Non-linear Models-III

CS771: Introduction to Machine Learning

Purushottam Kar

Outline of discussion

- Kernel PCA
- Accelerated Kernel Methods
- PML with kernels (if there is time)
- Next class: neural networks

Kernels

- Encode notions of similarity between two data points
$$K: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$
- Mercer kernels are basically dot products in a different space.
- For every Mercer kernel K over a universe $\mathcal{X}$, exists $\phi_K: \mathcal{X} \rightarrow \mathcal{H}_K$
$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$
- Examples
  Poly kernel $K(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^p$, $c \geq 0, p \in \mathbb{N}$ for *Mercer-ness*
  Gaussian kernel $K(\mathbf{x}, \mathbf{y}) = \exp(\gamma \cdot \|\mathbf{x} - \mathbf{y}\|_2^2)$, $\gamma > 0$
  Intersection kernel $K(X, Y) = |X \cap Y|$, $X, Y \subseteq \mathcal{U}$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a linear combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \mathbf{x}^i$$

- The prediction on a new data point depends on an inner product between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle$$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a linear combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \mathbf{x}^i$$

- The prediction on a new data point depends on an inner product between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot \langle \mathbf{x}^i, \mathbf{x}^t \rangle$$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a linear combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K\left(\mathbf{x}^i\right)$$

- The prediction on a new data point depends on an inner product between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot \left\langle \mathbf{x}^i, \mathbf{x}^t \right\rangle$$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a linear combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K\left(\mathbf{x}^i\right)$$

- The prediction on a new data point depends on an inner product between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot \left\langle \phi_K\left(\mathbf{x}^i\right), \phi_K\left(\mathbf{x}^t\right) \right\rangle$$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a li combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K(\mathbf{x}^i)$$

Just store the α_i values

- The prediction on a new data point depends on an inner product between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot \langle \phi_K(\mathbf{x}^i), \phi_K(\mathbf{x}^t) \rangle$$

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a li combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K(\mathbf{x}^i)$$

Just store the α_i values

- The prediction on a new data point depends on an between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot \langle \phi_K(\mathbf{x}^i), \phi_K(\mathbf{x}^t) \rangle$$

Replace by the kernel value

The Kernel Trick

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a li combination of the data points

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K(\mathbf{x}^i)$$

Just store the α_i values

- The prediction on a new data point depends on an between the model and the new data point

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot K(\mathbf{x}^i, \mathbf{x}^t)$$

Replace by the kernel value

The Kernel Trick

SVM, RR, k-means, k-NN, r-NN, Perceptron

- Kernels help in learning non-linear functions (classfn, regressn)
- The kernel trick exploits a property of ML algorithms
- The optimal model (MLE/MAP) can be written as a linear combination of the data points

Just store the α_i values

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \cdot \phi_K(\mathbf{x}^i)$$

- The prediction on a new data point depends on an inner product between the model and the new data point

Replace by the kernel value

$$\hat{y}^t = \langle \mathbf{w}, \mathbf{x}^t \rangle = \sum_{i=1}^{n} \alpha_i \cdot K(\mathbf{x}^i, \mathbf{x}^t)$$

Kool kernels and how to konstruct them

- Can construct new Mercer kernels from old ones, using new feature maps, or both!
- Domain knowledge very important for choice of kernel
- Choice of kernel part of the inductive bias
- Can learn the kernel too (search for Multiple kernel learning)
- E.g. learn the γ parameter in Gaussian kernel
- E.g. given *base* kernels $K_1, K_2, \ldots, K_L$, learn a kernel of the form

$$K_{\boldsymbol{\alpha}}(x, y) = \sum_{i=1}^{L} \boldsymbol{\alpha}_i \cdot K_i(x, y)$$

Non linearity is essential

- A map $\phi: \mathcal{X} \rightarrow \mathcal{H}$ where $\mathcal{X} \subseteq \mathbb{R}^d$ is called linear if

$$\phi(\mathbf{x} + \mathbf{y}) = \phi(\mathbf{x}) + \phi(\mathbf{y})$$
$$\phi(c \cdot \mathbf{x}) = c \cdot \phi(\mathbf{x})$$

- Exercise: show that for every linear map $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^D$
  1. Exists a matrix $M \in \mathbb{R}^{D \times d}$ so that $\phi(\mathbf{x}) = M\mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$
  2. For all $\mathbf{W} \in \mathbb{R}^D$, there exists $\boldsymbol{w} \in \mathbb{R}^d$ such that for all $\mathbf{x} \in \mathbb{R}^d$

$$\langle \mathbf{W}, \phi(\mathbf{x}) \rangle = \langle \mathbf{w}, \mathbf{x} \rangle$$

- If the map is linear and the algorithm is learning a linear function in $\mathbb{R}^D$ then the end result is a linear function in $\mathbf{x}$
- Kernels whose maps are linear are quite useless
- On the other hand, if ϕ is non-linear, even a linear function of $\phi(\mathbf{x})$ can be non-linear in $\mathbf{x}$

Parametric and non-parametric models

- Models with fixed "size" are called parametric. Let $\mathbf{x} \in \mathbb{R}^d$
- Linear model $\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle + b$, where $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$
  Model is fully specified by $d + 1$ parameters
  No matter how many training data points we use
- Quadratic model $\mathbf{x} \mapsto \mathbf{x}^\top A\mathbf{x} + \langle \mathbf{b}, \mathbf{x} \rangle + c$, where $A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d, c \in \mathbb{R}$
  Model is fully specified by $d^2 + d + 1$ parameters
  No matter how many training data points we use
- Kernel models are examples of non-parametric

$$\mathbf{w} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i y^i \cdot \phi\left(\mathbf{x}^i\right)$$

  requires n parameters to be stored, more data, more parameters

Parametric and non-parametric models

- Models with fixed "size" are called parametric. Le
  $\mathbf{x} \mapsto$ $b \in \mathbb{R}$
  spe
  w many training data points we use
- Quadratic model $\mathbf{x} \mapsto \mathbf{x}^T A\mathbf{x} + \langle \mathbf{b}, \mathbf{x} \rangle + c$, where $A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d, c \in \mathbb{R}$
  Model is fully specified by $d^2 + d + 1$ parameters
  No matter how many training data points we use
- Kernel models are examples of non-parametric
  $\sum_{i=1}$
  requires n parameters to be stored, more data, more parameters

Quadratic model learnt directly is parametric

However, if d^2 is too big, use K_{quad} to learn it and then it becomes non-parametric!

Same model can be learnt in param/non-param way

Storing the α_i values

A non-parametric model actually has an unbounded number of parameters!

