Orders of magnitude speed increase in Partial Least Squares feature selection with new simple indexing technique for very tall data sets

P. Stefansson, U. G. Indahl, K. H. Liland, and I. Burud

Faculty of Science and Technology, Norwegian University Of Life Sciences.

Feature selection is a challenging combinatorial optimization problem that tends to require a large number of candidate feature subsets to be evaluated before a satisfying solution is obtained. Due to the computational cost associated with estimating the regression coefficients for each subset, feature selection can be an immensely time consuming process and is often left inadequately explored. Here we propose a simple modification to the conventional sequence of calculations involved when fitting a number of feature subsets to the same response data with partial least squares model fitting. The modification consists in establishing the covariance matrix for the full set of features by an initial calculation and then deriving the covariance of all subsequent feature subsets solely by indexing into the original covariance matrix. By choosing this approach, which is primarily suitable for tall design matrices with significantly more rows than columns, we avoid redundant (identical) recalculations in the evaluation of different feature subsets. By benchmarking the time required to solve regression problems of various sizes, we demonstrate that the introduced technique outperforms traditional approaches by several orders of magnitude when used in conjunction with Partial Least Squares (PLS) modeling. In the supplementary material, we provide code for implementing the concept with kernel partial least squares regression.

Keywords: PLS, feature selection, variable selection, subset selection, Kernel PLS

1. INTRODUCTION

For centuries linear least squares fitting has been one of the most important statistical tools for mapping a set of independent variables, $\mathbf{X}$, to a dependent response variable, y. Its use has today spread to nearly every quantitative field of science, and in areas such as bioinformatics and chemometrics multiple linear regression is employed extensively. A serious challenge in the data driven fields consists in identifying which

column vectors contained within a potentially megavariate data matrix $\mathbf{X}$ are significantly correlated to the response vector y and which are not. The process of eliminating non-informative variables from $\mathbf{X}$ is typically referred to as either *feature selection*, *variable selection* or *subset selection*. Only an exhaustive search is guaranteed to identify the globally best feature subset, which becomes computationally infeasible as soon as the number of independent variables in the data are more than just a few. Some heuristic method is therefore often required in practice to identify a combination of variables that is considered 'good enough'. Examples of feature selection methods frequently used in bioinformatics and chemometrics include: *forward selection*, *backward selection*, *genetic algorithms* (GA), *simulated annealing* and *interval PLS* (iPLS) [1, 2]. Each feature selection technique has its own advantages and disadvantages; stepwise methods such as forward/backward selection are for instance relativley fast to use but are prone to getting stuck in local optimas [1]. Population based feature selection methods such as genetic algorithms can overcome such local optimas, but are often in comparison extremely slow, which limits the circumstances under which they may be applied sucessfully [3]. A characteristic trait of most wrapper-based feature selection methods is that they are driven by trial and error. As such, most methods generally require a large number of candidate subsets to be evaluated before a useful solution is obtained. Due to the computational cost of calibrating regression models for each of the subsets this process can be very time consuming for big data sets.

Here, we present a seemingly trivial insight to matrix multiplications which, when utilized for feature selection purposes in a specific way, leads to nontrivial speedups in the execution time required to explore the performance of a large number of variable subsets subject to linear modeling. The introduced technique is suitable to use together with any feature selection strategy that requires a large number of candidate subsets to be evaluated, such as any of the feature selection methods mentioned above. Our approach involves an extensive reuse of the results from identical calculations that occur across the evaluation of different variable subsets. The calculations available for reusing can easily be applied together with a particular version of the Partial Least Squares (PLS) method in order to obtain substantial improvements in computational performance.

2. THE CONCEPT

2.1. Relevant background information

In ordinary least squares (OLS) regression problems we assume that our $(\mathbf{X}, y)$–data are well described by a linear model of the form

$$y = \mathbf{X}\beta + \epsilon, \tag{1}$$

where the error term ϵ should be 'small'. The least squares solution of this equation is obtained by the estimated vector of regression coefficients, $\hat{\beta}$, minimizing the sum of squared errors $\|\epsilon\|^2 = \|y - \hat{y}\|^2$, where the vector of fitted response values $\hat{y}$ is calculated by the matrix-vector multiplication $\hat{y} = \mathbf{X}\hat{\beta}$.

When the $m \times n$ matrix $\mathbf{X}$ is composed of a large number of observations $m \gg n$ where n denotes the number variables, finding the OLS solution of the linear system with respect to β in the numerically most accurate way may become slow and computationally costly. The *normal equations* for the OLS solution of Eq. 1 is the $n \times n$-system

$$\mathbf{X}^\top y = \mathbf{X}^\top \mathbf{X}\beta. \tag{2}$$

Solving the normal equations directly is usually not recommended due to the potentially unfavourable condition number of $\mathbf{X}^\top \mathbf{X}$ with associated numerical issues. On the other hand, provided that $m \gg n$, Eq. 2 yields a much smaller system of equations which can be solved considerably faster also when accounting for the computational costs of forming the products $\mathbf{X}^\top y$ and $\mathbf{X}^\top \mathbf{X}$. In addition to the OLS use case, these products are also exploited in some partial least squares algorithms, such as kernel PLS, to reduce the dimensions of the regression problem under consideration and speed up the $\hat{\beta}$-estimation.

