High Dimensional Regression

Chenghui Li Haoxiang Wei Mufang Ying Qintao Ying

May 10, 2019

Abstract

In regression problems where the number of predictors greatly exceeds the number of observations, conventional regression techniques may produce unsatisfactory results. For this high dimensional regerssion situation, We used four methods, PCA, PLS, LASSO and SPCA, to address this problem. Supervised principal components is similar to conventional principal components analysis except that it uses a subset of the predictors selected based on their association with the outcome. Supervised principal components can be applied to regression and generalized regression problems. We compared it to the other three methods which can also account for the effects of the co-variates and help identify which predictor variables are most important. And we used two simulated gene datasets as illustration.

KEY WORDS: High Dimensional Regression; SPCA; Simulate; Gene.

1 Introduction

In regression problems where the number of predictors greatly exceeds the number of observations, linear regression approaches may lead to unsatisfactory results. Other conventional approaches like Principle Component Analysis can deal with such problem but there are other questions that may harm.

For this high dimensional situation, we will use PCA, LASSO, PLS and SPCA to address this problem. Some functions are already implemented in some R packages. We be will use test error and cross validation error to show the power of each method.

Among them, the supervised principal components analysis is the best. It approaches this difficult problem through a semi-supervised strategy, looking for gross structure in the data that aligns itself with the outcome. A key aspect of the method is the pre-selection of features according to their correlation with the outcome. This alleviates the effect of a larger number of noisy features on the prediction model.

And as far as we know, Bair and Tibshirani (2004) were the first to discuss the idea of supervised principal components in detail[3]. But other authors have presented related ideas. Ghosh (2002) prescreened genes before extracting principal components, but seemed to do so for computational reasons. Jiang et al. (2004) used a similar idea in the context of merging the results from two different datasets. Nguyen and Rocke (2002) and Hi and Gui (2004) discussed partial least squares (PLS) approaches to survival prediction from microarray data.

In the next section, we will show our statistical analysis using two simulated datasets.

2 Statistical Analysis

2.1 Data Simulation

We performed two simulation studies to compare the performance of the methods that we have considered. Each simulated dataset X considered of 5000 "genes"(rows) and 100 "micro-organisms"(columns). Let xij denote the "expression level" of the ith gene and jth micro-organism. In the first

study we generated data as

$$x_{ij} = \begin{cases} 3 + \epsilon_{ij} & (i \leq 50, j \leq 50) \\ 4 + \epsilon_{ij} & (i \leq 50, j > 50) \\ 3.5 + \epsilon_{ij} & (i > 51) \end{cases} \tag{1}$$

where the ϵ_{ij}'s are independent normal random variables with mean 0 and variance 1. We also let

$$y_j = \frac{\sum_{i=1}^{50} x_{ij}}{25} + \epsilon_j \tag{2}$$

where the ϵ_js are independent with normal random variables with mean 0 and standard deviation 1.5.

We designed this simulation so that there are two "sub-classes". The first 50 individuals belong to class 1, and have slightly lower average expression levels in the micro-organisms with class 2.

Next, we generated a "harder" simulated dataset. In this simulation, we generated each x_{ij} as follows:

$$x_{ij} = \begin{cases} 3 + \epsilon_{ij} & (i \leq 50, j \leq 50) \\ 4 + \epsilon_{ij} & (i \leq 50, j > 50) \\ 3.5 + 1.5 \cdot I(u_{1j} < 0.4) + \epsilon_{ij} & (51 \leq i \leq 100) \\ 3.5 + 0.5 \cdot I(u_{2j} < 0.7) + \epsilon_{ij} & (101 \leq i \leq 200) \\ 3.5 - 1.5 \cdot I(u_{3j} < 0.3) + \epsilon_{ij} & (201 \leq i \leq 300) \\ 3.5 + \epsilon_{ij} & (i > 301) \end{cases} \tag{3}$$

Here the u_{ij} are uniform random variables on (0,1) and $I(x)$ is an indicator function. For example, for each of the genes 51-100, a single value u_{1j} is generated for sample j; if this value is larger than 0.4, then all of the genes in that block get 1.5 added. The motivation for this simulation is that there are other clusters of genes with similar expression patterns that are unrelated to y.

2.2 Latent Model

We should know how good each model is when we have the practical MSE and also how much the model can explain the data given practical MSE. To learn this, we should introduce the latent model.

The question is that we need use x to estimate y. We will use some data to fit the relationship of x and y, and then use different x to predict y and thus use $\hat{y}$ to derive practical MSE. The models are used to find the underlying model of x and y.

The latent model is the underlying model that we do not know in prediction but we use this model to generate y. If we have already known what the model is, the best way to estimate y by using x is to use the unbiased estimator. However, even if we have already known what the latent model is, we still cannot derive totally correct y from x because there is randomness in the generation procedure. From the mathematical statistics, the best estimate should be the unbiased estimate with all data generation method known. So, it still has 1.5 standard error with latent model.

