Kernelizing PLS, Degrees of Freedom, and Efficient Model Selection

Nicole Krämer NKRAEMER@CS.TU-BERLIN.DE
Mikio L. Braun MIKIO@CS.TU-BERLIN.DE
Technische Universität Berlin, Franklinstr. 28/29, 10587 Berlin, Germany

Abstract

Kernelizing partial least squares (PLS), an algorithm which has been particularly popular in chemometrics, leads to kernel PLS which has several interesting properties, including a sub-cubic runtime for learning, and an iterative construction of directions which are relevant for predicting the outputs. We show that the kernelization of PLS introduces interesting properties not found in ordinary PLS, giving novel insights into the workings of kernel PLS and the connections to kernel ridge regression and conjugate gradient descent methods. Furthermore, we show how to correctly define the degrees of freedom for kernel PLS and how to efficiently compute an unbiased estimate. Finally, we address the practical problem of model selection. We demonstrate how to use the degrees of freedom estimate to perform effective model selection, and discuss how to implement cross-validation schemes efficiently.

1. Introduction

Kernel methods have proven to work extremely well for a large range of machine learning applications, and a multitude of different variants exist, ranging from support vector machines (SVMs) to kernel ridge regression (KRR). Each of these algorithms comes with its own strengths and weaknesses. For example, SVMs produce sparse results permitting fast predictions, while the complexity of implementing SVMs is significantly larger than, for example, KRR. On the other hand, KRR produces non-sparse solutions leading to scalability problems.

On the algorithmic-complexity-vs.-scalability axis, kernel PLS inhabits an interesting middle position: Kernel PLS can be implemented in a simple iterative scheme whose runtime scales sub-cubic in the number of training examples. In each iteration, only an matrix-vector multiplication is required. In this respect, kernel PLS is similar to using conjugate gradient type methods for solving KRRs, but as we argue below, kernel PLS usually converges to a good solution using fewer steps.

Kernel PLS therefore seems to be a promising addition to the toolbox of existing kernel methods. However, some ingredients for making kernel PLS readily available are still missing, and the goal of this paper is to address some of these issues.

On the theoretical side, we discuss several properties of kernel PLS not found in ordinary PLS, which help to explain how kernel PLS works. We illustrate that for kernels which can be interpreted as smoothing operators (e.g., Gaussian kernels), kernel PLS can be interpreted as an iterative refinement using smoothed residuals.

For ordinary PLS, there is a close relationship to conjugate gradient (CG) methods, namely that the PLS solutions are equal to the conjugate gradient approximations applied to the normal equation. We show that this connection is less strong between kernel PLS and KRR. In fact, it turns out that kernel PLS and CG for KRR optimize on the same subspaces, but the objective function used by kernel PLS is more suited to regression problems leading to faster convergence of the solutions.

Finally, for regression methods, the notion of degrees of freedom characterizes the complexity of the model well. We discuss how to define the degrees of freedom correctly for kernel PLS and propose an efficient algorithm for computing an unbiased estimate.

On the practical side, the issue of efficient and effective model selection is of prime importance for applications. Using the degrees of freedom estimate, we perform model selection in a minimum description length framework. For cross-validation schemes, we discuss how to efficiently implement testing across different number of components.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

2. Kernelizing PLS

We briefly introduce the ordinary PLS algorithm and derive a variant of kernelized PLS. We start with defining some notation. As usual, we consider a learning problem with inputs (or predictors) $\boldsymbol{x}_i$ and outputs y_i. The data matrix $\boldsymbol{X}$ is the matrix whose *rows* are the centered $\boldsymbol{x}_i$. The vector $\boldsymbol{y}$ consists of the centered outputs y_i. The number of observations is denoted by n, the number of variables is denoted by d. The number of PLS components is m. For any set of linearly independent vectors $\boldsymbol{S} = \boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$, we denote by $\mathcal{P}_{\boldsymbol{S}}$ the projection onto the space that is spanned by the vectors $\boldsymbol{s}_i$. In matrix notation, we have $\mathcal{P} = \boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T$. The kernel matrix $\boldsymbol{X}\boldsymbol{X}^T$ is denoted by $\boldsymbol{K}$. Furthermore, we set $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y}$. In the linear case, we consider the linear regression model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{e}$. We only discuss PLS for a one-dimensional response, although PLS can also handle multivariate responses. As we illustrate below, PLS is closely related to conjugate gradients and Krylov methods. On this account, we recall the definition of Krylov subspaces. (For more details on Krylov spaces see e.g. Golub & Van Loan, 1996) For a matrix $\boldsymbol{C} \in \mathbb{R}^{c\times c}$ and $\boldsymbol{c} \in \mathbb{R}^c$, we call the set of vectors $\boldsymbol{c}, \boldsymbol{C}\boldsymbol{c}, \ldots, \boldsymbol{C}^{m-1}\boldsymbol{c}$ the Krylov sequence of length m. The space spanned by these vectors is called a Krylov space and is denoted by

$$\mathcal{K}_m\left(\boldsymbol{C}, \boldsymbol{c}\right) = \operatorname{span}\left\{\boldsymbol{c}, \boldsymbol{C}\boldsymbol{c}, \ldots, \boldsymbol{C}^{m-1}\boldsymbol{c}\right\} .$$

2.1. Ordinary Partial Least Squares

The main idea of PLS is to build a few orthogonal components $\boldsymbol{T} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m)$ from the original predictors $\boldsymbol{X}$ and to use them in a least squares regression in place of $\boldsymbol{X}$. PLS is similar to principal components regression (PCR). The difference is that PCR extracts components that explain the variance in the predictor variables whereas PLS extracts components that have a large covariance with $\boldsymbol{y}$.

