Abstract

Various state-of-the-art machine learning problems rely on kernel based methods. These kernel methods have the ability to solve highly nonlinear problems by reformulating them in a linear context. Hereto, the dominant eigenspace of a (normalized) kernel matrix is often required. Unfortunately, due to the computational requirements of the existing kernel methods, this eigenspace can only be obtained for relatively small data sets.

This paper, the second in a series of two, focuses on a kernel based method for large data sets. More specifically, it investigates the performance of a tracking algorithm for the dominant eigenspace of a normalized kernel matrix, proposed in the first paper of this series [2]. It is found that the tracking algorithm yields a satisfactory approximation of the dominant eigenspace. The loss in accuracy with respect to batch SVD calculations is by far compensated by the minimal computational and memory requirements per iteration step, being $O(nm^2)$ and $O(nm)$, respectively, resulting in a drastic reduction in computation time.

Keywords: Kernel methods, Laplacian eigenspace, singular value decomposition, subspace tracking.

1 Introduction

Kernel based methods have become increasingly popular in various state-of-the-art machine learning problems, such as, e.g., classification, function estimation, pattern recognition, signal processing and system identification problems [8; 9]. These kernel based methods are capable of solving highly nonlinear problems by implicitly mapping all data points in an often high-dimensional space, in which linear techniques are used for solving the reformulated problem.

The majority of the currently employed kernel based methods, such as, e.g., kernel principal component analysis (KPCA)[7], fixed size least squares support vector machines (FS-LSSVM)[9], spectral clustering [11; 5], rely, either directly or indirectly, on the eigenvectors of the $n \times n$ symmetric kernel Gram matrix $K$, which provides a similarity measure for the $n$ available data points. Typically, only the dominant $m$ eigenvectors of the $n \times n$ kernel matrix $K$ are needed, where $m$ is much smaller than $n$.

Unfortunately, due to computational requirements of the singular value decomposition (SVD) needed to obtain the dominant eigenspace of the kernel matrix, the applicability of these otherwise well performing kernel methods is limited to relatively small data sets and off-line problems. Furthermore, whenever data points are added to or removed from the data set (such as, e.g., when tracking the dominant eigenspace of the kernel matrix), all computations must be repeated in each step.
Algorithms that try to alleviate these problems are currently being developed [6; 4]. To the best of the authors’ knowledge, a truly online algorithm has not been reported so far, although [3] proposed an algorithm for tracking the dominant subspace for solving KPCA problems quasi-online. Therefore, an algorithm capable of tracking the eigenspace of a normalized kernel was introduced in the first paper in this series of two [2]. The proposed algorithm requires substantially less computation time ($O(mn^2)$ operations) and memory space ($O(mn)$) when compared to batch SVD computations.

In this paper, the performance of the algorithm is assessed on the basis of two data sets.

## 2 Laplacian eigenspace tracking algorithm

Given a data set $S = \{x_1, x_2, \ldots, x_n\}$, with $x_i \in \mathbb{R}^p$, the $n \times n$ symmetric kernel Gram matrix $K$ can be computed as $[K]_{ij} = k(x_i, x_j)$, where the kernel function $k$ provides a pairwise similarity measure between data points. When working with a normalized kernel, the leading eigenvectors of a transformed version of the kernel Gram matrix $K$ are used instead of the kernel matrix itself. An example of such a transformation is the divisive normalization which uses the Laplacian $L$ of the kernel matrix $K$ [11; 5]

$$L = D^{-1/2}KD^{-1/2}$$

$$[D]_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ \sum_{k=1}^{n} [K]_{ik} & \text{if } i = j \end{cases}$$

The normalization matrix $D$ is usually defined slightly differently, i.e., $[D]_{ii} = \sum_{i \neq j} [K]_{ik}$, leading to slightly better embeddings of the data points, but having no significant impact upon the algorithm capable of tracking the dominant eigenspace of the normalized Laplacian kernel as proposed in [2]. For clarity, the complete tracking algorithm is summarized in Table 1.