We use the following formula to show this truth. Suppose the following is the latent model:

$$\begin{aligned} y_j &= \frac{\sum_{i=1}^{50} x_{ij}}{25} + \epsilon_j, \\ \epsilon_j &\sim N(0, \sigma^2) \end{aligned} \tag{4}$$

So, if we use x_{ij} to estimate y_j given this model, we will use the following estimator

$$\hat{y}_j = \frac{\sum_{i=1}^{50} x_{ij}}{25} \tag{5}$$

Then, we will have

$$\begin{aligned} &E[(\hat{y}_j - y_j)^2] = E[E[(\hat{y}_j - y_j)^2 | x_{ij}]] \\ &= E[E[(\frac{\sum_{i=1}^{50} x_{ij}}{25} - (\frac{\sum_{i=1}^{50} x_{ij}}{25} + \epsilon_j))^2 | x_{ij}]] \\ &= E[E[\epsilon_j^2 | x_{ij}]] = E[\epsilon_j^2] = Var[\epsilon_j] = \sigma^2 \end{aligned} \tag{6}$$

Also,

$$E(\hat{y}_j - y_j) = E[E(\hat{y}_j - y_j | X)] = E[E[\epsilon_j | X]] = 0 \tag{7}$$

Remember that based on the latent model, we can only have 2.25 variance of estimates, and the model like lasso or PCA are only used to find the latent

model and they also have nothing to deal with randomness, so in fact even if they can reveal what the latent model truly is, the mean of practical MSE of the latent model should still be no smaller than 2.25. The smaller MSE the model has, the more the model can explain for data. So, in conclusion, we expect the mean of practical MSE of all models should be no less than 2.25 given the latent model 5.

We should point out that the above deduction is true regardless the random generation way of X_{ij}. Namely, we cannot control the way to generate X_{ij} and also have no assumption about it, but once we are considering the expectation of $(\hat{y}_i - y_i)^2$, we can put a condition on it and see X matrix as known. The final result will be independent with what x_{ij} is.

However, in simulation procedure, there is randomness in the model fitting steps and data formulating process, which may affect practical MSE of each model. So, there may be some fluctuates in the final CV error and test error and this would be reasonable. We can only use more data and more times to simulate to approximate the real MSE. So, even if the practical MSE is slightly smaller than 2.25, the result will be acceptable.

2.3 PCA

When X is linearly dependent, PCA should be the most popular method for it because it is a technique for reducing the dimension of a data matrix. Technically, we find a direction which can maximize the variance of the data, in another word, maintain most of the information. Because this is a popular method, we will skip the steps of how to get principle components.

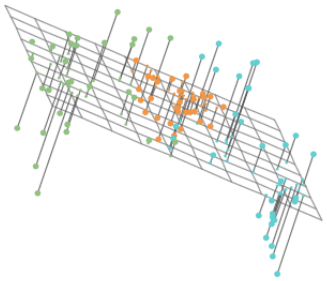

Figure 1: Geometry prespective of PCA

2.3.1 Advantages

The principle components are linear combinations of the features. If there is multicollinearity in data matrix, PCA would be a great choice for us to avoid the problem because principle components are orthogonal to each other. In computer like R language, it is easy to implement and there are many default functions for us to use.

2.3.2 Drawbacks

However, there are also some drawbacks of principle components in supervised learning. According to the procedure of finding principle components, principle components are not essentially related to the response variable. Besides, sometimes it is hard for us to interpret the meaning of different principle components.

2.3.3 PCA in high dimensional regression

In this setting, the number of columns in data matrix exceeds the number of rows. If we denote p as the number of features and n as the number of rows, we will see $p >> n$. Compared with the original case, we have severe multicollinearity here and the number of principle components would be at most **n**. This fact tells us we can pick at most **n** principle components.

Some functions in R fail because of the high dimensional setting - the matrix $X^T X$ is not of full rank. By using the svd algorithm, we will not have such trouble. The svd algorithm is implemented in the package **pls**. Now we will use two rules to determine the number of principle components here.

- Minimize the mean square prediction error
- Principle components can explain most of the variances

In Figure 2 and 3 are the validation plots for two datasets.

As we can see, the optimal number of principle components is not so clear in these cases. From my perspective of view, the best way to determine the number of principle components is to try different choices and compare them by test error.

For the first dataset in Figure 2, we see that the line is quite flat, which means it is not easy to determine the best number of principle components.

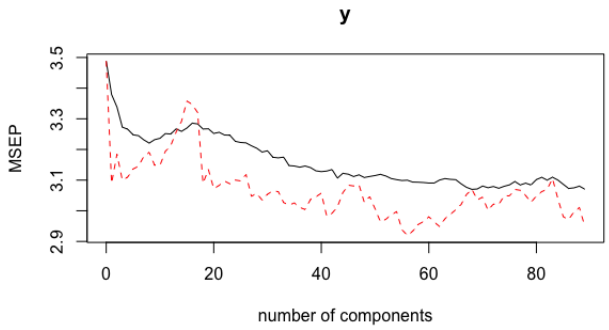

Figure 2: dataset1

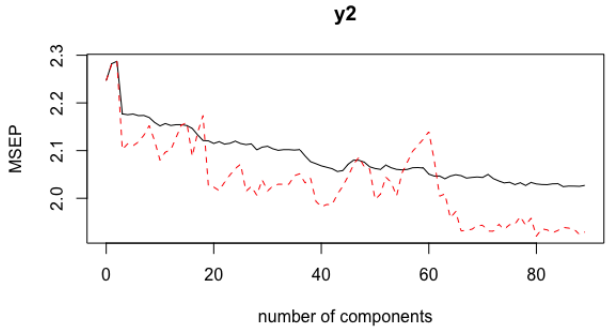

Figure 3: dataset2

But our goal is to compare prediction error. So I use a for loop to find the optimal number of principle components that minimizes the test error.

