direpack

Release 1.0.10

Sven Serneels and Emmanuel Jordy Menvouta

Jun 23, 2023

CONTENTS

The direpack package aims to establish a set of modern statistical dimension reduction techniques into the Python universe as a single, consistent package. The dimension reduction methods included resort into three categories: projection pursuit based dimension reduction, sufficient dimension reduction, and robust M estimators for dimension reduction. As a corollary, regularized regression estimators based on these reduced dimension spaces are provided as well, ranging from classical principal component regression up to sparse partial robust M regression. The package also contains a set of classical and robust pre-processing utilities, including generalized spatial signs, as well as dedicated plotting functionality and cross-validation utilities. Finally, direpack has been written consistent with the scikit-learn API, such that the estimators can flawlessly be included into (statistical and/or machine) learning pipelines in that framework.

CHAPTER

ONE

INSTALLATION

The package is distributed through PyPI, so use:

```
pip install direpack
```

CHAPTER

TWO

EXAMPLES

Example notebooks have been produced to showcase the use of direpack for statistical dimension reduction. These notebooks contain a ppdire example , sprm example and a sudire example .

CHAPTER

THREE

CONTENTS

3.1 ppdire

Beyond discussion, the class of dimension reduction with the longest standing history accessible through direpack, is projection pursuit (PP) dimension reduction. Let $\mathbf{X}$ be a data matrix that is a sample of n cases of a p variate random variable and $\mathbf{y}$ be a sample of a corresponding depending variable, when applicable. The set of projection pursuit scores $\mathbf{t}_i$ that span the columns of $\mathbf{T}$ are defined as linear combinations of the original variables: $\mathbf{T} = \mathbf{XW}$, where the $\mathbf{w}_i$ are the solution to the optimization problem:

$$\begin{aligned} \underset{\mathbf{a}}{\text{maximise}} \quad & \mathfrak{P}\left(\mathbb{S}\left(\mathbf{a}^T\mathbf{X}\right)\right) \\ \text{subject to} \quad & \mathbf{w}_i^T\mathbf{X}^T\mathbf{X}\mathbf{w}_j = 0 \text{ and } \parallel \mathbf{w}_i \parallel_2 = 1, \end{aligned}$$

where $i, j \in [1, \min(n, p)]$, $j > i$ and the set $\mathbb{S} = \{\mathbf{X}, \mathbf{y}\}$ if data for a dependent variable Y exist and is a singleton containing $\mathbf{X}$ otherwise. Maximization of this criterion is very flexible and the properties of the dimension reduction accomplished according to it can vary widely, mainly dependent on the presence or absence of dependent variable data, as well as on $\mathfrak{P}$, which in the PP literature is referred to as the projection index.

3.1.1 dicomo

The projection index determines which method is being calculated. In direpack, projection pursuit can be called through the ppdire subpackge and class object, which allows the user to pass any function of appropriate dimensionality as a projection index. However, a set of popular projection indices deriving from (co-)moments, are provided as well through the dicomo subpackage. For several of these, plugging them in leads to well-established methods. They comprise:

- Moment statistics: variance (PCA), higher order moments
- Co-moment statistics: covariance (PLS), higher order co-moments
- Standardized moments: skewness (ICA), kurtosis (ICA)
- Standardized co-moments: correlation coefficient (CCA), co-skewness, co-kurtosis
- Linear combinations of (standardized co-) moments. Here, the capi.py file in the ppdire subpackage delivers to co-moment analysis projection index (Serneels2019).
- Products of (co-)moments. Particularly the continuum association measure has been provided, which is given by $\text{cont}(\mathbf{X}, \mathbf{y}) = \text{cov}(\mathbf{X}, \mathbf{y})\ \text{var}(\mathbf{X})^{\alpha-1}$. Using this continuum measure produces continuum regression (CR, Stone and Brooks (1990)). CR is equivalent to PLS for $\alpha = 1$ and approaches PCA as $\alpha \to \infty$.

3.1.2 pp optimizers

Early ideas behind PP was the ability to scan all directions maximizing the projection index as denoted in (3.1). This essentially corresponds to a brute force optimization technique, which can be computationally very demanding. For instance, both PCA and PLS, can be solved analytically, leading to efficient algorithms that do not directly optimize (3.1). Whenever the projection index plugged in, leads to a convex optimization problem, it is advisable to apply an efficient numerical optimization technique. For that purpose,ppdire has the option to use scipy.optimize's sequential least squares quadratic programming optimization (SLSQP). However, for projection indices based on ordering or ranking data, such as medians or trimmed (co-)moments, the problem is no longer convex and cannot be solved through SLSQP. For those purposes, the grid algorithm is included, which was originally developed to compute RCR (Filzmoser, Serneels, Croux, andVan Espen 2006).

3.1.3 Regularized regression

While the main focus of direpack is dimension reduction, all dimension reduction techniques offer a bridge to regularized regression. This can be achieved by regressing the dependent variable onto the estimated dimension reduced space. The latter provides regularization of the covariance matrix, due to the constraints in (3.1), and allow to perform regression for an undersampled $\mathbf{X}$. The classical estimate is to predict $\mathbf{y}$ through least squares regression:

$$\hat{\mathbf{y}} = \hat{\mathbf{T}}\hat{\mathbf{T}}^T\mathbf{y}$$

which again leads to well-established methods such as principal component regression (PCR), PLS regression, etc.

