

---

# Greedy Spectral Embedding

---

**Marie Ouimet**

Dept. IRO, Université de Montréal  
P.O. Box 6128, Downtown Branch  
Montreal, H3C 3J7, Qc, Canada  
ouimema@iro.umontreal.ca

**Yoshua Bengio**

Dept. IRO, Université de Montréal  
P.O. Box 6128, Downtown Branch  
Montreal, H3C 3J7, Qc, Canada  
bengioy@iro.umontreal.ca

## Abstract

Spectral dimensionality reduction methods and spectral clustering methods require computation of the principal eigenvectors of an  $n \times n$  matrix where  $n$  is the number of examples. Following up on previously proposed techniques to speed-up kernel methods by focusing on a subset of  $m$  examples, we study a greedy selection procedure for this subset, based on the feature-space distance between a candidate example and the span of the previously chosen ones. In the case of kernel PCA or spectral clustering this reduces computation to  $O(m^2n)$ . For the same computational complexity, we can also compute the feature space projection of the non-selected examples on the subspace spanned by the selected examples, to estimate the embedding function based on all the data, which yields considerably better estimation of the embedding function. This algorithm can be formulated in an on-line setting and we can bound the error on the approximation of the Gram matrix.

## 1 Introduction

Many interesting algorithms have been proposed in recent years to perform non-parametric unsupervised learning, either for clustering (spectral clustering) or for manifold learning. Many of these methods can be seen as kernel methods with an adaptive, data-dependent kernel (Bengio et al., 2004). Like other kernel methods, methods such as LLE (Roweis and Saul, 2000), Isomap (Tenenbaum, de Silva and Langford, 2000), kernel Principal Components Analysis (PCA) (Schölkopf, Smola and Müller, 1998), Laplacian Eigenmaps (Belkin and Niyogi, 2003) and spectral clustering (Weiss, 1999; Ng, Jordan and Weiss, 2002), typically require computation of the principal eigenvectors of an  $n \times n$  matrix, where  $n$  is the number of examples.

In this paper we follow-up on previous work to speed-

up kernel methods, such as (Smola and Schölkopf, 2000; Smola and Bartlett, 2001; Williams and Seeger, 2001; Harmeling et al., 2002; Lawrence, Seeger and Herbrich, 2003; Engel, Mannor and Meir, 2004) but focus on spectral methods for **unsupervised learning**. Like in these methods, the main speed-up is obtained by focusing on a subset of the examples, chosen using a greedy selection algorithm. We call the set of selected examples the **dictionary**. Assuming that the kernel is positive semi-definite (which is not always exactly true, but is a good approximation), the above methods are equivalent to a special form of kernel PCA. The criterion used for greedy selection is the distance in feature space between a candidate example and its projection on the subspace spanned by the selected examples. This is a reasonable criterion because the embedding of a new  $x$  will be expressed as a linear combination of the  $K_D(x, x_i)$ , with  $x_i$  a dictionary example and  $K_D$  the data-dependent kernel associated with the particular method. However, if only the selected examples are used to derive the embedding (i.e. the principal eigenvectors of the feature space covariance matrix), then the resulting embedding will be biased, since the dictionary examples will tend to be distributed more uniformly than the original data (to better cover more directions in feature space). In addition, this projection of the non-dictionary examples can be used to enrich the estimation of the feature space covariance matrix. The resulting algorithm is  $O(m^2n)$ , where  $m$  is the dictionary size, and requires  $O(m^2)$  memory. We show how using a generalized eigen-decomposition algorithm it is possible to take advantage of the sparseness of the kernel. Finally, we also show an efficient way to obtain the kernel normalization required in kernel PCA (additive normalization) and in spectral clustering (divisive normalization).

In section 3 we give more justification as well as the details of this algorithm. We put it in perspective of previously proposed methods in section 4. In section 5 we show with several experiments that the greedy approximation works well and works better than (a) the same algorithm with a randomly selected dictionary (but all the data to estimate the eigenvectors), and (b) using only  $m$  random points to estimate the eigenvectors.

## 2 Spectral Embedding Algorithms and Motivation

Let  $D = \{x_1, \dots, x_n\}$  represent a training set, with  $x_i$  a training example. Spectral embedding algorithms (Schölkopf, Smola and Müller, 1998; Weiss, 1999; Roweis and Saul, 2000; Tenenbaum, de Silva and Langford, 2000; Ng, Jordan and Weiss, 2002; Belkin and Niyogi, 2003) provide a vector-valued coordinate for each training example, which would ideally correspond to its coordinate on a manifold near which the data lie. As discussed in (Bengio et al., 2004), spectral embedding methods can generalize the training set embedding to a new example  $x$  through the Nyström formula

$$f_k(x) = \frac{\sqrt{n}}{\lambda_k} \sum_{i=1}^n v_{ik} K_D(x, x_i) \quad (1)$$

for the  $k$ -th embedding coordinate, where  $K_D$  is a kernel that is defined using  $D$  (hence called data-dependent), and  $(v_k, \lambda_k)$  is the  $k$ -th (eigenvector, eigenvalue) pair of the matrix  $M$  with entries  $M_{ij} = K_D(x_i, x_j)$ . Note that this formula reduces to  $f_k(x_i) = \sqrt{n} v_{ik}$  for training examples. Depending on the choice of algorithm, the embedding can be further scaled separately in each dimension (e.g. by a factor  $\sqrt{\lambda_k}$  for kernel PCA, metric multidimensional scaling (MDS), Isomap, and spectral clustering, and by a factor  $\sqrt{n}$  for LLE). Note that  $K_D$  is guaranteed to be positive semi-definite for some of these methods (e.g. LLE, kernel PCA, spectral clustering, Laplacian Eigenmaps) but not for others (e.g. Isomap, MDS) but it is usually close to positive semi-definite (e.g. for Isomap the kernel converges to a positive semi-definite one as  $n \rightarrow \infty$ ).