The result is 47 principle components which can explain 53.45 percentage variances for data matrix X. As we can see, in interpretation and model fitting, this should not be a good result.

For the second dataset in 3, I get the solution using the same criterion. In the end, 74 principle components can explain 79.39 percentage variances for data matrix.

2.4 PLS

2.4.1 Introduction

PLS (Partial Least Squares Regression) is an extension of the multiple linear regression model (e.g., Multiple Regression or General Stepwise Regression). The simplest form is that a linear model specifies the (linear) relationship between a dependent (response) variable Y, and a set of predictor variables, the X's, so that $Y = b_0 + b_1X_1 + b_2X_2 + \ldots + b_pX_p$.

2.4.2 Description

The multiple linear regression model serves as the basis for a number of multivariate methods such as discriminant analysis, PCA , and canonical correlation. PLS has both the advantages of discriminant analysis, PCA, and canonical correlation. In PLS, prediction functions are represented by factors extracted from the $Y'XX'Y$ matrix. The number of such prediction functions that can be extracted typically will exceed the maximum of the number of Y and X variables. It can keep the relationship between X and Y while extracting linear uncorrelated components to decrease multicollinearity. As a result, PLS shows a better performance, because its number of such prediction functions (we can consider it as the components in PCA) that can be extracted typically will exceed the maximum of the number of Y and X variables. So this allows PLS to be used in situations where the use of traditional multivariate methods is severely limited, such as when there are fewer observations than predictor variables. Furthermore, PLS can be used as an exploratory analysis tool to select suitable predictor variables and to identify outliers before classical linear regression.

The working process of PLS is listed in following:

step 1: 1. Standardize each of the variables to have mean 0 and unit norm, and compute the univariate regression coefficients w=$\mathrm{X}^T\mathbf{y}$.

step 2: $\mathbf{u}_{\mathrm{PLS}} = \mathbf{Xw}$, and use it in a linear regression model with y.

step 3: Find w that $\max_{\|\mathbf{w}\|=1} \mathrm{corr}^2(\mathbf{y}, \mathbf{Xw})\,\mathrm{var}(\mathbf{Xw})$.

Bair, Hastie, Paul and Tibshirani concluded that the variance term dominates, and hence that PLS would in general be similar to PCA[1].

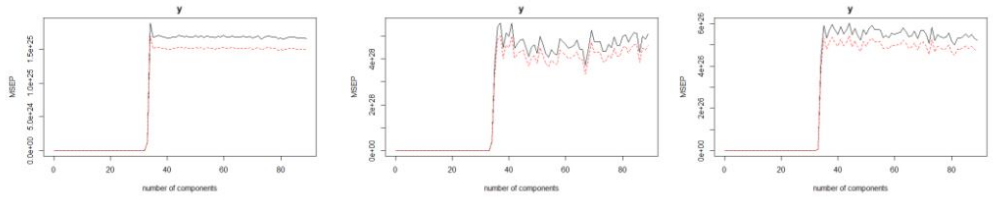

Figure 4: Scree plot

2.4.3 Potential Risk

One of PLS's risk is that PLS retains all features and can be adversely effected by the noise in the unimportant features, which means it cannot extract those unimportant features. We expect to solve this problem in the next algorithm: LASSO. Also, when we use scree plot in Figure 4 to find the most suitable component number **n** as we have done when using PCA, the outcome seems not reliable when **n** is large.

We can see when **n** approaches 30, the MSEP has a sudden explode and increase exponentially. So a large **n** is not reliable, we have to choose a small **n**. At last we chose **n=8** by comparing CV-error. But we have to admit that this choice is not very confident. The final results of CV-error and test error will be put in the conclusion section.

2.5 LASSO

2.5.1 Introduction

LASSO (Least Absolute Shrinkage and Selection Operator) is another regression analysis algorithm we often use when dealing with real problems. In general it is similar to ridge regression by restricting the size of variables, but LASSO change the penalty term to a absolute value form.

$$\min_{\beta} \|\mathbf{y} - \boldsymbol{\beta}_0 - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{8}$$

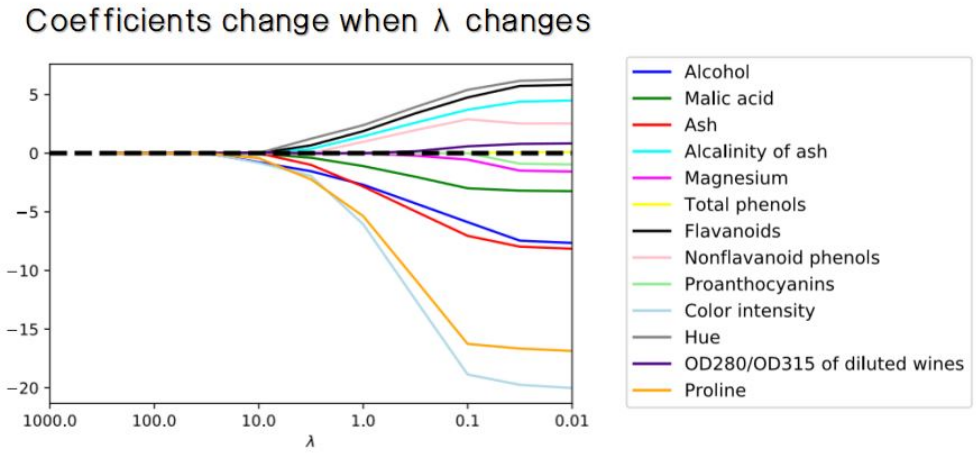

Figure 5: coefficients

2.5.2 Description and Advantages

LASSO is developed from ridge regression, so it also solves a restricted linear regression problem. Although sparse solutions are generally attractive, these solutions may be too sparse, because, for example, for micro-array data they would allow only N genes to appear in a given model.

