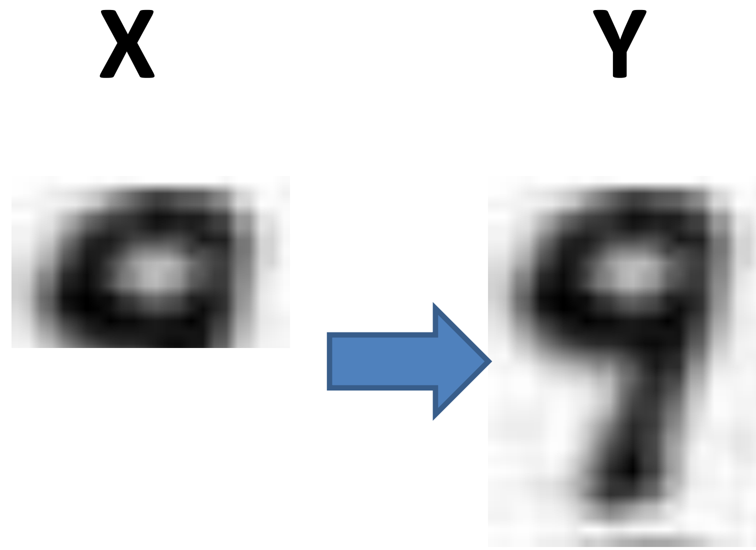
Kernel Dependency Estimation

Jason Weston, Oliver Chapelle, Andre Elisseeff,
Bernard Scholkopf and Vladmir Vapnik

Presentation by: Nathan Knerr and Anshumali
Shrivastava

Example Structured Prediction Problem

- Given half a digit, predict the other half
- We have some structure because it's a digit and we want to take advantage.
- We will come back to this near the end.



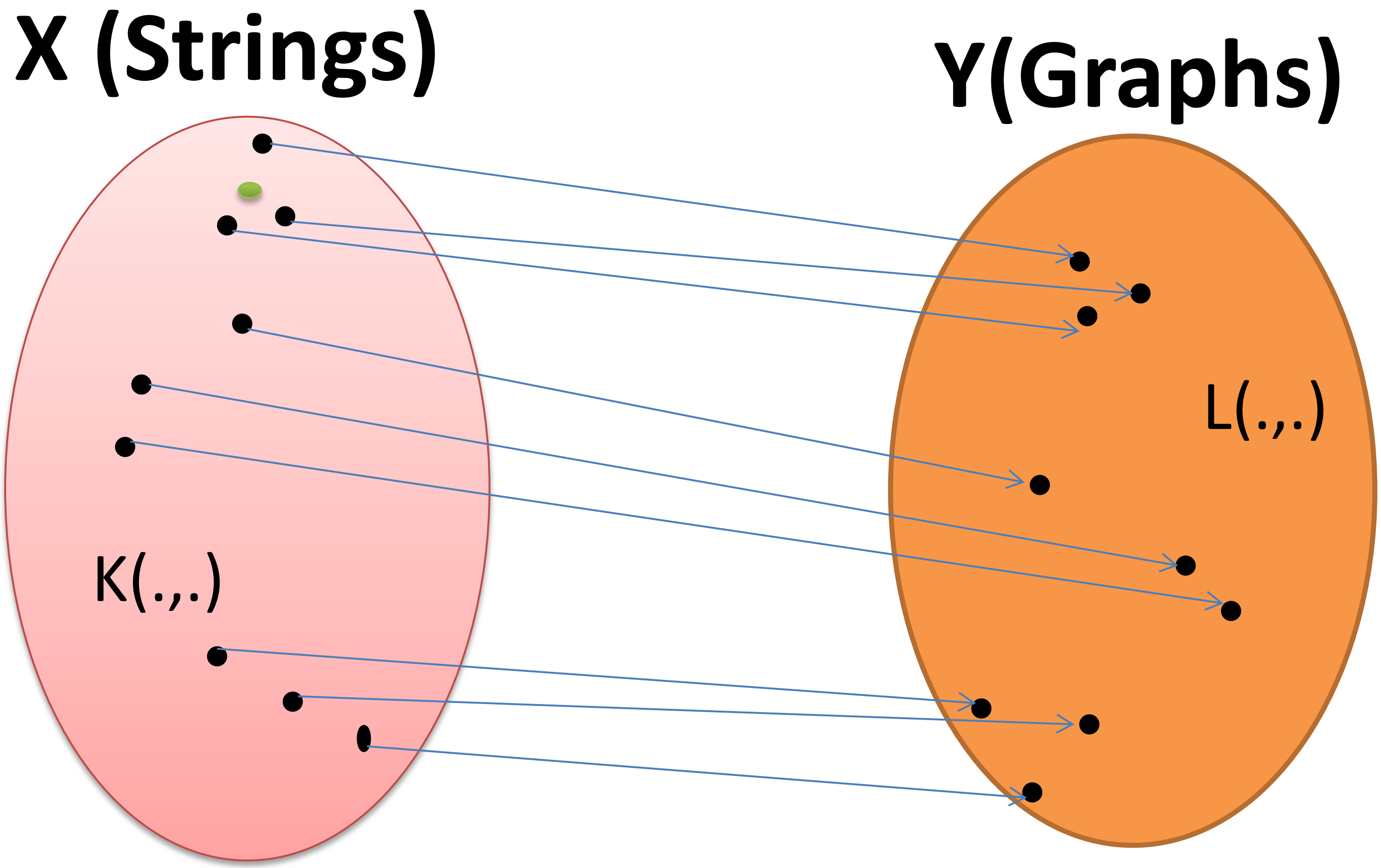
Reminder: Kernels

- Is generalized inner products.

- Creates a distance (d). i.e.
$$\mathrm{d}(\mathrm{x},\mathrm{y}) = \mathrm{k}(\mathrm{x},\mathrm{x}) + \mathrm{k}(\mathrm{y},\mathrm{y}) - 2\mathrm{k}(\mathrm{x},\mathrm{y})$$

- Behaves like vector spaces (Hilbert Spaces).