3.1.4 Usage

ppdire(projection_index[, pi_arguments, ...])	PPDIRE Projection Pursuit Dimension Reduction

direpack.ppdire.ppdire.ppdire

class ppdire(*projection_index*, *pi_arguments={}*, *n_components=1*, *trimming=0*, *alpha=1*, *optimizer='SLSQP'*, *optimizer_options={'maxiter': 100000}*, *optimizer_constraints=None*, *regopt='OLS'*, *center='mean'*, *center_data=True*, *scale_data=True*, *whiten_data=False*, *square_pi=False*, *compression=False*, *copy=True*, *verbose=True*, *return_scaling_object=True*)

PPDIRE Projection Pursuit Dimension Reduction

The class allows for calculation of the projection pursuit optimization either through *scipy.optimize* or through the grid algorithm, native to this package. The class provides a very flexible way to access optimization of projection indices that can lead to either classical or robust dimension reduction. Optimization through scipy.optimize is much more efficient, yet it will only provide correct results for classical projection indices. The native grid algorithm should be used when the projection index involves order statistics of any kind, such as ranks, trimming, winsorizing, or empirical quantiles. The grid optimization algorithm for projection pursuit implemented here, was outlined in:

> Filzmoser, P., Serneels, S., Croux, C. and Van Espen, P.J., Robust multivariate methods: The projection pursuit approach, in: From Data and Information Analysis to Knowledge Engineering, Spiliopoulou, M., Kruse, R., Borgelt, C., Nuernberger, A. and Gaul, W., eds., Springer Verlag, Berlin, Germany, 2006, pages 270–277.

Parameters

projection_index – dicomo and capi supplied in this package can both be used, but user defined projection indices can be processed

Attributes always provided

- *x_weights_*: X block PPDIRE weighting vectors (usually denoted W)
- *x_loadings_*: X block PPDIRE loading vectors (usually denoted P)
- *x_scores_*: X block PPDIRE score vectors (usually denoted T)
- *x_ev_*: X block explained variance per component
- *x_Rweights_*: X block SIMPLS style weighting vectors (usually denoted R)
- *x_loc_*: X block location estimate
- *x_sca_*: X block scale estimate
- *crit_values_*: vector of evaluated values for the optimization objective.
- *Maxobjf_*: vector containing the optimized objective per component.

Attributes created when more than one block of data is provided

- *C_*: vector of inner relationship between response and latent variables block
- *coef_*: vector of regression coefficients, if second data block provided
- *intercept_*: intercept
- *coef_scaled_*: vector of scaled regression coefficients (when scaling option used)
- *intercept_scaled_*: scaled intercept
- *residuals_*: vector of regression residuals
- *y_ev_*: y block explained variance
- *fitted_*: fitted response
- *y_loc_*: y location estimate
- *y_sca_*: y scale estimate

Attributes created only when corresponding input flags are `True`

- *whitening_*: whitened data matrix (usually denoted K)
- *mixing_*: mixing matrix estimate
- *scaling_object_*: scaling object from *VersatileScaler*

__init__(*projection_index*, *pi_arguments={}*, *n_components=1*, *trimming=0*, *alpha=1*, *optimizer='SLSQP'*, *optimizer_options={'maxiter': 100000}*, *optimizer_constraints=None*, *regopt='OLS'*, *center='mean'*, *center_data=True*, *scale_data=True*, *whiten_data=False*, *square_pi=False*, *compression=False*, *copy=True*, *verbose=True*, *return_scaling_object=True*)

Methods

__init__(projection_index[, pi_arguments, ...])	

fit(X, *args, **kwargs)	Fit a projection pursuit dimension reduction model.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator. :param deep: If True, will return the parameters for this estimator and contained subobjects that are estimators. :type deep: boolean, optional.
predict(Xn)	predicts the response on new data Xn
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator. Copied from ScikitLearn, adapted to avoid calling 'deep=True' :returns: * *self* * ------ * *Copied from ScikitLlearn instead of imported to avoid 'deep=True'*.
transform(Xn)	Computes the dimension reduction of the data Xn based on the fitted sudire model.

Attributes

dicomo([est, mode, center])	The *dicomo* class implements (co)-moment statistics, covering both clasical product-moment statistics, as well as more recently developed energy statistics.

direpack.dicomo.dicomo.dicomo

class dicomo(*est='arithmetic'*, *mode='mom'*, *center='mean'*)

The *dicomo* class implements (co)-moment statistics, covering both clasical product-moment statistics, as well as more recently developed energy statistics. The *dicomo* class also serves as a plug-in into *capi* and *ppdire*. It has been written consistently with *ppdire* such that it provides a wide range of projection indices based on (co-)moments. Ancillary functions for (co-)moment estimation are in *_dicomo_utils.py*.

Parameters

- **est** (*str*) – mode of estimation. The set of options are *'arithmetic'* (product-moment) or *'distance'* (energy statistics)
- **mode** (*str*) – type of moment. Options include : * *'mom'*: moment * *'var'*: variance * *'std'*: standard deviation * *'skew'*: skewness * *'kurt'*: kurtosis * *'com'*: co-moment * *'M3'*: shortcut for third order co-moment * *'cov'*: covariance * *'cos'*: co-skewness * *'cok'*: co-kurtosis * *'corr'*: correlation, * *'continuum'*: continuum association * *'mdd'*: martingale difference divergence (requires *est = 'distance'*) * *'mdc'*: martingale difference correlation (requires *est = 'distance'*) * *'ballcov'*: ball covariance (requires installing *Ball* and uncommenting the *import* statement)
- **center** (*str*) – internal centring used in calculation. Options are *mean* or *median*.

Attributes always provided

- *moment_*: The resulting (co-)moment Depending on the options picked, intermediate results are stored as well, such as *x_moment_*, *y_moment_* or *co_moment_*

__init__(*est='arithmetic'*, *mode='mom'*, *center='mean'*)

Methods

__init__([est, mode, center])	

fit(x, **kwargs)	Fit a dicomo model
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.

