Course 495: Advanced Statistical Machine Learning/Pattern Recognition

- Goal (Lecture): To present Kernel Principal Component Analysis (KPCA) (and give a small flavour of Auto-encoders).
Materials

- Pattern Recognition & Machine Learning by C. Bishop Chapter 12
Non-linear Component Analysis

\[ x_i \in \mathbb{R}^F \quad \rightarrow \quad \varphi(x_i) \in H \]

\( H \) can be of arbitrary dimensionality
(could be even infinite)
Kernel Principal Component Analysis

- $\varphi(.)$ may not be explicitly known or is extremely expensive to compute and store.

- What is explicitly known is the dot product in $H$ (also known as kernel $k$)

\[
\varphi(x_i)^T \varphi(x_j) = k(x_i, x_j)
\]

$(x_i, x_j) \in R^F \times R^F \rightarrow k(., .) \in R$

- All positive (semi)-definite functions can be used as kernels
KPCA-Kernel Matrix

• Given a training population of $n$ samples [$x_1, ..., x_n$] we compute the training kernel matrix (also called Gram matrix).

$$K = [\varphi(x_i)^T \varphi(x_j)] = [k(x_i, x_j)]$$

• All the computations are performed via the use of the kernel or the centralized kernel matrix.

$$\bar{K} = (\varphi(x_i) - m^\Phi)^T (\varphi(x_j) - m^\Phi) \quad m^\Phi = \frac{1}{n} \sum_{i=1}^{n} \varphi(x_i)$$
KPCA- Popular Kernels

Gaussian Radial Basis Function (RBF) kernel:

\[
k(x_i, x_j) = e^{-\|x_i - x_j\|^2/r^2}
\]

Polynomial kernel:

\[
k(x_i, x_j) = (x_i^T x_j + b)^n
\]

Hyperbolic Tangent kernel:

\[
k(x_i, x_j) = \tanh(x_i^T x_j + b)
\]
**KPCA-kernel Matrix**

Input space: \[ X = [x_1, \ldots, x_n] \]

Feature space: \[ X^\Phi = [\varphi(x_1), \ldots, \varphi(x_n)] \]

Centralised: \[ \bar{X}^\Phi = [\varphi(x_1) - m^\Phi, \ldots, \varphi(x_n) - m^\Phi] \]
\[ = X^\Phi (I - E) = X^\Phi M, \quad E = \frac{1}{n} 1 \, 1^T \]

Kernel: \[ K = [\varphi(x_i)^T \varphi(x_j)] = [k(x_i, x_j)] = X^\Phi^T X^\Phi \]

Centralised Kernel:
\[ \bar{K} = \left[ \left( \varphi(x_i) - m^\Phi \right)^T \left( \varphi(x_j) - m^\Phi \right) \right] = (I - E)X^\Phi^T X^\Phi (I - E) \]
\[ = (I - E)K(I - E) = K - EK - KE + EKE \]
KPCA-Optimization problem

- KPCA cost function

\[
U^\Phi_o = \arg \max_{U^\Phi} \text{tr}[U^\Phi^T S_t^\Phi U^\Phi] \\
= \arg \max_{U^\Phi} \text{tr}[U^\Phi^T X^\Phi X^\Phi^T U^\Phi]
\]

subject to \( U^\Phi^T U^\Phi = I \)

- The solution is given by the \( d \) eigenvectors that correspond to the \( d \) largest eigenvalues

\[
S_t^\Phi U^\Phi_o = U^\Phi_o \Lambda
\]
KPCA- Computing Principal Components

• Do you see any problem with that?

• How can we compute the eigenvectors of $S_t^\Phi$?
  We do not even know $\phi$!!!!

• Remember our Lemma that links the eigenvectors and eigenvalues of matrices of the form $AA^T$ and $A^TA$

$$\bar{K} = \bar{X}^\Phi^T \bar{X}^\Phi = V \Lambda V^T \quad \text{then} \quad U^\Phi_o = \bar{X}^\Phi V \Lambda^{-\frac{1}{2}}$$

• All computations are performed via the use of $\bar{K}$ (so-called kernel trick)
KPCA- Computing Principal Components

- Still $U \Phi_o = \bar{X} \Phi V \Lambda^{- \frac{1}{2}}$ cannot be analytically computed.

- But we do not want to compute $U \Phi_o$.

- What we want is to compute latent features.

- That is, given a test sample $x_t$ we want to compute $y = U \Phi_o^T \phi(x_t)$ (this can be performed via the kernel trick)
KPCA- Extracting Latent Features

\[ y = U^\Phi_o^T (\varphi(x_t) - m^\Phi) \]
\[ = \Lambda^{-\frac{1}{2}} V^T X^\Phi^T (\varphi(x_t) - m^\Phi) \]
\[ = \Lambda^{-\frac{1}{2}} V^T (I - E) X^\Phi^T \left( \varphi(x_t) - \frac{1}{n} X^\Phi 1 \right) \]
\[ = \Lambda^{-\frac{1}{2}} V^T (I - E) (X^\Phi^T \varphi(x_t) - \frac{1}{n} X^\Phi^T X^\Phi 1) \]
\[ = \Lambda^{-\frac{1}{2}} V^T (I - E) (g(x_t) - \frac{1}{n} K 1) \]

\[ g(x_t) = X^\Phi^T \varphi(x_t) = \begin{bmatrix} \varphi(x_1)^T \varphi(x_t) \\ \cdots \\ \varphi(x_n)^T \varphi(x_t) \end{bmatrix} = \begin{bmatrix} k(x_1, x_t) \\ \cdots \\ k(x_n, x_t) \end{bmatrix} \]
KPCA- Example
Latent Feature Extraction with Neural Networks

Remember the PCA model?

\[ y_i = W^T x_i \]

\[ \tilde{x}_i = WW^T x_i \]
Latent Feature Extraction with Neural Networks
Latent Feature Extraction with Neural Networks

\[ W_1 \rightarrow W_2 \rightarrow W_3 \rightarrow W_4 \rightarrow 30 \]

- **Pretraining**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- **RBM**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- **Encoder**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- **Unrolling**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- **Decoder**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- **Fine-tuning**
  - 2000
  - 1000
  - 500
  - 500
  - 30

- G E Hinton, and R R Salakhutdinov Science 2006;313:504-507
Toolboxes on Component Analysis

Matlab Toolbox for Dimensionality Reduction

http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html

Matlab 2013b has PPCA implemented

http://www.mathworks.co.uk/help/stats/ppca.html
Toolboxes on Component Analysis

ICA toolboxes for image and signal processing:

http://www.bsp.brain.riken.jp/ICALAB/

ICA for EEG Analysis:

http://mialab.mrn.org/software/

FastICA

http://research.ics.aalto.fi/ica/fastica/