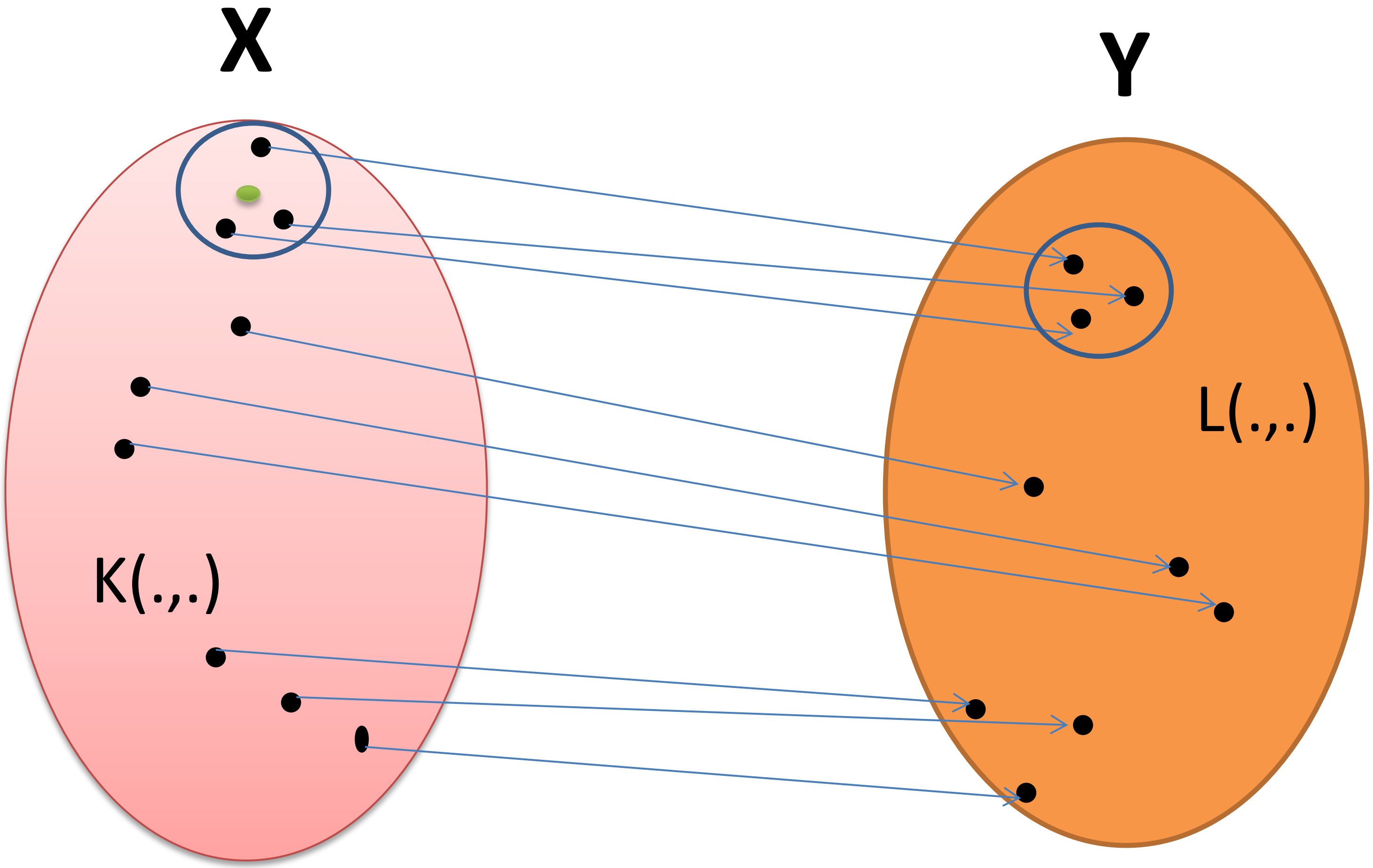
Are Kernels Enough ?



How do we reasonably classify the Green Point ?

Nearest Neighbor ?

Near Neighbors



What is the problem with this approach ?

Highlights of Paper

- Kernels (or distances) in the input and output spaces is sufficient for efficient structured prediction.
- Generic Framework for Structured Prediction
  - Need a notion of similarity in input space
  - Loss function serves as kernels in output space.
- Eliminates the need to perform feature extraction when kernels known.

Advantages of Kernels

- Right representation not always available.
  - Strings ?
  - Graphs ?
- Many applications dealing with complex objects have standard notions of similarity.
  - String Distances
  - Graph Kernels
- Feature representation may not be efficient.
  - Radial Basis Functions (RBF)

Example Kernels

- Multi-class pattern recognition:

$$l(y, y') = \frac{1}{2}(y == y')$$

- Regression Estimation:

$$l(y, y') = \vec{y} \cdot \overrightarrow{y'}$$

- Multinomial

$$l(y, y') = (y \cdot y' + 1)^p$$

- Radial Basis functions

$$l(y, y') = \exp\left(-\frac{\left||y - y'|\right|^2}{2\sigma^2}\right)$$

- Arbitrary distance matrix($\Delta\left(y_i, y_j\right) = D_{ij}$)

$$l\left(y_i, y_j\right) = \frac{\left(\left|D_{ij}\right|^2 - \sum_{p=1}^{m} c_p \left|D_{ip}\right|^2 - \sum_{q=1}^{m} c_q \left|D_{qj}\right|^2 + \sum_{p,q=1}^{m} c_p c_q \left|D_{pq}\right|^2\right)}{2}$$

Plenty of options!

See also: *Learning with Kernels* by Scholkopf and Smola (2002)

ALGORITHM

Problem

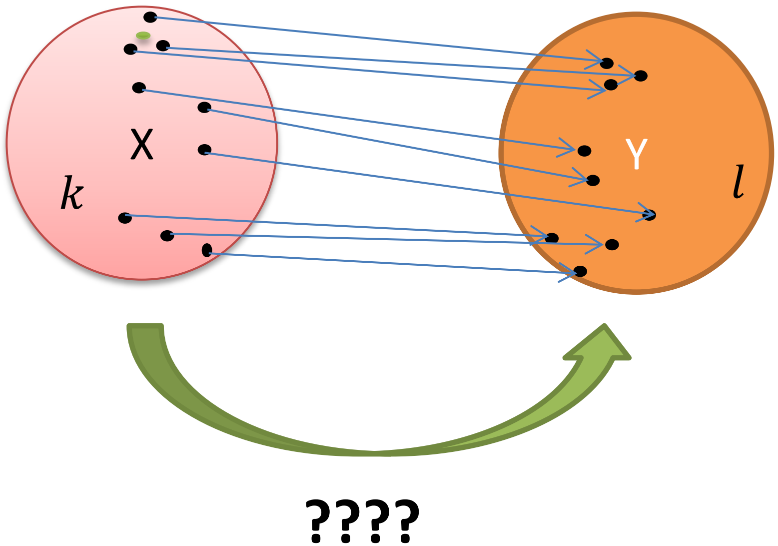
Goals

Givens

- Kernel in input k
- Loss Function (l)

Output

- A predicted structure y for some arbitrary structured input x.