Attributes

3.1.5 Dependencies

- From *sklearn.base*: *BaseEstimator*,`TransformerMixin`,`RegressorMixin`
- From *sklearn.utils*: *_BaseComposition*
- *copy*
- *scipy.stats*
- From *scipy.linalg*: *pinv2*
- From *scipy.optimize*: *minimize*
- *numpy*
- From *statsmodels.regression.quantile_regression*: *QuantReg*
- From *sklearn.utils.extmath*: *svd_flip*

3.1.6 References

1. Peter Filzmoser, Sven Serneels, Christophe Croux and Pierre J. Van Espen, Robust Multivariate Methods: The Projection Pursuit Approach, in: From Data and Information Analysis to Knowledge Engineering,Spiliopoulou, M., Kruse, R., Borgelt, C., Nuernberger, A. and Gaul, W., eds., Springer Verlag, Berlin, Germany, 2006, pages 270–277.
2. Sven Serneels, Projection pursuit based generalized betas accounting for higher order co-moment effects in financial market analysis, in: JSM Proceedings, Business and Economic Statistics Section. Alexandria, VA: American Statistical Association, 2019, 3009-3035.
3. Chen, Z. and Li, G., Robust principal components and dispersion matrices via projection pursuit, Research Report, Department of Statistics, Harvard University, 1981.
4. Peter Filzmoser, Christophe Croux, Pierre J. Van Espen, Robust Continuum Regression, Sven Serneels, Chemometrics and Intelligent Laboratory Systems, 76 (2005), 197-204.

5. Stone M, Brooks RJ (1990). "Continuum Regression: Cross-Validated Sequentially Constructed Prediction Embracing Ordinary Least Squares, Partial Least Squares and PrincipalComponents Regression."Journal of the Royal Statistical Society. Series B (Methodological),52, 237–269.

3.2 sudire

Sufficient dimension reduction (SDR) is a recent take on dimension reduction, where one aims to estimate a set of latent variables that are linear combinations of the original variables $\mathbf{T} = \mathbf{XW}$ in such a way that the subspace spanned by them contains all information relevant to the dependent variable in such a way that the subspace spanned by them contains all information relevant to the dependent variable: $\mathbf{Y}\mathbf{X} \mid \mathbf{T}$. Here, $\mathbf{X}$ is a sample of n cases of a p variate random variable and $\mathbf{Y}$ is a sample of the dependent variable, $\mathbf{W}$ is a $p \times q$ matrix with $q \leq p$, and $\perp\!\!\!\perp$ denotes statistical independence. A lot of research has been done over the last thirty years investigating different approaches in terms of asymptotics and assumptions made in each of the approaches. A good textbook providing an overview of approaches to SDR is Li (2018). The subpackage sudire contains implementations of a broad set of these approaches.

Generally speaking, SDR techniques roughly resort in three categories. At first, there is a successful set of approaches to SDR based on slicing the original space. Examples of these are sliced inverse regression (SIR, Li (1991)) and sliced-average variance estimation (SAVE, Cook (2000)). A second group of developments has involved selective focus on certain directions, which has resulted in, among others, directional regression (DR, Li (2007)), principal Hessian directions (PHD, Li (1992)) and the iterative Hessian transformations (IHT, Cook and Li (2002)).

While all of the aforementioned methods are included in sudire and would merit a broader discussion, at this point we would like to highlight that sudire contains implementations of a more recent approach as well. The latter has, so far, resulted in three methods, all three of which share the following advantages: they do not require conditions of linearity or constant covariance, nor do they need distributional assumptions, yet they may be computationally more demanding. This third group of SDR algorithms estimates a basis of the central subspace as:

$$\begin{aligned} \mathbf{W}_h =_{\mathbf{B}} \quad & \mathfrak{P}^2\left(\mathbf{XB}, \mathbf{Y}\right) \\ \text{subject to} \quad & \mathbf{B}^T\mathbf{X}^T\mathbf{XB} = \mathbf{I}_h, \end{aligned}$$

where $\mathbf{B}$ is an arbitrary $p \times h$ matrix, $h \in [1, \min(n, p)]$. Here, $\mathfrak{P}$ can be any statistic, that estimate a subspace whose complement is independent of $\mathbf{Y}$. Currently implemented $\mathfrak{P}$ statistics are :

- distance covariance (Székely, Rizzo, and Bakirov 2007), leading to option dcov-sdr (Sheng and Yin 2016);
- martingale difference divergence (Shao and Zhang 2014), leading to option mdd-sdr (Zhang, Liu, Wu, and Fang 2019);
- ball covariance (Pan, Wang, Xiao, and Zhu 2019), leading to option bcov-sdr (Zhang and Chen 2019)

3.2.1 Usage

sudire([sudiremeth, n_components, trimming, ...])	SUDIRE Sufficient Dimension Reduction

direpack.sudire.sudire.sudire

class sudire(*sudiremeth='dcov-sdr'*, *n_components=2*, *trimming=0*, *optimizer_options={'max_iter': 1000}*, *optimizer_constraints=None*, *optimizer_arguments=None*, *optimizer_start=None*, *center_data=True*, *center='mean'*, *scale_data=True*, *whiten_data=False*, *compression=False*, *n_slices=6*, *dmetric='euclidean'*, *fit_ols=True*, *copy=True*, *response_type='continuous'*, *verbose=True*, *return_scaling_object=True*)

SUDIRE Sufficient Dimension Reduction

The class allows for Sufficient Dimension Reduction using a variety of methods. If the method requires optimization of a function, This optimization is done through the Interior Point Optimizer (IPOPT) algorithm.

