Sparse Greedy Spectral Clustering and Kernel PCA

Marie Ouimet, Yoshua Bengio

Université de Montréal,
Informatique et Recherche Opérationnelle

IRIS Machine Learning Workshop, June 9, 2004

Kernel Methods

Global framework:

- You have n points in $\mathbb{R}^p$
- Construct a $n \times n$ Gram matrix M
  $M_{ij} = k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ is a dot products in feature space, a virtual space.
  $\phi : \mathbb{R}^p \mapsto \mathbb{R}^q$, where q can be ∞, so $\phi(x)$ is never computed explicitly.
- Normalize the matrix
- Find the eigen decomposition of the matrix: $Mv_k = \lambda_k v_k$
- You get a representation of the points in $\mathbb{R}^k$, $k < p$, using the k firsts eigen pairs (in decreasing order of the eigen values)
- You can project a new point in that space (generalize) using the same eigen pairs: $f_k(x) = \frac{1}{\lambda_k} \sum_{i=1}^{n} v_{ik} k(x, x_i)$

You get different Kernel methods using different normalizations:
Isomap, KPCA, Spectral Clustering, LLE and MDS.

(Bengio, Paiement, Vincent 2003)

The Problem

- The more data you have, the better is your generalization.
- But computing the eigen decomposition of a $n \times n$ matrix is too expensive if n is large.

What can we do?

The Idea

- Use only a subset of the examples to construct a Gram matrix but take advantage of the information given by the other examples.

  How to select the examples in the Gram matrix?

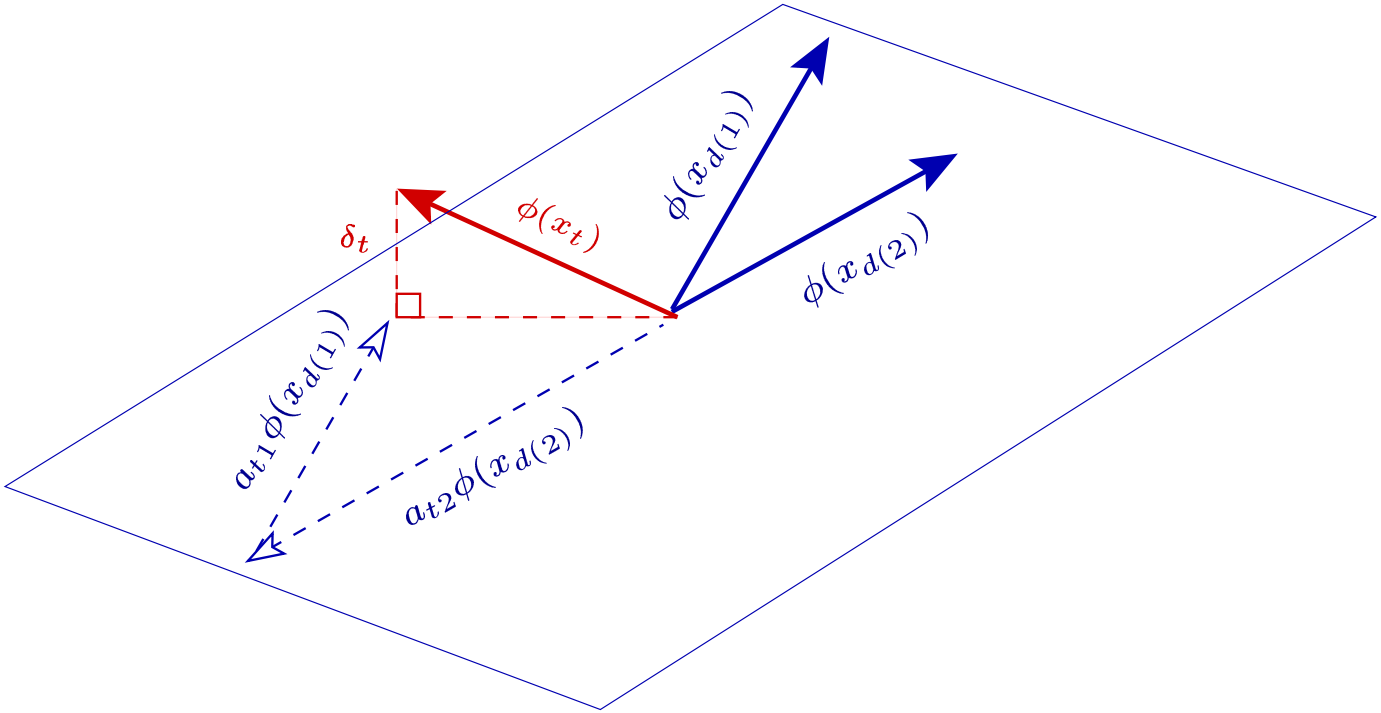
  - Naively: randomly
  - Better: those that are approximately linearly independant in feature space $\phi(x)$

Approximate linear independance

We construct iteratively a dictionary $D = \{x_{d(i)}\}_{i=1}^{m} \subset$ dataset $\{x_i\}_{i=1}^{n}$.