The feature selection problem requires comparison of a large number of competing models. The computational advantages of solving normal equations rather than the original systems may therefore justify a priority over numerical precision in the explorative phase of a feature selection study where most candidate

models will be discarded anyway. For the most attractive feature combination candidates found in the explorative phase, the final feature evaluation and -selection should be based on repeated modeling using numerically stable algorithms for solving the associated regression problems. In the following section we investigate a simple, but seemingly overlooked, aspect of the $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ calculations and demonstrate that these quantities, which may be the most computationally costly calculations involved in solving Eq. 2, only need to be calculated once regardless of how may column subsets of $\mathbf{X}$ one decides to evaluate.

2.2. Technique for reusing $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ calculations with indexing

Consider a simple design matrix, $\mathbf{X}$, containing three observations of four independent variables, each indicated with its own color:

$$\mathbf{X} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}.$$

And its transpose

$$\mathbf{X}^\top = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}.$$

When multiplying $\mathbf{X}^\top$ with $\mathbf{X}$ to form the quantity $\mathbf{X}^\top\mathbf{X}$ needed to solve Eq. 2, the colors—i.e. the independent variables—will 'blend' with each other according to the rules of linear algebra as shown in Fig. 1.

As can be seen in Fig. 1, the influence of the first column of $\mathbf{X}$—the blue one—can be traced along the first column and the first row of $\mathbf{X}^\top\mathbf{X}$. The second column of the original $\mathbf{X}$ matrix—the green one—only influences the second column and the second row of $\mathbf{X}^\top\mathbf{X}$. This pattern continues throughout the resulting matrix product regardless of what dimension the original $\mathbf{X}$ matrix is of. In summary; the influence of the j^{th} column of $\mathbf{X}$ is confined to the j^{th} row and column of $\mathbf{X}^\top\mathbf{X}$.

This means that if one already has determined the $\mathbf{X}^\top\mathbf{X}$ matrix using all available columns of $\mathbf{X}$ and

then, for instance, wishes to obtain what the matrix product would have been had the last column of $\mathbf{X}$—the red one—not been included, it is not necessary to redo the matrix multiplication with one less column. Instead the last row and column of the full $\mathbf{X}^\top\mathbf{X}$ matrix product can simply be discarded.

$$\mathbf{X}^\top\mathbf{X} = \begin{bmatrix} 1\cdot1+2\cdot2+3\cdot3 & 1\cdot4+2\cdot5+3\cdot6 & 1\cdot7+2\cdot8+3\cdot9 & 1\cdot10+2\cdot11+3\cdot12 \\ 4\cdot1+5\cdot2+6\cdot3 & 4\cdot4+5\cdot5+6\cdot6 & 4\cdot7+5\cdot8+6\cdot9 & 4\cdot10+5\cdot11+6\cdot12 \\ 7\cdot1+8\cdot2+9\cdot3 & 7\cdot4+8\cdot5+9\cdot6 & 7\cdot7+8\cdot8+9\cdot9 & 7\cdot10+8\cdot11+9\cdot12 \\ 10\cdot1+11\cdot2+12\cdot3 & 10\cdot4+11\cdot5+12\cdot6 & 10\cdot7+11\cdot8+12\cdot9 & 10\cdot10+11\cdot11+12\cdot12 \end{bmatrix}$$

FIG. 1: Illustration of how the elements of $\mathbf{X}^\top$and $\mathbf{X}$ would blend during the calculation of the matrix product $\mathbf{X}^\top\mathbf{X}$. Numbers are colored according to their column origin in the example $\mathbf{X}$ given in section 2.2.

In fact, when $\mathbf{X}^\top\mathbf{X}$ is calculated including the complete set of n variables in $\mathbf{X}$, each of the matrix products $\mathbf{X}_i^\top\mathbf{X}_i$ obtained by defining $\mathbf{X}_i$ as one of the (2^n-1) possible column subset matrices of $\mathbf{X}$ can be derived without additional calculations—simply by indexing into the already available full product $\mathbf{X}^\top\mathbf{X}$. The same idea holds true for finding $\mathbf{X}_i^\top y$ by indexing into $\mathbf{X}^\top y$. Thus for any variable subset matrix $\mathbf{X}_i$, the associated normal equations $\mathbf{X}_i^\top y = \mathbf{X}_i^\top\mathbf{X}_i\beta$ can be obtained directly by an appropriate indexing into the full normal equations (2). To see in mathematical notation why this is true, assume that the vector $y \in \mathbb{R}^m$ and the matrix $\mathbf{X} = [x_1\ x_2\ ...\ x_n]$ has dimension $m \times n$ where the column vectors $x_1,\ ...,\ x_n \in \mathbb{R}^m$. Note that except for the diagonal values of $\mathbf{X}^\top\mathbf{X}$, exactly two vectors are involved in the calculation of each entry for both

