

Theory

Harpeth Lee

3/15/2022

Theory

Principal Component Analysis

Due to the similarity between partial least squares and principal component regression, we will briefly review principal component analysis.

Principal component analysis(PCA) is a commonly used unsupervised learning technique. PCA works by finding the principle components of a dataset, these can be thought of as the direction along which the data varies the most in the feature space. For those of you with a background in linear algebra, the principal components will be the eigenvectors of the covariance matrix in order of the norm of the eigenvalues.

PCA is one of the most commonly used dimension reduction techniques as it is often able to capture a large amount of the variation in a data set using only a few features. PCA returns a series of principal components, these can be thought of as vectors in the original feature space. Principal components will be orthogonal to each other so the variance explained by each component will be unique. Given an $n \times p$ data set, n principal components can be created where each principal component is a vector of length p .

The algorithm for finding the first principal component of a standardized $n \times p$ data matrix \mathbf{X} is as follows:

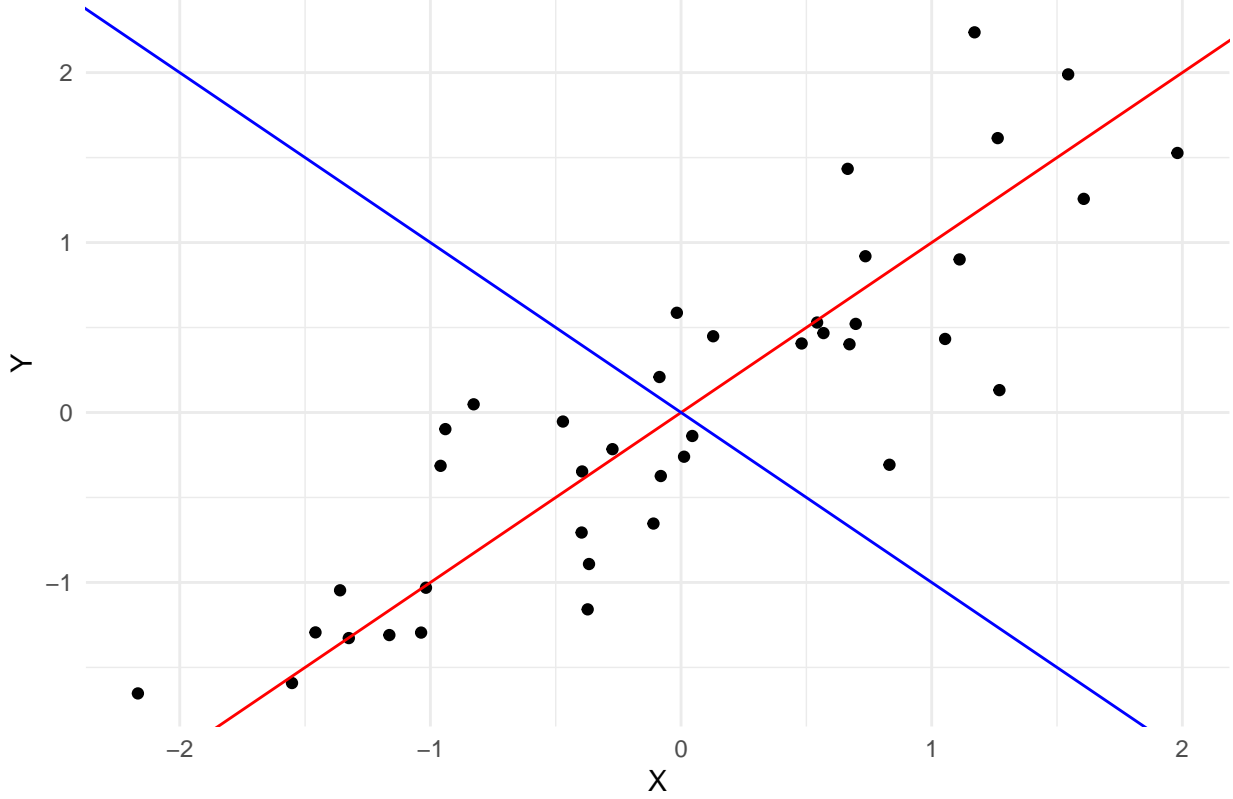
$$\mathbf{w}_1 = \operatorname{argmax}_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} \right\}$$

