The Journal of Open Source Software

IKPLS: Improved Kernel Partial Least Squares and Fast Cross-Validation Algorithms for Python with CPU and GPU Implementations Using NumPy and JAX

Ole-Christian Galbo Engstrøm [1,2,3¶], **Erik Schou Dreier** [1], **Birthe Møller Jespersen** [4], **and Kim Steenstrup Pedersen** [2,5]

1 FOSS Analytical A/S, Denmark **2** Department of Computer Science (DIKU), University of Copenhagen, Denmark **3** Department of Food Science (UCPH FOOD), University of Copenhagen, Denmark **4** UCL University College, Denmark **5** Natural History Museum of Denmark (NHMD), University of Copenhagen, Denmark ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 25 January 2024
Published: unpublished

Summary

The `ikpls` software package introduces fast and versatile implementations of both versions of Improved Kernel Partial Least Squares (IKPLS) (Dayal & MacGregor, 1997). `ikpls` also contains the fast cross-validation algorithm presented by Engstrøm (O.-C. G. Engstrøm, 2024) combined with both IKPLS algorithms.

`ikpls` offers NumPy-based CPU and JAX-based CPU/GPU/TPU implementations. The JAX implementations are also differentiable, allowing seamless integration with deep learning techniques. This versatility enables users to handle diverse data dimensions efficiently.

Benchmarks demonstrate the superior performance of `ikpls` compared to scikit-learn's NIPALS across various scenarios. Additionally, the software addresses the redundant structure in cross-validation, proposing an algorithmic improvement for substantial speedup without recomputing total matrix products.

In conclusion, `ikpls` empowers researchers and practitioners in chemometrics and related fields with efficient, scalable, and end-to-end differentiable tools for PLS modeling, facilitating optimal component selection and preprocessing decisions by offering implementations of

1. both variants of IKPLS for CPUs;
2. both variants of IKPLS for GPUs, both of which are end-to-end differentiable, allowing integration with deep learning models;
3. a new algorithm for cross-validation that yields a substantial speedup if the training set is larger than the validation set.

Statement of need

PLS (partial least squares) (H. Wold, 1966) is a standard method in chemometrics. PLS can be used as a regression model, PLS-R (PLS regression), (S. Wold et al., 1983) (S. Wold et al., 2001) or a classification model, PLS-DA (PLS discriminant analysis), (Barker & Rayens, 2003). PLS takes as input a matrix $\mathbf{X}$ with dimension (N, K) of predictor variables and a matrix $\mathbf{Y}$ with dimension (N, M) of response variables. PLS decomposes $\mathbf{X}$ and $\mathbf{Y}$ into A latent variables (also called components), which are linear combinations of the original $\mathbf{X}$ and $\mathbf{Y}$. Choosing the optimal number of components, A, depends on the input data and varies from task to task. Additionally, selecting the optimal preprocessing method is challenging to assess before model validation (Rinnan et al., 2009) (Sørensen et al., 2021) but is required for achieving optimal performance (Du et al., 2022). The optimal number of components and the optimal preprocessing method are typically chosen by cross-validation, which may be very computationally expensive. The implementations of the fast cross-validation algorithm (O.-C. G. Engstrøm, 2024) will significantly reduce the computational cost of cross-validation.

This work introduces the Python software package, `ikpls`, with novel, fast implementations of Improved Kernel PLS (IKPLS) Algorithm #1 and Algorithm #2 by Dayal and MacGregor (Dayal & MacGregor, 1997), which have previously been compared with other PLS algorithms and shown to be fast (Alin, 2009) and numerically stable (Andersson, 2009). The implementations introduced in this work use NumPy (Harris et al., 2020) and JAX (Bradbury et al., 2018). The NumPy implementations can be executed on CPUs, and the JAX implementations can be executed on CPUs, GPUs, and TPUs. The JAX implementations are also end-to-end differentiable, allowing integration into deep learning methods. This work compares the execution time of the implementations on input data of varying dimensions. It reveals that choosing the implementation that best fits the data will yield orders of magnitude faster execution than the common NIPALS (Nonlinear Iterative Partial Least Squares) (H. Wold, 1966) implementation of PLS, which is the one implemented by scikit-learn (Pedregosa et al., 2011), an extensive machine learning library for Python. With the implementations introduced in this work, choosing the optimal number of components and the optimal preprocessing becomes much more feasible than previously. Indeed, derivatives of this work have previously been applied to do this precisely (O.-C. G. Engstrøm et al., 2023a) (O.-C. G. Engstrøm et al., 2023b).

