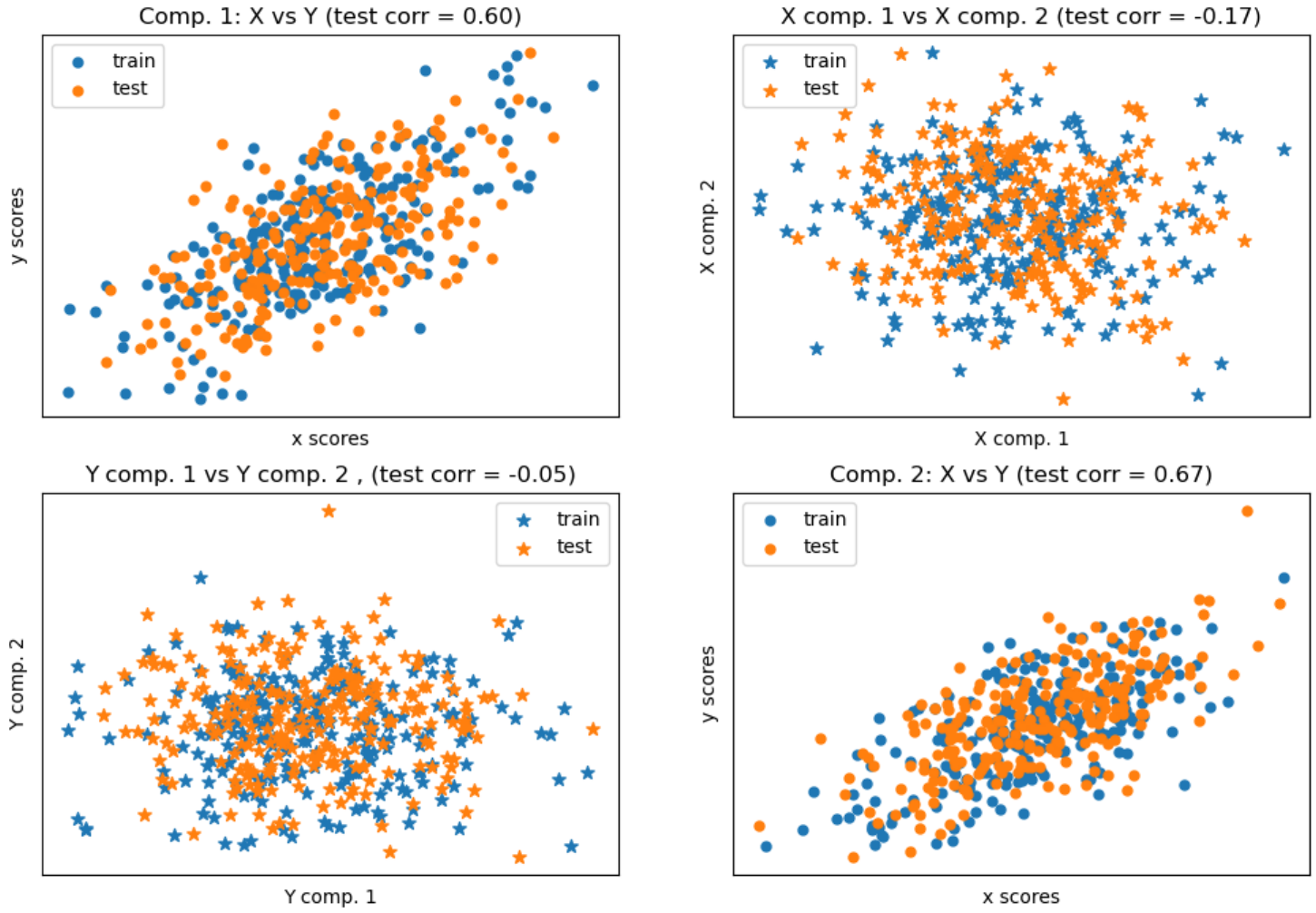
1.8. Cross decomposition

The cross decomposition module contains **supervised** estimators for dimensionality reduction and regression, belonging to the "Partial Least Squares" family.



Cross decomposition algorithms find the fundamental relations between two matrices (X and Y). They are latent variable approaches to modeling the covariance structures in these two spaces. They will try to find the multidimensional direction in the X space that explains the maximum multidimensional variance direction in the Y space. In other words, PLS projects both `X` and `Y` into a lower-dimensional subspace such that the covariance between `transformed(X)` and `transformed(Y)` is maximal.

PLS draws similarities with Principal Component Regression (PCR), where the samples are first projected into a lower-dimensional subspace, and the targets `y` are predicted using `transformed(X)`. One issue with PCR is that the dimensionality reduction is unsupervized, and may lose some important variables: PCR would keep the features with the most variance, but it's possible that features with a small variances are relevant from predicting the target. In a way, PLS allows for the same kind of dimensionality reduction, but by taking into account the targets `y`. An illustration of this fact is given in the following example: * Principal Component Regression vs Partial Least Squares Regression.

Apart from CCA, the PLS estimators are particularly suited when the matrix of predictors has more variables than observations, and when there is multicollinearity among the features. By contrast, standard linear regression would fail in these cases unless it is regularized.

Classes included in this module are **`PLSRegression`**, **`PLSCanonical`**, **`CCA`** and **`PLSSVD`**

1.8.1. PLSCanonical

We here describe the algorithm used in **`PLSCanonical`**. The other estimators use variants of this algorithm, and are detailed below. We recommend section [1] for more details and comparisons between these algorithms. In [1], **`PLSCanonical`** corresponds to "PLSW2A".