Note that $\mathbf{X}^T \mathbf{X}$ is the covariance matrix for \mathbf{X} . For all following components, we will find the matrix $\hat{\mathbf{X}}_k$ such that all variance explained by the first $k - 1$ principal components is removed. This is done by finding

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{j=1}^{k-1} \mathbf{X} \mathbf{w}_j \mathbf{w}_j^T$$

We then use the same equation used to find \mathbf{w}_1 to find the k th principle component, replacing \mathbf{X} with $\hat{\mathbf{X}}_k$.

The following plot shows the first two(and only) principals components for a set of two dimensional data. The red line is the first principal component while the green line is the second. Note that they are perpendicular. The red line describes as much variation in the data using a singular vector in the original space. Although it may look similar, the first principal component is not the same as the regression line which minimizes the residual distance.



The red line shows the first principal component of the data while the blue line shows the second principal component

Often principal components can be used as latent variables. Sometimes they may be meaningful and correspond to a property of the data that isn't directly measured. For example, the first principal component of data containing the nutritional values for food may correspond to how rich in vitamins a food is.

Principal Component Regression (PCR) is a regression technique that uses principal components as predictors. To perform PCR, one first finds the desired number of principal components and then performs linear regression using the selected principal components as predictors. Letting $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k$ be the first k principal components of our predictors \mathbf{X} . This will take the form

$$\mathbf{y} = \theta_0 + \sum_{i=1}^k \theta_i \mathbf{w}_i$$

L1 Sparsification check equation

The L_1 penalty can be added to a model by adding a term of $\lambda \sum_{i=1}^p |\beta_i|$ where λ is a parameter for the model. For linear regression, coefficients for the model will then be found by finding $\{\beta_0, \dots, \beta_n\}$ such that the following is minimized

$$\operatorname{argmin} \left\{ \frac{1}{2} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}$$

Note that as $\lambda \rightarrow 0$ the model will become unchanged by the penalty and as $\lambda \rightarrow \infty$, $\beta_i = 0$ for all i . If λ is sufficiently large, we will begin to see some $\beta_i = 0$ which makes the L_1 norm special among commonly used norms.