Assuming  $K_D$  positive semi-definite, there exists a “feature space”  $\phi(x)$  in which  $K_D$  is a dot product  $K_D(x, y) = \phi(x) \cdot \phi(y)$ . Eq. 1 therefore shows that the function of interest (the embedding of  $x$ ) can be written as a dot product between  $\phi(x)$  and a vector which is a linear combination of the feature space vectors  $\phi(x_i)$  associated with the training examples:

$$f_k(x) = \phi(x) \cdot \left( \frac{\sqrt{n}}{\lambda_k} \sum_{i=1}^n v_{ik} \phi(x_i) \right). \quad (2)$$

If we are going to work with a subset of the examples (the dictionary) to define the embedding, we can reduce the above linear combination to one over dictionary examples. There is a simple way to use a dictionary to reduce computation. We first compute the Gram matrix of the dictionary examples and its principal eigenvectors. Then we use the Nyström formula as above to predict the embedding of the other examples. This is essentially the basis for the “Landmark Isomap” algorithm (de Silva and Tenenbaum, 2003), as shown in (Bengio et al., 2004). This is one of the methods that we will evaluate experimentally, and compare to the proposed algorithm, on a Gaussian kernel with additive normalization (corresponding to kernel PCA

or metric MDS) and one with divisive normalization (corresponding to spectral clustering or to Laplacian Eigenmaps with Gaussian kernel).

In the two cases that we study here, the data-dependence of the kernel is due to this normalization. The data-dependent kernel  $K_D$  is obtained from a data-independent kernel  $\tilde{K}$  as follows. For additive normalization (kernel PCA, metric MDS) we have

$$K_D(x, y) = \tilde{K}(x, y) - E_v[\tilde{K}(v, y)] - E_w[\tilde{K}(x, w)] + E_{v,w}[\tilde{K}(v, w)] \quad (3)$$

and for divisive normalization (for a positive kernel) we have

$$K_D(x, y) = \frac{\tilde{K}(x, y)}{\sqrt{E_v[\tilde{K}(v, y)] E_w[\tilde{K}(x, w)]}}. \quad (4)$$

In both cases this normalization requires estimation of  $E_x[\tilde{K}(x, y)]$  which is normally done with the training set average of the kernel over one of its arguments:  $E_x[\tilde{K}(x_i, x)] = \frac{1}{n} \sum_{j=1}^n \tilde{K}(x_i, x_j)$ . We discuss below how to avoid this computation which would make the whole algorithm running time  $O(n^2)$ .

## 3 Proposed Algorithm

Let  $C$  denote the covariance matrix of the examples in feature space. One interpretation of the eigenvectors of the Gram matrix is that they correspond to the eigenvectors of  $C$ , which can be written as  $\frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^n v_{ik} \phi(x_i)$ . Again, this interpretation is only valid for positive semi-definite kernels, or when working in the subspace associated with non-negative eigenvalues (i.e. with the projection of the  $\phi(x)$  on that subspace).

Let  $\mathcal{P}$  be the subspace spanned by the dictionary examples in feature space (or the restriction to non-negative eigenvalues). If  $\phi(x_i)$  is not far from its projection on  $\mathcal{P}$ , i.e. it is well approximated by a linear combination of the examples spanning  $\mathcal{P}$ , then we can replace it by this projection without making a big error. Proposition 1 below formalizes this idea, and Figure 1 shows a geometric interpretation of the projection of  $\phi(x_i)$  on the span of the dictionary examples in feature space. Therefore, instead of using a random subset of examples for the dictionary and only using the dictionary to estimate the eigenvectors of  $C$ , we propose (1) to select the dictionary examples according to their distance to  $\mathcal{P}$ , and (2) to use the dictionary examples as well as the projection of the out-of-dictionary examples to estimate the principal eigenvectors of  $C$ , thus giving rise to a more precise estimate, almost as good as using the whole data set, according to our experiments. It turns out that (2) comes for free (i.e. for the same cost) once we have accepted to do the computations for (1). Another important consideration is that the selected examples may have statistics that are different from the original examples. For example, if they are chosen as a good “cover” of the training examples, the relative density of dictionary examples will be smaller

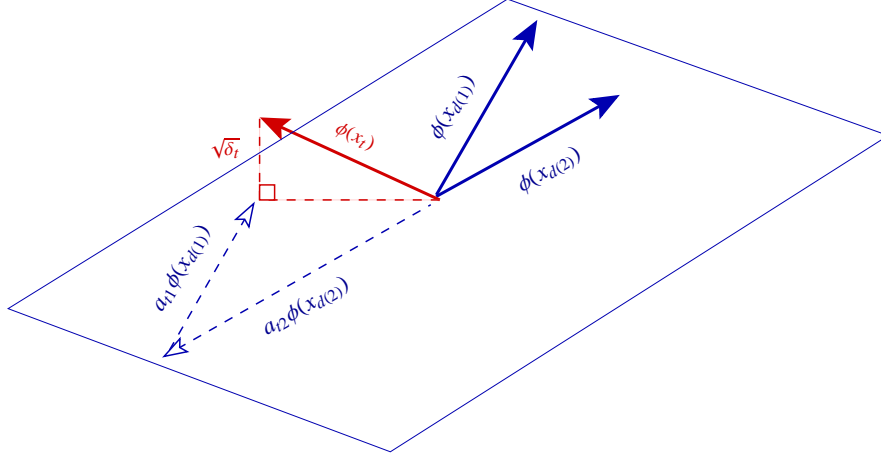


Figure 1: Out-of-dictionary example  $\phi(x_t)$  is approximated in feature space by a linear combination of the dictionary examples  $\phi(x_{d(i)})$ ,  $i = 1 \dots m$ .