Approach

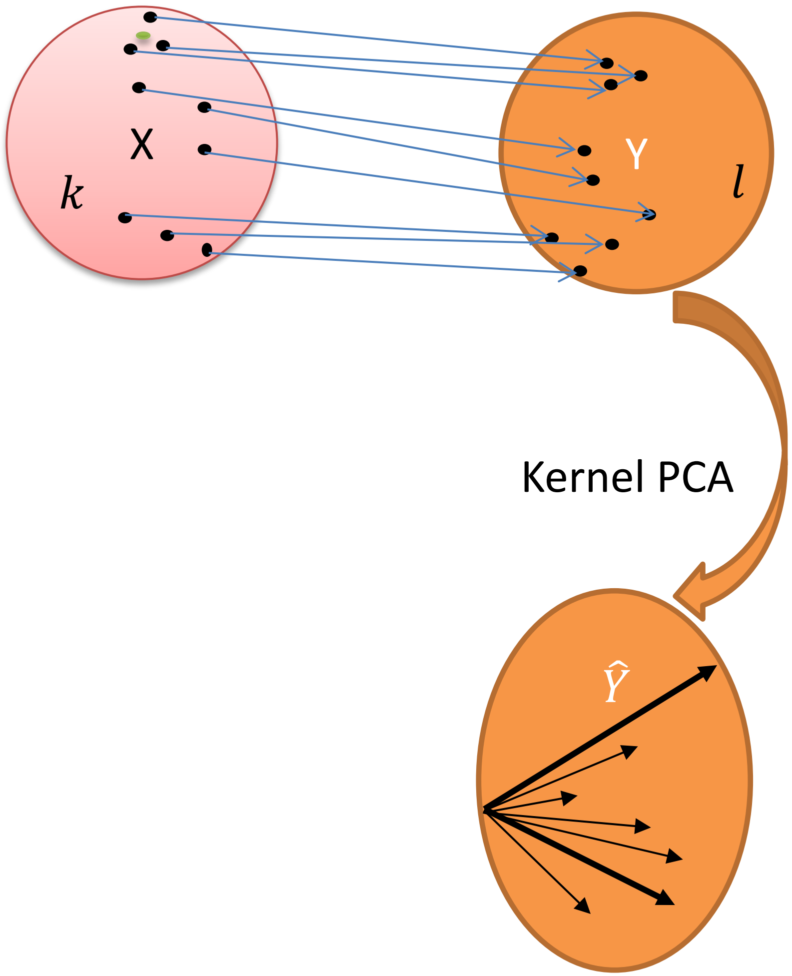
Goals

Givens

- Kernel in input k
- Loss Function (l)

Output

- A predicted structure y for some arbitrary structured input x.



Basic Steps

Learning

1) **Kernel PCA**
2) Ridge regression

Testing

1) Finding a “good” output

Approach

Goals

Givens

- Kernel in input k
- Loss Function (l)

Output

- A predicted structure y for some arbitrary structured input x.

Basic Steps

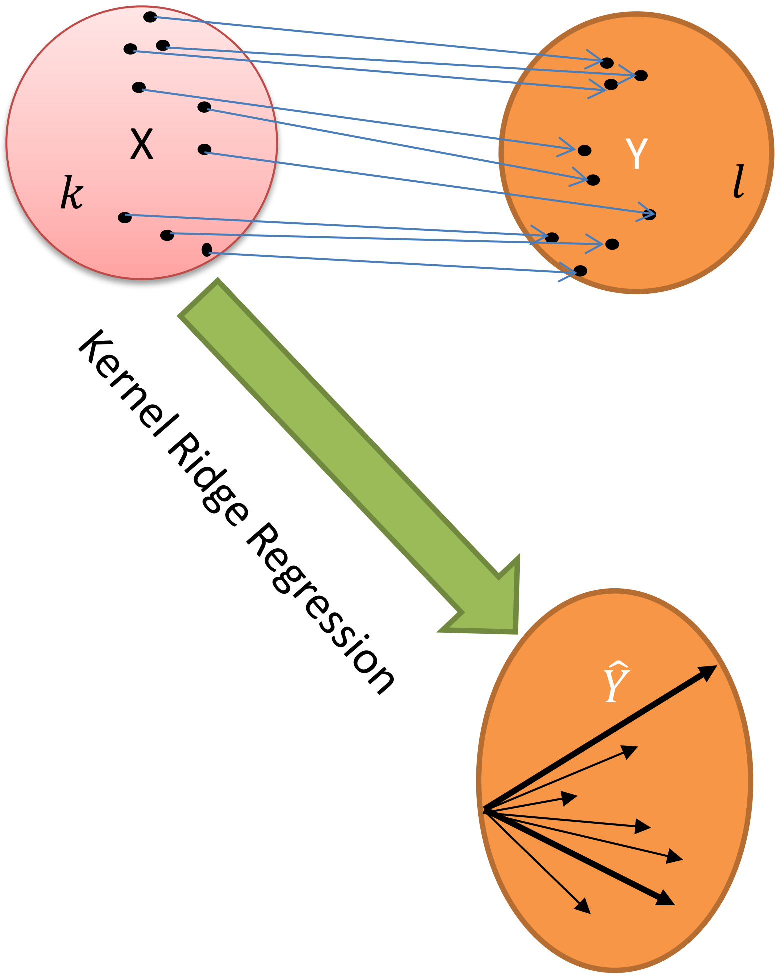
Learning

1) Kernel PCA

2) Ridge regression

Testing

1) Finding a “good” output



Approach

Goals

Givens

- Kernel in input k
- Loss Function (l)

Output

- A predicted structure y for some arbitrary structured input x.