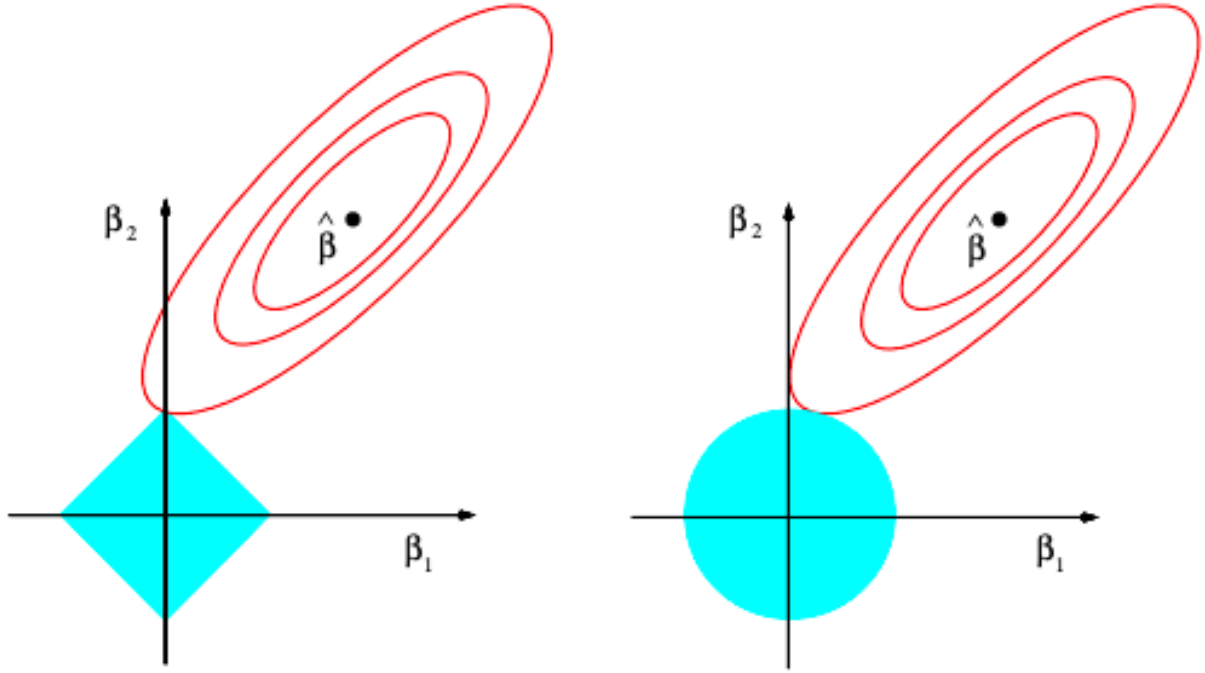


Figure 1: Comparison of L_1 (left) and L_2 (right) norms. As we can see restraints on the L_2 norms prevent coefficients from being sent to 0. Image taken from ESLR.

Latent Variable Models

Latent variable models such as PCR and PLS are built on the assumption that there exists a structure linking the predictors and response variables that isn't directly captured in the data. There are two ways we can try and conceptualize these latent variables, either as being derived from the observed data or being used to generate the observed data. In this case, it is most helpful to think of latent variables as generating the observed data.

We will assume that $\mathbf{X} = \phi\mathbf{A} + \epsilon_X$ and $\mathbf{Y} = \phi\mathbf{D} + \epsilon_Y$. Here ϕ is a matrix of latent variables linking \mathbf{X} and \mathbf{Y} . \mathbf{A} and \mathbf{D} are the transformation applied to ϕ to generate \mathbf{X} and \mathbf{Y} . ϵ_X and ϵ_Y are random noise.

$\mathbf{X} \in \mathbb{R}_{n \times p}$ and $\mathbf{Y} \in \mathbb{R}_{n \times q}$. Thus, $\phi \in \mathbb{R}_{n \times \mathcal{K}}$, $\mathbf{A} \in \mathbb{R}_{\mathcal{K} \times p}$, and $\mathbf{D} \in \mathbb{R}_{\mathcal{K} \times q}$. Note that $\mathcal{K} \neq K$ where K is the number of components that will be used in a PLS model.

Since all possible variance is explained by ϕ , \mathbf{A} , and \mathbf{D} we will find that $\text{Cov}(\phi, \epsilon_X) = \text{Cov}(\phi, \epsilon_Y) = \text{Cov}(\epsilon_X, \epsilon_Y) = 0$. Furthermore, $\text{Var}(\mathbf{X}) = \mathbf{A}\mathbf{A}^T + \text{Var}(\epsilon_X)$ and $\text{Var}(\mathbf{Y}) = \mathbf{D}\mathbf{D}^T + \text{Var}(\epsilon_Y)$. Finally, $\text{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbf{D}^T\mathbf{A}$.

While it may seem intuitive to build a model of the form $\mathbf{Y} = \mathbf{B}\mathbf{X} + \epsilon$ where \mathbf{B} satisfies $\mathbf{A}\mathbf{B} = \mathbf{D}$. This only provides a unique answer under certain circumstances. Instead we want to formulate a model that is more similar to how the data is structured. Thus, our model will be of the form $\mathbf{X} = \mathbf{tP} + \epsilon_X$ and $\mathbf{Y} = \mathbf{tC} + \epsilon_Y$.

Evaluating Model Performance

While building models is a difficult task selecting the correct model to use may be even more difficult. We want to find a model that fits our original data well without overfitting it. Unlike when building a model, there is usually more than one criteria that we want our model to perform well with. Ultimately, model selection often ends up being a subjective process dependent on a variety of factors. While there are standard measures designed to evaluate the performance of a model, I advise against putting too much faith in any singular statistic.

