Local PCA Suite

Python Documentation

Matteo Savarese

August 30, 2024

Contents

1 Introduction

Brief description of the package capabilities and functionalities

2 Installation

Instructions on how to install the package, including dependencies.

3 Code description

3.1 Preprocess.py

This file contains the class and functions for pre-processing the data. In particular, the scaler class (similar to scikit-learn) allows to scale and center data according to different methods, described below.

3.1.1 `Scaler`

The `Scaler` class is designed for preprocessing data by centering and scaling features using various methods. This class is useful for normalizing data before applying machine learning models or other statistical analyses. Below is a detailed description of the class methods and their functionalities.

Constructor

The constructor initializes the `Scaler` object with the specified scaling method and centering option.

- `method`: A string specifying the scaling method to be used. Possible values are:
  - `auto`: Standard scaling, divides by the standard deviation.
  - `pareto`: Divides by the square root of the standard deviation.
  - `range`: Scales to the range of the data.
  - `vast`: Multiplies the standard deviation by the mean.
  - `max`: Scales by the maximum value.
- `center`: A boolean flag (default: `True`) to indicate whether to center the data by subtracting the mean.

Example: `scaler = Scaler(method='auto', center=True)`

fit

The `fit` method calculates the centering and scaling factors based on the input data matrix `X`.

- **Input:**
  - `X` (`numpy.ndarray`): A 2D array where rows represent samples and columns represent features.
- **Output:**
  - The method returns the `Scaler` object itself, with the computed centering vector (`c`) and scaling factors (`gamma`) stored as attributes.

Example: `scaler.fit(X)`

fit_transform

The `fit_transform` method first fits the `Scaler` to the data and then applies the transformation.

- **Input:**
  - `X` (`numpy.ndarray`): A 2D array to be scaled.
  - `center` (`bool`, default=`True`): Whether to center the data during transformation.
- **Output:**
  - `Xs` (`numpy.ndarray`): The scaled data matrix, after centering (if applicable) and scaling.

Example: `Xs = scaler.fit_transform(X)`

transform

The `transform` method scales the data using the previously computed centering and scaling factors. This method assumes that the `fit` method has already been called.

- `X`: A 2D NumPy array to be scaled.
- `center`: A boolean flag (default: `True`) to indicate whether to center the data during transformation.
- Returns: The scaled data as a 2D NumPy array.
- **Input:**
  - `X` (`numpy.ndarray`): A 2D array to be scaled.
  - `center` (`bool`, default=`True`): Whether to center the data during transformation.
- **Output:**
  - `Xs` (`numpy.ndarray`): The scaled data matrix.

  Example: `Xs = scaler.transform(X)`

inverse_transform

The `inverse_transform` method reverses the scaling operation, returning the original data values.

- **Input:**
  - `X` (`numpy.ndarray`): A 2D array that has been previously scaled.
- **Output:**
  - `Xi` (`numpy.ndarray`): The original data matrix, after reversing the centering and scaling.

  Example: `Xi = scaler.inverse_transform(Xs)`

3.2 Clustering.py

This file contains the classes and methods that can be used mainly for clustering. The `vqpls` class can also be used for local linear regression, especially for PC transport applications.

3.2.1 `vqpca`

The `vqpca` class implements the Vector Quantization Principal Component Analysis (VQPCA) routine, which clusters the data based on the minimization of the local squared reconstruction error $\epsilon_i^{(j)}$.

Constructor

The constructor initializes the `vqpca` class as follows:

- **Input:**
  - `X` (`numpy.ndarray`): The data matrix where rows represent observations and columns represent variables.
  - `stopping_rule` (`str`, default=`"variance"`): The rule to determine when the algorithm should stop. Options include:
    - `"variance"`: Stop when a certain percentage of variance is explained.
    - `"n_eigs"`: Stop after extracting a fixed number of eigenvalues.
  - `itmax` (`int`, default=`200`): The maximum number of iterations allowed for convergence.
  - `atol` (`float`, default=`1e-8`): The absolute tolerance level for convergence.
  - `rtol` (`float`, default=`1e-8`): The relative tolerance level for reconstruction error convergence.
  - `ctol` (`float`, default=`1e-6`): The tolerance level for centroid convergence.

fit

The `fit` method is used to run the `vqpca` routine.