The result of LASSO will change when λ change. From Figure 5 we can clearly see the effect of different λ.

We can see the penalty will increase when λ increase, hence the coefficients decrease toward zero. The absolute value form of the penalty has the attractive property that it can shrink some coefficients exactly to zero. So the increase of λ can decrease multicollinearity. Moreover, since it can shrink some of the variables to zero, LASSO helps feature selection. The noise features usually have a small coefficients in simple linear regression, with the penalty term we can make their coefficients to zero and hence deleted these features.

2.5.3 Potential Risk

Computation of the lasso is more challenging than computation of ridge regression. LASSO is a convex optimization process, which can be very difficult if the number of features p is large. In application, we have to test

again and again to find the value of λ. Although it seems easy when dealing with simple cases by using R packages and functions (e.g. our simulation datasets), it might be very time consuming when dealing with large datasets. Another problem is that LASSO cannot choose more than **n** features (here n is sample size). So while meeting with high dimensional case in real problems, it is possible that LASSO performs bad (Although this is not the case in our project). The final results of CV-error and test error will be put in the conclusion section.

2.6 Supervised Principle Component Analysis

2.6.1 Introduction

We assume p is the number of features and N is the number of observations. Let X be $N * p$ matrix and y be a vector of N coordinates. We assume the regressor and predictors as quantitative variables. We are interested in the prediction, which is to use X matrix to predict the regressor y.

when $p >> N$, this is a high dimensional problem. The linear regression will not work. Naively, we can try PCA and PCA still can work. As we know, PCA is seen as the conventional model to handle the multicollinearity. However, there are many problems in PCA. The main problems are

- The maximum number of principle components is p, while the cross validation procedure may show that it need more than p components to reach the minimum MSE.
- PCA will choose the principle direction which can explain much of predictor, but may have no much relationship with the regressor.
- All predictors will be involved in model, which makes the model hard to interpret.

The first problem only shows in high dimensional problem and the second and third problems will show whatever the dimension of data is.

PLS is proposed based on these problems in PCA and it can fix the second problem. However, it has nothing to do with other problems.

Lasso can try to fix all problems of PCA. However, it is another way based on linear regression to explain data and it will lose advantages of PCA.

SPCA is proposed based on PCA. Compared with Lasso, it is more like PCA. So, SPCA will maintain the advantages of PCA and can also fix all

problems of PCA[2]. There will be further analysis after the description of this algorithm.

In the simulation part, we will see that SPCA behaves better than other 3 algorithms.

2.6.2 Description

Here we present what is proposed for SPCA:

step 1: Compute (univariate) standard regression coefficients for each feature.

step 2: Form a reduced data matrix consisting of only those features whose univariate coefficient exceeds a threshold in absolute value (is estimated by cross-validation).

step 3: Compute the first (or first few) principal components of the reduced data matrix.

step 4: Use these principal component(s) in a regression model to predict the outcome.

In fact, as we can see, these steps are followed the basic idea that some features may have no effect on the regressor.

So, it fixes PCA, but not like PLS which will tend to be overfitting, it will reduce the redundant predictor, which is like Lasso.[4]

In the first step, we use the correlation to determine whether features are correlated with the regressor. The uncorrelated predictors should be thrown away from our model.

In the second step, we consider shreshold. To find the shreshold, generally we will use cross validation. There are many principles in cross validation and in practice we will use likelihood ratio statistic.

In the third and fourth step, we use PCA way on the features correlated with regressor to make the model and predict.

Generally, we believe that PCA can give us a nice prediction, but we want to prevent the model is very related with X but not y, so we need to leave out some redundant features. Also, the step to get rid of the uncorrelated step can help us to avoid all predictors appear in the final model and thus the model would not become hard to interpret.

2.6.3 Potential Risk

Although it seems that SPCA can have a good performance in explaining data, there are some undeniable drawbacks.

This model should always be based on the regression model. In fact, if there is secondary order part in the real relationship of y and X, this would not give the hint of it. As we known, every linear model can hardly reveal the high order relationship and it will have a bad prediction result.

Formally, the relationship must be like

$$y = x\beta + \epsilon \tag{9}$$

This disadvantage appears either on the methods used to compare with supervised PCA. We should be aware of it so if the result is bad, we can explain it and change to another fitting model.

In this paper we will focus on the prediction in regression.

In simulation, even if the data generation process is a little complicated and there may be some randomness which leads the residual error part do not follow normal distribution, it is still under a regression model, so we do not need to worry too much about it.

2.6.4 The number of components in SPCA

Generally, we will use only 1 principle component in the PCA step in SPCA[1], however, the number of principle component will matter. So, in fact, if we only use 1 principle component, the model may be based on data but cannot reveal what the underlying model is.

I will use the first data simulation process to generate 100 observations and show an example about the number of principle components in the third step in SPCA. 70 observations are used to fit the model and 30 observations to test.

The fitted results show in Table 1. As the table shows, models all have a good prediction because MSE_p are all closed to 2.25 and 3 principle component can do the best.

