Machine Learning Homework 4

Ian Sinclair
ENCE 3631-1

March 29, 2022
Dr. Zhihui Zhu

Problem 1:

In this problem we will compare the performance of traditional least squares, ridge regression, and the LASSO on a real-world dataset.
For all parts of the problem below, I would like you to submit your code.

For the following problems consider the use of the Boston House Prices data set, which is partitioned into training and testing sets. Additionally, each set is then standardized.

ttfamily

```
boston = load_boston()
X = boston.data
y = boston.target

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.208,random_state=0)

X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)
```

Now regression models for least square, ridge, and LASSO can be constructed.

a) First, I would like you to evaluate the performance of least squares. You should implement this yourself using the equation we derived in class; see slide 22 of Lecture note 13. Report the performance of your algorithm in terms of mean-squared error (MSE) on the test set, i.e.,

$$\frac{1}{n_{test}}||y_{test} - A_{test}\hat{\theta}||_2^2$$

where $A_{test} = [1 \quad X_{test}]$ as in the lecture notes.

We consider construct a matrix A_{train} to build the regression model, such that, $A_{train} = [1 \quad X_{train}]$, by the following code.

ttfamily

```
n,m = X_train.shape
A_train = np.hstack((np.ones((n,1)),X_train))
```

Then the optimal $\hat{\theta}$ is computed by,

$$\hat{\theta} = (A^T A)^{-1} A^T y_{train},$$

Finally, the final accuracy of the regression is determined by the mean-square error, $\frac{1}{n_{test}}||y_{test} - A_{test}\hat{\theta}||_2^2$,

ttfamily

```
#Training Least square regression,
n,m = X_train.shape
A_train = np.hstack((np.ones((n,1)),X_train))

theta = np.linalg.inv(np.dot(A_train.T,A_train)).dot(A_train.T).dot(y_train)

#Mean-Square Test Error
n_test,m_test = X_test.shape
A_test = np.hstack((np.ones((n_test,1)),X_test))
MSE = (np.linalg.norm(y_test - np.dot(A_test,theta), ord=2)**2)/n_test
print(MSE)
```

Which results in the follow mean-least square error.

Mean-Square Error
34.59943

b) Next, using the formula derived in class (see slides 25 and 26), implement your own version of ridge regression. You will need to set the free parameter λ. You should do this using the training data in whatever manner you like (e.g., via a holdout set)—but you should not allow the testing dataset to influence your choice of λ. Report the value of λ selected and the performance of your algorithm in terms of mean-squared error on the test set.

Ridge regression is similar to least squares except the algorithm is penalized for using to many features. And so, we consider the expression,

$$\hat{\theta} = \arg\min_{\theta} ||y - A\theta||_2^2 + ||\Gamma\theta||_2^2,$$

where Γ is a weighting matrix. Specifically, for a particular λ define, Γ,

$$\Gamma = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda} & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{\lambda} \end{bmatrix}$$

From here is it possible to define a curve θ under a a particular λ, by

$$\theta = (A^TA + \Gamma^T\Gamma)^{-1}Ay,$$

Now, to fit the best λ we consider a range $\lambda \in \{2^{-10}, 2^{-9}, 2^{-8}, ..., 2^{30}\}$, which is done by fitting θ to a partition of the training set, and then finding the minimum MSE using a holdout set.

ttfamily

```
#Getting biased by small values of lambda
#Getting biased by small values of lambda
X_fit,X_holdout,y_fit,y_holdout = train_test_split(X_train,y_train,test_size=0.308,random_state=0)

MSE_min = float('inf')
#Assign a curve, theta, from a particular lambda, and test with holdout set
n,m = X_fit.shape
n_holdout,m_holdout = X_holdout.shape
A_holdout = np.hstack((np.ones((n_holdout,1)),X_holdout))
A_fit = np.hstack((np.ones((n,1)),X_fit))
for q in range(1,31,1):
    lamda = 2**q
    Gamma = ((lamda**0.5)*np.identity(m+1))
    Gamma[0][0] = 0
    theta = np.linalg.inv(np.dot(A_fit.T,A_fit) + (np.dot(Gamma.T,Gamma))).dot(A_fit.T).dot(y_fit)

    MSE = (np.linalg.norm(y_holdout - np.dot(A_holdout,theta), ord=2)**2)/n_holdout
    if(MSE < MSE_min) and (np.linalg.norm(np.dot(A_holdout,theta))<10):
        q_min = q
        MSE_min = MSE
        theta_optimal = theta

#Mean-Square Test Error
n_test,m_test = X_test.shape
```

```
A_test = np.hstack((np.ones((n_test,1)),X_test))
MSE = (np.linalg.norm(y_test - np.dot(A_test,theta_optimal), ord=2)**2)/(n_test)
print(MSE)
```