Inputs

- `k` (`int`, default=`2`): The number of clusters to form in the data.
- `q` (`float`, default=`0.99`): The amount of variance to explain (or the number of principal components) in each PCA model within a cluster.
- `n_start` (`int`, default=`2`): The number of principal components used during the first iteration of the algorithm.
- `init_centroids` (`str`, default=`'kmeans'`): Method for initializing the centroids. Available options include:
  - `'kmeans'`: Use the k-means algorithm to initialize centroids.
  - `'random'`: Randomly initialize centroids.
  - `'uniform'`: Uniformly initialize centroids.
- `verbose` (`bool`, default=`True`): If `True`, the method will print detailed information about the iteration process, including convergence diagnostics and timing.

Outputs

- `labels_new` (`numpy.ndarray`): A vector of integers where each element corresponds to the cluster label assigned to the corresponding observation in `X`.
- `self` (`vqpca` object): The updated `vqpca` object with the following attributes:
  - `pca_` (`list` of `PCA` objects): The list of PCA models fitted to each cluster.
  - `C_` (`numpy.ndarray`): The matrix of centroids for the clusters.
  - `labels_` (`numpy.ndarray`): The final cluster labels after convergence.

- `eps_` (`float`): The mean squared global reconstruction error.
- `A_` (`numpy.ndarray`): The matrix of PCA components.

Example Usage:

```
vq = vqpca(X)
labels = vq.fit(k=3, q=0.95, n_start=5, verbose=True)
```

reconstruct

This function can be used to test the reconstruction performance of the converged VQPCA basis. In particular, given a matrix `X`, this is first projected onto the low-dimensional manifold and then reconstructed. Note that the matrix `X` can also be a new one, like a test matrix different from the training one.

Inputs

- `X` (`2D numpy.ndarray`, default=`None`): The data matrix to be reconstructed. If `None`, the original data matrix `self.X_` used during the fitting process will be reconstructed.

Output

- `X_rec` (`2D numpy.ndarray`): The reconstructed data matrix, where each observation has been transformed back from its lower-dimensional representation using the principal components of its assigned cluster.

get_components

This function can be used to get the PC basis A

Output

- `components` (`list`): A list containing the principal component basis in each cluster

Example Usage:

```
model = vqpca()
model.fit()
comp = model.get_components()
```

write_output_files

Input Parameters

- `foldername` (`str`, default=`None`): The name of the folder where the output files will be saved. If `None`, a folder name will be generated based on the current date, the number of clusters `k`, and the PCA stopping criterion `q`.
- `format` (`str`, default=`'h5'`): The format of the output files. Available options are `'h5'` for HDF5 files and `'txt'` for plain text files.

Output

- None: The method does not return any values but creates and saves output files in the specified folder.

External outputs

The function generates files in the specified paths, namely:

- If `'h5'` option is chosen, a file named `'output.h5'` is generated containing the following fields: `labels`, `reconstruction_error`, `centroids`
- If `'txt'` option is chosen, two files are created: `labels.txt` and `centroids.txt` [1]

3.2.2 `vqpls`

This class implements the Vector Quantization Partial Least Square (VQPLS) routine, which is based on local PLS rather than local PCA.

Vector Quantization Partial Least Square

Partial Least Squares (PLS) is a terminology that refers to a wide range of methods whose intent is to model a relationship between some independent and some dependent variables by projecting them to a latent space to get new latent variables. The central idea of PLS is to identify orthogonal projections that maximize the covariance between the independent variables, denoted as $\mathcal{X} \subset \mathbb{R}^N$, and the dependent variables $\mathcal{Y} \subset \mathbb{R}^M$. Suppose that the observed data for $\mathcal{X}$ are stored in a matrix $\mathbf{X} \in \mathbb{R}^{n \times N}$, where the $i-$th rows represent the observation and the $j-$th column represent the variable in the $\mathcal{X}$ space, and the observed target variable $\mathcal{Y}$, similarly, is stored in the matrix $\mathbf{Y} \in \mathbb{R}^{n \times M}$. Then, we seek to project the observed matrix onto new sets of directions, such that:

$$\begin{cases} \mathbf{X} = \mathbf{T}\mathbf{P}^{\mathrm{T}} + \mathbf{E} \\ \mathbf{Y} = \mathbf{U}\mathbf{Q}^{\mathrm{T}} + \mathbf{F} \end{cases} \tag{1}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are $N \times p$ and $M \times p$ orthogonal basis, where p is the number of retained latent components. Therefore, the matrices $\mathbf{T} \in \mathbb{R}^{n \times p}$ and $\mathbf{U} \in \mathbb{R}^{n \times p}$ are the projections of $\mathbf{X}$ and $\mathbf{Y}$ onto the new directions, also known as scores. The matrices $\mathbf{E} \in \mathbb{R}^{n \times N}$ and $\mathbf{F} \in \mathbb{R}^{n \times M}$ are the residuals. It is evident that such a decomposition shows similarities with the well-known Principal Component Analysis (PCA). However, for PLS, the directions are identified where the covariance between $\mathbf{X}$ and $\mathbf{Y}$ is maximized using an iterative algorithm (NIPALS), while for PCA, the new directions of $\mathbf{X}$ are found where only the covariance of $\mathbf{X}$ is maximized.

In the Vector Quantization PLS routine, we seek to find a partition of the data which minimizes the sum of squared projection distances between the observed target data $\mathbf{Y}$ and their PLS reconstruction. The reconstruction error for the $i-$th observation can be written as $\epsilon_{rec,i} = |\mathbf{y_i} - \mathbf{u_i}\mathbf{Q}^{\mathrm{T}}|^2$. Therefore, for a given number of partitions k, we seek to find a set of local basis $\mathbf{Q}_{(l)}$, with $l = 1, ..., k$, such that the following distortion function is minimized:

$$J_{VQPLS} = \sum_{i=1}^{n} \sum_{l=1}^{k} \epsilon_{rec,i}^{(l)} \tag{2}$$

This procedure bears similarities with both the $k-$means and the Vector Quantization PCA (VQPCA) algorithms. This routine is solved in the conventional Lloyd-type scheme:

1. **Initialization**: the k centroids are initialized
2. **Local PLS**: PLS is performed in each cluster

[1] If large DNS datasets are analyzed, we recommend saving in h5 format

3. **Assignment**: each observation is assigned to the cluster where the local PLS projection of $\mathbf{y}_i^{(l)}$ is minimized
4. **Update**: the local basis are updated
5. **Convergence**: convergence is reached when centroids do not change significantly

The method is implemented in the `vqpls` class as follows.

Constructor

The constructor of the `vqpls` class initializes the parameters necessary for running the Vector Quantization Partial Least Square (VQPLS) routine. This class is designed to handle the VQPLS algorithm, which integrates vector quantization with partial least squares regression for dimensionality reduction and prediction.

Inputs

- `ctol` (`float`, default=`1e-6`): The tolerance level for the convergence of centroids. The algorithm stops when the change in centroids is less than this threshold.
- `Rtol` (`float`, default=`1e-6`): The tolerance level for the convergence of the R-squared value. The algorithm stops when the change in R-squared is below this threshold.
- `itmax` (`int`, default=`100`): The maximum number of iterations allowed for the algorithm to run. If this limit is reached, the algorithm will terminate.
- `verbose` (`bool`, default=`True`): Controls the verbosity of the output during the algorithm's execution. If set to `True`, progress and convergence information will be printed to the console.
- `early_stopping` (`bool`, default=`False`): Enables or disables early stopping. If `True`, the algorithm will stop early if there is no significant improvement in the results within a specified number of iterations.
- `patience` (`int`, default=`10`): The number of iterations with no significant improvement before the algorithm stops early. This parameter is only relevant if `early_stopping` is set to `True`.

fit

The fit function is used to run the VQPLS routine on a data matrix $\mathbf{X}$ and a target matrix $\mathbf{Y}$.