---

**Algorithm 1** Greedy Spectral Embedding Algorithm.

**Arguments:** randomly ordered data set  $D$ , tolerance  $\epsilon$ , embedding dimension  $p$ .

---

- 1:  $n = |D|$ , initialize  $\mathcal{D} = \{x_1\}$ ,  $M_1 = (K_D(x_1, x_1))$ ,  
 $M_1^{-1} = (K_D(x_1, x_1))^{-1}$ .
  - 2: **for**  $t = 2$  **to**  $n$  **do**
  - 3:   **for**  $i = 1$  **to**  $m_{t-1}$  **do**
  - 4:     compute  $(k_t(x_t))_i = K_D(x_t, x_{d(i)})$
  - 5:   **end for**
  - 6:   compute matrix-vector product  $\hat{a}_t = M_{t-1}^{-1} k_t(x_t)$
  - 7:   compute  $\alpha = 1 - \sum_i \hat{a}_{ti}$  and  $\beta = \sum_{i,j} (M_{t-1}^{-1})_{ij}$ .
  - 8:   compute projection weights  
 $a_t = \hat{a}_t + \frac{\alpha}{\beta} M_{t-1}^{-1} [1, \dots, 1]'$
  - 9:   compute projection error  
 $\delta_t = K_D(x_t, x_t) - k_{t-1}(x_t)' \hat{a}_t + \frac{\alpha^2}{\beta}$ .
  - 10:   **if**  $\delta_t > \epsilon$  **then**
  - 11:      $m_t = m_{t-1} + 1$ ,  $\mathcal{D} \leftarrow \mathcal{D} \cup \{x_t\}$
  - 12:      $M_t^{-1}$  is set according to eq. 9.
  - 13:   **else**
  - 14:      $m_t = m_{t-1}$ ,  $M_t^{-1} = M_{t-1}^{-1}$
  - 15:   **end if**
  - 16: **end for**
  - 17: **for**  $i = 1$  **to**  $m_n$  **do**
  - 18:   **for**  $j = 1$  **to**  $m_n$  **do**
  - 19:      $B_{ij} = A_{ii} A_{jj}$  where  $A_{ii}$  recovered from cache or  
 computed with eq. 10.
  - 20:   **end for**
  - 21: **end for**
  - 22: Find the  $p$  principal eigenvectors  $u_k$  and eigenvalues  $\lambda_k$  of the generalized eigen-problem with left matrix  $B = A'A$  and right matrix  $M_n^{-1}$ .
  - 23: Embeddings of all examples are given by  $v_k = Au_k$  for  $k$ -th coordinate.
  - 24: The embedding of a test example  $x$  is given by  $\frac{f_k(x)}{\sqrt{n}}$ , with  $f_k(x)$  as in eq. 1.
- 

than the true data density in areas of high density of the original examples. You can observe that phenomenon in the center image of figure 2: the region corresponding to digit 1 is much more dense and contains only 4 dictionary points. Since the directions with high variance are different for the dictionary points, using only this selected subset to form a Gram matrix without projecting the other points would give a completely different embedding.

Computing the distance between  $\phi(x_t)$  and the span of the dictionary examples in feature space can be reduced to computations with the kernel, thanks to the kernel trick:

$$\delta_t \doteq \min_{a_t} \|\phi(x_t) - \sum_{i=1}^{m_t} a_{it} \phi(x_{d(i)})\|^2 \quad (5)$$

where  $d(i)$  is the index of the  $i$ -th dictionary example, and  $m_t$  is the size of the dictionary after seeing  $t$  examples. We found better results with the constraint  $\sum_i a_{it} = 1$ . This constraint has the effect of making the solution independent to translations in feature space, since  $(\phi(x_t) - b) - \sum_{i=1}^{m_t} a_{it} (\phi(x_{d(i)}) - b) = \phi(x_t) - \sum_{i=1}^{m_t} a_{it} \phi(x_{d(i)})$ . In this paper we consider an application of these ideas to an **online** setting (or to reduce memory requirements, at most 2 passes through the data). Let  $a_t = (a_{1t}, \dots, a_{m_t t})$ , let  $M_{t-1}$  be the Gram matrix of the dictionary examples after seeing  $t-1$  examples, and let  $k_{t-1}(x)$  be the vector with  $m_t$  elements  $K_D(x, x_{d(i)})$ . The solution for  $a_t$  without constraint is

$$\hat{a}_t \doteq M_{t-1}^{-1} k_{t-1}(x_t). \quad (6)$$

The solution with constraint can then be obtained as follows:

$$a_t = \hat{a}_t + \frac{\alpha}{\beta} M_{t-1}^{-1} [1, \dots, 1]' \quad (7)$$

where  $'$  denotes transposition,  $\alpha = 1 - \sum_i \hat{a}_{ti}$  and  $\beta = \sum_{i,j} (M_{t-1}^{-1})_{ij}$ . The corresponding value of the projection distance is then obtained as follows:

$$\delta_t = \hat{\delta}_t + \frac{\alpha^2}{\beta}, \quad (8)$$

where  $\hat{\delta}_t = K_D(x_t, x_t) - k_{t-1}(x_t)' \hat{a}_t$ . We introduce the parameter  $\epsilon$  that controls the accuracy of the approximation.  $x_t$  is added to the dictionary if  $\delta_t > \epsilon$ , which means that  $x_t$  is far from its projection on  $\mathcal{P}$  in feature space. When a point is added to the dictionary,  $M_t^{-1}$  needs to be computed. To do so efficiently (in  $O(m_t^2)$  computations), we can take advantage of our knowledge of  $M_{t-1}^{-1}$  and the matrix inversion lemma:

$$M_t^{-1} = \frac{1}{\hat{\delta}_t} \begin{pmatrix} \hat{\delta}_t M_{t-1}^{-1} + \hat{a}_t \hat{a}_t' & -\hat{a}_t \\ -\hat{a}_t' & 1 \end{pmatrix} \quad (9)$$

following the recursive update approach in (Engel, Mannor and Meir, 2004) for kernel regression.