## 3 Performance assessment

The performance of the algorithm proposed in [2] is tested on two data sets. The first data set, depicted in Figure 1, is of an artificial nature. It consists of a number of data points $n$ equal to 1000, and the dimension $p$ of the data set is 2. These 1000 points are sampled from one of four prototypes located on $(1, 0)$, $(-1, 0)$, $(0, 1)$ and $(0, -1)$ with $N(0, \frac{1}{4})$ Gaussian noise.

The second data set is the well-known Abalone data set [1], which is often used as a benchmark in various machine learning problems. The Abalone set has a size $n$ of 3000, and the dimension $p$ of each sample is equal to 7.

For both data sets, the employed kernel function is a radial basis kernel function (or Gaussian kernel).

$$k(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$$

The kernel parameter $\sigma$ is set equal to 1 for the artificial data set, and equal to 10 for the Abalone data set. The normalization is performed using the alternative definition.

$$[D]_{ii} = \sum_{i \neq j} [K]_{ik}$$

The eigenspectra of the kernel matrices are depicted in Figure 2.

The relative difference between two matrices $A$ and $B$ is defined as

$$E_{rel}(A, B) = \frac{||A - B||_F}{||B||_F},$$

where $||\cdot||_F$ denotes the Frobenius norm of the matrix. In all other equations, the subscript...
Table 1: Overview of the Laplacian eigenspace tracking algorithm.

**Given** (i) $D_n$, the divisive normalization matrix of $K_n$, being a kernel matrix associated with the data points $x_k, \ldots, x_{k+n-1}$ and (ii) $U_{nm}$ and $\Sigma_n$, the $m$ leading eigenvectors and corresponding eigenvalues of the normalized kernel matrix $L_n = D_n^{-1/2} K_n D_n^{-1/2}$ of $K_n$,

**calculate** (i) $D_n'$, the divisive normalization matrix of $K_n'$, being a kernel matrix associated with the data points $x_{k+1}, \ldots, x_{k+n}$ and (ii) $U_{nm}'$ and $\Sigma_n'$, the $m$ leading eigenvectors and corresponding eigenvalues of the normalized kernel matrix $L_n' = D_n'^{-1/2} K_n' D_n'^{-1/2}$ of $K_n'$

by

- **updating:**
  - $D_{n+1} 1_{n+1} = \left[ D_n 1_n + a_u \right]$
  - $U_0 = \left[ U_{nm} \begin{bmatrix} 0_m \end{bmatrix} \right]$
  - $D_u = \begin{bmatrix} 1/b_u & 0 \\ 0 & -1/b_u \end{bmatrix}$
  - $A^* = \begin{bmatrix} \sqrt{n+1} D_n^{-1/2} a_u & \sqrt{n+1} D_n^{-1/2} a_u \\ \frac{b_u}{\sqrt{d_{n+1,L}}} \end{bmatrix}$
  - $Q_AR_A Q^R = (I_{n+1} - U_0 U_0^T) A^*$
  - $Q_u = [U_0 Q_A]$
  - $R_u = \begin{bmatrix} I_m & \sqrt{n+1} U_0^T A^* \\ \begin{bmatrix} 0_m \end{bmatrix} & \sqrt{n+1} R_A \end{bmatrix}$
  - $\Sigma_u = \frac{n}{n+1} R_u \begin{bmatrix} \Sigma_n & O_n \\ \begin{bmatrix} 0_n \end{bmatrix} & D_u \end{bmatrix} R_u^T$

- **downdating:**
  - $D_n' 1_n = \begin{bmatrix} 0_n & I_n \end{bmatrix} D_{n+1} \begin{bmatrix} 0_m^T \\ I_n \end{bmatrix} 1_n - a_d$
  - $\begin{bmatrix} \bar{u}^T \\ \bar{u}_u \end{bmatrix} \left[ \begin{bmatrix} \sqrt{1-||u||^2} \\ 0_{m-1} \end{bmatrix} \begin{bmatrix} 0_{m-1}^T \\ I_{m+1} \end{bmatrix} \right]$
  - $M = U_{nm}' M$
  - $Q_d = U_{nm}' M$
  - $\Sigma_d = \frac{n+1}{n} M^{-1} \Sigma_u M^{-T}$
Table 1: (cont.) Overview of the Laplacian eigenspace tracking algorithm.