Also, the less components will introduce less variables into the model. In fact, when only use 1 component in model, it will have only a few variables in model, which makes the model truly bad although it can have a good prediction. Otherwise, when we introduce more components, the model will behave better.

Table 1: Principle Components

# Components	Practical MSE	# Variables
1	2.37	9
2	2.03	56
3	1.94	56

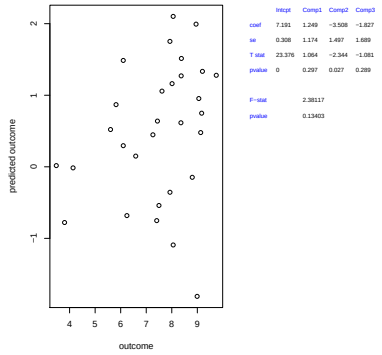

Figure 6: Residual plot

When 2 or 3 components, the number of variables in model is pretty closed to that of the latent model. Recall that the number of latent model is 50. So, 2 or 3 components models behave pretty good in this sense. They can not only have a good prediction but also reveal what the underlying model is.

Undoubtedly, if only 1 principle component introduced in SPCA, the chance to make SPCA approach the underlying model will be lost. Also, more components may make MSE smaller. So, we should not directly introduce only 1 principle component in the third step of supervised principle component analysis.

We will focus on the 3 components model because it is the best model among these models. residual plot will be presented as Figure 6.

Notice that there is no extreme points in the residual plot, so 3 components model should be good.

Even for the residuals plot of 1 or 2 component model in Figure 7, there may some residuals far from 0, but the absolute values are smaller than $2\sigma^2 = 4.5$, so the models are still good at predicting.

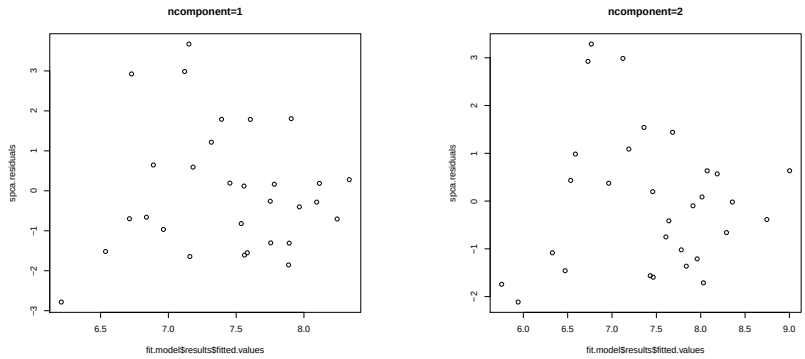

Figure 7: Residual Plot

In conclusion, SPCA can maintain less variables and thus can greatly simplify PCA model. What's more, the number of principle components will greatly affect the final result, so we should carefully test which is the best principle components for SPCA.

3 Conclusions

Here are the tables we made. The table 2 and table 3 are coresponding to the first simulated dataset and the second simulated dataset.

CV errors are computed using the dataset we simulated. The data is divided into 10 folders. We use 9 folders as training data and the left folder as test data(10-folders cross validation). Each time there is one folder left as test data. Then we can compute the sum of squared prediction errors.

According to the article, we need to simulate the same dataset again to get the test error. We use the first dataset to build the model and use the second one to compare prediction errors.

Table 2: Methods comparasion for the first dataset

Methods	**CV error**	**test error**
PCR-1	337.8244	304.4352
PCR	311.8756	284.7703
PLS	334.5241	301.5314
LASSO	169.80718	290.4857
SPCA	144.2776	237.2667

Table 3: Methods comparasion for the second dataset

Methods	**CV error**	**test error**
PCR-1	352.6884	288.2429
PCR	313.9984	244.7681
PLS	345.2164	261.6794
LASSO	174.82852	268.5382
SPCA	160.5622	223.5655

4 Contribution

- Chenghui Li: SPCA part coding, presentation and SPCA part paper work.
- Haoxiang wei: PLS part coding, presentation and LASSO and PLS part paper work.
- Mufang Ying: PCA part coding, presentation and PCA part paper work.
- Qintao Ying: Lasso part coding, presentation and introduction part paper work.

Finally, we need to say thank you to our teacher, without whose help we may not be able to find such an interesting topic. And thank teammates for consistent efforts. Last but not least, we admit, each part contains the contribution from our group members.

