Applied Statistical Methods

Linear Model Regularization - Part II

Xuemao Zhang
East Stroudsburg University

April 3, 2023

Outline

Principal Components Analysis

- The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set.
- This is achieved by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.
- Suppose that $\mathbf{x} = (x_1, x_2, \ldots, x_p)'$ is a vector of p random variables, and that the variances of the p random variables and the structure of the covariances/correlations between the p variables are of interest.

Principal Components Analysis

- Instead of looking at the p variances and $\frac{1}{2}p(p-1)$ covariances, we look for a few ($< p$) derived variables that preserve most of the information given by these variances and correlations/covariances.
    - PCA concentrates on variances.
- The first step is to look for a linear combination of the elements of $\mathbf{x}$ having maximum variance

$$\boldsymbol{\alpha}_1^{'}\mathbf{x} = \alpha_{11}x_1 + \cdots + \alpha_{1p}x_p = \sum_{j=1}^{p} \alpha_{1j}x_j.$$

- Next, look for a linear combination $\boldsymbol{\alpha}_2^{'}\mathbf{x}$, uncorrelated with $\boldsymbol{\alpha}_1^{'}\mathbf{x}$ having maximum variance, and so on. ... Up to p PCs could be found.
    - It is hoped, in general, that most of the variation in $\mathbf{x}$ will be accounted for by m PCs, where $m << p$.

Principal Components Analysis

- Derivation of the form of the PCs: note that

$$var(\boldsymbol{\alpha}_1^{'}\mathbf{x}) = \boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_1,$$

  where $\boldsymbol{\Sigma} = var(\mathbf{x})$. So the maximization must be subject to a normalization constraint

$$\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_1 = \sum_{j=1}^{p} \alpha_{1j}^2 = 1.$$

  Using the technique of Lagrange multipliers (Calculus III, λ_1 is called the Lagrange Multiplier), We maximize the function

$$\boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 - \lambda_1(\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_1 - 1)$$

  w.r.t. $\boldsymbol{\alpha}_1$ by differentiating w.r.t. to $\boldsymbol{\alpha}_1$.

Principal Components Analysis

This results in

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\alpha}_1}[\boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 - \lambda_1(\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_1 - 1)] = 0 \\ \boldsymbol{\Sigma}\boldsymbol{\alpha}_1 - \lambda_1\boldsymbol{\alpha}_1 = 0\end{aligned}$$

and thus

$$\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 = \lambda_1\boldsymbol{\alpha}_1.$$

- This should be recognizable as an eigenvector equation where $\boldsymbol{\alpha}_1$ is an eigenvector of $\boldsymbol{\Sigma}$ and λ_1 is the associated eigenvalue.

Principal Components Analysis

- Which eigenvector should we choose?

$$\boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_1^{'}\lambda_1\boldsymbol{\alpha}_1 = \lambda_1$$

- Then we should choose λ_1 to be as big as possible. So λ_1 is the largest eigenvalue of $\boldsymbol{\Sigma}$ and $\boldsymbol{\alpha}_1$ is the corresponding eigenvector.
- Then the solution to

$$\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 = \lambda_1\boldsymbol{\alpha}_1$$

  is the 1st PC(principal component) of **x**.

Principal Components Analysis

- The second PC, $\boldsymbol{\alpha}_2^{'}\mathbf{x}$ maximizes $\boldsymbol{\alpha}_2^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_2$ subject to

  $$\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_2 = 1$$

  and being uncorrelated with $\boldsymbol{\alpha}_1^{'}\mathbf{x}$:

  $$cov(\boldsymbol{\alpha}_1^{'}\mathbf{x}, \boldsymbol{\alpha}_2^{'}\mathbf{x}) = \boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_2^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_1 = \lambda_1\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_1 = \lambda_1\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_2 = 0.$$

- Using the technique of Lagrange multipliers with these two constraints, we maximize the function w.r.t $\boldsymbol{\alpha}_2$

  $$\boldsymbol{\alpha}_2^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 - \lambda_2(\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_2 - 1) - \phi\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_1$$

