HW4_arkam2

March 1, 2021

1 STAT 542: Homework 4

1.1 NAME - ARKA MITRA, netid: arkam2

1.2 SPRING 2021, by Ruoqing Zhu (rqzhu)

1.3 Due: Tuesday, Mar 2, 11:59 PM CT

2 Question 1 (40 Points) Correrlated Variables in Ridge

For this question, we will use the same Bitcoin data we already constructed in HW3. Use the same training and testing data construction. You may have already noticed that if a linear model is used to perform that task, some parameters cannot be properly estimated, for example, the n_transactions_total parameter is only estimated on day 1, while for day 2 and 3, their parameter estimates are NA because they are both highly linearly dependent on day 1. Co-linearity is a severe problem of linear regression but Ridge regression can be used to solve that problem.

2.1 (20 Points) A Simulation Study

For this sub-question you must write your own code without using loading any additional package, but you can still use the lm() function from the base package. Construct your data in the following way

$$Y = X_1 \times 0.7 + X_2 \times 0.8 + X_3 \times 0.9 + \epsilon$$

where X_1 and ϵ are generated independently from standard normal distribution, and the other two covariates are simply copies of X_1, meaning that $X_2 = X_3 = X_1$. Generate 100 samples from this model, make sure that you save a random seed. You should write down your own code to implement the ridge regression, and no intercept is needed. Answer the following questions before you actually fit the models:

What will happen (regarding the parameter estimates) when you fit a linear regression using data from this model? And why?

Will Ridge regression be able to address this problem? Will the parameters of the three variables be different or the same?

Using the singular value decomposition (SVD) and shrinkage understanding we developed during the lecture to explain why you would expect such results from the ridge regression.

Now, use your generated data to confirm your statements. You do no need to tune the penalty term λ, just fix it at any nonzero value. Display sufficient information to support your statements, this includes the eigen-values and the rotation matrix $\mathbf{V}$ from the SVD of X.

After preforming the above, modify your data by multiplying your X_1 by 2. Then use your code to fit the ridge regression again. What changes do you observe? Are the fitted value $\hat{y}$ changing?

2.2 Answer:

Before conducting the simulations, I think the answers to the questions are as follows:

(a) The correlation between X_n is 1, so the covariance matrix is singular. However, the parameter estimates cannot be calculated using $ (X^{TX)}{-1}X^Ty$ as the matrix $ (X^TX)$ is not invertible. So, in computation, the correlated covraiates (in this case, X_2 and X_3) are ignored, and the linear model will fit the regression only wrt X_1. Coefficients returned for all X_n will be equal.

(b) Yes, ridge regression solves the problem by removing the problem of non-invertibility of (X^TX). That is because the singular matrix in this case is $(X^TX + \lambda I)^{-1}X^T$. However, in this case as well, all X_n are exactly correlated, the coefficients of the regression will remain unchanged from linear regression.

(c) This will shrink the coefficients of the model by a factor given by $\frac{d_i^2}{d_i^2+\lambda}$, where d_i is the $i-th$ singular value of the SVD $X = UDV^T$.

[2]:
```
import numpy as np
from scipy.stats import norm
from IPython.display import display, Latex
from sklearn.linear_model import LinearRegression

X1 = norm.rvs(size = 100 , random_state = 1)
X2, X3 = X1, X1
X = np.append(np.append(X1.reshape(100,1) , X2.reshape(100,1), axis = 1) , X3.
 ↪reshape(100,1), axis = 1)
epsilon = norm.rvs(size = 100, random_state = 2)
Y = X.dot([0.7, 0.8, 0.9]) + epsilon

#Fitting a linear regression
reg = LinearRegression(fit_intercept = False)
reg.fit(X, Y)
display(Latex(r'$\hat\beta _{OLS}$ ')); display(str(reg.coef_))

#Fitting a ridge regression
ridgeRegression = lambda X, y, alpha = 1.0 : np.linalg.inv(X.T.dot(X) + alpha*np.
 ↪eye(X.shape[1])).dot(X.T.dot(y))
beta_ridge = ridgeRegression(X, Y, 5)
display(Latex(r'$\hat\beta _{ridge}$ ')); display(str(beta_ridge))

#Implementing SVD on covariance matrix
```

```
U,D,V = np.linalg.svd(X)
display(Latex(r'$\lambda_i$')); display(str(np.round(np.linalg.eig(X.T.
 ↪dot(X))[0])))
display(Latex(r'$d_i^2$ ')); display(str(np.round(D**2)))

display(Latex(r"The shrinkage ratio")) ; display(str(D[0]**2 / (D[0]**2 + 5)))
display(Latex(r"The theoretical shrinkage ratio")); display(str(beta_ridge[0]/
 ↪reg.coef_[0]))

#Multiplying X1 by 2, keeping X2 and X3 unchanged
X = np.append(np.append(2*X1.reshape(100,1) , X2.reshape(100,1), axis = 1) , X3.
 ↪reshape(100,1) , axis = 1)
epsilon = norm.rvs(size = 100, random_state = 2)
beta_ridge2 = ridgeRegression(X, Y, 5)
display(Latex(r'$\hat\beta _{ridge, new}$')); display(str(beta_ridge2))
```

