Technique Assignment 3: Linear regression

Cogs 109 Fall 2020

Xing Hong

A15867895

In [285…

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

1. (15 points) Datasets and variables

Find a dataset from the UCI machine learning repository that is suitable for linear regression. Provide a link to your chosen dataset and briefly describe its content.

Link: http://archive.ics.uci.edu/ml/datasets/Student+Performance# The data approach student achievement in secondary education of two Portuguese schools. The data attributes include student grades, demographic, social and school related features. Particularly, the attribute G3 (final year grade) has a strong correlation with attributes G2 and G1.

(9) List the following:

- number of variables : 33
- number of samples : 649
- labels (what is the label?) : sex in F or M, father's job and mother's job with many labels, absences number, etc. The most important ones are the grades seperated with labels of G1, G2, G3 -- which are first, second, third period grade (Math or Portuguese).

(6) Create and report a research question that you could answer using this dataset and some or all of the variables. Variables: The number of absences of different students, the Math G3 grades of them respectively. Question: whether there is a correlation between the absences of students and the Math grade in their final year. If so, what pattern will it be? Method: By building linear regression models, comparing the MES of different regression models, applying cross validation,and eventually plot the regression line of best scored model on scatter plot, we can see the correlation between absence and final year grade of students.

2. (20) Arrays and numpy

This tutorial should be very helpful: https://www.numpy.org/devdocs/user/quickstart.html

a. **(10)**

Use arange() and reshape() from numpy to create a 4x5 array containing the integers 1 through 20. Append a column of ones to the left of your array to create a 4x6 array. Multiply every element of the array by 2. Print **only** the resulting array.

In [286…

```
a = np.arange(1,21).reshape(4,5)
b = np.ones(4)
c = np.c_[b,a]*2
print(c)
```

```
[[ 2.  2.  4.  6.  8. 10.]
 [ 2. 12. 14. 16. 18. 20.]
 [ 2. 22. 24. 26. 28. 30.]
 [ 2. 32. 34. 36. 38. 40.]]
```

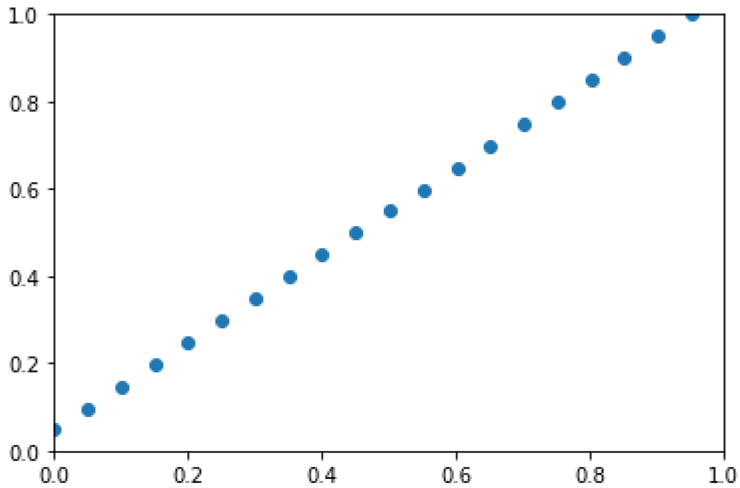
b. (10)

Use linspace() and reshape() to create a 20 x 20 array that contains a smooth range of values between 0 and 1, inclusive.

Create a scatter plot using the first (leftmost) column of your array as the x values and the last (rightmost) column of your array as the y values. Use x limits 0 and 1 and y limits 0 and 1 for your plot.

In [287…

```
d = np.linspace(0,1,num=400).reshape(20,20)
x = d[:,0]
y = d[:,-1]
plt.scatter(x,y)
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.show()
```



3. (40 points) Univariate linear regression

Note: Solutions to this problem must follow the method described in class and the linear regression handout. There is some flexibility in how your solution is coded, but you may not use special functions that automatically perform linear regression for you.