- performing an SVD and a rotation:
  \[ \Sigma'_{n} \leftarrow U_{\Sigma} \Sigma'_{n} U_{\Sigma}^T \quad {\mbox{SV D, m-rank}} \quad \Sigma_d \]
  \[ U'_{nm} = Q_d U_{\Sigma} \]

Figure 2: The first 50 eigenvalues of the eigenspectra of the normalized kernel matrices for the 1000 × 2 artificial data set (A) and the 3000 × 7 Abalone data set (B). The exponential decrease of the eigenvalues is typical for (normalized) kernels.

* will be omitted for brevity.
All tests were performed in Matlab 7.0 (The MathWorks, Inc., Natick, MA).

3.1 Single point updating performance

The single point updating performance of the algorithm given in Table 1 is studied by adding a random new data point to a base set of \( n \) training samples. Then, \( \tilde{L}_B \), the best rank-\( m \) approximation of the real Laplacian matrix obtained from batch SVD calculations, is compared with \( \tilde{L}_U \), the approximation obtained through the algorithm proposed in [2].

\[
E_{\text{rel}}(\tilde{L}_U, \tilde{L}_B) = \frac{\|\tilde{L}_U - \tilde{L}_B\|}{\|\tilde{L}_B\|} = \frac{\|U_{\Sigma} \Sigma'^{\top} \Sigma U_{\Sigma} - U_B \Sigma B U_B^T\|}{\|U_B \Sigma B U_B^T\|}
\]

This test is performed for various combinations of \((n, m)\) as listed in Table 2.

From these results, it is clear that the accuracy of the updating scheme is satisfactory and further improves as the number of training samples \( n \) increases. Therefore, the proposed algorithm is a valid option for the fast calculation of the embedding of a single new data point.

In addition to the in between relative error, an informative measure of accuracy is the ratio of the relative error of the updating algorithm over the relative error of the batch SVD calculations, relative being here with respect to the full-rank Laplacian matrix.

\[
E_{\text{rel}}(\tilde{L}_U, L) = \frac{\|\tilde{L}_U - L\|}{\|L\|} = \frac{\|U_{\Sigma} \Sigma U_{\Sigma} - U_B \Sigma B U_B^T\|}{\|L\|}
\]