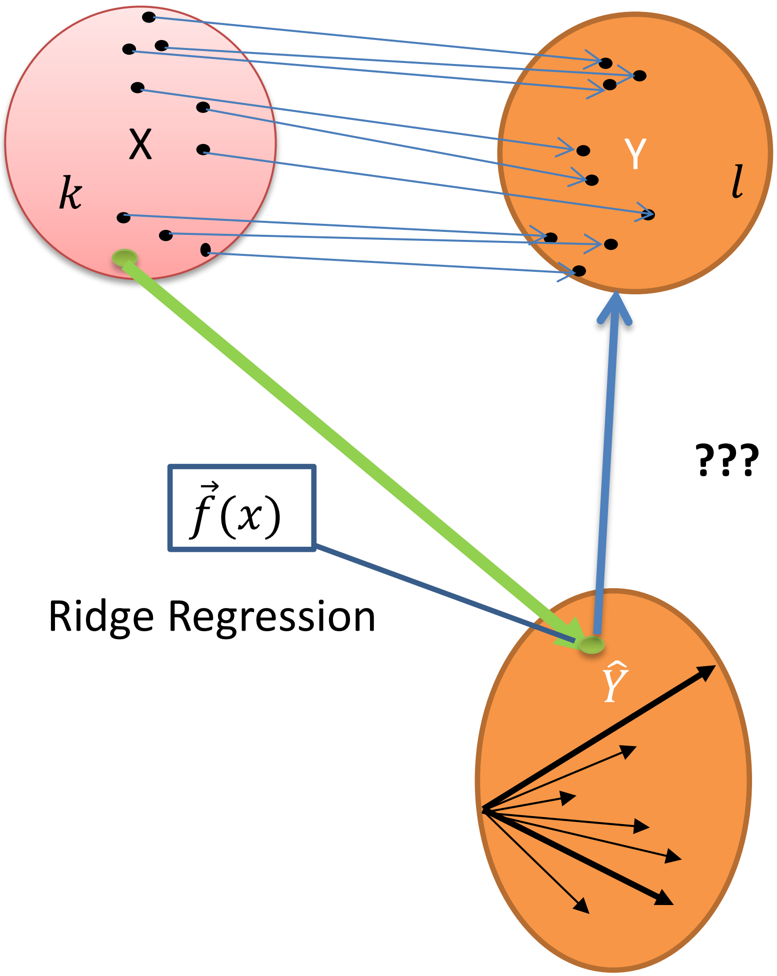
Basic Steps

Learning

1) Kernel PCA
2) Ridge regression

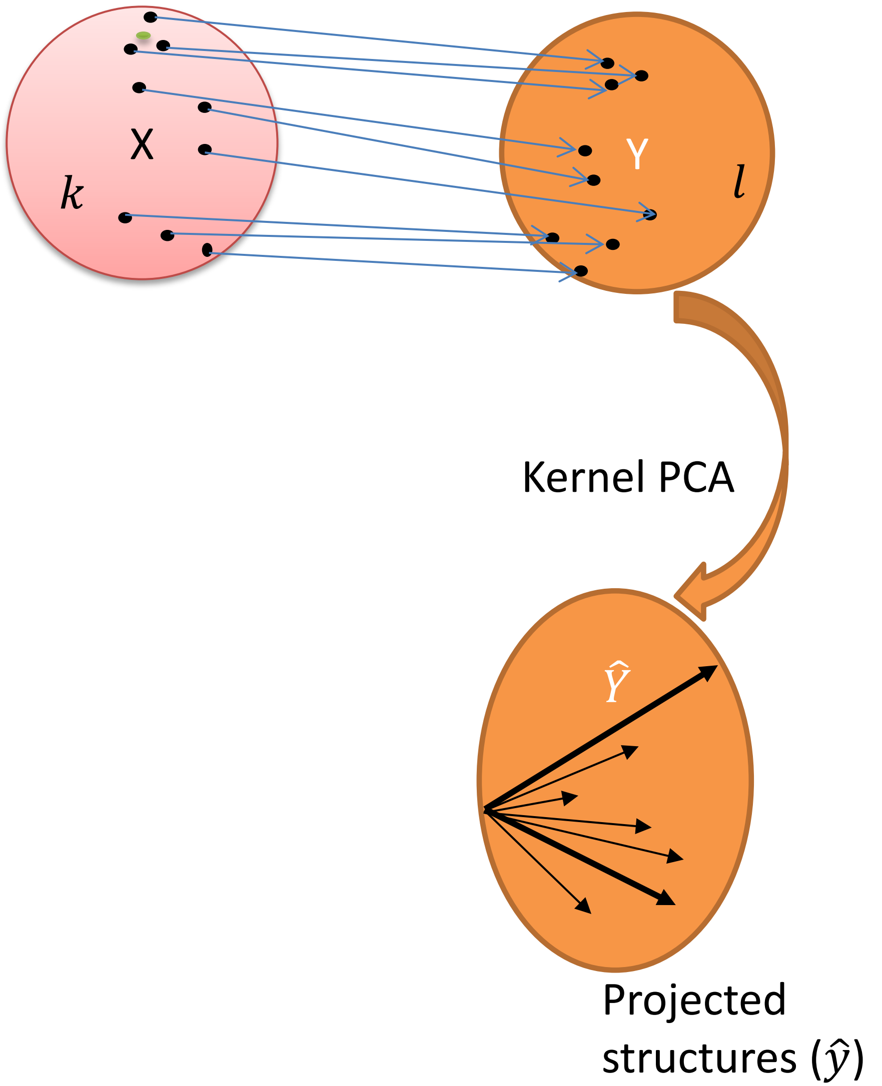
Testing

1) Finding a "good" output



Kernel PCA on Outputs - Goal

- Basically finding a set of vectors that yield a good representation of the labels.
  - Represents the output space as vectors that can learned in the next step
  - Kernelized analog of Principal Component Analysis



Kernel PCA-Setting

- Input: Data objects y_i with defined kernel functions $l(y_i, y_j)$.
- Output: A vector representation $\Phi_l(y) \in R^p$

$$\boxed{\text{s. t. } \; l(y, y') \cong \Phi_l(y) \cdot \Phi_l(y')}$$

Note: Only access to $l\,(\,\cdot,\cdot\,)$ allowed. How to compute such $\Phi_l(y)$?

Kernel PCA-Idea

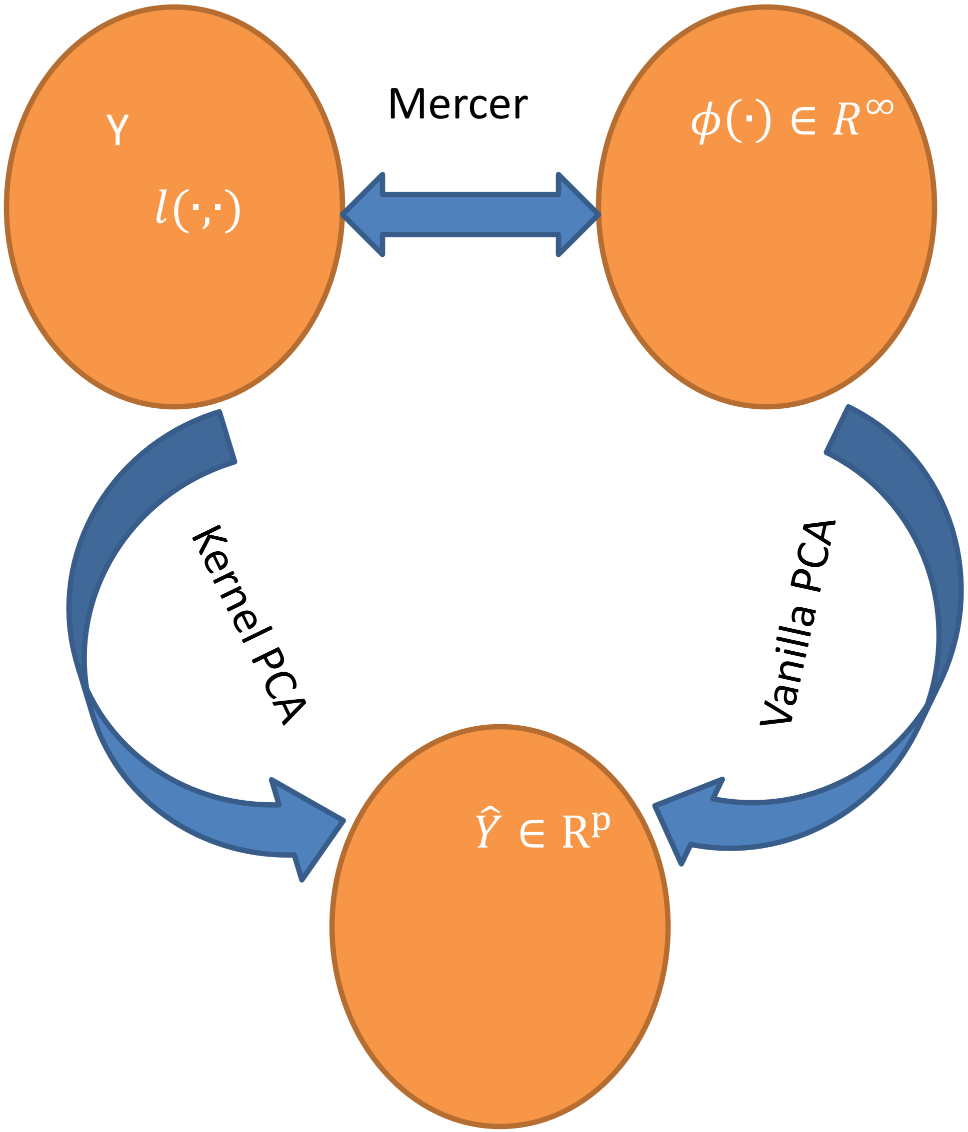
- Mercers Theorem: Every kernel $l(\cdot,\cdot)$ has an associated feature space $\phi(\cdot)$, such that

$$l(y_i, y_j) = \phi(y_i)^T \phi(y_j).$$

$\phi(\cdot)$ exist but we don't know how to find it.

