Homework 1, CPSC 8420, Spring 2022

Zhiyuan, Song

March 2, 2022

Solution to Problem 1

1. **Showing the Equivalence of two Ridge Regression Solutions:** One can take SVD for an arbitary $\mathbf{A} \in \mathbb{R}^{n\times p}$, $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}$. $\mathbf{U} \in \mathbb{R}^{n\times n}$ and $\mathbf{V} \in \mathbb{R}^{p\times p}$ are complex unitary matrices. $.\Sigma \in \mathbb{R}^{n\times p}$ is an rectangular diagonal matrix with non-negative real numbers on the diagonal. Concequently,

$$\mathbf{A}^T = \mathbf{V}\Sigma\mathbf{U}^T$$
$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\Sigma\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{V}\Sigma^2\mathbf{V}^T$$
$$\because \lambda\mathbf{I} = \lambda\mathbf{V}\mathbf{V}^T = \mathbf{V}\lambda\mathbf{V}^T = \mathbf{V}\lambda\mathbf{I}\mathbf{V}^T$$
$$\therefore \mathbf{A}^T\mathbf{A} + \lambda\mathbf{I} = \mathbf{V}(\Sigma^2 + \mathbf{I})\mathbf{V}^T$$

$$\therefore (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T = \mathbf{V}\Sigma(\Sigma^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^T \tag{1}$$

On the other hand,

$$\mathbf{A}\mathbf{A}^T = \mathbf{U}\Sigma^2\mathbf{U}^T$$
$$\because \lambda\mathbf{I} = \lambda\mathbf{U}\mathbf{U}^T = \mathbf{U}\lambda\mathbf{U}^T = \mathbf{U}\lambda\mathbf{I}\mathbf{U}^T$$
$$\therefore \mathbf{A}\mathbf{A}^T + \lambda\mathbf{I} = \mathbf{U}(\Sigma^2 + \lambda\mathbf{I})\mathbf{U}$$

$$\therefore \mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I})^{-1} = \mathbf{V}\Sigma(\Sigma^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^T \tag{2}$$

We can observe that eq2 is equivalent to eq1.

2. **Time Consumption for Different p of A:** Matrix $\mathbf{A} \in \mathbb{R}^{100\times p}$ with random elements was introduced for a certein p. β were computed using two equivalent solutions of ridge regression and the time for each execuation was recorded, as shown in Figure 1.

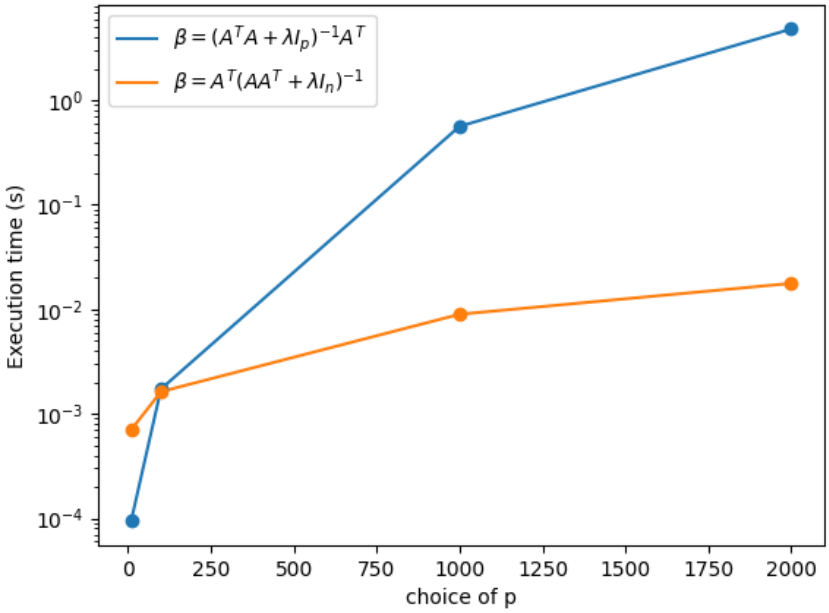

Figure 1: Time consumption for various p of $\mathbf{A}$.