Principal Components Analysis

- Differentiation of this quantity w.r.t. $\boldsymbol{\alpha}_2$ (and setting the result equal to zero) yields

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\alpha}_2}[\boldsymbol{\alpha}_2^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 - \lambda_2(\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_2 - 1) - \phi\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_1] = 0 \\ \boldsymbol{\Sigma}\boldsymbol{\alpha}_2 - \lambda_2\boldsymbol{\alpha}_2 - \phi\boldsymbol{\alpha}_1 = 0 \end{aligned}$$

- If we left multiply $\boldsymbol{\alpha}_1$ into this expression

$$\begin{aligned} \boldsymbol{\alpha}_1^{'}\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 - \lambda_2\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_2 - \phi\boldsymbol{\alpha}_1^{'}\boldsymbol{\alpha}_1 = 0 \\ 0 - 0 - \phi = 0 \end{aligned}$$

then we can see that ϕ must be zero.

Principal Components Analysis

- So we have
$$\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 = \lambda_2\boldsymbol{\alpha}_2.$$
Furthermore,
$$var(\boldsymbol{\alpha}_2^{'}\mathbf{x}) = \boldsymbol{\alpha}_2^{'}\lambda_2\boldsymbol{\alpha}_2 = \lambda_2\boldsymbol{\alpha}_2^{'}\boldsymbol{\alpha}_2 = \lambda_2.$$

Principal Components Analysis

- Thus, the second PC $\boldsymbol{\alpha}_2'\mathbf{x}$ is the solution to

$$\boldsymbol{\Sigma}\boldsymbol{\alpha}_2 = \lambda_2\boldsymbol{\alpha}_2,$$

  where λ_2 is the second largest eigenvalue of $\boldsymbol{\Sigma}$.

- This process can be repeated for $k = 1, 2, \ldots, p$ yielding up to p different eigenvectors of $\boldsymbol{\Sigma}$ along with the corresponding eigenvalues $\lambda_1, \ldots, \lambda_p$.

Principal Components Regression

- Given data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_p)$ with each $\mathbf{x}_i, i = 1, \ldots, p$ a column vector, we can estimate $\mathbf{\Sigma}$ by the sample covariance matrix

$$\mathbf{S} = \mathbf{X}^{'}\mathbf{X}/(n-1).$$

- PCA produces a low-dimensional representation of a dataset. It finds a sequence of linear combinations of the variables that have maximal variance, and are mutually uncorrelated.
- Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization.

Principal Components Regression

- Our data consist of n observations in a p-dimensional space.
- However, not all of those p dimensions are equally useful, especially when $p >> n$.
- Many are either completely redundant (correlated features) or uninformative (noise features).
- Can we find a low-dimensional representation of the variables that captures most of the variability in the data?
- We now explore a class of approaches that transform the predictors and then fit a least squares model using the transformed variables.
- This is a dimension reduction approach.

Principal Components Regression

- Let $Z_1, Z_2, \ldots, Z_M$ represent $M < p$ linear combinations of our original p predictors. That is,

$$Z_m = \sum_{j=1}^{p} \phi_{mj} X_j \tag{1}$$

- Use least squares to fit the model

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m Z_{im} + \varepsilon_i, i = 1, \ldots, n \tag{2}$$

- In other words, we perform least squares using M new predictors $Z_1, Z_2, \ldots, Z_M$.
- $Z_1, Z_2, \ldots, Z_M$ are chosen to be the **principal components** of the data.

Principal Components Regression

- Notice that from definition (1),

$$\sum_{m=1}^{M} \theta_m Z_{im} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{mj} x_{ij} = \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{mj} x_{ij} = \sum_{j=1}^{p} \beta_j x_{ij}$$