Load in the BodyBrainWeight.csv dataset. Perform linear regression using two different models:

M1: brain_weight = w0 + w1 x body_weight

M2: brain_weight = w0 + w1 x body_weight + w2 x body_weight^2

a. (15)

For each model, follow the steps shown in class to solve for w. Report the model, including w values and variable names for both models.

b. (10)

Use subplots to display two graphs, one for each model. In each graph, include:

- Labeled x and y axes
- Title
- Scatterplot of the dataset
- A smooth line representing the model

c. (10)

For each model, calculate the sum squared error (SSE). Show your 2 SSE values together in a bar plot.

c. (5)

Which model do you think is better? Why? Is there a different model that you think would better represent the data?

In [288…

```
## Load the dataset and extract Brain weight as X and Body weight as Y

bbdata = pd.read_csv("BodyBrainWeight.csv").values

X = bbdata[:,0]
Y = bbdata[:,1]

# This is good for debugging
X.shape
```

Out[288… (46,)

In [289…

```
## Create A, the augmented data array

ones = np.ones(46)
```

```
A1 = np.c_[ones,X]

## Solve for w, the weight vector

w1 = np.linalg.lstsq(A1,Y,rcond=None)[0]
print("Model M1: ")
print("Brain weight = {0} + {1}*body_weight".format(w1[0],w1[1]))
```

```
Model M1:
Brain weight = 124.92810522680392 + 0.9370390990165334*body_weight
```

In [290…

```
## Create A, the augmented data array

ones = np.ones(46)
squares = X**2

A2 = np.c_[ones,X,squares]

## Solve for w, the weight vector
w2 = np.linalg.lstsq(A2,Y,rcond=None)[0]
print("Model M2:")
print("Brain weight = {0} + {1}*body_weight + 
{2}*body_weight^2".format(w2[0],w2[1],w2[2]))
```

```
Model M2:
Brain weight = 49.09136906591356 + 1.5904622228726575*body_weight + -0.0001086
9568419802459*body_weight^2
```

In [291…

```
## Create a smooth set of X values for plotting the model
X_line = np.linspace(0,6700,100000)

## Send the X values for plotting through the linear model
Y_line_M1 = w1[0] + w1[1]*X_line
Y_line_M2 = w2[0] + w2[1]*X_line + w2[2]*(X_line**2)
```

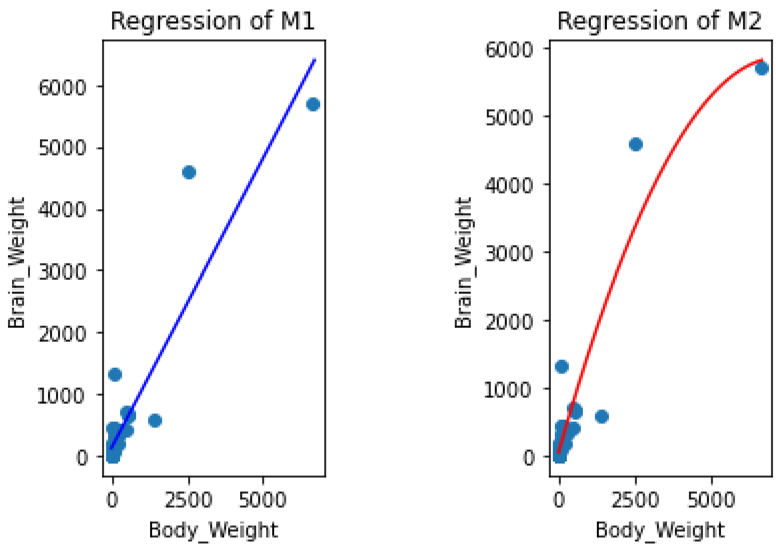
In [292…

```
## Plot the data along with the model
plt.subplot(1,2,1)
plt.scatter(X,Y)
plt.plot(X_line,Y_line_M1,'b')
plt.xlabel('Body_Weight')
plt.ylabel('Brain_Weight')
plt.title('Regression of M1')

plt.subplot(1,2,2)
plt.scatter(X,Y)
plt.plot(X_line,Y_line_M2,'r')
```

```
plt.xlabel('Body_Weight')
plt.ylabel('Brain_Weight')
plt.title('Regression of M2')
plt.subplots_adjust(wspace =1)
plt.show # This lets you plot multiple inputs on the same graph
```

