Package 'sparsepca'

November 6, 2026

Type Package

Title Sparse Principal Component Analysis

Version 1.0.0

Description Fits principal components with sparse loading vectors for interpretable dimension reduction. Variable projection combines an orthogonal reconstruction factor with elastic net or cardinality regularization of the loadings. Deterministic, randomized, and robust algorithms support exploratory analysis of high-dimensional data; the randomized algorithm also supports sparse input matrices through implicit centering and scaling. A robust formulation separates entrywise outliers from the low-rank reconstruction. Regularization paths and approximate feature-count searches reuse matrix factorizations, and prediction applies the fitted transformation to new observations. The methods are described in Erichson et al. (2020) <doi:10.1137/18M1211350>.

License GPL (>= 3)

Depends R (>= 4.4.0)

Imports Matrix (>= 1.7-0), methods, rsvd (>= 2.0.0)

Suggests testthat (>= 3.3.2)

Encoding UTF-8

URL https://github.com/erichson/spca

BugReports https://github.com/erichson/spca/issues

NeedsCompilation no

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Author N. Benjamin Erichson [aut, cre],
Peng Zheng [aut],
Sasha Aravkin [aut],
Maria A. Sans-Fuentes [ctb] (ORCID:
<https://orcid.org/0000-0002-6402-9211>, Reported the robust objective error)

Maintainer N. Benjamin Erichson <erichson@icsi.berkeley.edu>

Repository CRAN

Date/Publication 2026-11-06 13:30:02 UTC

Contents

sparsepca-package 2
predict.spca 4
robspca 6
rspca 10
spca 14
spca-methods 20
spca_path 22
spca_target 25

Index **30**

`sparsepca-package` *Sparse principal component analysis via variable projection*

Description

Fit sparse principal components with deterministic, randomized, or robust algorithms. The package also supports prediction, regularization paths, and searches for an approximate number of selected variables.

Details

Principal components summarize many measured variables in a few score coordinates. Ordinary PCA usually uses every variable in each component, which can obscure what a score represents. Sparse PCA permits some coefficients to be exactly zero, giving each score a shorter list of contributing measurements. This exchanges some reconstruction quality for a representation that may be easier to inspect and use.

Rows of the input are observations and columns are variables. For the centered and/or scaled data X, the fitted scores are XB, where B contains sparse coefficients. A separate orthogonal factor A reconstructs the data as XBA^T. Both factors have p rows and k columns, and the score matrix has n rows and k columns. The choice of orientation matters: transposing the data changes which objects get sparse coefficients.

The functions preserve the fitted coefficient magnitudes. Scores may be correlated, and the historical `eigenvalues` output describes adjusted score energies rather than general covariance eigenvalues. The `spca` topic develops the mathematical model, preprocessing, parameter scaling, output definitions, and these interpretation conventions in detail.

Requirements

The package requires R 4.4.0 or later, **Matrix** 1.7-0 or later, and **rsvd** 2.0.0 or later. The R and Matrix requirements match those of rsvd 2.0.0. Earlier versions are no longer supported.

Choosing a routine

`spca` A deterministic fit using the full processed data. It is a useful starting point and numerical reference for manageable dense problems. Sparse input is accepted but converted to a dense workspace.

`rspca` A fit to a randomized compressed problem, with full-data scores afterward. It is most useful when a sketch much smaller than the data captures the relevant variation. Sparse **Matrix** input retains sparse storage for products and implicit preprocessing.

`robspca` An additive entrywise sparse-error model with a Huber reconstruction loss. It separates unusually large residual entries from the low-rank part and uses dense workspaces.

`spca_path` A sequence of candidate penalties sharing preprocessing and a factorization or sketch, with warm starts.

`spca_target` A bounded penalty search returning the closest converged candidate to a desired number of variables.

`predict.spca` The fitted training transformation applied to future observations, with no refitting.

Sparse loadings use a subset of variables. Sparse input storage is a separate property of a matrix; only the randomized fitting methods preserve it during centering and scaling. All fits return dense loading and score matrices. A sparse input does not force sparse components, and sparse components do not make a dense input cheap to store.

A practical workflow

Choose the observation/variable orientation, retain numeric measurements, and decide whether centering and scaling match the scientific question. Choose `k` as the number of score coordinates. Explore the tradeoff between loading sparsity and reconstruction with a short `alpha` path, keeping the preprocessing fixed. Inspect the actual coefficients and score correlations, not only a rounded summary. When an application suggests a desired number of measurements, `spca_target` searches for a nearby count and reports the achieved count explicitly. Use training-only preprocessing and `predict` for validation or downstream predictive models.

Numerical convergence, approximation quality, and statistical usefulness are separate questions. The joint optimization is nonconvex; an objective-based `converged` flag is not a global-optimality or scientific-validity certificate. A randomized fit additionally needs a sufficiently informative sketch. Sparsity does not itself guarantee stable feature selection, generalization, or causal interpretation.

To make an analysis reproducible, record `k`, preprocessing, relative penalties, convergence settings, and the chosen model. For randomized methods also record `o`, `q`, and the seed; for robust fits record `gamma` in the processed units. Fitted factors can differ by signs or bases within tied subspaces while representing the same reconstruction.

Author(s)

N. Benjamin Erichson, Peng Zheng, and Sasha Aravkin. Maria A. Sans-Fuentes (ORCID) contributed the report and correction of the robust objective error.

References

Erichson, N. B., Zheng, P., Manohar, K., Brunton, S. L., Kutz, J. N., and Aravkin, A. Y. (2020). Sparse Principal Component Analysis via Variable Projection. *SIAM Journal on Applied Mathematics*, 80(2), 977–1002. doi:10.1137/18M1211350.

See Also

The repository https://github.com/erichson/spca#readme provides a tutorial, application illustrations, and reproducible performance experiments. See `stats::prcomp` for ordinary PCA.

Examples

```
X <- as.matrix(iris[, 1:4])
fit <- spca(X, k = 2, alpha = 0.01, scale = TRUE, verbose = FALSE)
dim(fit$loadings)  # variables by components
dim(fit$scores)    # observations by components
colSums(fit$loadings != 0)
summary(fit)
predict(fit, X[1:3, , drop = FALSE])
```

`predict.spca` *Predict sparse principal component scores*

Description

Apply a fitted sparse PCA transformation to new observations using the stored training means, scales, and loading coefficients.

Usage

```
## S3 method for class 'spca'
predict(object, newdata, ...)

## S3 method for class 'rspca'
predict(object, newdata, ...)

## S3 method for class 'robspca'
predict(object, newdata, ...)
```

Arguments

`object`	A fitted object returned by `spca`, `rspca`, or `robspca`.
`newdata`	A real numeric matrix, numeric data frame, or real **Matrix** object with the same variables as the training data. Missing and nonfinite values are not allowed. If both training and new data have column names, names must be unique and identify the exact same variable set; columns are reordered to training order. Otherwise supply columns in training order. If omitted, return the stored training scores.

... Additional arguments; currently ignored.

Details

Prediction first applies the stored column means and scale factors, skipping operations stored as `FALSE`, and then multiplies by `object$loadings`. It does not estimate new preprocessing parameters. Centering sparse input can be applied as a correction to the score product; the returned score matrix is dense.

Predictions for robust fits use this same linear score map. The method does not infer new outliers or subtract the training `sparse` matrix from new observations. The stored training scores and new scores are in component coordinates, not original variable coordinates. Multiplication by `t(object$transform)` reconstructs the low-rank part in processed coordinates; undo column scaling and centering to restore original units.

Value

A dense numeric matrix with one row per new observation and one column per fitted component, including 1 by k and 0 by k matrices for single-row and empty batches. Available new-data row names are preserved. With no `newdata`, returns `object$scores`, including its retained training-row order.

Training and future observations

Fit preprocessing and loading coefficients on the training observations. Then supply future observations in the same measurement units to `predict`. Re-centering each new batch by its own mean changes the coordinate system and removes shifts relative to the training population; that is not the transformation fitted by the model.

For validation, keep the training and validation data separate when estimating means, scales, components, and tuning parameters. New rows need all fitted variables, even when some variables have zero loadings; the public input contract preserves the complete training variable set. An outcome or identifier column should be removed before prediction.

Named variables are matched exactly, including their spelling. With unnamed input, the caller is responsible for the column order. One-row and zero-row batches are supported and retain matrix dimensions; use `drop = FALSE` when subsetting a matrix to a single row. Missing new observations are rejected rather than silently dropped, so output rows correspond one for one to the supplied batch. Missing training rows, by contrast, were omitted during fitting.

See Also

`spca`, `rspca`, `robspca`.

Examples

```
# Hold out every fifth row for this small illustration.
training <- seq_len(nrow(iris)) %% 5 != 0
fit <- spca(iris[training, 1:4], k = 2, alpha = 0.01,
            scale = TRUE, verbose = FALSE)
newX <- as.matrix(iris[!training, 1:4])
scores <- predict(fit, newX)
```

```
head(scores)

# Reproduce the map with the training means, scales, and coefficients.
processed <- scale(newX, center = fit$center, scale = fit$scale)
all.equal(scores, processed %*% fit$loadings)

# A complete named variable set can be supplied in a different order.
reordered <- predict(fit, newX[, 4:1])
all.equal(scores, reordered)
dim(predict(fit, newX[1, , drop = FALSE]))
dim(predict(fit, newX[FALSE, , drop = FALSE]))
identical(predict(fit), fit$scores)
```

`robspca` *Robust sparse principal component analysis*

Description

Fit sparse component loadings and a sparse additive error matrix to reduce the influence of large individual reconstruction residuals. The method combines variable projection with an entrywise Huber loss.

Usage

```
robspca(
  X,
  k = NULL,
  alpha = 1e-04,
  beta = 1e-04,
  gamma = 100,
  center = TRUE,
  scale = FALSE,
  max_iter = 1000,
  tol = 1e-05,
  verbose = TRUE,
  penalty = c("elasticnet", "l0"),
  B_init = NULL
)
```

Arguments

`X`	A real numeric matrix, numeric data frame, or real **Matrix** object, with observations in rows and variables in columns. Input is converted to a dense matrix. Missing rows are omitted with a warning; at least two complete rows are required. Infinite and complex values are unsupported.
`k`	Positive integer number of components. `NULL` uses `min(nrow(X), ncol(X))` after incomplete rows are omitted. Larger values are reduced to that limit. A component can have zero scores when the data are rank deficient or its loadings are all zero.

`alpha` Finite nonnegative sparsity penalty (default `1e-4`), relative to the squared largest singular value of the processed data. The effective penalty is `alpha * d1^2`; see Details. Increasing `alpha` encourages fewer nonzero coefficients but does not impose an exact feature count.

`beta` Finite nonnegative ridge penalty, using the same relative scaling as `alpha`. The default is `1e-4`. Ridge shrinkage can improve conditioning for strongly correlated variables.

`gamma` Finite nonnegative penalty on the absolute entries of the sparse error matrix (default `100`). This is a threshold in the centered/scaled input units and is not scaled by a singular value. Smaller values allow more residual entries into the sparse component. A sufficiently large value leaves that component zero and recovers the ordinary squared reconstruction loss.

`center` Logical; subtract column means before fitting (default `TRUE`). The fitted means are stored for prediction.

`scale` Logical; divide each processed column by its scale factor (default `FALSE`). Factors are square roots of column sums of squares divided by $n - 1$, after any centering. This is the sample standard deviation for centered data and a root-mean-square quantity otherwise. Factors below `1e-8` are replaced by one.

`max_iter` Positive integer limit on optimization iterations. Reaching the limit does not establish convergence.

`tol` Finite nonnegative tolerance for relative objective change. This controls an optimization stopping rule, not a reconstruction-error bound or a statistical accuracy guarantee. An objective increase larger than a roundoff allowance stops with a warning and does not count as convergence; see Details.

`verbose` Logical, or the equivalent numeric zero/one; print optimization progress when enabled.

`penalty` Sparsity penalty: `"elasticnet"` (the default) uses an entrywise absolute-value penalty and soft thresholding; `"l0"` counts nonzero coefficients and uses hard thresholding. With `penalty = "l0"` and `beta > 0`, the combination of cardinality and ridge penalties is also called ZeroNet.

`B_init` Optional real finite p by k matrix of initial loading coefficients. `NULL` initializes from leading right singular vectors. A previous fit's `loadings` can supply a warm start when variable order, component count, and preprocessing match. Initial loadings are not normalized.

Details

For processed data X, robust SPCA seeks to minimize

$$\frac{1}{2}\|X - XBA^T - S\|_F^2 + \lambda_1 P(B) + \frac{\lambda_2}{2}\|B\|_F^2 + \gamma\|S\|_1$$

subject to $A^TA = I$. The default penalty $P(B)$ is the sum of absolute entries. With `penalty = "l0"`, it counts nonzero entries. As in spca, λ_1 and λ_2 are the user-supplied `alpha` and `beta` multiplied by the squared leading singular value of the processed input.

For fixed B and S, an SVD of $(X - S)^TXB$ gives the orthogonal factor A. The algorithm takes a proximal gradient step for B, then soft-thresholds $X - XBA^T$ at `gamma` to update S. The objective

must include S in the squared reconstruction residual as well as charging its penalty. Eliminating S gives the entrywise Huber loss: quadratic near zero and linear beyond the threshold.

The predictor matrix in the score map remains the processed X, not $X - S$. Consequently `scores = X %*% loadings` and the low-rank reconstruction is `tcrossprod(scores, transform)`. Adding `sparse` gives the full fitted model in processed coordinates. To obtain original units, undo scaling for both components and add the training means only once. Prediction applies the learned linear map and does not estimate a new sparse-error matrix.

Robustness here concerns an additive sparse-error model for individual entries. It is not a guarantee against every form of contamination. Centering and scaling use ordinary means and sums of squares rather than robust location/scale estimates. `gamma = 0` allows the sparse component to absorb the entire remaining residual and can give a degenerate low-rank fit. Choose the threshold with the processed measurement units and the application in mind.

`robspca` retains dense input, residual, and sparse-error matrices; accepting a sparse **Matrix** does not imply sparse-memory computation. The optimization is nonconvex, and `max_iter` remains a work limit. Variance diagnostics summarize the returned full-data scores; they do not measure the amount of corruption removed or certify correct identification of outliers.

Value

A list of class `"robspca"` with the components documented for spca, plus `sparse`: a dense n by p matrix S of estimated sparse errors in processed input units. Its rows correspond to retained complete observations. The additional scalar `gamma` records the unscaled error threshold supplied by the user. `objective` includes the complete squared residual and all three penalties. The `print` and `summary` methods follow the corresponding `spca` methods.

Interpreting the sparse error matrix

The two sparse objects have different roles. Zeros in `loadings` simplify the recipes for the component scores. Zeros in `sparse` say that a particular observation-variable residual remains entirely in the quadratic loss. For a low-rank residual r, the fitted error is zero when $|r| \leq \gamma$; otherwise it is $\mathrm{sign}(r)(|r| - \gamma)$. The remaining residual is thus limited in magnitude to `gamma` at the completed error update.

A nonzero error is a model-based assignment, not a formal outlier test. It may reflect a recording error, an unusual observation, a component missing because `k` is too small, or an inappropriate scale or penalty. Inspect its pattern across observations and variables together with the low-rank reconstruction. A good low-rank fit and plausible error structure are both needed for a useful interpretation.

Choosing gamma and measurement units

`alpha` encourages zeros in the loading coefficients; `gamma` determines which reconstruction residuals are expensive to leave in the squared loss. They are not interchangeable controls. The historical default `gamma = 100` is not an automatically calibrated threshold, especially after standardizing variables. Consider plausible measurement-error sizes in the processed units, inspect residuals, and compare a small range of scientifically meaningful thresholds. Residuals from an ordinary fit can provide context, but they also include low-rank truncation and regularization error.

A small `gamma` allows more of the observations to be explained by `sparse`; making it arbitrarily small is not necessarily an improvement. A large value approaches the ordinary squared-loss fit. If

all measurements are multiplied by a nonzero constant c and column scaling is disabled, preserve the same model by multiplying `gamma` by $|c|$. The relative `alpha` and `beta` stay unchanged because their effective coefficients already scale with the squared data singular value.

Robust location or scale estimates can be applied externally when justified by the analysis. In that case, retain those transformations separately, fit with `center = FALSE, scale = FALSE`, and apply the same external transformation to future observations. The fit stores only preprocessing performed by the function itself; `predict` cannot recover an external preprocessing step automatically.

Note

Version 1.0.0 corrects an objective bookkeeping error reported by Maria A. Sans-Fuentes. Earlier versions charged the penalty on S but omitted it from the squared reconstruction residual, which could make the recorded objective rise and terminate optimization too early. Robust fits with nonzero `sparse` can therefore run longer and return different fitted values after the correction.

References

Erichson, N. B., Zheng, P., Manohar, K., Brunton, S. L., Kutz, J. N., and Aravkin, A. Y. (2020). Sparse Principal Component Analysis via Variable Projection. *SIAM Journal on Applied Mathematics*, 80(2), 977–1002. doi:10.1137/18M1211350.

See Also

`spca`, `rspca`, `predict.spca`.

Examples

```
set.seed(42)
signal <- matrix(rnorm(120 * 2), 120, 2)
weights <- matrix(0, 8, 2)
weights[1:4, 1] <- 1
weights[5:8, 2] <- 1
X <- tcrossprod(signal, weights) + matrix(rnorm(120 * 8, sd = 0.1), 120)
X[cbind(c(3, 40, 99), c(1, 6, 4))] <- c(8, -10, 9)
fit <- robspca(X, k = 2, alpha = 0.001, gamma = 0.5,
               verbose = FALSE)
summary(fit)
sum(fit$sparse != 0)
low_rank <- tcrossprod(fit$scores, fit$transform)
processed_fit <- low_rank + fit$sparse
# Here scale = FALSE; add the training means only once.
# The sparse matrix represents differences, so it receives no mean.
reconstructed <- sweep(processed_fit, 2, fit$center, "+")
low_rank_in_original_units <- sweep(low_rank, 2, fit$center, "+")
head(which(fit$sparse != 0, arr.ind = TRUE))

# Recompute the complete default elastic-net objective from the outputs.
processed <- scale(X, center = fit$center, scale = fit$scale)
remaining <- processed - low_rank - fit$sparse
criterion <- 0.5 * sum(remaining^2) +
  fit$alpha * sum(abs(fit$loadings)) +
```

```
  0.5 * fit$beta * sum(fit$loadings^2) +
  fit$gamma * sum(abs(fit$sparse))
all.equal(criterion, tail(fit$objective, 1))
```

---

rspca *Randomized sparse principal component analysis*

Description

Approximate a sparse principal component fit by compressing the rows into a randomized subspace before optimizing the loading coefficients. The final component scores use the complete processed data.

Usage

```
rspca(
  X,
  k = NULL,
  alpha = 1e-04,
  beta = 1e-04,
  center = TRUE,
  scale = FALSE,
  max_iter = 1000,
  tol = 1e-05,
  o = 20,
  q = 2,
  verbose = TRUE,
  penalty = c("elasticnet", "l0"),
  B_init = NULL
)
```

Arguments

X	A real numeric matrix, numeric data frame, or real **Matrix** object. Observations are rows and variables are columns. Sparse **Matrix** inputs retain sparse storage for matrix products and implicit centering/scaling. Missing rows are omitted with a warning; at least two complete rows are required. Infinite and complex values are unsupported.
k	Positive integer number of components. `NULL` uses `min(nrow(X), ncol(X))` after incomplete rows are omitted. Larger values are reduced to that limit. A component can have zero scores when the data are rank deficient or its loadings are all zero.
alpha	Finite nonnegative sparsity penalty (default `1e-4`), relative to the squared largest singular value of the sketch. Its effective value is `alpha * d1^2`, with `d1` computed from the sketch, not the full input.
beta	Finite nonnegative ridge penalty (default `1e-4`), using the same sketch-based relative scaling as `alpha`.

`center` Logical; subtract column means before fitting (default `TRUE`). The fitted means are stored for prediction.

`scale` Logical; divide each processed column by its scale factor (default `FALSE`). Factors are square roots of column sums of squares divided by $n - 1$, after any centering. This is the sample standard deviation for centered data and a root-mean-square quantity otherwise. Factors below `1e-8` are replaced by one.

`max_iter` Positive integer limit on optimization iterations. Reaching the limit does not establish convergence.

`tol` Finite nonnegative tolerance for relative objective change. This controls an optimization stopping rule, not a reconstruction-error bound or a statistical accuracy guarantee. An objective increase larger than a roundoff allowance stops with a warning and does not count as convergence; see Details.

`o` Nonnegative integer oversampling parameter (default `20`). The sketch width is `min(k + o, nrow(X), ncol(X))` after incomplete rows are omitted. Additional columns can improve the approximation while increasing memory and computation.

`q` Nonnegative integer number of power iterations (default `2`). Each iteration requires additional products with the processed input and its transpose, with intermediate orthogonalization. Increasing `q` can help for a slowly decaying singular spectrum.

`verbose` Logical, or the equivalent numeric zero/one; print optimization progress when enabled.

`penalty` Sparsity penalty: `"elasticnet"` (the default) uses an entrywise absolute-value penalty and soft thresholding; `"l0"` counts nonzero coefficients and uses hard thresholding. With `penalty = "l0"` and `beta > 0`, the combination of cardinality and ridge penalties is also called ZeroNet.

`B_init` Optional real finite p by k initial loading matrix. `NULL` uses the leading right singular vectors of the sketch. A warm start must use the same variable order, component count, and preprocessing. It does not reuse a sketch from an earlier function call; use `spca_path` for that.

Details

The algorithm forms an orthonormal basis Q of width at most $k + o$, using random combinations of the processed input columns. Power iterations refine this basis. It then replaces X with $C = Q^T X$ in the sparse fitting objective described by `spca`. The coefficients still correspond to the original variables. Returned scores are XB, and reconstruction is XBA^T in processed coordinates.

Both relative penalties are scaled by the squared leading singular value of the sketch C. The `objective` history refers to the sketched optimization problem, not the full-data reconstruction loss. Variance statistics are calculated from the full-data scores using the conventions documented for `spca`. Use full-data residuals or another common reference when comparing randomized and deterministic fit quality. The `converged` flag concerns optimization on the sketch; it does not certify the accuracy of that sketch. Oversampling and power iterations control subspace approximation, whereas `tol` and `max_iter` control optimizer stopping.

Randomized compression is most useful when $k + o$ is much smaller than the matrix dimensions and the data have a useful low-rank approximation. Small matrices, a large requested rank, or a

flat singular spectrum reduce the advantage. A full-rank noisy matrix can still admit an accurate low-rank approximation; the relevant quantity is the energy outside the retained subspace. Omitting k requests the full available component count and offers no compression advantage.

Call set.seed before fitting to reproduce random draws. The function uses R's current random-number stream and does not reset it. Numerical libraries can still choose different signs or bases for tied subspaces. A warm start reuses loading coefficients but a separate call creates its own sketch; spca_path reuses one sketch across the whole path.

Value

A list of class "rspca" with the same components and dimensions as a spca result. objective, effective alpha, and effective beta refer to the sketch; scores, var, score_variance, eigenvalues, and sdev use the full processed data. The print and summary methods follow the corresponding spca methods.

Understanding the compression

Random combinations of the columns reveal a space of observation patterns that the data can express. The columns of Q describe that space. Projecting onto them replaces many observation rows by ℓ linear combinations, while retaining all p original variable columns. The sparse coefficients therefore still refer to the original measured variables; the method does not select a random subset of variables or observations and fit only that subset.

If most data energy lies in a few directions, the smaller problem can contain almost all the information relevant to the requested fit. Truncation error from choosing k, sparsity-induced error from choosing alpha, and approximation error from the random sketch are distinct. More optimizer iterations address none of the energy omitted by the sketch.

Choosing and checking a sketch

Choose k for the statistical representation first. Oversampling o gives the sketch extra directions in which to find it; those directions are not extra returned components. The defaults o = 20 and q = 2 are a useful starting point. Setting q = 0 avoids the additional power-iteration products; increasing q emphasizes stronger singular directions and can improve accuracy when their separation is weak. Increasing either o or q costs extra work; increasing o also increases the dense sketch and basis storage.

On a manageable problem, compare full-data reconstruction error with spca at the same rank, preprocessing, and relative penalties, remembering that their effective penalties can differ slightly because the sketch estimates d1. For a pure PCA approximation check, set both penalties to zero and compare with stats::prcomp. For a large problem, repeat a fit with a larger sketch or more power iterations and compare the represented model and selected-variable patterns. Repeating selected seeds can reveal sketch sensitivity; signs or rotations within tied subspaces alone are not evidence of a bad fit.

A dense full-data residual is convenient for a small diagnostic, as in the example below. For large sparse input, evaluate residuals in row blocks using the fitted means and scales rather than densifying the entire matrix just to assess a sparse-memory fit. A small residual and stable support are useful evidence, not a guarantee of statistical generalization or a global sparse-PCA optimum.

Sparse input and memory

For sparse input, products with the centered/scaled matrix are evaluated without creating the entire centered dense matrix. For example, $(X - \mathbf{1}\mu^T)W = XW - \mathbf{1}(\mu^T W)$. This preserves sparse storage of X; the mean correction uses smaller dense factors. The basis Q, projected matrix C, loading factors, and score matrix are generally dense, with dimensions n by ℓ, ℓ by p, p by k, and n by k, respectively, where $\ell = \min(k + o, n, p)$. A full reconstruction is dense.

Sparse input and sparse loading coefficients are separate concepts: `rspca` supports both, but zero coefficients in the returned loadings are stored in an ordinary matrix. See `spca` and `robspca` for methods that use dense input workspace. Common sparse storage formats are converted to a general compressed column representation before fitting. That conversion and intermediate products can make copies; sparse support is not a promise of zero-copy execution. Memory savings diminish when ℓ approaches the input dimensions or when the dense score matrix itself is large.

References

Erichson, N. B., Zheng, P., Manohar, K., Brunton, S. L., Kutz, J. N., and Aravkin, A. Y. (2020). Sparse Principal Component Analysis via Variable Projection. *SIAM Journal on Applied Mathematics*, 80(2), 977–1002. doi:10.1137/18M1211350.

Erichson, N. B., Voronin, S., Brunton, S. L., and Kutz, J. N. (2019). Randomized Matrix Decompositions Using R. *Journal of Statistical Software*, 89(11), 1–48. doi:10.18637/jss.v089.i11.

See Also

`spca`, `robspca`, `spca_path`, `predict.spca`.

Examples

```
# A small example illustrating sparse storage and sparse coefficients.
X <- Matrix::bdiag(outer(1:20, 1:6), outer(1:20, 6:1),
                   outer(1:20, 1:6))
set.seed(42)
fit <- rspca(X, k = 3, alpha = 0.005, scale = TRUE,
             o = 5, q = 2, verbose = FALSE)
colSums(fit$loadings != 0)
summary(fit)
predict(fit, X[1:3, , drop = FALSE])

# This example is small enough for an explicit dense quality check.
# Keep the same training preprocessing for both the data and residual.
processed <- scale(as.matrix(X), center = fit$center, scale = fit$scale)
reconstructed <- tcrossprod(fit$scores, fit$transform)
norm(processed - reconstructed, "F") / norm(processed, "F")
c(iterations = fit$iterations, converged = fit$converged)
```

spca *Sparse principal component analysis*

Description

Fit interpretable component scores as sparse linear combinations of the input variables. Variable projection alternates an exact orthogonal reconstruction update with a proximal step for the loading coefficients.

Usage

```
spca(
  X,
  k = NULL,
  alpha = 1e-04,
  beta = 1e-04,
  center = TRUE,
  scale = FALSE,
  max_iter = 1000,
  tol = 1e-05,
  verbose = TRUE,
  penalty = c("elasticnet", "l0"),
  B_init = NULL
)
```

Arguments

X	A real numeric matrix, data frame with numeric columns, or real **Matrix** object. Rows are observations and columns are variables. Rows containing missing values are omitted with a warning; at least two complete observations and one variable are required. Logical inputs are treated as zero/one. Infinite and complex values are not supported. spca converts **Matrix** input to a dense matrix; see rspca for computation that preserves sparse input storage.
k	Positive integer number of components. NULL uses min(nrow(X), ncol(X)) after incomplete rows are omitted. Larger values are reduced to that limit. A component can have zero scores when the data are rank deficient or its loadings are all zero.
alpha	Finite nonnegative sparsity penalty (default 1e-4), relative to the squared largest singular value of the processed data. The effective penalty is alpha * d1^2; see Details. Increasing alpha encourages fewer nonzero coefficients but does not impose an exact feature count.
beta	Finite nonnegative ridge penalty, using the same relative scaling as alpha. The default is 1e-4. Ridge shrinkage can improve conditioning for strongly correlated variables.
center	Logical; subtract column means before fitting (default TRUE). The fitted means are stored for prediction.

`scale` Logical; divide each processed column by its scale factor (default `FALSE`). Factors are square roots of column sums of squares divided by $n - 1$, after any centering. This is the sample standard deviation for centered data and a root-mean-square quantity otherwise. Factors below `1e-8` are replaced by one.

`max_iter` Positive integer limit on optimization iterations. Reaching the limit does not establish convergence.

`tol` Finite nonnegative tolerance for relative objective change. This controls an optimization stopping rule, not a reconstruction-error bound or a statistical accuracy guarantee. An objective increase larger than a roundoff allowance stops with a warning and does not count as convergence; see Details.

`verbose` Logical, or the equivalent numeric zero/one; print optimization progress when enabled.

`penalty` Sparsity penalty: `"elasticnet"` (the default) uses an entrywise absolute-value penalty and soft thresholding; `"l0"` counts nonzero coefficients and uses hard thresholding. With `penalty = "l0"` and `beta > 0`, the combination of cardinality and ridge penalties is also called ZeroNet.

`B_init` Optional real finite p by k matrix of initial loading coefficients. `NULL` initializes from leading right singular vectors. A previous fit's `loadings` can supply a warm start when variable order, component count, and preprocessing match. Initial loadings are not normalized.

Details

Let X denote the retained, centered and/or scaled data, with n observations and p variables. The default method seeks to minimize

$$\frac{1}{2}\|X - XBA^T\|_F^2 + \lambda_1\|B\|_1 + \frac{\lambda_2}{2}\|B\|_F^2$$

subject to $A^TA = I$. Here A and B are p by k; $\|B\|_1$ is the sum of absolute entries, and $\|\cdot\|_F$ is the Frobenius norm. The effective coefficients are $\lambda_1 =$ `alpha` d_1^2 and $\lambda_2 =$ `beta` d_1^2, where d_1 is the largest singular value of the processed matrix. For `penalty = "l0"`, replace $\|B\|_1$ with the number of nonzero entries of B. The same numeric `alpha` does not imply equivalent sparsity under the two penalties.

For fixed B, an SVD of X^TXB gives the best orthogonal reconstruction factor A. A gradient step followed by thresholding updates B. The joint problem is nonconvex, so the solution can depend on initialization. Small objective change is a convergence diagnostic, not proof of a global minimum or a bound on the gradient. With `alpha = beta = 0` and the default SVD initialization, deterministic fitting recovers ordinary PCA with matching preprocessing, up to signs and bases within repeated singular subspaces. A custom `B_init` can remain in a different stationary subspace even with zero penalties.

Relative improvement divides the decrease in objective by the magnitude of the current objective, with a positive machine denominator at zero. The allowance for numerical stagnation is 64 times machine precision times the largest of the consecutive objective magnitudes and the squared data norm. Machine precision is `.Machine$double.eps`. Larger increases stop with a warning and `converged = FALSE`. For `rspca`, the squared data norm in this allowance is that of the sketch. The returned A was optimized immediately before the final B update (and the final S update for robust

fits), so objective convergence does not imply that every block's first-order optimality condition holds to `tol`. A tighter tolerance can refine those conditions, at the cost of more iterations.

Scores are $Z = XB$; reconstruction in processed coordinates is ZA^T. The loading matrix B retains its penalized scale. It is generally neither normalized nor orthogonal. The reconstruction factor A is not a matrix inverse of B. To reconstruct in input coordinates, multiply columns of ZA^T by the stored scales and then add the stored means, omitting operations stored as `FALSE`.

Value

A list of class `"spca"` containing:

loadings A dense p by k matrix B of fitted sparse coefficients. Zeros are stored in an ordinary R matrix.

transform A dense p by k reconstruction factor A with orthonormal columns.

scores A dense n by k matrix XB. Rows correspond to complete observations retained from the input; available input row names are preserved.

eigenvalues Numeric vector of length k containing sequentially adjusted score energies divided by $n - 1$; see the variance discussion. The field name is retained for compatibility.

sdev Square roots of `eigenvalues`, not generally the marginal standard deviations of correlated score columns.

score_variance Numeric vector of marginal score squared norms divided by $n - 1$; sample variances when centering is enabled.

var Total squared energy of the processed input divided by $n - 1$, including directions omitted from the model.

center, scale Numeric vectors of length p containing training means and scale factors, respectively, or `FALSE` when the corresponding operation is disabled. They describe the preprocessing performed by the function, in training variable order.

objective Numeric vector with one penalized objective value per completed optimization iteration, or a single zero for zero-energy input.

iterations Number of stored objective values.

converged Logical indicator of objective-based convergence. This is not a global-optimality certificate.

improvement Relative change used by the final convergence check.

alpha, beta Effective penalties, scaled by the squared leading singular value.

penalty The selected sparsity penalty.

feature_names Training column names used for prediction, or `NULL` for unnamed input.

The `summary` method returns a numeric matrix with four rows: adjusted variance, its square root, proportion of total input energy, and cumulative proportion. The historical row labels and three-decimal rounding are retained. The `print` method displays the fitted statistics and loadings.

Two matrices with different jobs

Ordinary PCA uses orthogonal loading vectors to make component scores and reconstruct the observations. Sparse PCA separates those jobs. Each column of `loadings` is a recipe for one score: a zero coefficient means that the corresponding variable is not directly used in that recipe. Each row of `scores` contains the resulting k coordinates for one observation. The dense `transform` maps those coordinates back to the p measured variables.

This distinction lets a component use a short, interpretable list of variables while its reconstruction describes relationships with many variables. A variable omitted from `loadings` can still have a nonzero reconstructed value through `transform`. Variables can contribute to more than one component. Sparsity concerns individual coefficients, so the total number of selected variables can be smaller than the sum of the per-component nonzero counts.

The fitted scale of `loadings` is part of the model. Normalizing its columns afterward changes scores, prediction, and the penalized objective. Use the original fitted coefficients for calculations; make a separate normalized copy only if a particular display requires it. A coefficient's magnitude or sign is not a significance test or a causal effect.

Preparing data and choosing parameters

Decide first what an observation and a variable represent. For example, rows might be repeated measurements and columns might be sensors. Transposing the data changes which objects receive sparse coefficients. Supply only numeric measurements; identifiers and outcome labels should not be included accidentally as additional variables. Missing rows are omitted, not imputed. Keep their original row identifiers if scores must later be joined to other data.

Centering asks for variation around the training means. Use `center = FALSE` when the origin itself has a meaningful role in the analysis. Scaling makes variables comparable in standardized units; it is useful when measurement units differ, but it also gives small variations in an otherwise nearly constant variable more influence. With `center = FALSE`, scaling uses uncentered squared norms with divisor $n - 1$, not centered standard deviations. The absolute `1e-8` scale cutoff is retained for compatibility, so constant or extremely small columns are not divided by a near-zero number.

A practical tuning workflow is:

1. Choose `k` for the representation you need, using domain knowledge and, where appropriate, held-out performance. It is the number of score coordinates, not the number of selected variables. The default `k = NULL` requests the full available count; it does not select a useful rank automatically.
2. Fit a weakly penalized model, then examine a range of `alpha` values with `spca_path`. Inspect both nonzero counts and what the selected variables mean. A grid such as `c(0.0001, 0.001, 0.01, 0.1)` is an exploration of scales, not a generally optimal set of penalties. If the application suggests a desired number of measured variables, `spca_target` searches for a fit with a nearby count.
3. Use `beta` to control ridge shrinkage and conditioning, particularly with correlated or redundant measurements. Increasing it also reduces loading magnitudes, so it changes the fitted model. The small default is a starting point, not an automatic selection rule.
4. Inspect `converged`, the objective history, and reconstruction quality. If the iteration limit was reached, a larger `max_iter` permits more work. A tighter `tol` refines objective stopping but does not repair an unsuitable rank or penalty.

Compare fits under the same preprocessing. Objectives at different penalties contain different costs and should not be ranked as if they were the same goodness-of-fit measure. Reconstruction error and validation performance answer complementary questions to sparsity. Hard thresholding can give abrupt support changes; the same numeric `alpha` should not be interpreted as equivalent to soft thresholding. Zero processed data return zero loadings and scores with a single zero objective value and `converged = TRUE`. Finite inputs can still exceed floating-point limits when squared. If the squared norm overflows, or underflows to zero for nonzero processed data, fitting stops with an error asking for rescaling. Bring measurements to a moderate scale before fitting. Automatic column scaling cannot resolve every extreme input because its own scale calculation uses squared values and retains the `1e-8` cutoff.

Sparse coefficients and sparse storage

A sparse loading matrix has many zero coefficients. A sparse input matrix is stored using a class from **Matrix** to avoid storing its many zero entries. Neither property implies the other. `spca` accepts sparse input but makes a dense working matrix for the deterministic factorization. Use `rspca` when retaining sparse input storage is important and a small randomized sketch is appropriate. Both methods return ordinary dense loading and score matrices. Forming `tcrossprod(scores, transform)` also allocates an n by p dense reconstruction; retaining the factors is often preferable until a reconstruction is actually needed.

Variance and loading interpretation

Sparse component scores can be correlated. The historical `eigenvalues` field contains sequentially adjusted score energies: project each score column onto the span of preceding columns, subtract that projection, and divide the squared residual norm by $n - 1$. This is equivalent to squared diagonal entries in an unpivoted QR decomposition of the scores, with dependent columns assigned zero. Rank is determined using base R's default QR tolerance, so numerically dependent score columns can also receive zero adjusted energy. This tolerance is separate from the optimizer's `tol`. The adjustment follows Zou, Hastie and Tibshirani (2006), applied to the returned, unnormalized loading coefficients. It depends on component order and loading scale; values are not necessarily decreasing.

For centered data these are adjusted sample variances. They are not generally eigenvalues of a covariance matrix. `score_variance` contains the marginal squared score norms divided by $n - 1$, without adjustment for overlapping components. If centering is disabled, both quantities are uncentered second moments with that denominator, rather than centered sample variances.

`var` is the total squared energy of the entire processed input divided by $n - 1$; for centered data it is the exact sum of input sample variances. Summary proportions use this denominator, including variation in omitted directions. They need not sum to one. For zero total energy, variance proportions are mathematically undefined; the summary displays zero by convention.

Loadings are coefficients, not variable–component correlations. For complete training data, correlations can be computed directly as `cor(as.matrix(X), fit$scores)`. Multiplying loadings by `sdev` does not generally produce these correlations for sparse components. A constant variable or an all-zero score has no defined correlation, so exclude such columns when calculating correlations.

Adjusted score energy is not generally the same as the reduction in reconstruction error from the fitted reconstruction factor. To assess reconstruction, evaluate the residual of `tcrossprod(scores, transform)` in the same processed units as the fit. Keep an ordinary PCA reference at the same rank and preprocessing when judging the price of interpretability.

Note

Version 1.0.0 corrects the variance statistics: older versions used singular values from an auxiliary Procrustes update, which were not the variances of the returned scores. Loading signs and bases within tied singular subspaces may differ between valid numerical implementations.

References

Erichson, N. B., Zheng, P., Manohar, K., Brunton, S. L., Kutz, J. N., and Aravkin, A. Y. (2020). Sparse Principal Component Analysis via Variable Projection. *SIAM Journal on Applied Mathematics*, 80(2), 977–1002. doi:10.1137/18M1211350.

Zou, H., Hastie, T., and Tibshirani, R. (2006). Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2), 265–286. doi:10.1198/106186006X113430.

See Also

`rspca` for randomized fitting, `robspca` for sparse additive errors, `spca_path` for a penalty path, `predict.spca` for new observations, `stats::prcomp` for ordinary PCA.

Examples

```
X <- as.matrix(iris[, 1:4])
fit <- spca(X, k = 2, alpha = 0.01, beta = 0.001,
            scale = TRUE, verbose = FALSE)
colSums(fit$loadings != 0)
summary(fit)

# Reconstruct in original measurement coordinates.
reconstructed <- tcrossprod(fit$scores, fit$transform)
reconstructed <- sweep(reconstructed, 2, fit$scale, "*")
reconstructed <- sweep(reconstructed, 2, fit$center, "+")

# Measure reconstruction error in the units used by the objective.
processed <- scale(X, center = fit$center, scale = fit$scale)
residual <- processed - tcrossprod(fit$scores, fit$transform)
norm(residual, "F") / norm(processed, "F")

# Read a component as a list of original variables and coefficients.
setNames(fit$loadings[, 1], colnames(X))
colnames(X)[fit$loadings[, 1] != 0]
cor(X, fit$scores)

# Hard thresholding with a ridge penalty (ZeroNet).
hard <- spca(X, k = 2, alpha = 0.005, beta = 0.001,
             penalty = "l0", scale = TRUE, verbose = FALSE)
colSums(hard$loadings != 0)

# Ordinary PCA with matching preprocessing is the unpenalized case.
ordinary <- spca(X, k = 2, alpha = 0, beta = 0,
                 scale = TRUE, verbose = FALSE)
reference <- stats::prcomp(X, rank. = 2, scale. = TRUE)
cbind(ordinary$sdev, reference$sdev[1:2])
```

spca-methods *Display and summarize sparse PCA fits*

Description

Print sparse loading coefficients and adjusted score statistics, or return a compact variance-summary matrix for a fitted model.

Usage

```
## S3 method for class 'spca'
print(x, ...)

## S3 method for class 'rspca'
print(x, ...)

## S3 method for class 'robspca'
print(x, ...)

## S3 method for class 'spca'
summary(object, ...)

## S3 method for class 'rspca'
summary(object, ...)

## S3 method for class 'robspca'
summary(object, ...)
```

Arguments

x, object	A fitted object returned by spca, rspca, or robspca.
...	Additional arguments; currently ignored.

Details

Statistics use the conventions in spca. The field eigenvalues contains adjusted score energies. Their square roots appear under the historical label "Standard deviations". Sparse components can be correlated. These values generally differ from covariance eigenvalues and marginal score standard deviations. The summary divides by total processed input energy, including directions omitted from the model, rather than by the sum of retained values. For zero input energy it displays zero proportions by convention.

With center = FALSE, the displayed quantities use uncentered squared energies divided by $n - 1$. For robust fits the statistics summarize the score map and do not measure the amount of outlier contamination removed. Inspect objective, iterations, and converged for optimization diagnostics.

Value

`summary` returns a numeric matrix with one column per component and rows named "Explained variance", "Standard deviations", "Proportion of variance", and "Cumulative proportion", rounded to three decimal places. The print methods display standard deviations, adjusted variances, and sparse loadings and invisibly return the rounded loading matrix.

Reading the summary

Each column describes one fitted score coordinate. The adjusted variance measures the part of that score not explained by earlier score columns, with the fitted loading scale retained. Therefore component order matters, the values need not decrease, and their proportions need not sum to one. The historical labels are retained for compatibility with existing code.

The sum of adjusted variance proportions is not generally $1 - \|X - XBA^T\|_F^2 / \|X\|_F^2$. The former describes score geometry; the latter measures the reconstruction produced by the fitted reconstruction factor. Use the actual residual to assess reconstruction and `cor(object$scores)` to inspect score correlations when no score column is constant. Neither statistic chooses a scientifically appropriate number of components automatically.

Display precision and computation

Printing and summaries round values to three decimal places. A small nonzero loading can therefore look like zero on screen. Count selected coefficients with `object$loadings != 0`, using the stored full-precision matrix, and use `object$eigenvalues`, `object$score_variance`, and `object$var` for numerical work. For a higher-precision display, call `print(object$loadings, digits = 6)` directly.

The print methods invisibly return the rounded loading matrix for compatibility; they do not return the complete fit. Keep the fitted object rather than replacing it with the result of `print`. Arguments in `...`, including a requested number of digits, are currently ignored by these fitted-object methods.

See Also

`spca`, `rspca`, `robspca`, `predict.spca`.

Examples

```
X <- as.matrix(iris[, 1:4])
fit <- spca(X, k = 2, alpha = 0.01, scale = TRUE, verbose = FALSE)
print(fit)
summary(fit)
cbind(adjusted = fit$eigenvalues, marginal = fit$score_variance)
colSums(fit$loadings != 0)

# Full-model reconstruction answers a different question to score variance.
processed <- scale(X, center = fit$center, scale = fit$scale)
residual <- processed - tcrossprod(fit$scores, fit$transform)
1 - sum(residual^2) / sum(processed^2)
```

spca_path *Sparse PCA regularization paths*

Description

Fit a sequence of sparsity penalties while reusing preprocessing and the initial matrix factorization or randomized sketch. Each fit provides the initial loading coefficients for the next penalty in the supplied order.

Usage

```
spca_path(
  X,
  k = NULL,
  alpha = c(1e-04, 0.001, 0.01),
  beta = 1e-04,
  center = TRUE,
  scale = FALSE,
  max_iter = 1000,
  tol = 1e-05,
  method = c("spca", "rspca"),
  o = 20,
  q = 2,
  verbose = TRUE,
  penalty = c("elasticnet", "l0"),
  B_init = NULL
)

## S3 method for class 'spca_path'
print(x, ...)
```

Arguments

X	A real numeric matrix, numeric data frame, or real **Matrix** object, with observations in rows and variables in columns. The deterministic method uses dense workspace; the randomized method preserves sparse input storage for products and implicit preprocessing. Missing rows are omitted with a warning. At least two complete rows are required; infinite and complex values are unsupported.
k	Positive integer number of components. NULL uses min(nrow(X), ncol(X)) after incomplete rows are omitted. Larger values are reduced to that limit. A component can have zero scores when the data are rank deficient or its loadings are all zero.
alpha	A nonempty numeric vector of finite nonnegative relative sparsity penalties. Fits follow this order without sorting. Increasing penalties from a weakly regularized fit is a convenient starting point.

beta	Finite nonnegative relative ridge penalty (default 1e-4), shared by all fits. Both penalties are scaled by the squared leading singular value of the processed input for method = "spca", or of the shared sketch for method = "rspca".
center	Logical; subtract column means before fitting (default TRUE). The fitted means are stored for prediction.
scale	Logical; divide each processed column by its scale factor (default FALSE). Factors are square roots of column sums of squares divided by $n - 1$, after any centering. This is the sample standard deviation for centered data and a root-mean-square quantity otherwise. Factors below 1e-8 are replaced by one.
max_iter	Positive integer limit on optimization iterations. Reaching the limit does not establish convergence.
tol	Finite nonnegative tolerance for relative objective change. This controls an optimization stopping rule, not a reconstruction-error bound or a statistical accuracy guarantee. An objective increase larger than a roundoff allowance stops with a warning and does not count as convergence; see Details.
method	Fit with deterministic "spca" (the default) or randomized "rspca". All randomized fits share one sketch. o and q are used only by the randomized method.
o	Nonnegative integer oversampling parameter (default 20). The sketch width is min(k + o, nrow(X), ncol(X)) after incomplete rows are omitted. Additional columns can improve the approximation while increasing memory and computation.
q	Nonnegative integer number of power iterations (default 2). Each iteration requires additional products with the processed input and its transpose, with intermediate orthogonalization. Increasing q can help for a slowly decaying singular spectrum.
verbose	Logical, or the equivalent numeric zero/one; print optimization progress when enabled.
penalty	Sparsity penalty: "elasticnet" (the default) uses an entrywise absolute-value penalty and soft thresholding; "l0" counts nonzero coefficients and uses hard thresholding. With penalty = "l0" and beta > 0, the combination of cardinality and ridge penalties is also called ZeroNet.
B_init	Optional real finite p by k loading matrix for the first fit. Later fits initialize from the preceding fit's raw loadings.
x	An object returned by spca_path.
...	Additional arguments; currently ignored by the print method.

Details

A path saves the preprocessing and factorization work that separate calls would repeat. Warm starts may also reduce optimization iterations, but this is not guaranteed. The objective is nonconvex, so a path fit can differ from a fresh SVD initialization at the same penalty.

Penalties encourage sparse coefficients; they do not impose an exact count. Counts can jump or be nonmonotone. nonzero reports the number of active coefficients in each component, whereas selected_features counts variables active in at least one component. These answer different questions when components overlap. Inspect reconstruction quality and stability in addition to

counts. Use spca_target to search penalties for an approximate desired number of variables without retaining every candidate fit.

Centering, scaling, the component count, the ridge penalty, and the chosen penalty type are shared across fits. For randomized paths, set.seed controls the single initial sketch. Each fit uses the result class, variance conventions, and convergence diagnostics of the selected method. Predict with an individual element of fits.

Value

A list of class "spca_path" with:

fits A list of fitted spca or rspca objects in the supplied penalty order.

alpha The user-supplied penalties before singular-value scaling.

nonzero A matrix of counts with one row per penalty and one column per component, containing colSums(loadings != 0).

selected_features The number of variables with a nonzero loading in at least one component, for each fit.

iterations The number of stored objective values for each fit; this equals the completed iterations except for the single zero value used for zero-energy input.

method, penalty The selected fitting method and sparsity penalty.

The print method displays a compact table of penalties, selected variable counts, iteration counts, and convergence indicators; per-component nonzero counts are available in nonzero.

Choosing a member of the path

A path is a collection of candidate models, not an automatic model selection procedure. Inspect the tradeoff between reconstruction, interpretable variables, and the purpose of the analysis. Compare validation rows in the shared training-centered/scaled coordinates; do not estimate a separate transformation for each validation batch. The example below reports validation reconstruction error beside the number of selected variables without claiming a universally best penalty.

Penalized objectives at different alpha values are different criteria, so the smallest recorded objective does not by itself select the best path member. A small number of active coefficients is not sufficient either: an all-zero loading matrix is sparse but uninformative. nonzero counts active coefficients per component, while selected_features counts the union of variables used anywhere. For example, two components with three coefficients each can together use only four variables if their supports overlap.

Warm starts follow the supplied sequence, even if it decreases or repeats penalties. They can lead to a different fit than restarting from the default singular vectors. If the path settles on an implausible representation, compare a shortlisted penalty with an independent fresh fit using the same settings. When comparing component patterns across fits, account for possible sign changes, reordering, and tied subspaces rather than relying on signed elementwise equality.

Computation and storage

The fits share preprocessing and one factorization or randomized sketch. Their effective penalties use the same spectral scale, which makes the supplied alpha sequence easy to interpret. The function still returns a complete fit for every member, including an n by k dense score matrix and two

p by k factors. Long paths can therefore use substantial output memory even when input storage is sparse. A concise initial grid followed by closer investigation of useful values is often more informative than retaining a very long grid.

See Also

spca, rspca, spca_target, predict.spca.

Examples

```
training <- seq_len(nrow(iris)) %% 5 != 0
path <- spca_path(iris[training, 1:4], k = 2,
                  alpha = c(0.001, 0.01, 0.1, 0.3),
                  scale = TRUE, verbose = FALSE)
print(path)
# A strong penalty can leave a requested component with zero loadings.
path$nonzero

# All candidates use the same transformation estimated on training rows.
newX <- as.matrix(iris[!training, 1:4])
first <- path$fits[[1]]
processed <- scale(newX, center = first$center, scale = first$scale)
error <- vapply(path$fits, function(fit) {
  reconstructed <- tcrossprod(predict(fit, newX), fit$transform)
  sqrt(mean((processed - reconstructed)^2))
}, numeric(1))
data.frame(alpha = path$alpha, variables = path$selected_features,
           validation_RMSE = error)
predict(path$fits[[2]], newX[1:3, , drop = FALSE])

# A sparse-input path shares one smaller random sketch.
X <- Matrix::bdiag(outer(1:12, 1:8), outer(1:12, 8:1),
                   outer(1:12, 1:8))
set.seed(42)
randomized <- spca_path(X, k = 3, alpha = c(0.001, 0.01),
                        method = "rspca", o = 3, q = 2,
                        scale = TRUE, verbose = FALSE)
randomized$nonzero
```

---

spca_target *Sparse PCA with an approximate number of selected variables*

Description

Search sparsity penalties for a fitted representation using approximately a requested number of variables. The search reuses preprocessing and one deterministic factorization or randomized sketch, then returns the closest converged candidate and a record of every evaluated penalty.

Usage

```
spca_target(
  X,
  k = NULL,
  n_features,
  alpha = NULL,
  beta = 1e-04,
  center = TRUE,
  scale = FALSE,
  max_iter = 1000,
  tol = 1e-05,
  method = c("spca", "rspca"),
  o = 20,
  q = 2,
  verbose = TRUE,
  penalty = c("elasticnet", "l0"),
  B_init = NULL,
  max_evals = 30L,
  count_tol = 0L
)

## S3 method for class 'spca_target'
print(x, ...)
```

Arguments

X	A real numeric matrix, numeric data frame, or real **Matrix** object, with observations in rows and variables in columns. The deterministic method uses dense workspace; the randomized method preserves sparse input storage for products and implicit preprocessing. Missing rows are omitted with a warning. At least two complete rows are required; infinite and complex values are unsupported.
k	Positive integer number of components. `NULL` uses `min(nrow(X), ncol(X))` after incomplete rows are omitted. Larger values are reduced to that limit. A component can have zero scores when the data are rank deficient or its loadings are all zero.
n_features	Integer desired number of input variables, between zero and `ncol(X)`. A variable is selected if its loading is nonzero in at least one component. This counts the union across components, not the number of coefficients in each component.
alpha	Optional nonempty numeric vector of finite nonnegative relative penalties for the initial search. Values are evaluated in the supplied order, with warm starts. `NULL` uses an increasing grid containing zero and logarithmically spaced positive penalties up to one, with at most 13 initial values. A supplied grid must contain no more than `max_evals` values. Refinement stays within the initial grid's range.
beta	Finite nonnegative relative ridge penalty (default `1e-4`), shared by all fits. Both penalties are scaled by the squared leading singular value of the processed input for `method = "spca"`, or of the shared sketch for `method = "rspca"`.

center	Logical; subtract column means before fitting (default TRUE). The fitted means are stored for prediction.
scale	Logical; divide each processed column by its scale factor (default FALSE). Factors are square roots of column sums of squares divided by $n - 1$, after any centering. This is the sample standard deviation for centered data and a root-mean-square quantity otherwise. Factors below 1e-8 are replaced by one.
max_iter	Positive integer limit on optimization iterations. Reaching the limit does not establish convergence.
tol	Finite nonnegative tolerance for relative objective change. This controls an optimization stopping rule, not a reconstruction-error bound or a statistical accuracy guarantee. An objective increase larger than a roundoff allowance stops with a warning and does not count as convergence; see Details.
method	Fit with deterministic "spca" (the default) or randomized "rspca". All randomized fits share one sketch. o and q are used only by the randomized method.
o	Nonnegative integer oversampling parameter (default 20). The sketch width is min(k + o, nrow(X), ncol(X)) after incomplete rows are omitted. Additional columns can improve the approximation while increasing memory and computation.
q	Nonnegative integer number of power iterations (default 2). Each iteration requires additional products with the processed input and its transpose, with intermediate orthogonalization. Increasing q can help for a slowly decaying singular spectrum.
verbose	Logical, or the equivalent numeric zero/one; print optimization progress when enabled.
penalty	Sparsity penalty: "elasticnet" (the default) uses an entrywise absolute-value penalty and soft thresholding; "l0" counts nonzero coefficients and uses hard thresholding. With penalty = "l0" and beta > 0, the combination of cardinality and ridge penalties is also called ZeroNet.
B_init	Optional real finite p by k loading matrix for the first candidate. NULL uses the shared factorization's leading right singular vectors. Subsequent initial-grid candidates use the preceding candidate's loadings; refinement uses the best converged candidate so far, or the closest candidate if none has converged.
max_evals	Positive integer maximum number of candidate fits, including the initial grid and refinement. The default is 30. Each candidate has its own max_iter optimization limit.
count_tol	Nonnegative integer acceptable absolute difference between the requested and achieved variable counts. The default zero searches for an exact match, while still returning the closest converged candidate if no exact match is found. This controls the search stopping rule; tol controls each fit's objective stopping rule.
x	An object returned by spca_target.
...	Additional arguments; currently ignored by the print method.

Details

The number of score coordinates k and the number of measured variables n_features answer different questions. For example, two components can each use three measurements while together

using only four distinct measurements. This function searches the total number of variables needed to calculate all requested scores.

Each candidate solves the unchanged penalized objective of spca or rspca. The procedure evaluates an initial penalty grid, then prioritizes intervals whose observed endpoint counts lie on either side of the request, followed by intervals near the closest counts. It stops when a converged candidate meets count_tol, the evaluation budget is exhausted, or there are no further penalties to evaluate. It does not assume that the count is monotone in alpha. Counts can jump over the requested value, and the nonconvex optimization can reach different solutions under different warm starts. An exact count need not be attainable within the explored candidates.

Only candidates with converged = TRUE are eligible for selection. The selected candidate minimizes the absolute count difference, with ties resolved by fewer selected variables and then earlier evaluation. Its penalized objective is not compared with objectives at other penalty strengths. A count within count_tol ends the search immediately; the procedure does not compare all representations having that count. If no candidate converges, the function warns and returns fit = NULL, with the attempted fits still recorded in search. A larger max_iter or revised fit settings may be needed in that case.

The result reports both the requested and achieved counts. A nonzero mismatch is an ordinary search outcome and leaves target_met false when it exceeds count_tol. Increasing max_evals permits more refinement, and supplying alpha changes the range and starting sequence explored. Neither change guarantees an exact count or a globally optimal subset. Coefficients are the fitted values; the function does not trim or pad the final loading matrix.

Value

A list of class "spca_target" with:

fit The selected spca or rspca object, or NULL if none of the evaluated candidates converged.

n_features The requested number of variables.

selected_features The achieved number of selected variables, or NA when no candidate was eligible.

alpha The selected relative penalty before spectral scaling, or NA. In contrast, fit$alpha contains the effective coefficient used in the objective.

target_met Whether an eligible candidate's count is within count_tol of n_features.

count_tol The allowed absolute difference in variable count.

index Row of search identifying the selected candidate, or NA when none was eligible.

search A data frame in evaluation order with columns alpha, selected_features, difference, iterations, converged, and warm_start. difference is the achieved count minus the requested count; negative values mean that fewer variables were selected. warm_start is the index of the earlier candidate supplying initial loadings, or zero for the caller-supplied/default start.

nonzero A matrix of nonzero counts, with one row per evaluated candidate and one column per component.

method, penalty The chosen fitting method and sparsity penalty.

stop_reason One of "target_met", "max_evals", "no_refinement", or "no_converged_fit".

The print method reports the requested and achieved counts, selected penalty, target status, and number of evaluated candidates.

Computation and sparse input

All candidates share the training means, scales, component count, ridge penalty, penalty type, and spectral penalty scale. The randomized method uses one sketch, so `set.seed` controls the shared approximation. With `method = "rspca"`, sparse **Matrix** input retains the implicit centering and scaling behavior described in `rspca`. The deterministic method uses dense workspace.

The search stores candidate counts and optimization summaries, with full-data scores constructed only for the selected fit. Use `spca_path` when every complete candidate fit is needed. The returned `fit` has the usual class, prediction methods, variance conventions, and loading scale of the selected method.

Interpreting the selected representation

Reaching a desired count is a computational convenience. Inspect the variables and reconstruction quality, and assess stability or validation performance when these matter to the analysis. The count does not establish that the selected measurements are unique, necessary, or causally important. A component can have zero loadings, and fewer than `k` selected variables cannot produce `k` independent scores. Zero processed data have zero selected variables regardless of the requested count.

See Also

`spca_path`, `spca`, `rspca`, `predict.spca`.

Examples

```
X <- as.matrix(iris[, 1:4])
target <- spca_target(X, k = 2, n_features = 3,
                      beta = 0.001, scale = TRUE,
                      tol = 1e-8, max_iter = 3000, verbose = FALSE)
print(target)
target$search
if (!is.null(target$fit)) {
  colnames(X)[rowSums(target$fit$loadings != 0) > 0]
  predict(target$fit, X[1:3, , drop = FALSE])
}
```

Index

predict.robspca *(*predict.spca*)*, 4
predict.rspca *(*predict.spca*)*, 4
predict.spca, *3*, 4, *9*, *13*, *19*, *21*, *25*, *29*
print.robspca *(*spca-methods*)*, 20
print.rspca *(*spca-methods*)*, 20
print.spca *(*spca-methods*)*, 20
print.spca_path *(*spca_path*)*, 22
print.spca_target *(*spca_target*)*, 25

robspca, *3–5*, 6, *13*, *19–21*
rspca, *3–5*, *9*, 10, *14*, *18–21*, *25*, *28*, *29*

sparsepca *(*sparsepca-package*)*, 2
sparsepca-package, 2
spca, *2–5*, *7–9*, *11–13*, 14, *20*, *21*, *25*, *28*, *29*
spca-methods, 20
spca_path, *3*, *11–13*, *17*, *19*, 22, *29*
spca_target, *3*, *17*, *24*, *25*, 25
summary.robspca *(*spca-methods*)*, 20
summary.rspca *(*spca-methods*)*, 20
summary.spca *(*spca-methods*)*, 20