where

$$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{mj}. \tag{3}$$

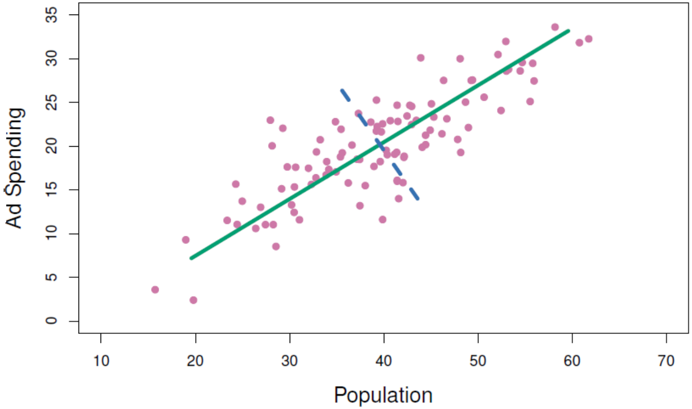
- Hence model (2) can be thought of as a special case of the original linear regression model.
- Dimension reduction serves to constrain the estimated β_j coefficients, since now they must take the form (3).
- Can win in the bias-variance tradeoff.

Principal Components Regression

- Here we apply principal components analysis (PCA) to define the linear combinations of the predictors, for use in our regression. By the theory of Principal Components,
    - The first principal component is that (normalized) linear combination of the variables with the largest variance.
    - The second principal component has largest variance, subject to being uncorrelated with the first.
    - And so on.
- Hence with many correlated original variables, we replace them with a small set of principal components that capture their joint variation.

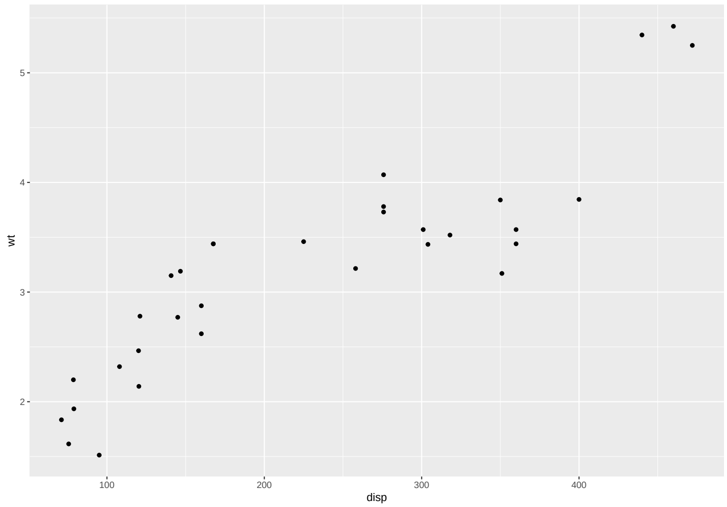
Principal Components Regression

- The population size (`pop`) and ad spending (`ad`) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component, and the blue dashed line indicates the second principal component.

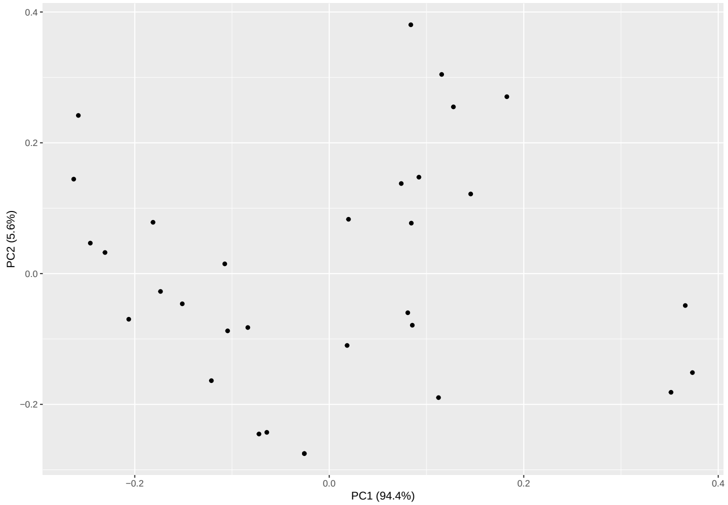


Principal Components Regression

- Consider the `mtcars` data.



Principal Components Regression



Principal Components Regression

- PCs are the linear combinations of the variables that contain as much as possible of the variability in the features.
- **PCR doesn't yield feature selection** - all of the original predictors are involved in the final model.
- But when M is small, then PCR can **avoid overfitting** and can give good results.
- Choose M by cross-validation or validation set approach.
  - Also consider the percentage of variance in X or Y by the predictors
- With $M = p$, we just get least squares regression: no dimension reduction occurs!