Parameters

- **sudiremeth** (*function or class. sudiremeth in this package can also be used,*) –
- **are** (*but user defined functions can be processed. Built in options*) – save : Sliced Average Variance Estimation

  sir : Slices Inverse Regression

  dr : Directional Regression

  iht : Iterative Hessian Transformations

  dcov-sdr : SDR via Distance Covariance

  mdd-sdr : SDR via Martingale Difference Divergence.

  bcov-sdr : SDR via ball covariance
- **n_components** (*int*) – dimension of the central subspace.
- **trimming** (*float*) – trimming percentage to be entered as pct/100
- **optimizer_options** (*dict*) – with options to pass on to the optimizer.Includes:
- **max_iter** (*int*) – Maximal number of iterations.
- **tol** (*float*) – relative convergence tolerance
- **constr_viol_tol** (*float*) – Desired threshold for the constraint violation.
- **optimizer_constraints** (*dict or list of dicts*) – further constraints to be passed on to the optimizer function.
- **optimizer_arguments** (*dict*) – extra arguments to be passed to the sudiremeth function during optimization.
- **optimizer_start** (*numpy array*) – starting value for the optimization.
- **center** (*str*) – how to center the data. options accepted are options from sprm.preprocessing
- **center_data** (*bool*) – If True, the data will be centered before the dimension reduction
- **scale_data** (*bool*) – if set to False, convergence to correct optimum is not a given. Will throw a warning.
- **compression** (*bool*) – Use internal data compresion step for flat data.
- **n_slices** (*int*) – The number of slices for SAVE, SIR, DR
- **is_distance_mat** (*bool*) – if the inputed matrices for x and y are distance matrices.
- **dmetric** (*str*) – distance metric used internally. Defaults to 'euclidean'

- **fit_ols** (*bool*) – if True, an OLS model is fitted after the dimension reduction.
- **copy** (*bool*) – Whether to make a deep copy of the input data or not.
- **verbose** (*bool*) – Set to True prints the iteration number.
- **return_scaling_object** (*bool.*) – If True, the scaling object will be return after the dimension reduction.

Attributes always provided

- *x_loadings_*: Estimated basis of the central subspace
- *x_scores_*: The projected X data.
- *x_loc_*: location estimate for X
- *x_sca_*: scale estimate for X
- ` ols_obj` : fitted OLS objected
- *y_loc_*: y location estimate
- *y_sca_*: y scale estimate

Attributes created only when corresponding input flags are `True`

- *whitening_*: whitened data matrix (usually denoted K)
- *scaling_object_*: scaling object from *VersatileScaler*

__init__(*sudiremeth='dcov-sdr'*, *n_components=2*, *trimming=0*, *optimizer_options={'max_iter': 1000}*, *optimizer_constraints=None*, *optimizer_arguments=None*, *optimizer_start=None*, *center_data=True*, *center='mean'*, *scale_data=True*, *whiten_data=False*, *compression=False*, *n_slices=6*, *dmetric='euclidean'*, *fit_ols=True*, *copy=True*, *response_type='continuous'*, *verbose=True*, *return_scaling_object=True*)

Methods

__init__([sudiremeth, n_components, ...])	

fit(X, y, *args, **kwargs)	Fit a Sufficient Dimension Reduction Model.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator. :param deep: If True, will return the parameters for this estimator and contained subobjects that are estimators. :type deep: boolean, optional.
predict(Xn[, is_distance_mat])	predicts the response on new data Xn
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator.
transform(Xn[, distance_mat])	Computes the dimension reduction of the data Xn based on the fitted sudire model.

Attributes

3.2.2 Dependencies

- From *sklearn.base*: *BaseEstimator*,`TransformerMixin`,`RegressorMixin`
- From *sklearn.utils*: *_BaseComposition*
- *copy*
- From *scipy.stats* : *trim_mean*
- From *scipy.linalg*: *inv*, *sqrtm*
- *cython*
- From *ipopt* : *minimize_ipopt*
- *numpy*
- From *statsmodels.regression.linear_model*: *OLS*
- *statsmodels.robust*

3.2.3 References

1. Wenhui Sheng and Xiangrong Yin Sufficient Dimension Reduction via Distance Covariance, in: Journal of Computational and Graphical Statistics (2016), 25, issue 1, pages 91-104.
2. Yu Zhang, Jicai Liu, Yuesong Wu and Xiangzhong Fang, A martingale-difference-divergence-based estimation of central mean subspace, in: Statistics and Its Interface (2019), 12, number 3, pages 489-501.
3. Li K-C, Sliced Inverse Regression for Dimension Reduction, Journal of the American Statistical Association (1991), 86, 316-327.
4. R.D. Cook, and Sanford Weisberg, Sliced Inverse Regression for Dimension Reduction: Comment, Journal of the American Statistical Association (1991), 86, 328-332.
5. B. Li and S.Wang, On directional regression for dimension reduction, Journal of the American Statistical Association (2007), 102:997–1008.
6. K.-C. Li., On principal hessian directions for data visualization and dimension reduction:Another application of stein's lemma, Journal of the American Statistical Association(1992)., 87,1025–1039.
7. R. D. Cook and B. Li., Dimension Reduction for Conditional Mean in Regression, The Annals of Statistics(2002)30(2):455–474.
8. Jia Zhang and Xin Chen, Robust Sufficient Dimension Reduction Via Ball Covariance Computational Statistics and Data Analysis 140 (2019) 144–154
9. Li B, Sufficient Dimension Reduction: Methods and Applications with R. (2018) Chapman& Hall /CRC, Monographs on Statistics and Applied Probability, New York

3.3 sprm

Sparse partial robust M regression (SPRM) is a sparse and robust alternative to PLS that can be calculated efficiently (Hoffmann, Serneels, Filzmoser,and Croux 2015). The subpackage is organized slightly differently from the other two mainsubpackages. Because SPRM combines the virtues of robust regression with sparse dimension reduction, besides the SPRM estimators itself, each of these building blocks are provided themselves as class objects that can be deployed in sklearn pipelines. The class objects rm, snipls and sprm are sourced by default when importing direpack.