\[
E_{\text{rel}}(\tilde{L}_B, L) = \frac{\|\tilde{L}_B - L\|}{\|L\|} = \frac{\|U_B \Sigma B U_B^T - L\|}{\|L\|}
\]
Table 2: The relative difference between \( \tilde{L}_U \), obtained using the updating algorithm, and \( \tilde{L}_B \), obtained using batch SVD calculations, together with their relative deviation from the full rank Laplacian \( L \), for single point updating.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( m )</th>
<th>Relative difference between approximations ( \frac{|L_U - \tilde{L}_B|}{|\tilde{L}_B|} )</th>
<th>Relative error updating ( \frac{|L_U - L|}{|L|} )</th>
<th>Relative error batch SVD ( \frac{|L_B - L|}{|L|} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>4</td>
<td>1.349 \cdot 10^{-3}</td>
<td>6.306 \cdot 10^{-2}</td>
<td>6.305 \cdot 10^{-2}</td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>1.324 \cdot 10^{-3}</td>
<td>2.919 \cdot 10^{-3}</td>
<td>5.768 \cdot 10^{-3}</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>1.334 \cdot 10^{-3}</td>
<td>1.339 \cdot 10^{-3}</td>
<td>1.126 \cdot 10^{-4}</td>
</tr>
<tr>
<td>500</td>
<td>40</td>
<td>1.317 \cdot 10^{-3}</td>
<td>1.317 \cdot 10^{-3}</td>
<td>1.942 \cdot 10^{-7}</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>6.603 \cdot 10^{-4}</td>
<td>6.310 \cdot 10^{-2}</td>
<td>6.209 \cdot 10^{-2}</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>6.600 \cdot 10^{-4}</td>
<td>5.965 \cdot 10^{-3}</td>
<td>5.928 \cdot 10^{-3}</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>6.439 \cdot 10^{-4}</td>
<td>6.619 \cdot 10^{-4}</td>
<td>1.528 \cdot 10^{-4}</td>
</tr>
<tr>
<td>1000</td>
<td>40</td>
<td>6.465 \cdot 10^{-4}</td>
<td>6.465 \cdot 10^{-4}</td>
<td>3.700 \cdot 10^{-7}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n )</th>
<th>( m )</th>
<th>Relative difference between approximations ( \frac{|L_U - \tilde{L}_B|}{|\tilde{L}_B|} )</th>
<th>Relative error updating ( \frac{|L_U - L|}{|L|} )</th>
<th>Relative error batch SVD ( \frac{|L_B - L|}{|L|} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>8</td>
<td>4.868 \cdot 10^{-6}</td>
<td>4.995 \cdot 10^{-6}</td>
<td>1.029 \cdot 10^{-6}</td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>5.149 \cdot 10^{-6}</td>
<td>5.150 \cdot 10^{-6}</td>
<td>9.144 \cdot 10^{-8}</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>4.990 \cdot 10^{-6}</td>
<td>4.990 \cdot 10^{-6}</td>
<td>5.620 \cdot 10^{-10}</td>
</tr>
<tr>
<td>500</td>
<td>40</td>
<td>6.896 \cdot 10^{-6}</td>
<td>6.896 \cdot 10^{-6}</td>
<td>7.995 \cdot 10^{-12}</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>2.576 \cdot 10^{-6}</td>
<td>3.178 \cdot 10^{-6}</td>
<td>1.742 \cdot 10^{-6}</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>2.736 \cdot 10^{-6}</td>
<td>2.740 \cdot 10^{-6}</td>
<td>1.235 \cdot 10^{-7}</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>2.542 \cdot 10^{-6}</td>
<td>2.542 \cdot 10^{-6}</td>
<td>6.937 \cdot 10^{-10}</td>
</tr>
<tr>
<td>1000</td>
<td>40</td>
<td>2.943 \cdot 10^{-6}</td>
<td>2.943 \cdot 10^{-6}</td>
<td>9.702 \cdot 10^{-12}</td>
</tr>
</tbody>
</table>

The results of this comparison are also listed in Table 2. The updating algorithm does not exhibit the pinpoint accuracy of the batch SVD calculations, since the ratio of the relative errors deviates significantly from the ideal value of 1 for larger values of \( m \). However, the updating algorithm exhibits fairly small relative errors, with more than acceptable approximation quality.

For the artificial data set, the relative errors of the updating scheme are more significant for small values of \( m \). However, the batch SVD calculations suffer from the same loss in accuracy when \( m \) decreases. Hence, it can be concluded that the updating algorithm is performing well for single point updating in all test cases.

### 3.2 Multiple point updating performance

In a next step, the convergence of the updating algorithm to the real eigenspace of a data set is investigated. Starting from an initial eigenspace of dimension \( n \times m \), new data points are added and the eigenspace is updated in each step. After adding 2000 data points to the data set, the difference between the obtained approximation \( \tilde{L}_U \) and the batch SVD results for the full-rank Laplacian, \( \tilde{L}_B \), is computed, in a way similar to the procedure described in Section 3.1.

The results of this test for various initial training set sizes \( n \) and eigenspace dimensions \( m \) are summarized in Table 3. A can be seen, the
approximation quality observed in Section 3.1 is retained. Furthermore, the relative errors are only slightly larger than in the single point updating case. When comparing the deviation of both approximations from the full rank Laplacian, as expressed by (9) and (11), the performance of the updating algorithm is again more than satisfactory, even if it does not reach the exceptional quality level of the batch SVD calculations. The batch SVD calculations reach this exceptional accuracy by recalculating the eigenspace from scratch whenever a data point is added to the set. The updating algorithm sacrifices some of this accuracy in order to obtain results quickly.

| Artificial data set |
|---------------------|---------------------|---------------------|
| $n$ | $m$ | Relative difference between approximations $\| \tilde{L}_U - \tilde{L}_B \| / \| \tilde{L}_B \|$ | Relative error updating $\| \tilde{L}_U - L \| / \| L \|$ | Relative error batch SVD $\| \tilde{L}_B - L \| / \| L \|$ |
| 500 | 4 | $1.456 \cdot 10^{-2}$ | $6.441 \cdot 10^{-2}$ | $6.275 \cdot 10^{-2}$ |
| 500 | 10 | $6.277 \cdot 10^{-3}$ | $8.431 \cdot 10^{-3}$ | $5.634 \cdot 10^{-3}$ |
| 500 | 20 | $8.024 \cdot 10^{-3}$ | $8.025 \cdot 10^{-3}$ | $1.442 \cdot 10^{-4}$ |
| 500 | 40 | $6.892 \cdot 10^{-3}$ | $6.892 \cdot 10^{-3}$ | $7.337 \cdot 10^{-7}$ |
| 1000 | 4 | $7.822 \cdot 10^{-3}$ | $6.243 \cdot 10^{-2}$ | $6.195 \cdot 10^{-2}$ |
| 1000 | 10 | $7.777 \cdot 10^{-3}$ | $9.789 \cdot 10^{-3}$ | $5.949 \cdot 10^{-3}$ |
| 1000 | 20 | $5.599 \cdot 10^{-3}$ | $5.601 \cdot 10^{-3}$ | $1.468 \cdot 10^{-4}$ |
| 1000 | 40 | $7.390 \cdot 10^{-3}$ | $7.390 \cdot 10^{-3}$ | $6.668 \cdot 10^{-7}$ |

| Abalone data set |
|---------------------|---------------------|---------------------|
| $n$ | $m$ | Relative difference between approximations $\| \tilde{L}_U - \tilde{L}_B \| / \| \tilde{L}_B \|$ | Relative error updating $\| \tilde{L}_U - L \| / \| L \|$ | Relative error batch SVD $\| \tilde{L}_B - L \| / \| L \|$ |
| 500 | 8 | $7.747 \cdot 10^{-3}$ | $7.748 \cdot 10^{-3}$ | $1.533 \cdot 10^{-6}$ |
| 500 | 10 | $7.341 \cdot 10^{-5}$ | $7.341 \cdot 10^{-5}$ | $1.279 \cdot 10^{-7}$ |
| 500 | 20 | $3.478 \cdot 10^{-5}$ | $3.478 \cdot 10^{-5}$ | $7.805 \cdot 10^{-10}$ |
| 500 | 40 | $5.036 \cdot 10^{-5}$ | $5.036 \cdot 10^{-5}$ | $1.593 \cdot 10^{-11}$ |
| 1000 | 8 | $5.360 \cdot 10^{-5}$ | $5.362 \cdot 10^{-5}$ | $1.512 \cdot 10^{-6}$ |
| 1000 | 10 | $3.356 \cdot 10^{-5}$ | $3.356 \cdot 10^{-5}$ | $1.304 \cdot 10^{-7}$ |
| 1000 | 20 | $3.692 \cdot 10^{-5}$ | $3.692 \cdot 10^{-5}$ | $1.030 \cdot 10^{-7}$ |
| 1000 | 40 | $4.767 \cdot 10^{-5}$ | $4.767 \cdot 10^{-5}$ | $1.579 \cdot 10^{-11}$ |