Principal Components Regression

- PCR directions are identified in an unsupervised way, since the response Y is not used to help determine the principal component directions.
- That is, the response does not supervise the identification of the principal components.
- Consequently, PCR suffers from a potentially serious drawback: there is **no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response**.

PCR with Python

- Scikit-klearn does not have an implementation of PCA and regression combined like the 'pls' package in R. See https://cran.r-project.org/web/packages/pls/vignettes/pls-manual.pdf

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import scale
from sklearn.model_selection import KFold, cross_val_score

Hitters=pd.read_csv("../data/Hitters.csv", header=0, na_values='NA')
Hitters = Hitters.dropna()
dummies = pd.get_dummies(Hitters[['League', 'Division', 'NewLeague']])
y = Hitters['Salary']
X_ = Hitters.drop(['Salary', 'League', 'Division', 'NewLeague'],
axis=1).astype('float64')
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
```

PCR with Python

- `scale`: scale predictor variables: This tells Python that each of the predictor variables should be scaled to have a mean of 0 and a standard deviation of 1. This ensures that no predictor variable is overly influential in the model if it happens to be measured in different units.

```
pca = PCA()
regr = LinearRegression()
mse = []

X_reduced = pca.fit_transform(scale(X))

kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)
```

PCR with Python

- Calculate MSE with only the intercept

```
import numpy as np
score = -1*cross_val_score(regr,np.ones((len(X_reduced),1)), y,
cv=kf_10, scoring='neg_mean_squared_error').mean()
mse.append(score)
```

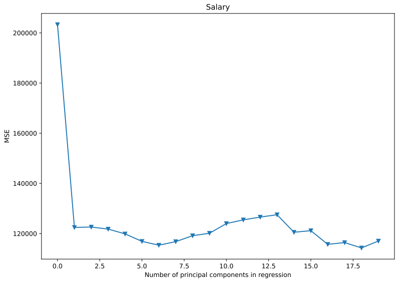
- Calculate MSE using cross-validation, adding one component at a time

```
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X_reduced[:,:i], y,
    cv=kf_10, scoring='neg_mean_squared_error').mean()
    mse.append(score)
```

PCR with Python

- Plot results

```
plt.plot(mse, '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary')
plt.show()
```



PCR with Python

- How many PC's do we choose?
- The smallest cross-validation error occurs when 18 components are used. There is no much dimension reduction!

```
min(mse)
```

```
## 114283.53982915755
```

```
mse.index(min(mse))
```

```
## 18
```

- We can calculate the percentage of variance or information, in the predictors explained by adding in each principal component to the model:
- By using the first 6 PC's only, we can explain 92.26% of the variation in the predictors.
- Note that we'll always be able to explain more variance by using more principal components, but we can see that adding more principal components doesn't actually increase the percentage of explained variance by much.

```
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
## array([38.31, 60.15, 70.84, 79.03, 84.29, 88.63, 92.26, 94.96, 96.28,
##        97.25, 97.97, 98.64, 99.14, 99.46, 99.73, 99.88, 99.95, 99.98,
##        99.99])
```

PCR with Python

- In the LS regression, 54.61% of variation of the response can be explained by all predictors (19 PCs)

```
fullmodel=regr.fit(X,y)
fullmodel.score(X,y)
```

```
## 0.5461158619125323
```

PCR with Python

- sklearn.pipeline.make_pipeline can be used to combine PCA and regression
  - https://scikit-learn.org/stable/auto_examples/cross_decomposition/plot_pcr_vs_pls.html
- Let's calculate the percentage of variance in the response variable explained by adding in each principal component to the model?

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
import numpy as np

score_pcr=np.zeros(19)
for i in range(19):
    pcr=make_pipeline(PCA(n_components=i+1), LinearRegression())
    pcr.fit(X, y)
    score_pcr[i]=pcr.score(X, y)
```

```
## Pipeline(steps=[('pca', PCA(n_components=1)),
##                 ('linearregression', LinearRegression())])
## Pipeline(steps=[('pca', PCA(n_components=2)),
##                 ('linearregression', LinearRegression())])
## Pipeline(steps=[('pca', PCA(n_components=3)),
##                 ('linearregression', LinearRegression())])
## Pipeline(steps=[('pca', PCA(n_components=4)),
##                 ('linearregression', LinearRegression())])
```