$$\mathbf{X}^\top\mathbf{X} = \begin{bmatrix} x_1^\top x_1 & x_1^\top x_2 & ... & x_1^\top x_n \\ x_2^\top x_1 & x_2^\top x_2 & ... & x_2^\top x_n \\ : & ... & x_i^\top x_j & : \\ x_n^\top x_1 & x_2^\top x_n & ... & x_n^\top x_n \end{bmatrix} \text{ and } \mathbf{X}^\top y = \begin{bmatrix} x_1^\top y \\ x_2^\top y \\ : \\ x_i^\top y \\ : \\ x_n^\top y \end{bmatrix}.$$

Each matrix– and vector entry is calculated by taking dot products of the form $x_i^\top x_j$ and $x_i^\top y$, respectively. Elimination of all entries involving any specific vector x_i will therefore remove its contribution to the final product—clearly without influencing any of the remaining dot products. As will be demonstrated below, the performance implications of this observation are profound when using a kernel PLS algorithm for evaluating multiple variable combinations.

3. APPLICATION TO FEATURE SELECTION WITH PARTIAL LEAST SQUARES, PLS

In fields such as chemometrics where multicolinearity amongst the variables in $\mathbf{X}$ is common, a partial least squares approach is often used instead of ordinary least squares due to the robustness benefits that comes with using latent rather than actual variables under such conditions [4, 7]. For large data sets where the $\mathbf{X}$ matrix is either very tall or very wide *Kernel PLS* algorithms have been developed as faster alternatives to the conventional NIPALS PLS fitting procedure [5, 8, 9]. The primary focus in this paper lies on situations where $\mathbf{X}$ is tall, i.e. has substantially more rows than columns ($m \gg n$). In such cases, the original Kernel PLS algorithm [5] or any of the existing derivatives/improvements of which [6] generally constitutes good algorithm choices in terms of computational efficiency. One property of these kernel algorithms—that turns out to be greatly beneficial when conducting feature selection—is that they base their entire parameter estimation process around the information content of the covariance matrices $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$. Because they operate on the covariance matrices, the indexing strategy introduced in section 2.2 can easily be incorporated into the fitting process in order to further increase the computational efficiency of evaluating multiple feature subsets. Furthermore, calculating the quantities $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ is generally amongst the very first steps involved in kernel based PLS algorithms, which makes the necessary alteration required to incorporate the indexing technique from section 2.2 trivial to implement and limited to a small part of the calculation sequence. To speed up the coefficient fitting of a batch of feature selections with a kernel PLS algorithm, the only modification required compared to a conventional naive approach is to loop through the subsets one by one *after* the initial covariance matrices have been calculated, rather than

placing the same loop around the entire PLS algorithm. Essentially this means that the variable selection procedure is placed inside the PLS algorithm rather than wrapped around it. The fundamental differences between the two approaches are respectively made clear in algorithms 1 and 2 which depict the concepts behind conventional variable selection using kernel PLS and the suggested modified implementation. Practically implementing the matrix indexing technique described in section 2.2 is exceptionally simple in most high-level programming languages: first the quantities $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ are calculated with the full set of variables. Then, the relevant covariance elements for any feature subset can be extracted from these quantities by applying the same indexing logic across all dimensions. The most straightforward approach to achieving this is to represent a particular feature selection as an n-dimensional Boolean vector and then applying the vector as a means of indexing into $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$, i.e. only including the dot products of intersecting $true$-elements. The process can then be repeated for all feature subsets in a loop as shown in algorithm 2.

Algorithm 1 PLS(X, y, SubSets)

```
/* Loop over candidate subsets */
for i ← 0 to k do
  /* XX and Xy for relevant subset */
  X_i = X[:,SubSets[i,:]]
  XX_i = X_i^⊤ × X_i
  Xy_i = X_i^⊤ × y

  /* Kernel PLS using XX_i and Xy_i */
  β_i = KernelPLS(XX_i,Xy_i)
end for
```

Algorithm 1: Pseudocode explaining how a batch of feature selections conventionally would be estimated using kernel PLS. Input variables are assumed to be an $m \times n$ design matrix $\mathbf{X}$, an $m \times 1$ response vector y and a $k \times n$ Boolean matrix $SubSets$ containing k different subsets represented as $1 \times n$ vectors.