- `X` (`2D NumPy array`): The matrix of independent variables. Each row represents an observation, and each column represents a variable.
- `Y` (`2D NumPy array`): The matrix of target variables. It should have the same number of rows as `X`.
- `init` (`str`, default=`'kmeans'`): The method used for initializing the centroids. Available options are:
  - `'kmeans'`: Initialize centroids using k-means clustering.
  - `'random'`: Initialize centroids randomly.
  - `'global_pls'`: Initialize centroids using a global PLS model.
- `n_clusters` (`int`, default=`2`): The number of clusters to divide the data into.
- `n_components` (`int`, default=`3`): The number of components to retain in the PLS models.

Output

- `self`: The method updates the class instance with the following attributes:
  - `self.labels`: The cluster labels assigned to each observation in `X`.
  - `self.centroids`: The centroids of the clusters after convergence.
  - `self.pls`: The list of PLS models, one for each cluster.
  - `self.mse`: The mean squared error for the model.
  - `self.r2`: The R-squared value for the model.

Algorithm Description

1. **Initialization:**
   - The centroids are initialized using the method specified in the `init` parameter.
   - PLS models are initialized for each cluster.
2. **Iteration:** The algorithm iteratively assigns each observation to the cluster that minimizes the sum of squared residuals. The centroids and PLS models are updated in each iteration.
3. **Convergence:** The algorithm checks for convergence based on:
   - **Centroid movement:** If the change in centroids is below a certain tolerance (`ctol`), the algorithm converges.
   - **MSE and R-squared:** If the change in Mean Squared Error (MSE) or R-squared (`Rtol`) is below a threshold, the algorithm converges.
   - **Early stopping:** If early stopping is enabled (`es=True`), the algorithm stops if there is no improvement in MSE for a specified number of iterations (`patience`).
   - **Maximum iterations:** The algorithm stops if the maximum number of iterations (`itmax`) is reached.
4. **Output:** The method returns the updated class instance, which includes the cluster labels, centroids, and PLS models.

Example usage

```
vq = vqpls()
vq.fit(X, Y, init='kmeans', n_clusters=5, n_components=3)
```

fit_from_labels

Description: This method trains the local Partial Least Squares (PLS) models given a set of labels without performing the full Vector Quantization Partial Least Squares (VQPLS) routine. This can be useful when working with large datasets where previously computed cluster labels are available, allowing for direct PLS model training.

Inputs:

- `X` (2D NumPy array): Independent variables data matrix.
- `Y` (2D NumPy array): Target variables data matrix.
- `labels` (1D NumPy array): Cluster labels vector.

- `n_components` (int): Number of retained components in the PLS model. The default is 3.

Output:

- `self` (`vqpls` class): The updated class instance with trained PLS models and updated centroids.

predict

This function can be used to predict the target variables from test data $\mathbf{X_{test}}$.

Inputs

- `X` (2D NumPy array): Independent variables data matrix for which predictions are to be made.

Output:

- `Y_pred` (2D NumPy array): Predicted values of the dependent variables, with the same number of rows as $\mathbf{X}$ and `self.nY` columns.

transform

This function projects both the independent matrix $\mathbf{X}$ and the independent variables matrix $\mathbf{Y}$ onto the new directions found by the local PLS basis.

Inputs:

- `X` (2D NumPy array): Independent variables data matrix that needs to be projected onto the score space.

Output:

- `Z_score` (2D NumPy array): The data matrix projected onto the score space, with the same number of rows as X and `self.n_components` columns.

train_linear_models

This function is used to perform local linear regression in the clusters, in case we want to optimize the regression models and improve performance from the local PLS ones. The `scikit-learn` implementation for linear regression is used [ref.].