3.3 Tracking performance

Finally, the combination of the updating and downdating schemes into the tracking algorithm is tested. Here, the $500 \times m$ eigenspace of both the artificial and the Abalone datasets is tracked during 2000 iteration steps. In each step, a new point is added, and the oldest point is removed from the data set. After each iteration step, the relative deviation of the approximations obtained through the tracking scheme and batch SVD calculations from the full rank Laplacian is calculated, by means of (9) and (11). The evolution of these relative errors is depicted in Figure 3 for the case where $m$ equals 10. Other values of $m$ result in similar plots. As was also observed in Section 3.2, the algorithm does not reach the exceptional quality level of batch SVD calculations when $m$ in-
creases, but nevertheless has a more than acceptable performance.

Another important observation is that the relative error of the updating algorithm reaches a stationary value after about 250 iteration steps. This is a valuable result, as it implies that the updating scheme is an appropriate tool for tracking the eigenspace during a large number of iteration steps before retraining is required.

Table 4 summarizes the stationary approximation errors of the tracking algorithm and batch SVD computation, averaged over the last 1500 iteration steps for various values of $m$. The conclusions drawn in Section 3.2 also apply here.

4 Conclusions

Kernel based methods, frequently exploited in classification, pattern recognition, function estimation, system identification and signal processing problems are very powerful tools, because they are capable of solving highly nonlinear problems by reformulating them in a linear context. Hereto, the dominant eigenspace of a (normalized) kernel matrix is often a prerequisite. Unfortunately, the computational requirements of the current methods restrict their applicability to relatively small data sets.

In this paper, the performance of an algorithm able to efficiently track the dominant eigenspace of a normalized kernel matrix for large data sets, proposed in a previous paper [2], was investigated. When tested on some simple data sets (based on an artificial and the well-known Abalone benchmark data set), it is clear that the tracking algorithm yields a satisfactory approximation of the dominant eigenspace of the normalized kernel matrix. The loss in accuracy with respect to batch SVD calculations is by far compensated by the reduction in computational and memory requirements per iteration step, and the associated time savings. Further research must validate these results on larger and more complex data sets.

Acknowledgements

Work supported in part by Project OT/03/30 of the Research Council of the Katholieke Universiteit Leuven and the Belgian Program on Interuniversity Poles of Attraction, initiated by the Belgian Federal Science Policy Office. Ilse Smets is a postdoctoral fellow with the Fund for Scientific Research Flanders-Belgium (FWO-Vlaanderen). The scientific responsibility is assumed by its authors.

References

Table 4: Mean value of the stationary relative error, averaged over the final 1500 iteration steps of the tracking experiment.

<table>
<thead>
<tr>
<th>Artificial data set</th>
<th>Relative error updating $|\tilde{L}_U - L|/|L|$</th>
<th>Relative error batch SVD $|\tilde{L}_B - L|/|L|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$m$</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>4</td>
<td>6.801 · 10$^{-2}$</td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>2.051 · 10$^{-2}$</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>1.627 · 10$^{-2}$</td>
</tr>
<tr>
<td>500</td>
<td>40</td>
<td>2.078 · 10$^{-2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abalone data set</th>
<th>Relative error updating $|\tilde{L}_U - L|/|L|$</th>
<th>Relative error batch SVD $|\tilde{L}_B - L|/|L|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$m$</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>8</td>
<td>7.291 · 10$^{-4}$</td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>7.203 · 10$^{-4}$</td>
</tr>
<tr>
<td>500</td>
<td>20</td>
<td>7.295 · 10$^{-4}$</td>
</tr>
<tr>
<td>500</td>
<td>40</td>
<td>7.380 · 10$^{-4}$</td>
</tr>
</tbody>
</table>