PCR with Python

- 46.69% of variation of the response can be explained by the 6 PCs.
- 48.07% of variation of the response can be explained by the 7 PCs.
- 48.07% of variation of the response can be explained by the 8 PCs.

```
print(score_pcr)
```

```
## [0.28148566 0.37222424 0.38230534 0.43613293 0.43698424 0.43768642
##  0.48072174 0.48072179 0.49552549 0.50325527 0.50438393 0.51687864
##  0.52688361 0.52710525 0.52781946 0.52792357 0.53068243 0.5454956
##  0.54611586]
```

```
print(score_pcr[5])
```

```
## 0.4376864248787323
```

```
print(score_pcr[6])
```

```
## 0.48072173539949037
```

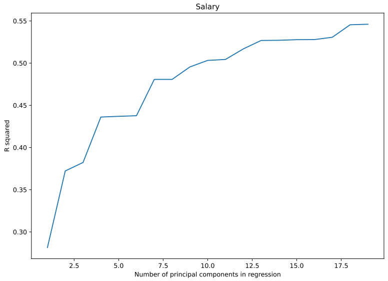
```
print(score_pcr[7])
```

```
## 0.48072179371469426
```

PCR with Python

- Plot of number of PC's in PCR versus R^2

```
plt.plot(range(1,20), score_pcr)
plt.xlabel('Number of principal components in regression')
plt.ylabel('R squared')
plt.title('Salary')
plt.show()
```



- As a result of the way PCR is implemented, the final model is more difficult to interpret compared to Ridge and Lasso because it does not perform any kind of variable selection or even directly produce coefficient estimates.

Partial Least Squares

- PCR identifies linear combinations, or directions, that best represent the predictors $X_1, \ldots, X_p$.
- These directions are identified in an **unsupervised way**, since the response Y is not used to help determine the principal component directions.
- That is, the response does not supervise the identification of the principal components.
- Consequently, PCR suffers from a potentially serious drawback: there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.

Partial Least Squares

- Like PCR, PLS is a dimension reduction method, which first identifies a new set of features $Z_1, \ldots, Z_M$ that are linear combinations of the original features, and then fits a linear model via OLS using these M new features.
- But unlike PCR, PLS identifies these new features in a **supervised way** - that is, it makes use of the response Y in order to identify new features that not only approximate the old features well, but also that are **related to the response**.
- Roughly speaking, the PLS approach attempts to find directions that help explain both the response and the predictors.

Details of Partial Least Squares

- After standardizing the p predictors, PLS computes the first direction Z_1 by setting each ϕ_{1j} in $Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j$ equal to the coefficient from the **simple linear regression of Y onto X_j**.
- One can show that this coefficient is proportional to the **correlation** between Y and X_j.
- Hence, in computing $Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j$, PLS places the highest weight on the variables that are most strongly related to the response.
- Subsequent directions $Z_m = \sum_{j=1}^{p} \phi_{mj} X_j, m = 2, \ldots, M$ are found by taking residuals after regression of the original data on Z_{m-1}, and Z_m is calculated in the same way as Z_{m-1} for the residuals data (orthogonalized data), then repeating the above prescription.

PLS with Python

- Scikit-learn `PLSRegression` gives same results as the `pls` package in R when using method=‘oscorespls’. However, the standard method used is `kernelpls`.