$\hat{\beta}_{OLS}$

```
'[0.77048776 0.77048776 0.77048776]'
```

$\hat{\beta}_{ridge}$

```
'[0.75451258 0.75451258 0.75451258]'
```

λ_i

```
'[236.  -0.   0.]'
```

d_i^2

```
'[236.   0.   0.]'
```

The shrinkage ratio

```
'0.9792661495347265'
```

The theoretical shrinkage ratio

```
'0.9792661495347379'
```

$\hat{\beta}_{ridge,new}$

```
'[0.7624165  0.38120825 0.38120825]'
```

The above results are consistent with our expectations. Also, multiplying X_1 by 2 halves the other coefficients.

2.3 (20 Points) Bitcoin Price Prediction Revisited

For this question, take the same training and testing split from the Bitcoin data, and fit a ridge regression. You can use any existing package to perform this.

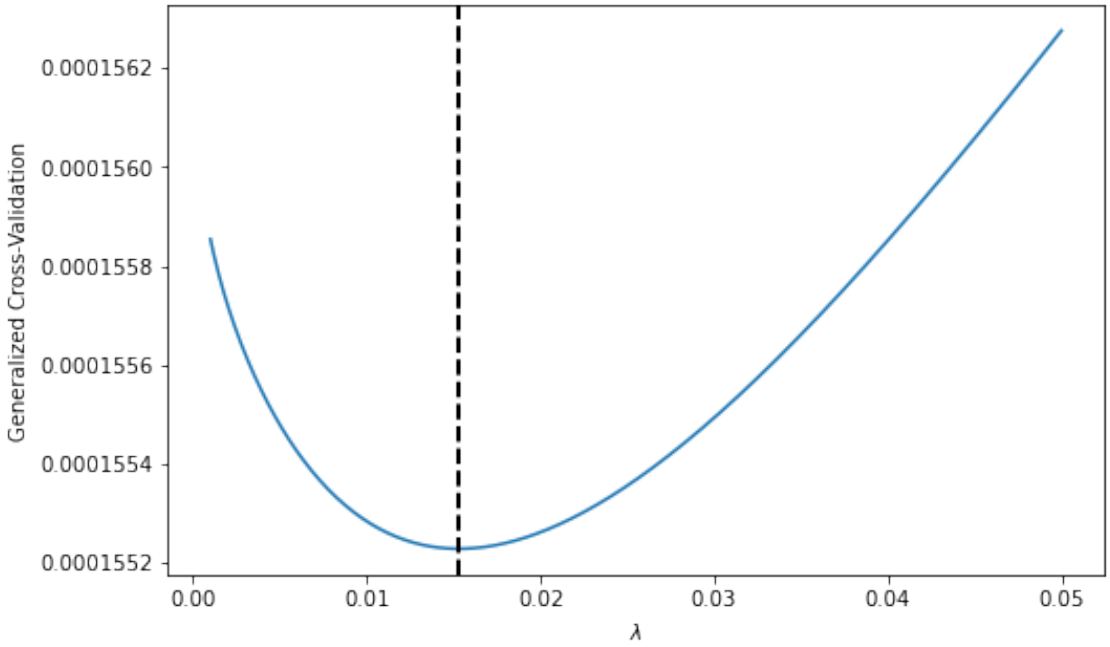
State what criteria is used to select the best λ.