3.3.1 Robust M regression

M regression is a generalization of least squares regression in the sense that it minimizes a more general objective that allows to tune the estimator's robustness. In M regression, the vector of regression coefficients is defined as:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_i \rho\left(\frac{r_i(\boldsymbol{\beta})}{\hat{\sigma}}\right)$$

where r_i are the casewise regression residuals and $\hat{\sigma}$ is a robust scale estimator thereof. The ρ function defines the properties of the estimator. Identity to the least squares estimator is obtained if $\rho(r) = r^2$, but robustness can be introduced by taking a different function, for instance a function that is approximately quadratic for small (absolute) r, but increases more slowly than r^2 for larger values of r. Objective (3.3.1) can be solved numerically, but it is well known that its solution can equivalently be obtained through an iteratively reweighting least squares (IRLS), which is how it is implemented in sprm. In the package, the Fair, Huber or Hampel reweighting functions can be picked, which will lead to different robustness properties.

3.3.2 Sparse NIPALS

A second building block in the package is the SNIPLS algorithm. It is a sparse version of the NIPALS algorithm for PLS and as such, essentially a computationally efficient implementation of univariate sparse PLS. Again, the SNIPLS components are linear combinations of the original variables through a set of weighting vectors $\mathbf{w}_i$ that maximize:

$$\begin{aligned} \mathbf{w}_i \qquad & =_{\mathbf{a}} \operatorname{cov}^2\left(\mathbf{a}^T\mathbf{X}, \mathbf{y}\right) + \lambda \parallel \mathbf{a} \parallel_1 \\ \text{subject to} \quad & \mathbf{w}_i^T\mathbf{X}^T\mathbf{X}\mathbf{w}_j = 0 \text{ and } \parallel \mathbf{w}_i \parallel_2 = 1 \end{aligned}$$

which in sparse PLS is typically maximized through a surrogate formulation. However, in this case, the exact solution to Criterion (3.3.2) can be obtained, which is what the SNIPLS algorithm builds upon. For details on the algorithm, the reader is referred to Hoffmann, Filzmoser, Serneels, and Varmuza (2016). At this point, remark that the SNIPLS algorithm has also become a key building block to analyze outlyingness (Debruyne, Höppner, Serneels,and Verdonck 2019).

3.3.3 Sparse partial robust M

Sparse partial robust M dimension reduction unites the benefits of SNIPLS and robust M estimation: it yields an efficient sparse PLS dimension reduction, while at the same time, it is robust against both leverage points and virtual outliers through robust M estimation. It is defined similarly as in (3.3.2) but instead maximizing a weighted covariance, with case weights that depend on the data. Consistent with robust M estimation, it can be calculated through iteratively reweighting SNIPLS. SPRM improves upon the original reweighted PLS proposal by (i) yielding a sparse estimate, (ii) having a reweighting scheme as well as starting values that weight both in the score and residual spaces and (iii) by allowing different weight functions, the most tuneable one being the Hampel function.

3.3.4 Usage

sprm([n_components, eta, fun, probp1, ...])	SPRM Sparse Partial Robust M Regression

direpack.sprm.sprm.sprm

class sprm(*n_components=1*, *eta=0.5*, *fun='Hampel'*, *probp1=0.95*, *probp2=0.975*, *probp3=0.999*, *centre='median'*, *scale='mad'*, *verbose=True*, *maxit=100*, *tol=0.01*, *start_cutoff_mode='specific'*, *start_X_init='pcapp'*, *columns=False*, *copy=True*)

SPRM Sparse Partial Robust M Regression

Algorithm first outlined in:
Sparse partial robust M regression, Irene Hoffmann, Sven Serneels, Peter Filzmoser, Christophe Croux, Chemometrics and Intelligent Laboratory Systems, 149 (2015), 50-59.

Parameters

- **eta** (*float.*) – Sparsity parameter in [0,1)
- **n_components** (*int*) – min 1. Note that if applied on data, n_components shall take a value <= min(x_data.shape)
- **fun** (*str*) – downweighting function. 'Hampel' (recommended), 'Fair' or 'Huber'
- **probp1** (*float*) – probability cutoff for start of downweighting (e.g. 0.95)
- **probp2** (*float*) – probability cutoff for start of steep downweighting (e.g. 0.975, only relevant if fun='Hampel')
- **probp3** (*float*) – probability cutoff for start of outlier omission (e.g. 0.999, only relevant if fun='Hampel')
- **centre** (*str*) – type of centring (*'mean'*, *'median'*, *'l1median'*, or *'kstepLTS'*, the latter recommended statistically, if too slow, switch to *'median'*)
- **scale** (*str*) – type of scaling ('std','mad', 'scaleTau2' [recommended] or 'None')
- **verbose** (*booleans*) – specifying verbose mode
- **maxit** (*int*) – maximal number of iterations in M algorithm
- **tol** (*float*) – tolerance for convergence in M algorithm
- **start_cutoff_mode** (*str,*) – values:'specific' will set starting value cutoffs specific to X and y (preferred); any other value will set X and y stating cutoffs identically. The latter yields identical results to the SPRM R implementation available from CRAN.
- **start_X_init** (*str,*) – values: 'pcapp' will include a PCA/broken stick projection to calculate the staring weights, else just based on X; any other value will calculate the X starting values based on the X matrix itself. This is less stable for very flat data (p >> n), yet yields identical results to the SPRM R implementation available from CRAN.
- **columns** (*(def false) Either boolean, list, numpy array or pandas Index*) – if False, no column names supplied; if True, if X data are supplied as a pandas data frame, will extract column names from the frame throws an error for other data input types if a list, array or Index (will only take length x_data.shape[1]), the column names of the x_data supplied in this list, will be printed in verbose mode.
- **copy** (*(def True) boolean, whether to copy data*) –