Given two centered matrices $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^{n \times t}$, and a number of components K, **`PLSCanonical`** proceeds as follows:

Set X_1 to X and Y_1 to Y. Then, for each $k \in [1, K]$:

- a) compute $u_k \in \mathbb{R}^d$ and $v_k \in \mathbb{R}^t$, the first left and right singular vectors of the cross-covariance matrix $C = X_k^T Y_k$. u_k and v_k are called the *weights*. By definition, u_k and v_k are chosen so that they maximize the covariance between the projected X_k and the projected target, that is $\text{Cov}(X_k u_k, Y_k v_k)$.
- b) Project X_k and Y_k on the singular vectors to obtain *scores*: $\xi_k = X_k u_k$ and $\omega_k = Y_k v_k$
- c) Regress X_k on ξ_k, i.e. find a vector $\gamma_k \in \mathbb{R}^d$ such that the rank-1 matrix $\xi_k \gamma_k^T$ is as close as possible to X_k. Do the same on Y_k with ω_k to obtain δ_k. The vectors γ_k and δ_k are called the *loadings*.

Toggle Menu

- d) *deflate* X_k and Y_k, i.e. subtract the rank-1 approximations: $X_{k+1} = X_k - \xi_k \gamma_k^T$, and $Y_{k+1} = Y_k - \omega_k \delta_k^T$.

At the end, we have approximated X as a sum of rank-1 matrices: $X = \Xi \Gamma^T$ where $\Xi \in \mathbb{R}^{n \times K}$ contains the scores in its columns, and $\Gamma^T \in \mathbb{R}^{K \times d}$ contains the loadings in its rows. Similarly for Y, we have $Y = \Omega \Delta^T$.

Note that the scores matrices Ξ and Ω correspond to the projections of the training data X and Y, respectively.

Step *a)* may be performed in two ways: either by computing the whole SVD of C and only retain the singular vectors with the biggest singular values, or by directly computing the singular vectors using the power method (cf section 11.3 in [1]), which corresponds to the `'nipals'` option of the `algorithm` parameter.

1.8.1.1. Transforming data

To transform X into $\bar{X}$, we need to find a projection matrix P such that $\bar{X} = XP$. We know that for the training data, $\Xi = XP$, and $X = \Xi \Gamma^T$. Setting $P = U(\Gamma^T U)^{-1}$ where U is the matrix with the u_k in the columns, we have $XP = XU(\Gamma^T U)^{-1} = \Xi(\Gamma^T U)(\Gamma^T U)^{-1} = \Xi$ as desired. The rotation matrix P can be accessed from the `x_rotations_` attribute.

Similarly, Y can be transformed using the rotation matrix $V(\Delta^T V)^{-1}$, accessed via the `y_rotations_` attribute.

1.8.1.2. Predicting the targets Y

To predict the targets of some data X, we are looking for a coefficient matrix $\beta \in R^{d \times t}$ such that $Y = X\beta$.

The idea is to try to predict the transformed targets Ω as a function of the transformed samples Ξ, by computing $\alpha \in \mathbb{R}$ such that $\Omega = \alpha \Xi$.

Then, we have $Y = \Omega \Delta^T = \alpha \Xi \Delta^T$, and since Ξ is the transformed training data we have that $Y = X\alpha P \Delta^T$, and as a result the coefficient matrix $\beta = \alpha P \Delta^T$.

β can be accessed through the `coef_` attribute.

1.8.2. PLSSVD

PLSSVD is a simplified version of **PLSCanonical** described earlier: instead of iteratively deflating the matrices X_k and Y_k, **PLSSVD** computes the SVD of $C = X^T Y$ only *once*, and stores the `n_components` singular vectors corresponding to the biggest singular values in the matrices `U` and `V`, corresponding to the `x_weights_` and `y_weights_` attributes. Here, the transformed data is simply `transformed(X) = XU` and `transformed(Y) = YV`.

If `n_components == 1`, **PLSSVD** and **PLSCanonical** are strictly equivalent.

1.8.3. PLSRegression

The **PLSRegression** estimator is similar to **PLSCanonical** with `algorithm='nipals'`, with 2 significant differences:

- at step a) in the power method to compute u_k and v_k, v_k is never normalized.
- at step c), the targets Y_k are approximated using the projection of X_k (i.e. ξ_k) instead of the projection of Y_k (i.e. ω_k). In other words, the loadings computation is different. As a result, the deflation in step d) will also be affected.

These two modifications affect the output of `predict` and `transform`, which are not the same as for **PLSCanonical**. Also, while the number of components is limited by `min(n_samples, n_features, n_targets)` in **PLSCanonical**, here the limit is the rank of $X^T X$, i.e. `min(n_samples, n_features)`.

PLSRegression is also known as PLS1 (single targets) and PLS2 (multiple targets). Much like **Lasso**, **PLSRegression** is a form of regularized linear regression where the number of components controls the strength of the regularization.

1.8.4. Canonical Correlation Analysis

Canonical Correlation Analysis was developed prior and independently to PLS. But it turns out that **CCA** is a special case of PLS, and corresponds to PLS in “Mode B” in the literature.

CCA differs from **PLSCanonical** in the way the weights u_k and v_k are computed in the power method of step a). Details can be found in section 10 of [1].

Since **CCA** involves the inversion of $X_k^T X_k$ and $Y_k^T Y_k$, this estimator can be unstable if the number of features or targets is greater than the number of samples.

Reference:

[1] (1,2,3,4)
A survey of Partial Least Squares (PLS) methods, with emphasis on the two-block case JA Wegelin

Examples:

- Compare cross decomposition methods
- Principal Component Regression vs Partial Least Squares Regression