- x_t is added to the dictionary if $\phi(x_t)$ is approximately linearly independant of $\{\phi(x_{d(i)})\}_{i=1}^{m}$
- Otherwise $\exists\ a_t$ such that $\phi(x_t) \approx \sum_{i=1}^{m} a_{ti}\phi(x_{d(i)})$

(Engel, Mannor, Meir 2003)



Approximate linear independance condition

x_t is added to the dictionary if

$$\delta_t = \min_a || \sum_{i=1}^{m_{t-1}} a_{ti} \phi(x_{d(i)}) - \phi(x_t)||^2 > \epsilon$$

under the constraint $\sum_{i=1}^{m_{t-1}} a_{ti} = 1$.

Using the *kernel trick* we find that at optimum:

$$a_t = \hat{a}_t + \frac{\alpha}{\beta} \tilde{M}_{t-1}^{-1} [1, \ldots, 1]^\top$$
$$\delta_t = k(x_t, x_t) - \tilde{k}_{t-1}(x_t)^\top \hat{a}_t + \frac{\alpha^2}{\beta}$$

where

$\hat{a}_t = \tilde{M}_{t-1}^{-1} \tilde{k}_{t-1}(x_t)$,

$\alpha = 1 - \sum_i \hat{a}_{ti}$,

$\beta = \sum_{i,j} (\tilde{M}_{t-1}^{-1})_{ij}$,

ϵ controls the accuracy of the approximation,

$[\tilde{M}_{t-1}]_{ij} = k(x_{d(i)}, x_{d(j)})$,

$[\tilde{k}_{t-1}(x)]_i = k(x_{d(i)}, x)$,

for $i, j = 1, \ldots, m_{t-1}$.

Two cases of update

case 1: If $\delta_t \leq \epsilon$, then $D_t = D_{t-1}$, $m_t = m_{t-1}$ and $\tilde{M}_t = \tilde{M}_{t-1}$.

case 2: If $\delta_t > \epsilon$, then $D_t = D_{t-1} \cup \{x_t\}$, $m_t = m_{t-1} + 1$ and

$$\tilde{M}_t = \begin{bmatrix} \tilde{M}_{t-1} & \tilde{k}_{t-1}(x_t) \\ \tilde{k}_{t-1}(x_t)^\top & k(x_t, x_t) \end{bmatrix}$$

$$\Rightarrow \tilde{M}_t^{-1} = \frac{1}{\delta_t} \begin{bmatrix} \delta_t \tilde{M}_{t-1}^{-1} + \hat{a}_t \hat{a}_t^\top & -\hat{a}_t \\ -\hat{a}_t^\top & 1 \end{bmatrix}$$

When we have enough examples

When the dictionary stops growing we update our a_i coeff. with last $\tilde{M}^{-1}$.

For $x_i \notin D$: $a_i = \tilde{M}^{-1}\tilde{k}(x_i) + \frac{\alpha}{\beta}\tilde{M}^{-1}[1, \ldots, 1]^\top$

For $x_i \in D$: $a_i = (0, \ldots, \underset{\substack{\uparrow \\ i^{\text{th}} \text{ position}}}{1}, \ldots, 0)^\top$

We get $A = \begin{bmatrix} a_1^\top \\ \vdots \\ a_n^\top \end{bmatrix}$.

Find the eigen decomposition of M

In order to get an embedding for our points, we need to compute the eigen decomposition of M.

Our approximation of M is $A\tilde{M}A^\top$.

With the generalized eigen decomposition of $\tilde{M}$ and $A^\top A$ ($2\ m \times m$ matrices) we get an eigen decomposition of $A\tilde{M}A^\top$ ($n \times n$ matrix):

$$\begin{aligned} & \tilde{M}A^\top Av = \lambda v \\ \Rightarrow \quad & A\tilde{M}A^\top Av = \lambda Av \end{aligned}$$

so the eigen values are the same for the two matrix and the k^{th} eigen vector of $A\tilde{M}A^\top$ is Av_k.