Attributes always provided

- *x_weights_*: X block PLS weighting vectors (usually denoted W)
- *x_loadings_*: X block PLS loading vectors (usually denoted P)
- *C_*: vector of inner relationship between response and latent variablesblock re
- *x_scores_*: X block PLS score vectors (usually denoted T)
- *coef_*: vector of regression coefficients
- *intercept_*: intercept
- *coef_scaled_*: vector of scaled regression coeeficients (when scaling option used)
- *intercept_scaled_*: scaled intercept
- *residuals_*: vector of regression residuals
- *x_ev_*: X block explained variance per component
- *y_ev_*: y block explained variance
- *fitted_*: fitted response
- *x_Rweights_*: X block SIMPLS style weighting vectors (usually denoted R)
- *x_caseweights_*: X block case weights
- *y_caseweights_*: y block case weights
- *caseweights_*: combined case weights
- *colret_*: names of variables retained in the sparse model
- *x_loc_*: X block location estimate
- *y_loc_*: y location estimate
- *x_sca_*: X block scale estimate
- *y_sca_*: y scale estimate
- *non_zero_scale_vars_*: indicator vector of variables in X with nonzero scale

__init__(*n_components=1*, *eta=0.5*, *fun='Hampel'*, *probp1=0.95*, *probp2=0.975*, *probp3=0.999*, *centre='median'*, *scale='mad'*, *verbose=True*, *maxit=100*, *tol=0.01*, *start_cutoff_mode='specific'*, *start_X_init='pcapp'*, *columns=False*, *copy=True*)

Methods

__init__([n_components, eta, fun, probp1, ...])	

fit(X, y)	Fit a SPRM model.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(Xn)	Predict using a SPRM model.
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator.
transform(Xn)	Transform input data.
valscore(Xn, yn, scoring)	Specific score function for validation data
weightnewx(Xn)	Calculate case weights for new data based on the projection in the SPRM score space

Attributes

snipls([eta, n_components, verbose, ...])	SNIPLS Sparse Nipals Algorithm

direpack.sprm.snipls.snipls

class snipls(*eta=0.5*, *n_components=1*, *verbose=True*, *columns=False*, *centre='mean'*, *scale='None'*, *copy=True*)

SNIPLS Sparse Nipals Algorithm

Algorithm first outlined in:

Sparse and robust PLS for binary classification, I. Hoffmann, P. Filzmoser, S. Serneels, K. Varmuza, Journal of Chemometrics, 30 (2016), 153-162.

Parameters

- **eta** (*float.*) – Sparsity parameter in [0,1)
- **n_components** (*int,*) – min 1. Note that if applied on data, n_components shall take a value <= min(x_data.shape)
- **verbose** (*Boolean (def true)*) – to print intermediate set of columns retained
- **columns** (*Either boolean, list, numpy array or pandas Index (def false)*) – if False, no column names supplied; if True, if X data are supplied as a pandas data frame, will extract column names from the frame throws an error for other data input types if a list, array or Index (will only take length x_data.shape[1]), the column names of the x_data supplied in this list, will be printed in verbose mode.
- **centre** (*str,*) – type of centring (*'mean'* [recommended], *'median'* or *'l1median'*),
- **scale** (*str,*) – type of scaling ('std','mad' or 'None')
- **copy** (*(def True): boolean,*) – whether to copy data. Note : copy not yet aligned with sklearn def - we always copy

Attributes always provided

- *x_weights_*: X block PLS weighting vectors (usually denoted W)
- *x_loadings_*: X block PLS loading vectors (usually denoted P)
- *C_*: vector of inner relationship between response and latent variablesblock re
- *x_scores_*: X block PLS score vectors (usually denoted T)
- *coef_*: vector of regression coefficients
- *intercept_*: intercept
- *coef_scaled_*: vector of scaled regression coeeficients (when scaling option used)
- *intercept_scaled_*: scaled intercept
- *residuals_*: vector of regression residuals
- *x_ev_*: X block explained variance per component
- *y_ev_*: y block explained variance
- *fitted_*: fitted response
- *x_Rweights_*: X block SIMPLS style weighting vectors (usually denoted R)
- *colret_*: names of variables retained in the sparse model
- *x_loc_*: X block location estimate
- *y_loc_*: y location estimate
- *x_sca_*: X block scale estimate
- *y_sca_*: y scale estimate
- *centring_*: scaling object used internally (from *VersatileScaler*)

__init__(*eta=0.5*, *n_components=1*, *verbose=True*, *columns=False*, *centre='mean'*, *scale='None'*, *copy=True*)

Methods

__init__([eta, n_components, verbose, ...])	

fit(X, y)	Fit a SNIPLS model.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(Xn)	Predict using a SNIPLS model.
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator.
transform(Xn)	Transform input data.