Which results in,

λ	Mean Square Error
4	35.1249

c) Finally, I would like you to evaluate the performance of the LASSO.

```
ttfamily
"""
LASSO: Problem 1 part C
"""

from sklearn import linear_model

X_fit,X_holdout,y_fit,y_holdout = train_test_split(X_train,y_train,test_size=0.308,random_state=0)

lamda = 1
MSE_min = float('inf')
for q in range(-10,21,1):
    reg = linear_model.Lasso(alpha = 2*q)
    reg.fit(X_fit,y_fit)
    B = reg.predict(X_holdout)
    n_holdout = B.size
    MSE = np.linalg.norm(B,ord=2)**2/n_holdout
    if(MSE < MSE_min):
        MSE_min = MSE
        lamda = 2*q
reg = linear_model.Lasso(alpha = lamda)
reg.fit(X_train,y_train)
B = reg.predict(X_test)
n_test = B.size
MSE = np.linalg.norm(B,ord=2)**2/n_test
numZeros = reg.coef_
print(MSE)
```

α	Mean Square Error	Number of zeros in θ
64	511	12

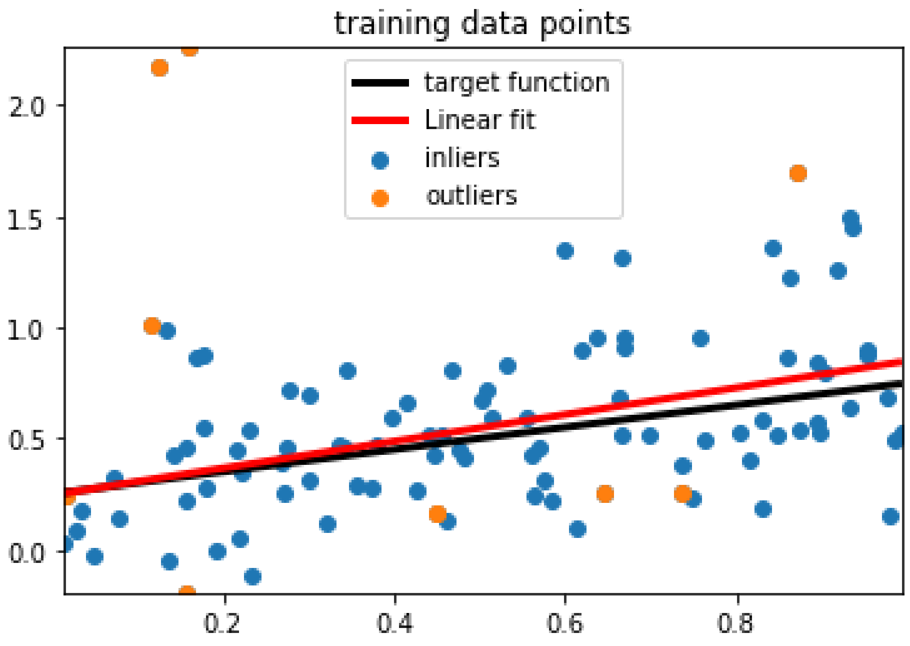
Problem 2:

In this problem I'd like you to use the following code to generate a data set to evaluate various approaches to regression in the presence of outliers.

a) To begin, find a linear fit using the code for ridge regression that you produced in the first problem. Report the value of λ that you selected, the slope and intercept of your linear fit, and plot your linear fit together with the data points and the target function. Submit your code.

```
ttfamily
from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
np.random.seed(2020)
n = 100
xtrain = np.random.rand(n)
ytrain = 0.25 + 0.5*xtrain + np.sqrt(0.1)*np.random.randn(n)
idx = np.random.randint(0,100,10)
ytrain[idx] = ytrain[idx] + np.random.randn(10)


#Construct A matrix
B = np.ones([n,1])
A = np.insert(B, 1, xtrain, axis=1)
lamda = 11
Gamma = (lamda*np.identity(2))
Gamma[1][1] = 0

theta = np.linalg.inv(np.dot(A.T,A)+Gamma).dot(A.T).dot(ytrain)

## Plot the training points
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00'])
plt.scatter(xtrain, ytrain, cmap=cmap_light)
plt.xlim(xtrain.min(), xtrain.max())
plt.ylim(ytrain.min(), ytrain.max())
plt.title("training data points")
plt.scatter(xtrain[idx], ytrain[idx], cmap=cmap_bold)
## plot the target function
t1 = np.arange(0.0, 1.0, 0.01)
plt.plot(t1,0.25 + 0.5*t1,'k',linewidth=3.0)

plt.plot(t1,theta[0] + theta[1]*t1,'r',linewidth=3.0)

plt.legend(['target function','Linear fit','inliers','outliers'])
## Show the plot
plt.show()
```