Inputs:

- `X` (2D NumPy array): Independent variables data matrix.
- `Y` (2D NumPy array): Target variables data matrix.
- `test_size` (float): Proportion of the dataset to include in the test split (default is 0.20).
- `regressor` (str): Type of regression model to use. Available options are:
  - `'linear'`: Standard linear regression (default).
  - `'Lasso'`: Lasso regression (L1 regularization).
  - `'Ridge'`: Ridge regression (L2 regularization).
- `random_state` (int): Seed for the random number generator used in the train-test split (default is 0).

Outputs:

- `self.linear_models` (list): List of trained linear models for each cluster.
- `local_mse` (list): List of mean squared error values for each cluster's test set.
- `local_r2` (list): List of R^2 scores for each cluster's test set.

Notes:

- The method raises a `ValueError` if the VQPLS model has not been trained before calling this function.
- It checks that the input matrices X and Y have compatible dimensions and that X corresponds to the original matrix used in the VQPLS model.
- For each cluster, the data is split into training and testing sets according to the specified `test_size`.
- The method supports linear, Lasso, and Ridge regression models. If an unrecognized regressor is specified, it defaults to linear regression.

Example Usage:

```
linear_models, mse, r2 = vqpls_model.train_linear_models(X, Y, regressor='Ridge')
```

predict_linear_models_unseen

This function can be used to predict the target variables from unseen data.
Inputs:

- `X` (2D NumPy array): Independent variables data matrix for the new observations to be predicted.

Outputs:

- `Y_pred` (2D NumPy array): Predicted target variables for the input data points. Each row corresponds to the prediction for an individual observation.

Notes:

- The method raises a `ValueError` if any of the following conditions are not met:
  - The VQPLS model must be trained first (i.e., `labels` must be an attribute).
  - The local linear models must be trained (i.e., `linear_models` must be an attribute).
  - The classifier must be trained (i.e., `classifier` must be an attribute).
- The method utilizes the classifier to determine the appropriate local linear model for each new data point and then performs the prediction using that model.

Example Usage:

```
predictions = vqpls_model.predict_linear_models_unseen(new_data)
```

train_classifier

This method trains a Random Forest Classifier (RFC) on the data used in the Vector Quantization Partial Least Squares (VQPLS) routine. The classifier is then used to assign unseen observations to clusters, ensuring that the correct model is used for prediction. The `scikit-learn` implementation is used.

Inputs

- `X` (2D NumPy array): Independent variables data matrix.
- `test_size` (float): Proportion of the dataset to include in the test split (default is 0.20).
- `random_state` (int): Seed for the random number generator used in the train-test split (default is 0).
- `n_estimators` (int): Number of trees in the random forest classifier (default is 100).

Outputs:

- `rf` (RandomForestClassifier object): The trained random forest classifier.
- `self` (vqpls object): The updated class instance with the trained classifier as an attribute.

classify_unseen

This function is used to classify unseen observations. Note that the classifier must be already trained.

Inputs

- `X` (2D NumPy array): Independent variables data matrix for the unseen observations.

Outputs:

- `idxc` (1D NumPy array): Cluster indices assigned to the input data points.

3.3 Evaluation.py

This file implements various methods for evaluating the clustering *goodness*. Clustering evaluation is necessary when comparing multiple solutions for the correct choice of the number of clusters and, in the case of VQPCA and VQPLS, also for the correct number of components.

3.3.1 `evaluate`

This is the main function to perform clustering evaluation and requires the data matrix **X** and the labels array. Please note that this function can also be used to judge clustering solutions from other algorithms, i.e. k-means.

Inputs:

- `X` (2D NumPy array): Data matrix where rows represent observations and columns represent features.
- `labels` (1D NumPy array): Array of cluster labels corresponding to the observations in `X`.

- `q` (float, optional): Variance threshold or number of principal components to retain for the 'ILPCA' method. Default is 0.99.
- `score` (str, optional): Method for evaluation. Available options are 'ILPCA', 'DB', and 'silhouette'. Default is 'ILPCA'.

Outputs:

- `metric` (float): Value of the evaluated index. The metric value represents the quality of clustering based on the chosen evaluation method.

Definition:

```
metric = evaluate(X, labels, q=0.99, score='ILPCA')
```

4 Examples

Examples to come.