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, Erik Schou Dreier, Birthe Møller Jesperersen, and Kim Steenstrup Pedersen

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

DOI: 10.21105/joss.06533

Summary

The ikpls software package provides fast and efficient tools for PLS (Partial Least Squares) modeling. This package is designed to help researchers and practitioners handle PLS modeling faster than previously possible - particularly on large datasets. The PLS implementations in ikpls use the fast IKPLS (Improved Kernel PLS) algorithms (Dayal & MacGregor, 1997), providing a substantial speedup compared to scikit-learn’s (Pedregosa et al., 2011) PLS implementation, which is based on NIPALS (Nonlinear Iterative Partial Least Squares) (H. Wold, 1966). The ikpls package also offers an implementation of IKPLS combined with the fast cross-validation algorithm by O.-C. G. Engstrøm (2024), significantly accelerating cross-validation of PLS models - especially when using a large number of cross-validation splits.

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.

In conclusion, ikpls empowers researchers and practitioners in machine learning, 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. IKPLS combined with a cross-validation algorithm that yields a substantial speedup compared to the classical cross-validation algorithm.

Statement of need

PLS (H. Wold, 1966) is a standard method in machine learning and chemometrics. PLS can be used as a regression model, PLS-R (PLS regression) (S. Wold et al., 1983, 2001), or a classification model, PLS-DA (PLS discriminant analysis) (Barker & Rayens, 2003). PLS takes as input a matrix X with dimension \((N, K)\) of predictor variables and a matrix Y with dimension \((N, M)\) of response variables. PLS decomposes X and Y into \(A\) latent variables (also called components), which are linear combinations of the original X and 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
This work introduces the Python software package, `ikpls`, with novel, fast implementations of IKPLS Algorithm #1 and Algorithm #2 by Dayal & MacGregor (1997), which have previously been compared with other PLS algorithms and shown to be fast (Alin, 2009) and numerically stable (Anderson, 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.

Implementations

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

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>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motherboard</td>
<td>ASUS PRO WS X570-ACE</td>
</tr>
<tr>
<td>CPU</td>
<td>AMD Ryzen 9 5950X</td>
</tr>
<tr>
<td>CPU Cooler</td>
<td>NZXT Kraken X73</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA GeForce RTX3090 Ti, CUDA 11.8</td>
</tr>
<tr>
<td>RAM</td>
<td>4x32GB, DDR4, 3.2GHz, C16</td>
</tr>
</tbody>
</table>

Table 1: Hardware used in the execution time experiments.
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