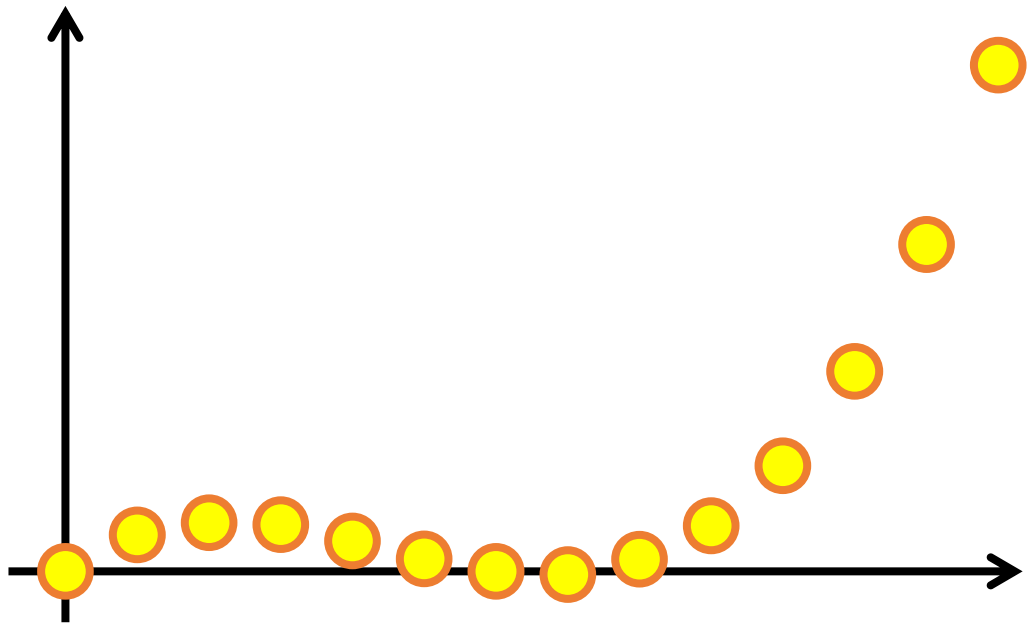
Weird naming ☺

Kernel PCA

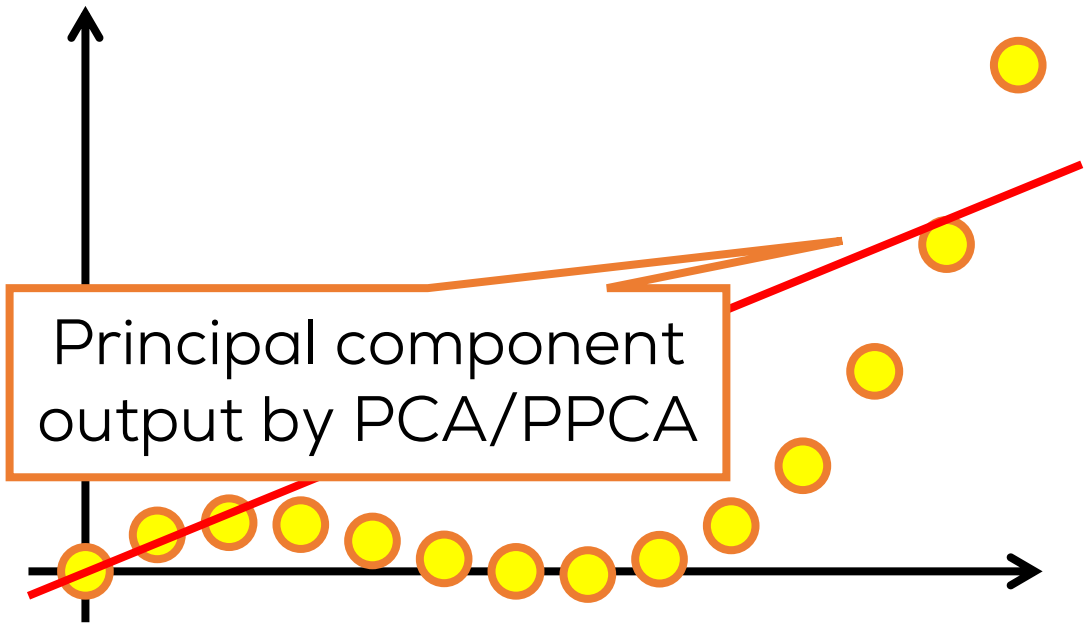
Non-linear components

Non-linear components

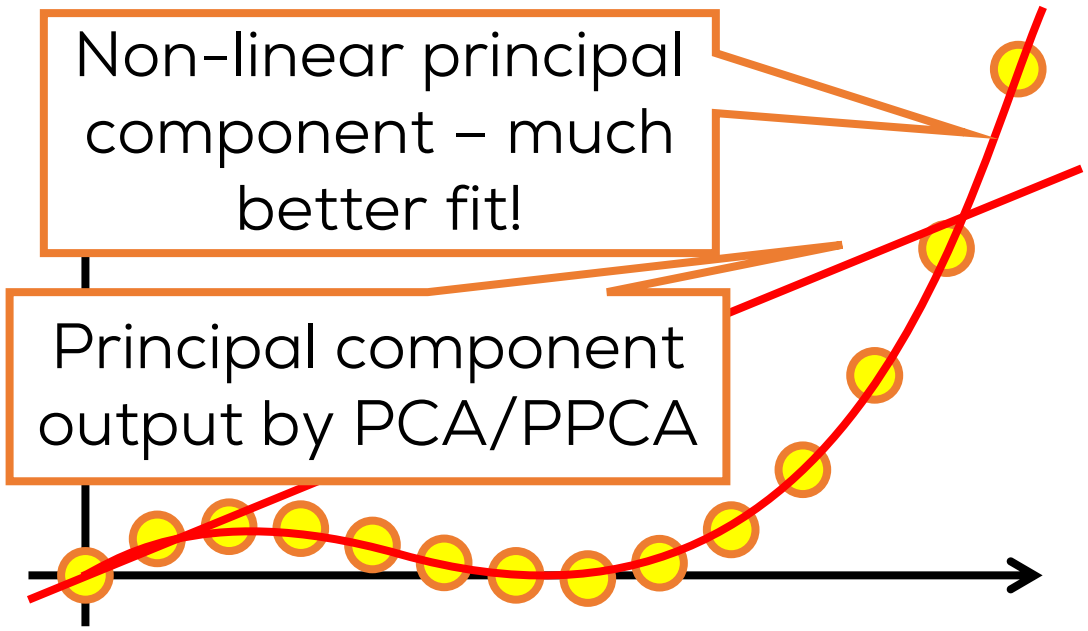
Non-linear components



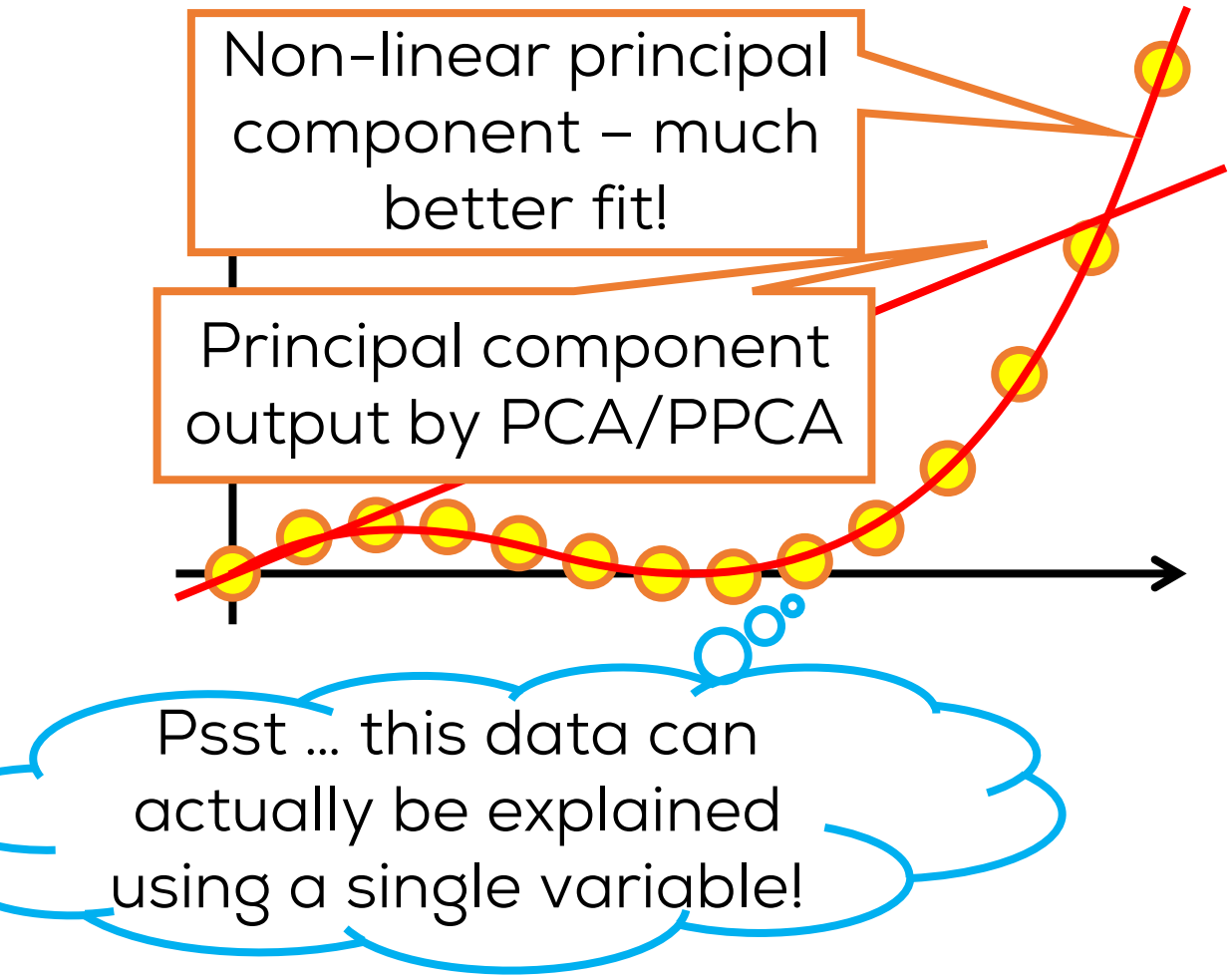
Non-linear components



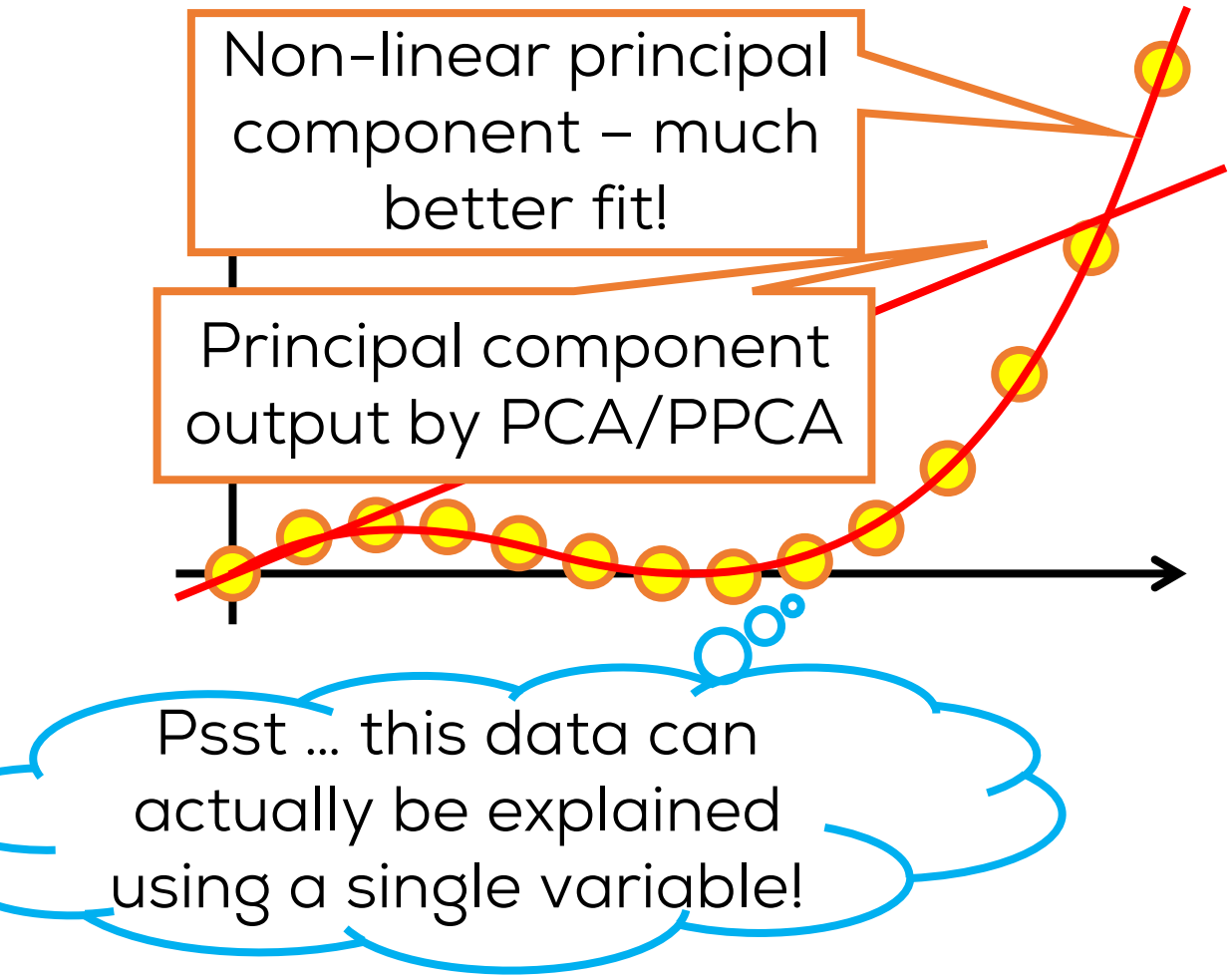
Non-linear components



Non-linear components

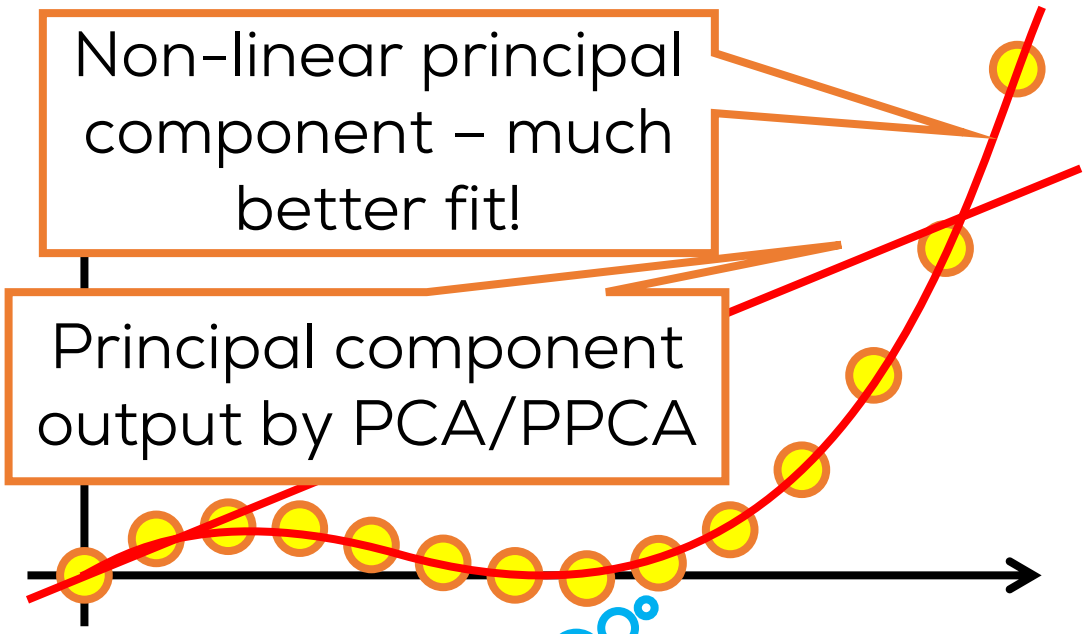


Non-linear components



$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Non-linear components

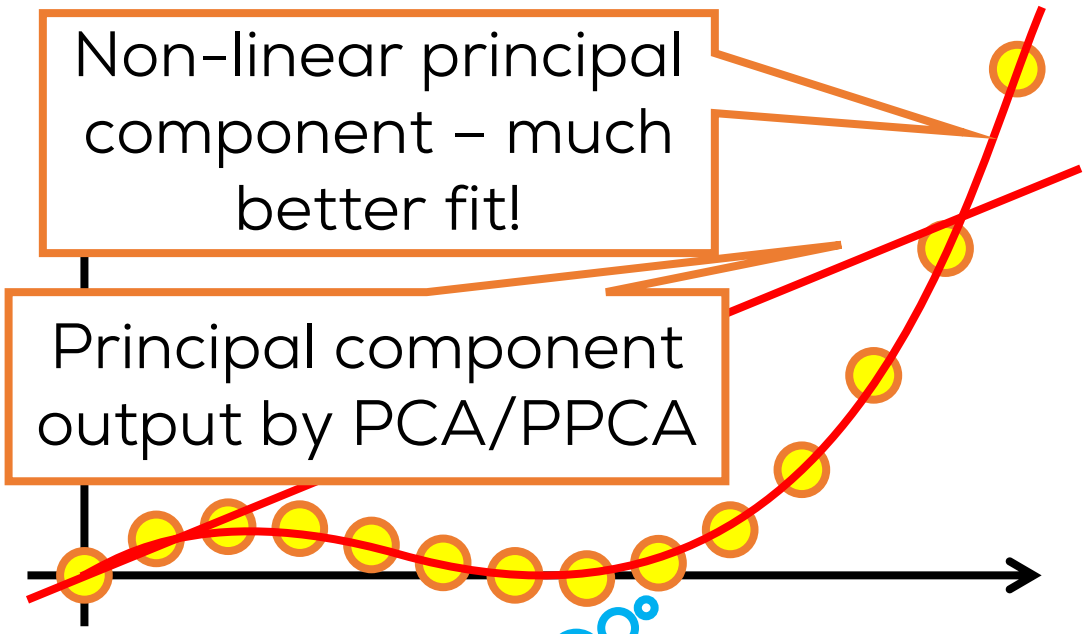


Psst … this data can actually be explained using a single variable!

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Even though data looks 2D, it is inherently 1D

Non-linear components



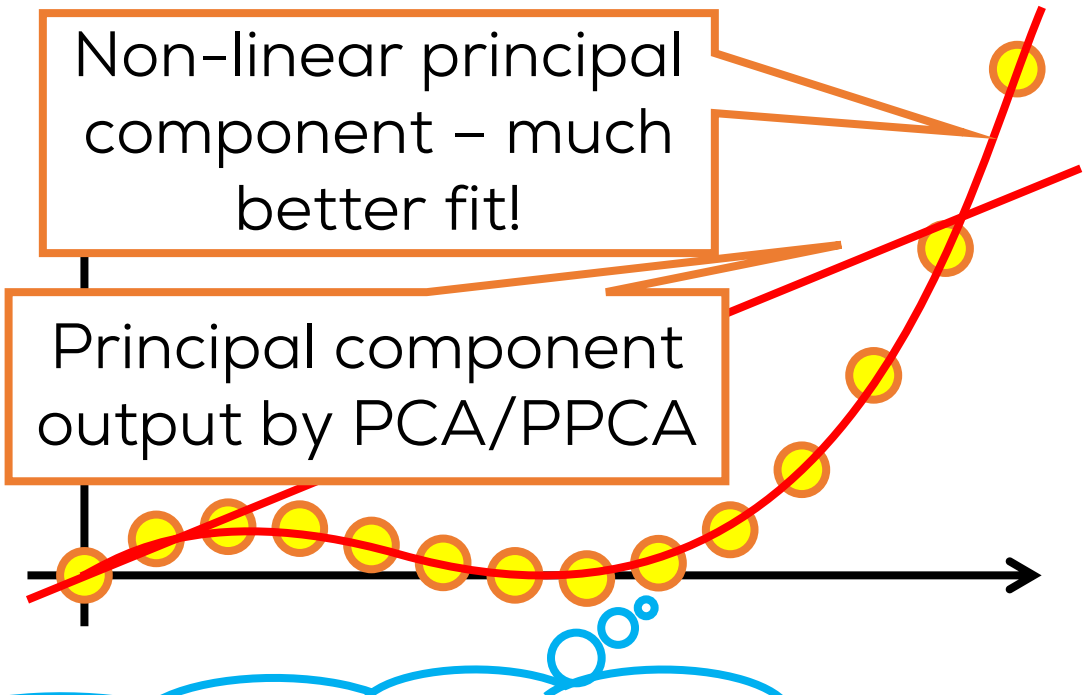
Psst … this data can actually be explained using a single variable!

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

Non-linear components



Psst … this data can actually be explained using a single variable!