Solution to Problem 2

1. **Error Equation Dervation:**

$$
\begin{aligned}
Err(x_0) &= E\left[(Y - \hat{f}(x_0))^2 | X = x_0\right] \\
&= E\left[(Y - f(x_0) + f(x_0 - \hat{f}(x_0)))^2\right] \\
&= E\left[(Y - f(x_0))^2 + (f(x_0) - \hat{f}(x_0))^2 + 2(Y - f(x_0))(f(x_0) - \hat{f}(x_0))\right] \\
&= \sigma^2 + E\left[(f(x_0) + E[\hat{f}(x_0)\right] - E\left[\hat{f}(x_0)] - \hat{f}(x_0))^2\right] + 2E\left[(Y - f(x_0))(f(x_0) - \hat{f}(x_0))\right]
\end{aligned}
$$

Where $2E[(Y - f(x_0))(f(x_0) - \hat{f}(x_0)) = 0$ because $E(\epsilon) = 0$. Meanwhile, $E[(Y - f(x_0))^2] = E[\epsilon^2] = Var(\epsilon) + E^2[\epsilon] = \sigma^2$. Therefore,

$$
\begin{aligned}
Err(x_0) = \sigma^2 &+ E\left[(f(x_0) - E[\hat{f}(x_0)])^2\right] + E\left[(\hat{f}(x_0) - \hat{f}(x_0))^2\right] \\
&+2E\left[(f(x_0) - E[\hat{f}(x_0)])(E[\hat{f}(x_0)] - \hat{f}(x_0))\right] \\
= \sigma^2 &+ E\left[(f(x_0) - E[\hat{f}(x_0)])^2\right] + E\left[(\hat{f}(x_0) - \hat{f}(x_0))^2\right] \\
&+2(f(x_0) - E[\hat{f}(x_0)])E\left[(E[\hat{f}(x_0)] - \hat{f}(x_0))\right]
\end{aligned}
$$

Because $E\left[E[\hat{f}(x_0)] - \hat{f}(x_0)\right] = 0$, leading to 0 cross term. Consequently,

$$\begin{aligned} Err(x_0) &= \sigma^2 + E\left[(f(x_0) - E[\hat{f}(x_0)])^2\right] + E\left[(E[\hat{f}(x_0)] - \hat{f}(x_0))^2\right] \\ &= \sigma^2 + (f(x_0) - E[\hat{f}(x_0)])^2 + E\left[(E[\hat{f}(x_0)] - \hat{f}(x_0))^2\right] \\ &= \sigma^2 + Bias(\hat{f}(x_0))^2 + Var(\hat{f}(x_0)) \end{aligned}$$

In the case of k-NN, the expectation term, $E[\hat{f}(x_0)]$, can be evalueated

$$\begin{aligned} E[\hat{f}(x_0)] &= E\left[\frac{1}{k}\sum_{l=1}^{k} Y_l\right] \\ &= E\left[\frac{1}{k}\sum_{l=1}^{l}(f(x_l) + \epsilon_l)\right] \\ &= \frac{1}{k}\sum_{l=1}^{k}(f(x_l) + \frac{1}{k}\sum_{l=1}^{k} E[\epsilon_l]) \\ &= \frac{1}{k}\sum_{l=1}^{k}(f(x_l) \end{aligned}$$

Next, we can simplify the bias term using the evaluation of the expectation.

$$\begin{aligned} Bias^2(x_0) &= (f(x_0) - E[\hat{f}(x_0)])^2 \\ &= \left(f(x_0) - \frac{1}{k}\sum_{l=1}^{k} f(x_l)\right)^2 \end{aligned}$$

The variance of k-NN can be simplified

$$\begin{aligned} Var(\hat{f}(x_0)) &= E\left[\left(\hat{f}(x_0) - E[\hat{f}(x_0)]\right)^2\right] \\ &= E\left[\left(\frac{1}{k}\sum_{l=1}^{k} Y_l - \frac{1}{k}\sum_{l=1}^{k} f(x_l)\right)\right] \\ &= E\left[\left(\frac{1}{k}\sum_{l=1}^{k}(f(x_l) + \epsilon_l) - \frac{1}{k}\sum_{l=1}^{k} f(x_l)\right)\right] \\ &= \frac{1}{k^2}E\left[\left(\sum_{l=1}^{k}\epsilon_l\right)^2\right] \\ &= \frac{1}{k^2}Var\left(\sum_{l=1}^{k}\epsilon_l\right) \end{aligned}$$

Because $Var\left(\sum_{l=1}^{k}\epsilon_l\right) = E\left[\left(\sum_{l=1}^{k}\epsilon_l - E\left[\sum_{l=1}^{k}\epsilon_l\right]\right)^2\right]$ and $E\left[\sum_{l=1}^{k}\epsilon_l\right] = \sum_{l=1}^{k} E[\epsilon_l] = 0$.