One of the most widely used statistics to evaluate model performance is the root mean squared error(RMSE). The RMSE is calculated using the formula

$$RMSE = \sqrt{\frac{\sum_{j=1}^q \sum_{i=1}^n (\hat{y}_{i,j} - y_{i,j})^2}{n}}$$

where $\hat{y}_{i,j}$ is the predicted value of a response variable. Here we want a low RMSE as this means that the model tends to estimate values close to the true value.

Another popular statistic used is R^2 . Simply, $R^2 = 1 - \frac{RSS}{TSS}$ where RSS is the real sum of squares and TSS is the total sum of squares. The RSS is calculated using the equation $RSS = \sum_{j=1}^q \sum_{i=1}^n (\hat{y}_{i,j} - y_{i,j})^2$ while the TSS is calculated using $TSS = \sum_{j=1}^q \sum_{i=1}^n (\bar{y}_{i,j} - y_{i,j})^2$ where $\bar{y}_{i,j}$ is the empirical mean of the observed data. The idea of R^2 is to evaluate how a model performs compared to using mean estimation. Mean estimation is perhaps the most crude form of regression available, if our model does not perform well when using the R^2 metric, it is almost always a sign that we should not use the model. Values of R^2 usually range between 0 and 1. Values less than 0 mean that the model performs worse than mean estimation. Values equal to 0 mean the two perform the same (usually this mean the model is just performing mean estimation). Values between 0 and 1 mean that the model performs better than mean estimation with larger values signifying better performance compared to mean estimation. If $R^2 = 1$ this means that the model is able to perfectly estimate every response value in the data.

Before moving onto the next metric now is a good time to discuss test-training splits, cross validation(sometimes notated as CV), and bootstrapping.. All of these methods are used to try and answer questions about how the model will perform on data that is not used to train the model. Common to all of these methods is the idea of in-bag(IB) and out-of-bag(OOB) observations. IB observations are the data that are used to build our model while OOB observations are the data that the model is not built with that can be used to test it. All of the following methods are based on some permutation of this idea of creating IB and OOB groupings of the data.

Test-training splits are probably the most basic of the three methods relying on splitting our data into two groups, the training group and the test group. Here the IB group will be the training data and the OOB group will be the test data, these groups will not change during the process. To create the test and training sets we randomly split our data into two groups. Usually a smaller percentage of the data is relegated to the training set (e.g. 30% is assigned to the training set while the remaining 70% will be the test set). The model will then be built using the training data and assessed using the test data. This will help tend to help with overfitting as it will make sure the model can perform well on data that isn't used to build it. This still makes the assumption that our larger sample mirrors the general population that we hope to use the model with.

Cross validation is one of the most common tools for model assessment and parameter tuning. Cross validation involves splitting the data into test and training sets multiple times and then assessing model performance in each instant. It can be helpful to think of cross validation as a repeated use of test-training splits. There are two commonly used types of cross validation, K -fold CV and Leave-one-out CV(LOOCV). K -fold CV randomly divides the data into K equal sized groups and then uses $K - 1$ of the groups as the training set and the remaining one as the test set. This is repeated K times until all groups have been used as the test set. LOOCV is technically a form of K -fold where K is equal to the total number of observations. Thus, the test set will only consist of 1 observation.

CV is often used to select parameters, as we will build models with a number of parameters and then assess its performance across parameters using cross validation. The choice of statistic used to assess parameter performance often varies depending on the model we are building although R^2 is a common choice. One of the problems with this approach is that it is computationally expensive as it requires the model to be built across a range of parameters. Due to this, when working with a large number of observations, it is generally advisable to use K -fold CV with a smaller number of folds (10 is often recommended) as this will require many fewer computations the LOOCV.

Bootstrapping relies on randomly sampling with replacement from our data to build the test and training sets. When bootstrapping we will expect to see repeated values in the training set (no matter the size of our

dataset we expect to see 63.2% in our bootstrap sample). The observations not included in the bootstrap sample will then be the test set.