Algorithm 2 FastPLS(X, y, SubSets)

```
/* Calculate full covariance matrices */
FullXX = X^T×X
FullXy = X^T×y

/* Loop over candidate subsets */
for i ← 0 to k do
  /* XX and Xy for relevant subset */
  XX_i = FullXX[SubSets[i,:],SubSets[i,:]]
  Xy_i = FullXy[SubSets[i,:]]

  /* Kernel PLS using XX_i and Xy_i */
  β_i = KernelPLS(XX_i,Xy_i)
end for
```

Algorithm 2: Pseudocode illustrating modifications to algorithm 2 necessary to reuse covariance calculations between variable subsets to speed up the fitting of a batch of feature selections using a kernel PLS algorithm.

3.1. BENCHMARK OF FEATURE SELECTION WITH RANDOM SEARCH

To experimentally validate the supposed performance benefits that comes with reusing the full $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ calculations for all feature subsets rather than individually determining them for each variable subset, both methods (algorithm 1 & 2) were benchmarked in terms of execution time over various problem sizes. Four different $\mathbf{X}$ matrices with 100 columns and 10^4, 10^5, 10^6 and 10^7 rows, respectively, were generated and populated with pseudorandom data together with five y vectors with the same numbers of rows. Batches containing 1, 1000, 2000, 3000, 4000 and 5000 feature selections were randomly generated with a uniform distribution of active and inactive variables. Using the same input data the PLS regression between $\mathbf{X}$ and y was then performed using both the method that reuses covariance calculations (algorithm 2) and the conventional kernel PLS method (algorithm 1). For both algorithms the maximum number of PLS components was set to 15. The kernel algorithm used to perform the parameter estimation during the benchmark was the *Modified kernel algorithm #2* [6]. In appendix 1, a MATLAB implementation of this algorithm is provided with the indexing technique from algorithm 2 incorporated. The implementation of the Modified kernel algorithm #2 provided in the appendix

differs slightly from Dayal and MacGregor's original algorithm [6] in the sense that we have included a stabilizing reorthogonalization (line 42, appendix 1), eliminated some redundant intermediate calculations (the lines 44-49, appendix 1) and simplified the regression coefficient calculations (line 50, appendix 1).

The runtimes for the subset sizes k ($1 \leq k \leq 5000$) that were not directly evaluated in our benchmark experiments were linearly interpolated to provide a more coherent trend line. Figure 2 shows the result of the benchmark and indicates that the performance benefits of reusing the full $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ for each feature subset grows as the number of evaluated subsets increases. When only evaluating one feature subset the modification described in section 2.2 naturally offers no performance benefits at all and is consistently slightly slower than the conventional approach. When evaluating a large number of feature subsets for a regression problem with many observations however, the technique described in section 2.2 is several orders of magnitude faster than the conventional method and peaks in our tests at a runtime decrease of roughly 5920x ($m = 10^7$, $k = 5000$).

A drawback with PLS algorithms in terms of computational efficiency is that they are inherently serial in their execution since each fitted component builds upon the previous one. Because $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ which a kernel PLS algorithm operates on are typically very small in size compared to the full $\mathbf{X}$ and y, kernel PLS algorithms require very little working memory as they run. An advantage of this is that it allows multiple kernel PLS instances to be executed in parallel across several threads such that several feature subsets are evaluated simultaneously—even though each individual algorithm runs in serial. In the supplementary material found online [LINK?], a GPU implementation of algorithm 2 written in CUDA C is available which assigns one thread—of potentially thousands available on modern GPUs—to the fitting of each of the k feature selections. The GPU implementation can also be called from MATLAB through the MEX interface. The benchmark results of this implementation are shown in green in Fig. 2. When including the CUDA implementation in the comparison, the speedup increases even more and peaks at around 7316x at $m = 10^7$ $k = 5000$ compared to algorithm 1.

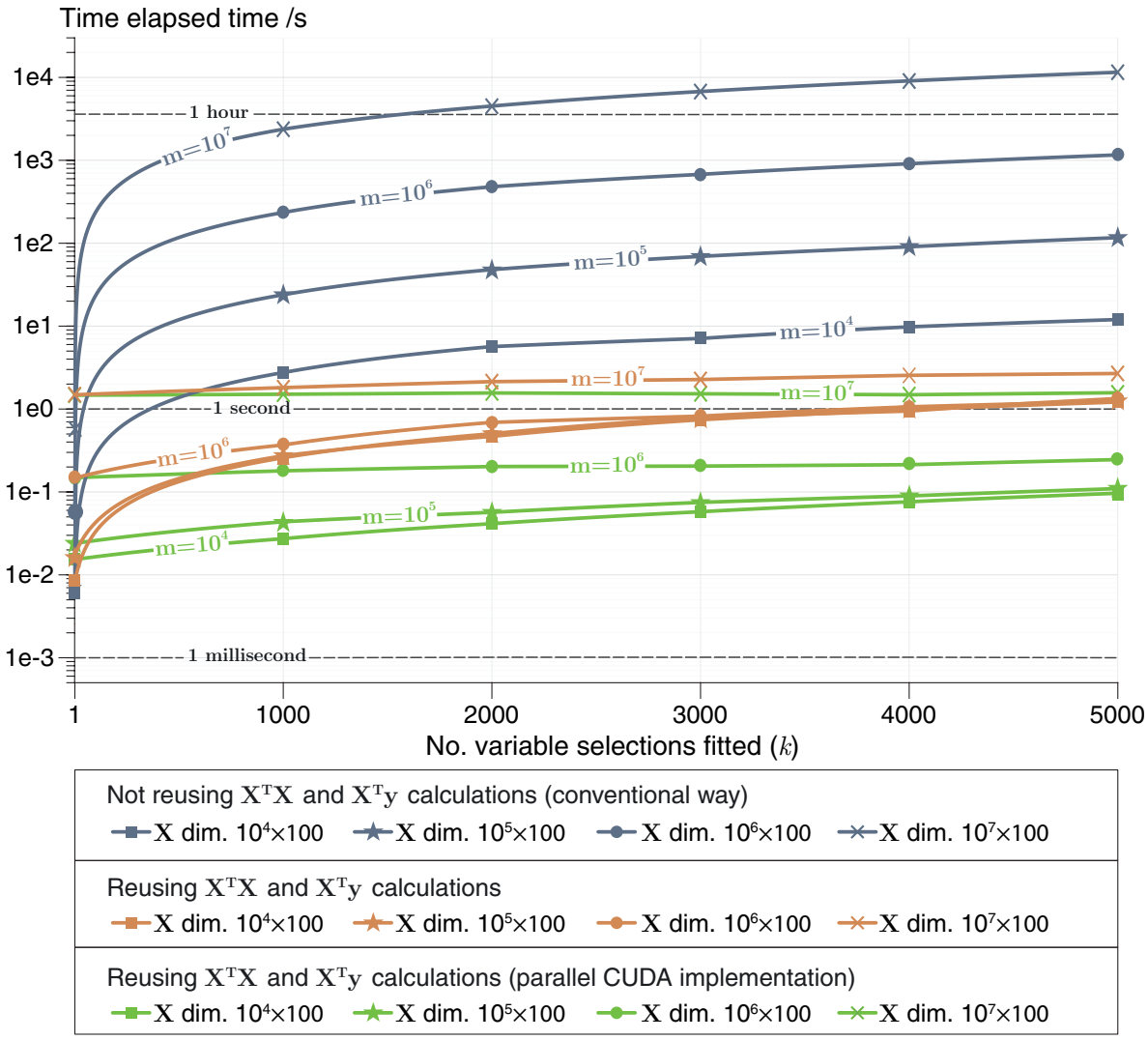

FIG. 2: Benchmark results from fitting batches of feature selections of varying sizes to random data using an improved version of the modified kernel #2 PLS algorithm with and without reusing calculations. Blue lines represents calculations performed according to algorithm 2, orange lines according to algorithm 3. CPU benchmarks were performed on an Intel i7-7700K @ 4.2 GHz. Green lines represents the parallel CUDA implementation of algorithm 3 executed on an Nvidia GTX1080ti @ 1.6 GHz. The maximum number of PLS components was set to 15 in all cases.

3.2. BENCHMARK OF COMMONLY USED FEATURE SELECTION METHODS

The results from the random search benchmark in section 3.1 provides a good overview of how the PLS calculation time scales with problem size. It does not, however, clearly convey what speedup one could expect in practice when implementing the indexing technique from section 2.2 together with commonly used non-random feature selection methods along with real data. In this section, three commonly used feature selection methods: forward selection, backward selection and a genetic algorithm, are therefore benchmarked together with a hyperspectral data set to fill this void. The data set used in the benchmark consists of six vis-NIR hyperspectral time series sequences where each sequence depicts a separate wood sample of the species Scots pine (*Pinus sylvestris*). Initially, each wood sample was submerged entirely under water and left to soak for 24 hours. After the soaking period the wood samples were taken from the water and placed one by one on a digital scale, which in turn was positioned underneath a hyperspectral camera. Over the course of roughly 21 hours, the absorbance of each wood sample was then monitored by the camera using 190 bands in the $500 - 1005$ nm region as the wood dried. In total, 843 hyperspectral images were taken and the absorbance spectra from all images are used as $\mathbf{X}$ in the data set. The digital scale placed underneath the wood samples was used to measure the weight of the wood samples as it decreased over time due to moisture evaporation. The sample weight was then recalculated into an average moisture content of each wood sample for each point in time. The time dependent moisture content is used as the response (y) in the data set. For more information on the data set the reader is referred to [11]. When performing regression on hyperspectral data, each pixel of the involved hyperspectral images can be viewed as a unique observation. When arranging such a data set into a two-dimensional design matrix, the number of rows (m) corresponds to the total number of pixels in all images—which can easily add up to several million or billion in number. To make the data set easier to work with, the spatial resolution of the original hyperspectral images can be lowered by averaging together neighboring pixels, resulting in an $\mathbf{X}$ matrix with any desired number of rows. During the benchmark in this section the spectral resolution (n) of the data set was kept constant at 190 bands, while the spatial resolution (m) of the design matrix was down-sampled to $0.5\text{e}3$, $1.0\text{e}3$, $0.5\text{e}4$, $1.0\text{e}4$, $0.5\text{e}5$, $1.0\text{e}5$, $0.5\text{e}6$ and $1.0\text{e}6$ respectively. In each feature selection algorithm a 10-fold cross-validation was performed and the cross-validated root mean square error,

