The Journal of Open Source Software

open_nipals: An sklearn-compatible python package for NIPALS dimensional reduction

Niels Schlusser [1*], **Ryan Wall** [2*], and **David R. Ochsenbein** [1]

1 Cilag GmbH International, Schaffhausen, Switzerland **2** Johnson & Johnson Innovative Medicine, Titusville, New Jersey, USA * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Yuanqing Wang

Reviewers:

- @maximtrp
- @rsenne
- @ankur-gupta

Submitted: 28 April 2025
Published: unpublished

Summary

open_nipals is a python package that implements the Nonlinear Iterative Partial Least Squares (NIPALS) algorithm (Geladi & Kowalski, 1986) for Partial Least Squares (PLS) regression as well as Principle Component Analysis (PCA). It employs the data transformation methods `fit()` and `transform()` from scikit-learn (Pedregosa et al., 2011) and leverages Nelson's Single Component Projection (SCP) method for the imputation of missing data (Nelson et al., 1996). The NIPALS algorithm represents an alternative to the common Singular Value Decomposition (SVD) procedure for both PCA and PLS implemented in `scikit-learn` (Pedregosa et al., 2011). It is an iterative procedure that processes the data and internal matrices vector-wise and iteratively. When combined with SCP, NIPALS allows natural handling of missing data and setting tailored accuracy goals.

Statement of Need

Python has emerged as a popular and comparatively simple programming environment for the development of machine learning and data science applications. Packages like `numpy` for vector operations (Harris et al., 2020), `pandas` for the handling of tabular data (team, 2020), and `scikit-learn` for orthodox machine learning techniques like Random Forests, Support Vector Machines (SVM), and Principal Component Analyses (PCA) (Pedregosa et al., 2011) promote python's success in extracting patterns from big and complex data sets. However, `scikit-learn` relies on Singular Value Decomposition (SVD) for its PCA and PLS classes, with negative effects on performance for applications like batch manufacturing, where missing data is common (Nelson et al., 1996). PCA and PLS models require unit scaled and mean centered input data, a feature that is nicely implemented in `scikit-learn`'s `StandardScaler` class.
To this end, we felt the need to complement `scikit-learn` with an implementation of the NIPALS algorithm for PCA and PLS.

Related Software

To our knowledge, the only other maintained open-source python package that implements the NIPALS algorithm for PCA and PLS is Salvador García Muñoz' `pyphi` (Garcia Munoz et al., 2019). Our implementation is different in the following aspects: 1. `open_nipals` follows the template of `scikit-learn`, which allows: 1. Integration with other `scikit-learn` modules, e.g. the `StandardScaler` 2. Accumulation of multiple transformation steps into a `sklearn.pipeline`. 2. `open_nipals` uses Nelson's single component projection method (Nelson et al., 1996) for score calculation in the face of missing values. 3. The utility class of `open_nipals` contains `ArrangeData`, another `scikit-learn` style data transformer object that ensures correct ordering and quantity of input columns.

Functionality

Wherever possible, open_nipals inherits structures from parent classes in scikit-learn. In principle, its functionality can be split into three parts: 1. Utility functions for data preprocessing 2. Principal Component Analysis 3. Partial Least Squares regression.

We decided to combine PCA and PLS functionality into one package, such that they can share common utility functions, e.g. – but not limited to – the ArrangeData class and matrix multiplication with missing values.

Data Preprocessing, and Utility Functions

It is strongly encouraged to mean-center the input data for both PCA and PLS, and scale their variance to unity, e.g. with sklearn's StandardScaler. Moreover, the ArrangeData class of open_nipals ensures correct ordering of the input columns, as well as proper formatting. A code example for preprocessing could therefore look like:

```
from open_nipals.utils import ArrangeData
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load data
df = pd.read_csv('my_data.csv')

# Create objects
arrdat = ArrangeData()
scaler = StandardScaler()

# Fit and transform data using both objects
data = scaler.fit_transform(arrdat.fit_transform(df))
```

PCA

Principal Component Analyses with open_nipals utilize a NipalsPCA transformer object, that can be fitted to and transform input data (and both at once), e.g. with:

```
from open_nipals.nipalsPCA import NipalsPCA

model = NipalsPCA()
transformed_data = model.fit_transform(data)
```

The number of fitted components can be specified with the n_components argument in the constructor, which defaults to n_components=2. After having constructed the object, components can be added or subtracted using the set_components() function. Once fitted, components are stored so they do not have to be fitted again. This saves compute time should the developer decide to use lower number of components than are fitted and later move back to a higher number of principal components. A pseudo-inverse can be calculated with the inverse_transform() function. The distance of a given data point from the average of the training data within the PCA model (in-model distance, IMD) can be calculated with calc_imd(), where $\mathrm{Hotelling's\,T^2}$ (Hotelling, 1931) is implemented and could be extended to other IMD metrics (e.g. Mahalanobis Distance). Conversely, the out-of-model distance (OOMD, calculated by calc_oomd()) gives a measure of the distance to the model hyperplane. This is available as two metrics, DModX and QRes (Eriksson et al., 1999).