Accordingly, we can obtain the variance term

$$\begin{aligned} Var(\hat{f}(x_0)) &= \frac{1}{k^2}\sum_{l=1}^{k} Var(\epsilon_l) \\ &= \frac{k\sigma^2}{k^2} = \frac{\sigma^2}{k} \end{aligned}$$

As a result, we can combine Bias term and Variance term together, one can obtain error formula in the case of k-NN

$$Err(x_0) = \sigma^2 + [f(x_0) - \frac{1}{k}\sum_{l=1}^{k} f(x_l)]^2 + \frac{\sigma^2}{k}$$

2. **Justification of Bias and Variance change when k increases:** The number of neighbors k is inversely related to the model complexity. For small k, the estimate $\hat{f}_k(x)$ can potentially adapt itself better to the underlying $f(x)$. When k increases, the the squared difference between $f(x_0)$ and the average of $f(x)$ at the k-nearest neighbors (i.e., the bias) will typically increase and the mean of the data set includes less and less information. While the variance will decrease according to its role in the variance term.

Solution to Problem 3

For vanilla linear regression model:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{LS} &= \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 \\ \nabla\|\mathbf{A}\boldsymbol{\beta} - \mathbf{y}\|_2^2 &= 2\mathbf{A}^T(\mathbf{A}\boldsymbol{\beta} - \mathbf{y}) \end{aligned}$$

Let $\nabla\|\mathbf{A}\boldsymbol{\beta} - \mathbf{y}\|_2^2 = 0$, one can obtain

$$\mathbf{A}^T\mathbf{A}\boldsymbol{\beta} = \mathbf{A}^T\mathbf{y}$$

Therefore,

$$\hat{\boldsymbol{\beta}}_{LS} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} = \mathbf{A}^T\mathbf{y} \tag{3}$$

For ridge regression model:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{\lambda}^{Ridge} &= \min_{\boldsymbol{\beta}} \|\mathbf{A}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \\ \nabla(\|\mathbf{A}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \lambda\boldsymbol{\beta}) &= 2\mathbf{A}^T(\mathbf{A}\boldsymbol{\beta} - \mathbf{y}) + 2\lambda\boldsymbol{\beta} \end{aligned}$$

Applying same method, one can get

$$\mathbf{A}^T(\mathbf{A}\boldsymbol{\beta} - \mathbf{y}) + \lambda\boldsymbol{\beta} = 0$$

Accordingly,

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Ridge} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T = \frac{A^T\mathbf{y}}{1+\lambda} = \frac{\hat{\boldsymbol{\beta}}_{LS}}{1+\lambda} \tag{4}$$

For Lasso model:

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso} = \min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$$

We know that, if function f and g are both concex functions, $f + g$ must be convex. Hence, Lasso model must have a globla minimium theoretically. As for function, $f(x) = |x|$, it is not defferatiable at $x = 0$. However, if sub-defferentials is introduced, one can get modified differentials for $f(x) = |x|$ at $x = 0$, $\delta f(x) = \text{sign}(x)$. $\text{sign}(x)$ is defined as,

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{5}$$

With the complement of the defferential at $x = 0$, we can take derivative for Lasso model to obatin the minimization by making the defferential term equal to 0.

$$\nabla \frac{1}{2}\|\mathbf{A}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \delta\lambda\|\boldsymbol{\beta}\|_1 = 0$$
$$\therefore \mathbf{A}^T(\mathbf{A}\boldsymbol{\beta} - \mathbf{y}) + \lambda \text{ sign}(\boldsymbol{\beta}) = 0$$

Note that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$, we can obtain

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso} = \mathbf{A}^T\mathbf{y} - \lambda \text{ sign}(\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso})$$

we can notice that $\mathbf{A}^T\mathbf{y}$ is the solution of $\hat{\boldsymbol{\beta}}_{LS}$.Accordingly, we can keep writing the expression

