sklearn.decomposition.PCA ¶

class `sklearn.decomposition.`**`PCA`**(*n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*) [source]

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

Notice that this class does not support sparse input. See `TruncatedSVD` for an alternative with sparse data.

Read more in the User Guide.

Parameters:

n_components : *int, float, None or str*

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'` and `svd_solver == 'full'`, Minka's MLE is used to guess the dimension. Use of `n_components == 'mle'` will interpret `svd_solver == 'auto'` as `svd_solver == 'full'`.

If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

If `svd_solver == 'arpack'`, the number of components must be strictly less than the minimum of n_features and n_samples.

Hence, the None case results in:

```
n_components == min(n_samples, n_features) - 1
```

copy : *bool, default=True*

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

whiten : *bool, optional (default False)*

When True (False by default) the `components_` vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

svd_solver : *str {'auto', 'full', 'arpack', 'randomized'}*

If auto :

The solver is selected by a default policy based on `X.shape` and `n_components`: if the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards.

If full :

run exact full SVD calling the standard LAPACK solver via `scipy.linalg.svd` and select the components by postprocessing

If arpack :

run SVD truncated to n_components calling ARPACK solver via `scipy.sparse.linalg.svds`. It requires strictly 0 < n_components < min(X.shape)

If randomized :

run randomized SVD by the method of Halko et al.

New in version 0.18.0.

Toggle Menu **>= 0, optional (default .0)**

Tolerance for singular values computed by svd_solver == 'arpack'.

New in version 0.18.0.

iterated_power : *int >= 0, or 'auto', (default 'auto')*
Number of iterations for the power method computed by svd_solver == 'randomized'.

New in version 0.18.0.

random_state : *int, RandomState instance or None, optional (default None)*
If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`. Used when `svd_solver` == 'arpack' or 'randomized'.

New in version 0.18.0.

Attributes:

components_ : *array, shape (n_components, n_features)*
Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by `explained_variance_`.

explained_variance_ : *array, shape (n_components,)*
The amount of variance explained by each of the selected components.

Equal to n_components largest eigenvalues of the covariance matrix of X.

New in version 0.18.

explained_variance_ratio_ : *array, shape (n_components,)*
Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

singular_values_ : *array, shape (n_components,)*
The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

New in version 0.19.

mean_ : *array, shape (n_features,)*
Per-feature empirical mean, estimated from the training set.

Equal to `X.mean(axis=0)`.

n_components_ : *int*
The estimated number of components. When n_components is set to 'mle' or a number between 0 and 1 (with svd_solver == 'full') this number is estimated from input data. Otherwise it equals the parameter n_components, or the lesser value of n_features and n_samples if n_components is None.

n_features_ : *int*
Number of features in the training data.

n_samples_ : *int*
Number of samples in the training data.

noise_variance_ : *float*
The estimated noise covariance following the Probabilistic PCA model from Tipping and Bishop 1999. See "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf. It is required to compute the estimated data covariance and score samples.

Equal to the average of (min(n_features, n_samples) - n_components) smallest eigenvalues of the covariance matrix of X.

See also:

`KernelPCA`
Kernel Principal Component Analysis.

`SparsePCA`
Sparse Principal Component Analysis.

`VD`

Dimensionality reduction using truncated SVD.

`IncrementalPCA`
Incremental Principal Component Analysis.

References

For n_components == 'mle', this class uses the method of *Minka, T. P. "Automatic choice of dimensionality for PCA". In NIPS, pp. 598-604*

Implements the probabilistic PCA model from: Tipping, M. E., and Bishop, C. M. (1999). "Probabilistic principal component analysis". Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61(3), 611-622. via the score and score_samples methods. See http://www.miketipping.com/papers/met-mppca.pdf

For svd_solver == 'arpack', refer to `scipy.sparse.linalg.svds`.