A latent component $\boldsymbol{t}$ is a linear combination $\boldsymbol{t} = \boldsymbol{X}\boldsymbol{w}$ of the predictor variables. The vector $\boldsymbol{w}$ is called the weight vector. We want to find a component with maximal covariance to $\boldsymbol{y}$, that is, for the first component $\boldsymbol{t}_1 = \boldsymbol{X}\boldsymbol{w}_1$, we maximize

$$\boldsymbol{w}_1 = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmax}} \frac{\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{y}\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{y} . \qquad (1)$$

Subsequent components $\boldsymbol{t}_2, \boldsymbol{t}_3, \ldots$ are chosen such that they maximize the squared covariance to $\boldsymbol{y}$ and that all components are mutually orthogonal. Orthogonality can be ensured by deflating the original predictor variables $\boldsymbol{X}$. That is, we only consider the part of $\boldsymbol{X}$ that is orthogonal on all components $\boldsymbol{t}_j, j < i$:

$$\boldsymbol{X}_i = \boldsymbol{X} - \mathcal{P}_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_{i-1}}\boldsymbol{X} . \qquad (2)$$

We then replace $\boldsymbol{X}$ by $\boldsymbol{X}_i$ in (1). This is called the NIPALS method (Wold, 1975). We remark that there are different techniques to extract subsequent components. More information on different versions of PLS can be found in Rosipal and Krämer (2006).

In order to predict the output for new observations, we have to determine the vector of regression coefficients $\widehat{\boldsymbol{\beta}}_m$ defined by

$$\widehat{\boldsymbol{y}}_m = \mathcal{P}_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_m}\boldsymbol{y} = \boldsymbol{X}\widehat{\boldsymbol{\beta}}_m .$$

Therefore, we need a representation of the components in terms of the original data, i.e.

$$\boldsymbol{t}_i = \boldsymbol{X}_i\boldsymbol{w}_i = \boldsymbol{X}\widetilde{\boldsymbol{w}}_i . \qquad (3)$$

This can be done by exploiting the fact (Manne, 1987) that the matrix $\widetilde{\boldsymbol{L}} = \boldsymbol{T}^T\boldsymbol{X}\boldsymbol{W}$ is upper bidiagonal, i.e. $\widetilde{l}_{ij} = 0$ for $i > j$ or $i < j-1$. Furthermore, setting $\boldsymbol{D} = \operatorname{diag}\left(1/\|\boldsymbol{t}_1\|, \ldots, 1/\|\boldsymbol{t}_m\|\right)$, we have

$$\boldsymbol{X}\boldsymbol{W} \quad = \quad (\boldsymbol{T}\boldsymbol{D})\left(\boldsymbol{D}\widetilde{\boldsymbol{L}}\right) . \qquad (4)$$

In particular, the columns of $\boldsymbol{T}$ and the columns of $\boldsymbol{X}\boldsymbol{W}$ span the same space. Note that (4) is in fact the QR-factorization of $\boldsymbol{X}\boldsymbol{W}$. Below, we derive an efficient algorithm for both kernel PLS and its degrees of freedom based on this relationship. We conclude this section by remarking that (Helland, 1988)

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{b}) \quad = \quad \operatorname{span}\left(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m\right) , \qquad (5)$$

$$\mathcal{K}_m(\boldsymbol{K}, \boldsymbol{K}\boldsymbol{y}) \quad = \quad \operatorname{span}\left(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m\right) . \qquad (6)$$

which already points at the close relationship of PLS and Krylov spaces.

2.2. Kernel Partial Least Squares

A derivation of PLS in terms of the kernel matrix $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T$ and $\boldsymbol{y}$ is already defined in Rännar et al. (1994) in order to speed up the computation of PLS. The extension of PLS to nonlinear regression and classification using the kernel trick is proposed in Rosipal and Trejo (2001) and Rosipal et al. (2003). There, a more general setting with multivariate output is considered. For the univariate case, the derivation can however be simplified. There are different equivalent possibilities to define the PLS solution in terms of the Kernel matrix. All of them are based on (6) which implies that

$$\widehat{\boldsymbol{y}}_m = \mathcal{P}_{\boldsymbol{T}}\boldsymbol{y} = \mathcal{P}_{\mathcal{K}_m(\boldsymbol{K},\boldsymbol{K}\boldsymbol{y})}\boldsymbol{y} = \boldsymbol{K}\boldsymbol{\alpha}_m . \qquad (7)$$

Here, $\boldsymbol{\alpha}_m$ are the coefficients in the kernel expansion

$$f(\boldsymbol{x}) \quad = \quad \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)$$

for PLS with m components. In order to compute $\boldsymbol{\alpha}_m$, we start by defining an ordinary basis $\widetilde{\boldsymbol{T}} = (\widetilde{\boldsymbol{t}}_1, \ldots, \widetilde{\boldsymbol{t}}_m)$ of $\mathcal{K}_m(\boldsymbol{K}, \boldsymbol{K}\boldsymbol{y})$. E.g. we might simply choose $\widetilde{\boldsymbol{t}}_i = \boldsymbol{K}^i\boldsymbol{y}$. In order to determine $\boldsymbol{\alpha}_m$ we first represent the basis $\widetilde{\boldsymbol{T}}$ as $\widetilde{\boldsymbol{T}} = \boldsymbol{K}\boldsymbol{R}$. Let us denote the QR-factorization of $\widetilde{\boldsymbol{T}}$ by $\widetilde{\boldsymbol{T}} = \boldsymbol{T}\boldsymbol{L}$. It follows that

$$\boldsymbol{T} \quad = \quad \widetilde{\boldsymbol{T}}\boldsymbol{L}^{-1} = \boldsymbol{K}\boldsymbol{R}\boldsymbol{L}^{-1}. \tag{8}$$

Combining (7) and (8), we obtain

$$\widehat{\boldsymbol{y}}_m = \boldsymbol{T}\boldsymbol{T}^T\boldsymbol{y} = \boldsymbol{K}\boldsymbol{R}\boldsymbol{L}^{-1}\boldsymbol{T}^T\boldsymbol{y}.$$

It follows that

$$\boldsymbol{\alpha}_m \quad = \quad \boldsymbol{R}\boldsymbol{L}^{-1}\boldsymbol{T}^T\boldsymbol{y}.$$