After we have selected the dictionary, we want to compute the projection of all the examples on its span. This will be different from the projection computed online in eq. 7 because it will use the final dictionary and  $M_n^{-1}$  instead of  $M_{t-1}^{-1}$ :

$$\begin{aligned} \hat{A}_t &\doteq M_n^{-1} k_n(x_t) \\ A_t &\doteq \hat{A}_t + \frac{1 - \sum_i \hat{A}_{ti}}{\sum_{i,j} (M_n^{-1})_{ij}} M_n^{-1} [1, \dots, 1]' \end{aligned} \quad (10)$$

Let  $A$  denote the matrix with rows  $A_t$ . It can be shown that the complete Gram matrix is approximated by  $AM_n A'$ . We can bound the error on each entry of the Gram matrix by the same parameter we used to select dictionary examples:

### Proposition 1

For all  $x_t, x_u \in D$ ,  $|K_D(x_t, x_u) - (AM_n A')_{tu}| \leq \epsilon$ .

**Proof:** Let  $r_t = \phi(x_t) - \sum_{i=1}^{m_n} a_{it} \phi(x_{d(i)})$ . Notice that if  $x_t \in \mathcal{D}$ , then  $\|r_t\| = 0$ , otherwise,  $\|r_t\| = \sqrt{\delta_t}$  and  $r_t$  is orthogonal to the span of  $\phi(x_{d(i)})$ ,  $i = 1, \dots, m_n$ . Then we have

$$\begin{aligned} K_D(x_t, x_u) &= \phi(x_t)' \cdot \phi(x_u) \\ &= (r_t + \sum_{i=1}^{m_n} a_{it} \phi(x_{d(i)}))' \cdot (r_u + \sum_{i=1}^{m_n} a_{iu} \phi(x_{d(i)})) \\ &= r_t' \cdot r_u + r_t' \cdot \sum_{i=1}^{m_n} a_{iu} \phi(x_{d(i)}) + r_u \cdot \sum_{i=1}^{m_n} a_{it} \phi(x_{d(i)}) \\ &\quad + (\sum_{i=1}^{m_n} a_{it} \phi(x_{d(i)}))' \cdot (\sum_{i=1}^{m_n} a_{iu} \phi(x_{d(i)})) \\ &= r_t' \cdot r_u + (\sum_{i=1}^{m_n} a_{it} \phi(x_{d(i)}))' \cdot (\sum_{i=1}^{m_n} a_{iu} \phi(x_{d(i)})) \\ &= r_t' \cdot r_u + (AM_n A')_{tu}. \end{aligned}$$

So we have

$$\begin{aligned} |K_D(x_t, x_u) - (AM_n A')_{tu}| \\ = |r_t' \cdot r_u| \leq \sqrt{\delta_t} \sqrt{\delta_u} \leq \epsilon. \end{aligned}$$

The eigenvectors of  $AM_n A'$  would give us the embedding of all the training examples, but this is an  $n \times n$  matrix. We want to take advantage of its factorization. Note that if  $u_k$  is an eigenvector of  $M_n A' A$  then  $v_k = A u_k$  is an eigenvector of  $AM_n A'$  with the same eigen values. Unfortunately,  $M_n A' A$  is not symmetric. But we can still compute the eigenvectors efficiently in time  $O(m^3)$ , by solving the  $m \times m$  generalized eigen-system  $Lu = \lambda Ru$  with left matrix  $L = A' A$  and right matrix  $R = M_n^{-1}$  (which we already have from our recursive updates). Finally this gives rise to algorithm 1.

### 3.1 Computational Cost and Memory Usage

The expensive steps of the algorithm 1 are step 6 ( $O(m_{t-1}^2)$  per step,  $O(nm^2)$  overall), step 12 ( $O(m_{t-1}^2)$  per step,  $O(nm^2)$  overall), step 19 ( $O(nm^2)$ ), step 22 ( $O(m^3)$ ), and step 23 ( $O(pnm)$ ). The memory usage is  $O(m^2)$  to store  $M_t$  (which becomes  $M_n$  at the end), if the  $a_{it}$  are recomputed in step 23, or  $O(nm)$  to store  $A$  if they are not recomputed (depending on how large  $n$  is and how much memory is available). Hence time is  $O(nm^2)$  and memory either  $O(m^2)$  or  $O(nm)$  (time-memory trade-off only in the constant factor for time).