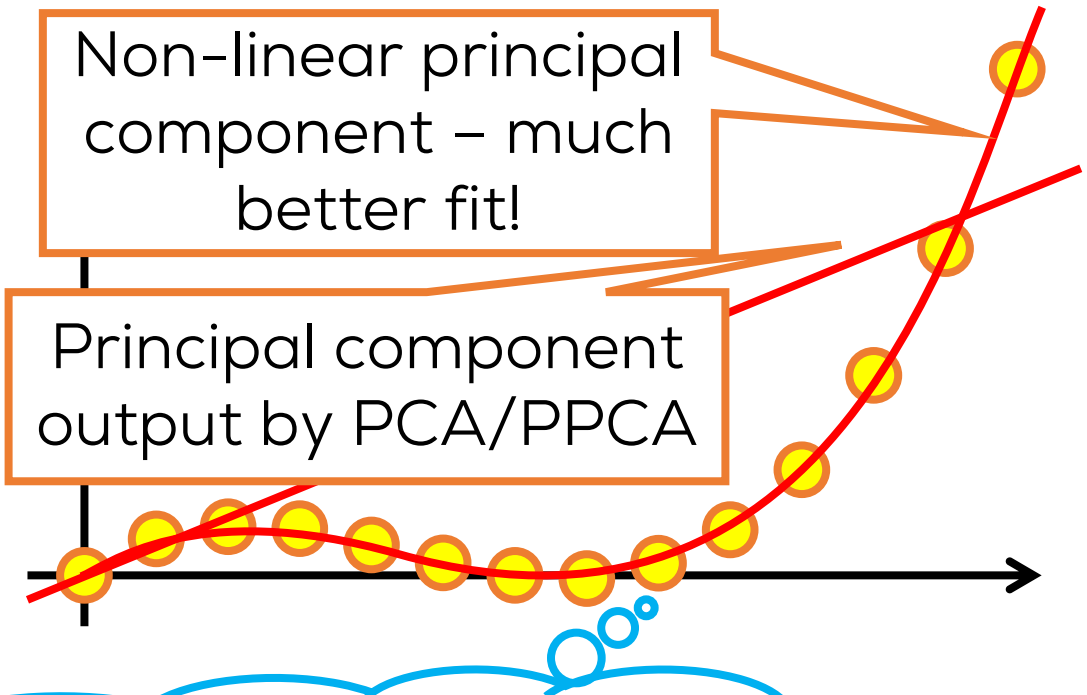
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

Non-linear components



Psst … this data can actually be explained using a single variable!

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

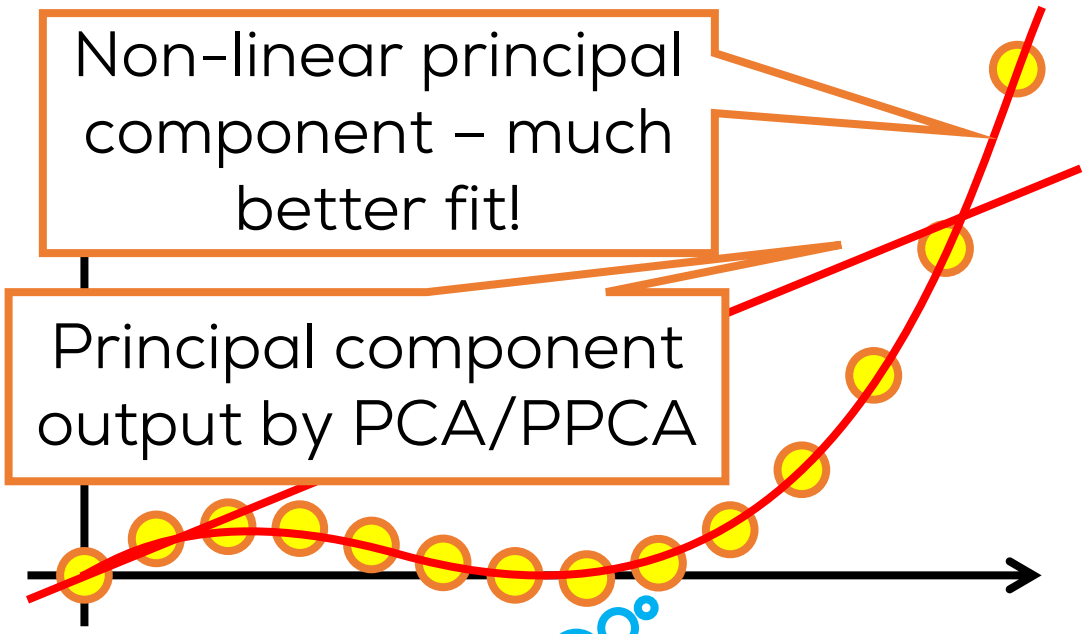
Use K_{poly} with $p = 3, c = 0$

Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

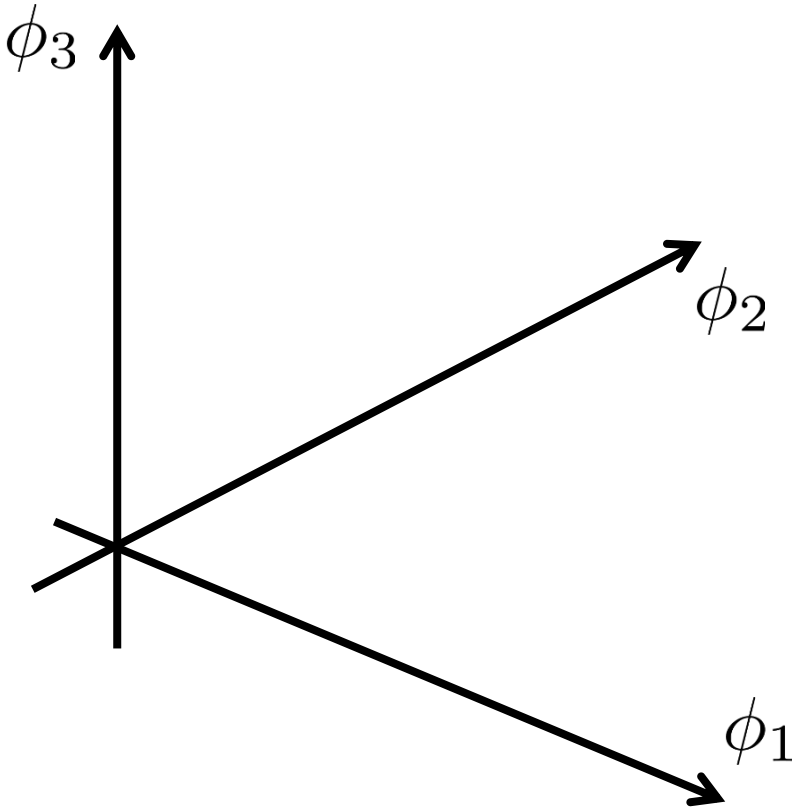
Non-linear components



Non-linear principal component – much better fit!

Principal component output by PCA/PPCA

Psst … this data can actually be explained using a single variable!



$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

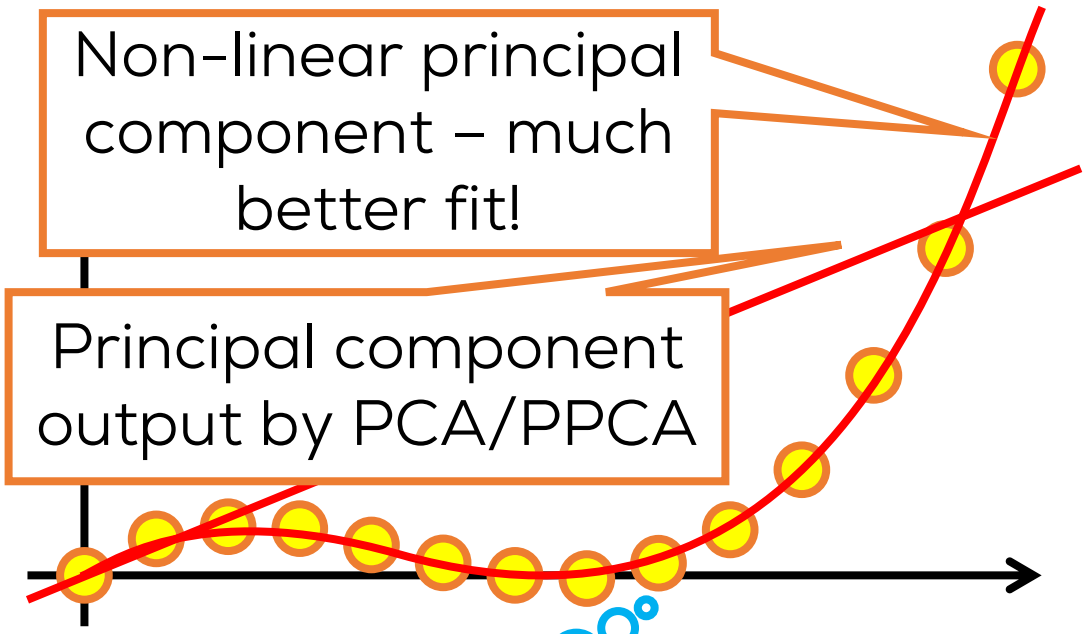
Use K_{poly} with $p = 3, c = 0$

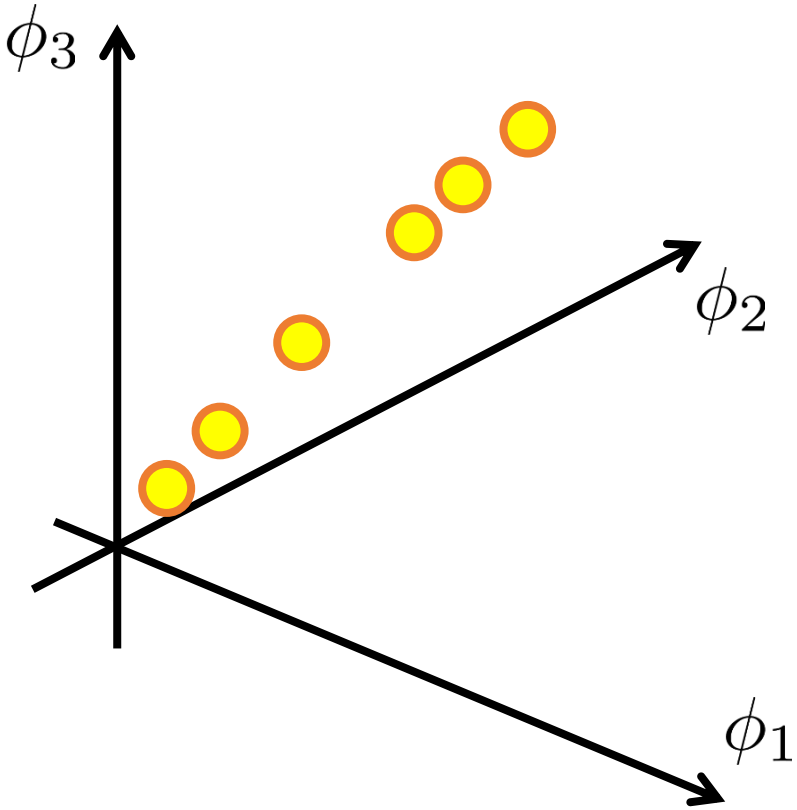
Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Non-linear components





Psst … this data can actually be explained using a single variable!

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

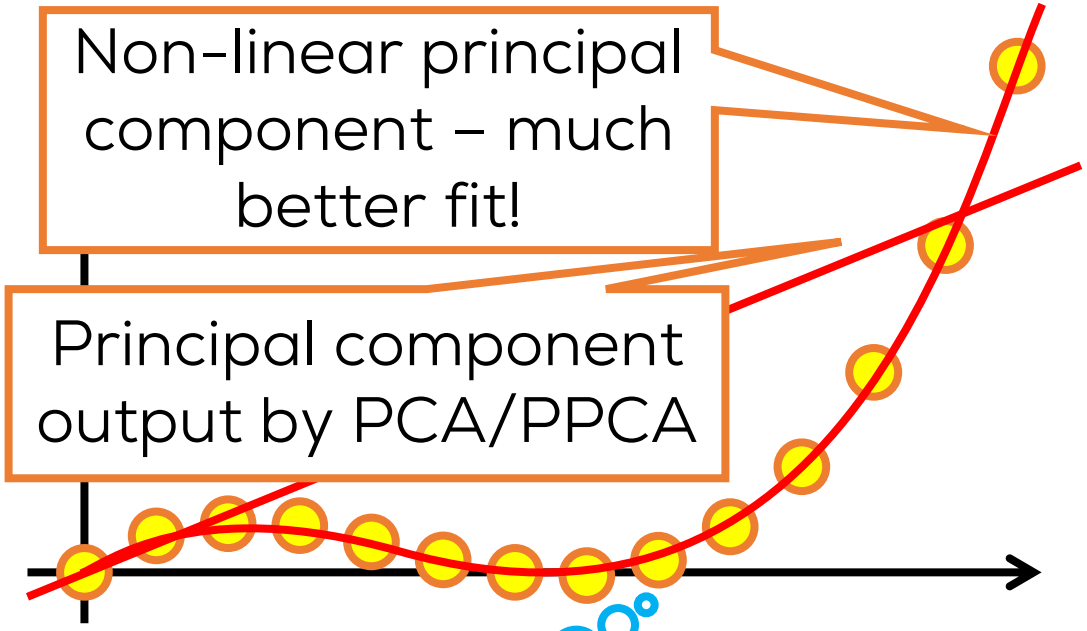
Use K_{poly} with $p = 3, c = 0$

Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

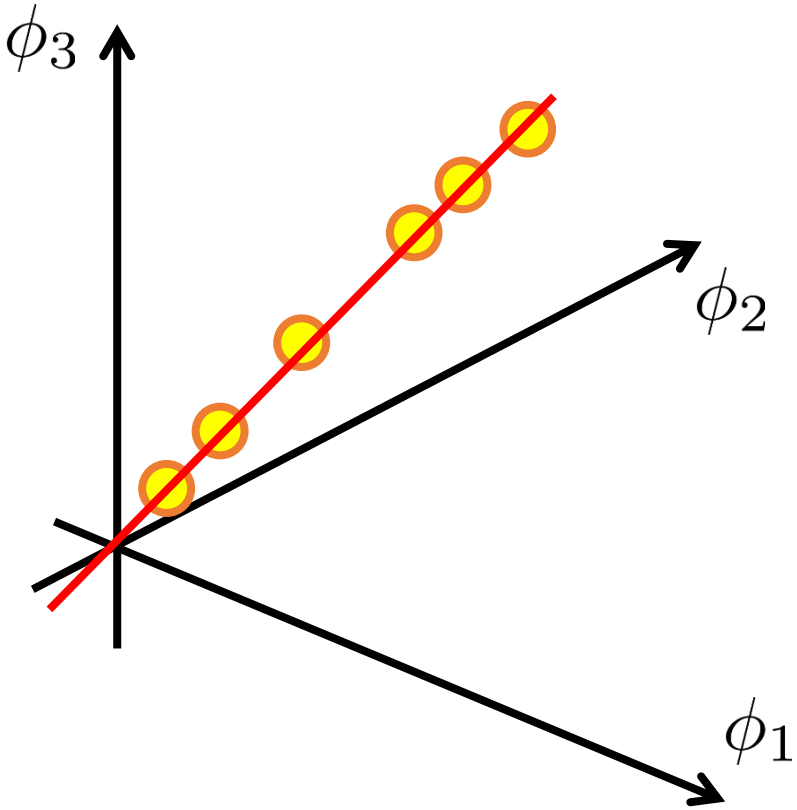
$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Non-linear components



Non-linear principal component – much better fit!

