Kernel Dependency Estimation

Jason Weston, Oliver Chapelle, Andre Elisseeff, Bernard Scholkopf and Vladimir 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.

\[ d(x, y) = k(x, x) + k(y, y) - 2k(x, y) \]

• Behaves like vector spaces (Hilbert Spaces).
Are Kernels Enough?

X (Strings)  Y (Graphs)

K(.,.)  L(.,.)

How do we reasonably classify the Green Point?
Nearest Neighbor?
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:
  \[ l(y, y') = \bar{y} \cdot \bar{y}' \]

- Multinomial
  \[ l(y, y') = (y \cdot y' + 1)^p \]

- Radial Basis functions
  \[ l(y, y') = \exp \left( -\frac{|y - y'|^2}{2\sigma^2} \right) \]

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

Plenty of options!

See also: *Learning with Kernels* by Scholkopf and Smola (2002)
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 be 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 $\Phi_l(y) \in \mathbb{R}^p$

\[
\text{s. t. } l(y, y') \cong \Phi_l(y) \cdot \Phi_l(y')
\]

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 \( l(\cdot, \cdot) \) !!
Vanilla PCA in $\phi(\cdot)$

- $\phi(y_i) = \phi(y_i) - \frac{1}{m} \sum_{k=1}^{n} \phi(y_k)$
- Solve $C \nu_j = \lambda_j \nu_j$
  
  $C = \frac{1}{m} \sum_{i=1}^{m} \phi(y_i)\phi(y_i)^T$
- The $j^{th}$ component of $\hat{y}_i$
  
  $\hat{y}_i^j = \phi(y_i)^T \nu_j$
- Ensure $\nu_j^T \nu_j = 1$

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

$\phi(y_i)^T \phi(y_j) = l(y_i, y_j)$
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 \( \frac{1}{m} L' \alpha_j = \lambda_j \alpha_j \)

- \( \hat{y}^j = \sum_{i=1}^{m} \alpha_j^i l(y_i, y) \)
  
  where \( \alpha_j^i \) is the \( i^{th} \) component of \( \alpha_j \).

- Ensure \( \alpha_j^T L' \alpha_j = 1 \).

\( \phi(\cdot) \) never used !!
Substitution Activity

Materialize $\phi(y)$ - PCA

- Solve $Cv_j = \lambda_j v_j$ where

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

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

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

- Ensure $v_j^Tv_j = 1$

Just use $l(\cdot,\cdot) -$ 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 $\alpha_j^i$ is the $i^{th}$ component of $\alpha_j$.

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

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

$$\phi(y_i)^T \phi(y_j) = l(y_i, y_j)$$
Kernel Ridge Regression

• Recall we know a ‘kernelized’ version of the input \((X)\)

• Want to map the input feature space to vectorized outputs

\[
x \rightarrow \begin{bmatrix}
\hat{y}^1 \\
... \\
\hat{y}^p 
\end{bmatrix}
\]
Kernel Ridge Regression

• Objective (primal version):

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

• 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?
• Yes !!

• Use kernel PCA on input space to get vector representation

• Input output both vector spaces. Use linear regression.

\[
\begin{align*}
\hat{X}^1 & \rightarrow \hat{Y}^1 \\
... & \rightarrow ... \\
\hat{X}^p & \rightarrow \hat{Y}^p
\end{align*}
\]
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 \( \hat{f}(x) \).
Inference

• Project all Ys to $\hat{\gamma}$

• Find $\hat{\gamma}_n$ nearest to $\tilde{f}(x)$ in the vector space $\hat{\gamma}$. 
Inference

- Formally:

\[ y(x) = \arg \min_{y \in Y} \left\| \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_p \end{bmatrix} - \begin{bmatrix} f_1(x) \\ \vdots \\ f_p(x) \end{bmatrix} \right\| \]

- Where \( y \rightarrow [\hat{y}_1, \hat{y}_2, ..., \hat{y}_p] \) 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:

\[ y(x) = \arg\min_{y \in Y} \left\| \begin{bmatrix} \widehat{y}_1 \\ \vdots \\ \widehat{y}_p \end{bmatrix} - \begin{bmatrix} f_1(x) \\ \vdots \\ f_p(x) \end{bmatrix} \right\| \]

• 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

<table>
<thead>
<tr>
<th>Class</th>
<th>Base output</th>
<th>Uniform or Prefer Repeat</th>
<th>Input Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abad</td>
<td>Uniform</td>
<td>[a,b,c,d]</td>
</tr>
<tr>
<td>2</td>
<td>dbbd</td>
<td>0.7 repeat. Uniform otherwise</td>
<td>[a,b,c,d]</td>
</tr>
<tr>
<td>3</td>
<td>aabc</td>
<td>0.7 repeat. Uniform otherwise</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th></th>
<th>Kernel Dependency Est.</th>
<th>K-Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Loss</td>
<td>0.676 +/- 0.030</td>
<td>0.985 +/- 0.029</td>
</tr>
<tr>
<td>Classification Loss</td>
<td>0.125 +/- 0.012</td>
<td>0.205 +/- 0.026</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Dependency Estimation</td>
<td>0.8384±0.0077</td>
</tr>
<tr>
<td>K Nearest Neighbors</td>
<td>0.8960±0.0052</td>
</tr>
<tr>
<td>Hopfield Net</td>
<td>1.2190±0.0072</td>
</tr>
</tbody>
</table>
USPS Optical Character Recognition

- Same USPS database as before
- Different Experiment
- 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

<table>
<thead>
<tr>
<th>Kernel Dependency Est</th>
<th>1-vs-rest SVM</th>
<th>K-Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 loss</td>
<td>0.0798 +/- 0.0067</td>
<td>0.0847 +/- 0.0064</td>
</tr>
</tbody>
</table>

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?