- But we can get the PCA representation of

$\phi(\cdot)$, using only access to $l(\cdot,\cdot)$!!



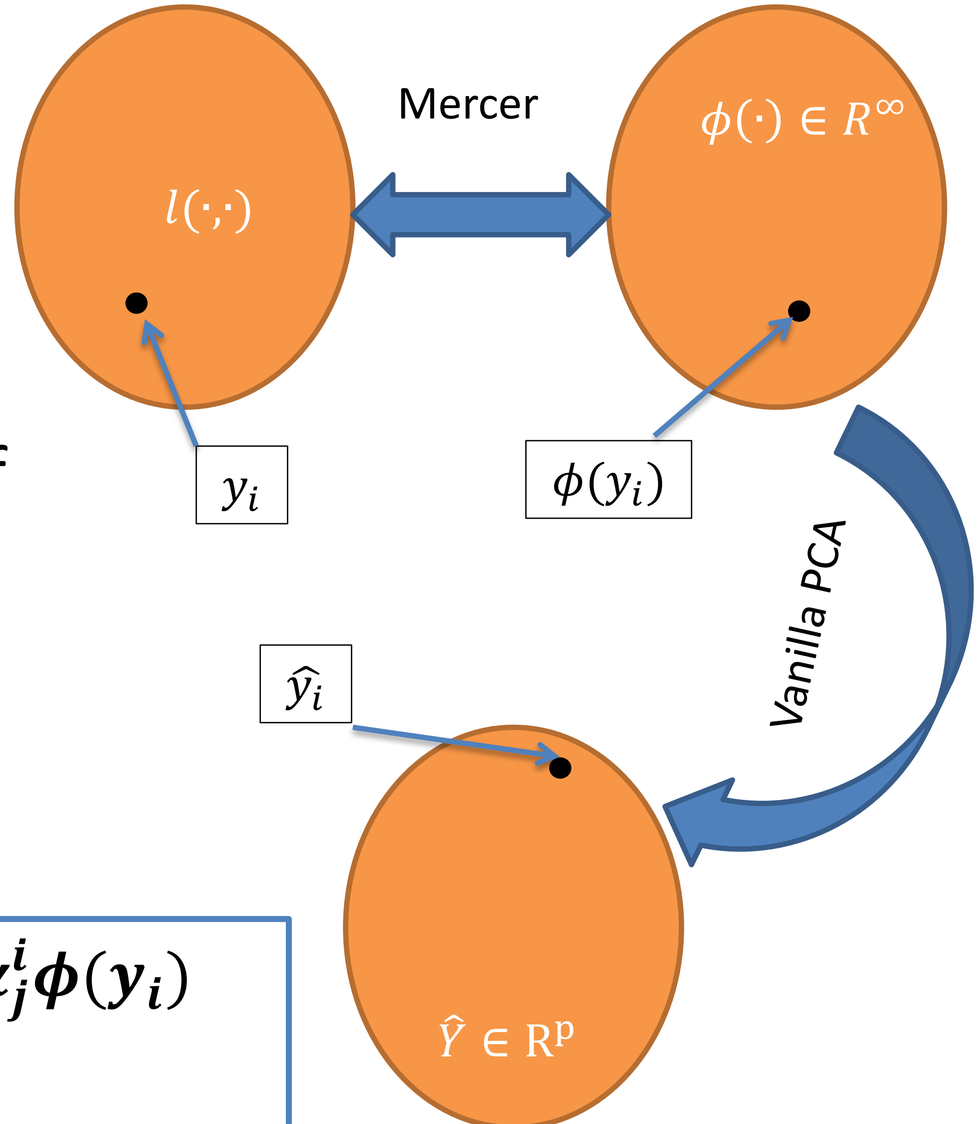
Y
$l(\cdot;\cdot)$
Mercer
$\phi(\cdot) \in R^{\infty}$
Kernel PCA
Vanilla PCA
$\hat{Y} \in \mathrm{R}^{\mathrm{p}}$

Vanilla PCA in $\phi(\cdot)$

- $\boldsymbol{\phi(y_i) = \phi(y_i) - \frac{1}{m}\sum_{k=1}^{n}\phi(y_k)}$
- Solve $Cv_j = \lambda_j v_j$

  $C = \frac{1}{m}\sum_{i=1}^{m}\phi(y_i)\phi(y_i)^T$
- The j^{th} component of $\widehat{y_i}$

  $$\widehat{y_i}^{\,j} = \phi(y_i)^T v_j$$
- Ensure $v_j^T v_j = 1$



Key observation : $\boldsymbol{v_j = \sum_{i=1}^{m}\alpha_j^i\phi(y_i)}$

$\boldsymbol{\phi(y_i)^T\phi(y_j) = l(y_i, y_j)}$

Kernel PCA

- $L' = \left(I - \frac{1}{m}1_{mm}\right) L \left(I - \frac{1}{m}1_{mm}\right)$

L is gram matrix, $L_{i,j} = l(y_i, y_j)$.

1_{mm} $m \times m$ matrix of 1's. I is the identity.