Principal component output by PCA/PPCA



Psst … this data can actually be explained using a single variable!

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

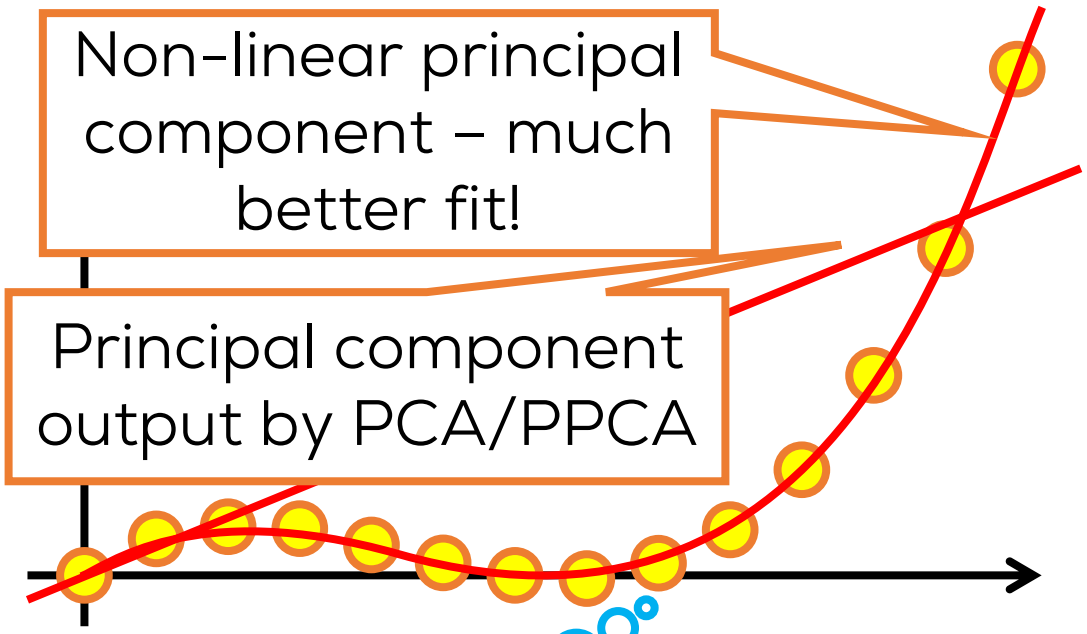
Use K_{poly} with $p = 3, c = 0$

Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

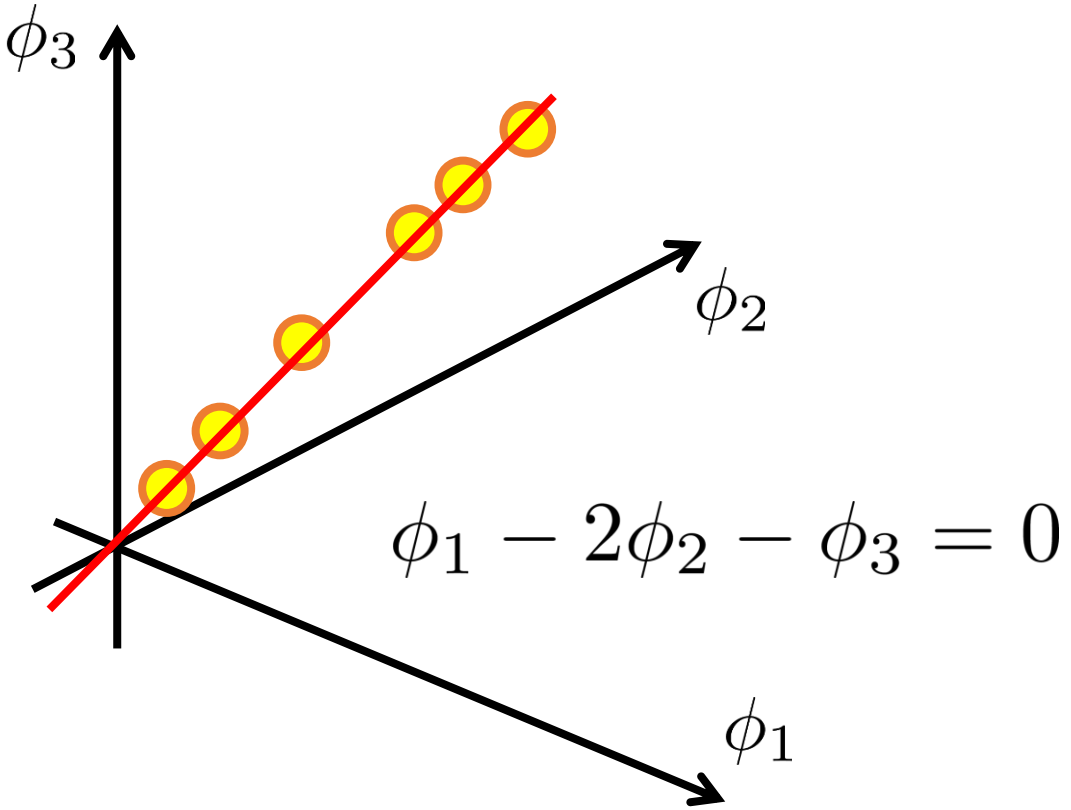
Non-linear components



Non-linear principal component – much better fit!

Principal component output by PCA/PPCA

Psst … this data can actually be explained using a single variable!



$$\phi_1 - 2\phi_2 - \phi_3 = 0$$

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [y, x^2, x^3] \in \mathbb{R}^3$$

Use K_{poly} with $p = 3, c = 0$

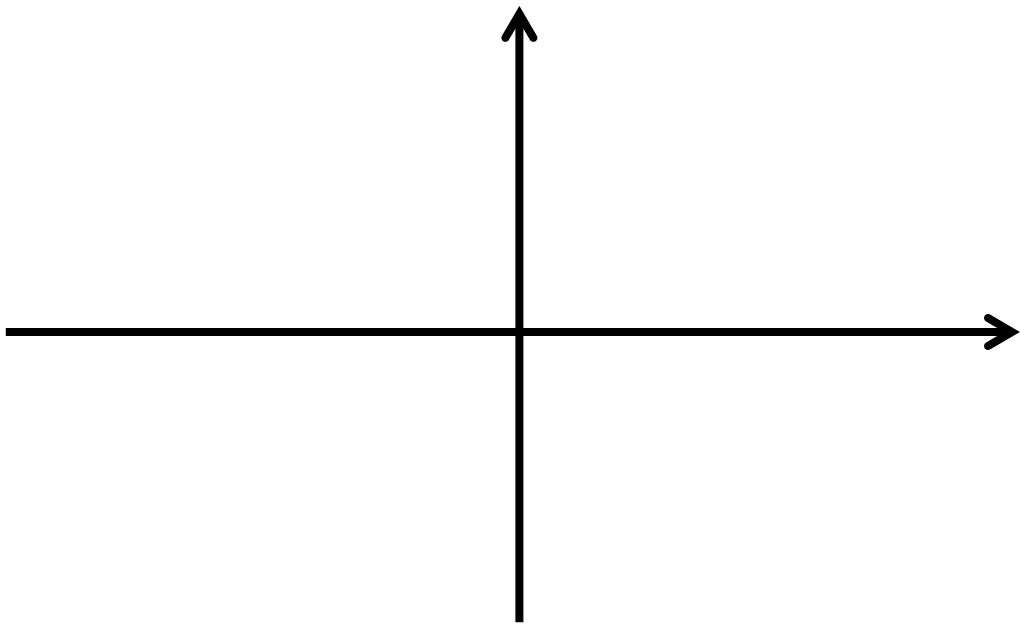
Even though data looks 2D, it is inherently 1D

Can kernels recover this 1D structure?

