scanpy.tl.pca

scanpy.tl.pca(*data*, *n_comps=None*, *zero_center=True*, *svd_solver='arpack'*, *random_state=0*, *return_info=False*, *use_highly_variable=None*, *dtype='float32'*, *copy=False*, *chunked=False*, *chunk_size=None*)

Principal component analysis [Pedregosa11].

Computes PCA coordinates, loadings and variance decomposition. Uses the implementation of *scikit-learn* [Pedregosa11].

Changed in version 1.5.0: In previous versions, computing a PCA on a sparse matrix would make a dense copy of the array for mean centering. As of scanpy 1.5.0, mean centering is implicit. While results are extremely similar, they are not exactly the same. If you would like to reproduce the old results, pass a dense array.

Parameters:

data : `Union` [`AnnData` , `ndarray` , `spmatrix`]

The (annotated) data matrix of shape `n_obs` × `n_vars`. Rows correspond to cells and columns to genes.

n_comps : `Optional` [`int`] (default: `None`)

Number of principal components to compute. Defaults to 50, or 1 - minimum dimension size of selected representation.

zero_center : `Optional` [`bool`] (default: `True`)

If `True`, compute standard PCA from covariance matrix. If `False`, omit zero-centering variables (uses `TruncatedSVD`), which allows to handle sparse input efficiently. Passing `None` decides automatically based on sparseness of the data.

svd_solver : `str` (default: `'arpack'`)

SVD solver to use:

`'arpack'` (the default)

for the ARPACK wrapper in SciPy (`svds()`)

`'randomized'`

for the randomized algorithm due to Halko (2009).

`'auto'`

chooses automatically depending on the size of the problem.

`'lobpcg'`

An alternative SciPy solver.

Changed in version 1.4.5: Default value changed from `'auto'` to `'arpack'`.

Efficient computation of the principal components of a sparse matrix currently only works with the `'arpack'` or `'lobpcg'` solvers.

---

random_state : **`Union`** [**`None`** , **`int`** , **`RandomState`**] (default: `0`)

Change to use different initial states for the optimization.

---

return_info : **`bool`** (default: `False`)

Only relevant when not passing an **`AnnData`**: see “**Returns**”.

---

use_highly_variable : **`Optional`** [**`bool`**] (default: `None`)

Whether to use highly variable genes only, stored in `.var['highly_variable']`. By default uses them if they have been determined beforehand.

---

dtype : **`str`** (default: `'float32'`)

Numpy data type string to which to convert the result.

---

copy : **`bool`** (default: `False`)

If an **`AnnData`** is passed, determines whether a copy is returned. Is ignored otherwise.

---

chunked : **`bool`** (default: `False`)

If `True`, perform an incremental PCA on segments of `chunk_size`. The incremental PCA automatically zero centers and ignores settings of `random_seed` and `svd_solver`. If `False`, perform a full PCA.

---

chunk_size : **`Optional`** [**`int`**] (default: `None`)

Number of observations to include in each chunk. Required if `chunked=True` was passed.

Return type: **`Union`** [**`AnnData`** , **`ndarray`** , **`spmatrix`**]

Returns: : X_pca : `spmatrix`, `ndarray`

If `data` is array-like and `return_info=False` was passed, this function only returns `X_pca`…

adata : `AnnData`

…otherwise if `copy=True` it returns or else adds fields to `adata`:

`.obsm['X_pca']`

PCA representation of data.

`.varm['PCs']`

The principal components containing the loadings.

`.uns['pca']['variance_ratio']`

Ratio of explained variance.

`.uns['pca']['variance']`

Explained variance, equivalent to the eigenvalues of the covariance matrix.