State what values of λ are considered and report your best lambda value.

Compare your model fitting results with the linear model in terms of their performances on the testing data.

[3]:
```
import pandas as pd
from sklearn import metrics
from decimal import Decimal
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler

#We use 'temp_df' dataframe to store bitcoin data, forward fill takes care of␣
 ↪missing data
temp_df = pd.read_csv('bitcoin_dataset.csv', usecols = range(1,24)).
 ↪fillna(method ='ffill', axis = 0)
#Interval and prediction day
n_days, pred_day = 3 , 7

#Scaling the data
ss = StandardScaler()
norm_df = pd.DataFrame(ss.fit_transform(temp_df.values) , columns = temp_df.
 ↪columns)

#Removing outliers
outlier_id = np.where(norm_df > 10)[0]
temp_df.drop(outlier_id , inplace = True)
norm_df.drop(outlier_id , inplace = True)

#Define a new dataframe df for the preprocessed data
df = pd.DataFrame()

#Rearrage the data from each column into 3 columns
for i in range(len(temp_df.columns)):
    for j in range(n_days):
        df[i*n_days + j] = np.array(norm_df.iloc[j:-(pred_day-1)+j , i])

#Rename the columns of the new dataframe:
df.columns = [col+str(i) for col in temp_df.columns for i in range(1,n_days+1)]

#Insert the prediction price column
df.insert(0, 'output', np.array(norm_df['btc_market_price'][(pred_day - 1):]) )
```

```
dates = np.array(pd.to_datetime(pd.read_csv('bitcoin_dataset.csv')['Date'].
 ↪drop(outlier_id))[(pred_day - 1):])
df.insert(0, 'output date', dates[:len(df)])

# Defining the training and testing datasets and saving them in text files
train_df, test_df = df[df['output date'] < '2017-01-01'] , df[df['output date']␣
 ↪>= '2017-01-01']
X_train, y_train = train_df.iloc[:,2:].values , train_df.iloc[:,1].values
X_test, y_test = test_df.iloc[:,2:].values , test_df.iloc[:,1].values

#We use the RidgeCV package from scikit-learn to fit lambda
#Search for lambda in 1000 values from 1e-03 to 8e-02
lambda_range = np.linspace(1e-03, 5e-02, 500)
ridge = RidgeCV(alphas = lambda_range , fit_intercept = False, store_cv_values =␣
 ↪True)
ridge.fit(X_train, y_train)

plt.figure(figsize = (8,5))
plt.plot(lambda_range , np.mean(ridge.cv_values_, axis=0))
plt.axvline(x = ridge.alpha_ , lw = 2, ls = '--', color = 'black')
plt.ylabel('Generalized Cross-Validation')
plt.xlabel('$ \lambda $')
plt.show()

#Test Error Comparison
y_predicted = ridge.predict(X_test)
MSE_Ridge = sum((y_predicted - y_test)**2)/len(y_test)
#For Linear Regression
MSE_OLS = sum((LinearRegression().fit(X_train, y_train).predict(X_test) -␣
 ↪y_test)**2)/len(y_test)
display(Latex(r'$\lambda_{best}$')); display(np.round(ridge.alpha_,4))
display(Latex(r'$MSE_{Ridge}$')); display((MSE_Ridge))
display(Latex(r'$MSE_{OLS}$')); display((MSE_OLS))
```



λ_{best}

`0.0153`

MSE_{Ridge}

`0.14306047402357086`

MSE_{OLS}

`0.1993372102019394`

We have used Generalized cross-validation as penalty to derive the value for $\lambda_{best} = 0.0153$. This was based off of considering λ between 0.001 and 0.05. Comparing MSE for Ridge-regression versus linear regression, we can attest ridge-regression performs better.