For svd_solver == 'randomized', see: *Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". SIAM review, 53(2), 217-288.* and also *Martinsson, P. G., Rokhlin, V., and Tygert, M. (2011). "A randomized algorithm for the decomposition of matrices". Applied and Computational Harmonic Analysis, 30(1), 47-68.*

Examples

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
>>> print(pca.singular_values_)
[6.30061... 0.54980...]
```

```
>>> pca = PCA(n_components=2, svd_solver='full')
>>> pca.fit(X)
PCA(n_components=2, svd_solver='full')
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.00755...]
>>> print(pca.singular_values_)
[6.30061... 0.54980...]
```

```
>>> pca = PCA(n_components=1, svd_solver='arpack')
>>> pca.fit(X)
PCA(n_components=1, svd_solver='arpack')
>>> print(pca.explained_variance_ratio_)
[0.99244...]
>>> print(pca.singular_values_)
[6.30061...]
```

Methods

`fit`(self, X[, y])	Fit the model with X.
`fit_transform`(self, X[, y])	Fit the model with X and apply the dimensionality reduction on X.
`get_covariance`(self)	Compute data covariance with the generative model.
`get_params`(self[, deep])	Get parameters for this estimator.
`get_precision`(self)	Compute data precision matrix with the generative model.
`inverse_transform`(self, X)	Transform data back to its original space.
`score`(self, X[, y])	Return the average log-likelihood of all samples.
`score_samples`(self, X)	Return the log-likelihood of each sample.
`set_params`(self, **params)	Set the parameters of this estimator.
`transform`(self, X)	Apply dimensionality reduction to X.

`__init__`(*self, n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None*) [source]

Initialize self. See help(type(self)) for accurate signature.

`fit`(*self, X, y=None*) [source]

Fit the model with X.

Parameters:

X : *array-like, shape (n_samples, n_features)*

Training data, where n_samples is the number of samples and n_features is the number of features.

Toggle Menu

y : *None*
Ignored variable.

Returns:

self : *object*
Returns the instance itself.

`fit_transform`(*self*, *X*, *y=None*) [source]

Fit the model with X and apply the dimensionality reduction on X.

Parameters:

X : *array-like, shape (n_samples, n_features)*
Training data, where n_samples is the number of samples and n_features is the number of features.

y : *None*
Ignored variable.

Returns:

X_new : *array-like, shape (n_samples, n_components)*
Transformed values.

Notes

This method returns a Fortran-ordered array. To convert it to a C-ordered array, use 'np.ascontiguousarray'.

`get_covariance`(*self*) [source]

Compute data covariance with the generative model.

`cov = components_.T * S**2 * components_ + sigma2 * eye(n_features)` where S**2 contains the explained variances, and sigma2 contains the noise variances.

Returns:

cov : *array, shape=(n_features, n_features)*
Estimated covariance of data.

`get_params`(*self*, *deep=True*) [source]

Get parameters for this estimator.

Parameters:

deep : *bool, default=True*
If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns:

params : *mapping of string to any*
Parameter names mapped to their values.

`get_precision`(*self*) [source]

Compute data precision matrix with the generative model.

Equals the inverse of the covariance but computed with the matrix inversion lemma for efficiency.

Returns:

precision : *array, shape=(n_features, n_features)*
Estimated precision of data.

`inverse_transform`(*self*, *X*) [source]

Toggle Menu

data back to its original space.

In other words, return an input X_original whose transform would be X.

Parameters:

X : *array-like, shape (n_samples, n_components)*
New data, where n_samples is the number of samples and n_components is the number of components.

Returns:

X_original array-like, shape (n_samples, n_features)

Notes

If whitening is enabled, inverse_transform will compute the exact inverse operation, which includes reversing whitening.

`score`(*self*, *X*, *y=None*) [source]

Return the average log-likelihood of all samples.

See. "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf

Parameters:

X : *array, shape(n_samples, n_features)*
The data.

y : *None*
Ignored variable.

Returns:

ll : *float*
Average log-likelihood of the samples under the current model.

`score_samples`(*self*, *X*) [source]

Return the log-likelihood of each sample.

See. "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf

Parameters:

X : *array, shape(n_samples, n_features)*
The data.

Returns:

ll : *array, shape (n_samples,)*
Log-likelihood of each sample under the current model.

`set_params`(*self*, ***params*) [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters:

****params : *dict***
Estimator parameters.

Returns:

self : *object*
Estimator instance.

`transform`(*self*, *X*) [source]

Apply dimensionality reduction to X.

X is projected on the first principal components previously extracted from a training set.

Parameters:

X : *array-like, shape (n_samples, n_features)*

New data, where n_samples is the number of samples and n_features is the number of features.

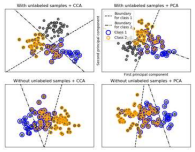
Returns:

X_new : *array-like, shape (n_samples, n_components)*

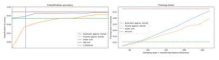
Examples

