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:

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

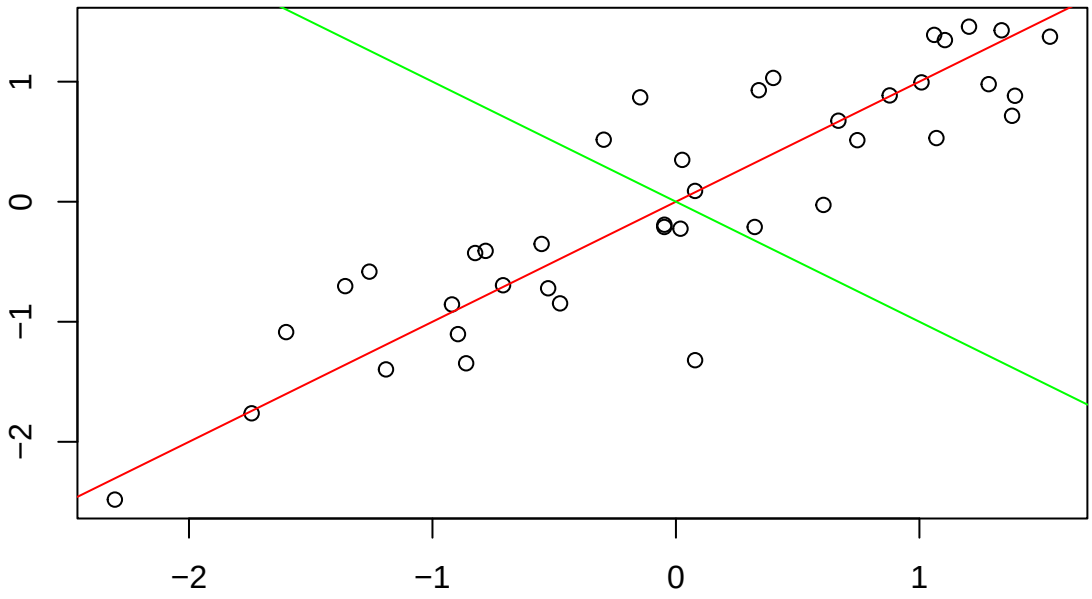
Note that $\mathbf{X}^T\mathbf{X}$ is the covariance matrix for $\mathbf{X}$. For all following components, we will find the matrix $\hat{\mathbf{X}}_k$ such that all variance explained by the first $k-1$ principal components is removed. This is done by finding

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

We then use the same equation used to find $\mathbf{w}_1$ to find the kth 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.

```
## Warning: package 'MASS' was built under R version 4.1.2
```



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 $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_k$ be the first k principal components of our predictors $\mathbf{X}$. This will take the form

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

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, $\text{Var}(\mathbf{X}) = \mathbf{A}\mathbf{A}^{\text{T}} + \text{Var}(\epsilon_X)$ and $\text{Var}(\mathbf{Y}) = \mathbf{D}\mathbf{D}^{\text{T}} + \text{Var}(\epsilon_Y)$. Finally, $\text{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbf{D}^{\text{T}}\mathbf{A}$.

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 $\mathbf{X} = \mathbf{t}\mathbf{P} + \epsilon_X$ and $\mathbf{Y} = \mathbf{t}\mathbf{C} + \epsilon_Y$.

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 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 situation, 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 mth 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 $\mathbf{M} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}$ 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 all $k \in [1 : K]$,

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

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

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

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

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

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 $\mathbf{v}^{\mathrm{T}}\mathbf{M}_k\mathbf{u}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. 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 $S_\lambda(\mathbf{M}) = \text{argmin}\left(||\mathbf{M} - \mathbf{\Sigma}||^2 + 2\lambda|\mathbf{\Sigma}|\right)$ with $\mathbf{\Sigma} \in \mathbb{R}^{q\times p}$ and where $\mathbf{M}$ is again our estimate of the covariance matrix.

The sPLS algorithm is as follows:

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

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

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

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

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

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

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

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

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 $||(n-1)\mathbf{M}_k - \mathbf{v}\mathbf{u}^{\mathrm{T}}||^2 + \lambda_u^k|\mathbf{u}| + \lambda_v^k|\mathbf{v}|$ for $\mathbf{u}$ and $\mathbf{v}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. 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 sparisty on the empirical covariance matrix instead of its eigenvector decomposition.

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

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

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

ddsPLS maximizes $\mathbf{v}^{\mathrm{T}} S_{\lambda_k}(\mathbf{M}_k^{\mathrm{T}})\mathbf{u}$ for $\mathbf{u}$ and $\mathbf{v}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. 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.