$RMSE_{cv}$, was used to drive the search. In the forward and backward selection benchmark, the selection process was terminated as soon as an iteration caused the $RMSE_{cv}$ to increase. The genetic algorithm used a population size of 200 and ran for 200 generations before terminating. In all benchmarks the maximum number of considered PLS components was set to 15. The results from the three benchmarks are shown in Fig.3. As can be seen Fig.3, the calculation time required to perform backward selection and genetic algorithm was greatly decreased by the use of the indexing technique from section 2.2, while forward selection benefited substantially less from the indexing technique. Table 1 summarizes the average and maximum observed speedup of algorithm 2 compared to algorithm 1 for each of the feature selection methods in the benchmark. As demonstrated by the random search benchmark, the speedup is directly related to the number of subsets being evaluated, which differs greatly between the feature selection methods. In the case of the genetic algorithm, which terminated after a fixed number of generations, the number of subsets to evaluate is deterministic and often high compared to the other two methods, which is why it is natural for the GA to benefit a lot from the suggested method. In the case of forward- and backward selection, the number of subsets evaluated throughout the feature selection process depends on when the termination criterion is triggered, which in turn is data set specific. In the present example, backwards selection was sped up a great deal more by the suggested indexing technique than forward selection, it should be noted however, that this pattern could well be reversed for data sets that converges on a large number of active features. Furthermore, when performing calculations on a GPU there is always an overhead accosted with transferring data onto and from the device. When the computational workload is low, such as during the initial stages of a forward selection algorithm, the cost of transferring data to and from the GPU is too high to be completley amortized away by the offered parallelization of the PLS computation. This is why the CPU implementation of algorithm 2 can be seen to outperform the GPU implementation of algorithm 2 under some circumstances in the benchmark.