```
>>> import numpy as np
>>> from sklearn.decomposition import IncrementalPCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> ipca = IncrementalPCA(n_components=2, batch_size=3)
>>> ipca.fit(X)
IncrementalPCA(batch_size=3, n_components=2)
>>> ipca.transform(X) # doctest: +SKIP
```

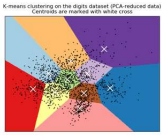
Examples using `sklearn.decomposition.PCA`



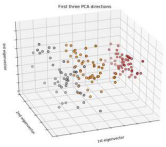
Multilabel classification



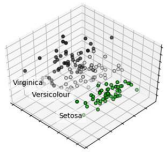
Explicit feature map approximation for RBF kernels



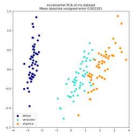
A demo of K-Means clustering on the handwritten digits data



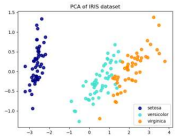
The Iris Dataset



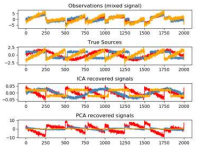
PCA example with Iris Data-set



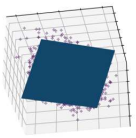
Incremental PCA



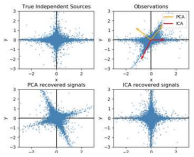
Comparison of LDA and PCA 2D projection of Iris dataset



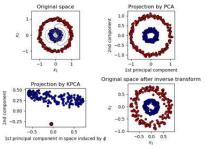
Blind source separation using FastICA



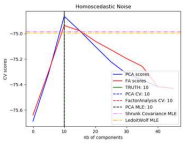
Principal components analysis (PCA)



FastICA on 2D point clouds



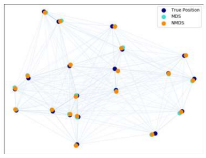
Kernel PCA



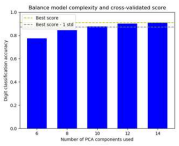
Model selection with Probabilistic PCA and Factor Analysis (FA)

Faces dataset decompositions

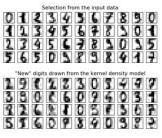
Faces recognition example using eigenfaces and SVMs



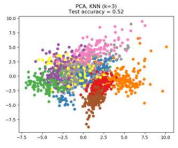
Multi-dimensional scaling



Balance model complexity and cross-validated score

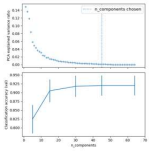


Kernel Density Estimation



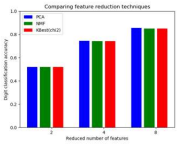
Dimensionality Reduction with Neighborhood Components Analysis

Concatenating multiple feature extraction methods

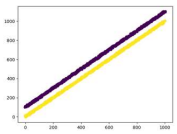


Pipelining: chaining a PCA and a logistic regression

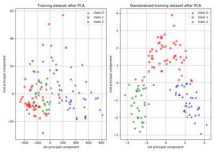
Toggle Menu



[Selecting dimensionality reduction with Pipeline and GridSearchCV]



[Using FunctionTransformer to select columns]



[Importance of Feature Scaling]