Although a priori any basis $\widetilde{\boldsymbol{T}}$ will do, it is crucial to derive an algorithm that is computationally efficient. This can be achieved by exploiting (4). More precisely, we choose $\widetilde{\boldsymbol{T}} = \boldsymbol{X}\boldsymbol{W}$ and derive the QR-factorization iteratively by exploiting that $\widetilde{\boldsymbol{L}}$ is bidiagonal. First, a kernel representation of $\widetilde{\boldsymbol{T}} = \boldsymbol{K}\boldsymbol{R}$ can be derived by exploiting (2), which implies that $\boldsymbol{w}_i = \boldsymbol{X}^T\boldsymbol{r}_i$ with $\boldsymbol{r}_i = \boldsymbol{y} - \widehat{\boldsymbol{y}}_i$. We set $\widetilde{\boldsymbol{t}}_i = \boldsymbol{X}\boldsymbol{w}_i = \boldsymbol{K}\boldsymbol{r}_i$, and derive a recursive formula for the expansion (recall (3))

$$\boldsymbol{t}_i = \boldsymbol{X}\widetilde{\boldsymbol{w}}_i =: \boldsymbol{K}\boldsymbol{\gamma}_i.$$

Note that by definition, $\boldsymbol{t}_i = (\boldsymbol{X} - \mathcal{P}_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_m})\,\boldsymbol{w}_i$. As $\widetilde{\boldsymbol{L}} = \boldsymbol{T}^T\widetilde{\boldsymbol{T}}$ is upper bidiagonal, we have

$$\boldsymbol{t}_i \quad = \quad \widetilde{\boldsymbol{t}}_i - \mathcal{P}_{\boldsymbol{t}_{i-1}}\widetilde{\boldsymbol{t}}_i,$$

from which we conclude that

$$\boldsymbol{\gamma}_i \quad = \quad \boldsymbol{r}_i - \frac{\boldsymbol{t}_{i-1}^T\widetilde{\boldsymbol{t}}_i}{\boldsymbol{t}_{i-1}^T\boldsymbol{t}_{i-1}}\boldsymbol{\gamma}_{i-1}.$$

Finally, we have

$$\widehat{\boldsymbol{y}}_i = \widehat{\boldsymbol{y}}_{i-1} + \mathcal{P}_{\boldsymbol{t}_i}\boldsymbol{y} = \boldsymbol{K}\boldsymbol{\alpha}_{i-1} + \frac{\boldsymbol{t}_i^T\boldsymbol{y}}{\boldsymbol{t}_i^T\boldsymbol{t}_i}\boldsymbol{K}\boldsymbol{\gamma}_i.$$

From this, we derive a recursive formula for $\boldsymbol{\alpha}_i$. The results are summarized in algorithm 1. To ensure numeric stability, we include the stopping criterium in line 13.

Note that the formulas in steps 9, 11 and 15 in algorithm 1 are redundant. E.g. for the computation of $\boldsymbol{t}_i$ in step 11, it is computationally more efficient to use the relationship $\boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{\gamma}_i$. However, for the interpretation of Kernel PLS as an iterative refinement (see next subsection) and for the estimation of its degrees of freedom, the recursive formula $\boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{r}_i - \mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{K}\boldsymbol{r}_i$ is more convenient.

Algorithm 1 Iterative Computation of kernel PLS

1: Input: $\boldsymbol{X}$, $\boldsymbol{y}$, number of components m
2: Initialize: $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{\alpha}_0 = \widehat{\boldsymbol{y}}_0 = \boldsymbol{0}$
3: **for** $i = 1, \ldots, m$ **do**
4: **if** $i = 1$ **then**
5: $\boldsymbol{r}_i = \boldsymbol{\gamma}_i = \boldsymbol{y}$
6: $\widetilde{\boldsymbol{t}}_i = \boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{y}$
7: **else**
8: $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{y}$
9: $\widetilde{\boldsymbol{t}}_i = \boldsymbol{K}\boldsymbol{r}_i [= \widetilde{\boldsymbol{t}}_{i-1} - \boldsymbol{K}\mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{y}]$
10: $\boldsymbol{\gamma}_i = \boldsymbol{r}_i - \frac{\boldsymbol{t}_{i-1}^T\widetilde{\boldsymbol{t}}_i}{\boldsymbol{t}_{i-1}^T\boldsymbol{t}_{i-1}}\boldsymbol{\gamma}_{i-1}$
11: $\boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{\gamma}_i [= \boldsymbol{K}\boldsymbol{r}_i - \mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{K}\boldsymbol{r}_i]$
12: **end if**
13: **exit if** $|\angle(\boldsymbol{t}_i, \boldsymbol{t}_j)| > 0.1$ **for any** $1 \leq j < i$
14: $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \frac{\boldsymbol{t}_i^T\boldsymbol{y}}{\boldsymbol{t}_i^T\boldsymbol{t}_i}\boldsymbol{\gamma}_i$
15: $\widehat{\boldsymbol{y}}_i = \widehat{\boldsymbol{y}}_{i-1} + \mathcal{P}_{\boldsymbol{t}_i}\boldsymbol{y} [= \boldsymbol{K}\boldsymbol{\alpha}_i]$
16: **end for**

2.3. Kernel PLS as Iterative Refinement

Kernel PLS can also be interpreted as an iterative refinement of $\widehat{\boldsymbol{y}}$ using smoothed residuals. In order to compute $\widehat{\boldsymbol{y}}_i$, the ith residual $\boldsymbol{r}_i = \boldsymbol{y} - \widehat{\boldsymbol{y}}_{i-1}$ is computed first (line 8). The residual $\boldsymbol{r}_i$ is then mapped through $\boldsymbol{K}$, and $\boldsymbol{t}_i$ is constructed by orthogonalizing $\boldsymbol{K}\boldsymbol{r}_i$ with respect to $\boldsymbol{t}_{i-1}$ (line 11, due to the bidiagonality, $\boldsymbol{t}_i$ is then also orthogonal to all $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{i-1}$). Finally, $\widehat{\boldsymbol{y}}_i$ is updated by adding the projection of $\boldsymbol{y}$ to $\boldsymbol{t}_i$ (line 15).

The key ingredient here is the application of $\boldsymbol{K}$ to $\boldsymbol{r}_i$. A component of $\boldsymbol{r}_i$ along the eigenvector $\boldsymbol{u}_i$ of $\boldsymbol{K}$ is scaled by the corresponding eigenvalue λ_i. Since smooth kernels, as are typically employed in machine learning, have rapidly decaying eigenvalues, most of these components will be effectively suppressed, and $\boldsymbol{t}_i$ will reflect only the part of the residual corresponding to directions with large eigenvalues. It turns out that this is a very effective strategy for learning. As discussed by Braun et al. (2007), in a kernel setting, the relevant information about the outputs $\boldsymbol{y}$ is contained in a number of leading eigendirections. Kernel PLS iteratively refines $\widehat{\boldsymbol{y}}$ to minimize the residual on this subspace, thereby extracting the relevant information and discarding the noise.

2.4. Kernel PLS vs. Conjugate Gradients

It is well-known that PLS is closely connected to Krylov subspaces and conjugate gradient (CG) methods (Lingjærde & Christophersen, 2000; Phatak & de Hoog, 2002). The connection is given as follows. The ordinary least-squares problem $\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$

is solved by finding the solution of the associated normal equation (recall that $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y}$)

$$\boldsymbol{A}\boldsymbol{\beta} = \boldsymbol{b}\,. \tag{9}$$

One can show that applying the CG method to this equation starting with the initialization $\boldsymbol{\beta}_0 = 0$ produces the same solution as PLS (Helland, 1988; Phatak et al., 2002). The CG method iteratively computes approximate solutions of (9) by minimizing the quadratic function

$$\phi(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{\beta}^T\boldsymbol{b}. \tag{10}$$

along directions that are **A**-orthogonal. The CG algorithm is also closely related to Krylov subspaces and the Lanczos algorithm (Lanczos, 1950). One can show that the CG algorithm minimizes (10) over the Krylov spaces. It follows from (4) that the PLS estimate $\widehat{\boldsymbol{\beta}}_m$ obtained after m steps is obtained by minimizing ϕ over $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{b})$

$$\begin{aligned}\widehat{\boldsymbol{\beta}}_m &= \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathcal{K}_m(\boldsymbol{A},\boldsymbol{b})} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \\ &= \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathcal{K}_m(\boldsymbol{A},\boldsymbol{b})} \left(\frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{\beta}^T\boldsymbol{b}\right).\end{aligned} \tag{11}$$

This connection is remarkable since Krylov spaces are also known to allow good approximations of leading eigenvalues and eigenspaces in the context of Lanczos methods, such that PLS can be interpreted of making use of low-rank approximations to $\boldsymbol{K}$ in a similar spirit as principal component regression, which approximates $\boldsymbol{K}$ by restricting $\boldsymbol{K}$ to the exact leading eigenspaces.

In the context of kernel PLS, a similar result concerning the Krylov spaces holds. Using (6), we conclude that the mth kernel PLS solution can be written as

$$\begin{aligned}\boldsymbol{\alpha}_m &= \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{K}_m(\boldsymbol{K},\boldsymbol{y})} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|^2 \\ &= \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{K}_m(\boldsymbol{K},\boldsymbol{y})} \left(\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{y}\right)\end{aligned} \tag{12}$$

While this term looks very similar to conjugate gradient applied to the equation $\boldsymbol{K}^2\boldsymbol{\alpha} = \boldsymbol{K}\boldsymbol{y}$, note that the Krylov-space is different, since it should read $\mathcal{K}_m(\boldsymbol{K}^2, \boldsymbol{K}\boldsymbol{y})$ for CG.

Using the conjugate gradients approach to the solution of kernel ridge regression (KRR) leads to optimization over the same Krylov spaces, however with a different objective function. In KRR, we have to solve

$$(\boldsymbol{K} + \lambda\boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{y}\,,$$

for some $\lambda > 0$. Since explicitly inverting the matrix is impractical for large numbers of examples, one alternative consists in solving this equation iteratively using the CG method. This often leads to a significant reduction in computation time since one iteration only requires a matrix-vector operation (and additional vector-vector operations). Furthermore, the CG method is guaranteed to converge to the optimal solution after n steps, but usually converges much faster.

Now if we compare the CG cost function for the KRR problem, we get that (with $\boldsymbol{K}_\lambda = \boldsymbol{K} + \lambda\boldsymbol{I}$)

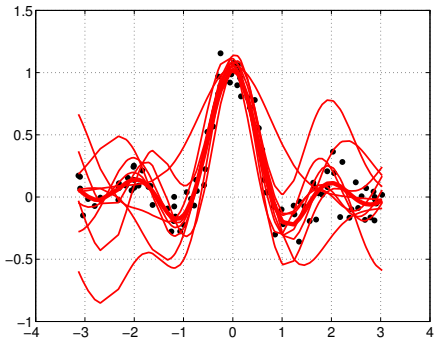
$$\begin{aligned}\phi_{\mathrm{KRR}}(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{K}_\lambda\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\boldsymbol{y} \\ &= \frac{1}{2}(\boldsymbol{K}_\lambda\boldsymbol{\alpha} - \boldsymbol{y})^T\boldsymbol{K}_\lambda^{-1}(\boldsymbol{K}_\lambda\boldsymbol{\alpha} - \boldsymbol{y}) + \text{const.}\end{aligned}$$

Comparing ϕ_{KRR} with (12), we see that both methods minimize the sum of squared residual, however, CG applied to KRR (CG-KRR) rescales the errors by $\boldsymbol{K}_\lambda^{-1}$, leading to an amplification of the error along directions of small eigenvalues of $\boldsymbol{K}$ of up to $1/\lambda$, emphasizing some components of the residual. In other words, the intermediate solutions of CG applied to KRR tries to find a solution with higher accuracy than actually specified by the optimization problem.

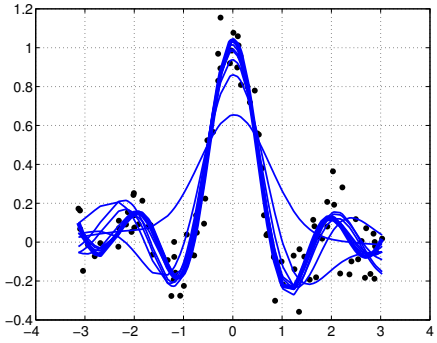
Practically, this results in larger errors and more iterations necessary to obtain a good solution compared to kernel PLS. Consider the following example. We draw 100 inputs X_i uniformly from $[-\pi, \pi]$, and $Y_i = \operatorname{sinc}(X_i) + \varepsilon_i$ with ε_i being normally distributed and having standard deviation 1/10. We choose an rbf-kernel of width 1/2 and set $\lambda = 10^{-3}$. Both kernel PLS and CG-KRR find good solutions after 10 iterations. Comparing the resulting fit functions for CG-KRR and kernel PLS (Figures 1(a) and 1(b)), one can see that the kernel PLS solutions converge faster. This is also reflected by the sum of squared residuals (Figure 1(c)). The observations on the different objective functions is also reflected by Figure 1(d), which plots the difference between $\boldsymbol{\alpha}_5$ and $\boldsymbol{\alpha}_{10}$ decomposed by eigenvector directions of $\boldsymbol{K}$. One can see that kernel PLS is more accurate in the leading eigenvectors, and less accurate on the directions corresponding to small eigenvectors, while the CG-KRR solutions distributes the error more evenly. By mapping $\boldsymbol{\alpha}$ through $\boldsymbol{K}$, any information contained in directions corresponding to small eigenvalues is discarded anyway, such that kernel PLS leads to better fits.

3. Degrees of Freedom

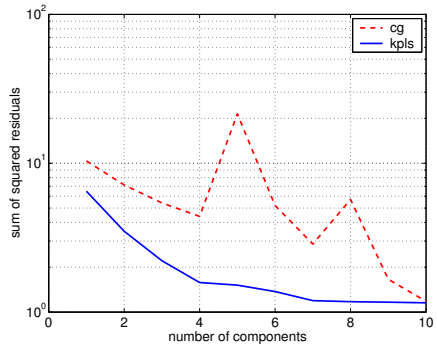
In kernel PLS, both the number of PLS components and the kernel parameters have to be estimated.



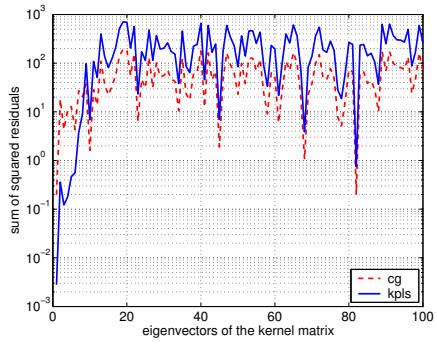
(a) CGs applied to KRR



(b) Kernel PLS



(c) sum of squared residuals



(d) residual by eigendirections

Figure 1. Comparing kernel PLS to conjugate gradients (CG) applied to kernel ridge regression (KRR).

Roughly, we can distinguish between two approaches for model selection.

The k-fold cross-validation method is based on a repeated random splitting of the sample into training and test data. As kernel PLS has a sub-cubic runtime for learning, cross-validation is very efficient as long as k is not too large. The runtime of leave-one-out-cross-validation (i.e. $k = n$) is however cubic in the number of samples, and in contrast to, e.g., kernel ridge regression, there is no efficient computation in terms of generalized cross-validation (Wahba, 1990). This is mainly due to the fact that kernel PLS is not linear in the sense that $\widehat{\boldsymbol{y}}$ depends nonlinearly on $\boldsymbol{y}$.

Information criteria rely on the fact that the true error of a model can be estimated in terms of its training error and its complexity. In regression problems, the complexity is defined in terms of degrees of freedom. For any regression method that fits the response data $\boldsymbol{y}$ by a vector $\widehat{\boldsymbol{y}}$, Efron (2004) defines

$$\mathrm{df} \quad = \quad E_{\boldsymbol{y}} \left[\mathrm{trace} \left(\frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{y}} \right) \right] .$$

Note that if $\widehat{\boldsymbol{y}}$ is linear in $\boldsymbol{y}$, i.e. there is a $n \times n$ matrix $\boldsymbol{H}$ (called the hat-matrix) such that $\widehat{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y}$, we retrieve $\mathrm{df} = \mathrm{trace}(\boldsymbol{H})$, which coincides with the well-known definition given in Hastie and Tibshirani (1990). We obtain an unbiased estimate of the degrees of freedom via

$$\widehat{\mathrm{df}} \quad = \quad \mathrm{trace} \left(\frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{y}} \right) .$$

Degrees of freedom are not only relevant for model selection, they also allow us to quantify the intrinsic complexity of a regression method. As kernel PLS depends nonlinearly on $\boldsymbol{y}$ (via $\boldsymbol{T}$), we derive the estimated degrees of freedom by an explicit computation of the first derivative of $\widehat{\boldsymbol{y}}_m$. In contrast to Phatak et al. (2002), who compute the first derivative of $\widehat{\boldsymbol{\beta}}_m$ based on relationship (11) (which turns out to be extremely time consuming and numerically instable), we follow the iterative approach of Serneels et al. (2004) for the derivative of $\widehat{\boldsymbol{\beta}}_m$, and make explicit use of the recursive definition of kernel PLS of Algorithm 1. To this end, we have to determine the first derivative of the projection operator $\mathcal{P}_{\boldsymbol{v}}\boldsymbol{z} = \boldsymbol{v}\left(\boldsymbol{v}^T\boldsymbol{v}\right)^{-1}\boldsymbol{v}^T\boldsymbol{z}$, with both vectors $\boldsymbol{v}$ and $\boldsymbol{z}$ depending on $\boldsymbol{y}$.

Theorem 1. *The first derivate of* $\mathcal{P}_{\boldsymbol{v}}\boldsymbol{z}$ *is*

$$\frac{\partial \mathcal{P}_{\boldsymbol{v}}\boldsymbol{z}}{\partial \boldsymbol{y}} = \frac{1}{\boldsymbol{v}^T\boldsymbol{v}} \left(\boldsymbol{v}\boldsymbol{z}^T + \boldsymbol{v}^T\boldsymbol{z}\boldsymbol{I}_n \right) \left(\boldsymbol{I}_n - \mathcal{P}_{\boldsymbol{v}} \right) \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{y}} + \mathcal{P}_{\boldsymbol{v}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} .$$

Proof. We split the computation of the derivative into two parts by using the fact that after normalization of

the vector $\boldsymbol{v}$ via $\boldsymbol{v} \leftarrow \boldsymbol{v}/\|\boldsymbol{v}\|$, the projection operator is simply $\boldsymbol{v}\boldsymbol{v}^t\boldsymbol{z}$. Using elementary calculus, we see that

$$\frac{\partial\left(\boldsymbol{v}/\|\boldsymbol{v}\|\right)}{\partial \boldsymbol{y}} \quad = \quad \frac{1}{\|\boldsymbol{v}\|}\left(\boldsymbol{I}_n - \mathcal{P}_{\boldsymbol{v}}\right)\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{y}}\,.$$

Furthermore, using the product rule, we have

$$\frac{\partial\left(\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{z}\right)}{\partial \boldsymbol{y}} \quad = \quad \left(\boldsymbol{v}\boldsymbol{z}^T + \boldsymbol{v}^T\boldsymbol{z}\boldsymbol{I}_n\right)\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{y}} + \boldsymbol{v}\boldsymbol{v}^T\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}\,.$$

Combining these two results via the chain rule, we obtain the desired result. □

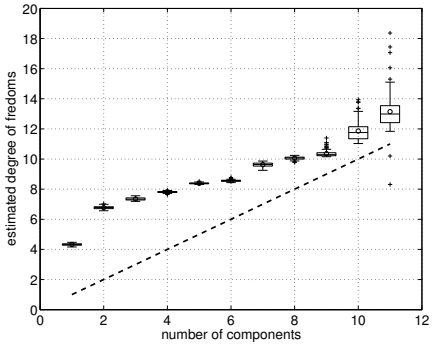
We can now derive the degrees of freedom of kernel PLS by using Theorem 1 and by differentiating the recursive formulas in Algorithm 1. This is displayed in Algorithm 2. Figure 2 illustrates the behavior of

Algorithm 2 Kernel PLS with first derivative of PLS

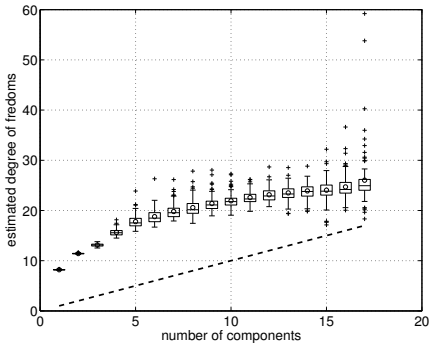
Input: $\boldsymbol{K}$, $\boldsymbol{y}$, m
Initialize: $\widehat{\boldsymbol{y}}_0 = \boldsymbol{\alpha}_0 = \boldsymbol{0}_n$, $\frac{\partial \widehat{\boldsymbol{y}}_0}{\partial \boldsymbol{y}} = \boldsymbol{0}_{n\times n}$
for $i = 1, \ldots, m$ **do**
  if $i = 1$ **then**
    $\boldsymbol{r}_i = \boldsymbol{\gamma}_i = \boldsymbol{y}$
    $\widetilde{\boldsymbol{t}}_i = \boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{y}$
    $\frac{\partial \boldsymbol{t}_i}{\partial \boldsymbol{y}} = \frac{\partial \widetilde{\boldsymbol{t}}_i}{\partial \boldsymbol{y}} = \boldsymbol{K}$
  else
    $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{y}$
    $\widetilde{\boldsymbol{t}}_i = \boldsymbol{K}\boldsymbol{r}_i = \widetilde{\boldsymbol{t}}_{i-1} - \boldsymbol{K}\mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{y}$
    $\frac{\partial \widetilde{\boldsymbol{t}}_i}{\partial \boldsymbol{y}} = \frac{\partial \widetilde{\boldsymbol{t}}_{i-1}}{\partial \boldsymbol{y}} - \boldsymbol{K}\left(\frac{\partial \mathcal{P}_{\boldsymbol{t}_{i-1}}\boldsymbol{y}}{\partial \boldsymbol{y}}\right)$
    $\boldsymbol{\gamma}_i = \boldsymbol{r}_i - \frac{\boldsymbol{t}_{i-1}^T\widetilde{\boldsymbol{t}}_i}{\boldsymbol{t}_{i-1}^T\boldsymbol{t}_{i-1}}\boldsymbol{\gamma}_{i-1}$
    $\boldsymbol{t}_i = \boldsymbol{K}\boldsymbol{\gamma}_i = \widetilde{\boldsymbol{t}}_i - \mathcal{P}_{\boldsymbol{t}_{i-1}}\widetilde{\boldsymbol{t}}_i$
    $\frac{\partial \boldsymbol{t}_i}{\partial \boldsymbol{y}} = \frac{\partial \widetilde{\boldsymbol{t}}_i}{\partial \boldsymbol{y}} - \frac{\partial \mathcal{P}_{\boldsymbol{t}_{i-1}}\widetilde{\boldsymbol{t}}_i}{\partial \boldsymbol{y}}$
  end if
  exit if $|\angle(\boldsymbol{t}_i, \boldsymbol{t}_j)| > 0.1$ **for any** $1 \le j < i$
  $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \frac{\boldsymbol{t}_i^T\boldsymbol{y}}{\boldsymbol{t}_i^T\boldsymbol{t}_i}\boldsymbol{\gamma}_i$
  $\widehat{\boldsymbol{y}}_i = \boldsymbol{K}\boldsymbol{\alpha}_i = \widehat{\boldsymbol{y}}_{i-1} + \mathcal{P}_{\boldsymbol{t}_i}\boldsymbol{y}$
  $\frac{\partial \widehat{\boldsymbol{y}}_i}{\partial \boldsymbol{y}} = \frac{\partial \widehat{\boldsymbol{y}}_{i-1}}{\partial \boldsymbol{y}} + \frac{\partial \mathcal{P}_{\boldsymbol{t}_i}\boldsymbol{y}}{\partial \boldsymbol{y}}$
end for

the degrees of freedom of kernel PLS. We use the sinc-data introduced in Section 2.4. We use rbf kernels with bandwidths 1/2 and 1/10 respectively. Keeping the input data $\boldsymbol{X}$ fixed, we generate the output $\boldsymbol{y}$ 100 different times (using a standard deviation of ε_i of 1/10) and estimate the degrees of freedom of kernel PLS. The graphs in Figure 2 depict the mean degrees of freedom and their boxplots.

In both cases the degrees of freedom is a concave function, i.e., the degrees of freedom of kernel PLS exceed the number of components. To our experience from



(a) Rbf kernel with width 1/2



(b) Rbf kernel with width 1/10

Figure 2. Estimated degrees of freedom for Kernel PLS.

simulated and real world data, the concavity of the graphs is typical for kernel PLS. This supports the conjecture voiced by Frank and Friedman (1993) that $\mathrm{df} \ge m$. In particular, applying the naïve approach $\mathrm{df}(\mathrm{PLS}) = m$ tends to underestimate the complexity of kernel PLS and is expected to choose models parameters that overfit. In the next section, we compare the performance of kernel PLS with three different information criteria for model selection. We use the Akaike information criterion (AIC), the Bayesian information criterion (BIC) and generalized minimum description length (Hansen & Yu, 2001), which is defined as

$$\mathrm{gMDL} = \frac{n}{2}\log S + \frac{d}{2}\log F + \frac{1}{2}\log n,$$
$$S = RSS/(n-d), \qquad F = (\boldsymbol{y}^T\boldsymbol{y} - RSS)/dS,$$

where RSS stands for the residual sums of squares.

4. Experiments

We now present experimental results for the proposed model selection schemes. We will also discuss how to efficiently implement these procedures practically. Note that we do not want to prove the competitiveness of kernel PLS. This has been done before, e.g. by Rosipal and Trejo (2001).

We compare three sets of model selection schemes: (1) the AIC, BIC, and gMDL methods based on our degrees of freedom estimate, (2) leave-one-out cross-validation as a baseline, and (3) 5-fold cross-validation as a computationally faster variant.

In order to implement cross-validation schemes efficiently for kernel PLS, note that the iterative computation of kernel PLS from Algorithm 1 not only computes the final parameter vector for m components, but also computes all $\boldsymbol{\alpha}_i$ for $1 \leq i \leq m$. Therefore, one computation allows to compute the test errors for all choices up to m components. Since the main algorithm requires $O(mn^2)$ computations, it follows that the leave-one-out cross-validation error can be computed in $O(mn^3)$ for a given kernel and all numbers of components up to m. This computational complexity is similar to that of computing the leave-one-out cross-validation error for KRR in closed form (Wahba, 1990). For k-fold cross-validation, the computational cost become significantly better, as the runtime is of order $O(kmn^2)$.

The degrees of freedom estimate also computes all intermediate quantities, such that all degrees of freedom can be computed at once. However, the algorithm involves one matrix-matrix multiplication, such that the overall complexity is practically cubic (although the theoretical run-time is sub-cubic). Note that matrix-matrix multiplications can be parallelized efficiently in contrast to matrix inversions. In summary, all m degrees of freedom estimates can be computed in $O(mn^3)$.

As data sets, we chose the "bank" and "kin" regression data sets from the delve repository[1]. Both data sets are synthetic data sets generated from simulations. The bank data set models customers waiting in a queue, and the tasks consists in predicting the percentage of customers who loose their patience before arriving at the head of the queue. The kin data set is based on a model of a robotic arm, and the task consists in predicting the position of the arm based on the angles of its joints. The data sets come in four different flavors, "fh", "fm", "nh", "nm", where "f" indicates fairly linear connection between inputs and outputs, "n" non-linear connection, "h" "high noise", and "m" medium levels of noise. Both data sets are eight-dimensional, and consist of 8192 data points.

For our experiments, the data sets are split into 20 realizations of 100 training examples and 8092 test examples. Furthermore, the inputs are normalized to lie in the interval $[-1, 1]$. We use a Gaussian kernel with widths chosen from 20 logarithmically spaced points between 1 and 10^4, and the maximum number of components are set to 50. Both kernel width and the number of components are chosen using the respective model selection criterion. Tables 1 and 2 show the resulting mean squared errors and standard deviations over the 20 realizations.

As we expected, leave-one-out cross-validation leads to the best performances. While the AIC and BIC perform poorly on the data sets, the gMDL criterion is on par with leave-one-out cross-validation. But also the computationally much more efficient 5-fold cross-validation still leads to comparably good results.

In summary, we see that the gMDL criterion leads to effective model selection giving further evidence for the practical relevance of our degrees of freedom estimator. On the cross-validation side, even the relatively coarse, but fast, 5-fold cross validation performs well. The additional benefit of the gMDL criterion is that it also produces a degrees of freedom estimate which can then be used to compare kernel PLS to other (linear) methods. Both methods lead to performances on par to those achieved by leave-one-out cross-validation.

5. Conclusions

We have discussed several topics related to kernel PLS relevant to the more widespread adoption of kernel PLS in a machine learning setting. It has turned out that kernel PLS has some interesting properties not shared with ordinary PLS, allowing for alternative interpretations of the algorithm beyond co-optimizing covariance. Furthermore, we have addressed practical issues including efficient implementations, estimation of degrees of freedom, and model selection, using either degrees of freedom or cross-validation based criteria. In practical experiments, we have shown how to perform efficient and effective model selection.

References

Braun, M. L., Buhmann, J. M., & Müller, K.-R. (2007). Denoising and Dimension Reduction in Feature Space. *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press.

Efron, B. (2004). The Estimation of Prediction Error: Covariance Penalties and Cross-Validation. *Journal of the American Statistical Association*, *99*, 619–633.