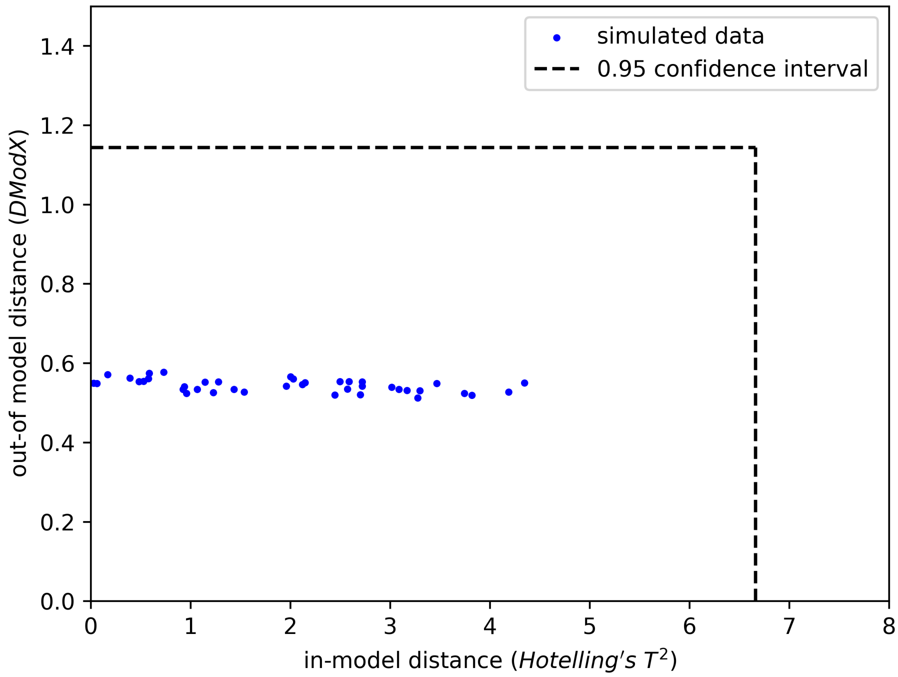

Figure 1: Figure 1: PCA-modelled data points on the IMD-OOMD plane with 0.95 confidence interval.

Finally, the `calc_limit()` function calculates theoretical limits on both IMD and OOMD such that a specified fraction `alpha` of the data lies within these limits. The IMD-OOMD plot (see (**?**)) is assumed to follow an f-distribution (Brereton, 2016).

PLS

Similarly, Partial Least Squares regressions require a `NipalsPLS` object. Basic functionality includes the `fit()`, `transform()`, and `fit_transform()` methods:

```
from open_nipals.nipalsPLS import NipalsPLS

model = NipalsPLS()
transformed_x_data, transformed_y_data = model.fit_transform(data_x, data_y)
```

`NipalsPLS` similarly contains a pseudo-inverse tranform that returns simulated data given a set of PLS scores with `inverse_transform()`, `calc_oomd()` for the out-of-model distance with either `QRes` or `DModX` as implemented metrics, `calc_imd()` for the in-model distance, using the $\mathrm{Hotelling's\,T^2}$ metric. `NipalsPLS` differs primarily from `NipalsPCA` by the inclusion of a `predict()` method to predict a y-matrix from an x-matrix with a previously fitted model, and the calculation of the regression vector with `get_reg_vector()`.

Availability

`open_nipals` is available open-source under APACHE 2.0 license from this github repository (Ochsenbein et al., 2025). We appreciate your feedback and contributions.

Acknowledgements

We acknowledge support by Johnson & Johnson Innovative Medicine. In particular, we would like to express our gratitude to Samuel Tung and Tyler Roussos of the Open Source Working Group (OSWG) within J&J, who helped driving the publication process of `open_nipals`.

References

Brereton, R. G. (2016). Hotelling's t squared distribution, its relationship to the f distribution and its use in multivariate space. *Journal of Chemometrics*, *30*(1), 18–21. https://doi.org/10.1002/cem.2763

Eriksson, L., Johansson, E., Kettaneh-Wold, N., & Wold, S. (1999). *Introduction to multi- and megavariate data analysis using projection methods (PCA and PLS)* (p. 468). Umetrics.

Garcia Munoz, S., Codgers, J. A., & Johnson, B. J. (2019). *pyphi - A python package for multivariate analysis* (Version 4.1). https://github.com/salvadorgarciamunoz/pyphi

Geladi, P., & Kowalski, B. R. (1986). Partial least-squares regression: A tutorial. *Analytica Chimica Acta*, *185*, 1–17. https://doi.org/10.1016/0003-2670(86)80028-9

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., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hotelling, H. (1931). The Generalization of Student's Ratio. *The Annals of Mathematical Statistics*, *2*(3), 360–378. https://doi.org/10.1214/aoms/1177732979

Nelson, P. R. C., Taylor, P. A., & MacGregor, J. F. (1996). Missing data methods in PCA and PLS: Score calculations with incomplete observations. *Chemometrics and Intelligent Laboratory Systems*, *35*(1), 45–65. https://doi.org/10.1016/S0169-7439(96)00007-X

Ochsenbein, D. R., Schlusser, N., & Wall, R. (2025). *open_nipals* (Version 1.0). https://github.com/johnsonandjohnson/open_nipals

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.

team, T. pandas development. (2020). *Pandas-dev/pandas: pandas* (latest). Zenodo. https://doi.org/10.5281/zenodo.3509134