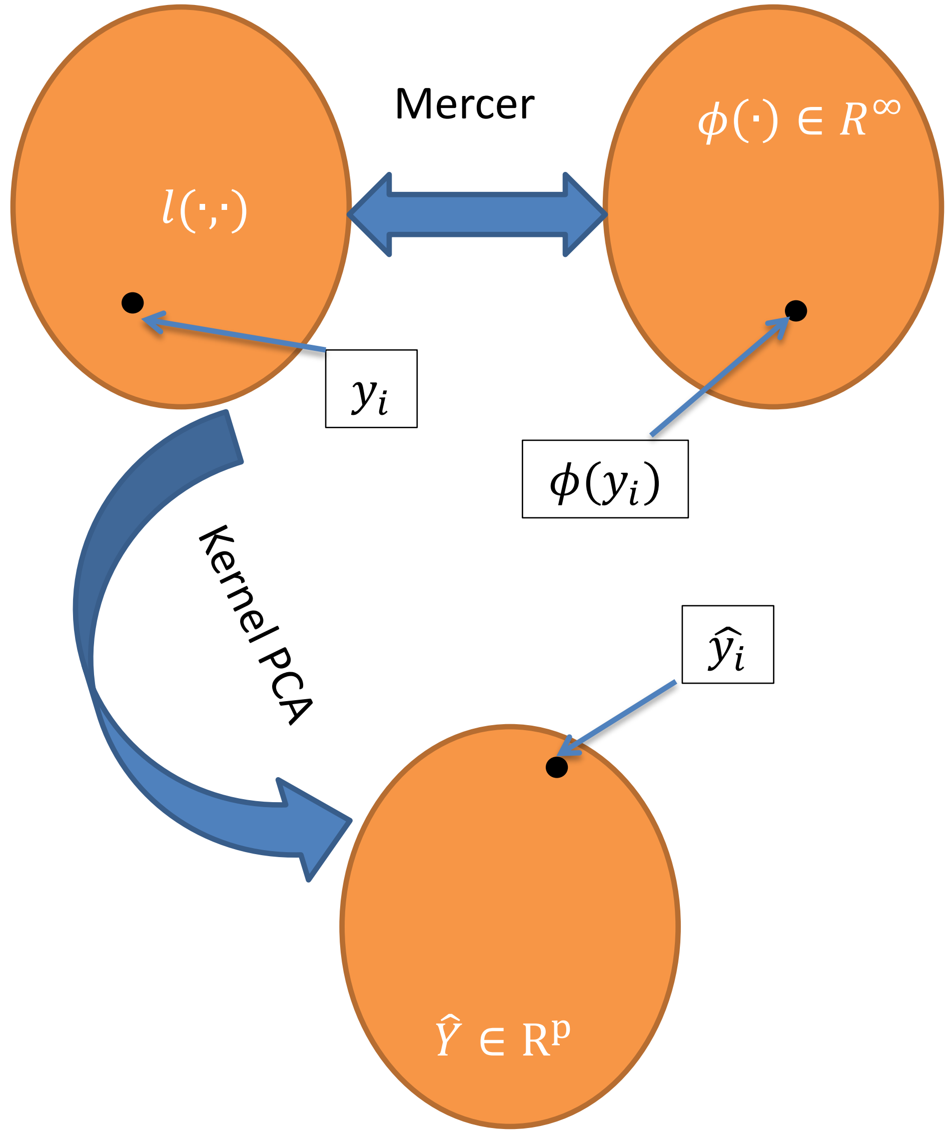
- Solve $\frac{1}{(m)} L' \alpha_j = \lambda_j \alpha_j$

- $\boxed{\hat{y}^j = \sum_{i=1}^{m} \alpha_j^i l(y_i, y)}$

where α_j^i is the i^{th} component of α_j.

- Ensure $\alpha_j^T L' \alpha_j = 1$.

$\phi(\cdot)$ never used !!



Substitution Activity

Materialize $\boldsymbol{\phi}(y)$ - PCA

- Solve $C v_j = \lambda_j v_j$ where

  $$C = \frac{1}{m}\sum_{i=1}^{m} \phi(y_i)\phi(y_i)^T$$

- The j^{th} component of $\hat{y}$

  $$\hat{y}^j = \phi(y)^T v_j$$

- Ensure $v_j^T v_j = 1$

Just use $\boldsymbol{l}(\cdot,\cdot)$ – Kernel PCA

- Solve $\frac{1}{(m)} L' \alpha_j = \lambda_j \alpha_j$

- $\hat{y}^j = \sum_{i=1}^{m} \alpha_j^i l(y_i, y)$ where

α_j^i is the i^{th} component of α_j.

- Ensure $\alpha_j^T L' \alpha_j = 1.$

Key connection : $\boldsymbol{v_j = \sum_{i=1}^{m} \alpha_j^i \phi(y_i)}$

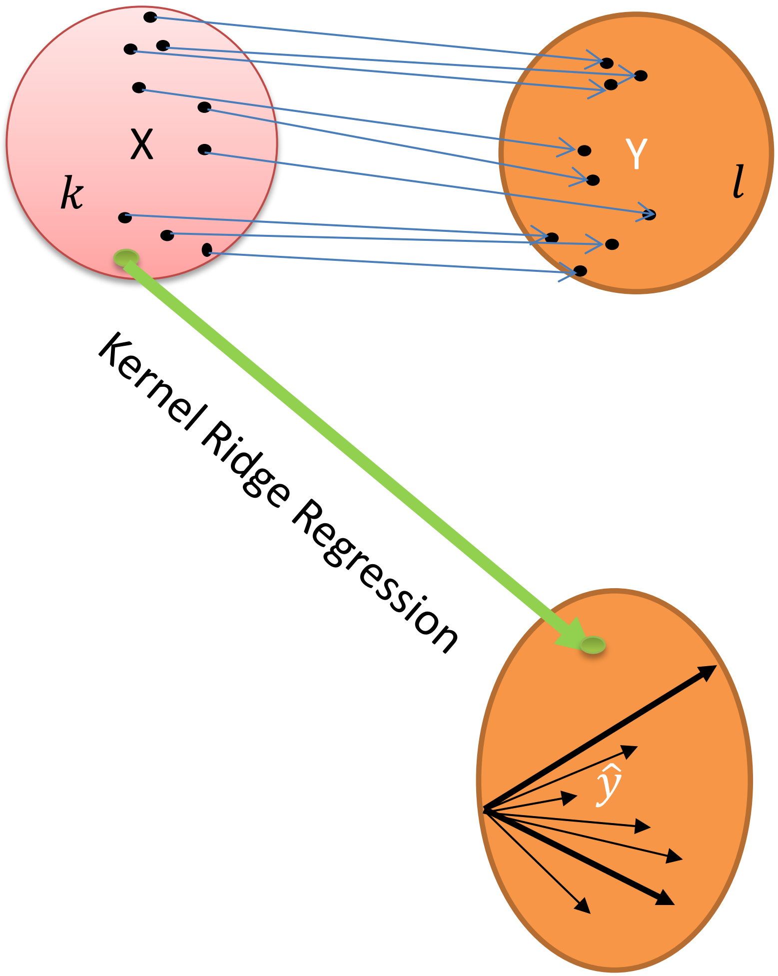
$$\boldsymbol{\phi(y_i)^T \phi(y_j) = l(y_i, y_j)}$$

Kernel Ridge Regression

- Recall we know a 'kernelized' version of the input (X)

- Want to map the input feature space to vectorized outputs

- $x \rightarrow \begin{bmatrix} \hat{y}^1 \\ \dots \\ \hat{y}^p \end{bmatrix}$



Kernel Ridge Regression

- Objective (primal version):

$$\min_{w}\left(\gamma\left|\left|w\right|\right|^2+\frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}_i-\left(\beta\cdot\Phi_k(x_i)\right)\right)^2\right)$$

- Convert to dual form and solve to find the predicted location in the projected y space:

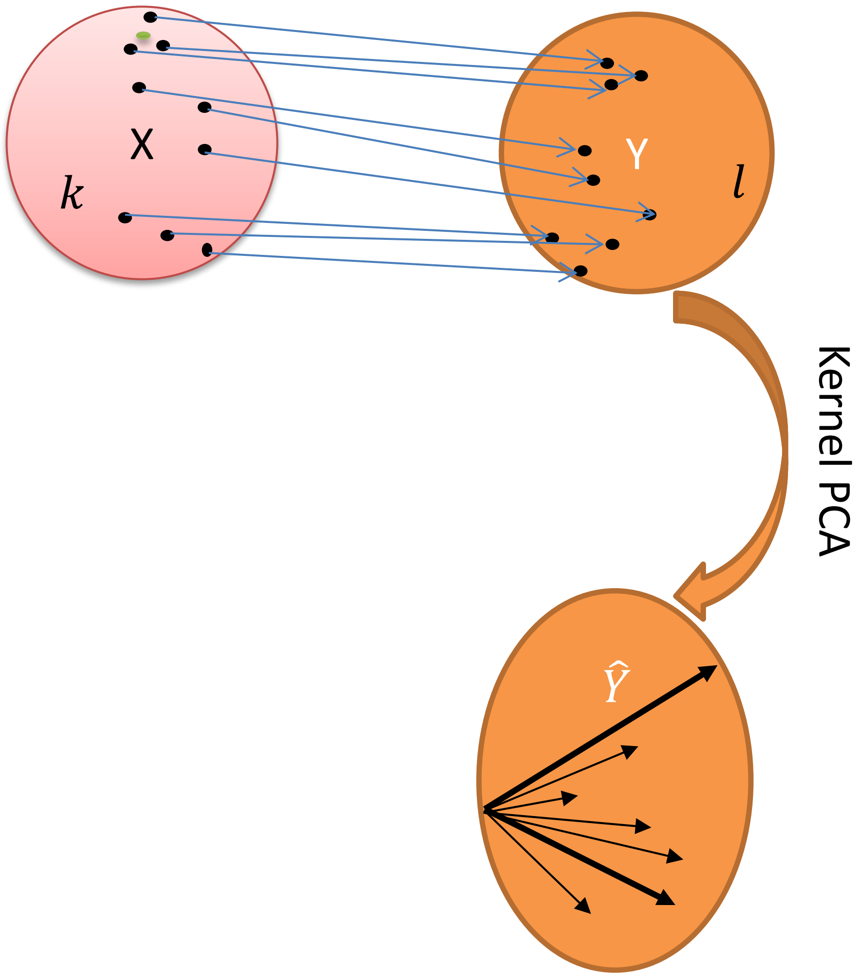
$$f_n(x)=\sum_{i=1}^{m}\beta_i^n k(x_i,x)$$

Where

$$\beta^n=(K+\gamma I)^{-1}\hat{y}^n$$

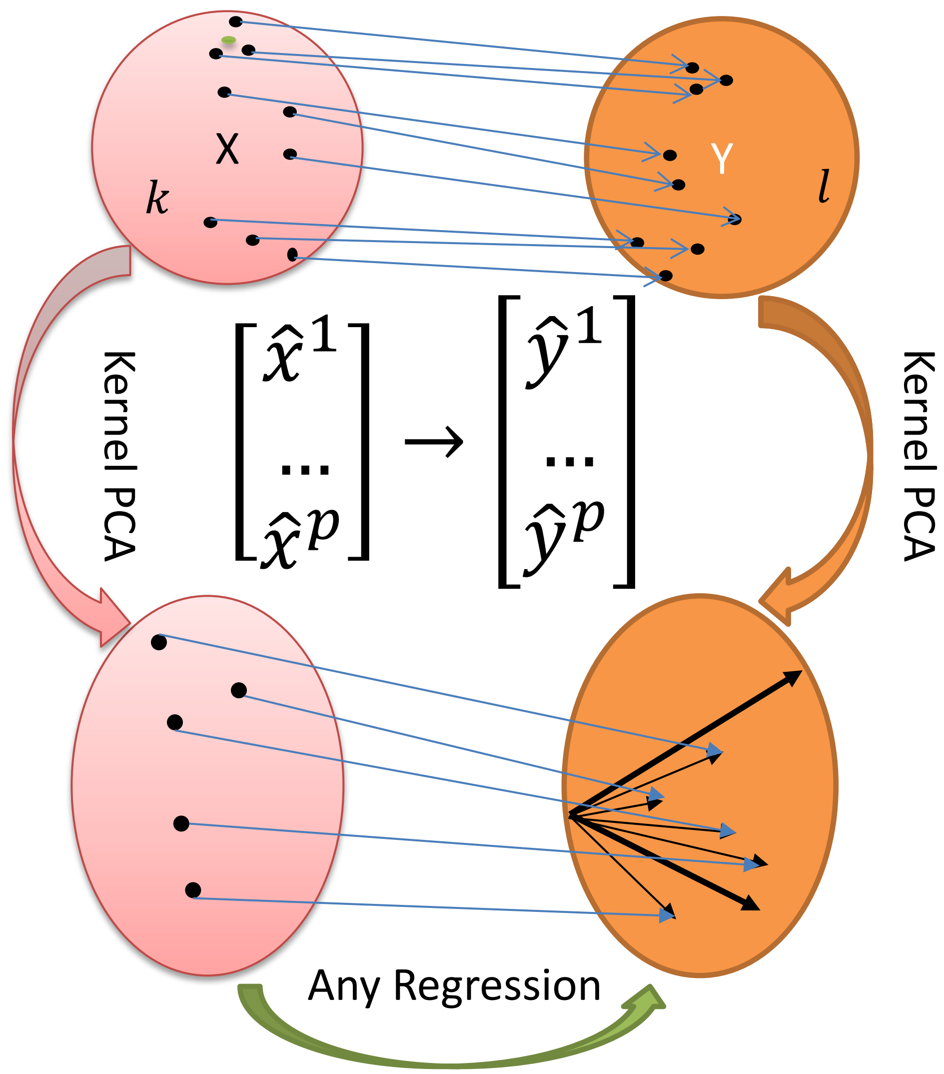
Activity

- I don’t know Kernel (Ridge) regression
- But I know Kernel PCA and Linear (Ridge) Regression
- Can I still make it work ?



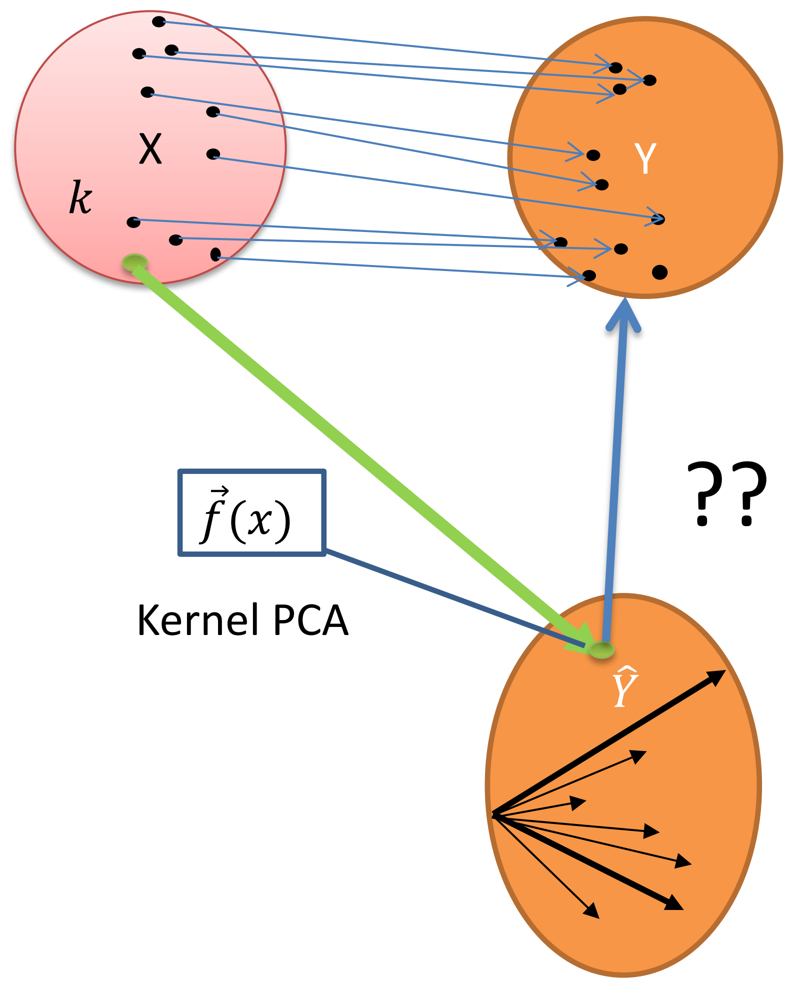
Activity

- Yes !!
- Use kernel PCA on input space to get vector representation
- Input output both vector spaces. Use linear regression.