Following this discussion of IB and OOB observations and methods of generating them we will discuss the Q^2 statistic. Q^2 is very similar to the R^2 statistic with the only difference being that it is exclusively calculated using OOB observations. The Q^2 statistic is only commonly used when discussing PLS models. There is often some ambiguity as to what data is used to calculate the R^2 statistic. It can sometimes be calculated on the training set to help build the model or on the test set to evaluate model performance. Differentiating R^2 and Q^2 solves this ambiguity. R^2 for a test-training split b is found with the equation

$$R_b^2 = 1 - \frac{\sum_{j=1}^q \sum_{i \in IB(b)} (\hat{y}_{i,j}^b - y_{i,j})^2}{\sum_{j=1}^q \sum_{i \in IB(b)} (\bar{y}_{i,j}^b - y_{i,j})^2}$$

Q^2 for the split b is calculated as follows

$$Q_b^2 = 1 - \frac{\sum_{j=1}^q \sum_{i \in OOB(b)} (\hat{y}_{i,j}^b - y_{i,j})^2}{\sum_{j=1}^q \sum_{i \in OOB(b)} (\bar{y}_{i,j}^b - y_{i,j})^2}$$

where $\hat{y}_{i,j}^b$ are predictions from the model built using the training group b and $\bar{y}_{i,j}^b$ are the mean estimates for rows of the training group b . We will then take the means across a B different splits so that $\bar{R}_B^2 = \frac{1}{B} \sum_{b=1}^B R_b^2$ and $\bar{Q}_B^2 = \frac{1}{B} \sum_{b=1}^B Q_b^2$.

Despite the similarity of the R^2 and Q^2 metrics, the two metrics will improve under different conditions for the model. Maximizing R^2 will favor more complex models that closely fit the training data while maximizing Q^2 will favor less complex models that address the underlying structure of the data. Initially, increasing model complexity will improve both metrics. However, past a certain threshold, increasing model complexity will cause Q^2 to decrease while R^2 continues to increase. Due to this, minimizing $R^2 - Q^2$ is recommended for selecting values of λ_k for the ddsPLS model.

Ideally we want a high R^2 and Q^2 meaning the model performs well with both the test and training data. In practice R^2 is almost always an inflated metric of performance and Q^2 will be more accurate for assessing model performance. For this reason minimizing $R^2 - Q^2$ may be used for parameter selection as low values indicate models that perform well on both IB and OOB observations. Although it may seem likely to pick parameters where both R^2 and Q^2 perform poorly, R^2 tends to increase more quickly and fall off more slowly than Q^2 leaving a difference at these extremes.

Partial Least Squares

Partial Least Squares (PLS) is a regression technique similar to PCR. Both techniques work by finding latent variables in the original data with which to perform linear regression. PLS is considered to be one of the best tools for modeling the underlying structure between \mathbf{X} and \mathbf{Y} in feature space. As suggested by its creator, the name "Projection onto Latent Structures" is a more descriptive acronym as the original data is projected onto latent structures which are then used for linear regression.

Unlike PCR, PLS creates latent variables using the covariance matrix instead of the variance matrix. Implicit in PCR is the assumption that direction of high variance among the predictors explains much of the variance among the response variables. While this assumption is usually reasonable, PLS circumvents this assumption by directly building the latent variables along the direction in feature space along where the responses vary the most along the predictors.

PLS is designed to model situations where there are a high number of predictors in relation to the number of observations ($n \approx p/n < p$). In these situations, most models will struggle to differentiate noise from signal and will be highly variable depending on the data used to train them. PLS circumvents this problem by simplifying the data into a smaller number of latent variables that still strongly reflect the original data set. Unlike models that depend solely on variable selection, PLS will not discard relevant information from all observed variables in order to fit a model.

PLS is also designed to perform well in situations where predictors are highly correlated and noisy. The data is projected onto underlying structures in order to handle correlation between predictors. Correlated predictors can often be better explained along a singular vector in feature space that captures all correlated predictors. This correlation will be captured in the latent structures that the data is project onto. These underlying structures captured in the latent variables will also be less sensitive to noise among the predictors.

As previously mentioned, PLS models often follow a five step process outline by Lorenzo. To briefly review, these steps are a) estimate the covariance matrix, b) find the space associated with the first eigenvector, c) project the covariate and response matrices into the space from the previous step, d) perform linear regression between the projection of the covariate and response matrices, and e) remove the predicted information.

Like PCA, PLS also require all variables to be standardized with $\mu = 0$ and $\sigma^2 = 1$. We then find the m th latent variable \mathbf{z}_m using the following formula

$$\mathbf{z}_m = \sum_{i=1}^p \hat{\varphi}_{mi} \mathbf{x}_j^{(m-1)}$$

where $\hat{\varphi}_{mi} = \langle \mathbf{x}_j^{(m-1)}, \mathbf{y} \rangle$.

We then find the regression coefficient $\hat{\theta}_m$ for \mathbf{z}_m as follows

$$\hat{\theta}_m = \frac{\langle \mathbf{z}_m, \mathbf{y} \rangle}{\langle \mathbf{z}_m, \mathbf{z}_m \rangle}$$

Note that this is identical to the formula used in classic linear regression and PCR at this step.

NIPALS-PLS The NIPALS-PLS algorithm (sometimes referred to as the PLS2 algorithm) is the first PLS model suggested by Wold and still one of the more commonly used PLS models. Similar basic PLS models almost exclusively differ in the method used to calculate the covariance matrix, especially when dealing with missing data.

Before going into detail on the model it is important to note that all PLS algorithms require the data to be centered and standardized in order to prevent variables with different scales from exhibiting an outsize impact on the final outcome.

We will assume that $\mathbf{M} = \frac{1}{n} \sum_{i=1}^n (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$ will be used to estimate the covariance matrix unless otherwise noted. This estimate, \mathbf{M} is the empirical covariance matrix calculated using the observed data. Estimating \mathbf{M} is the most sensitive step as this estimation is sensitive to the curse of dimensionality.

The NIPALS-PLS algorithm is as follows:

For $k \in [1 : K]$,

a) $\mathbf{u}_k = \overrightarrow{\text{RSV}}(\mathbf{M}_k)$

b) $\mathbf{t}_k = \mathbf{X}_k \mathbf{u}_k$

c) $\mathbf{p}_k = \frac{\mathbf{X}_k^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

d) $\mathbf{c}_k = \frac{\mathbf{Y}_k^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k \mathbf{p}_k^T, \mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k \mathbf{c}_k^T$

Walking through these steps; a) finds the vector that explains the most variance in the covariance matrix for the remaining data, b) estimates the score, c) and d) estimate the linear regression matrices for \mathbf{X} and \mathbf{Y} , and e) deflates the data.

The NIPALS-PLS algorithm maximizes $\mathbf{v}^T \mathbf{M}_k \mathbf{u}$ with the constraint $\mathbf{v}^T \mathbf{v} = \mathbf{u} \mathbf{u}^T = 1$. The only parameter that needs tuning is K , the number of components used in the model.

sPLS Here we will discuss a sparse PLS algorithm outlined by Le Cao et.al. It is important to note that there exist other sparse PLS algorithms most notably the one created by Chun and Keles.

First we must define the soft-thresholding function $S_\lambda(\mathbf{M}) = \operatorname{argmin} (||\mathbf{M} - \boldsymbol{\Sigma}||^2 + 2\lambda|\boldsymbol{\Sigma}|)$ with $\boldsymbol{\Sigma} \in \mathbb{R}^{q \times p}$ and where \mathbf{M} is again our estimate of the covariance matrix.