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso} = \begin{cases} \hat{\boldsymbol{\beta}}_{LS} + \lambda & \text{if } \hat{\boldsymbol{\beta}}_{LS} < -\lambda \\ 0 & \text{if } |\hat{\boldsymbol{\beta}}_{LS}| \leq \lambda \\ \hat{\boldsymbol{\beta}}_{LS} - \lambda & \text{if } \hat{\boldsymbol{\beta}}_{LS} > \lambda \end{cases} = \text{sign}(\hat{\boldsymbol{\beta}}_{LS})(|\hat{\boldsymbol{\beta}}_{LS}| - \lambda)_+ \tag{6}$$

The equation above is recognized as a soft-thresholding function defined as

$$\mathbb{S}_{\lambda}(x) = \begin{cases} x + \lambda & \text{if } x < -\lambda \\ 0 & \text{if } |x| \leq \lambda \\ x - \lambda & \text{if } x > \lambda \end{cases}$$

Therefore, we can denote the solution for Lasso model as

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Lasso} = \mathbb{S}_{\lambda}(\hat{\boldsymbol{\beta}}_{LS}) \tag{7}$$

For subset selection model: We can solve the subset model using

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_0$$

It is equivalent to solve non-convex problem

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 \text{ subject to } \|\boldsymbol{\beta}\|_0 \leq k$$

Where the l_0 norm of a vector $\boldsymbol{\beta}$ counts the number of non zero results in $\boldsymbol{\beta}$ and is given by $\|\boldsymbol{\beta}\|_0 = \sum_{i=1}^{p} 1(\boldsymbol{\beta} \neq \mathbf{0})$. According to the definition of subset selection model, the solution can be expressed

$$\hat{\boldsymbol{\beta}}_{\lambda}^{Subset} = \hat{\boldsymbol{\beta}}_{LS} I(|\hat{\boldsymbol{\beta}}_{LS}| \geq |\hat{\boldsymbol{\beta}}_{(M)}|) \tag{8}$$

For those coefficients less than the M^{th} largest coefficient will be set to 0.

Solution to Problem 4

1. **Loading and shuffling data from house.data** The code starts from importing useful global packages to perform array calculations, standardization and linear regressions. The functions of linear regression, ridge regression and mean of square error were defined beforehand. Data shuffling was performed and the seed (seed = 2023) was provided for checking. Data splitting and standardization will be taken in the coming sub problems.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.stats as stats
import sklearn.linear_model as sl
from sklearn.metrics import mean_squared_error
fontsize_label = 18
fontsize_ticks = 15
#define functions for linear regression
def Vanilla_y_hat(A,y):
    #(A^T*A)^{-1}*A^T*y
    result = np.linalg.pinv((A.T).dot(A)).dot(A.T)
    beta = result.dot(y)
    return beta
def Ridge_y_hat(A,y,lam):
    #(A^T*A+lambda*I)^{-1}*A^T*y
    iden_part = lam*np.identity(len(A.T))
    iden_part[0,0] = 1
    result = np.linalg.pinv((A.T).dot(A)+iden_part).dot(A.T)
    beta = result.dot(y)
    return beta

def MSE(y,beta,x):
    #(y-y_hat)^T*(y-y_hat)
    y_hat = x.dot(beta)
    return ((y-y_hat).T).dot(y-y_hat)[0][0]/len(y)
#import data
data = np.loadtxt('housing.data')
#remove any possible ordering fx
seed = 2023; np.random.seed(seed)
randperm = np.random.randint(0,len(data),len(data))
data = data[randperm,:]
```

2. **Mean squared error as a function of training set** Data was standardized using zscore after splitting for each training set and test set. Linear regression, which maximize least square error, were performed for training set. The result of MSE calculations between test set of output and predicted output using trained coefficients was demonstrated in Figure 2. Herein, the linear regression was defined as a function using matrix computations and examined by build-in python package, sklearn. We can see results from both methods coincided in the case of either training or test, meaning the same predictions they were able to get and validating the self-built function. As more training data was included, the MSE of the training MSE grew slower due to better fittings but more errors introduced. Whereas the predictions became more accurate and the test set size was reduced, leading to a decrease of MSE for the set.

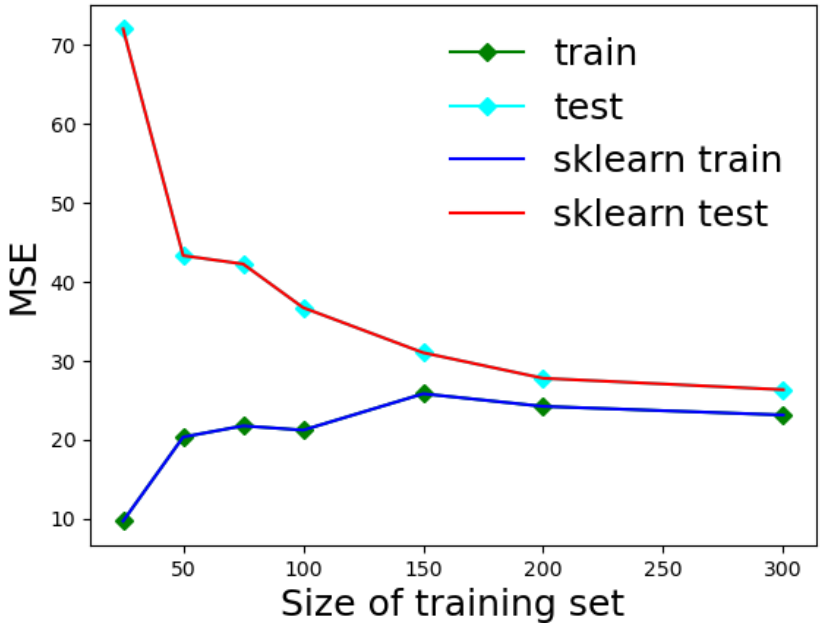

Figure 2: MSE for training and test data as a function of training set size

The two curves will eventually meet because the training curve is typically increasing while test MSE is decreasing.

```
n = [25,50,75,100,150,200,300]
MSE_test = []
MSE_train = []
MSE_SL_train = []
MSE_SL_test = []
for Ntrain in n:
    Xtrain = stats.zscore(data[:Ntrain,:-1],axis=0)
    ytrain = data[:Ntrain,-1][...,None]
    Xtest = stats.zscore(data[Ntrain:,:-1],axis=0)
    ytest = data[Ntrain:,-1][...,None]
    Xtrain_mt = np.concatenate((np.ones((len(Xtrain),1)),Xtrain),axis=1)
    Xtest_mt = np.concatenate((np.ones((len(Xtest),1)),Xtest),axis=1)
    reg = sl.LinearRegression().fit(Xtrain, ytrain)
    beta = Vanilla_y_hat(Xtrain_mt, ytrain)
    MSE_train.append(MSE(ytrain,beta,Xtrain_mt))
    MSE_test.append(MSE(ytest,beta,Xtest_mt))
    MSE_SL_train.append(mean_squared_error(ytrain,reg.predict(Xtrain)))
    MSE_SL_test.append(mean_squared_error(ytest,reg.predict(Xtest)))
plt.plot(n,MSE_train,color='green',label='train',marker='D')
plt.plot(n,MSE_test,color='cyan',label='test',marker='D')
plt.plot(n,MSE_SL_train,color='blue',label='sklearn train')
plt.plot(n,MSE_SL_test,color='red',label='sklearn test')
plt.legend(frameon=False,fontsize=fontsize_label)
```

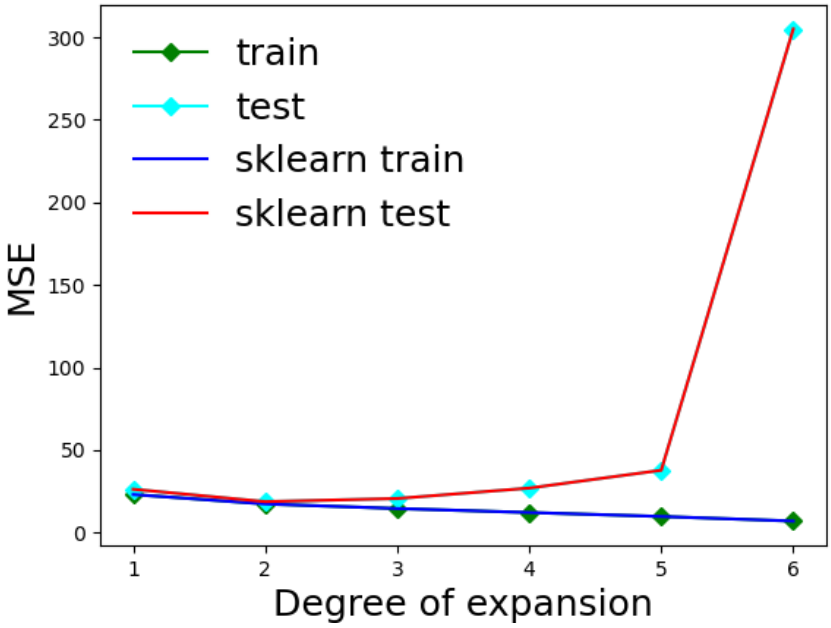

Figure 3: MSE for training and test data as a function of expansion degree

```
plt.ylabel('MSE',fontsize=fontsize_label)
plt.xlabel('Size of training set',fontsize=fontsize_label)
plt.savefig('4_2_fig.png')
```

3. **MSE vs Degree expansion** Similar treatments and global packages were applied before the degree expansion. The feature set was expanded every time when the order of magnitude of the original features increases. The expanded features were standardized before they incorporated with the data previously treated. Because there is a feature of house prising containing either 0 or 1, the degree extension of the data set results in the same data set, causing an invertable $\mathbf{A}^A A$. Hence, we need to apply pseudoinverse for the inversion of $\mathbf{A}^A A$ (pinv() instead of inv() used in the code). The result of self-built function was examined by calling python sklearn package, as shown in Figure 3. And we can observe a slight decrease at degree 2 for test set, indicating the degree extension of 2 can offer the best prediction in terms of MSE. While, MSE of training set kept reducing as higher degree features were involved, which resulted in an overfitting. Hence, such an overfitting caused the steep increase of the test set MSE as the degree increased.

```
Ntrain = 300
MSE_test = []
MSE_train = []
MSE_SL_test = []
MSE_SL_train = []
Xtrain = stats.zscore(data[:Ntrain,:-1],axis=0)
ytrain = data[:Ntrain,-1][...,None]
Xtrain_mt = np.concatenate((np.ones((len(Xtrain),1)),Xtrain),axis=1)
Xtest = stats.zscore(data[Ntrain:,:-1],axis=0)
```

```
ytest = data[Ntrain:,-1][...,None]
Xtest_mt = np.concatenate((np.ones((len(Xtest),1)),Xtest),axis=1)
reg = sl.LinearRegression().fit(Xtrain,ytrain)
beta = Vanilla_y_hat(Xtrain_mt, ytrain)
MSE_train.append(MSE(ytrain,beta,Xtrain_mt))
MSE_test.append(MSE(ytest,beta,Xtest_mt))
MSE_SL_train.append(mean_squared_error(ytrain, reg.predict(Xtrain)))
MSE_SL_test.append(mean_squared_error(ytest,reg.predict(Xtest)))
degree = [2,3,4,5,6]
features = data[:,:-1]
for deg in degree:
    features = np.concatenate((features,data[:,:-1]**deg),axis=1)
    Xtrain = stats.zscore(features[:Ntrain,:],axis=0)
    ytrain = data[:Ntrain,-1][...,None]
    Xtrain_mt = np.concatenate((np.ones((len(Xtrain),1)),Xtrain),axis=1)
    Xtest = stats.zscore(features[Ntrain:,:],axis=0)
    ytest = data[Ntrain:,-1][...,None]
    Xtest_mt = np.concatenate((np.ones((len(Xtest),1)),Xtest),axis=1)
    reg = sl.LinearRegression().fit(Xtrain,ytrain)
    beta = Vanilla_y_hat(Xtrain_mt, ytrain)
    MSE_train.append(MSE(ytrain,beta,Xtrain_mt))
    MSE_test.append(MSE(ytest,beta,Xtest_mt))
    MSE_SL_train.append(mean_squared_error(ytrain,reg.predict(Xtrain)))
    MSE_SL_test.append(mean_squared_error(ytest,reg.predict(Xtest)))
plt.plot([1]+degree,MSE_train,color='green',label='train',marker='D')
plt.plot([1]+degree,MSE_test,color='cyan',label='test',marker='D')
plt.plot([1]+degree,MSE_SL_train,color='blue',label='sklearn train')
plt.plot([1]+degree,MSE_SL_test,color='red',label='sklearn test')
plt.legend(frameon=False,fontsize=fontsize_label)
plt.ylabel('MSE',fontsize=fontsize_label)
plt.xlabel('Degree of expansion',fontsize=fontsize_label)
plt.savefig('4_3_fig.png')
```

4. **MSE vs Ridge Regression** The solution of ridge regression will approach that of the conventional regression if λ is extremely small. Therefore, we should observe a similar big gap between training and test curves with 6-degree extension (Figure 3) when λ is significantly small. The result of self-built function was examined by calling python sklearn package, as shown in Figure 4. Pseudo-inverse was also applied here as a results of degree extension. Besides, the offset was considered due to the standardized input but the original output, leading to the identity of $p+1$ dimension in the ridge regression. However, the intercept term keeps constant. Therefore, the λ was only allowed to multiply the remaining $p \times p$ identity and leave the first term as shown in the code. As λ increases, it starts to regulate the variance and force the minimization involving the $\|\boldsymbol{\beta}\|_2^2$ term, avoiding overfitting and leading to a drop of test MSE. However, some significant coefficients will become very small if λ becomes too large, causing an increase of the bais. Accordingly, it will lead to an increase of MSE. AS for training set, overfitting problem will be regulated but some coefficients' decreases lead to larger variance, causing the increase of its MSE.

```
Ntrain = 300
MSE_test = []
MSE_train = []
MSE_SL_test = []
```

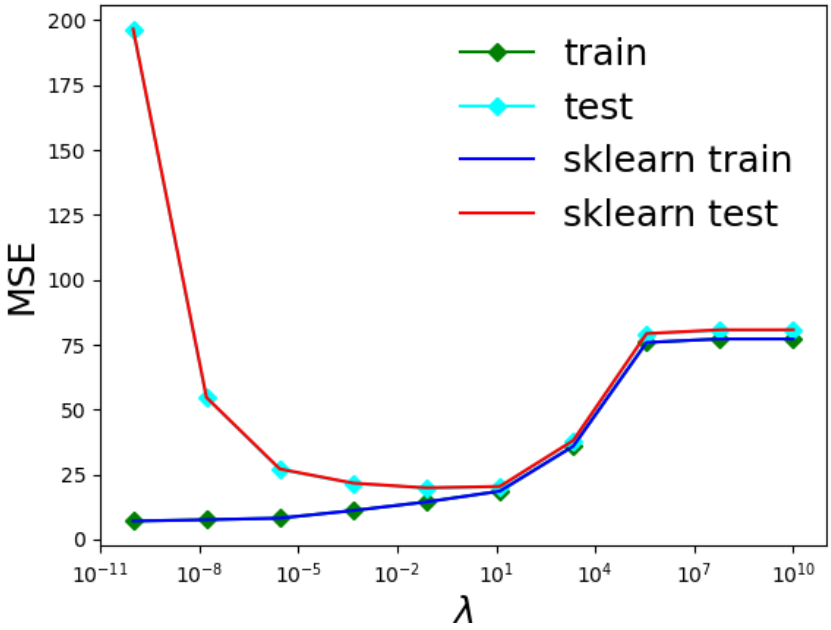

Figure 4: MSE for training and test data as a function of λ in ridge regression

```
MSE_SL_train = []
lamb = np.logspace(-10,10,10)
degree = [2,3,4,5,6]
features = data[:,:-1]
for deg in degree:
    features = np.concatenate((features,data[:,:-1]**deg),axis=1)
Xtrain = stats.zscore(features[:Ntrain,:],axis=0)
ytrain = data[:Ntrain,-1][...,None]
Xtest = stats.zscore(features[Ntrain:,:],axis=0)
ytest = data[Ntrain:,-1][...,None]
Xtrain_mt = np.concatenate((np.ones((len(Xtrain),1)),Xtrain),axis=1)
Xtest_mt = np.concatenate((np.ones((len(Xtest),1)),Xtest),axis=1)
for lam in lamb:
    beta = Ridge_y_hat(Xtrain_mt, ytrain,lam)
    reg = sl.Ridge(lam).fit(Xtrain,ytrain)
    MSE_train.append(MSE(ytrain,beta,Xtrain_mt))
    MSE_test.append(MSE(ytest,beta,Xtest_mt))
    MSE_SL_train.append(mean_squared_error(ytrain, reg.predict(Xtrain)))
    MSE_SL_test.append(mean_squared_error(ytest,reg.predict(Xtest)))
plt.plot(lamb,MSE_train,color='green',label='train',marker='D')
plt.plot(lamb,MSE_test,color='cyan',label='test',marker='D')
plt.plot(lamb,MSE_SL_train,color='b',label='train')
plt.plot(lamb,MSE_SL_test,color='r',label='test')
plt.xscale('log')
plt.legend(frameon=False,fontsize=fontsize_label)
```