Inference

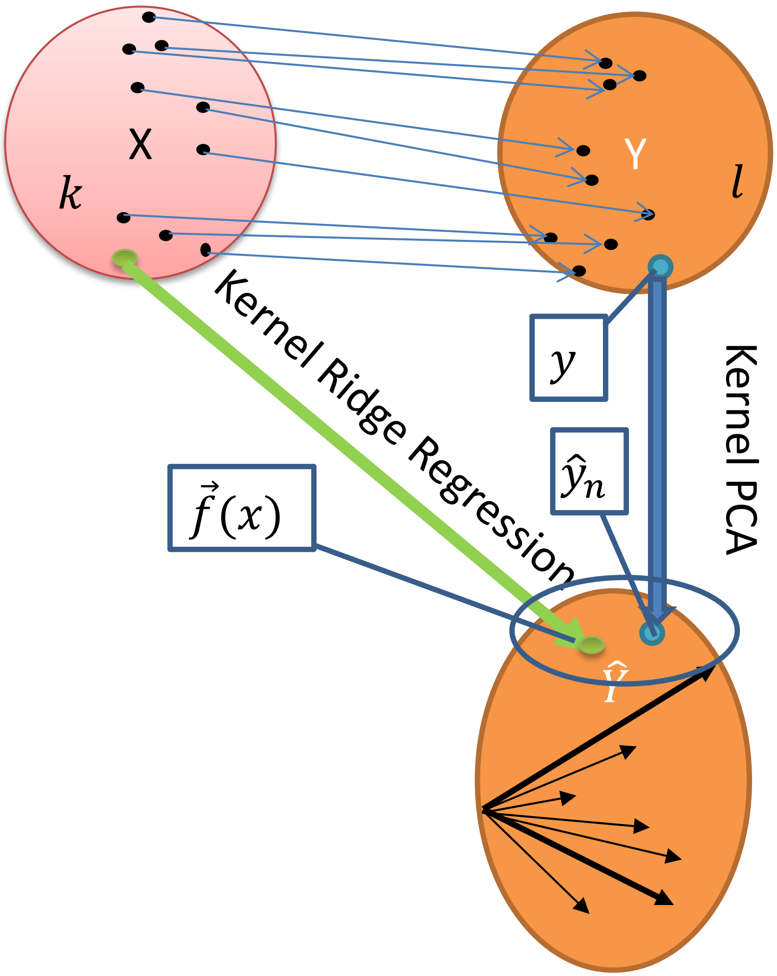
- Just found a way to estimate what we think the input (x) should map to in the projected space
- Need to find the actual structured output y that most closely matches $\vec{f}(x)$.



Inference

- Project all Ys to $\hat{y}$

- Find $\widehat{y_n}$ nearest to $\vec{f}(x)$ in the vector space $\hat{y}$.



Inference

- Formally:

$$y(x) = argmin_{y \in Y} \left\| \begin{bmatrix} \widehat{y_1} \\ \dots \\ \widehat{y_p} \end{bmatrix} - \begin{bmatrix} f_1(x) \\ \dots \\ f_p(x) \end{bmatrix} \right\|$$

- Where $y \to \left[\widehat{y_1}, \widehat{y_2}, \dots, \widehat{y_p}\right]$ via Kernel PCA.
- $f_n(x)$ is done from learned Kernel Ridge Regression.
- Some kernels can be inverted explicitly
- Paper simply searched *all possible y's*

More info: Scholkopf et. al. “Input space Vs feature space in kernel-based methods”

Expensive

- Formally:

$$y(x) = argmin_{y \in Y} \left\| \begin{bmatrix} \widehat{y_1} \\ \dots \\ \widehat{y_p} \end{bmatrix} - \begin{bmatrix} f_1(x) \\ \dots \\ f_p(x) \end{bmatrix} \right\|$$

- Note: slow.
  The recall the formula for kernel PCA is:

$$\widehat{y_j} = \sum_{i=1}^{m} \alpha_j^i l(y_i, y)$$

That is, for each $y \in Y$ sum over all training data

- Must be done each time we look at a new $y \in Y$

EXPERIMENTS

Strings to Strings

Class	Base output	Uniform or Prefer Repeat	Input Alphabet
1	abad	Uniform	[a,b,c,d]
2	dbbd	0.7 repeat. Uniform otherwise	[a,b,c,d]
3	aabc	0.7 repeat. Uniform otherwise	[c,d]

- All outputs subject to a 0.3 chance of a random insert/delete and a 0.15 chance of 2 random inserts/delete
- 200 strings, 5 fold cross validated
- Substring Kernel, normalized in both input and output
- Loss is computed via the kernel in the output.
- In the space induced by the input kernel, used RBF kernel

	Kernel Dependency Est.	K-Nearest Neighbors
String Loss	0.676 +/- 0.030	0.985 +/- 0.029
Classification Loss	0.125 +/- 0.012	0.205 +/- 0.026

For more details see the paper

USPS Image Reconstruction

- Given top half of a USPS digit want the lower half
- Not given the digit—have to infer from top half
- The tricky part is choosing a good loss function
- Use an RBF kernel with a width designed to match k-means
- 1000 digits 5-fold Cross Validated
- Hopfield net is a neural network

	Loss
Kernel Dependency Estimation	0.8384+/-0.0077
K Nearest Neighbors	0.8960+/-0.0052
Hopfield Net	1.2190+/-0.0072

USPS Optical Character Recognition

- Same USPS database as before
- Different Expiriment
- Classifying handwritten digits
- 1000 16x16 pixel digits with 5 folds.
- Variables for all algorithms optimized on one fold
- RBF kernel for input
- 0-1 loss multi-class loss on the output

	Kernel Dependency Est	1-vs-rest SVM	K-Nearest Neighbors
0-1 loss	0.0798 +/- 0.0067	0.0847 +/- 0.0064	0.1250 +/- 0.0075

For reference (from Learning with Kernels from Smola):
One-versus rest SVM trains one classifier per class and then assigns it to the maximal class:

Conclusions

- Structured Output Prediction
- only need a loss function kernel and a kernel in the input space.
- Kernels are capable of modeling things that would require infinitely many features to represent
- Kernels PCA gives an implicit feature representation

Any Questions?