Additive and divisive normalizations

Kernel PCA:

To obtain Kernel PCA, we can perform the additive normalization by only modifying A because the approximation is invariant to translation in feature space:

$$\hat{A}_{i\cdot} = A_{i\cdot} - E_x[A_{x\cdot}]$$

Spectral Clustering:

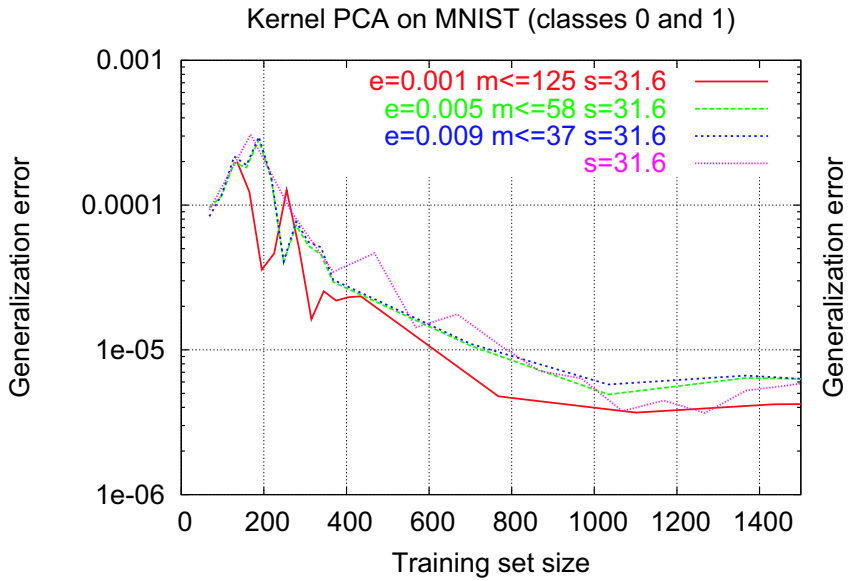
The divisive normalisation is applied before the approximation and the averages are made on a random subset of size m^2 at max:

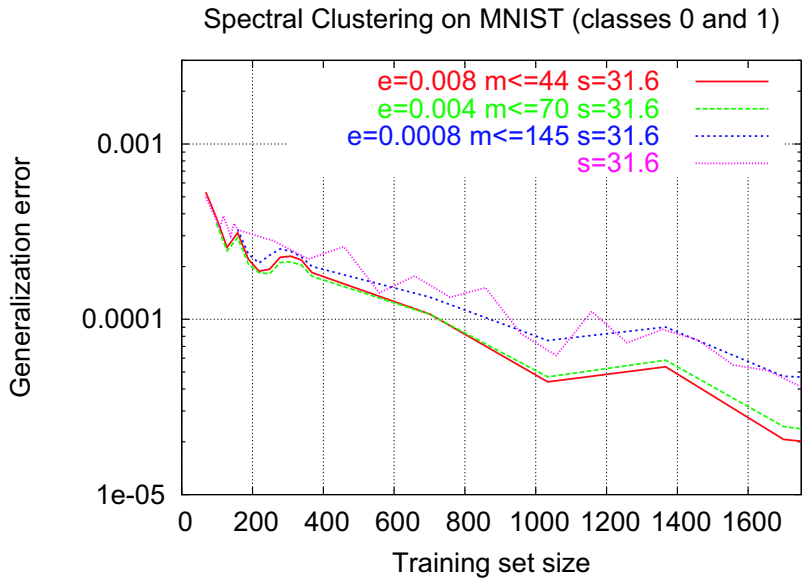
$$\hat{k}(x_i, x_j) = \frac{k(x_i, x_j)}{\sqrt{E_y[k(x_i, y] E_x[k(x, x_j]}}$$

Complexity analysis

- **Original kernel methods**: $O(kn^2)$ at least
  - eigen decomposition of a $n \times n$ matrix: $O(kn^2)$
- **Kernel method using dictionary**: $O(m^2 n)$
  - construction of the dictionary: $O(m^2 n)$
  - update of a_i coefficients: $O(m^2 n)$
  - normalization of A: $O(mn)$
  - generalized eigen decomposition of $\tilde{M} A^\top A$: $O(m^3)$
  - compute Av_i for $i = 1, \ldots, k$: $O(mnk)$