Other implementations of other PLS algorithms with NumPy and scikit-learn exist, even for more specialized tasks such as multiblock PLS (Baum & Vermue, 2019). These implementations, however, are not as fast as IKPLS (Alin, 2009). Implementations of IKPLS exist in R and MATLAB. To the best of the authors' knowledge, however, there are no Python implementations of IKPLS that simultaneously correctly handle all possible dimensions of $\mathbf{X}$ and $\mathbf{Y}$. To the best of the authors' knowledge, no other PLS algorithms exist in JAX, nor do implementations of IKPLS in other frameworks with automatic differentiation.

Implementations

Improved Kernel PLS (Dayal & MacGregor, 1997) comes in two variants: Algorithm #1 and Algorithm #2. The implementations compute internal matrices $\mathbf{W}$ ($\mathbf{X}$ weights) of dimension (K, A), $\mathbf{P}$ ($\mathbf{X}$ loadings) of dimension (K, A), $\mathbf{Q}$ ($\mathbf{Y}$ loadings) of dimension (M, A), $\mathbf{R}$ ($\mathbf{X}$ rotations) of dimension (K, A) and a tensor $\mathbf{B}$ (regression coefficients) of dimension (A, K, M). Algorithm #1 also computes $\mathbf{T}$ ($\mathbf{X}$ scores) of dimension (N, A).

IKPLS (Dayal & MacGregor, 1997) offers two variants: Algorithm #1 and Algorithm #2, computing internal matrices such as $\mathbf{W}$ (X weights), $\mathbf{P}$ (X loadings), $\mathbf{Q}$ (Y loadings), $\mathbf{R}$ (X rotations), and a tensor $\mathbf{B}$ (regression coefficients). Algorithm #1 additionally computes $\mathbf{T}$ (X scores).

The `ikpls` package has been rigorously tested for equivalence against scikit-learn's NIPALS using NIR spectra data from (Dreier et al., 2022) and scikit-learn's PLS test-suite. Examples are provided for core functionalities, demonstrating fitting, predicting, cross-validating on CPU and GPU, and gradient propagation through PLS fitting.

NumPy

`ikpls` includes a Python class implementing both NumPy-based CPU IKPLS algorithms. It subclasses scikit-learn's `BaseEstimator`, facilitating integration with functions like `cross_validate`. Another class with IKPLS and fast cross-validation (O.-C. G. Engstrøm, 2024) is available.

JAX

For GPU/TPU acceleration, ikpls provides Python classes for each IKPLS algorithm using JAX. JAX combines Autograd (Maclaurin et al., 2015) with XLA (Accelerated Linear Algebra) for high-performance computation on various hardware. Automatic differentiation in forward and backward modes enables seamless integration with deep learning techniques, supporting user-defined metric functions.

Benchmarks

Benchmarks compare ikpls implementations with scikit-learn's NIPALS across varying data dimensions and component numbers. Single fits and leave-one-out cross-validation (LOOCV) scenarios are explored. To estimate execution time in a realistic scenario, the reported execution times for LOOCV include calibration of the PLS models and computation of the root mean squared error on the validation sample for all components from 1 to A.

The benchmarks use randomly generated data with fixed seeds for consistency. Default parameters are $N = 10,000$, $K = 500$, and $A = 30$, testing both single-target (PLS1) and multi-target (PLS2) scenarios.

The results in Figure 1 suggest CPU IKPLS for single fits, with a preference for IKPLS #2 if $N \gg K$. GPU usage is advised for larger datasets. In cross-validation, IKPLS options consistently outperform scikit-learn's NIPALS, with CPU IKPLS #2 (fast cross-validation) excelling, especially for large datasets. GPU IKPLS #1 is optimal in specific cases, considering preprocessing constraints. Fast cross-validation delivers significant speedup, more pronounced for IKPLS #2, especially when dealing with a larger number of target variables (M) (O.-C. G. Engstrøm, 2024).