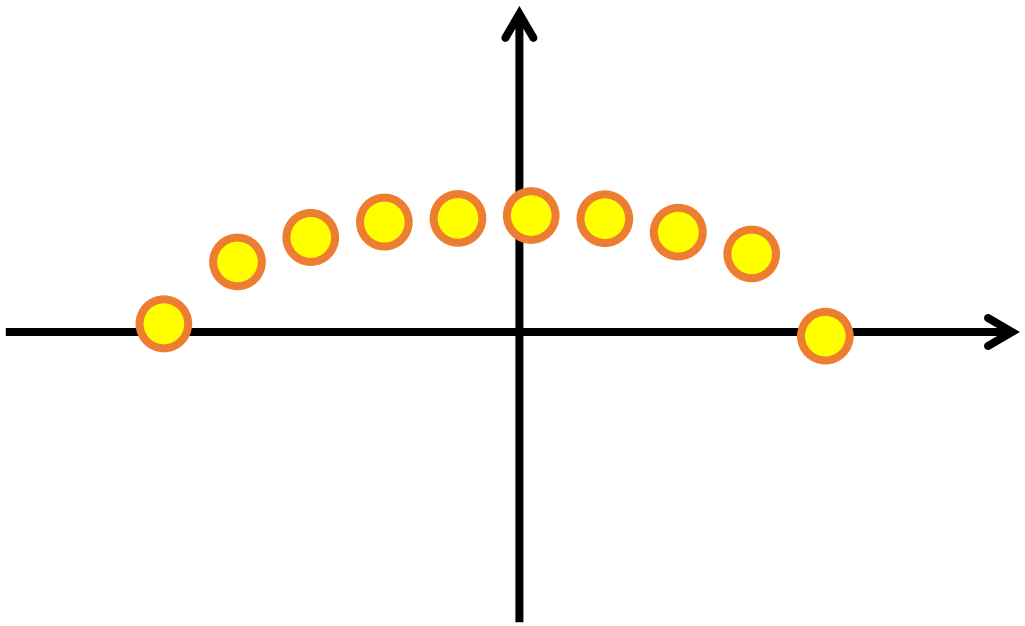
$$x = t$$
$$y = 2 \cdot t^2 + t^3$$

Non-linear components

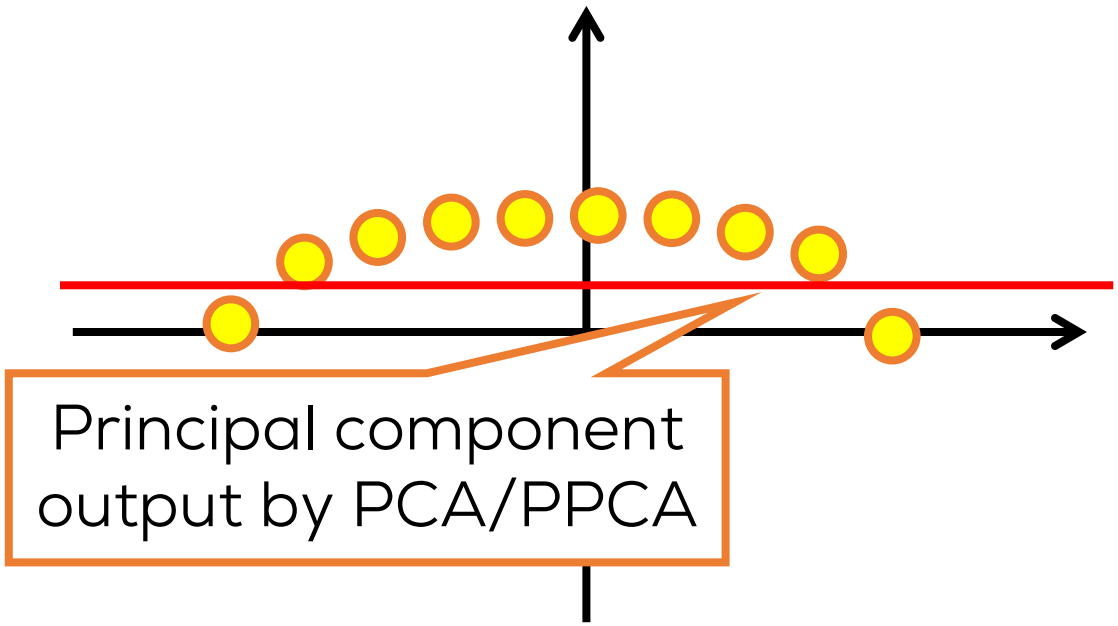
Non-linear components



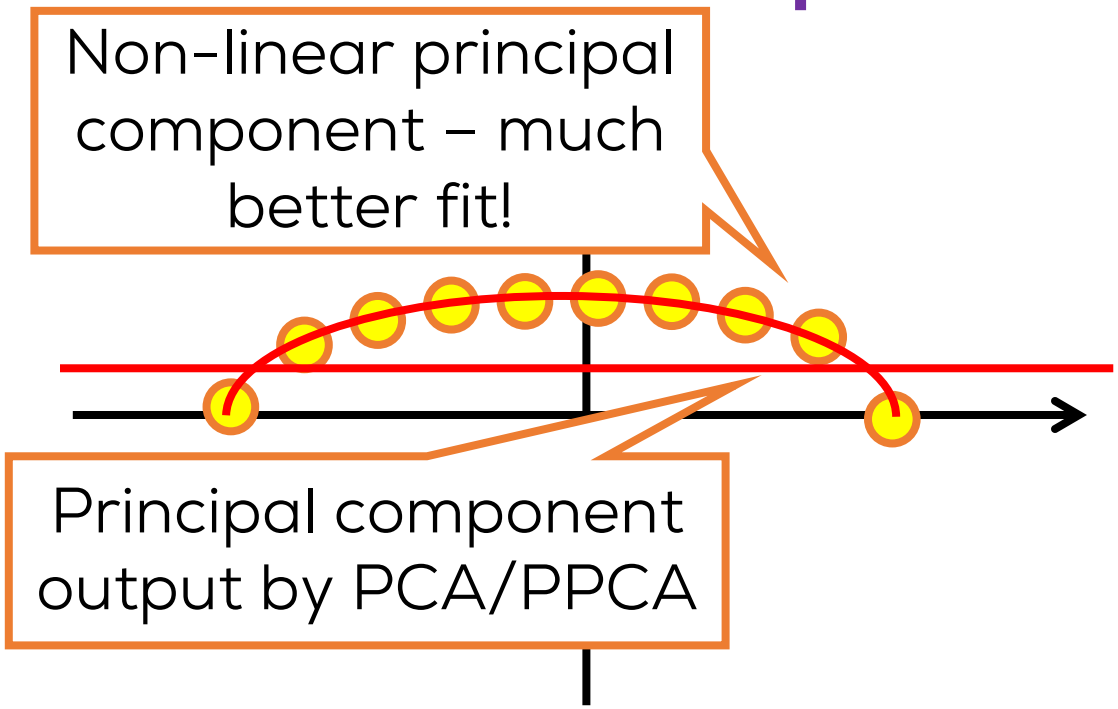
Non-linear components



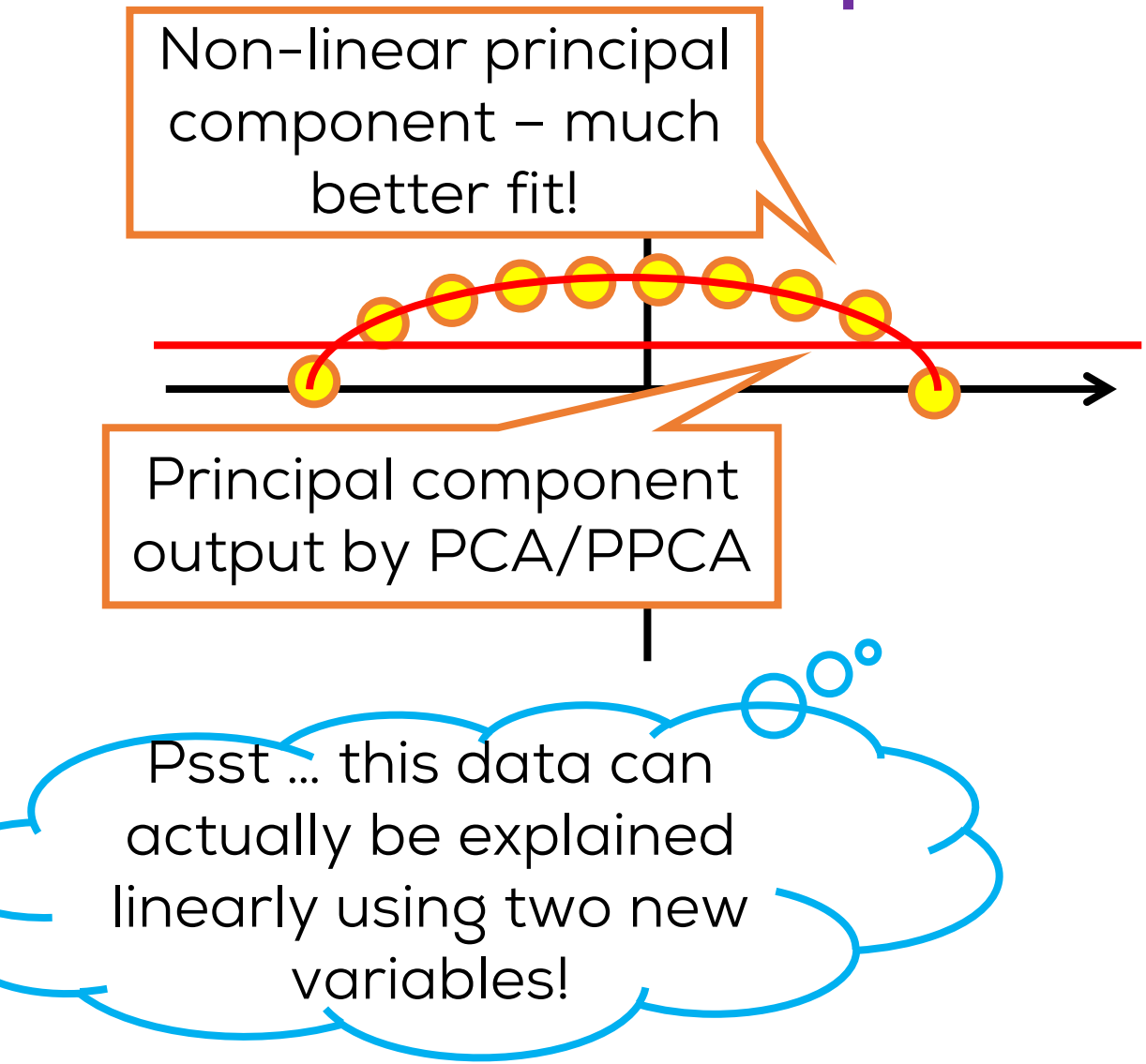
Non-linear components



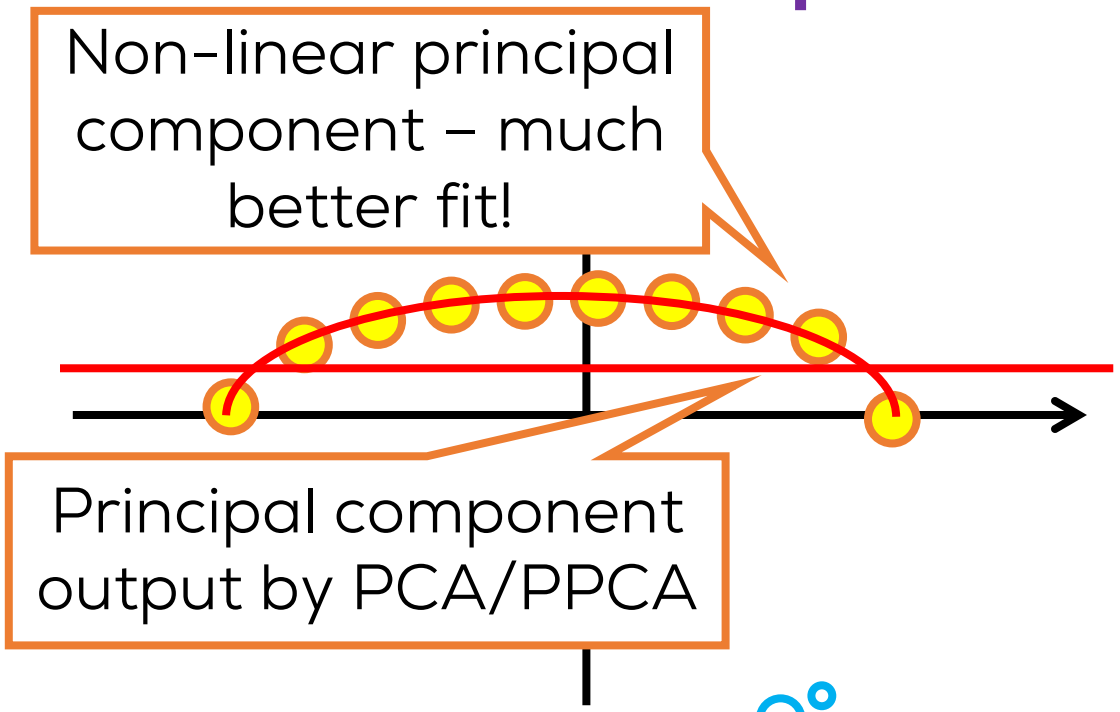
Non-linear components



Non-linear components



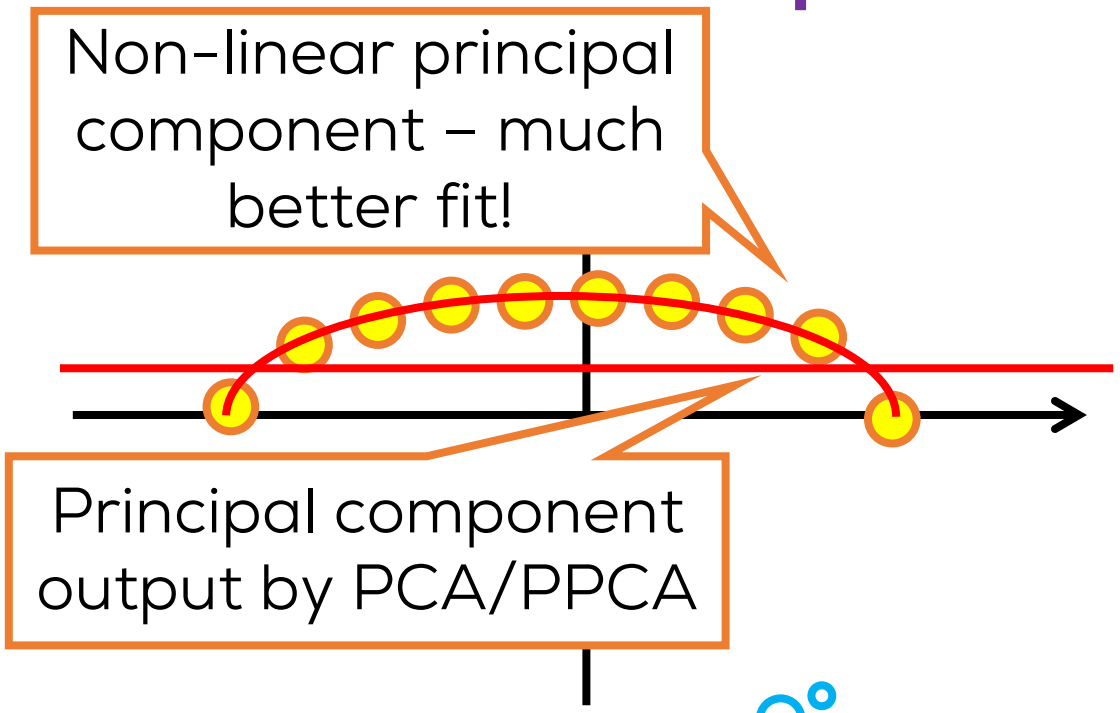
Non-linear components



Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

Non-linear components



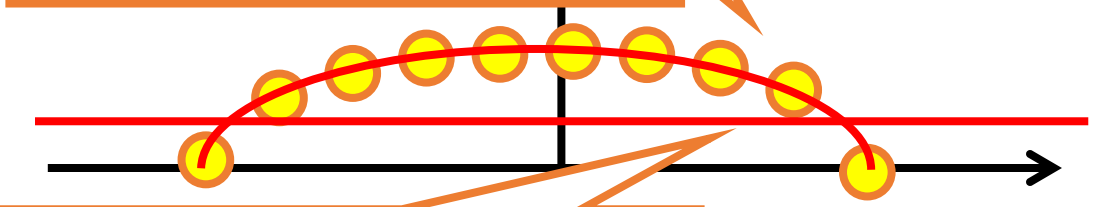
Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA

Psst … this data can actually be explained linearly using two new variables!

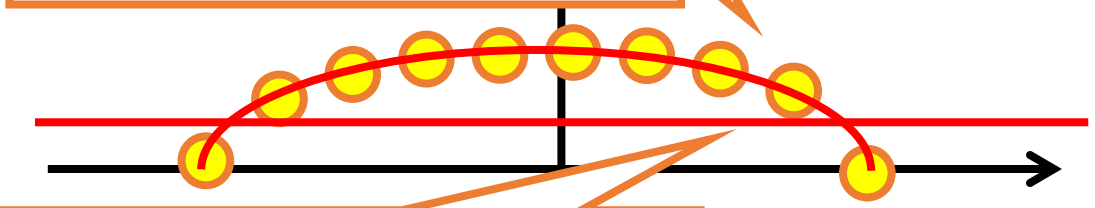
$x^2 + 4y^2 = 1$

$x^2 = t$
$y^2 = r$

Data is inherently 1D

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA

Psst … this data can actually be explained linearly using two new variables!

$x^2 + 4y^2 = 1$

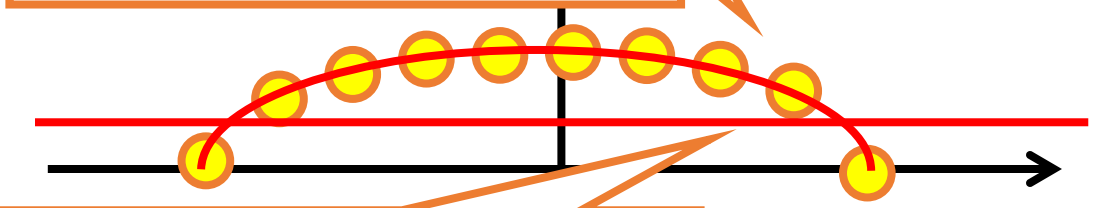
$x^2 = t$
$y^2 = r$

Data is inherently 1D

Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA

Psst … this data can actually be explained linearly using two new variables!