3 Question 2 (60 Points) One Variable Lasso

Based on our development in the lecture, fitting a one variable Lasso is simply a soft-thresholding problem. Use the following code the generate a normalized data (if you use python, then do import random and call random.seed(1)):

```
set.seed(1)
 n = 100
 X = rnorm(n)
 X = X / sqrt(sum(X*X))
```

```
Y = X + rnorm(n)
```

Please be aware that instead of $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ in the lecture note, we have $\mathbf{X}^T\mathbf{X} = n\mathbf{I}$. However, the derivation in page 33, 34 and 35 from the lecture notes remains largely unchanged except some scaling issues. We will derive a new result using the following objective function for this question:

$$\arg\min_{\beta} \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

3.1 (30 Points) The Soft-Thresholding Function

Consider a model without intercept. Perform the rest:

Re-derive the $\hat{\beta}^{Lasso}$ formula in page 34 and 35 based on this one variable Lasso problem with $\mathbf{X}^T\mathbf{X} = n\mathbf{I}$.

What is the difference between this and the original one?

After you obtaining the soft-thresholding solution similar to page 35, write a function in the form of soft_th <- function(b, lambda) to calculate it, where b is the OLS estimator, and λ is the penalty level.

Apply this function to your data and obtain the Lasso solution. Report your results. Try a few different λ values to see how that would alter the solution. You do not need to tune it.

3.2 Answer:

Given objective function to be minimized:

$$\arg\min_{\beta} \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

$$= \arg\min_{\beta} \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\hat{\beta}^{OLS} + \mathbf{X}\hat{\beta}^{OLS} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

Expanding $\|\mathbf{y} - \mathbf{X}\hat{\beta}^{OLS} + \mathbf{X}\hat{\beta}^{OLS} - \mathbf{X}\beta\|^2$, we see that $\|\mathbf{y} - \mathbf{X}\hat{\beta}^{OLS}\|^2$ (cross-term = 0). The objective function, clearly, does not depend on β. Thus:

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \frac{1}{2n}\|\mathbf{X}\hat{\beta}^{OLS} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

$$= \arg\min_{\beta} \frac{1}{2n}(\hat{\beta}^{OLS} - \beta)^T\mathbf{X}^T\mathbf{X}(\hat{\beta}^{OLS} - \beta) + \lambda\|\beta\|_1$$

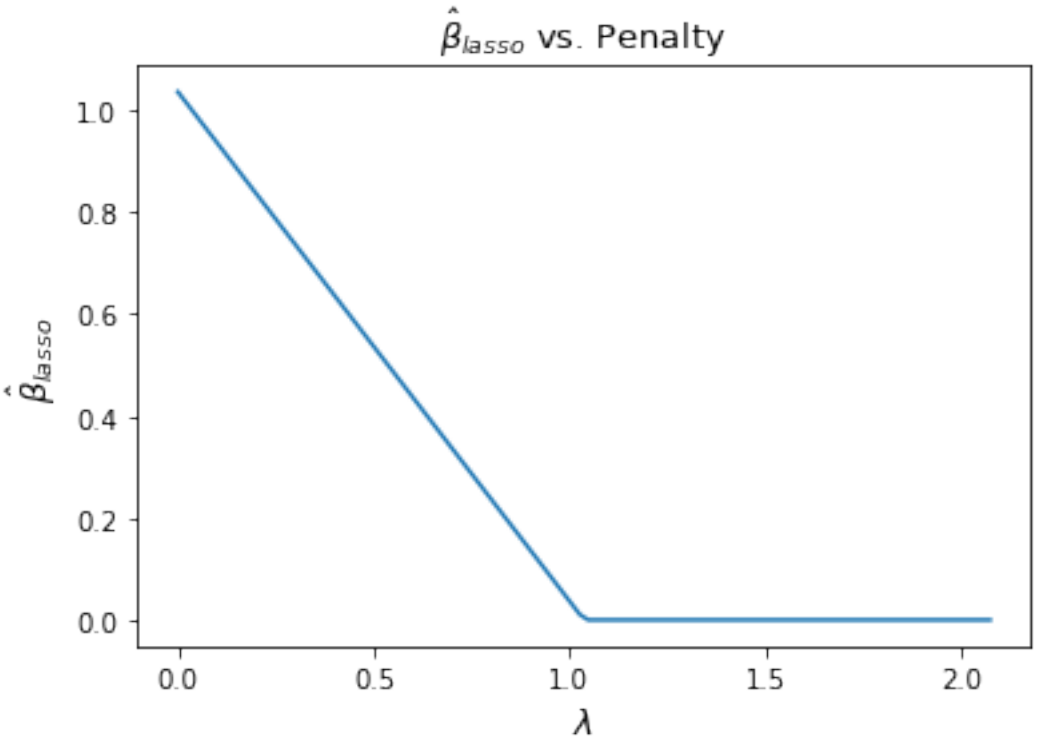
$$= \arg\min_{\beta} \frac{1}{2}\sum_{j=1}^{p}(\hat{\beta}_j^{OLS} - \beta_j)^2 + \lambda\|\beta\|_1 (\hat{\beta}_j^{lasso} = sign(\hat{\beta}_j^{OLS})(|\hat{\beta}_j^{OLS}| - \lambda)_+$$