Out[292…

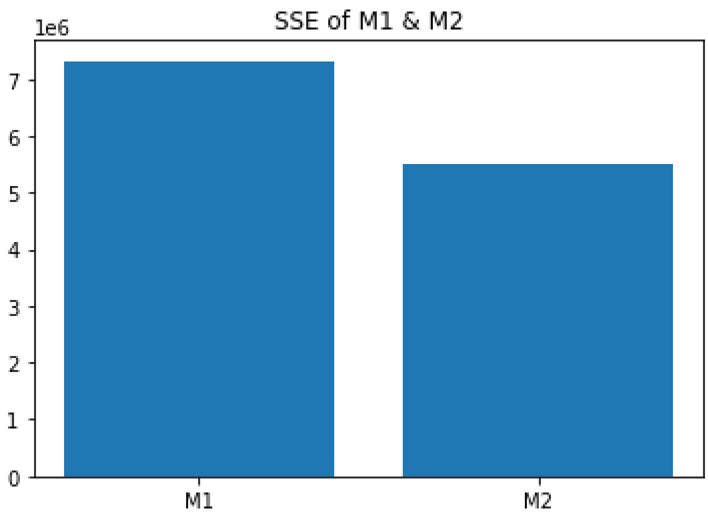
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [293…

```
Y_SSE_M1 = w1[0] + w1[1]*X
Y_SSE_M2 = w2[0] + w2[1]*X + w2[2]*(X**2)
SSE_M1 = sum((Y_SSE_M1 - Y)**2)
SSE_M2 = sum((Y_SSE_M2 - Y)**2)
print(SSE_M1, SSE_M2)

plt.bar(['M1', 'M2'], [SSE_M1, SSE_M2])
plt.title('SSE of M1 & M2')
plt.show()
```

```
7317401.2702330705 5497572.515005937
```



Based on the SSE of two regression models, I think M2 is more accurate, since it's SSE is lower, which means that the error of prediction is lower. logarithm function as

predictor may also have a good performance.

Outliers

Note: Since a few intances in the data could be considered as outlier, and I wasn't sure if we can remove these instances for this project.(data cleaning) Therefore I decide to do all the above without the outliers.

In [294…

```
array_new = bbdata[bbdata[:,0] < 600, :]
array_new = array_new[array_new[:,1] < 1000, :]
X = array_new[:,0]
Y = array_new[:,1]
X.shape
```

Out[294…

```
(42,)
```

In [295…

```
ones = np.ones(42)
A1 = np.c_[ones,X]
w1 = np.linalg.lstsq(A1,Y,rcond=None)[0]
print("Model M1: ")
print("Brain weight = {0} + {1}*body_weight".format(w1[0],w1[1]))
```

```
Model M1:
Brain weight = 59.975815588616065 + 1.2092588736819594*body_weight
```

In [296…

```
ones = np.ones(42)
squares = X**2
A2 = np.c_[ones,X,squares]
w2 = np.linalg.lstsq(A2,Y,rcond=None)[0]
print("Model M2:")
print("Brain weight = {0} + {1}*body_weight + 
{2}*body_weight^2".format(w2[0],w2[1],w2[2]))
```

```
Model M2:
Brain weight = 37.53376736292012 + 2.113175362433763*body_weight + -0.00190527
87879908107*body_weight^2
```

In [297…

```
X_line = np.linspace(0,600,10000)


Y_line_M1 = w1[0] + w1[1]*X_line
Y_line_M2 = w2[0] + w2[1]*X_line + w2[2]*(X_line**2)
```