**Algorithm 2** Constructs a dictionary of specified size  $m_n$  with an almost minimal level of error.

**Arguments:** randomly ordered data set  $D$ , wanted subset size  $m_n$ .

- 
- 1: Initialize  $\mathcal{D} = \{x_1\}$ ,  $M_t = (K_D(x_1, x_1))^{-1}$ ,  $M_t^{-1} = (K_D(x_1, x_1))^{-1}$ .
  - 2: Set  $\epsilon^+ = 0$  or something for  $m_n^+ > m_n$
  - 3: Set  $\epsilon^- = 1$  or something for  $m_n^- < m_n$
  - 4: Check that  $m_n^- < m_n$  by running steps 2 to 16 of algo. 1 using  $\epsilon^-, \mathcal{D}, M_t, M_t^{-1}$  and data set  $D - \mathcal{D}$ , stopping if  $m_t^- > m_n$ . Store the obtained dictionary and corresponding matrices in  $\mathcal{D}^-, M_t^-, M_t^{-1}$ .
  - 5: **if**  $m_t^- = m_n$  **then**
  - 6:     **return**  $\mathcal{D}^-, M_t^-, M_t^{-1}$
  - 7: **else if**  $m_t^- > m_n$  **then**
  - 8:      $\epsilon^+ = \epsilon^-$
  - 9:      $\epsilon^- = 10\epsilon^-$
  - 10:    **goto** step 4
  - 11: **else**
  - 12:     Set  $\mathcal{D} = \mathcal{D}^-, M_t = M_t^-, M_t^{-1} = M_t^{-1}$
  - 13: **end if**
  - 14: **for**  $i = 1$  **to** 40 **do**
  - 15:     Set  $\epsilon' = \epsilon^+ + (\epsilon^- - \epsilon^+)/2$
  - 16:     Run steps 2 to 16 of algo. 1 using  $\epsilon', \mathcal{D}, M_t, M_t^{-1}$  and data set  $D - \mathcal{D}$ , stopping if  $m_t' > m_n$ . Store the obtained dictionary and corresponding matrices in  $\mathcal{D}', M_t', M_t^{-1}$ .
  - 17:     **if**  $m_t' = m_n$  **then**
  - 18:         **return**  $\mathcal{D}', M_t', M_t^{-1}$
  - 19:     **else if**  $m_t' > m_n$  **then**
  - 20:         Set  $\epsilon^+ = \epsilon'$
  - 21:     **else**
  - 22:         Set  $\epsilon^- = \epsilon', \mathcal{D} = \mathcal{D}', M_t = M_t', M_t^{-1} = M_t^{-1}$
  - 23:     **end if**
  - 24: **end for**
  - 25: Run steps 2 to 16 of algo. 1 using  $\epsilon^+, \mathcal{D}, M_t, M_t^{-1}$  and data set  $D - \mathcal{D}$ , stopping when  $m_t^+ = m_n$ . Store the obtained dictionary and corresponding matrices in  $\mathcal{D}^+, M_t^+, M_t^{-1}$ .
  - 26: **return**  $\mathcal{D}^+, M_t^+, M_t^{-1}$
- 

Although the goal was to get an online algorithm, it is also possible to formulate the method in a setting more similar

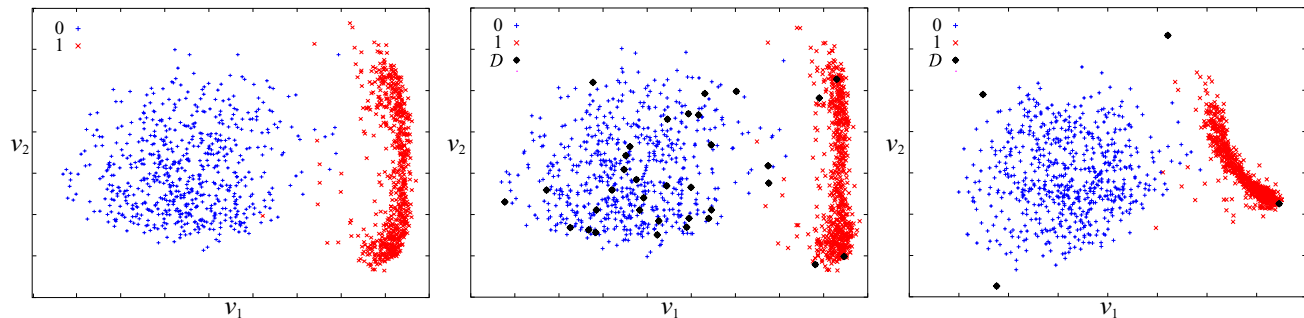


Figure 2: First two dimensions of the embeddings obtained with kernel PCA on classes 0 (+) and 1 (x) of the MNIST data set ( $\sigma = 31.6$ ,  $n = 1300$ ). The dictionary points are noted by  $\blacklozenge$ . The image on the left is the embedding obtained with the full Gram matrix. With 34 points in the dictionary, we obtain an almost identical embedding. Even with only 4 points, the embedding is reasonable and similar.

to previously proposed methods where instead of specifying the accuracy parameter  $\epsilon$ , we specify the subset size  $m$ . To do so, one needs to first search for a good value of  $\epsilon$  before applying algorithm 1. This can also be done in  $O(nm^2)$  operations and  $O(m^2)$  in memory usage. Algorithm 2 constructs a dictionary of the wanted size with an almost minimal error level. Beginning with an  $\epsilon$  that gives a too small dictionary and one giving a too large dictionary, it does a binary search for the right  $\epsilon$  while always keeping small dictionaries and only adding points to it. It first adds to the dictionary the points that are the farthest from  $\mathcal{P}$  and gradually adds points that are closer and closer. This ensures that the error level will be near optimal. One gets the desired embedding by running steps 17 to 24 of algorithm 1 with the obtained dictionary and corresponding matrices.

### 3.2 Additive and Divisive Normalizations

We have not discussed how to perform additive or divisive normalization yet. If we compute the usual complete training set averages to estimate  $E_x[\tilde{K}(x, y)]$ , then the whole algorithm becomes  $O(n^2)$  because we have to compute the values of the data-independent kernel  $\tilde{K}(x_i, x_j)$  for all data pairs. There are two solutions to this problem, which gave similar results in our experiments. One is to use a random subset of the examples (or the dictionary examples) to estimate these averages. If we average over less than  $m^2$  examples the overall algorithm would remain  $O(nm^2)$ . The other solution is to compute the exact normalization associated with the implicit estimated Gram matrix  $AM_nA'$ .