```
plt.ylabel('MSE',fontsize=fontsize_label)
plt.xlabel(r'$\lambda$',fontsize=fontsize_label)
plt.savefig('4_4_fig.png')
```

5. **coefficients vs λ in Lasso regression** To show a better variation for coefficients, the range for λ was set from 10^{-4} to 10^{2}. The solution of the ridge regression will approach the result of the conventional regression if λ is significantly small. However, some coefficients become zero as the λ increases, indicating the corresponding features hardly contribute to the output. All coefficients will reach zero if λ increases. It l results in a trivial solution finally that contains nothing but the intercept, $\boldsymbol{\beta}_0$, as shown in Figure 5.

```
Ntrain = 300
lamb = np.logspace(-4,2)
Xtrain = stats.zscore(data[:Ntrain,:-1],axis=0)
ytrain = data[:Ntrain,-1][...,None]
beta_set = []
ax = plt.axes()
for lam in lamb:
    reg = sl.Lasso(alpha=lam).fit(Xtrain,ytrain)
    beta_set.append(reg.coef_)
beta_set = np.array(beta_set).transpose()
colors = matplotlib.cm.jet(np.linspace(0,1,len(beta_set)))
for i in range(len(beta_set)):
    plt.plot(lamb,beta_set[i],label=r'$\beta$'+str(i+1),color=colors[i])
ax.spines["right"].set_visible(False)
ax.spines["top"].set_visible(False)
plt.title(r'$\beta$0 = '+str('%.3f'%(reg.intercept_[0])))
plt.xscale('log')
plt.legend(frameon=False, bbox_to_anchor=(0.585, 0.5, 0.5, 0.5))
plt.ylabel(r'$\beta$',fontsize=fontsize_label)
plt.xlabel(r'$\lambda$',fontsize=fontsize_label)
plt.savefig('4_5_fig.png')
```

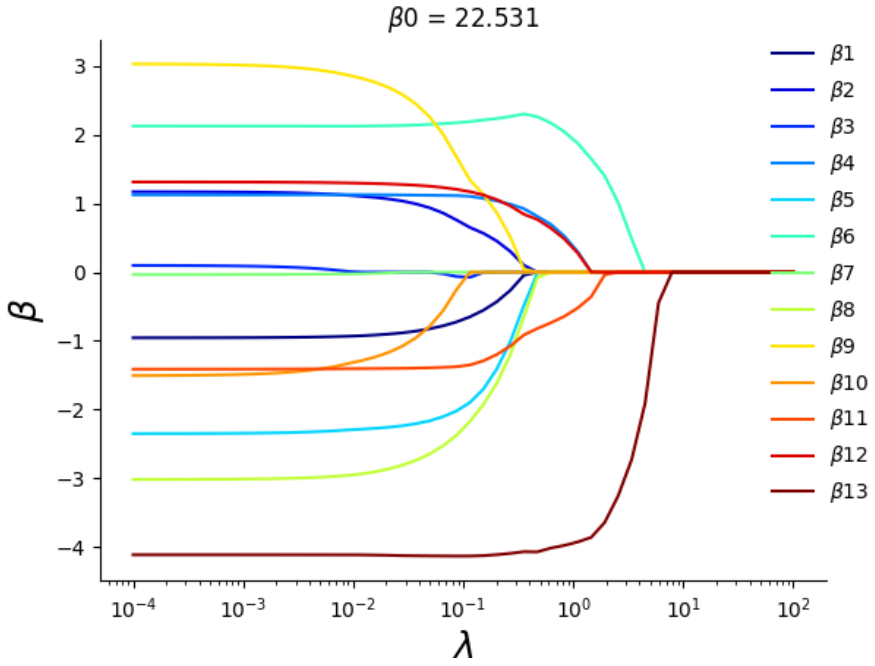

Figure 5: $\boldsymbol{\beta}$ as a function of λ in Lasso regression