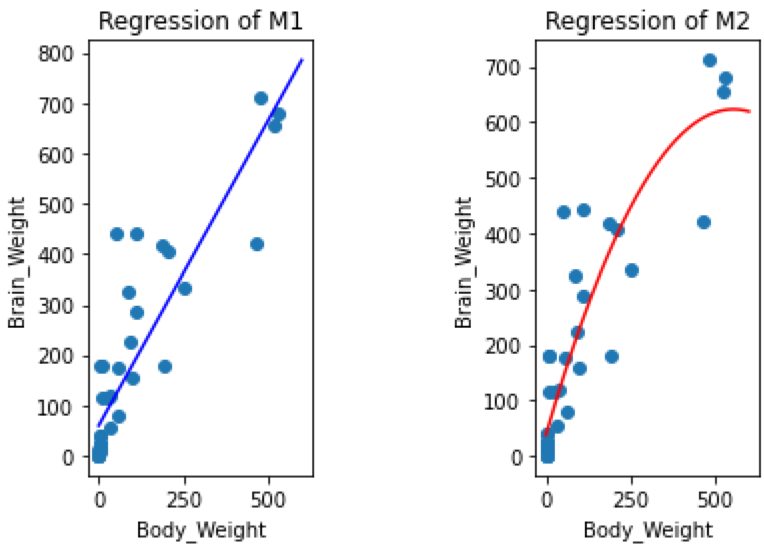
In [298…

```
## Plot the data along with the model
plt.subplot(1,2,1)
plt.scatter(X,Y)
plt.plot(X_line,Y_line_M1,'b')
```

```
plt.xlabel('Body_Weight')
plt.ylabel('Brain_Weight')
plt.title('Regression of M1')

plt.subplot(1,2,2)
plt.scatter(X,Y)
plt.plot(X_line,Y_line_M2,'r')
plt.xlabel('Body_Weight')
plt.ylabel('Brain_Weight')
plt.title('Regression of M2')
plt.subplots_adjust(wspace =1)
plt.show # This lets you plot multiple inputs on the same graph
```

Out[298…

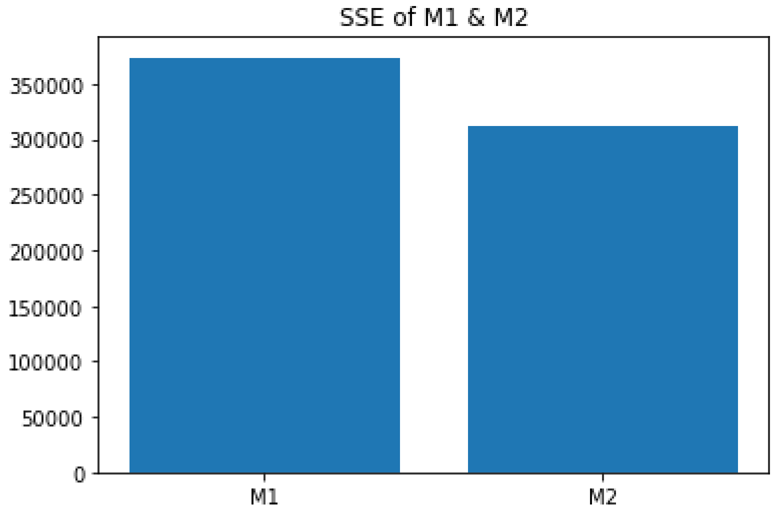
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [299…

```
Y_SSE_M1 = w1[0] + w1[1]*X
Y_SSE_M2 = w2[0] + w2[1]*X + w2[2]*(X**2)
SSE_M1 = sum((Y_SSE_M1 - Y)**2)
SSE_M2 = sum((Y_SSE_M2 - Y)**2)
print(SSE_M1, SSE_M2)

plt.bar(['M1', 'M2'], [SSE_M1, SSE_M2])
plt.title('SSE of M1 & M2')
plt.show()
```

```
373889.5726862877 312188.95500153996
```



Which leads to conclusion, without outliers, M2 is still better then M1 for the reason of lower SSE. Without outliers, Logarithm can be a better model which worth to try.

4. (25 points) Multivariate linear regression with cross validation

Using the dataset found in Housing.csv, build a multivariate model to predict house price using lot size and the number of bedrooms as predictors.

Hint: You may use this as your model:

Price = w0 + w1 x Lot size + w2 x Bedrooms

First, split your data into a training set (80%) and a test set (20%). Then perform linear regression using the **training data** only. Report your model and show the mean squared error (MSE) for your **training** and **test** data using a bar graph.