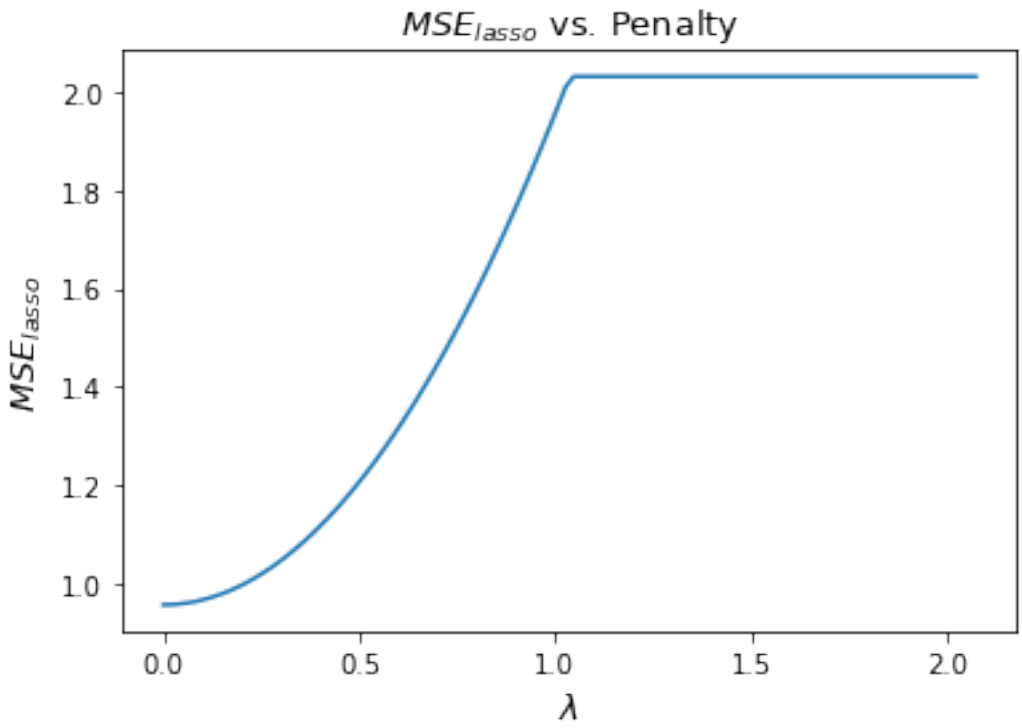
The $\frac{\lambda}{2}$ term in the original derivation has now been replaced by λ

[9]:
```
import random
random.seed(50)
```

```
X = np.array([random.gauss(0,1) for i in range(100)])
epsilon = np.array([random.gauss(0,1) for i in range(100)])
X = X / np.sqrt(sum(X**2)/100)

Y = X + epsilon

# soft thresholding function
def soft_th(beta , penalty = 1):
    '''
    Function that given beta_OLS, calculates Lasso Parameter

    Variables:
    beta (Input) :: Estimator from OLS
    penalty (Input) :: Penalty (Lambda from theory)
    beta_lasso - Estimator from lasso

    '''
    if beta > penalty:
        return beta - penalty
    elif beta < (-penalty):
        return beta + penalty
    elif abs(beta) <= penalty:
        return 0
    else:
        print("Input not in range!")
        return -9e9

#Since we have only one variable Lasso, we can summarize results in a plot as␣
 ↪follows:
beta_ols = 1/(X.T.dot(X)) * X.T.dot(Y)
lamda = np.linspace(0,abs(2*beta_ols),100)
beta_lasso = []
mse_lasso = []
for l in lamda:
    bl = soft_th(beta_ols , l)
    beta_lasso.append(bl)
    MSE = sum((X*bl - Y)**2)/len(Y)
    mse_lasso.append(MSE)

fig = plt.figure(figsize = (6,4)); ax = plt.gca()
ax.plot(lamda , beta_lasso)
ax.set_title(r'$\hat\beta_{lasso}$ vs. Penalty', fontsize=13)
ax.set_xlabel(r'$\lambda$', fontsize=13)
ax.set_ylabel(r'$\hat\beta_{lasso} $', fontsize=13)

fig = plt.figure(figsize = (6,4)); ax = plt.gca()
ax.plot(lamda , mse_lasso)
```

```
ax.set_title(r'$MSE_{lasso}$ vs. Penalty', fontsize=13)
ax.set_xlabel(r'$\lambda$', fontsize=13)
ax.set_ylabel(r'$MSE_{lasso}$', fontsize=13)


plt.show()
```





3.3 (30 Points) The Intercept, Centering and Scaling Issues

Re-generate your data based on the following code:

```
set.seed(1)
n = 100
X = rnorm(n, mean = 1, sd = 2)
Y = 1 + X + rnorm(n)
```

For this question, we will use a technique to deal with the center and scale of X based on the intuition:

$$\frac{Y - \bar{Y}}{\text{sd}_y} =amp; \sum_{j=1}^{p} \frac{X_j - \bar{X}_j}{\text{sd}_j} \gamma_j \tag{1}$$

$$Y =amp; \underbrace{\bar{Y} - \sum_{j=1}^{p} \bar{X}_j \frac{\text{sd}_y \cdot \gamma_j}{\text{sd}_j}}_{\beta_0} + \sum_{j=1}^{p} X_j \underbrace{\frac{\text{sd}_y \cdot \gamma_j}{\text{sd}_j}}_{\beta_j}, \tag{2}$$

A common practice when dealing with the intercept and scaling is to perform the following:

Center and scale both **X** (column-wise) and **y** and denote the processed data as $\frac{Y-\bar{Y}}{\text{sd}_y}$ and $\frac{X_j-\bar{X}_j}{\text{sd}_j}$ in the above formula.

Fit a linear regression (or Lasso) using the processed data based on the no-intercept model, and obtain the parameter estimates γ_j. In our case, there is only one j, i.e. $p = 1$. Recover the original parameters β_0 and β_j's.

Understand and implement this procedure to our one-variable Lasso problem and obtain the Lasso solution on the original scale. You must write your own code and use the soft_th() function previously defined. For the choice of λ, use a value that does not give you a trivial solution. A trivial solution is $\beta^{Lasso} = 0$ or β^{OLS}.

[12]:
```
random.seed(50)

X = np.array([random.gauss(1,2) for i in range(100)])
Y = 1 + X + np.array([random.gauss(0,1) for i in range(100)])
mean_x , sd_x = np.mean(X) , np.std(X)
mean_y , sd_y = np.mean(Y) , np.std(Y)
X_centered = (X - mean_x) / sd_x
Y_centered = (Y - mean_y) / sd_y

beta_ols = 1/(X_centered.T.dot(X_centered)) * X_centered.T.dot(Y_centered)
penalty = 0.01; beta_lasso = soft_th(beta_ols , penalty)
display(Latex(r'$\hat\beta^{OLS}_{best, centered}$')); display(np.
 ↪round(beta_ols,2))
display(Latex(r'$\hat\beta^{Lasso}_{\lambda = 0.01, centered}$')); display(np.
 ↪round(beta_lasso,2))

#Recovering original parameters
beta_0 = mean_y - mean_x * (sd_y/sd_x)*beta_lasso
beta_1 = (sd_y/sd_x)*beta_lasso
display(Latex(r'Original parameters of regression'));
display(Latex(r'$\hat\beta_{0, original}$')); display(np.round(beta_0,2))
display(Latex(r'$\hat\beta_{1, original}$')); display(np.round(beta_1,2))
```

$\hat{\beta}^{OLS}_{best,centered}$

0.91

$\hat{\beta}^{Lasso}_{\lambda=0.01,centered}$

0.9

Original parameters of regression

$\hat{\beta}_{0,original}$

1.16

$\hat{\beta}_{1,original}$

1.01