Appendix: R code

```
######################## First Data Generation ########################
#In fact, in the test error and cv error parts we will generate it again.
#So, this is for convience.

# First dataset
set.seed(1)
X1 = matrix(0,nrow = 5000,ncol = 100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X1[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X1[i,j] = 4+rnorm(1)
    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}

y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)

train = sample(1:nrow(t(X1)),floor(nrow(t(X1))*0.7))
X1 = t(X1)


######################## First Data Generation ########################

set.seed(1)
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
```

```
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)
X2 = t(X2)


######################## PCA: First Dataset ########################
set.seed(1)
  X1 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X1[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X1[i,j] = 4+rnorm(1)
      }else{
        X1[i,j] = 3.5 + rnorm(1)
      }
    }
  }
  y = apply(X1[1:50,], 2, sum)
  y = y/25+rnorm(100,sd = 1.5)
  X1 = t(X1)

pcr.model = pcr(y ~ X1, scale = T,validation = "CV")
validationplot(pcr.model,val.type = "MSEP")


X1 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X1[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X1[i,j] = 4+rnorm(1)
      }else{
```

```
        X1[i,j] = 3.5 + rnorm(1)
      }
    }
  }
  y = apply(X1[1:50,], 2, sum)
  y = y/25+rnorm(100,sd = 1.5)
  X1 = t(X1)


test_error = 100000
k = 100
for(i in 1:89){
pred.y = predict(pcr.model,X1,ncomp = i)
error = sum((y-pred.y)^2)
if(i == 1){
  cat("The test error for one principle component is ",error,"\n")
}
if(error<test_error){
  k = i
  test_error = error
}
}

cat("The optimal number is ",k,"\n")
cat("The minimum test error is ",error,"\n")


############################## PCA: Second Dataset ###################################
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
```

```
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y2 = apply(X2[1:50,],2,sum)
y2 = y2/25+rnorm(100, sd = 1.5)
X2 = t(X2)

pcr.model2 = pcr(y2 ~ X2, scale = T,validation = "CV")
validationplot(pcr.model2,val.type = "MSEP")

X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y2 = apply(X2[1:50,],2,sum)
y2 = y2/25+rnorm(100, sd = 1.5)
X2 = t(X2)


test_error = 100000
k = 1000
```

```
for(i in 1:89){
pred.y = predict(pcr.model2,X2,ncomp = i)
error = sum((y2-pred.y)^2)
if(i == 1){
  cat("The test error for one principle component is ",error,"\n")
}
if(error<test_error){
  k = i
  test_error = error
}
}
cat("The optimal number is ",k,"\n")
cat("The minimum test error is ",error,"\n")
```

```
############################## PLS ###################################
library(pls)

#################### PLS: First Dataset ##############################
set.seed(1)
u=c()
for(times in 1:10){
  X1 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X1[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X1[i,j] = 4+rnorm(1)
      }else{
        X1[i,j] = 3.5 + rnorm(1)
      }
    }
  }

  y = apply(X1[1:50,], 2, sum)
  y = y/25 + rnorm(100, sd = 1.5)

  X1 = t(X1)

  # use {y,X1} train model
  pls.model = plsr(y ~ X1, scale=TRUE, validation = "CV",segments = 10)
  validationplot(pls.model, val.type="MSEP")
```

```
  #new data generation
  X2 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X2[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X2[i,j] = 4+rnorm(1)
      }else{
        X2[i,j] = 3.5 + rnorm(1)
      }
    }
  }

  y1 = apply(X2[1:50,],2,sum)
  y1 = y1/25 + rnorm(100, sd = 1.5)

  X2 = t(X2)
  #use {y1,X2} to predict model and gain the prediction error

  pls.pred = predict( pls.model , X2, ncomp = 8)
  u[times] = sum((pls.pred - y1)^2)

  #assign error
}
mean(u)

######################## PLS: Second Dataset ########################


test.error = c()

set.seed(1)
for(times in 1:10){

X1 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X1[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
```

```
      X1[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X1[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X1[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X1[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)
X1 = t(X1)

  # use {y,X1} train model
  pls.model2 = plsr(y ~ X1, scale=TRUE, validation = "CV",segments = 10)
  # validationplot(pls.model2, val.type="MSEP")

  #new data generation

X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
```

```
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)
X2 = t(X2)
  #use {y1,X2} to predict model and gain the prediction error

  pls.pred = predict( pls.model2 , X2, ncomp = 8)
  test.error[times] = sum((pls.pred - y1)^2)

  #assign error
}

mean(test.error)

############################## LASSO ###################################
#lasso
library(glmnet)

######################### First Dataset ##############################
# test error
set.seed(1)
cv.error = c()
u=c()
for(times in 1:10){
  X1 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X1[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X1[i,j] = 4+rnorm(1)
      }else{
        X1[i,j] = 3.5 + rnorm(1)
      }
    }
  }

  y = apply(X1[1:50,],2,sum)
  y = y/25+rnorm(100,sd=1.5)

  X1 = t(X1)
```

```
  #use {y,X1} train model

  cv.model = cv.glmnet(X1, y, type.measure = "mse", nfolds = 10)
  model = glmnet(X1, y)