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

$x^2 + 4y^2 = 1$

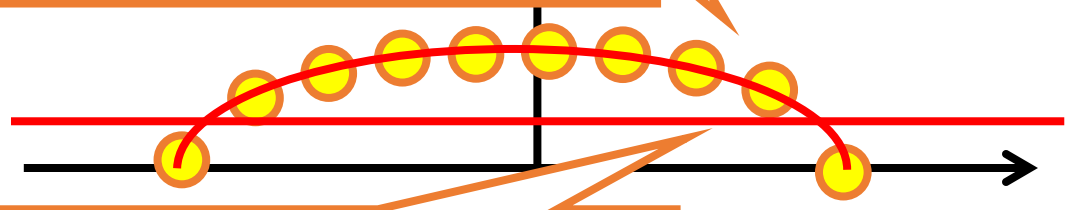
$x^2 = t$
$y^2 = r$

Data is inherently 1D

Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA

Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

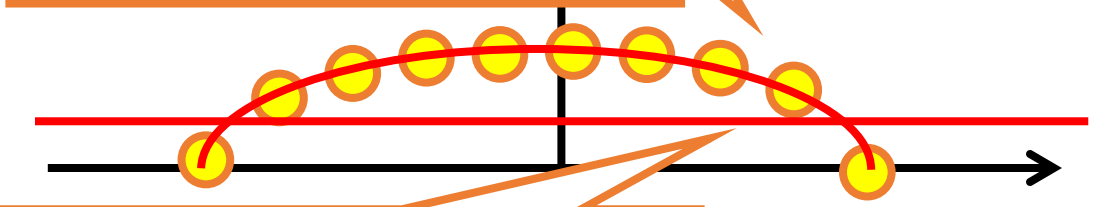
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

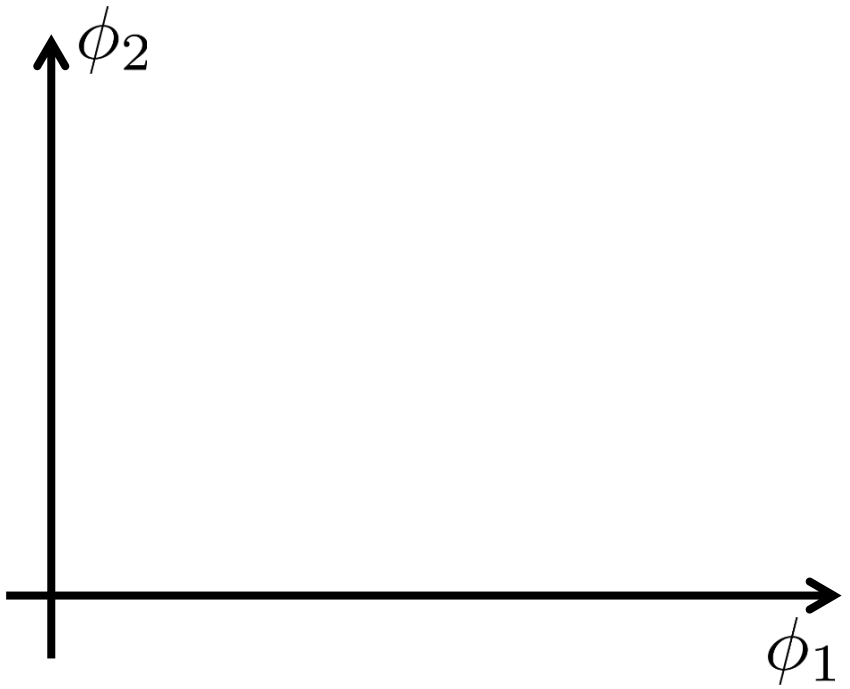
Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA



Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

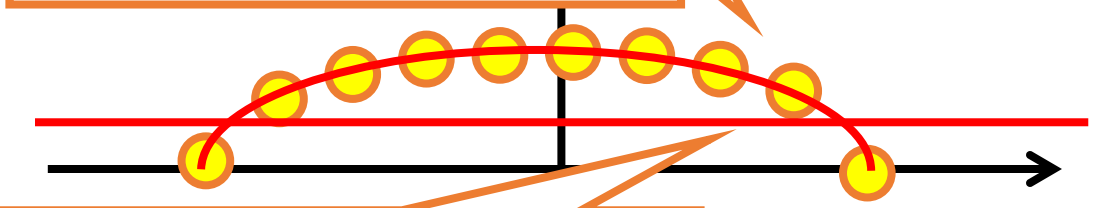
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

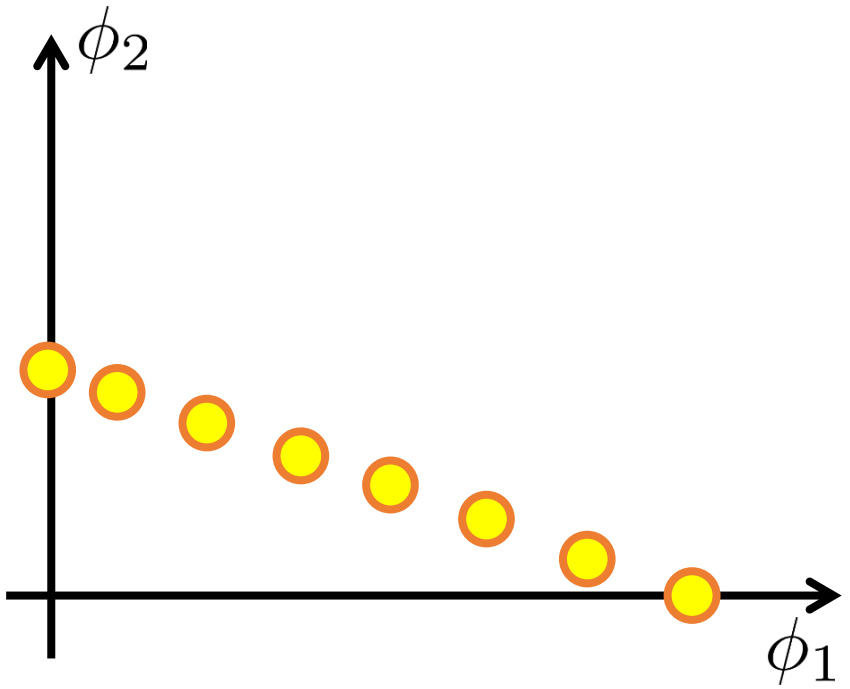
Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA



Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

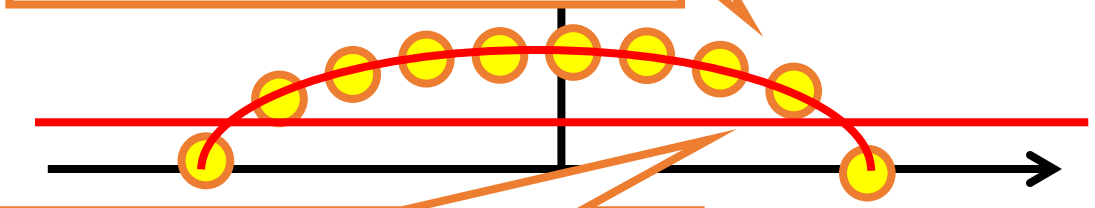
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

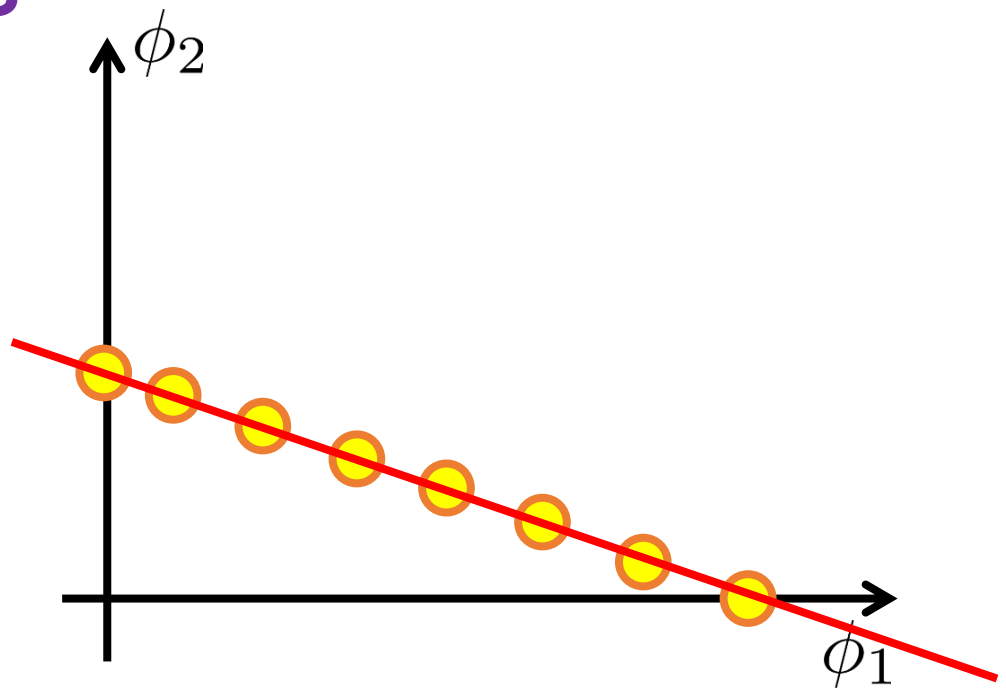
Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA



Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

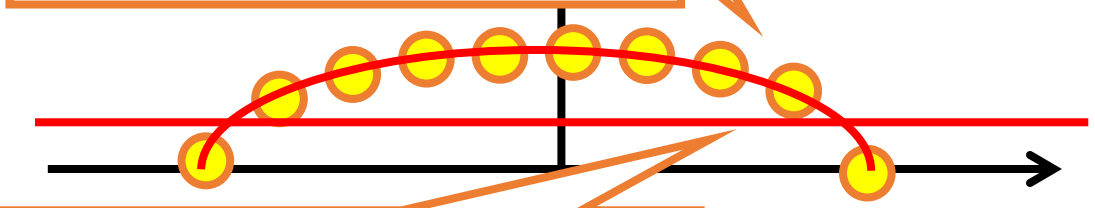
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

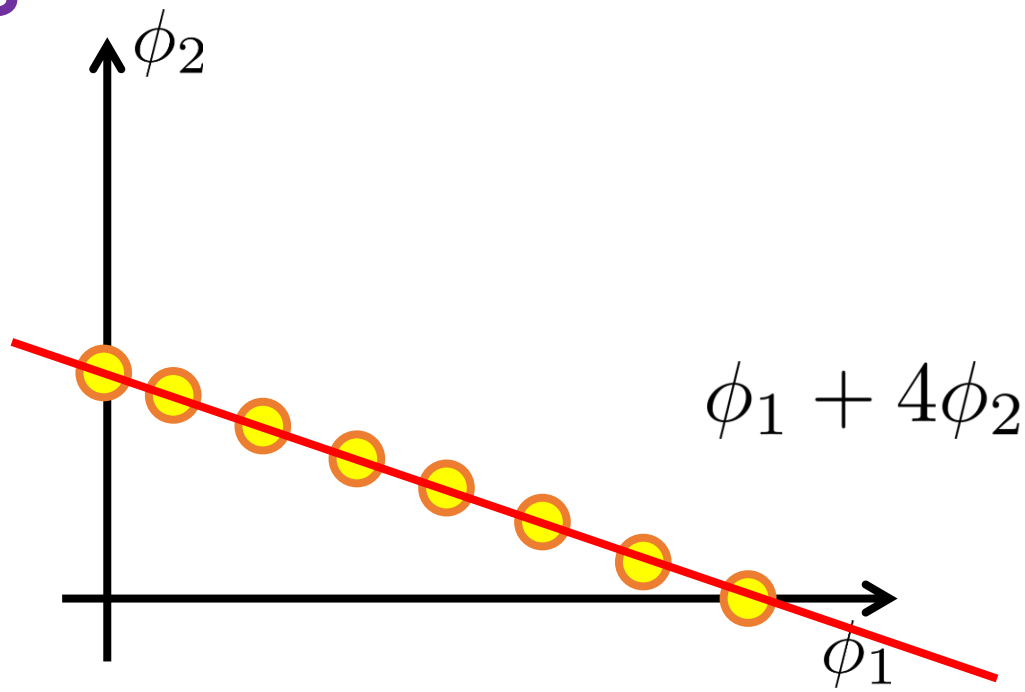
Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA



$$\phi_1 + 4\phi_2 = 1$$

Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

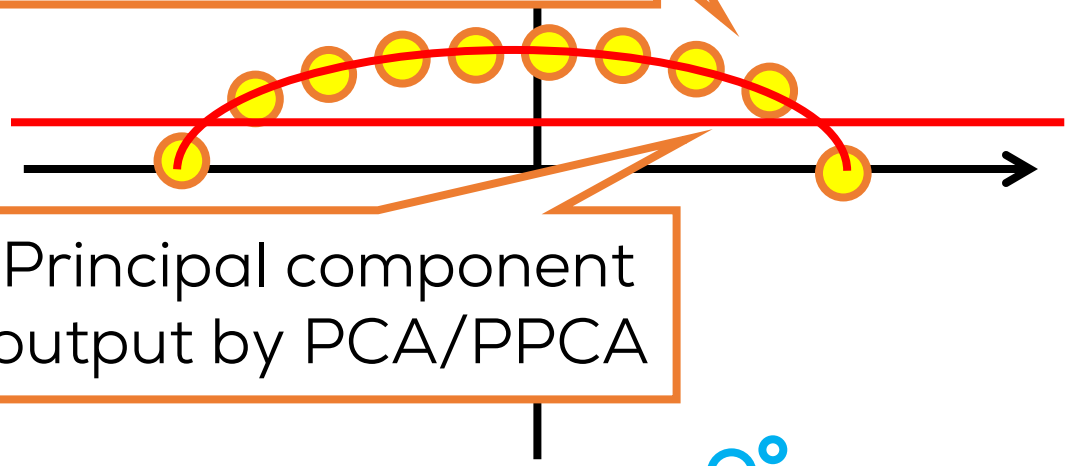
$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

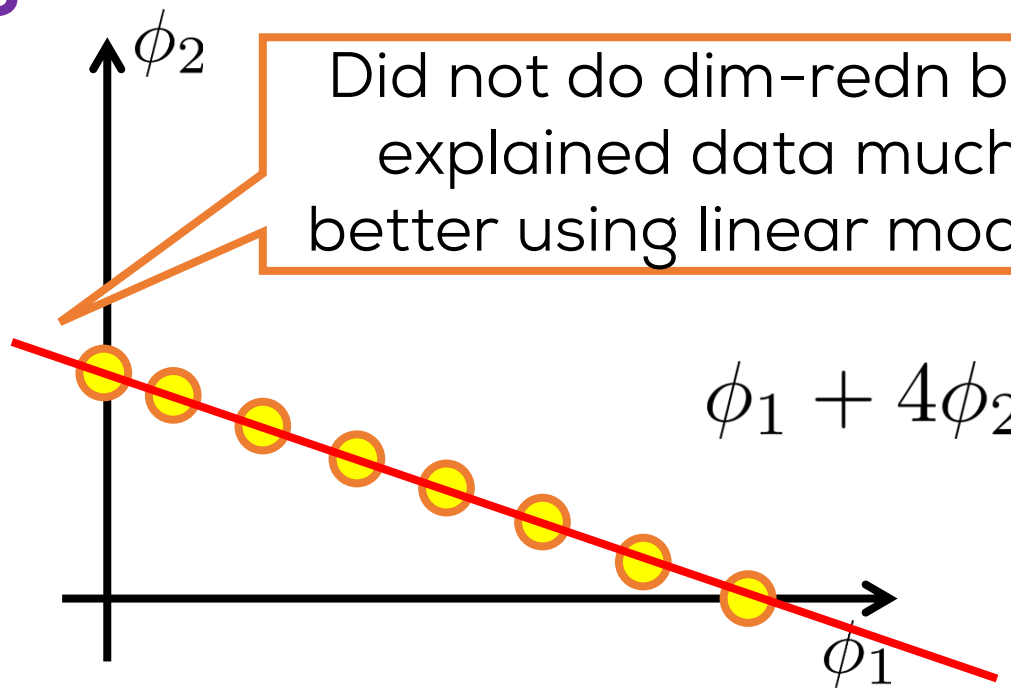
Can kernels help represent this linearly?

Non-linear components

Non-linear principal component – much better fit!



Principal component output by PCA/PPCA



Did not do dim-redn but explained data much better using linear model!

$$\phi_1 + 4\phi_2 = 1$$

Psst … this data can actually be explained linearly using two new variables!

$$x^2 + 4y^2 = 1$$

$$x^2 = t$$
$$y^2 = r$$

Data is inherently 1D

$$\mathbb{R}^2 \ni (x, y) \mapsto \phi(x, y) = [x^2, y^2] \in \mathbb{R}^2$$

Use K_{quad}

Can kernels help represent this linearly?

Umm … how will this ever work RKHS $\mathcal{H}$?

- Recall PCA required getting eigenvectors of emp. cov. Matrix

$$S = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^i \left(\mathbf{x}^i\right)^\top$$

- Are we going to work with an infinite by infinite matrix now?!

$$S = \frac{1}{n}\sum_{i=1}^{n} \phi\left(x^i\right)\phi\left(x^i\right)^\top$$

- Well … for simplicity, for now assume $\mathcal{H} = \mathbb{R}^D$
- A better way is to represent S as an *operator*
- Instead of thinking of $M^i = \phi\left(x^i\right)\phi\left(x^i\right)^\top$ as a inf. square matrix, think of it as an operator than maps $\phi(x) \mapsto K\left(x^i, x\right) \cdot \phi\left(x^i\right)$

Umm … how will this ever work RKHS $\mathcal{H}$?

- Recall PCA required getting eigenvectors of emp. cov. Matrix

$$S = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^i \left(\mathbf{x}^i\right)^\top$$

- Are we going to work with an infinite by infinite matrix now?!

$$S = \frac{1}{n}\sum_{i=1}^{n} \phi\left(x^i\right)\phi\left(x^i\right)^\top$$

- Well … for simplicity, for now assume $\mathcal{H} = \mathbb{R}^n$
- A better way is to represent S as an *operator*

$$M\phi(x) = \phi\left(x^i\right)\phi\left(x^i\right)^\top \phi(x)$$

- Instead of thinking of $M^i = \phi\left(x^i\right)\phi\left(x^i\right)^\top$ as a inf. square matrix, think of it as an operator than maps $\phi(x) \mapsto K\left(x^i, x\right) \cdot \phi\left(x^i\right)$

Executing PCA in an RKHS $\mathcal{H}$

- For sake of argument let $\Phi = [\phi(x^1), \ldots, \phi(x^n)]$ so that $S = \frac{1}{n}\Phi\Phi^\top$
- Goal is to compute eigenvectors of S i.e. $\mathbf{v} \in \mathcal{H}$ such that

$$S\mathbf{v} = \lambda \cdot \mathbf{v}$$

$$\frac{1}{n}\sum_{i=1}^{n} \phi(x^i)\phi(x^i)^\top \mathbf{v} = \lambda \cdot \mathbf{v}$$

$$\mathbf{v} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i \cdot \phi(x^i) = \Phi\boldsymbol{\alpha}, where\ \boldsymbol{\alpha}_i = \frac{1}{\lambda n}\phi(x^i)^\top \mathbf{v}$$

- Nice! Got a "dual" representation that can be stored efficiently
- Putting back gives us

$$\Phi\Phi^\top\Phi\boldsymbol{\alpha} = \lambda n \cdot \Phi\boldsymbol{\alpha}$$

since $S = \frac{1}{n}\Phi\Phi^\top$ and $\mathbf{v} = \Phi\boldsymbol{\alpha}$

Executing PCA in an RKHS $\mathcal{H}$

- Hmm … so instead of finding $\mathbf{v}$, I will find $\boldsymbol{\alpha}$ and for that I have
$$\Phi\Phi^{\mathsf{T}}\Phi\boldsymbol{\alpha} = \lambda n \cdot \Phi\boldsymbol{\alpha}$$
- Still not good enough … still an infinite dimensional system
- Wait … isn't $\Phi^{\mathsf{T}}\Phi = G$, the Gram matrix $G_{ij} = K(x^i, x^j)$? Yes!
$$\Phi^{\mathsf{T}}\Phi\Phi^{\mathsf{T}}\Phi\boldsymbol{\alpha} = \lambda n \cdot \Phi^{\mathsf{T}}\Phi\boldsymbol{\alpha}$$
$$G^2\boldsymbol{\alpha} = \lambda n \cdot G\boldsymbol{\alpha}$$
- All solutions of the above can be found by instead solving (Verify!)
$$G\boldsymbol{\alpha} = \tilde{\lambda} \cdot \boldsymbol{\alpha}$$
- Great, so all I need to do is apply the Power Method and Peeling Method to find eigenvectors of G and I will get eigenvectors of S
- Can find the top k eigenvectors in $\mathcal{O}(n^2k)$ time (Verify!)

Executing PCA in an RKHS $\mathcal{H}$

- Hmm … so instead of finding $\mathbf{v}$, I will find $\boldsymbol{\alpha}$ and for that I have

$$\Phi\Phi^{\top}\Phi\boldsymbol{\alpha} = \lambda n \cdot \Phi\boldsymbol{\alpha}$$

- Still not good enough … still an infinite dimensional system
- Wait … isn't $\Phi^{\top}\Phi = G$, the Gram matrix $G_{ij} = K(x^i, x^j)$? Yes!

$$\Phi^{\top}\Phi\Phi^{\top}\Phi\boldsymbol{\alpha} = \lambda n \cdot \Phi^{\top}\Phi\boldsymbol{\alpha}$$

$$G^2\boldsymbol{\alpha} = \lambda n \cdot G\boldsymbol{\alpha}$$

with $\tilde{\lambda} = \lambda n$

Even when G is non-invertible

- All solutions of the above can be found by instead solving (Verify!)

$$G\boldsymbol{\alpha} = \tilde{\lambda} \cdot \boldsymbol{\alpha}$$

- Great, so all I need to do is apply the Power Method and Peeling Method to find eigenvectors of G and I will get eigenvectors of S
- Can find the top k eigenvectors in $\mathcal{O}(n^2k)$ time (Verify!)

Utilizing components learnt in the RKHS $\mathcal{H}$

- Okay, so I got hold of an eigenpair $(\boldsymbol{\alpha}, \tilde{\lambda})$ of G, now what?
- Get an eigenvector for S by applying $\mathbf{v} = \Phi\boldsymbol{\alpha}$
- Wait … $\mathbf{v}$ should be unit norm right?

$$\|\mathbf{v}\|_{\mathcal{H}}^2 = \mathbf{v}^\top\mathbf{v} = \boldsymbol{\alpha}^\top\Phi^\top\Phi\boldsymbol{\alpha} = \boldsymbol{\alpha}^\top G\boldsymbol{\alpha} = \tilde{\lambda}\cdot\|\boldsymbol{\alpha}\|_2^2 = \tilde{\lambda}$$

- Hmm … so there is a normalization issue here. I need to do

$$\mathbf{v} = \frac{1}{\sqrt{\tilde{\lambda}}}\cdot\Phi\boldsymbol{\alpha}$$

- Why did we have to do this?
- Because when we wrote $\mathbf{v} = \Phi\boldsymbol{\alpha}$, we did not ask $\boldsymbol{\alpha}$ to be normalized but the eqn $G\boldsymbol{\alpha} = \tilde{\lambda}\cdot\boldsymbol{\alpha}$ forces it to be normalized

Utilizing components learnt in the RKHS $\mathcal{H}$

- Okay, so I got hold of the eigenvectors $\{\mathbf{v}^i\}$ of S, now what?
- How do I get back a "latent"/low-dim vector $\mathbf{z}$ for a data point x?
- In PCA we used $\mathbf{z} = W^\top \mathbf{x}$ (W stores the eigenvectors)
- Need to kernelize this step as well!
- Suppose I need the top k principal components
- I will take the top k eigenvectors of S, i.e. for $j = 1, \ldots, k$

$$\mathbf{v}^j = \frac{1}{\sqrt{\lambda_j}} \cdot \Phi \boldsymbol{\alpha}^j$$

- ... and calculate $\mathbf{z}_j$ as

$$\mathbf{z}_j = \langle \mathbf{v}^j, \phi(x) \rangle = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} \boldsymbol{\alpha}_i^j K(x^i, x)$$

Mean Centering in RKHS $\mathcal{H}$

- Recall: to make PPCA/PCA work properly, need to center data
- Given: $\mathbf{x}^1, \ldots, \mathbf{x}^n \in \mathbb{R}^d$, calculate mean $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^i$ and work with

$$\tilde{\mathbf{x}}^i = \mathbf{x}^i - \boldsymbol{\mu}$$

- Thus, in an RKHS, we should be working with $\tilde{\phi}_i = \phi(x^i) - \boldsymbol{\mu}$, where $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \phi(x^i)$
- Of course we wont work with $\tilde{\phi}_i$ directly, just the new Gram matrix

$$\tilde{G}_{ij} = \langle \tilde{\phi}_i, \tilde{\phi}_j \rangle$$

- Can show that

$$\tilde{G} = \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right) G \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right)$$

- Do all operations with the *centered* Gram matrix $\tilde{G}$

Mean Centering in RKHS $\mathcal{H}$

- Recall: to make PPCA/PCA work properly, need to center data
- Given: $\mathbf{x}^1, \ldots, \mathbf{x}^n \in \mathbb{R}^d$, calculate mean $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^i$ and work with

$$\tilde{\mathbf{x}}^i = \mathbf{x}^i - \boldsymbol{\mu}$$

- Thus, in an RKHS, we should be working with $\tilde{\phi}_i = \phi(x^i) - \boldsymbol{\mu}$, where $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \phi(x^i)$
- Of course we wont work with $\tilde{\phi}_i$ directly, just the

$$\tilde{G}_{ij} = \langle \tilde{\phi}_i, \tilde{\phi}_j \rangle$$

$$\mathbf{1}_n = \left(\underbrace{1,1,\ldots,1}_{n}\right)^\top$$

- Can show that

$$\tilde{G} = \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right) G \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right)$$

- Do all operations with the *centered* Gram matrix $\tilde{G}$

Exercise

The Kernel PCA

KERNEL PCA

1. Data points $x^1, \ldots, x^n \in \mathcal{X}$, #components $k \leq n$
2. Find centered Gram matrix $\tilde{G} = \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right) G \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right)$
3. Find k largest eigenvectors/values of $\tilde{G}$ as $\left(\boldsymbol{\alpha}^j, \lambda_j\right)_{j=1,\ldots,k}$
4. Let $A = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^k] \in \mathbb{R}^{n \times k}$ and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k \times k}$
5. For a point $x \in \mathcal{X}$, find k-dim representation
   1. Let $\mathbf{k}^x \in \mathbb{R}^n$ such that $\mathbf{k}_i^x = K(x^i, x)$
   2. Return $\mathbf{z} = \sqrt{\Lambda^{-1}} A^\top \mathbf{k}^x \in \mathbb{R}^k$

