# PCA

**PCA( method = 'defaultDense',covariance = "defaultDense", normalization="zscore", nComponents = 0,isDeterministic = False,fptype=float64, resultsToCompute = 0)**

Constructor to PCA computation parameters

*parameters:*

**method:** 'defaultDense'/'svdDense', default: 'defaultDense'

used to decide the calculation method. 'defaultDense' is covariance

**covariance:** (applicable only if method='defaultDense'), default : 'defaultDense'

available values - singlePassDense, sumDense, fastCSR, singlePassCSR, sumCSR

**normalization :** (applicable only if method='svdDense')default: 'zscore'

available values - zscore, minmax

**nComponents:** default 0

if 0 number of components is number of features

**isDeterministic**: default: False

If true, the algorithm applies the "sign flip" technique to the results.

**resultsToCompute**: default: None

The 64-bit integer flag that specifies which optional result to compute.

Provide one of the following values to request a single characteristic or use bitwise OR to request a combination of the characteristics:

mean

variance

eigenvalue

## Methods:

1. **compute(data, corrDataFlag=False)**

*parameters:*

inuput data(type nT), corrDataFlag(Is input data correlation matrix)(type Boolean): default(False) *returns*:

pca computation result

1. **getResults (pcaResult)**

*parameters:*

computed pca results(type pca object)

*returns*:

Dictionary with eigenvectors(type nT), eigenvalues(type nT), means(type nT), variances(type nT), dataForTransform(type nT)

1. **transform(pcaResult,data, nComponents=None, useDataForTransformation = False)**

*parameters :*

computed pca results(type pca result object)

data to be transformed(type nT)

number of priciple components(type int): default: 0(use all components)

useDataForTransformation(type Bool): default: Flase(refer dataForTranform parameter for developer's guide for more details)

*returns:*

transformed data

1. **compress(arrayData)**

*parameters:*

serialized numpy array

*returns*:

Compressed numpy array

1. **decompress(arrayData)**

*parameters:*

deserialized numpy array

*returns*:

decompressed numpy array

1. **serialize(data, fileName=None, useCompression=False)**

*parameters:*

Method 1: data(type nT/model)

-Returns serialized numpy array

Method 2: data(type nT/model), fileName(.npy file to save serialized array to disk)

- Saves serialized numpy array as "fileName" argument

Method 3: data(type nT/model), useCompression = True

-Returns compressed numpy array

Method 4: data(type nT/model), fileName(.npy file to save serialized array to disk), useCompression = True

-Saves compressed numpy array as "fileName" argument

1. **deserialize(serialObjectDict=None, fileName=None, useCompression=False)**

*parameters:*

serialized/ compressed numpy array or serialized/ compressed .npy file

*returns*:

deserialized/ decompressed numeric table/model