The sPLS algorithm is as follows:

For $k \in [1 : K]$,

a[†]) Start with initial \mathbf{v}_k and iterate until the two steps converge.

i[†]) $\mathbf{u}_k = S_{\lambda_u^k}(\mathbf{M}_k^T \mathbf{v}_k)$, $\mathbf{u}_k = \frac{\mathbf{u}_k}{||\mathbf{u}_k||}$

ii[†]) $\mathbf{v}_k = S_{\lambda_v^k}(\mathbf{M}_k^T \mathbf{u}_k)$, $\mathbf{v}_k = \frac{\mathbf{v}_k}{||\mathbf{v}_k||}$

b) $\mathbf{t}_k = \mathbf{X}_k \mathbf{u}_k$

c) $\mathbf{p}_k = \frac{\mathbf{X}_k^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

d) $\mathbf{c}_k = \frac{\mathbf{Y}_k^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k \mathbf{p}_k^T$, $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k \mathbf{c}_k^T$

Notice that step a) is the only step that differs from the NIPALS-PLS algorithm. This is where sparsity is imposed upon \mathbf{u} and \mathbf{v} which are used to calculate the score and loadings. Note that the sparsity is only directly imposed on \mathbf{X} .

The algorithm minimizes $|(n-1)\mathbf{M}_k - \mathbf{v}\mathbf{u}^T|^2 + \lambda_u^k |\mathbf{u}| + \lambda_v^k |\mathbf{v}|$ for \mathbf{u} and \mathbf{v} with the constraint $\mathbf{v}^T \mathbf{v} = \mathbf{u}^T \mathbf{u} = 1$. sPLS has $2K + 1$ parameters with two tuning parameters for each component and then the number of components.

Data-driven Sparse Partial Least Squares Data-driven sparse partial least squares(ddsPLS) is a new PLS method recently proposed by Lorenzo. Unlike previous sparse partial least squares algorithms, ddsPLS directly imposes sparsity on the empirical covariance matrix instead of its eigenvector decomposition.

For $k \in [1 : K]$,

a) $\mathbf{u}_k = \overrightarrow{RSV}(S_{\lambda_k}(M_k))$, $\mathbf{v}_r = \overrightarrow{RSV}(S_{\lambda^{(r)}}(M^{(r)T}))$

b) $\mathbf{t}_k = \mathbf{X}_k \mathbf{u}_k$

c) $\mathbf{p}_k = \frac{\mathbf{X}_k^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

d) $\Pi_k = \operatorname{diag}(\{\delta_{\neq 0}(\mathbf{v}_k)_j\}_{j \in [1:q]})$ $\mathbf{c}_k = \operatorname{argmin} ||\mathbf{Y}_k \Pi_r - \mathbf{t}_k \mathbf{V}^T||^2 = \frac{(\mathbf{Y}_k \Pi_r)^T \mathbf{t}_k}{\mathbf{t}_k^T \mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k \mathbf{p}_k^T$ $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k \mathbf{c}_k^T$

Note that if $\lambda = 0$ the outcome is identical to the NIPALS-PLS algorithm.

ddsPLS maximizes $\mathbf{v}^T S_{\lambda_k}(\mathbf{M}_k^T) \mathbf{u}$ for \mathbf{u} and \mathbf{v} with the constraint $\mathbf{v}^T \mathbf{v} = \mathbf{u}^T \mathbf{u} = 1$. The ddsPLS algorithm has $K + 1$ parameters as there is only one parameter that needs to be maximized for each component. This gives it a better computation time when compared to other sparse PLS algorithms as cross validation or bootstrapping only needs to be performed for one parameter per component.

Along with the ddsPLS algorithm, Lorenzo also proposes a standard method for selecting the number of components to include in the model. If the following 3 conditions are met, the k th component is built. If the conditions are not built, the model will include only $k - 1$ components. The conditions are as follows,

1. The k th component performs better than mean estimation, $\bar{Q}_{B,k}^2 > 0$.

2. The model containing K components performs than the model containing $K - 1$ components, $\bar{Q}_{B,K}^2 > \bar{Q}_{B,K-1}^2$.
3. The bootstrapped explained variance is close to the boot-strapped cross-validated explained variance, $\bar{R}_B^2 - \bar{Q}_B^2$ is minimum on the set of hyperparameters.