Attributes

3.3.5 Dependencies

- *pandas*
- *numpy*

3.3.6 References

1. Irene Hoffmann, Sven Serneels, Peter Filzmoser, Christophe Croux, Sparse partial robust M regression, Chemometrics and Intelligent Laboratory Systems, 149 (2015), 50-59.
2. Sven Serneels, Christophe Croux, Peter Filzmoser, Pierre J. Van Espen, Partial robust M regression, Chemometrics and Intelligent Laboratory Systems, 79 (2005), 55-64.
3. Hoffmann I., P. Filzmoser, S. Serneels, K. Varmuza, Sparse and robust PLS for binary classification, Journal of Chemometrics, 30 (2016), 153-162.
4. Filzmoser P, Höppner S, Ortner I, Serneels S, Verdonck T. Cellwise robust M regression. Computational Statistics and Data Analysis,147 (2020).

3.4 Pre-processing

The first step in most meaningful data analytics projects will be to pre-process the data, hence direpack proposes a set of tools for data pre-processing.

3.4.1 Data standardization

A first, well accepted way to pre-process data is to center them and scale them to unit variance on a column wise basis. This corresponds to transforming a $\mathbf{x}$ variable into z-scores:

$$\mathbf{z} = \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}}{\hat{\boldsymbol{\sigma}}}$$

where $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ are estimates of location and scale, respectively. For normally distributed data, the appropriate way to accomplish this is by centering about the mean and dividing by the column wise standard deviation. However, when the marginal distributions in the data significantly deviate from the normal, outliers could throw the result of that data standardization off, and robust or nonparametric alternatives become a more reliable choice. Essentially, all robust statistics are subject to a trade-off between efficiency and robustness, which means that the variance of the estimates will increase as the estimator can resist a higher fraction of outliers. While scikit-learn provides highly robust nonparametric standardization in its RobustScaler, the estimators included therein are known to have a low statistical efficiency (these are the median for location and the interquartile range for scale). Since autoscaling the data is often an essential step, a few location and scale estimators have been implemented. For location, with increasing performance in terms of the robustness—efficiency trade-off, these are: the column wise median, the spatial median (also called L_1-median, although it minimizes an L_2 norm) and the k step least trimmed squares (LTS, Rousseeuw and Leroy (1987)) estimator. For scale, the consistency corrected median absolute deviation (MAD) and the τ estimator of scale (Maronna and Zamar 2002) have been included. Generally, it holds true that the more statistically efficient the estimator in these lists is, the higher its computational cost. In preprocessing, these estimators can be accessed through its VersatileScaler class, which takes the names of these estimators as strings, but it will also accept functions of location and scale estimators, should the user prefer to apply other ones.

3.4.2 Spatial sign pre-processing

Besides standardizing data, it can be beneficial to transform data to some sort of signs. The generalized spatial sign transformation consists of transforming a variable $\mathbf{x}$ into

$$\mathbf{u} = (\mathbf{x} - \hat{\boldsymbol{\mu}}) \times f\left(\mathbf{x} - \hat{\boldsymbol{\mu}}\right)$$

where the spatial sign is obtained by setting $f(x) = \parallel x \parallel^{-1}$ and $\parallel \cdot \parallel$ denotes the norm (in all published literature in this context, the L_2 norm). Since spatial sign pre-processing (SS-PP) consists of dividing the data by their Euclidean norm, it is also known as normalizing and as such, is available in scikit-learn's Normalizer. Spatial sign pre-processing has been shown to convey moderate robustness to multivariate estimators that are entirely based on covariance estimates, such as PCA or PLS (Serneels, De Nolf, and Van Espen 2006). Moderate robustness means in this case that the resulting estimator can resist up to 50% of outliers, but will have a sizeable bias even for small fractions of contamination. The reason why this happens is that the spatial sign transform projects all cases onto the unit sphere indiscriminately, which can drastically change data topology, and thereby introduce bias. Recently, the generalized spatial sign transform has been proposed (Raymaekers and Rousseeuw 2019). These authors examine a set of different functions that can be plugged into the expression for $\mathbf{u}$, some of which will only transform those cases in the data that exceed a certain eccentricity threshold. These functions are the quadratic radial, ball, shell, Winsor and linear redescending (LR) functions, all of which can be accessed through direpack's GenSpatialSignPreprocessor.

3.4.3 Usage

VersatileScaler([center, scale, trimming])	VersatileScaler Center and Scale data about classical or robust location and scale estimates

direpack.preprocessing.robcent.VersatileScaler

class VersatileScaler(*center='mean'*, *scale='std'*, *trimming=0*)

VersatileScaler Center and Scale data about classical or robust location and scale estimates

Parameters

- **center** (*str or callable, location estimator. String has to be name of the*) – function to be used, or 'None'.
- **scale** (*str or callable, scale estimator*) –
- **trimming** (*trimming percentage to be used in location and scale estimation.*) –

Arguments for methods

- *X*: array-like, n x p, the data.
- *trimming*: float, fraction to be trimmed (must be in (0,1)).

Remarks

Options for classical estimators 'mean' and 'std' also give access to robust trimmed versions.

__init__(*center='mean'*, *scale='std'*, *trimming=0*)

Initialize values. Check if correct options provided.

Methods

__init__([center, scale, trimming])	Initialize values.
fit(X)	Estimate location and scale, store these in the class object.
fit_transform(X)	Estimate center and scale for training data and scale these data
get_params([deep])	Get parameters for this estimator.
inverse_transform([Xs])	Transform scaled data back to their original scale
predict(Xn)	Standardize new data on previously estimated location and scale.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Center and/or scale training data to pre-estimated location and scale

Attributes

GenSpatialSignPreProcessor([center, fun])	GenSpatialSignPreProcessor Generalized Spatial Sign Pre-Processing as a scikit-learn compatible object that can be used in ML pipelines.

direpack.preprocessing.gsspp.GenSpatialSignPreProcessor

class GenSpatialSignPreProcessor(*center='l1median'*, *fun='linear_redescending'*)

GenSpatialSignPreProcessor Generalized Spatial Sign Pre-Processing as a scikit-learn compatible object that can be used in ML pipelines.