λ	*Slope*	*intercept*
11	0.245658	0.602902

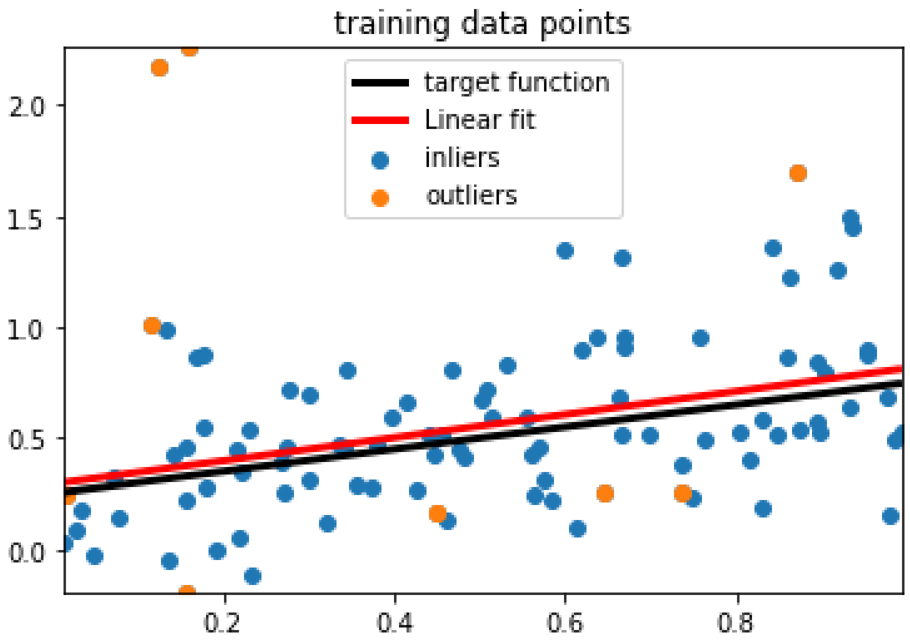


b) Next, I would like you to find a linear fit using the Huber loss. This can be done via $reg = linearmodel.HuberRegressor(\epsilon = 1.35, \alpha = 0.001)$ $reg.fit(xtrain.reshape(-1, 1), ytrain)$ You have two parameters to choose here: ϵ (which controls the shape of the loss function and needs to be greater than 1.0) and α (the regularization parameter). Report the values of ϵ and α you selected, the slope and intercept of your linear fit (see $reg.intercept$ and $reg.coef$), and plot your linear fit together with the data points and the target function. Submit your code.

ttfamily

```
reg = linear_model.HuberRegressor(epsilon = 2.50, alpha=0.050)
reg.fit(xtrain.reshape(-1,1),ytrain)

## Plot the training points
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00'])
plt.scatter(xtrain, ytrain, cmap=cmap_light)
plt.xlim(xtrain.min(), xtrain.max())
plt.ylim(ytrain.min(), ytrain.max())
plt.title("training data points")
plt.scatter(xtrain[idx], ytrain[idx], cmap=cmap_bold)
## plot the target function
t1 = np.arange(0.0, 1.0, 0.01)
plt.plot(t1,0.25 + 0.5*t1,'k',linewidth=3.0)

plt.plot(t1,reg.intercept_ + reg.coef_*t1,'r',linewidth=3.0)

plt.legend(['target function','Linear fit','inliers','outliers'])
## Show the plot
plt.show()
```

ϵ	α	Slope	Intercept
2.5	0.05	0.2935	0.5212



Appendix

ttfamily

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 16 10:59:17 2021

@author: IanSi
"""

import numpy as np
from sklearn.datasets import load_boston
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

boston = load_boston()
X = boston.data
y = boston.target

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.208,random_state=0)

X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)


"""
Least Mean Square: Problem 1 part A
"""

