Georgia Institute of Technology

ECE 4803: Fundamentamentals of Machine Learning (FunML)

Spring 2022

Homework Assignment # 3

Due: Friday, February 18, 2022 @8PM

(grace period) Saturday, February 19, 2022 @5PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., Jupyter Lab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Print a PDF copy of the notebook with all its outputs printed and submit the **PDF** on `Canvas` under Assignments.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Rename the PDF according to the format: ***LastName_FirstName_ECE_4803_sp22_assignment_#.pdf***
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.

- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for lectures 8, 9 to help you with this assignment.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working

on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
Solution to Problem 2 Part (c). Failing to do so may result in a *20% penalty* of the total grade.

Assignment Objectives:

- Better understand the regression algorithms we discussed in class
- Experiment those algorithms on multiple datasets
- Understand the idea of regularization and the effect it can have in different scenarios
- Perform analyses between algorithms.
- Advance in your `Python` knowledge and experience

Guide for Exporting Ipython Notebook to PDF:

Here is a video summarizes how to export Ipythin Notebook into PDF.

- **[Method1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print]** and then choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method2: colab-pdf script]**
  The author of that video provided an alternative method that can generate better layout PDF. However, it only works for Ipythin Notebook without embedded images.
  How to use: Put the script below into cells at the end of your Ipythin Notebook. After you run the fisrt cell, it will ask for google drive permission. Executing the second cell will generate the PDF file in your google drive home directory. Make sure you use the correct path and file name.

```
## this will link colab with your google drive
from google.colab import drive

drive.mount('/content/drive')
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('LastName_FirstName_ECE_4803_sp22_assignment_#.ipynb') ## change path and file name
```

- **[Method3: GoFullPage Chrome Extension] (most recommended)**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Note: Georgia Tech provides a student discount for Adobe Acrobat subscription. Further information can be found here.

Problem 1: Least Squares for Linear Regression (20pts)

In this problem, you will be dealing with a dataset called `California Housing.` The dataset contains 8 feature attributes: holder income, house age, average number of rooms, etc. All these 8 feature attributes were obtained for each of the $n = 20,640$ houses that are included in this dataset. Finally, the dataset includes, for every house, the average house value in units of \$100k. Click HERE to learn more about the dataset.

Problem 1 (a)
Refer to the slides in Lectures 8 and 9, particularly check the way we set up the problem for Linear Regression. In this question, determine the values for N, P, the length of the vector $\hat{\theta}$, and the dimensions of the matrix X. (Note that some of these values depend on the size of the training set, and in (a) we assume we use all examples for training. Remember that the structure of matrix X and the vectore θ should account for the bias, the intercept term.)

Problem 1 (b)
Write down the Least Square Cost Function as we defined it in lecture. Define every variable and parameter you use in this definition.

Problem 1 (c)
In class, we derived the solution for the Least Square function. We called the solution `Normal Equations.` Write down the Normal Equations solution. Then, check the dimensionalities of all terms in the Normal Equations to make sure they match for the multiplications of matrices and vectors in the Normal Equations. Explicitly include in your solutions the dimensions and show that they match. (in terms of N and P)

Problem 1 (d)

Write down the prediction equation you will use to predict the outcome. Define every variable and paramter in your equation.

Problem 1 (e)

You are provided below a class template called `MyLeastSquares` to implement your very own least squares class. In this question, you will complete the `fit` function within the class below. Match the inputs and their dimensionalities (and shapes) to what you worked on in the above **(a) - (c)**. The solution for this part is the code for `fit` and you have to clearly comment on your code to explain every part of the code.

Problem 1 (f)

In this part, you will complete the `predict()` function within the class below. Match the inputs and their dimensionalities (and shapes) to what you worked on in the above **(a) - (e)**. The solution for this part is the code for `predict()` and you have to clearly comment on your code to explain every part of the code.

Problem 1 (g)

Now it is time to run the cell after you complete the above steps to **train** and **predict** your **Linear** regressor on the `California Housing` dataset. *Hint: The expected training and test errors are in the range* 0 *to* 100.

Problem 1 (h)

In this part, you will vary the size of the training dataset from 10 to $10,000$, and compute the MSE on the training data and the MSE in the testing data. Choose 200 as interval. Plot the two curves on the same plot, where the x-axis is the size of the training dataset and the y-axis is log of MSE. Describe and explain the underlying trend. Does it make sense? Write well commented code for the analysis and generate clear, well-labeled plots.

Hint:

- Consider the use of `numpy.linalg.inv` and `numpy.matmul` functions.

Solution Problem 1 (a)

Determine the values:

- N: it could be up to 20640 (number of data samples), but in the following setup, **`N=1000`**
- **`P = 8`** (features)
- Length of vector $\hat{\theta}$ = P+1 (bias) = **`9`**; it will be a (P+1, 1) vector
- Dimension of matrix X = (N,P+1) = **`1000 x 9`** array

These values are obtained with following code:

```
import numpy as np
from sklearn.datasets import fetch_california_housing

cal_housing = fetch_california_housing()
print('Number of feature is: ' + str(len(cal_housing['feature_names'])) )

print('Number of target classes is: ' + str(len(cal_housing['target_names'])))

X = cal_housing.data

print('N = ' + str(np.shape(X)[0]))
print('P = ' + str(np.shape(X)[1]))
print('Length of theta = ' + str(np.shape(X)[1] + 1))
print('Dimension of X = ' + str(np.shape(X)))
```

```
Number of feature is: 8
Number of target classes is: 1
N = 20640
P = 8
Length of theta = 9
Dimension of X = (20640, 8)
```

Solution Problem 1 (b)

Least square cost function:

$L(\hat{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{\theta}^T x_i - y_i)^2$

- N: numbers of sample
- $\hat{\theta}^T$: the transpose of the estimated coefficient (weight)
- x_i: input samples
- y_i: scalar output

Solution Problem 1 (c)

1) Normal Equation:
$\hat{\theta} = (X^T X)^{-1} X^T y$

2) $\hat{\theta}$'s Dimension Calculation (row, columns):

- X = (N, P+1)
- $X^T X$ = (P+1, N) * (N, P+1) = (P+1, P+1)
- $(X^T X)^{-1} X^T$ = (P+1, P+1) * (P+1, N) = (P+1, N)
- $(X^T X)^{-1} X^T y$ = (P+1, N) * (N, 1) = (P+1, 1) = $\hat{\theta}$

Therefore, we conclude that this fit the dimenstion of $\hat{\theta}$, which is (P+1, 1) vector

Solution Problem 1 (d)

$\hat{y} = X\hat{\theta}$

- $\hat{y}$: estimated y(output)
- X: input sample matrix
- $\hat{\theta}$ = estimated coefficients

Solution Problem 1 (e) (f) (g)

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

class MyLeastSquares:
    def __init__(self, X_train, y_train):
        """Function stores feature matrix and corresponding target data.

        Parameters:
        -----------
        X_train: array_like, shape(N,P)
```

```
            ndarray containing N training examples, each with P feature values.

        y_train: array_like, shape(N,1)
            ndarray containing target values for each of N examples in X_train."""

        self.X_train = X_train
        self.y_train = y_train
        # print(np.shape(X_train))

    def fit(self):
        """Function computes the weight vector theta of shape (P+1, 1) for regression"""

        # append ones for bias
        X = np.concatenate((np.ones((np.shape(self.X_train)[0], 1)), self.X_train), axis=1) ##TODO

        self.theta = np.reshape(np.matmul( np.matmul( np.linalg.inv(np.matmul( np.transpose(X) , X)) , np.transpose(X)) , self.y_


    def predict(self, X_test):
        """Function predicts targets for given X_test.

        Parameters:
        -----------
        X_test: array_like, shape(N,P)
            ndarray containing N test examples with P features each.

        Returns: array_like, shape(N,1).
            ndarray containing predicted targets of shape (N,1).
        """

        # append ones for bias
        X = np.concatenate((np.ones((np.shape(X_test)[0], 1)), X_test), axis=1)
        y_pred = np.matmul(X, self.theta)

        return y_pred


#-----------------Don't change anything below-----------------------#


# load dataset
cali_houses = fetch_california_housing()
X, y = cali_houses.data, cali_houses.target
```

```
# train-test split
n_train = 1000
n_test = X.shape[0] - n_train

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=n_train, test_size=n_test, random_state=4803)

# train theta
LS = MyLeastSquares(X_train, y_train)
LS.fit()

# test
y_pred = LS.predict(X_test)

# evaluate performance
print('MSE on Training Data: {:0.3f}'.format(np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size))
print('MSE on Test Data: {:0.3f}'.format(np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size))
```

```
MSE on Training Data: 0.440
MSE on Test Data: 8.897
```

Solution Problem 1 (h)

```
# for part (h)
## plot train size vs log MSE

import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

# load dataset
cali_houses = fetch_california_housing()
X, y = cali_houses.data, cali_houses.target


# define training set sizes
training_set_sizes = np.linspace(10, 10000, 50).astype(int)

# MSE  array
```

```
MSE_training = np.zeros(np.shape(training_set_sizes)[0])
MSE_test = np.zeros(np.shape(training_set_sizes)[0])

counter = 0

for train_size in training_set_sizes:

    # define train and test sizes
    N_train = train_size
    N_test = X.shape[0] - N_train
    # print(N_train)
    # print(N_test)

    current_MSE_training = 0;
    current_MSE_test = 0;

    for ii in range(30):

      X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=N_train, test_size=N_test)

      # train theta
      LS = MyLeastSquares(X_train, y_train)
      LS.fit()

      # test
      y_pred = LS.predict(X_test)

      # Calculate MSE
      current_MSE_training = current_MSE_training + np.log(np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size
      current_MSE_test = current_MSE_test + np.log(np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size)

    # Update MSE
    MSE_training[counter] = current_MSE_training / 30
    MSE_test[counter] = current_MSE_test / 30

    counter = counter + 1

# PLot


plt.plot(training_set_sizes,MSE_training, label="MSE_training")
plt.plot(training_set_sizes,MSE_test, label="MSE_test")
```

```
# Add Title
plt.title("log(MSE) vs size of training dataset for both traning and testing data")

# Add Axes Labels

plt.xlabel("size of training dataset")
plt.ylabel("log(MSE)")
plt.legend()

# Display

plt.show()
```

The plot for Problem1 (h) make sense since it looks like the Model Validation learning curves:

- Training error should start very low when training set is small, and increases as more training data added.
- Testing error should start high and decreases as more training data added.

Problem 2: Singular Value Decomposition and Low-rank Approximation (20pts)

In Linear Regression, and in Regression in general, we deal with features. One of the challegnes arises when a feautre or more is (or are) a

linear combination of other features. This is called linearly dependent features and some call it collinearity of features. When this happens, the resulting matrix X is ill-conditioned, which may lead to unstable solutions or even singular matrices. Recall the problem with singular matrices, we cannot find the inverse. The underlying cause of this instability is the very small (or even zero-valued) singular values in the singular value decomposition (SVD) of the data matrix X, stemming from the fact that one or more of its columns happen to be linear combinations of the other columns. A potential solution around this problem is to reconstruct X via a low rank matrix approximation,$\hat{X}$, using only the largest singular values in the original SVD of X. This leads to a more stable solution for the regression algorithm than if used with the original X.

For an $N \times (P+1)$ matrix X and a corresponding $N \times 1$ vectore $\mathbf{y}$, the least squares formulation is stated as follows:

$$X\theta = \mathbf{y}, \tag{1}$$

where θ is a $(P+1) \times 1$ parameter vector. We could replace X with its SVD to get:

$$U\Sigma V^T\theta = \mathbf{y}, \tag{2}$$

where U is an $N \times N$ orthogonal matrix (containing the left singular vectors), Σ is an $N \times (P+1)$ diagonal matrix (containing the singular values on its main diagonal and zeros elsewhere), and V is an $(P+1) \times (P+1)$ orthogonal matrix (containing the right singular vectors). An r-rank approximation of X is given as follows, where $r \leq N$ and $r \leq P+1$:

$$\hat{X}_r = U(:, :r) \times \Sigma(:r, :r) \times V^T(:r, :) \tag{3}$$

Replacing X by $\hat{X}_r$ in Equation 2 followed by the inverse, we obtain the following solution for theta:

$$\hat{U}_r\hat{\Sigma}_r\hat{V}_r^T\theta = \mathbf{y}, \tag{4}$$

$$\hat{\theta} = \hat{V}_r\hat{\Sigma}_r^{-1}\hat{U}_r^T\mathbf{y}. \tag{5}$$

Check the Linear Algebra Handout and the SVD-Low Rank Approximation Notes in the Course Content Page on Canvas.

Problem 2 (a)

Calculate the SVD for the following Matrix and compare the result from `np.linalg.svd`. Start with A^TA and eigen decomposition. Do not use any built-in SVD-related functions, but you may use python to obtain any eigenvalue decompositions as needed.

$$\mathbf{A} = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 8 & -2 \end{pmatrix}.$$

Problem 2 (b)

What is the rank of the following Matrix:

$$\begin{pmatrix} 1 & 1 & 0 & -1 \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} -1 & -1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

Problem 2 (c)

Use the rank you calcualted in **Part (b)** to reconstruct a low-rank version of $\mathbf{A}$.

Solution Problem 2 (a)

```
A = np.array([[4, 2, 1],
              [2, 8, -2]])

#-----------------Don't change anything above-----------------------#

ATA = np.matmul(np.transpose(A), A) ##TODO

Lambda_A, V_A = np.linalg.eig(ATA) ##TODO


sigmas = np.sqrt(Lambda_A) ##TODO


svd_U_A = np.matmul(A, np.matmul( np.linalg.inv(np.transpose(V_A)), np.linalg.inv(np.diag(sigmas)) )) ##TODO
svd_Sigma_A = np.diag(sigmas) ##TODO
svd_V_A = V_A ##TODO

#-----------------Don't change anything below-----------------------#
U_A, Sigma_A, V_A_transpose = np.linalg.svd(A, full_matrices=True)

print('Built-in svd:')
print('U_A: \n', np.concatenate((U_A, np.zeros((2, 1))), axis=1))
print('Sigma_A: \n', np.diag(np.append(Sigma_A, 0)))
print('V_A: \n', np.transpose(V_A_transpose))

print('A: \n', U_A@np.diag(np.append(Sigma_A, 0))[:2, :]@V_A_transpose, '\n')

print('Your implementation:')
print('U_A: \n', svd_U_A)
print('Sigma_A: \n', svd_Sigma_A)
print('V_A: \n', svd_V_A)

print('A: \n', svd_U_A@svd_Sigma_A@np.transpose(svd_V_A))
```

```
Built-in svd:
U_A:
 [[-0.34845607 -0.93732511  0.        ]
 [-0.93732511  0.34845607  0.        ]]
Sigma_A:
 [[8.95425196 0.         0.        ]
 [0.         3.58069431 0.        ]
 [0.         0.         0.        ]]
V_A:
 [[-0.36501927 -0.8524571  -0.37426972]
 [-0.91526496  0.25497803  0.31189143]
 [ 0.17044351 -0.45640234  0.87329601]]
A:
 [[ 4.  2.  1.]
 [ 2.  8. -2.]]

Your implementation:
U_A:
 [[-3.48456074e-01  9.37325111e-01 -5.96046448e-08]
 [-9.37325111e-01 -3.48456074e-01 -2.98023224e-08]]
Sigma_A:
 [[8.95425196e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 3.58069431e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.83616829e-08]]
V_A:
 [[-0.36501927  0.8524571  -0.37426972]
 [-0.91526496 -0.25497803  0.31189143]
 [ 0.17044351  0.45640234  0.87329601]]
A:
 [[ 4.  2.  1.]
 [ 2.  8. -2.]]
```

Solution Problem 2 (b)

Since row 2 of B is the multiplication of row 1 of B, the rank of B will be 3-1 = 2, which is also proven in the code below using numpy.

- Answer: $rank(B) = 2$

```
B = np.array([[1, 1, 0, -1],
              [-1, -1, 0, 1],
```

```
              [1, 1, 0, 0]])
B_rank = np.linalg.matrix_rank(B)
print('rank of B: ', B_rank)
```

```
rank of B:  2
```

Solution Problem 2 (c)

Lower rank A Calculations:

$$A = U_{2,2}S_{2,2}V_{2,2}^T = \begin{pmatrix} -0.34845607 & -0.93732511 \\ -0.93732511 & 0.34845607 \end{pmatrix} \begin{pmatrix} 0 & 8.95425196 \\ 3.58069431 & 0 \end{pmatrix} \begin{pmatrix} -0.36501927 & 0.8524571 \\ -0.91526496 & 0.25497803 \\ 0.17044351 & 0.45640234 \end{pmatrix}$$

Thus,

$$A = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 8 & -2 \end{pmatrix}$$

Using python for calculation:

```
U_A, Sigma_A, V_A_transpose = np.linalg.svd(A, full_matrices=True)

U = U_A
S = np.diag(np.append(Sigma_A, 0))[:2, :]
V = V_A_transpose
# print(np.shape(U))


Lower_rank_A = U[:, 0:2]@S[:, 0:2]@V[0:2, :]
print("Lower rank version A: \n", Lower_rank_A)
```

```
Lower rank version A:
 [[ 4.  2.  1.]
 [ 2.  8. -2.]]
```

Problem 3: Singular Value Decomposition and Least Squares (20pts)

In this problem, you will be dealing with the `diabetes` dataset that contains 442 examples and each example originally contains 10 features. However, we modified the dataset by replacing one of the 10 features by new feature that is a simple linear combination of the other 9 features. This results in a rank-deficient data matrix X. In this problem, you will be repeating the work you did in the Problem 1 where you calcualted the output using Normal Equations. But now instead of using X, you will use $\hat{X}$. Given below is the structure of the class `MySVDLeastSquares`. Answer the following questions.

Problem 3 (a)

Write a code that computes the approximation of X via SVD and a given `rank` parameter. Check the `numpy` library for the proper function that provides you with SVD.

Problem 3 (b)

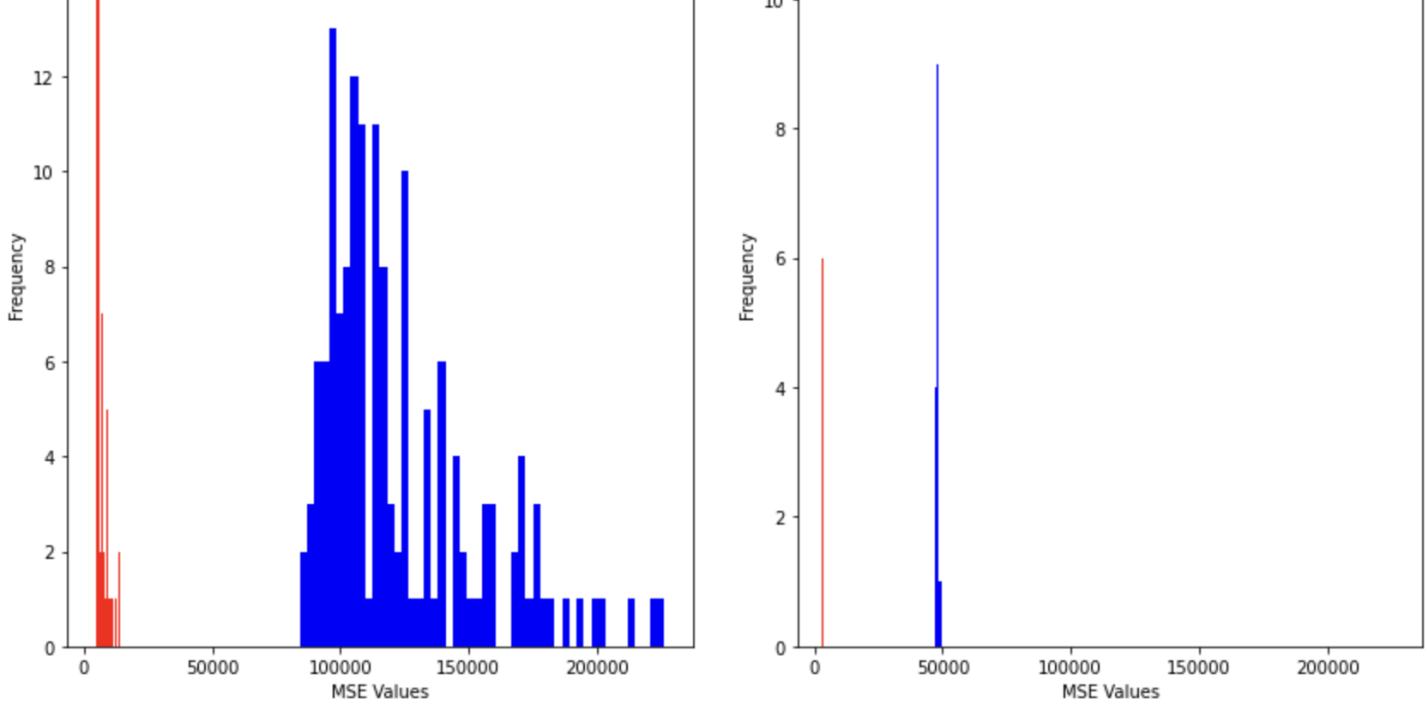
Within the class `MySVDLeastSquares` given below, implement the `fit()` function that computes the $\hat{\theta}$ vector via a low-rank approximation of X as depicted in Equation (5) above. The rank is provided/input by the user. Start from your solution for Problem 1 in this Assignment.

Problem 3 (c)

Within the class `MySVDLeastSquares` given below, implement the `predict()` function. Start from your solution for Problem 1 in this Assignment.

Problem 3 (d)

One of the practices in machine learning is to test the performance of Regression by adding noise to the labels and recording the MSE on both the training and the test data. Then, we plot the histograms of the MSE values. In doing so, we generated two histograms (given below), where the one on the left is the histogram when we run the code on the origianl full rank X and the plot on the right hand side is when the code is run on the low rank $\hat{X}$. Comment on these two plots and explain the differences and what you conclude about using X versus $\hat{X}$.



Problem 3 (e) [BONUS question - 15 points] Your task in this bonus question is to generate the code that can generate the two plots in **(d)**.

Hints:

- Don't forget to set up your problem matrices and vectors while accounting for the bias (i.e., intercept term) within the matrices and vectors.
- Check the functions: `numpy.linalg.svd` and `numpy.linalg.matrix_rank`.
- Carry out the experiment for a reasonable large number of trials
- Fix the random seed in `train_test_split` function. The randomness should only be in the noise added to $\mathbf{y_train}$.

Solution Problem 3 (a)(b)(c)

```
# for parts (a), (b), and (c)

from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split


class MySVDLeastSquares:
    def __init__(self, X_train, y_train):
```

```
    """Function stores feature matrix and corresponding target data.

    Parameters:
    -----------
    X_train: array_like, shape(N,P)
        ndarray containing N training examples, each with P feature values.

    y_train: array_like, shape(N,1)
        ndarray containing target values for each of N examples in X_train."""

    self.X_train = X_train
    self.y_train = y_train

def fit(self, rank):
    """Function computes the weight vector of shape (P+1, 1) for regression via a low rank approximation
    with given rank"""

    # append ones for bias
    X = self.X_train ##TODO

    self.theta = 0 ##TODO

    # My code:

    # append ones for bias
    X = np.concatenate((np.ones((np.shape(X)[0], 1)), X), axis=1) ##TODO

    ## SVD
    U_X, Sigma_X, V_X_transpose = np.linalg.svd(X, full_matrices=True)
    U = U_X

    S = np.diag(Sigma_X, 0)
    # print(Sigma_X)

    V = V_X_transpose

    X = U[:, 0:rank]@S[0:rank, 0:rank]@V[0:rank, :]



    updated_S = np.linalg.inv(S[0:rank, 0:rank])
    updated_S[updated_S > 1e5] = 0
```

```
        # truncate S according to the rank
        V = V_X_transpose[0:rank, :]
        U = np.transpose(U[:, 0:rank])
        pseudo_inv_X = np.matmul(np.matmul(np.transpose(V), updated_S), U)

        self.theta = np.reshape( np.matmul(pseudo_inv_X , self.y_train), [-1,1])


    def predict(self, X_test):
        """Function predicts targets for given X_test.

        Parameters:
        -----------
        X_test: array_like, shape(N,P)
            ndarray containing N test examples with D features each.

        Returns: array_like, shape(N,1).
            ndarray containing predicted targets of shape (N,1).
        """

        # append ones for bias
        X = np.concatenate((np.ones((np.shape(X_test)[0], 1)), X_test), axis=1)
        ##TODO

        y_pred = np.matmul(X, self.theta)
        ##TODO

        return y_pred



#-----------------Don't change anything below-----------------------#

# load dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
X[:,[-1]] = np.sum(X[:,:-1],axis=1,keepdims=True)


# train-test split
n_train = 40
n_test = X.shape[0] - n_train
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=n_train, test_size=n_test, random_state=4803)

# train theta
LS = MySVDLeastSquares(X_train, y_train)
LS.fit(rank=10)

# # test
y_pred = LS.predict(X_test)

# evaluate performance
print('MSE on Training Data: {:0.3f}'.format(np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size))
print('MSE on Test Data: {:0.3f}'.format(np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size))
```

```
MSE on Training Data: 3210.214
MSE on Test Data: 3273.707
```

Solution Problem 3 (d)

The first plot using full-rank has a larger MSE values with a wider distribution for both "Traning MSE" and "Test MSE"(overall having a larger range of error), comparing to the second plot with a lower rank. This is due to the fact that lower rank eliminate the small sigma value in the S matrix taht will contribute to a large error in the case of full rank.

Solution Problem 3 (e) [bonus]

```
error = 5 * np.random.rand(1) * np.ones(7)
print(error)
```

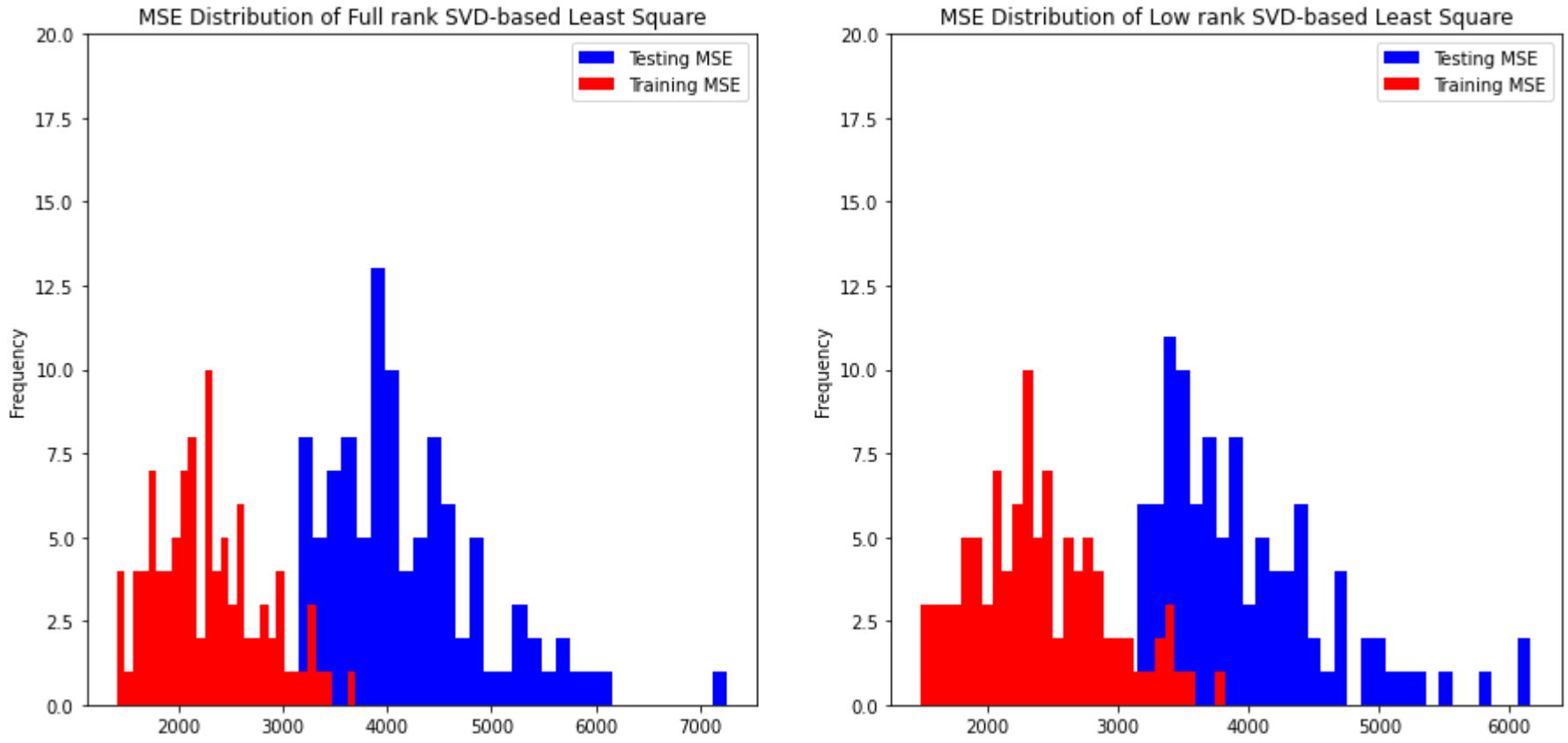
```
[4.03528518 4.03528518 4.03528518 4.03528518 4.03528518 4.03528518
 4.03528518]
```

```
# for part (e)

## plot MSE histogram of full-rank and low-rank SVD LS under noise
import matplotlib.pyplot as plt
```

```
# set up plots
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,7))

# load dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
X[:,[-1]] = np.sum(X[:,:-1],axis=1,keepdims=True)


# Parameter setup
errors = 50
k = 100
full_rank_MSE_testing = np.zeros(k)
full_rank_MSE_training = np.zeros(k)
low_rank_MSE_testing = np.zeros(k)
low_rank_MSE_training = np.zeros(k)

current_f_test = 0
current_f_train = 0
current_l_test = 0
current_l_train = 0
counter = 0

# print(np.random.rand(3))


for ii in range(k):
    # train-test split
    n_train = 40
    n_test = X.shape[0] - n_train

    error = (ii*15) * np.random.rand(1) #* np.ones(np.shape(y)[0])
    # y_new = y  + error

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=n_train, test_size=n_test)

    y_train += 0.1*np.random.normal(y_train.mean(), 0.5*y_train.std(), size=y_train.shape)


    # train theta
    LS = MySVDLeastSquares(X_train, y_train)
    LS.fit(rank=11)
```

```
    # # test
    y_pred = LS.predict(X_test)

    # Calculate MSE
    current_f_test = np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size
    current_f_train = np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size

    ##########################################################################################################################

    # train theta
    LS = MySVDLeastSquares(X_train, y_train)
    LS.fit(rank = 8)

    # test
    y_pred = LS.predict(X_test)

    # Calculate MSE
    current_l_test = np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size
    current_l_train = np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size

    # Calculate MSE
    full_rank_MSE_training[ii] = current_f_train
    full_rank_MSE_testing[ii] = current_f_test
    low_rank_MSE_training[ii] = current_l_train
    low_rank_MSE_testing[ii] = current_l_test
    counter += 1

#################################################################
## Plot

# the histogram of the data
b = 30

n, bins, patches = ax1.hist(full_rank_MSE_testing, bins=b, color = 'b', label='Testing MSE')
n, bins, patches = ax1.hist(full_rank_MSE_training, bins=b, color = 'r', label='Training MSE')
ax1.legend(prop={'size': 10})


# print(full_rank_MSE_training)

n, bins, patches = ax2.hist(low_rank_MSE_testing, bins=b, color = 'b', label='Testing MSE')
n, bins, patches = ax2.hist(low_rank_MSE_training, bins=b, color = 'r', label='Training MSE')
```

```
ax2.legend(prop={'size': 10})

ax1.set_xlabel('MSE')
ax1.set_ylabel('Frequency')
ax1.set_title('MSE Distribution of Full rank SVD-based Least Square')
# ax1.set_xlim(100, 100000)
ax1.set_ylim(0, 20)

ax2.set_xlabel('MSE')
ax2.set_ylabel('Frequency')
ax2.set_title('MSE Distribution of Low rank SVD-based Least Square')
# ax2.set_xlim(100, 100000)
ax2.set_ylim(0, 20)

plt.show()
```



From the plot above, we can see that the MSE with lower rank has a smaller range of MSE for testing data.

Problem 4: Polynomial Features with Linear Least Squares on Multi-Output Regression(20pts)

Problem 4: Polynomial Features with Linear Least Squares on Multi-Output Regression(20pts)

As the name implies, linear least squares is able to model only linear combinations of its features. In some cases, however, the output may have a more complex, non-linear relationship with input feature data. A simple way to encode such non-linear relationships into the framework for linear least squares is to take powers of existing features and introduce them into the original data matrix as new features. The standard least squares framework is then able to learn weights to assign to these "new" features.

Moverover, we have seen the multiple-output regression in the lecture 8. Instead of having a vector of $\mathbf{y}$ and $\hat{\theta}$, we have a matrix of Y and $\hat{\Theta}$ now. Luckily, our least square `Normal equation` can extend to this multiple-output regression by replacing $\mathbf{y}$ and $\hat{\theta}$ into Y and $\hat{\Theta}$ directly.

Consider the problem below, where we artificially generate some data, X and corresponding non-linear targets Y as shown below:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \; Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ \vdots & \vdots \\ y_{N1} & y_{N2} \end{bmatrix} \quad \forall x_i, y_{ik} \in \mathbb{R}, \tag{7}$$

where $K = 2$ and it has the relationship: $y_{i1} = sinh(x_i)sin(x_i)$ and $y_{i2} = sin(x_i)$ for all $i = 1 \cdots N$.

Problem 4 (a)

You are provided a function called `generate_polynomial_features()` that takes in X and appends to it new features generated by raising the original feature values to a range of powers, starting from 1 until the specified degree. So calling `generate_polynomial_features(X, degree=3)` would for example return:

$$X = \begin{bmatrix} x_1 & x_1^2 & x_1^3 \\ x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots \\ x_N & x_N^2 & x_N^3 \end{bmatrix}, \; \forall x_i \in \mathbb{R}. \tag{8}$$

Complete the function to get this behavior, following the format laid out in the function definition. The new feature data is then trained via `np.linalg.lstsq` function on the target data. Execute the cell with `degree=10` to visualize the prediction superimposed on the ground-truth targets. Run the cell again, this time with `degree=1` (to produce a linear prediction). Which one looks better?

Problem 4 (b)

Suppose we use `degree=10`, What is the `Normal equation` of this polynomial features on multi-output regression? What is the dimension of $\hat{\Theta}$?

Problem 4 (c)

You should have been convinced in **(a)** as to the superiority of polynomial features in better modeling non-linear relationships in data. Does it make sense to believe that increasing the degree of the features would lead to an even greater improvement in the performance? Let's find out.

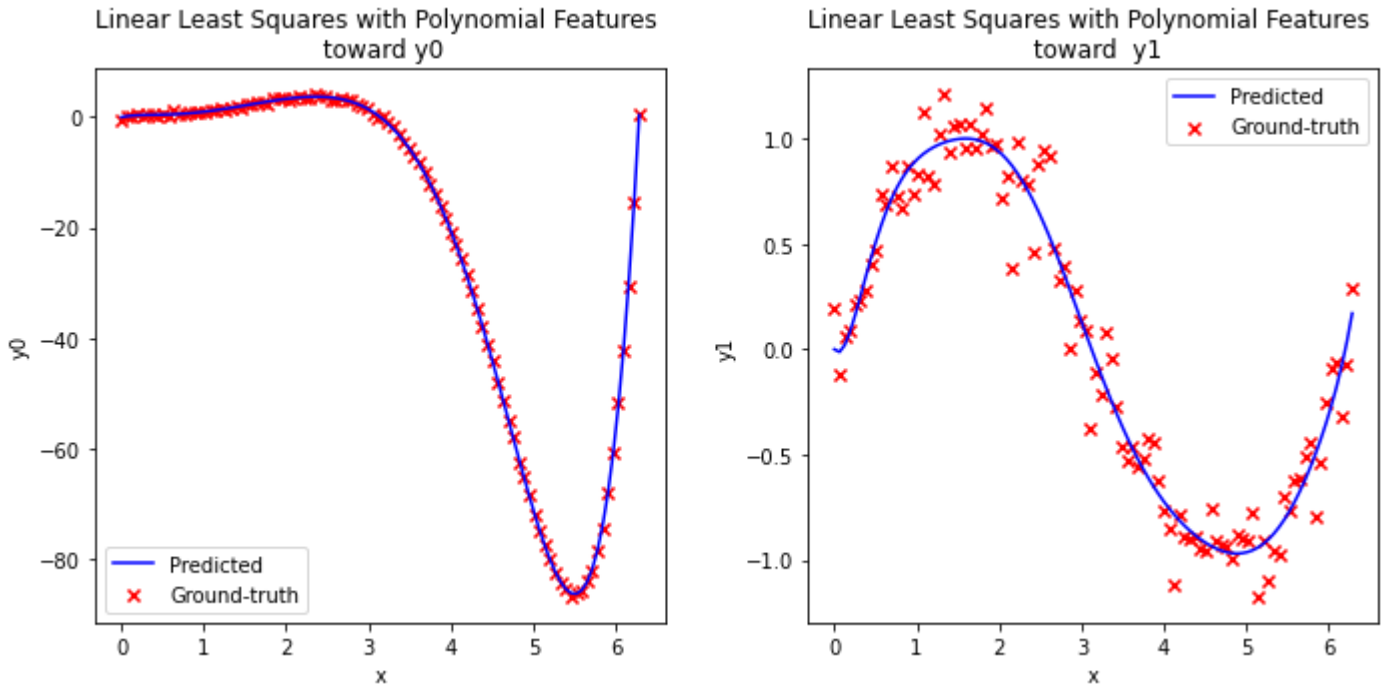
Plot the log MSE of the regressor on the artificial data with different degrees, starting from 1 to 50. What do you observe? Does it make sense? Provide a clear, well labeled plot to accompany your answer.

Solution Problem 4 (a)

```
# for part (a)

import numpy as np
import matplotlib.pyplot as plt

def gen_nonlinear_X_y():
    """Function generates a non-linear vector function and its indices for use in this question.

    Returns
    -------
    X: array_like of shape (Number of samples,1).
        vector of indices used to generate non-linear mapping.

    y: array_like of shape (Number of samples, 1)
        output function vector corresponding to X.
    """

    X = np.linspace(0,2*np.pi, 100).reshape(-1,1)  # generate X
    y1 = np.sinh(X) * np.sin(X) + np.random.normal(0, 0.3, X.shape)  # generate noisy targets
    y2 = np.sin(X) + np.random.normal(0, 0.15, X.shape)

    y = np.concatenate((y1.reshape(-1, 1), y2.reshape(-1, 1)), axis=1)

    return X,y


#-----------------Don't change anything above-----------------------#

def generate_polynomial_features(X, degree):
```

```
    """Function generates and appends polynomial features of a given
    data array X.

    Parameters
    ----------
    X: array_like of shape (N, 1)
        input feature data containing N training examples

    degree: int
        integer specifying the degree of the polynomial features
        generated. Must be greater than or equal to 2. default=2

    Returns
    -------
    X_poly: array_like of shape (N, power)
        ndarray of polynomial features containing original data plus the powers
        from 2,...,degree.
    """

    ##TODO
    X_poly = np.zeros([np.shape(X)[0], degree])

    for i in  range(degree):
      X_poly[:, i:i+1] = np.power(X, i+1)

    return X_poly



X, y = gen_nonlinear_X_y()  # generate data
X = generate_polynomial_features(X, degree=10) # generate linear and quadratic features

#-----------------Don't change anything below-----------------------#

theta = np.linalg.lstsq(X, y, rcond=None)[0]

y_pred = X @ theta


fig2, (ax3, ax4) = plt.subplots(1,2, figsize=(10, 5))

ax3.scatter(X[:,0], y[:, 0], marker='x', color='red', label='Ground-truth')
ax3.plot(X[:,0], y_pred[:, 0], color='blue', label='Predicted')
```

```
ax3.set_xlabel('x')
ax3.set_ylabel('y0')
ax3.set_title('Linear Least Squares with Polynomial Features \n toward y0')
ax3.legend()


ax4.scatter(X[:,0], y[:, 1], marker='x', color='red', label='Ground-truth')
ax4.plot(X[:,0], y_pred[:, 1], color='blue', label='Predicted')
ax4.set_xlabel('x')
ax4.set_ylabel('y1')
ax4.set_title('Linear Least Squares with Polynomial Features \n toward  y1')
ax4.legend()

plt.tight_layout()
plt.show()
```



The prediction of `degree=10` looks like a way better fitting than `degree=1` .

Solution Problem 4 (b)

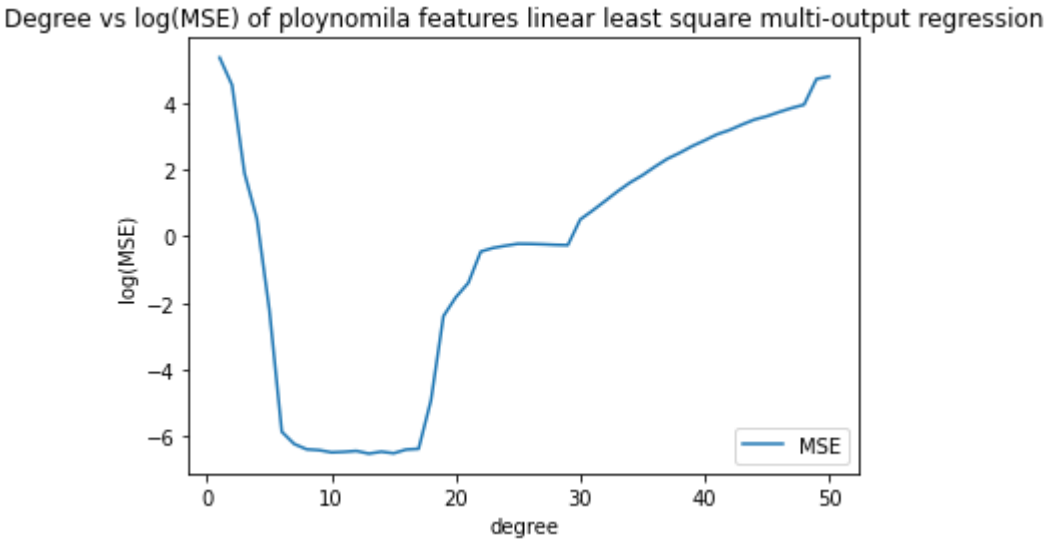
Normal Equation: $\hat{\theta} = (\boldsymbol{\phi}(X)^T \boldsymbol{\phi}(X))^{-1} \boldsymbol{\phi}(X)^T y$

- The length of $\hat{\theta}$ is degree+1 = 11. Therefore, it will be a `11 by 1` =(11, 1) column vector.

Solution Problem 4 (c)

```
# for part (c)

# plot degree vs log MSE
import numpy as np
import matplotlib.pyplot as plt

degrees = np.arange(1, 51)

# MSE  array
MSE_y0 = np.zeros(np.shape(degrees)[0])
MSE_y1 = np.zeros(np.shape(degrees)[0])


k = 50


for n in degrees:
  current_MSE_y0 = 0
  current_MSE_y1 = 0

  for ii in range(k):
    X, y = gen_nonlinear_X_y()  # generate data
    X = generate_polynomial_features(X, degree=n) # generate linear and quadratic features
    theta = np.linalg.lstsq(X, y, rcond=None)[0]
    y_pred = X @ theta

    current_MSE_y0 += np.log(np.sum((y_pred[:, 0] - y[:, 0])**2)/y_pred[:, 0].size)
    current_MSE_y1 += np.log(np.sum((y_pred[:, 1] - y[:, 1])**2)/y_pred[:, 1].size)


  MSE_y0[n-1] = current_MSE_y0 / k
  MSE_y1[n-1] = current_MSE_y1 / k
```

```
# # PLot
plt.plot(degrees,MSE_y0+MSE_y1, label="MSE")

# Add Title
plt.title("Degree vs log(MSE) of ploynomila features linear least square multi-output regression")

# Add Axes Labels

plt.xlabel("degree")
plt.ylabel("log(MSE)")
plt.legend()

# Display

plt.show()
```



This curve with the trend of decreasing then increasing error as the degree goes up is what I expected, since:

- As the degrees first start increasing, the error decreases since as we observed from Problem 4 (b), high degree does fit the data better.
- However, as the degree goes higher, the problem of overfitting will occur and thus increases the error.

Problem 5: Regularizing Regression via the Ridge Penalty (20pts)

Another method commonly used in many machine learning approaches to regularize limited labeled data problems and prevent overfitting is `Ridge Regression`. Ridge Least Squares, as you studied in class, introduced an L-2 norm penalty to the cost function. Among other things, this leads to the regression algorithm trying to find a parameter weight vector with less overall magnitude in hopes that it would prevent the algorithm to overfit on any one feature of the data. The weight applied to the ridge penalty is something prespecified by the user.

In this problem, we work with the `California Housing` dataset provided by the `sklearn` library. 8 feature attributes were obtained for each of the $n = 20,640$ houses that are included in this dataset, as well as the average house value in units of $100k. Find more details HERE.

Problem 5 (a)

Problem 5 (a)

Refer to slides in Lectures 9, in particular check the way we set up the problem for Ridge Linear Regression. In this part, determine the values for N, P, the length of the vector $\hat{\theta}$, and the dimensions of the matrix X. (Note that some of these values depend on the size of the training set, and in (a) we assume we use all examples for training. Remember that the structure of matrix X and the vectore θ should account for the bias, the intercept term.)

Problem 5 (b)

Write down the Ridge Cost Function as we defined it in lecture. Define every variable and parameter you use in this definition.

Problem 5 (c)

In class, we derived the solution for the Ridge Least Square function. We called the solution `Normal Equations.` Write down the Normal Equations solution. Then, check the dimensionalities of all terms in the Normal Equations to make sure they match for the multiplications of matrices and vectors in the Normal Equations. Explicitly include in your solutions the dimensions and show that they match. (in terms of N and P)

Problem 5 (d)

Write down the prediction equation you will use to predict the outcome. Define every variable and paramter in your equation. Also, explicitly state, in words, the quantity that you predicting for the dataset you are working on. Check the diemnsionality of the output and verify it.

Problem 5 (e)

In the code cell below, you are provided a class template called `MyRidgeLeastSquares`. Complete the `fit()` and `predict()` methods as before to train a regression algorithm for the `California Housing` dataset in `sklearn`. Use the normal solution method provide in the slides. *Don't forget to account for the intercept term in the θ vector!*

Problem 5 (f)

You are now required to use this class and practice with various values of the ridge penalty weight α using $N_{train} = 40$. What happens when α is too low? Too high? Select a value for α that works the best for you. Now taking this value, perform a robustness analysis to various amounts of training data quantities (e.g., $10, 20, 50, 80$ etc.) for the Ridge Loss. For each of those quantities, also train and report the results for the naive Least Squares algorithm you implemented in problem 1. How does the performance compare for both? For high data quantities? For low data quantities? Provide a comprehensive answer. The answer should be accompanied by clear, good-looking plots for these analysis. You are encouraged to write your own code for the analysis part. (***Note:*** For very small data sizes, you may get a singular matrix error when implementing ordinary least squares. Increase the minimum data size in this case and it should go away.)

Problem 5 (g)

Refer to the lecture 9, we cover another regularization called Least Absolute Shrinkage and Selection Operator (Lasso). What is the Lasso Cost Function as we defined it in lecture? What is the difference between Ridge and Lasso Cost Function?

Problem 5 (h)

In the lecture, we mention Lasso does not have closed-form solution as Normal Equation. Please explain why?

Solution Problem 5 (a)

Determine the values:

- N: it could be up to 20640 (number of data samples), but in the following setup, `N = 40`
- `P = 8` (features)
- Length of vector $\hat{\theta}$ = P+1 (bias) = `9`
- Dimension of matrix X = (N,P+1) = `40 x 9` array

Solution Problem 5 (b)

Least square cost function: $L(\hat{\theta}) = \frac{1}{N}(X\theta - y)^T(X\theta - y) + \frac{\gamma}{N}\theta^T\theta$

- N: numbers of sample
- θ: the regression coefficient (weight) vector
- X: input sample matrix
- y: scalar output
- γ: bias factor

Solution Problem 5 (c)

1) Normal Equation:

$\theta = (X^TX + \gamma I)^{-1}X^Ty$

2) θ's Dimension Calculation (row, columns):

- X = (N, P+1)
- $X^T X$ = (P+1, N) * (N, P+1) = (P+1, P+1)
- $(X^T X + \gamma I)^{-1} X^T$ = (P+1, P+1) * (P+1, N) = (P+1, N)
- $(X^T X)^{-1} X^T y$ = (P+1, N) * (N, 1) = (P+1, 1) = θ

Therefore, we conclude that this fit the dimenstion of θ of (P+1, 1)

Solution Problem 5 (d)

$$y = X\theta$$

- y: estimated y(output), a **"N by 1" (N, 1)=(40, 1)** matrix
- X: input sample matrix, a **"N by P+1" (N, P+1)=(40, 9)** matrix
- θ = estimated coefficients, a **"P+1 by 1" (P+1,1)=(9, 1)** vector

Solution Problem 5 (e)

```
# for part (e)

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

class MyRidgeLeastSquares:
    def __init__(self, X_train, y_train, alpha=0):
        """Function stores feature matrix and corresponding target data.

        Parameters:
        -----------
        X_train: array_like, shape(N,P)
            ndarray containing N training examples, each with D feature values.

        y_train: array_like, shape(N,1)
            ndarray containing target values for each of N examples in X_train.

        alpha: float
```

```
        float specifying the multiplier of the L2 penalty in Ridge loss"""

    self.X_train = X_train
    self.y_train = y_train
    self.alpha = alpha

def fit(self):
    """Function computes the weight vector of shape (P+1, 1) for regression"""
    # append ones for bias
    X = np.concatenate((np.ones((np.shape(self.X_train)[0], 1)), self.X_train), axis=1) ##TODO
    # print(np.shape(X))

    XTX = np.matmul( np.transpose(X) , X)
    # print(np.shape(np.linalg.inv( XTX + self.alpha*np.identity(np.shape(XTX)[1]))))

    alpha_I = self.alpha*np.identity(np.shape(XTX)[1])
    first_part = np.matmul(np.linalg.inv(XTX + alpha_I) , np.transpose(X))
    theta = np.matmul(first_part,self.y_train)

    self.theta = np.reshape(theta, [-1,1])
    # print(np.shape(self.theta))
    ##TODO

def predict(self, X_test):
    """Function predicts targets for given X_test.

    Parameters:
    -----------
    X_test: array_like, shape(N,P)
        ndarray containing N test examples with P features each.

    Returns: array_like, shape(N,1).
        ndarray containing predicted targets of shape (N,1).
    """
    # append ones for bias
    X = np.concatenate((np.ones((np.shape(X_test)[0], 1)), X_test), axis=1)  ##TODO
    # print(np.shape(X))
    y_pred = np.matmul(X, self.theta)   ##TODO

    # print(np.shape(y_pred))

    return y_pred
```

```
#-----------------Don't change anything below-----------------------#

# load dataset
cali_houses = fetch_california_housing()
X, y = cali_houses.data, cali_houses.target

X = (X - np.min(X, axis=0, keepdims=True)) / (np.max(X, axis=0, keepdims=True) - np.min(X, axis=0, keepdims=True))

# train-test split
n_train = 40
n_test = X.shape[0] - n_train

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=n_train, test_size=n_test, random_state=4803)

# train theta
LS = MyRidgeLeastSquares(X_train, y_train, alpha=0)
LS.fit()

# test
y_pred = LS.predict(X_test)

# Mycode
# print(np.shape(X_test))
# print(np.shape(y_pred))
# print(np.shape(y_test.reshape(-1, 1)))

# evaluate performance
print('MSE on Training Data: {:0.3f}'.format(np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size))
print('MSE on Test Data: {:0.3f}'.format(np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size))
```

```
MSE on Training Data: 0.267
MSE on Test Data: 4.025
```

Solution Problem 5 (f)

```
# for part (f)
```

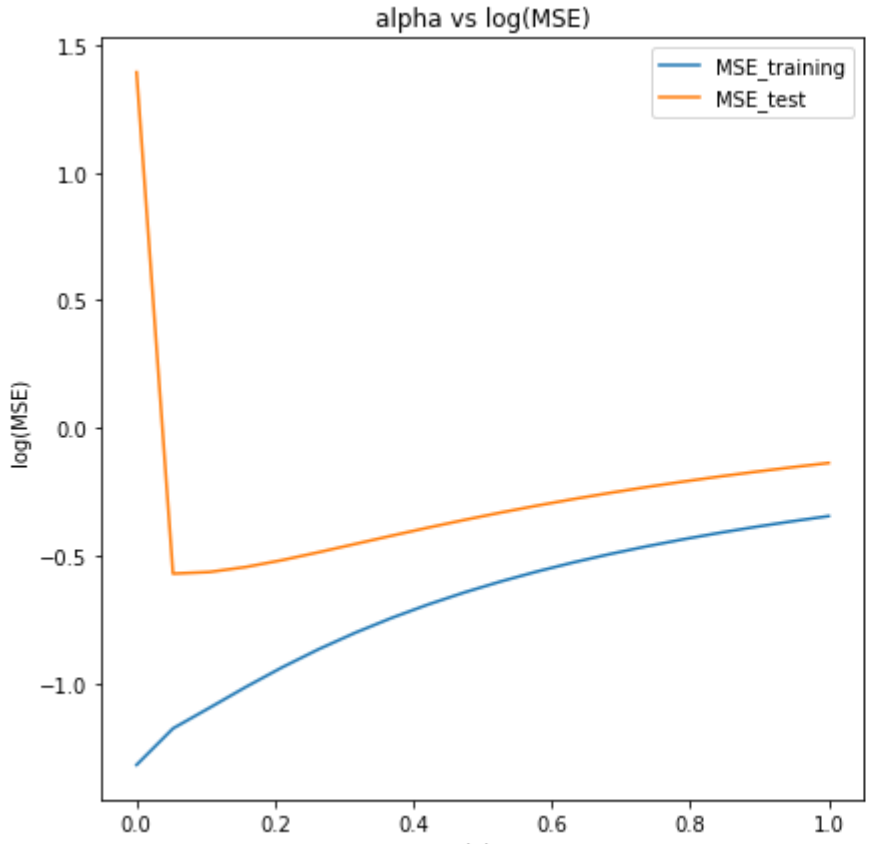
```
# set up plots
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,7))

# load dataset
cali_houses = fetch_california_housing()
X, y = cali_houses.data, cali_houses.target
X = (X - np.min(X, axis=0, keepdims=True)) / (np.max(X, axis=0, keepdims=True) - np.min(X, axis=0, keepdims=True))

# plot different alpha vs test MSE


alpha = np.linspace(0, 1, num=20)

# print(np.shape(degrees))

# MSE  array
MSE_training = np.zeros(np.shape(alpha)[0])
MSE_test = np.zeros(np.shape(alpha)[0])
counter = 0
k = 50

for n in alpha:
  current_MSE_training = 0;
  current_MSE_test = 0;

  # train-test split
  n_train = 40
  n_test = X.shape[0] - n_train

  for ii in range(k):

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=n_train, test_size=n_test, random_state=4803)

    # train theta
    LS = MyRidgeLeastSquares(X_train, y_train, alpha=n)
    LS.fit()

    # test
    y_pred = LS.predict(X_test)

    # Calculate MSE
    current_MSE_training = current_MSE_training + np.log(np.sum((LS.predict(X_train) - y_train.reshape(-1,1))**2)/y_train.size)
```

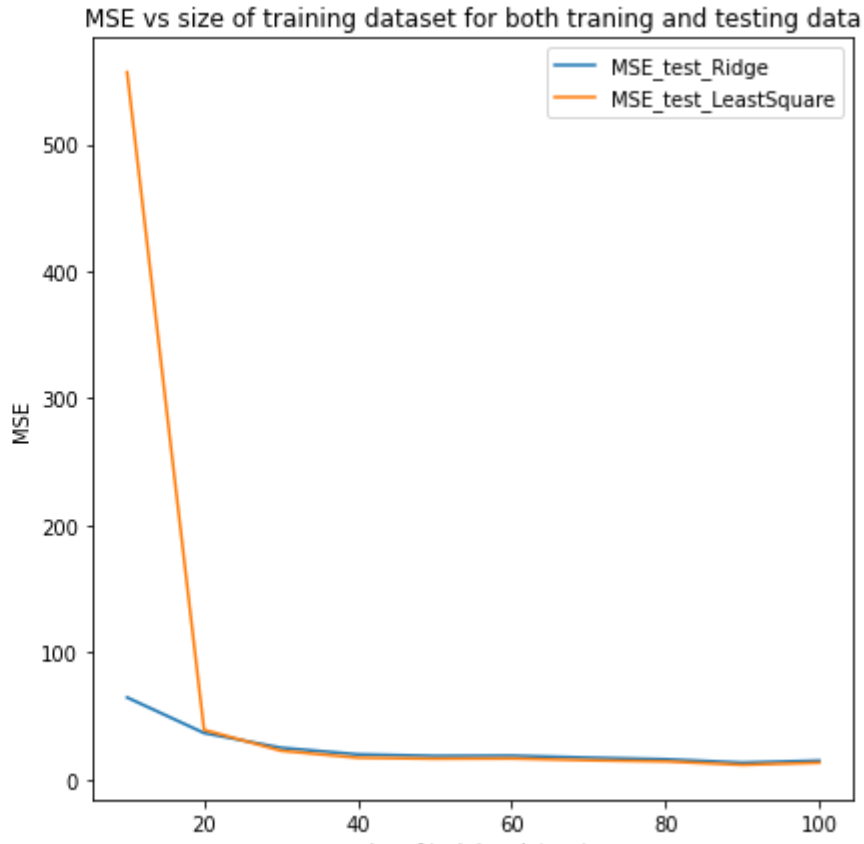
```
    current_MSE_training = current_MSE_training + np.log(np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size)
    current_MSE_test = current_MSE_test + np.log(np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size)

  # Update MSE
  MSE_training[counter] = current_MSE_training / k
  MSE_test[counter] = current_MSE_test / k

  counter = counter + 1


# PLot

ax1.plot(alpha,MSE_training, label="MSE_training")
ax1.plot(alpha,MSE_test, label="MSE_test")

# Add Title
ax1.set_title('alpha vs log(MSE)')

# Add Axes Labels

ax1.set_xlabel("alpha")
ax1.set_ylabel("log(MSE)")
ax1.legend()


# print( MSE_test)
best = alpha[1]
print("The best alpha: \n", alpha[1])


# plot different train data size vs test MSE under an optimal alpha

# load dataset
cali_houses = fetch_california_housing()
X, y = cali_houses.data, cali_houses.target

# define training set sizes
training_set_sizes = np.linspace(10, 100, 10).astype(int)

# MSE  array
MSE_training = np.zeros(np.shape(training_set_sizes)[0])
MSE_test = np.zeros(np.shape(training_set_sizes)[0])
MSE_training_no_alpha = np.zeros(np.shape(training_set_sizes)[0])
```

```
MSE_test_no_alpha = np.zeros(np.shape(training_set_sizes)[0])
current_MSE_training_no_alpha = 0
current_MSE_test_no_alpha = 0

counter = 0

k = 100

for train_size in training_set_sizes:

    # define train and test sizes
    N_train = train_size
    N_test = X.shape[0] - N_train
    # print(N_train)
    # print(N_test)

    current_MSE_training = 0;
    current_MSE_test = 0;
    current_MSE_training_no_alpha = 0
    current_MSE_test_no_alpha = 0

    for ii in range(k):

      X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=N_train, test_size=N_test)

      # Calculate with the best alpha

      # train theta
      LS = MyRidgeLeastSquares(X_train, y_train, alpha=best)
      LS.fit()

      # test
      y_pred = LS.predict(X_test)

      # Calculate MSE
      current_MSE_training = current_MSE_training + np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y_train.size
      current_MSE_test = current_MSE_test + np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size

      # Calculate with no alpha

      # train theta
      LS = MyRidgeLeastSquares(X_train, y_train, alpha=0)
      LS.fit()
```

```
      # test
      y_pred = LS.predict(X_test)

      # Calculate MSE
      current_MSE_training_no_alpha = current_MSE_training_no_alpha + np.sum((LS.predict(X_train) - y_train.reshape(-1, 1))**2)/y
      current_MSE_test_no_alpha = current_MSE_test_no_alpha + np.sum((y_pred - y_test.reshape(-1, 1))**2)/y_pred.size

    # Update MSE
    MSE_training[counter] = current_MSE_training / k
    MSE_test[counter] = current_MSE_test / k
    MSE_training_no_alpha[counter] = current_MSE_training_no_alpha / k
    MSE_test_no_alpha[counter] = current_MSE_test_no_alpha / k

    counter = counter + 1

# PLot

# ax2.plot(training_set_sizes,MSE_training, label="MSE_training_Ridge")
ax2.plot(training_set_sizes,MSE_test, label="MSE_test_Ridge")
# ax2.plot(training_set_sizes,MSE_training_no_alpha, label="MSE_training_LeastSquare")
ax2.plot(training_set_sizes,MSE_test_no_alpha, label="MSE_test_LeastSquare")

# Add Title
ax2.set_title('MSE vs size of training dataset for both traning and testing data')

# Add Axes Labels

ax2.set_xlabel("size of training dataset")
ax2.set_ylabel("MSE")
ax2.legend()

# Display

plt.show()
```

```
The best alpha:
 0.05263157894736842
```





First, from the plot of alpha vs log(MSE), we can see

- If α is too small, it will be similar to the performance of least square linear regression (without regularization), and the problem of overfitting will still occur.
- If α is too big, it will over regularized (or over-fit) and lead to increase in error.

From the plot of robustness analysis of Ridge penalty vs Least square, we can see:

- At the very small training dataset, the Least square without penalty term will over fit and lead to a way higher MSE comparing to using Ridge penalty.
- For large-sized training dataset, two models performs similarly.

Solution Problem 5 (g)

Lasso's cost function:

$L(\hat{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(x_i\theta - y_i)^2 + \gamma\sum_{i=1}^{P}|\theta_i|$

The difference between Ridge and Lasso Regularization is their penalty term,
where Ridge has $\frac{\gamma}{N}\sum_{i=1}^{P}\theta_i^2$ and Lasso has $\gamma\sum_{i=1}^{P}|\theta_i|$

Solution Problem 5 (h)

Lasso does not have closed-form solution as Normal Equation since the L1 regularization term $\gamma\sum_{i=1}^{P}|\theta_i|$ is not differentiable at $\theta_j = 0$.