#### Proposition 2

Additive normalization of  $AM_nA'$  can be obtained by using instead of  $AM_nA'$  the Gram matrix  $\tilde{A}M_n\tilde{A}'$  with  $\tilde{A} = A - B$ , and  $B$  the matrix whose rows are all identical and equal to the average row of  $A$ ,  $E_i[A_i]$ . Similarly, divisive normalization can be obtained by using the Gram matrix  $\tilde{A}M_n\tilde{A}'$  with  $\tilde{A}_i = \frac{A_i}{\sqrt{nA_i(M_nE_i[A_i])}}$ .

The proof is straightforward comparing  $\tilde{A}_iM_n\tilde{A}_j'$  to  $K_D(x_i, x_j)$  in eq. 3 and 4. Using this proposition, the al-

gorithm remains the same except that the matrix  $A$  is replaced by  $\tilde{A}$  in the last steps (22 and 23). Under constraint  $\sum_i a_{it} = 1$ , additive normalization before or after the computation of  $A$  made no empirical difference, since the approximation is invariant to a translation in feature space. The only difference is the number of points over which the averages are made. This is not the case for the divisive normalization, where if we want to compute the approximation in the feature space induced by the normalized kernel, the normalization should be applied before, using a subset of the examples. For this normalization, the constraint  $\sum_i a_{it} = 1$  does not help and could be removed, but it did not make a measurable difference in the experiments.

## 4 Related work

Other sparse greedy kernel methods in  $O(m^2n)$  have been proposed. (Smola and Bartlett, 2001) and (Lawrence, Seeger and Herbrich, 2003) present algorithms for greedy Gaussian processes. The main difference with our method is that they do  $m$  passes on the whole data set (or a random subset of it to reduce computation) and each time they select the example that improves the most a global criterion. They also use a recursive update for inverting matrices. In (Smola and Schölkopf, 2000) the setting is very general, aiming to approximate a matrix  $K$  by  $\tilde{K}$ , a lower rank matrix, by minimizing the Frobenius norm of the residual  $K - \tilde{K}$ . This is again done with  $m$  passes to select the best basis function among a random subset of all the  $n - m_t$  function. Our algorithm is different from these methods by requiring at most 2 passes on the data set. Instead, the approximation we use is close to the approach presented in (Engel, Mannor and Meir, 2004) where it was used in a sequential and supervised setting to perform Kernel RLS. (Williams and Seeger, 2001) suggests to compute the eigen-decomposition on a subset of  $m$  examples and then use the Nyström formula to get an approximation of the eigen-decomposition for the  $n - m$  other points. They argue that their method, although less precise than (Smola and Schölkopf, 2000), is much faster. In our experiments, we will compare our greedy technique to this