#Training Least square regression,

n,m = X_train.shape
A_train = np.hstack((np.ones((n,1)),X_train))

theta = np.linalg.inv(np.dot(A_train.T,A_train)).dot(A_train.T).dot(y_train)

#Mean-Square Test Error
n_test,m_test = X_test.shape
A_test = np.hstack((np.ones((n_test,1)),X_test))
```

```
MSE = (np.linalg.norm(y_test - np.dot(A_test,theta), ord=2)**2)/n_test
print(MSE)



"""
Ridge Regression: Problem 1 part B
"""

#Getting biased by small values of lambda
X_fit,X_holdout,y_fit,y_holdout = train_test_split(X_train,y_train,test_size=0.308,random_state=0)

MSE_min = float('inf')
#Assign a curve, theta, from a particular lambda, and test with holdout set
n,m = X_fit.shape
n_holdout,m_holdout = X_holdout.shape
A_holdout = np.hstack((np.ones((n_holdout,1)),X_holdout))
A_fit = np.hstack((np.ones((n,1)),X_fit))
for q in range(1,31,1):
    lamda = 500
    Gamma = ((lamda**0.5)*np.identity(m+1))
    Gamma[0][0] = 0
    theta = np.linalg.inv(np.dot(A_fit.T,A_fit) + (np.dot(Gamma.T,Gamma))).dot(A_fit.T).dot(y_fit)

    MSE = (np.linalg.norm(y_holdout - np.dot(A_holdout,theta), ord=2)**2)/n_holdout
    if(MSE < MSE_min):
        q_min = q
        MSE_min = MSE
        theta_optimal = theta

#Mean-Square Test Error
n_test,m_test = X_test.shape
A_test = np.hstack((np.ones((n_test,1)),X_test))
MSE = (np.linalg.norm(y_test - np.dot(A_test,theta_optimal), ord=2)**2)/(n_test)
print(MSE)



"""
LASSO: Problem 1 part C
"""

from sklearn import linear_model

X_fit,X_holdout,y_fit,y_holdout = train_test_split(X_train,y_train,test_size=0.308,random_state=0)

lamda = 1
MSE_min = float('inf')
for q in range(-10,21,1):
    reg = linear_model.Lasso(alpha = 2*q)
    reg.fit(X_fit,y_fit)
    B = reg.predict(X_holdout)
    n_holdout = B.size
    MSE = np.linalg.norm(B,ord=2)**2/n_holdout
    if(MSE < MSE_min):
        MSE_min = MSE
        lamda = 2*q
reg = linear_model.Lasso(alpha = lamda)
reg.fit(X_train,y_train)
B = reg.predict(X_test)
n_test = B.size
MSE = np.linalg.norm(B,ord=2)**2/n_test
numZeros = reg.coef_
print(MSE)


"""
Outliers: Problem 2
"""

from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
np.random.seed(2020)
n = 100
xtrain = np.random.rand(n)
ytrain = 0.25 + 0.5*xtrain + np.sqrt(0.1)*np.random.randn(n)
idx = np.random.randint(0,100,10)
ytrain[idx] = ytrain[idx] + np.random.randn(10)
```

```
"""
Problem 2 part A
"""

#Construct A matrix
B = np.ones([n,1])
A = np.insert(B, 1, xtrain, axis=1)
lamda = 11
Gamma = (lamda*np.identity(2))
Gamma[1][1] = 0

theta = np.linalg.inv(np.dot(A.T,A)+Gamma).dot(A.T).dot(ytrain)
plt.plot(t1,theta[0] + theta[1]*t1,'r',linewidth=3.0)



"""
Problem 2 part B
"""

reg = linear_model.HuberRegressor(epsilon = 2.50, alpha=0.050)
reg.fit(xtrain.reshape(-1,1),ytrain)

## Plot the training points
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00'])
plt.scatter(xtrain, ytrain, cmap=cmap_light)
plt.xlim(xtrain.min(), xtrain.max())
plt.ylim(ytrain.min(), ytrain.max())
plt.title("training data points")
plt.scatter(xtrain[idx], ytrain[idx], cmap=cmap_bold)
## plot the target function
t1 = np.arange(0.0, 1.0, 0.01)
plt.plot(t1,0.25 + 0.5*t1,'k',linewidth=3.0)

plt.plot(t1,reg.intercept_ + reg.coef_*t1,'r',linewidth=3.0)

plt.legend(['target function','Linear fit','inliers','outliers'])
## Show the plot
plt.show()
```