MSE can be found by dividing SSE by the number of samples in your data.

In [300…

```
from sklearn.model_selection import train_test_split
```

In [270…

```
## Load the housing data

df = pd.read_csv("Housing.csv")
print('df shape', df.shape)
# Extract the variables

Y = df['price'].values
X1 = df['lotsize'].values
X2 = df['bedrooms'].values
print(X1.shape, X2.shape, Y.shape)

ones = np.ones(546)
X_array = np.c_[ones,X1,X2]
X_train, X_test, Y_train, Y_test = train_test_split(X_array, Y,
```

```
test_size=0.2, random_state=0)
print (X_train.shape, Y_train.shape)
print (X_test.shape, Y_test.shape)
```

```
df shape (546, 13)
(546,) (546,) (546,)
(436, 3) (436,)
(110, 3) (110,)
```

In [271…

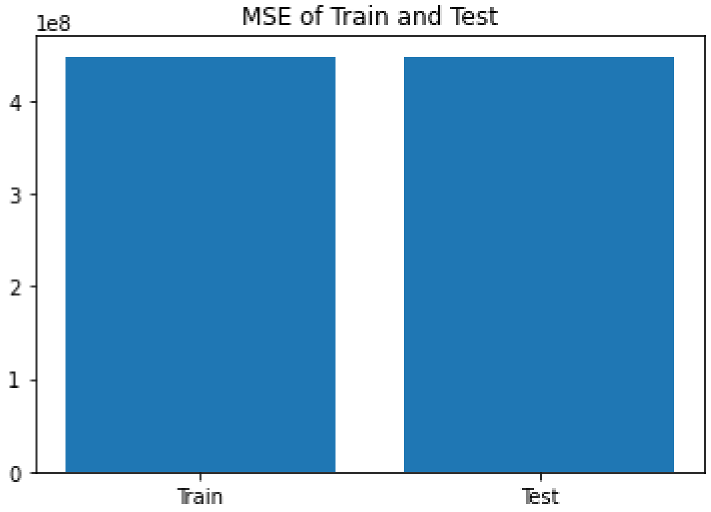
```
w3 = np.linalg.lstsq(X_train,Y_train,rcond=None)[0]
print("Model multi:")
print("Price = {0} + {1}*Lot_size +
{2}*Bedrooms".format(w3[0],w3[1],w3[2]))
```

```
Model multi:
Price = 4995.926608045902 + 6.275451573294228*Lot_size + 10471.848450761476*Be
drooms
```

In [272…

```
#MSE for train
Y_SSE_M3_train = w3[0] + w3[1]*X_train[:,1] + w3[2]*X_train[:,2]
MSE_M3_train = sum((Y_SSE_M3_train - Y_train)**2)/436

#MSE for test
Y_SSE_M3_test = w3[0] + w3[1]*X_test[:,1] + w3[2]*X_test[:,2]
MSE_M3_test = sum((Y_SSE_M3_test - Y_test)**2)/110

print(MSE_M3_train, MSE_M3_test)

plt.bar(['Train', 'Test'], [MSE_M3_train, MSE_M3_test])
plt.title('MSE of Train and Test')
plt.show()
```

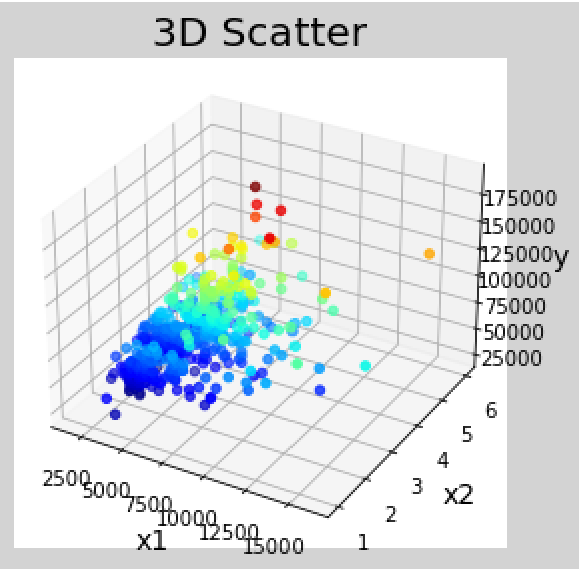
```
448450630.68091935 448612842.5056583
```