Frank, I., & Friedman, J. (1993). A Statistical View of Some Chemometrics Regression Tools. *Technometrics*, *35*, 109–135.

[1] http://www.cs.toronto.edu/~delve

Table 1. Results on the "bank" data set (**best result**, *on par* = within 2× standard deviation of best result)

METHOD	FH	FM	NH	NM
AIC	$9.38 \pm 2.49 \times 10^{-3}$	$4.18 \pm 1.71 \times 10^{-3}$	$6.09 \pm 1.11 \times 10^{-3}$	$1.77 \pm 0.17 \times 10^{-3}$
BIC	$\mathbf{6.25 \pm 0.58} \times 10^{-3}$	$\mathit{2.18 \pm 0.41} \times 10^{-3}$	$\mathit{4.32 \pm 0.75} \times 10^{-3}$	$\mathit{1.23 \pm 0.17} \times 10^{-3}$
GMDL	$\mathbf{6.25 \pm 0.58} \times 10^{-3}$	$\mathbf{2.16 \pm 0.38} \times 10^{-3}$	$\mathbf{4.00 \pm 0.43} \times 10^{-3}$	$\mathit{1.23 \pm 0.17} \times 10^{-3}$
5-FOLD CV	$\mathit{6.29 \pm 0.46} \times 10^{-3}$	$\mathit{2.31 \pm 0.44} \times 10^{-3}$	$\mathit{4.16 \pm 0.56} \times 10^{-3}$	$\mathbf{1.16 \pm 0.11} \times 10^{-3}$
LOO-CV	$\mathit{6.30 \pm 0.53} \times 10^{-3}$	$\mathit{2.28 \pm 0.35} \times 10^{-3}$	$\mathit{4.12 \pm 0.65} \times 10^{-3}$	$\mathit{1.18 \pm 0.16} \times 10^{-3}$

Table 2. Results on the "kin" data set (**best result**, *on par* = within 2× standard deviation of best result)

METHOD	FH	FM	NH	NM
AIC	$3.17 \pm 0.43 \times 10^{-2}$	$2.99 \pm 0.47 \times 10^{-2}$	$6.92 \pm 0.33 \times 10^{-2}$	$4.91 \pm 0.46 \times 10^{-2}$
BIC	$2.44 \pm 1.37 \times 10^{-2}$	$2.77 \pm 7.15 \times 10^{-3}$	$6.92 \pm 0.33 \times 10^{-2}$	$4.91 \pm 0.46 \times 10^{-2}$
GMDL	$\mathit{2.31 \pm 0.15} \times 10^{-3}$	$\mathit{4.84 \pm 0.51} \times 10^{-4}$	$\mathit{5.73 \pm 1.07} \times 10^{-2}$	$4.84 \pm 0.91 \times 10^{-2}$
5-FOLD CV	$\mathbf{2.29 \pm 0.12} \times 10^{-3}$	$\mathbf{4.38 \pm 0.65} \times 10^{-4}$	$\mathbf{5.14 \pm 0.53} \times 10^{-2}$	$\mathit{4.23 \pm 0.32} \times 10^{-2}$
LOO-CV	$\mathit{2.30 \pm 0.17} \times 10^{-3}$	$\mathit{4.60 \pm 0.36} \times 10^{-4}$	$\mathit{5.26 \pm 0.63} \times 10^{-2}$	$\mathbf{4.19 \pm 0.30} \times 10^{-2}$

Golub, G., & Van Loan, C. (1996). *Matrix Computations*. Johns Hopkins University Press.

Hansen, M., & Yu, B. (2001). Model Selection and Minimum Descripion Length Principle. *Journal of the American Statistical Association*, *96*, 746–774.

Hastie, T., & Tibshirani, R. (1990). *Generalized Additive Models*. Chapman and Hall, London.

Helland, I. (1988). On the Structure of Partial Least Squares Regression. *Communications in Statistics, Simulation and Computation*, *17*, 581–607.

Lanczos, C. (1950). An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators. *Journal of Research of the National Bureau of Standards*, *45*, 225–280.

Lingjærde, O., & Christophersen, N. (2000). Shrinkage Structure of Partial Least Squares. *Scandinavian Journal of Statistics*, *27*, 459–473.

Manne, R. (1987). Analysis of Two Partial-Least-Squares Algorithms for Multivariate Calibration. *Chemometrics and Intelligent Laboratory Systems*, *2*, 187–197.

Phatak, A., & de Hoog, F. (2002). Exploiting the Connection between PLS, Lanczos Methods and Conjugate Gradients: Alternative Proofs of Some Properties of PLS. *Journal of Chemometrics*, *16*, 361–367.

Phatak, A., Rilley, P., & Penlidis, A. (2002). The Asymptotic Variance of the Univariate PLS Estimator. *Linear Algebra and its Applications*, *354*, 245–253.

Rännar, S., Lindgren, F., Geladi, P., & Wold, S. (1994). A PLS Kernel Algorithm for Data Sets with many Variables and Fewer Objects, Part I: Theory and Applications. *Journal of Chemometrics*, *8*, 111–125.

Rosipal, R., & Krämer, N. (2006). Overview and Recent Advances in Partial Least Squares. In *Subspace, latent structure and feature selection techniques*, Lecture Notes in Computer Science, 34–51. Springer.

Rosipal, R., & Trejo, L. (2001). Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Spaces. *Journal of Machine Learning Research*, *2*, 97–123.

Rosipal, R., Trejo, L., & Matthews, B. (2003). Kernel PLS-SVC for Linear and Nonlinear Classification. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 640–647). Washington, DC.

Serneels, S., Lemberge, P., & Espen, P. V. (2004). Calculation of PLS Prediction Intervals Using Efficient Recursive Relations for the Jacobian Matrix. *Journal of Chemometrics*, *18*, 76–80.

Wahba, G. (1990). *Spline Models For Observational Data*. Society for Industrial and Applied Mathematics.

Wold, H. (1975). Path models with Latent Variables: The NIPALS Approach. In et H. B. al. (Ed.), *Quantitative Sociology: International Perspectives on Mathematical and Statistical Model Building*, 307–357. Academic Press.