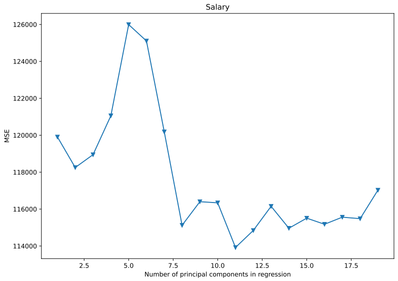
```
from sklearn.cross_decomposition import PLSRegression, PLSSVD
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)

mse = []

for i in np.arange(1, 20):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(pls, scale(X), y, cv=kf_10,
    scoring='neg_mean_squared_error').mean()
    mse.append(-score)

print(mse)
```

```
## [119906.18876648725, 118251.53530632716, 118948.07398183263, 121055.6701
```

PLS with Python

- Plot results

```
plt.plot(np.arange(1, 20), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary')
plt.show()
```



PLS with Python

- The smallest cross-validation error occurs when 11 components are used.

```
min(mse)
```

```
## 113920.50255648288
```

```
mse.index(min(mse))+1
```

```
## 11
```

PLS with Python

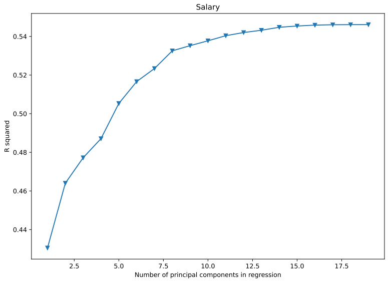
```python
score_pls=np.zeros(19)

for i in np.arange(1, 20):
    pls = PLSRegression(n_components=i)
    pls.fit(X,y)
    score_pls[i-1]=pls.score(X,y)
```

```
## PLSRegression(n_components=1)
## PLSRegression()
## PLSRegression(n_components=3)
## PLSRegression(n_components=4)
## PLSRegression(n_components=5)
## PLSRegression(n_components=6)
## PLSRegression(n_components=7)
## PLSRegression(n_components=8)
## PLSRegression(n_components=9)
## PLSRegression(n_components=10)
## PLSRegression(n_components=11)
## PLSRegression(n_components=12)
## PLSRegression(n_components=13)
## PLSRegression(n_components=14)
## PLSRegression(n_components=15)
```

PLS with Python

- Plot of number of PC's in PLS versus R^2

```
plt.plot(range(1,20), score_pls, '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('R squared')
plt.title('Salary')
plt.show()
```



PLS with Python

- Recall that in the LS regression, 54.61% of **variation of the response** can be explained by all predictors (19 PCs)

```
regr = LinearRegression()
fullmodel=regr.fit(X,y)
fullmodel.score(X,y)
```

```
## 0.5461158619125323
```

- 54.04% of variation in the response can be explained by the 11 PCs.

```
print(score_pls[11-1])
```

```
## 0.5403846500241162
```

A simulation study with PLS

- A simulation study

```
import numpy as np
from numpy import random
random.seed(1)
xtr=random.normal(0, 1, size=(100, 100))
beta=np.array([1]*10+[0]*90) # vector of length 100
ytr=np.dot(xtr, beta)+random.normal(0, 1,size=100)

import matplotlib.pyplot as plt
from sklearn.cross_decomposition import PLSRegression
from sklearn.preprocessing import scale
from sklearn.model_selection import KFold, cross_val_score
```

A simulation study with PLS

- A simulation study

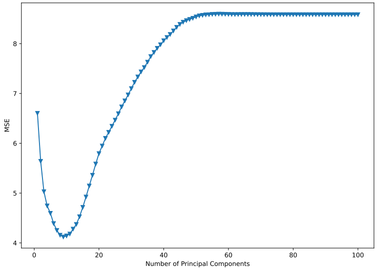
```
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)
mse = np.zeros(100)

for i in np.arange(1, 101):
  pls = PLSRegression(n_components=i)
  score = cross_val_score(pls, scale(xtr), ytr, cv=kf_10,
  scoring='neg_mean_squared_error').mean()
  mse[i-1]=-score
```

```
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site
##   warnings.warn(f"Y residual is constant at iteration {k}")
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site
##   warnings.warn(f"Y residual is constant at iteration {k}")
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site
##   warnings.warn(f"Y residual is constant at iteration {k}")
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site
##   warnings.warn(f"Y residual is constant at iteration {k}")
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site
```

A simulation study with PLS

- Plot the CV errors

```
plt.plot(np.arange(1, 101), np.array(mse), '-v')
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.show()
```



License