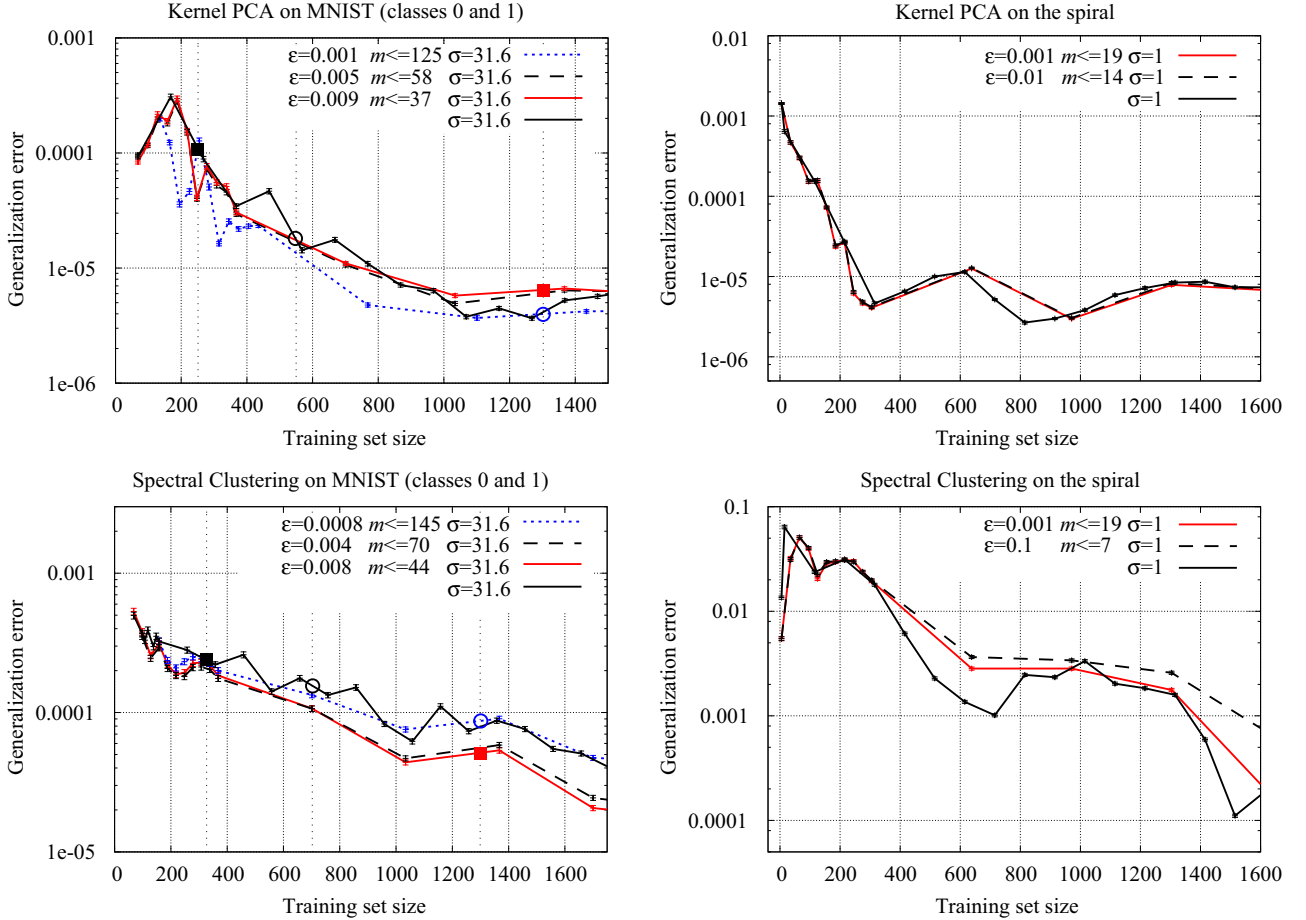


Figure 3: Even for really small dictionary sizes, the generalization performance is comparable to the one obtained with the Gram matrix of the whole training set. Curves corresponding to the dictionary methods are indexed with  $\sigma$ , the standard deviation of the Gaussian kernel,  $\epsilon$ , the accuracy parameter and  $m$ , the resulting maximum number of points in the dictionary (obtained for the maximum  $n$ ). The curves corresponding to non-dictionary methods are indexed with  $\sigma$  only. There are 3 principal components for MNIST and 2 for the spiral. The individual experiments associated with the symbols  $\blacksquare$  and  $\blacksquare$  have the same running time, but the generalization performance is far better for the dictionary method. We observe the same phenomenon for symbols  $\circ$  and  $\circ$  corresponding to a longer running time. More details on the relationship between running time and generalization error are given in table 1.

Nyström method showing that for the same computational time we get better performance. Another related method is (Harmeling et al., 2002) in which we search for the largest random subset of examples for which the Gram matrix is of full rank and project all the examples on this subset. This technique is more computationally expensive than the greedy selection since it requires to do several matrix decompositions to compute the rank. We will also see in the experimental section that random subset selection leads to larger subsets for the same error level.

## 5 Experimental Evaluation

The new algorithm was evaluated on two data sets: a 2-D artificially generated spiral, and images of the digits 0 and 1 from the MNIST data set (scaled down from  $28 \times 28$  to  $14 \times 14$ ). Two embedding methods were compared and

evaluated out-of-sample using the Nyström formula: spectral clustering and kernel PCA, both with the Gaussian kernel, with different values of standard deviation (shown in Figure 3). An example of the embeddings obtained with the dictionary method on the MNIST data set is given in figure 2.

The goals of the experiments were (1) to verify that the algorithm worked (in the sense of giving an embedding close to the one obtained when training with the whole data set) and (2) to verify that it worked better than (a) the same algorithm with randomly selected dictionary (Harmeling et al., 2002) and (b) training on a small subset and generalization with the Nyström formula (Williams and Seeger, 2001; Bengio et al., 2004). To compare the embeddings given by these different methods, we will need a reference embedding. We first divide our data set in three parts:  $D_1$ ,

Table 1: Comparison between generalization errors for the dictionary and non-dictionary methods at a fixed running time. The errors and the  $n$  corresponds to the ones in Figure 3 for MNIST. Some of these results are plotted in Figure 3 with square and circle symbols. Similar results are obtained for a fixed memory space.

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	$n = 1300$	$7.93e^{-5} \pm 3.4e^{-6}$		

$D_2, D_3$ , where  $D_2$  and  $D_3$  are large. We compute a reference embedding by applying standard kernel PCA or spectral clustering to  $D_2 \cup D_3$  and keeping the part corresponding to  $D_2$ . We then train the methods we want to compare on  $D_1$ . Using the obtained eigenvectors, we compute the out-of-sample embedding for every element of the test set  $D_2$  with the Nyström formula. We will compare this embedding to the reference embedding of  $D_2$  by aligning them with a simple linear regression (that maps each example’s coordinate in one embedding with the same example’s coordinate in the other embedding). In our experiments, the reported generalization error is the average squared difference between the corresponding points of these aligned embeddings. We also show the 95% confidence intervals associated to the standard errors of these averages. For the spiral data, we used sets of sizes  $|D_1| \leq 3333$ ,  $|D_2| = 3330$  and  $|D_3| = 3332$ . For the MNIST data,  $|D_1| \leq 3365$ ,  $|D_2| = 3267$  and  $|D_3| = 3332$ .

In the experiments shown in Figure 3, we verify that the greedy algorithm works about as well as when we use the full Gram matrix. On the horizontal axis, we vary the size of the training set  $D_1$  and on the vertical axis, we show the out-of-sample errors obtained with the Nyström formula. The dictionary method (curves indexed with  $\epsilon, m, \sigma$ ) is trained with fixed values of  $\epsilon$ , yielding different subset sizes as the training set grows. The largest subset size obtained is shown as “ $m \leq \dots$ ”. To compare, we compute the eigenvectors of the full Gram matrix corresponding to same training set (ordinary kernel PCA and spectral clustering). The corresponding generalization errors are reported by the curves indexed with  $\sigma$  only (full black curves). The main feature to note is that for a training set of size  $n$ , the results with the dictionary of size  $m \ll n$  are very close to the results using ordinary training with all the examples, i.e. the greedy algorithm generalizes about as well as full training. The other important conclusion from this figure is that the dictionary method works much better than the simple Nyström method with eigenvectors estimated on a random dictionary (compare for example the error of the full Gram