The Kernel PCA

PCA could have given us at most d components

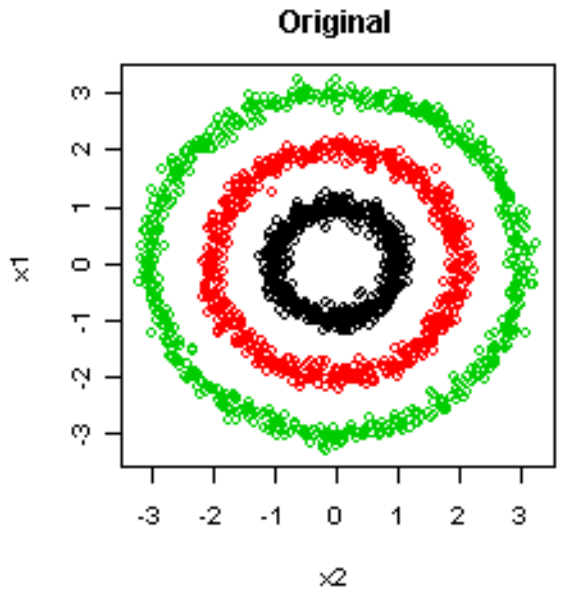
Can recover upto n components even if $\mathcal{X} \subseteq \mathbb{R}^d$ and $d \leq n$

KERNEL PCA

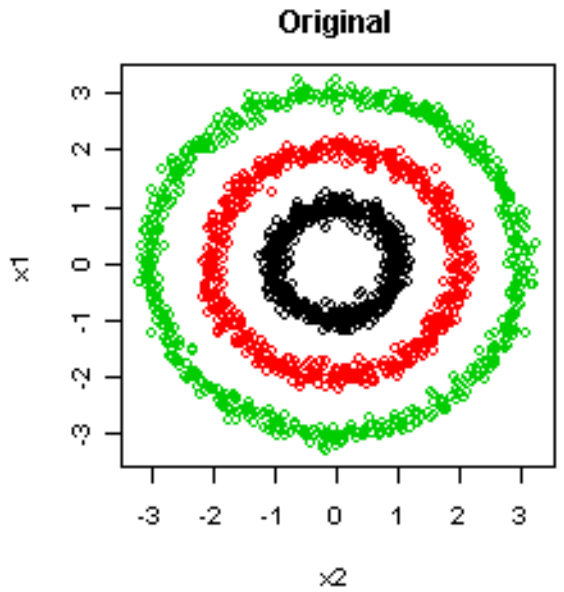
1. Data points $x^1, \ldots, x^n \in \mathcal{X}$, #components $k \leq n$
2. Find centered Gram matrix $\tilde{G} = \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right) G \left(I_n - \frac{\mathbf{1}_n \mathbf{1}_n^\top}{n}\right)$
3. Find k largest eigenvectors/values of $\tilde{G}$ as $\left(\boldsymbol{\alpha}^j, \lambda_j\right)_{j=1,\ldots,k}$
4. Let $A = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^k] \in \mathbb{R}^{n \times k}$ and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k \times k}$
5. For a point $x \in \mathcal{X}$, find k-dim represen
   1. Let $\mathbf{k}^x \in \mathbb{R}^n$ such that $\mathbf{k}_i^x = K\left(x^i, x\right)$
   2. Return $\mathbf{z} = \sqrt{\Lambda^{-1}} A^\top \mathbf{k}^x \in \mathbb{R}^k$

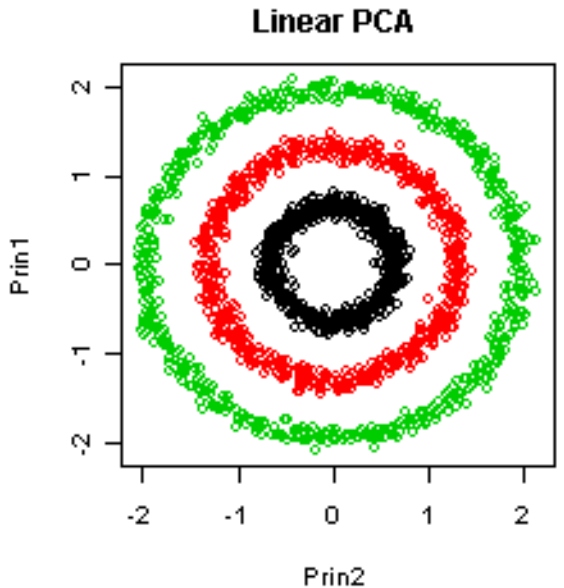
Use Power Method
Takes $\mathcal{O}(kn^2)$ time

Kernel PCA at work

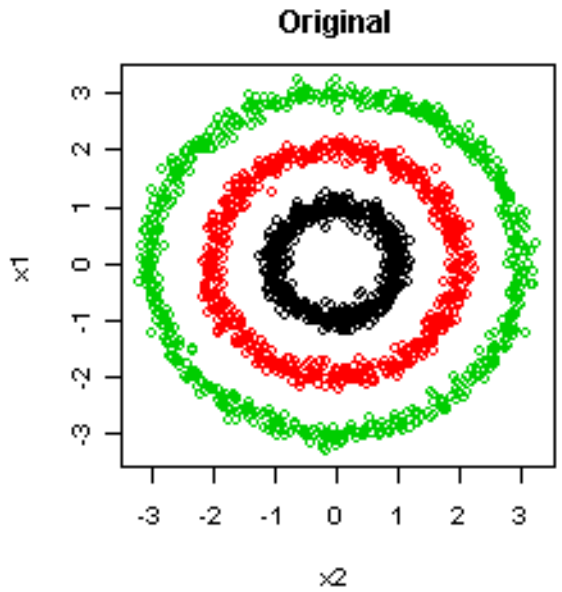


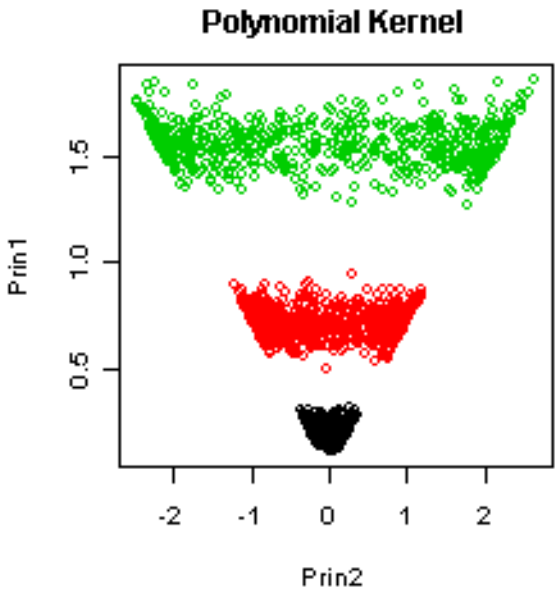
Kernel PCA at work

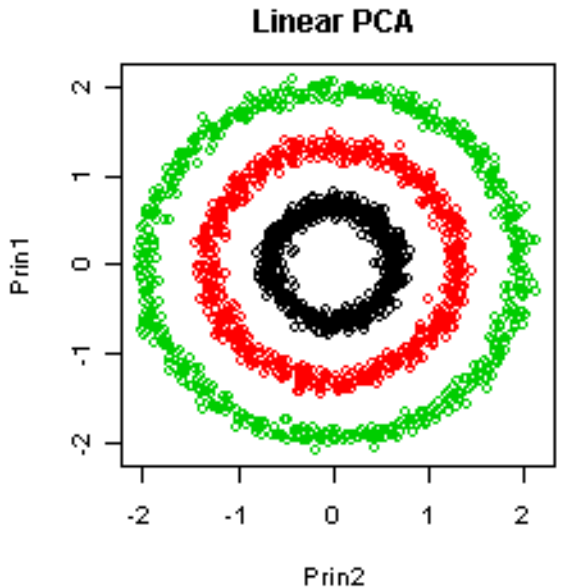




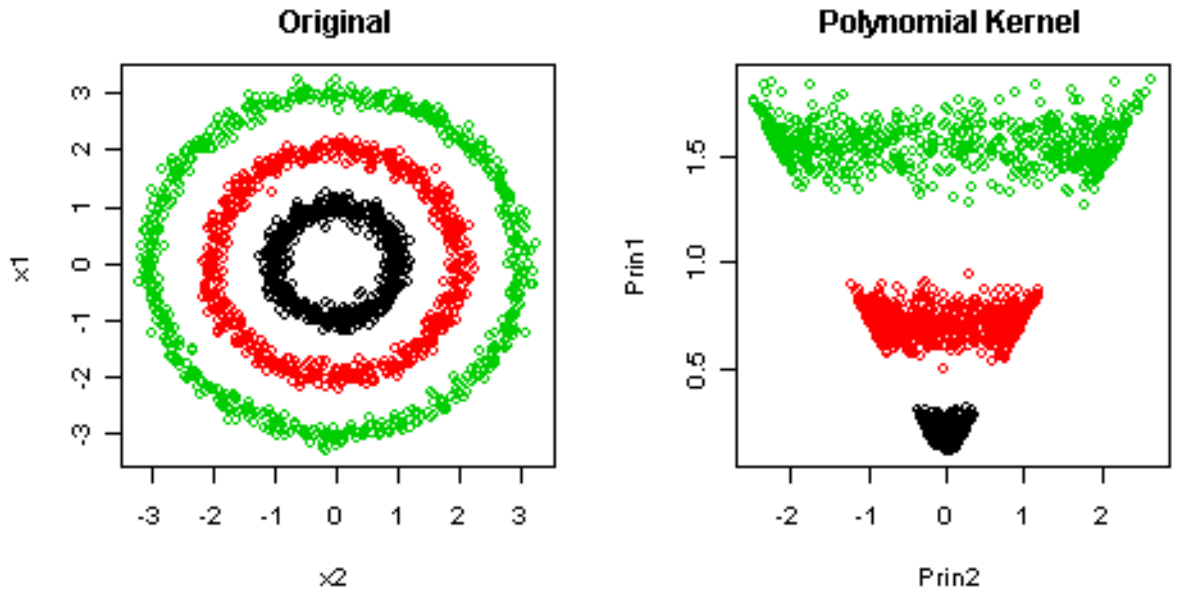
Kernel PCA at work

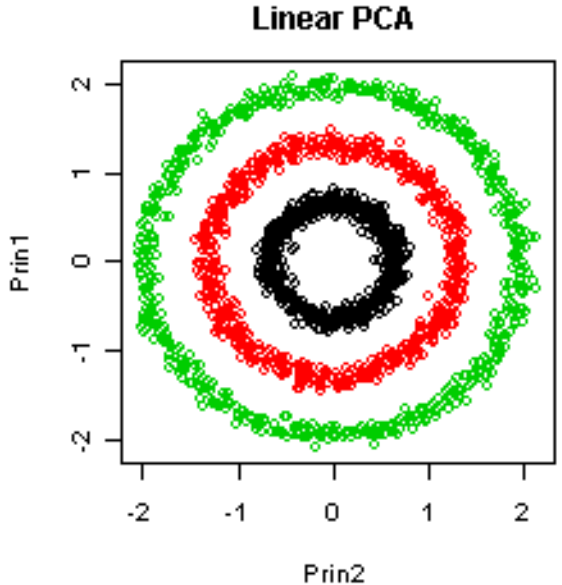


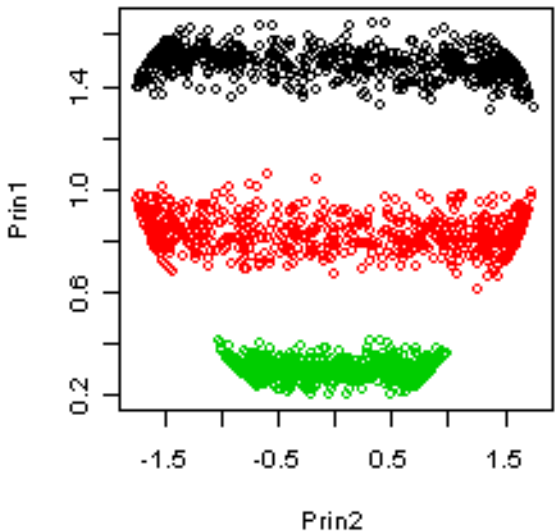




Kernel PCA at work







Accelerated Kernel Methods

Kernel methods can be slow ☹

- Training has to be in the dual since primal is infinite dim.
- Even coordinate descent in kernel SVM takes $O(dn)$ time
- A large portion of training data needs to be stored $O(\tilde{n}d)$
- Long time to predict on a test point $O(\tilde{n}d)$
- Model size and prediction time being large are serious
- Several methods devoted to reducing these two
- Some of them reduce training time as a side effect
- We will see a few of them for kernel SVM
- Similar techniques possible for RR, PCA with kernels too – Explore!

The Tale of a Trio of Techniques

- **Post-processing techniques**: learn the kernel SVM (a bit costly), but then make the model cheaper to store and predict

- **Approximate training techniques**: directly learn a kernel SVM model that is cheap to store and predict

- **Kernel approximation techniques**: use a different kernel than the one you wanted to, so that the new kernel mimics the original one but always gives models that are cheap to store and predict

- Kernel approximation is the most successful of the three

Post Processing Techniques

- Learn the kernel SVM, obtain $\tilde{n}$ support vectors $\left\{x_{i_j}, \alpha_{i_j}\right\}_{j=1,\ldots,\tilde{n}}$
- Obtain a *reduced set* of support vectors of size $k \ll \tilde{n}$
- Use k-means clustering in $\mathbb{R}^d$ on support vectors
- Get a set of k vectors that "approximates" the set $\left\{x_{i_j}\right\}$
- Use these as support vectors and recompute α values for them

- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.

Approximate Training Techniques

- Notice that support vectors are always a subset of training data
- Maybe removing this restriction can reduce their number?
- Learn support vectors as well!
- Learn vectors $\mathbf{z}^1, \ldots, \mathbf{z}^k \in \mathbb{R}^d$ and weights $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$ so that

$$\mathbf{w} = \sum_{i=1}^{k} \alpha_i \cdot \phi_K(\mathbf{z}^i)$$

  is a good model (classifier, regressor etc)
- k chosen based on budget (space, time) of application
- $\mathcal{O}(kd)$ storage and $\mathcal{O}(kd)$ time for prediction

- Joachims and Yu. Sparse Kernel SVMs via Cutting-Plane Training, Machine Learning 76(2):179-193, 2009
- Tsang et al. Core Vector Machines, JMLR 6:363-392, 2005

Please give your Feedback

http://tinyurl.com/ml17-18afb