In an attempt to give guidelines for algorithm choice for the most common use cases, we report the execution time of the implementations with varying values for each of the parameters above. Specifically, we define a list of values for each parameter to take while the rest of the parameters maintain their default settings. We use

$N \in [10^1, 10^2, 10^3, 10^4, 10^5, 10^6]$, $K \in [30, 50, 10^2, 5 \cdot 10^2, 10^3, 5 \cdot 10^3, 10^4]$, $A \in [10, 20, 30, 50, 100, 200, 500]$, and $M \in [1, 10]$.

All the experiments are executed on the hardware shown in Table 1 on a machine running Ubuntu 22.04 Jammy Jellyfish.

Table 1: Hardware used in the execution time experiments.

Component	Name
Motherboard	ASUS PRO WS X570-ACE
CPU	AMD Ryzen 9 5950X
CPU Cooler	NZXT Kraken X73
GPU	NVIDIA GeForce RTX3090 Ti, CUDA 11.8
RAM	4x32GB, DDR4, 3.2GHz, C16

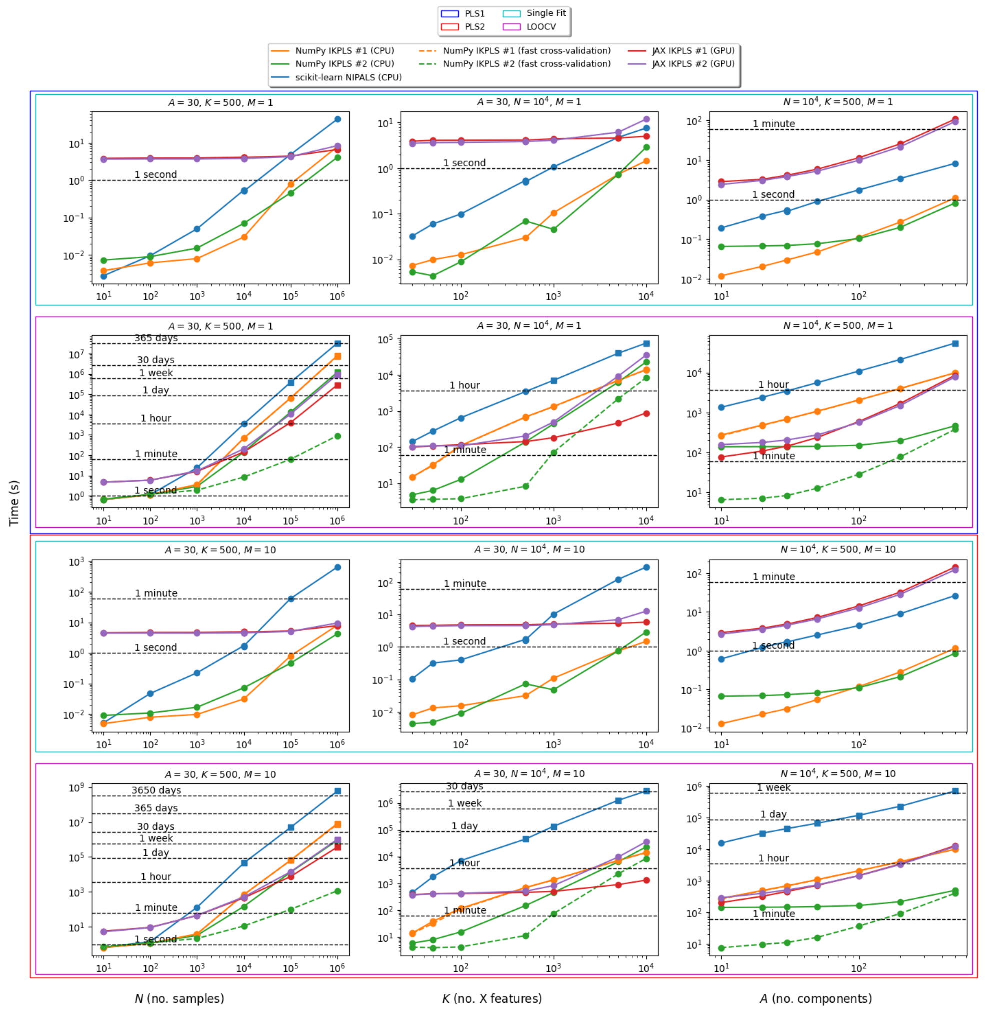
Figure 1: Results of our timing experiments. We vary N, K, and A in the first, second, and third columns. The first two rows are PLS1. The last two rows are PLS2. The first and third rows are single-fit. The second and fourth rows are leave-one-out cross-validation, computing the mean squared error and best number of components for each validation split. A circle indicates that the experiment was run until the end, and the time reported is exact. A square means that the experiment was run until the time per iteration had stabilized and used to forecast the time usage if the experiment was run to completion.