matrix method trained with a dataset of size  $n = 125$  with the dictionary method for  $n = 1400$  and  $m = 125$  for kernel PCA on MNIST). To be more fair, one can compare the Nyström method to the dictionary method at equal running time instead of equal subset size. Table 1 shows that for the same running time, the dictionary method yields quite smaller generalization error. In figure 3 and in table 1 you can notice a slight increase in error as the dictionary grows for spectral clustering on the MNIST data set, but as expected, the curve for the larger dictionary is closer to the one for the whole dataset. This increase in error is probably due to overfitting; using a small dictionary is a way to regularize. Finally, in Figure 4 we compare the proposed selection algorithm vs a random selection procedure. In both cases we project the other points on the span of the subset in feature space. We also show the performance of the Nyström technique with the eigen-decomposition made on a random subset of points, ignoring the other points of the training set. In these experiments we used a training set of size 3365 and show the test set error for different values of subset size  $m$ . The first thing we notice is that projecting the other examples makes a huge difference. For this data set, we need a subset of size about 900 for the Nyström technique to achieve the performance we get with less than 100 points in the subset if we project the 3265 other points. Although the greedy selection and the random selection behave the same for subsets of size 65 and more, this is not the case for smaller sizes. The greedy selection strategy reduces generalization error significantly, the more so for smaller dictionaries, as expected. Note that for this problem a greedy dictionary of size 44 gave performances similar to when the complete Gram matrix was used, as shown in figure 3, but figure 4 suggests that a dictionary of size about 22 would have been enough. For these dictionary sizes, the performance of the random dictionary method is not as good and consequently, an approach like (Harmeling et al., 2002) will need to pick a subset of size at least 65 to reach about the same error level.

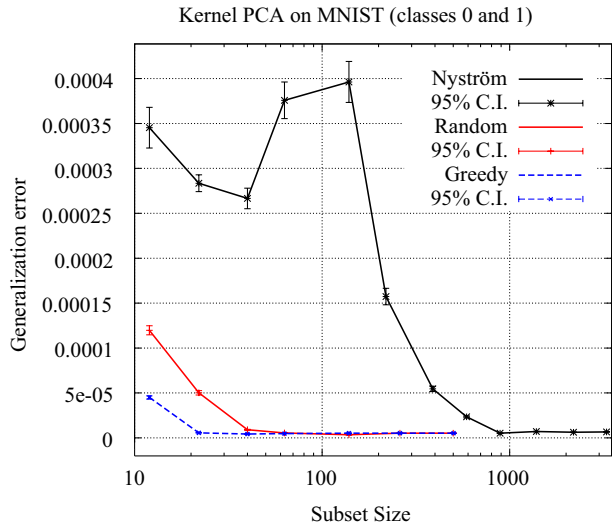


Figure 4: Comparison of the dictionary method vs the Nyström method where the eigen-decomposition is made only on a random subsets of the training set. We show two subset selection strategies for the dictionary method: the proposed greedy selection and a simple random selection. These experiments were performed with a training set of size 3365, with 3 principal components.

## 6 Conclusion

Spectral embedding methods are useful for manifold learning (non-linear dimensionality reduction) and clustering (spectral clustering) but require an expensive  $O(n^3)$  operation with  $O(n^2)$  memory requirement, with  $n$  the number of examples. In this paper we propose an efficient algorithm that yields almost as good results and reduces computation to  $O(nm^2)$  and memory to  $O(m^2)$  where  $m$  is the size of a small subset of selected examples (the dictionary). The algorithm selects examples greedily and sequentially based on their distance to the span of the already selected ones, in the kernel feature space. It then uses the projection of all  $n$  examples on that span to estimate the required eigenvectors. We show how to formulate kernel PCA and spectral clustering within this framework. In theory one can also write a functional form for other data dependant kernels like LLE and Isomap kernels, but the update of the matrix becomes relatively ineffective in these cases and should be the subject of future research. Experiments show that for kernel PCA and spectral clustering, the selection method is significantly better than random selection, and better than simply using the Nyström method to generalize (as in Isomap’s landmark variant (de Silva and Tenenbaum, 2003)). In fact it worked almost as well as training with all the data.

## Acknowledgements

The authors would like to thank the following funding organizations for support: NSERC, FQRNT, MITACS, IRIS,

and the Canada Research Chairs.

## References

- Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396.
- Bengio, Y., Delalleau, O., Le Roux, N., Paiement, J.-F., Vincent, P., and Ouimet, M. (2004). Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16(10):2197–2219.
- de Silva, V. and Tenenbaum, J. (2003). Global versus local methods in nonlinear dimensionality reduction. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 705–712, Cambridge, MA. MIT Press.
- Engel, Y., Mannor, S., and Meir, R. (2004). The kernel recursive least squares algorithm. *IEEE Trans. Sig. Proc.*, 52(8):2275–2285.
- Harmeling, S., Ziehe, A., Kawanabe, M., and Müller, K.-R. (2002). Kernel feature spaces and nonlinear blind source separation. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.
- Lawrence, N., Seeger, M., and Herbrich, R. (2003). Fast sparse gaussian process methods: The informative vector machine. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press.
- Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: analysis and an algorithm. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.
- Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.
- Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319.
- Smola, A. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In Langley, P., editor, *International Conference on Machine Learning*, pages 911–918, San Francisco. Morgan Kaufmann.
- Smola, A. J. and Bartlett, P. (2001). Sparse greedy gaussian process regression. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*.
- Tenenbaum, J., de Silva, V., and Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.
- Weiss, Y. (1999). Segmentation using eigenvectors: a unifying view. In *Proceedings IEEE International Conference on Computer Vision*, pages 975–982.
- Williams, C. K. I. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688, Cambridge, MA. MIT Press.