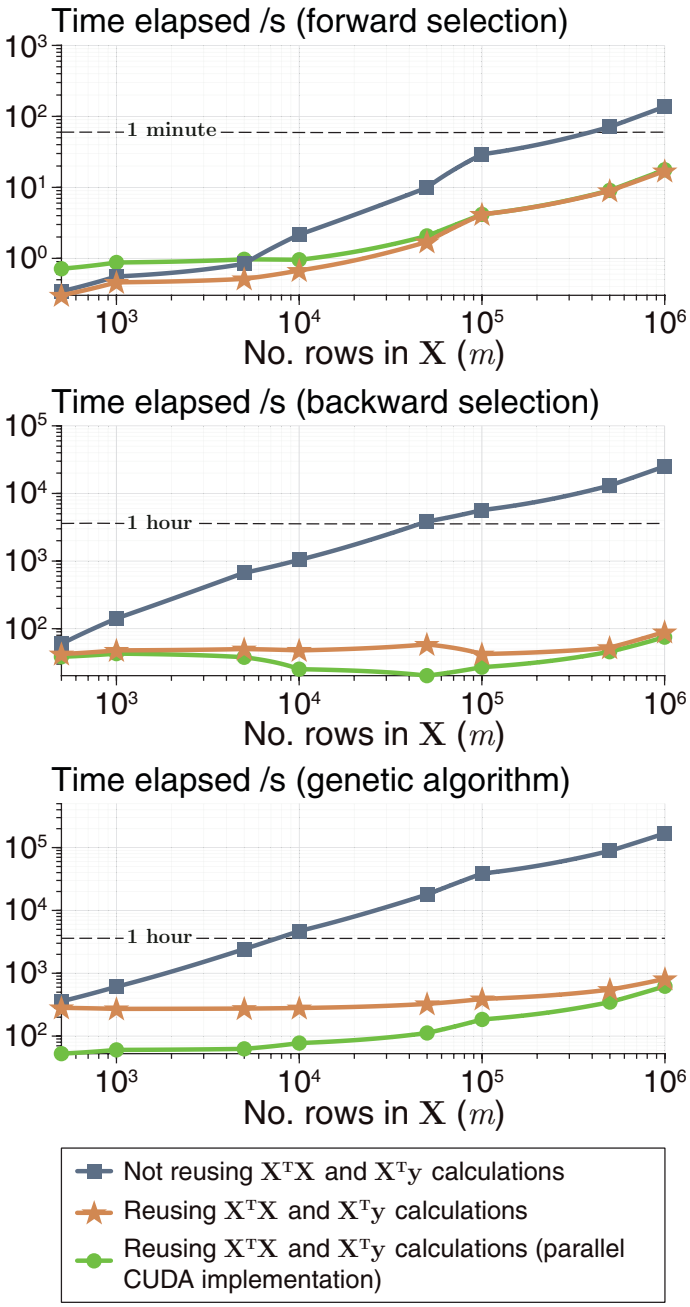

FIG. 3: Benchmark results from performing feature selection with kernel PLS on $\mathbf{X}$ matrices of a varying number of rows (m) using three commonly used feature selection algorithms; forward selection (upper), backward selection (middle) and genetic algorithm (lower). Blue lines represents feature selection performed using algorithm 2, orange lines according to algorithm 3. CPU benchmarks were performed on an Intel i5-6300HQ @ 2.3 GHz. Green lines represents the parallel CUDA implementation of algorithm 3 excecuted on an Nvidia GTX1080ti @ 1.6 GHz. The benchmark results only include the time elapsed when fitting regression coefficients $\hat{\beta}$ with kernel PLS, the additional time required to compute $RMSE_{cv}$ is not included since it is unrelated to the choice of PLS algorithm and identical in the three cases.

TABLE I: Average and maximum observed speedup of algorithm 2 compared to algorithm 1 for the three benchmarked feature selection methods.

Feature selection method	**Avg. speedup**	**Max. speedup**
Forward selection (CPU)	5x	8x
Forward selection (GPU)	4x	8x
Backward selection (CPU)	96x	281x
Backward selection (GPU)	136x	335x
Genetic algorithm (CPU)	69x	207x
Genetic algorithm (GPU)	127x	271x

4. CONCLUSIONS AND DISCUSSION

Many heuristic feature selection algorithms are largely driven by trial and error, because of this they tend to be rather time consuming - which creates a demand for computationally fast PLS fitting procedures such as kernel PLS. By taking advantage of a new simple indexing technique, the computational cost of fitting multiple variable subsets of the same data with PLS regression is substantially reduced. In cases were the design matrix $\mathbf{X}$ consists of a far greater number of observations than variables, we have demonstrated that the proposed technique offers a speedup of several orders of magnitude compared to the conventional approach when evaluating regression models for a large number of different variable subsets. The speedup is achieved by performing the computationally expensive covariance matrix calculations $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top y$ only once using the complete set of variables within $\mathbf{X}$ and then reusing the already calculated results for all subsequent feature subsets, rather can recalculating the covariances for each individual subset. It should be emphasized that the performance benefits of this technique becomes greater the larger the number of evaluated feature subsets becomes. In the special case of considering only one feature subset, the method offers no improvements at all since there are no calculations to reuse.

Because the success of many heuristic variable selection algorithms depend on the ability to explore a search space by evaluating the performance of a large number of subsets, the technique introduced here has the potential of improving essentially all wrapper-based feature selection methods by enabling more feature subsets to be evaluated per unit of time than previously possible. The kernel PLS algorithms are, however, not among the most numerically stable PLS alternatives. It is therefore recommended to recalibrate the most promising feature combination(s) by using a numerically more stable PLS algorithm, such as bidiag2 [10], before carefully evaluating, choosing and deploying the final model. It should also be mentioned that there are other flavors of PLS, such as sparse partial least squares regression (SPLS) which circumvents the need for conventional feature selection by producing sparse linear combinations of the original features within the algorithm [12]. Although SPLS requires the optimization of additional built-in parameters, it may in some cases—such as when the computational cost is not of critical importance or when the data set is small—be worthwhile to consider as a viable alternative approach.

Lastly it should also be mentioned that the indexing technique introduced in this paper is not limited to partial least squares regression. Indeed, in cases where solving an ordinary least squares regression problem through the normal equations is numerically acceptable, the indexing technique introduced in this paper is trivial to implement together with OLS and offers speedups on par with the ones demonstrated for PLS in section 3.

REFERENCES

[1] Y. Saeys, I. Inza and P. Larranaga, *A review of feature selection techniques in bioinformatics*. Bioinformatics, vol. 23, no. 19, pp. 2507-2517, 2007.

[2] T. Mehmood, K. Liland, L. Snipen and S. Sæbø, *A review of variable selection methods in Partial Least Squares Regression*. Chemometrics and Intelligent Laboratory Systems, vol. 118, pp. 62-69, 2012.

[3] Z. Xiaobo, Z. Jiewen, M. Povey, M. Holmes and M. Hanpin, *Variables selection methods in near-infrared spectroscopy*. Analytica Chimica Acta, vol. 667, no. 1-2, pp. 14-32, 2010.