Parameters

- **center** (*str or function,*) – location estimator for centring.str options: 'mean', 'median', 'l1median', 'kstepLTS', 'None'
- **fun** (*str or function,*) – radial transformation function, str options: 'ss' (the non-generalized spatial sign, equivalent to sklearn's Normalizer), 'ball', 'shell', 'quad' (quadratic), 'winsor', or 'linear_redescending' Methods: sklearn API: *fit(X)*, *transform(X)* and *fit_transform(X)* with

Attributes always provided

- *gss_* : the generalized spatial signs
- *Xm_* : the centred data
- *centring_* : VersatileScaler centring object

- *X_gss_pp_* : Data preprocessed by Generalized Spatial Sign

__init__(*center='l1median'*, *fun='linear_redescending'*)

Methods

__init__([center, fun])	

fit(X)	Calculate and store generalized spatial signs
fit_transform(X)	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Calculate Generalized Spatial Sign Pre-Pprocessed Data

3.4.4 References

1. Maronna RA, Zamar RH (2002). "Robust estimates of location and dispersion for high-dimensional datasets." Technometrics, 44(4), 307–317.
2. Rousseeuw PJ, Leroy AM (1987). Robust Regression and Outlier Detection. Wiley and Sons, New York
3. Raymaekers J, Rousseeuw PJ (2019). "A generalized spatial sign covariance matrix." Journal of Multivariate Analysis, 171, 94–111.
4. Serneels S, De Nolf E, Van Espen PJ (2006). "Spatial Sign Preprocessing: A Simple Way ToImpart Moderate Robustness to Multivariate Estimators." Journal of Chemical Information and Modeling, 46, 1402–1409.

3.5 Cross-validation and plotting

Each of the sudire, ppdire and sprm subpackages in direpackare wrappers around a broad class of dimension reduction methods. Each of these methods will have at least one tune-able hyperparameter; some have many more. The user will want to be able to find the optimal hyperparameters for the data at hand, which can be done through cross-validation or bayesian optimization. It is not the aim of direpack to provide its own hyperparameter tuning algorithms, as ample cross-validation utilities are available in scikit-learn's model selection subpackage and the direpack estimations have been written consistently with the scikit-learn API, such that these model selection tools from scikit-learn can directly be applied to them. However, some caution should be taken when training the robust methods. While all classical (non-robust) methods could just use scikit-learn's default settings, when tuning a robust model, outliers are expected to be in the data, such that it becomes preferable to apply a robust cross-validation metric as well. Thereunto, it is possible to use scikit-learn's median_absolute_error, which is an MAE (L1) scorer that is less affected by extreme values than the default mean_squared_error. However, particularly in the case of robust M estimators, a more model consistent approach can be pursued. The robust M estimators provide a set of case weights, and these can be used to construct a weighted evaluation metric for cross-validation. Exactly this is provided in the robust_loss function that is a part of the direpack cross-validation utilities.

Similar to hyperparameter tuning, direpack's mission is not to deliver a broad set of plotting utilities, but rather focus on the dimension reduction statistics. However, some plots many users would like to have in this context, are provided for each of the methods. These are :

- Projection plots. These plots visualize the scores $\mathbf{t}_i$ and a distinction can be made in the plots between cases that the model had been trained with, and test set cases.

- Parity plots. For the regularized regressions based on the estimated scores, these visualize the predicted versus actual responses, with the same distinction as for the scores.

For the special case of SPRM, the plots have enhanced functionality. Since SPRM provides case weights, which can also be calculated for new cases, the SPRM plots can flag outliers. In the sprm_plot function, this is set up with two cut-offs, based on the caseweight values,and visualized asregular cases,moderate outliersorharsh outliers. For SPRM, there is anoption as well to visualize the case weights themselves.

Examples of direpack's plotting functionalities are available in the example notebooks of ppdire, sprm and sudire .

3.6 Contributing

No package is complete and the authors would like to see direpack extend its functionality in the future. Some possible additions could be :

- Cellwise robust dimension reduction methods : For instance, a cellwise robust version of the robust M regression method, included in sprm, has recently been published (Filzmoseret al.2020), and could be included in direpack.
- Uncertainty quantification : The methods provided through direpack provide point estimates. In the future, the package could, e.g. be augmented with appropriate bootstrapping techniques, as was done for a related dimension reduction context
- GPU flexibility : There are many matrix manipulations in direpack, which can possiblybe sped up by allowing a GPU compatibility, which could be achieved by providing a TensorFlowor PyTorch back-end. However, this would be a major effort, since thepresent back-end integrally builds upon numpy.
- More (and better) unit tests.

3.6.1 Guidelines

Testing

Contributions should be accompanied by unit tests similar to those already available. Contrbutors can use the datasets presented in the example notebooks.

Documentation

We have followed PEP8 style when building this project and ask that contributors do so, for ease of maintainability.

3.6.2 Article

An article with further information on the package is available. Menvouta, E.J., Serneels, S., Verdonck, T., 2023. direpack: A python 3 package for state-of-the-art statistical dimensionality reduction methods. SoftwareX 21, 101282.

3.6.3 Contacts

- Dr Sven Serneels is co-founder at Gallop Data, Inc. and can be contacted at svenserneel (at) gmail.com.
- Emmanuel Jordy Menvouta is a PhD researcher in Statistics and Data Science at KU Leuven and can be contacted at emmanueljordy.menvoutankpwele (at) kuleuven.be.
- Prof Tim Verdonck is Professor of Statistics and Data Science at University of Antwerp and KU Leuven. He can be reached at tim.verdonck (at) uantwerp.be.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

D

G

P

S

V