  cv.error[times] = 100 * cv.model$lambda.1se

  #new data generation
  X2 = matrix(0,nrow = 5000,ncol = 100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X2[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
        X2[i,j] = 4+rnorm(1)
      }else{
        X2[i,j] = 3.5 + rnorm(1)
      }
    }
  }

  y1 = apply(X2[1:50,],2,sum)
  y1 = y1/25+rnorm(100, sd = 1.5)

  X2 = t(X2)
  #use {y1,X2} to predict model and gain the prediction error
  u[times] = sum((predict(model, newx = X2, s = cv.model$lambda.min) - y1)^2)
}

mean(u)
mean(cv.error)

########################## Second Dataset ################################
set.seed(1)
cv.error = c()
u=c()
for(times in 1:10){
  X1 = matrix(0,nrow = 5000,ncol = 100)
  u1 = runif(100)
  u2 = runif(100)
  u3 = runif(100)
  for(i in 1:5000){
    for(j in 1:100){
      if(i<=50 & j <= 50){
        X1[i,j] = 3+rnorm(1)
      }else if(i <=50 & j >50){
```

```
      X1[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X1[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)
    }else if(i <=200){
      X1[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)
    }else if(i <=300){
      X1[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)
    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)
X1 = t(X1)

#use {y,X1} train model

cv.model = cv.glmnet(X1, y, type.measure = "mse", nfolds = 10)
model = glmnet(X1, y)
cv.error[times] = 100 * cv.model$lambda.1se

#new data generation
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)
    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)
    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)
    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
```

```
  y = y/25+rnorm(100,sd=1.5)
  X2 = t(X2)
  #use {y1,X2} to predict model and gain the prediction error
  u[times] = sum((predict(model, newx = X2, s = cv.model$lambda.min) - y1)^2)
}


mean(u)
mean(cv.error)

############################## SPCA ###################################
library(superpc)
featurenames <- paste("feature",as.character(1:100),sep="")

######################## SPCA: First Dataset ##########################

### SPCA part components number part

#one example test
#####data generation
set.seed(1)
X1 = matrix(0,nrow = 5000,ncol = 100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X1[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X1[i,j] = 4+rnorm(1)
    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}

y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)

train = sample(1:nrow(t(X1)),floor(nrow(t(X1))*0.7))
X1 = t(X1)


train = sample(1:nrow(t(X1)),floor(nrow(t(X1))*0.7))

traindata=list(x=t(X1[train,]),y=y[train], featurenames=featurenames)
testdata=list(x=t(X1[-train,]),y=y[-train], featurenames=featurenames)
```

```
superpc.train.2<-superpc.train(traindata,type = "regression")
superpc.cv.result<-superpc.cv(superpc.train.2,traindata)

# best shreshold is 12th
#superpc.cv.result$thresholds[12]=1.348038


fit.cts<- superpc.predict(superpc.train.2, traindata,testdata,
threshold=1.348038, n.components=1,    prediction.type="continuous")

fit.model=superpc.fit.to.outcome(superpc.train.2,
testdata,score = fit.cts$v.pred,print=F)


sum((testdata$y-fit.model$results$fitted.values)^2)
sum(fit.cts$which.features)

#48 parameters

#IN FACT, the following plot is the same as the next two one.
spca.residuals=fit.model$results$fitted.values-testdata$y
pdf("residualsplot_1.pdf")
plot(fit.model$results$fitted.values,spca.residuals,main="ncomponent=1")
dev.off()

pdf("predictionplot_1.pdf")
superpc.predictionplot(superpc.train.2,traindata,testdata,1.348038)
dev.off()

fit.red<- superpc.predict.red(superpc.train.2,
traindata,testdata, threshold= 1.348038)
fit.redcv<-superpc.predict.red.cv( fit.red,superpc.cv.result,
traindata, threshold=1.348038)
superpc.plotred.lrtest(fit.redcv)

pdf("cvplot.pdf")
superpc.plotcv(superpc.cv.result,cv.type=c("full","preval"))
dev.off()



### SPCA test error

# test error
set.seed(1)
```

```
u=c()
for(times in 1:10){
X1 = matrix(0,nrow = 5000,ncol = 100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X1[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X1[i,j] = 4+rnorm(1)
    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}

y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)

X1 = t(X1)

#use {y,X1} train model
traindata=list(x=t(X1),y=y, featurenames=featurenames)
superpc.train.2<-superpc.train(traindata,type = "regression")
superpc.cv.result<-superpc.cv(superpc.train.2,traindata)


thresholds=superpc.cv.result$thresholds[superpc.cv.result$scor[1,]==
max(superpc.cv.result$scor[1,])]



#new data generation
X2 = matrix(0,nrow = 5000,ncol = 100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
```

```
y1 = apply(X2[1:50,],2,sum)
y1 = y1/25+rnorm(100,sd=1.5)

X2 = t(X2)
#use {y1,X2} to predict model and gain the prediction error
testdata=list(x=t(X2),y=y1, featurenames=featurenames)
fit.cts<- superpc.predict(superpc.train.2, traindata,testdata,
threshold=thresholds, n.components=1,    prediction.type="continuous")

fit.model=superpc.fit.to.outcome(superpc.train.2,testdata,score =
fit.cts$v.pred,print=F)


u[times]=sum((y1-fit.model$results$fitted.values)^2)# write down the error part
}

### SPCA cross validation

#10-folds
#cross validation

#first we should split the total datas as 10 folds and then we can simulate

u=c()
set.seed(1)
for(counttime in 1:10){