[4] M. Anzanello and F. Fogliatto, *A review of recent variable selection methods in industrial and chemometrics applications*. European J. of Industrial Engineering, vol. 8, no. 5, p. 619, 2014.

[5] F. Lindgren, P. Geladi and S. Wold, *The kernel algorithm for PLS*. Journal of Chemometrics, vol. 7, no. 1, pp. 45-59, 1993.

[6] B. Dayal and J. MacGregor, *Improved PLS algorithms*. Journal of Chemometrics, vol. 11, no. 1, pp. 73-85, 1997.

[7] D. Kepplinger, P. Filzmoser, K. Varmuza, *Variable selection with genetic algorithms using repeated cross-validation of PLS regression models as fitness measure*. https://arxiv.org/pdf/1711.06695.pdf.

[8] N. Kettaneh, A. Berglund and S. Wold, *PCA and PLS with very large data sets*. Computational Statistics & Data Analysis, vol. 48, no. 1, pp. 69-85, 2005.

[9] S. Rännar, F. Lindgren, P. Geladi and S. Wold, *A PLS kernel algorithm for data sets with many variables and fewer objects. Part 1: Theory and algorithm*. Journal of Chemometrics, vol. 8, no. 2, pp. 111-125, 1994.

[10] Å. Björck and U. Indahl, *Fast and stable partial least squares modelling: A benchmark study with theoretical comments*. Journal of Chemometrics, vol. 31, no. 8, p. e2898, 2017.

[11] P. Stefansson, J. Fortuna, H. Rahmati, I. Burud, T. Konevskikh and H. Martens, *Hyperspectral time series analysis: Hyperspectral image data streams interpreted by modeling known and unknown variations*. Hyperspectral imaging. Analysis and applications. , 1st ed., 2018. [IN PRESS]

[12] H. Chun and S. Keleş, *Sparse partial least squares regression for simultaneous dimension reduction and variable selection*. Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 72, no. 1, p. 3-25, 2010.

Appendix 1: PLS coefficient estimation of a batch of feature selections with reused covariance

```
function [ Beta ] = PLSvarsel(X, y, A, VarSels)
% Filename: PLSvarsel.m
% Description: Matlab function which estimates regression coefficients for a batch of
    variable
% selections using the PLS algorithm 'Modified kernel algorithm #2'. For reference
    regarding
% the fitting sequence see: Improved PLS algorithms, Journal of chemometrics Vol. 11 p
     73-85.
% Inputs:
%        1. a [m-by-n] double-precision design matrix X.
%        2. a [m-by-1] double-precision response vector y.
%        3. a [1-by-1] double-precision scalar, A, specifying maximum PLS components.
%        4. a [k-by-n] logical matrix with k variable selections.
% Outputs:
%        1. a [n-by-A-by-k] array, Beta, with fitted coefficients for all feature
    selections.
%                Inactive variables are given a coefficient value of 0.
% Syntax:
%        Beta = PLSvarsel(X,y,A,VarSels);
%
% Written  2017-10-04 by Petter Stefansson.
% Modified 2018-03-01 by Ulf Indahl.
%% ------------------------- Calculate full covariance matrices X'X and X'y
    ------------------------

XX = X'*X;
```

```
21  Xy = X'*y;
22  % Memory allocation for Beta.
23  k = size(VarSels,1); n = size(X,2);
24  Beta = zeros(n,A,k);
25
26  % Loop over all variable selections.
27  for v = 1 : k
28      %% ------ Index into XX and Xy using a variable selection to acquire new
            covariances ------
29      smallXX = XX(VarSels(v,:),VarSels(v,:));
30      smallXy = Xy(VarSels(v,:));
31      smalln  = size(smallXX,1);
32      % Ensure number of PLS components <= number of variables.
33      if A > smalln;  MaxComps = smalln;  else;  MaxComps = A;  end
34
35      %% ---------- PLS on extracted covariances using Modified Kernel#2 algorithm
            ----------
36      % Memory allocation for matrices W, P, R and vector b.
37      W = nan(smalln,MaxComps);    P = nan(smalln,MaxComps);
38      R = nan(smalln,MaxComps);    b = zeros(smalln,1);
39
40      % PLS Component loop.
41      for i = 1 : MaxComps
42          w = smallXy - W(:,1:i-1)*(W(:,1:i-1)'*smallXy);
43          w = w/sqrt(w'*w);
44          r = w - R(:,1:i-1)*(P(:,1:i-1)'*w);
45          smallXXr = smallXX*r;
```

```
46          tt = r'*smallXXr;
47          p = smallXXr/tt;
48          q = (r'*smallXy)/tt;
49          smallXy = smallXy - smallXXr*q;
50          b = b + r*q;
51          W(:,i) = w;
52          R(:,i) = r;
53          P(:,i) = p;
54          Beta(VarSels(v,:),i,v) = b;
55      end
56  end
```