Acknowledgements

This work is part of an industrial Ph.D. project receiving funding from FOSS Analytical A/S and The Innovation Fund Denmark. Grant Number: 1044-00108B.

References

Alin, A. (2009). Comparison of PLS algorithms when number of objects is much larger than number of variables. *Statistical Papers*, *50*(4), 711.

Andersson, M. (2009). A comparison of nine PLS1 algorithms. *Journal of Chemometrics: A Journal of the Chemometrics Society*, *23*(10), 518–529.

Barker, M., & Rayens, W. (2003). Partial least squares for discrimination. *Journal of Chemometrics: A Journal of the Chemometrics Society*, *17*(3), 166–173.

Baum, A., & Vermue, L. (2019). Multiblock PLS: Block dependent prediction modeling for python. *Journal of Open Source Software*, *4*(34), 1190.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/google/jax

Dayal, B. S., & MacGregor, J. F. (1997). Improved PLS algorithms. *Journal of Chemometrics: A Journal of the Chemometrics Society*, *11*(1), 73–85.

Dreier, E. S., Sørensen, K. M., Lund-Hansen, T., Jespersen, B. M., & Pedersen, K. S. (2022). Hyperspectral imaging for classification of bulk grain samples with deep convolutional neural networks. *Journal of Near Infrared Spectroscopy*, *30*(3), 107–121.

Du, Z., Tian, W., Tilley, M., Wang, D., Zhang, G., & Li, Y. (2022). Quantitative assessment of wheat quality using near-infrared spectroscopy: A comprehensive review. *Comprehensive Reviews in Food Science and Food Safety*, *21*(3), 2956–3009.

Engstrøm, O.-C. G. (2024). *Shortcutting cross-validation: Efficiently deriving column-wise centered and scaled training set $\mathbf{X}^\mathbf{T}\mathbf{X}$ and $\mathbf{X}^\mathbf{T}\mathbf{Y}$ without full recomputation of matrix products or statistical moments*. https://arxiv.org/abs/2401.13185

Engstrøm, O.-C. G., Dreier, E. S., Jespersen, B. M., & Pedersen, K. S. (2023a). Improving deep learning on hyperspectral images of grain by incorporating domain knowledge from chemometrics. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 485–494.

Engstrøm, O.-C. G., Dreier, E. S., Jespersen, B. M., & Pedersen, K. S. (2023b). Analyzing near-infrared hyperspectral imaging for protein content regression and grain variety classification using bulk references and varying grain-to-background ratios. *arXiv Preprint arXiv:2311.04042*.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Maclaurin, D., Duvenaud, D., & Adams, R. P. (2015). Autograd: Effortless gradients in numpy. *ICML 2015 AutoML Workshop*, *238*, 5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rinnan, Å., Berg, F. van den, & Engelsen, S. B. (2009). Review of the most common pre-processing techniques for near-infrared spectra. *TrAC Trends in Analytical Chemistry*, *28*(10), 1201–1222.

Sørensen, K. M., Berg, F. van den, & Engelsen, S. B. (2021). NIR data exploration and regression by chemometrics—a primer. *Near-Infrared Spectroscopy: Theory, Spectral Analysis, Instrumentation, and Applications*, 127–189.

Wold, H. (1966). Estimation of principal components and related models by iterative least squares. *Multivariate Analysis*, 391–420.

Wold, S., Albano, C., Dunn, W. J., Esbensen, K., Hellberg, S., Johansson, E., Sjöström, M., Martens, H., & Russwurm, J. (1983). Food research and data analysis. *London: H. Martens and H. Russwurn Jr.*

Wold, S., Sjöström, M., & Eriksson, L. (2001). PLS-regression: A basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, *58*(2), 109–130.