Experiments

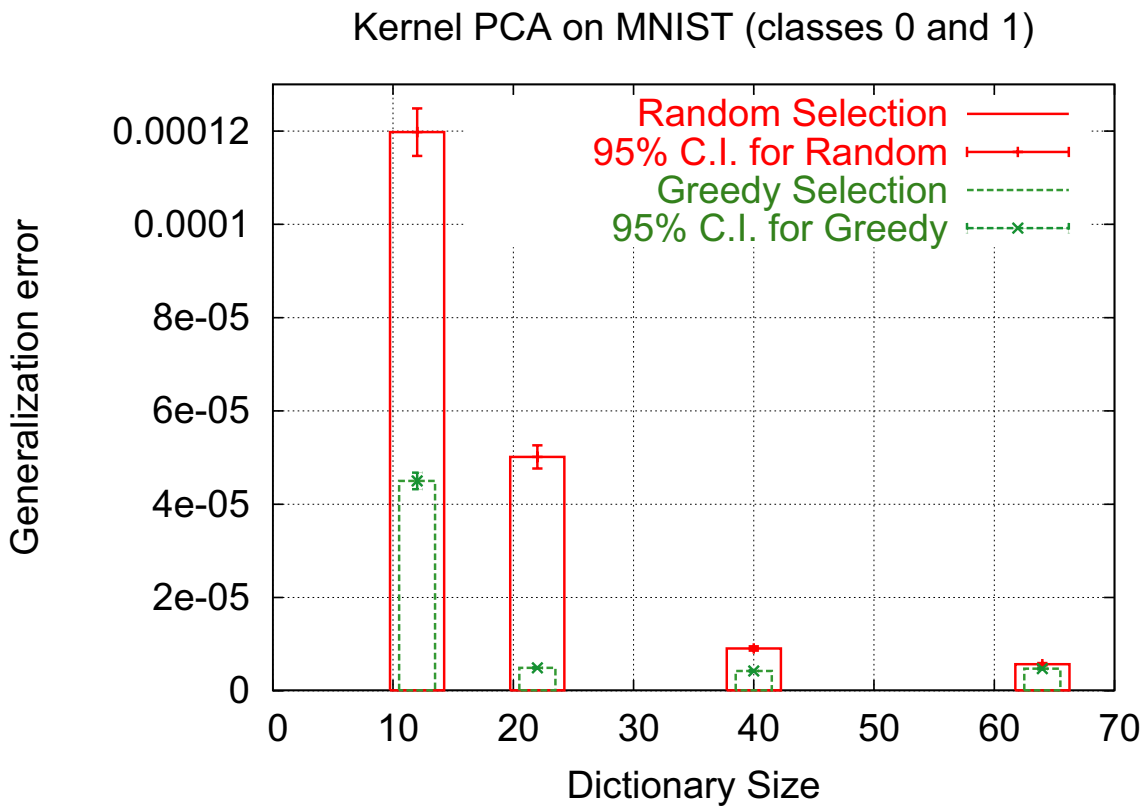




	Without dictionary		With dictionary	
Time KPCA (s)	Max. size	Gen. error $\pm 95\%$ C.I.	Max. size	Gen. error $\pm 95\%$ C.I.
05.71	$n = 250$	$1.48e^{-4} \pm 8.9e^{-6}$	$n = 1300,\ m = 34$	$6.64e^{-6} \pm 2.3e^{-7}$
05.72	$n = 250$	$1.48e^{-4} \pm 8.9e^{-6}$	$n = 1300,\ m = 55$	$6.40e^{-6} \pm 2.1e^{-7}$
06.46	$n = 550$	$8.27e^{-6} \pm 4.0e^{-7}$	$n = 1300,\ m = 126$	$3.99e^{-6} \pm 1.5e^{-7}$
14.02	$n = 1300$	$3.47e^{-6} \pm 1.4e^{-7}$		
Time SC (s)				
05.80	$n = 325$	$2.41e^{-4} \pm 1.2e^{-5}$	$n = 1300,\ m = 41$	$5.16e^{-5} \pm 2.3e^{-6}$
05.95	$n = 400$	$2.37e^{-4} \pm 1.2e^{-5}$	$n = 1300,\ m = 65$	$5.61e^{-5} \pm 2.5e^{-6}$
07.18	$n = 700$	$1.58e^{-4} \pm 7.3e^{-6}$	$n = 1300,\ m = 138$	$8.74e^{-5} \pm 3.8e^{-6}$
14.06	$n = 1300$	$7.93e^{-5} \pm 3.4e^{-6}$		

Experiments

The difference between choosing the members of the dictionary randomly and using the proposed greedy selection.



Conclusion

Classical kernel methods suffers from computational limitations:

- $O(kn^2)$ algorithm
- $O(n^2)$ memory required

The dictionary method is more efficient:

- $O(m^2n)$ algorithm
- $O(m^2)$ memory required

For a fixed time and space, you can handle larger problems using a sparse kernel, so you get better generalization.