Extra

In [273…
```
from mpl_toolkits.mplot3d import axes3d
```

In [274…
```
plt.figure("3D Scatter", facecolor="lightgray")
ax3d = plt.gca(projection="3d")
plt.title('3D Scatter', fontsize=20)
ax3d.set_xlabel('x1', fontsize=14)
ax3d.set_ylabel('x2', fontsize=14)
ax3d.set_zlabel('y', fontsize=14)
plt.tick_params(labelsize=10)

d = np.sqrt(X1 ** 2 + X2 ** 2 + Y ** 2)
ax3d.scatter(X1, X2, Y, s=20, c=d, cmap="jet", marker="o")
plt.tight_layout()
```



From the graph, I can see a rise of Y(price) with a tendency of flattening at the end when X2(bedrooms) are more and relatively unchanged when X1(Lot) getting bigger. So I will try Logarithm grow on X2.

In [282…
```
X_array = np.c_[ones,X1, np.log(X2)]
X_train, X_test, Y_train, Y_test = train_test_split(X_array, Y,
test_size=0.2, random_state=0)
```

In [283…
```
w4 = np.linalg.lstsq(X_train,Y_train,rcond=None)[0]
print("Model multi-2:")
print("Price = {0} + {1}*Lot_size +
{2}*Bedrooms^2".format(w4[0],w4[1],w4[2]))
```

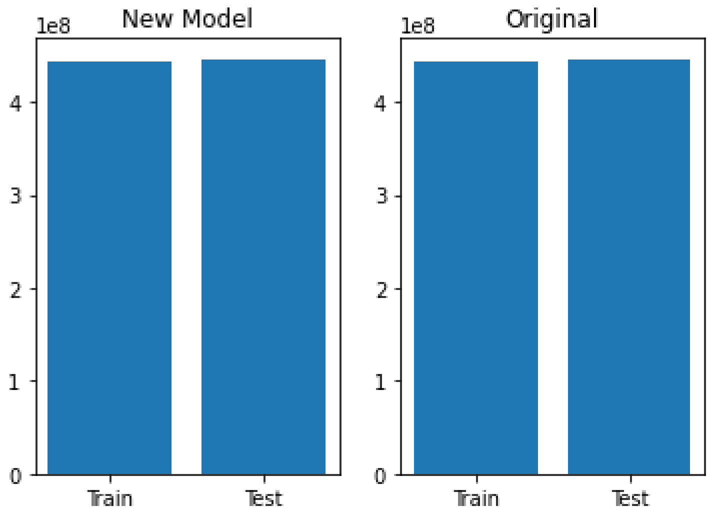
```
Model multi-2:
Price = 2751.991886079015 + 6.221962229880475*Lot_size + 31865.177384041413*Be
drooms^2
```

In [284…
```
#MSE for train
```

```python
Y_SSE_M4_train = w4[0] + w4[1]*X_train[:,1] + w4[2]*X_train[:,2]
MSE_M4_train = sum((Y_SSE_M4_train - Y_train)**2)/436

#MSE for test
Y_SSE_M4_test = w4[0] + w4[1]*X_test[:,1] + w4[2]*X_test[:,2]
MSE_M4_test = sum((Y_SSE_M4_test - Y_test)**2)/110

print("New Model:", MSE_M4_train, MSE_M4_test)
print("Original: ", MSE_M3_train, MSE_M3_test)

plt.subplot(1,2,1)
plt.bar(['Train', 'Test'], [MSE_M4_train, MSE_M4_test])
plt.title('New Model')

plt.subplot(1,2,2)
plt.bar(['Train', 'Test'], [MSE_M4_train, MSE_M4_test])
plt.title('Original')
```

```
New Model: 442474788.73482835 446321796.54369694
Original:  448450630.68091935 448612842.5056583
```

Out[284… Text(0.5, 1.0, 'Original')



In [ ]: