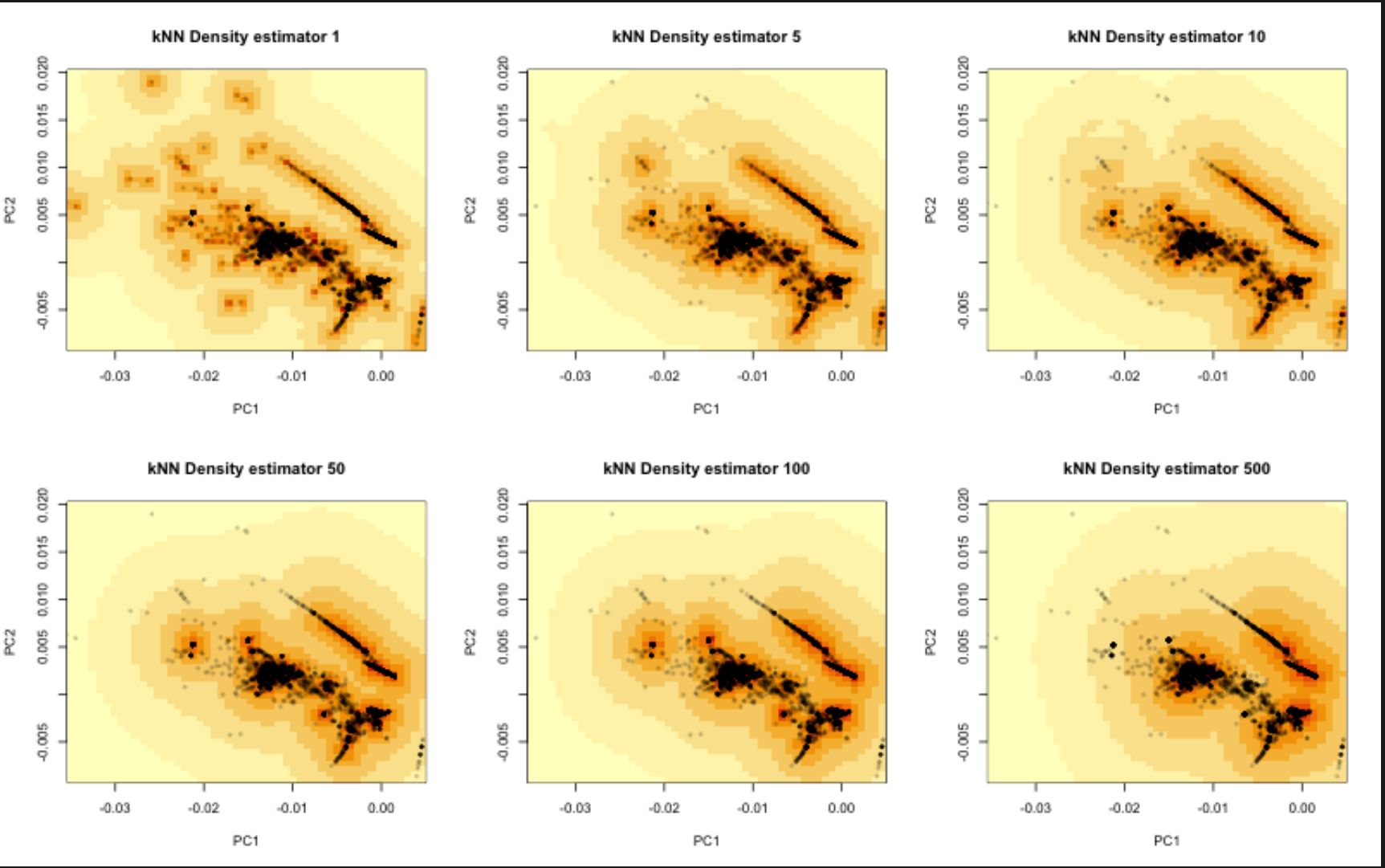
$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

- so $V_1 = 2$, $V_2 = \pi$, $V_3 = \frac{4}{3}\pi$.
- NB Like the distance function, k is a **parameter**!

k-NN density estimation

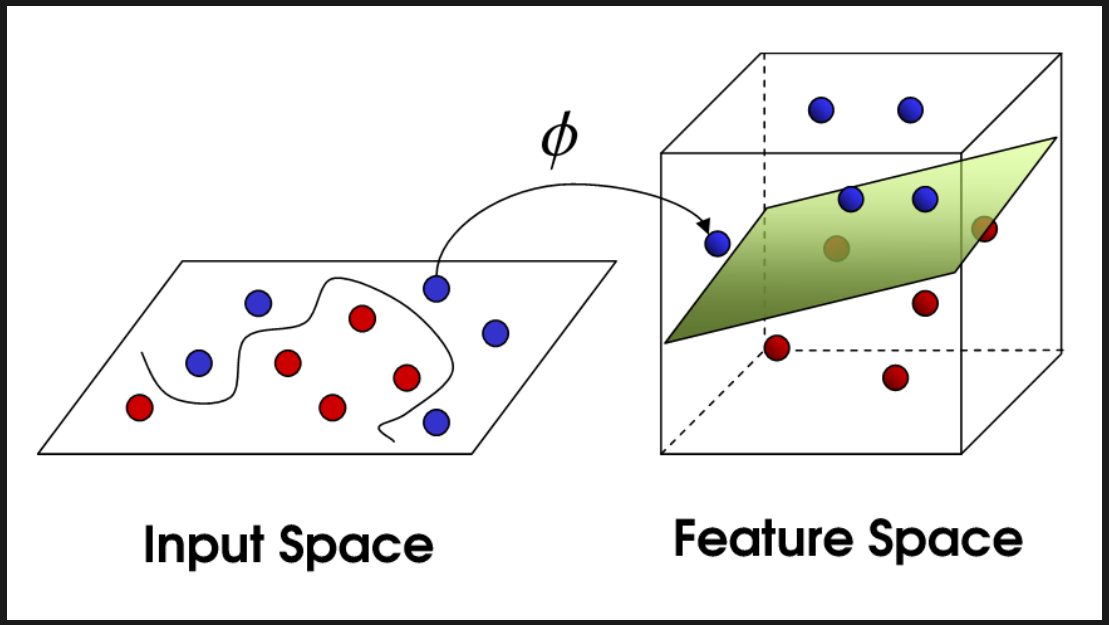
```
library("TDA")
Xseq <- seq(-0.035, 0.0046, length.out=50)
Yseq <- seq(-0.009, 0.02, length.out=50)
Grid <- expand.grid(Xseq, Yseq)

klist=c(1,2,5,10,20,50)
knnlist=lapply(klist,function(k){
    KNN <- knnDE(testdata_all.svd$u[,1:2], Grid, k)
    KNNm=matrix(KNN,nrow=length(Xseq),ncol=length(Yseq))
})
```

k-NN density estimation



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**:

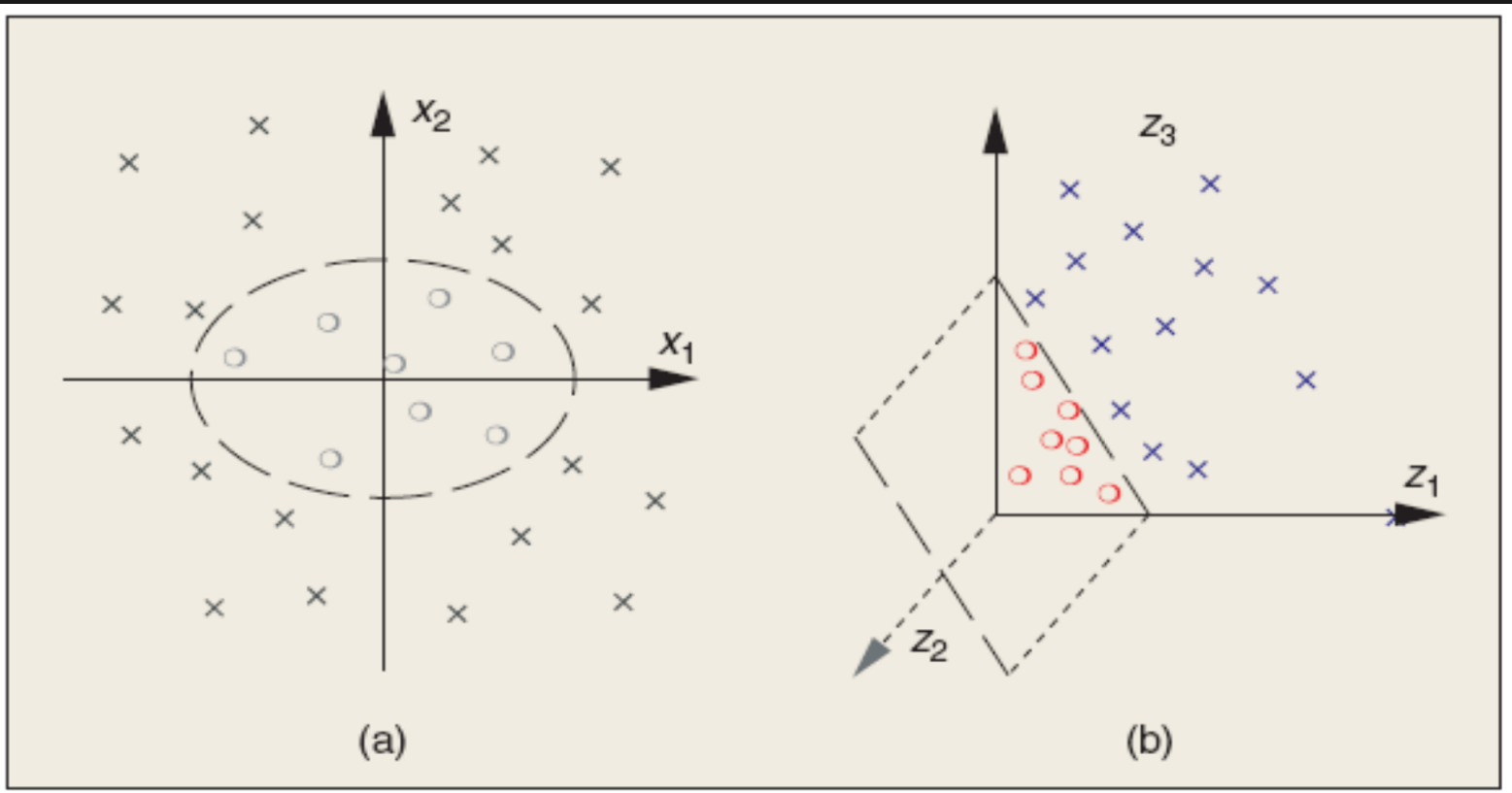
$$\phi : X = (x_1, x_2) \to (x_1^2, x_2^2, \sqrt{2}x_1x_2) \in \mathbb{R}^3$$

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

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

- 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:

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

- This is a pre-computation: we compute the kernel between all pairs once, at the beginning, from which all subsequent computations follow.
- Gram matrices should be positive semi-definite. 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 $1/n$.

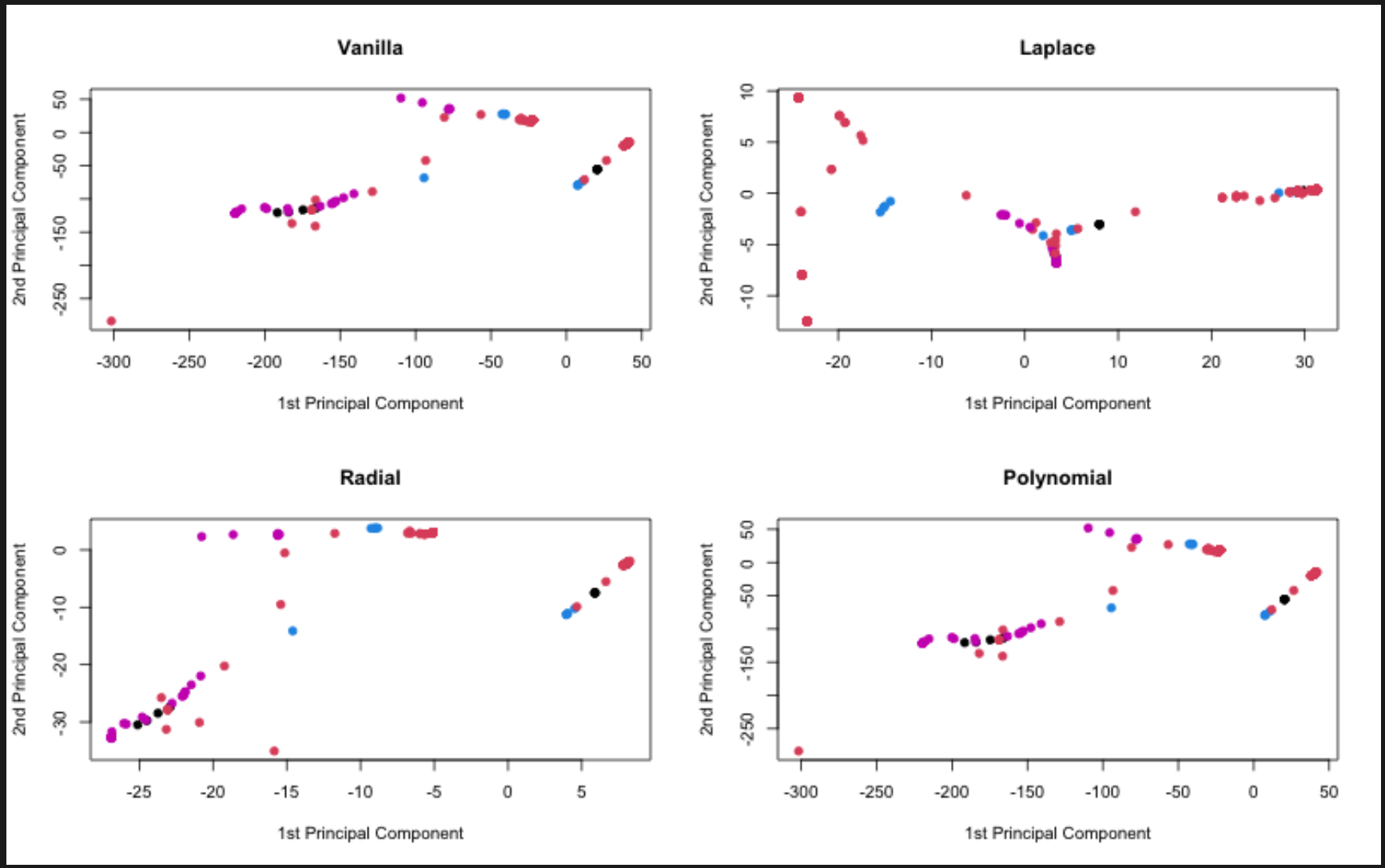
Kernel PCA example

- 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 role could transforms play in classification?
  - How do you know if they are working?
- How do these transforms generalise? What parameters does this introduce?
- What is the benefit of the Kernel Trick? What is the cost?
  - How would you apply it in practice?
- When should you estimate density by KDE vs KNN?
  - What does the density estimate at a point mean?
  - How could it be used in classification?
  - What are its other uses?

Signposting

- Transforms are clearly linked to PCA from Block 03
- Further reading for nonparametric statistics:
  - Nonparametric Statistics by Eduardo García Portugués
  - Basis Expansions: Chapter 5 of The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Friedman, Hastie and Tibshirani).
- Further reading for Kernel Density Estimation:
  - Kernel Smoothing: Chapter 6 of The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Friedman, Hastie and Tibshirani).
  - For kNN Yen-Chi Chen's notes on kNN and the Basis

Signposting (2)

- 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".