X1 = matrix(0,nrow = 5000,ncol = 100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X1[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X1[i,j] = 4+rnorm(1)
    }else{
      X1[i,j] = 3.5 + rnorm(1)
    }
  }
}

y = apply(X1[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)

train = sample(1:nrow(t(X1)),floor(nrow(t(X1))*0.7))
X1 = t(X1)
```

```
featurenames <- paste("feature",as.character(1:5000),sep="")
n=length(y)

#first we should split the total datas as 10 folds and then we can simulate
require(caret)
flds <- createFolds(1:100, k = 10, list = TRUE,
returnTrain = FALSE)#we name it as split procedure
s=c()
for(i in 1:10){
  testdata=list(x=t(X1[flds[[i]],]),y=y[flds[[i]]], featurenames=featurenames)
  traindata<-list(x=t(X1[-flds[[i]],]),y=y[-flds[[i]]], featurenames=featurenames)
  superpc.train.1<-superpc.train(traindata,type = "regression")
  superpc.cv.result<-superpc.cv(superpc.train.1,traindata)

  fit.cts<- superpc.predict(superpc.train.1, traindata,testdata,
  threshold=superpc.cv.result$thresholds[superpc.cv.result$scor[1,]==
  max(superpc.cv.result$scor[1,])], n.components=3,    prediction.type="continuous")
  #a<-superpc.fit.to.outcome(superpc.train.1, newdata1,
  fit.cts$v.pred)
  fit.model=superpc.fit.to.outcome(superpc.train.1,data.test =
  testdata,score = fit.cts$v.pred,print=F)
  s[i]=sum((testdata$y-fit.model$results$fitted.values)^2)
}
u[counttime]=sum(s)

# we can simulate the 10-folds experiments multiple times by using loop
# the code in each loop is cv for 100observations.
}
#144.2776

######################## SPCA: Second Dataset ##########################

# test error

set.seed(1)
u=c()
for(times in 1:10){
X1 = matrix(0,nrow = 5000,ncol = 100)
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
```

```
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
y.train = y/25+rnorm(100,sd=1.5)
X.train = t(X2)

#use {y,X1} train model
traindata=list(x=t(X.train),y=y.train, featurenames=featurenames)
superpc.train.2<-superpc.train(traindata,type = "regression")
superpc.cv.result<-superpc.cv(superpc.train.2,traindata)


thresholds=superpc.cv.result$thresholds[superpc.cv.result$scor[1,]==
max(superpc.cv.result$scor[1,])]



#new data generation
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)
```

```
    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
y.test = y/25+rnorm(100,sd=1.5)
X.test = t(X2)

#use new {y1,X2} to predict model and gain the prediction error
testdata=list(x=t(X.test),y=y.test, featurenames=featurenames)
fit.cts<- superpc.predict(superpc.train.2, traindata,testdata,
threshold=thresholds, n.components=1,    prediction.type="continuous")

fit.model=superpc.fit.to.outcome(superpc.train.2,testdata,score = fit.cts$v.pred,print=F)


u[times]=sum((y.test-fit.model$results$fitted.values)^2)# write down the error part
}


#test error 223.5655

#cv 10folds
#10-folds
#cross validation

#first we should split the total datas as 10 folds and then we can simulate
set.seed(1)
u=c()
for(counttime in 1:10){
X1 = matrix(0,nrow = 5000,ncol = 100)
X2 = matrix(0,nrow = 5000,ncol = 100)
u1 = runif(100)
u2 = runif(100)
u3 = runif(100)
for(i in 1:5000){
  for(j in 1:100){
```

```
    if(i<=50 & j <= 50){
      X2[i,j] = 3+rnorm(1)
    }else if(i <=50 & j >50){
      X2[i,j] = 4+rnorm(1)
    }else if(i <= 100){
      X2[i,j] = 3.5+1.5*(u1[j]<0.4)+rnorm(1)

    }else if(i <=200){
      X2[i,j] = 3.5+0.5*(u2[j]<0.7)+rnorm(1)

    }else if(i <=300){
      X2[i,j] = 3.5-1.5*(u3[j]<0.3)+rnorm(1)

    }else{
      X2[i,j] = 3.5 + rnorm(1)
    }
  }
}
y = apply(X2[1:50,],2,sum)
y = y/25+rnorm(100,sd=1.5)
X2  = t(X2)

featurenames <- paste("feature",as.character(1:5000),sep="")
n=length(y)

#first we should split the total datas as 10 folds and then we can simulate
require(caret)
flds <- createFolds(1:100, k = 10, list = TRUE,
returnTrain = FALSE)#we name it as split procedure
s=c()
for(i in 1:10){
  testdata=list(x=t(X2[flds[[i]],]),y=y[flds[[i]]], featurenames=featurenames)
  traindata<-list(x=t(X2[-flds[[i]],]),y=y[-flds[[i]]], featurenames=featurenames)
  superpc.train.1<-superpc.train(traindata,type = "regression")
  superpc.cv.result<-superpc.cv(superpc.train.1,traindata)

  fit.cts<- superpc.predict(superpc.train.1, traindata,testdata, threshold=
  superpc.cv.result$thresholds[superpc.cv.result$scor[1,]==
  max(superpc.cv.result$scor[1,])], n.components=3,    prediction.type="continuous")
  #a<-superpc.fit.to.outcome(superpc.train.1, newdata1, fit.cts$v.pred)
  fit.model=superpc.fit.to.outcome(superpc.train.1,data.test =
  testdata,score = fit.cts$v.pred,print=F)
  s[i]=sum((testdata$y-fit.model$results$fitted.values)^2)
}
u[counttime]=sum(s)
```

```
# we can simulate the 10-folds experiments multiple times by using loop
#164.8654 for one time cross validation
}
#160.5622 for loop
```

References

[1] Eric Bair, Trevor Hastie, Debashis Paul, and Robert Tibshirani. Prediction by supervised principal components. *Journal of the American Statistical Association*, 101(473):119–137, 2006.

[2] Eric Bair, Trevor Hastie, Debashis Paul, and Robert Tibshirani. Prediction by supervised principal components. *Journal of the American Statistical Association*, 101(473):119–137, 2006.

[3] Eric Bair and Robert Tibshirani. Semi-supervised methods to predict patient survival from gene expression data. *PLoS biology*, 2(4):e108, 2004.

[4] Fariba Khodayar, Saeed Sojasi, and Xavier Maldague. Infrared thermography and ndt: 2050 horizon. *Quantitative InfraRed Thermography Journal*, 13(2):210–231, 2016.