QBUS6850: Tutorial 3 – **Practicing Linear Regression**

Objectives

- To learn how linear regression is implemented;
- To learn how to use linear regression model from scikit;

1. Linear Regression Manually

In the first task of this tutorial, you have a chance to work out a python program for linear regression yourself. You will start with the linear regression solution formula

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

where $\boldsymbol{X}$ is the data matrix.

Step 1: Let's get some data to play with. Write the following code:

```
from sklearn.datasets.samples_generator import make_regression

N = 20

x, y, true_coef = make_regression(n_samples = N, n_features =
1, noise=20, coef=True)
```

Step 2: Now by using variable explorer, check the value of variables x and y, and true_coef as well. Write down the size of x, y, and true_coef. Is this x a matrix, or two dimensional array? From this information, can you guess what the linear model is.

Now you can even visualize the data by using the following code:

```
import matplotlib.pyplot as plt

# Scatter plot
plt.figure()
plt.scatter(x, y)
```

Step 3: In lecture, you were told that in order to build a linear regression model with the intercept, we shall expand the data matrix to the one with all 1's in its first column. Write code first this? Or your tutor will tell you how to do this

```
# Write your code to make a new X such that the first column
is all 1's and other column(s) from the current x

or ask your tutor(s)
```

Suppose you have correctly produced the new data matrix $\boldsymbol{X}$ (the first column all 1s), now add the following code to your program

```
import numpy as np

X = np.asmatrix(X)
```

The meaning of the second statement is to convert a python array to a matrix type in numpy. Thus we can use matrix operations in numpy easier. If you are familiar with python array, this step is not necessary.

Step 4: When this ***X*** is a matrix, we can "translate" the solution easily to the python statement as:

```
# Estimate linear regression coefficients
lin_betas = np.linalg.inv(X.T*X) * X.T * y.reshape(N,1)
```

If you compare the above statement with the solution formula, you will find how we get the inverse of a matrix, and how we change the (1D) array y into a column vector. Can you identify these in the statement?

Step 5: As this is a simple linear regression, there are two parameters beta(s), that is beta_0 (intercept) and a coefficient beta_1. They have been given in the variable lin_betas. You can check them by using the following code:

```
# beta_0
lin_intercept = lin_betas[0,0]
print("intercept (beta_0): {0:.2f}".format(lin_intercept))

# beta_1
lin_beta = lin_betas[1,0]
print("beta_1: {0:.2f}".format(lin_beta))
```

Step 6: Now we have done the linear regression. How can we use this model to predict y for a new x. Here is an example

```
x_1 = 1

prediction = lin_intercept + lin_beta * x_1
print("Predicted value at x = {0}: {1:.2f}".format(x_1, 
prediction))
```

The new x is x_1=1. In the second statement above, you must clearly write out your linear model with the learned intercept value and coefficient value. This will give the predicted value.

2. Linear Regression and Prediction

In task 1, we managed work the linear regression solution manually. Although it is tedious, it is still quite easy to modify the program for other datasets.

In this second task, you will learn how to use a python machine learning package sklearn to do regression. You can leave all the chores to the package such as making matrix, calling inverse operation, preparing model for prediction etc.

Step 1: Linear regression model can be loaded by using the following code:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
```

You shall follow the third statement to load LinearRegression model. Before we use the model, we need prepare data. This time we load data from a data file.

```
# Load data
clock_auction_df = pd.read_csv("auction.txt",
delimiter="\t")
#Get some info of the data
clock_auction_df.info()
clock_auction_df.head()
clock_auction_df.describe()
```

We assume everyone understands the above statement. Ask yourself a question: what is the data type of clock_auction_df? How many columns, and what are they?

We would like to build a linear regression to regress the price variable over others. Given that we are going to use linear regression model, we certainly need make sure the target variable price actually has some linear relation with some or all features. This process is called Exploratory Data Analysis (EDA).

Step 2: Looking at the correlation is one of wonderful EDA ways to start with. This can be easily done by using the pandas package. Now type the following into your program

```
clock_auction_df.corr()
```

This statement helps you check linear correlation between variables. You will see that variable Age has stronger linear association with Price than Bidders. Why?

Hence we fit a model Price (t) ~ Age (x).

Now we start prepare data X and t. They should be taken from the DataFrame variable clock_auction_df

```
x_data = np.reshape(clock_auction_df["Age"].values,
(len(clock_auction_df), 1))

y_data = np.reshape(clock_auction_df["Price"].values,
(len(clock_auction_df), 1))
```

Now data has been prepared. It is time to define a linear regression model. This is the first example from which you can learn how to use sklearn package to do machine learning. We always start with the model you want.

Step 3: Define the Linear Regression model:

```
# Create the linear regression object
lr_obj = LinearRegression()
```

Fit (or train) the model by using the following code:

```
# Estiamte coefficients
lr_obj.fit(x_data, y_data)
```

Step 4: Predict the test data and get the predicted value:

```
# Predict the sale price of a 121 year old clock
age = 121     # new data age 121

predicted_price = lr_obj.predict(age)

print("Estimated Sale Price:
${0:.2f}".format(predicted_price[0, 0]))
```

When you have finished the above steps, please address the following questions:

(1) In this workflow, have you explicitly seen the algorithm (like our formula for beta or gradient descent algorithm)?

(2) Did you prepare the special column of all 1s if we want to estimate the intercept?

(3) Did you see the estimated intercept beta_0 and coefficient beta_1 as what we did in the first task?

(4) Did we actually write out the fitted linear model?

Below is the sample answers to the above questions:

(1) All the algorithms (actually either the formula or gradient descent) are hidden from us by simply calling model’s “fit” function to data. Leave us alone with preparing data only. So preparing data becomes our major tasks when we do machine learning with sklearn.

(2) This time, we don't need prepare the special data column of all 1s. The fit method handles it implicitly. But you have opportunity to tell the model whether you need the intercept by using

```
# Create the linear regression object with intercept
# or without intercept
lr_obj = LinearRegression(it_intercept=True)
#
lr_obj = LinearRegression(it_intercept=False)
```

(3) sklearn hides all the model information in the model object variable lr_obj. If we really wish to see what the estimated beta_0 and beta_1 are. You may use the following statements:

```
# For the intercept value
print(lr_obj.intercept_)
# For other beta
print(lr_obj.coef_)
```

(4) No. We never write the model. To do model prediction, we simply call lr_obj's predict function on the new data.

3. Linear Regression Gradient Descent Manually

In this task, we will build our own Gradient Descent algorithm for linear regression.

Import package numpy and matplotlib.pyplot:

```
import numpy as np
import matplotlib.pyplot as plt
```

Step 1: Define the algorithm. We now build a gradient descent function, name it as "Gradient_Descent_Algo": The function will take feature data x, target data y, the initial guessed coefficients beta, learning rate alpha, number of data N, and the number of iteration numIterations. The function will return the estimated coefficient beta. The function is defined as follows (please copy into your program)

```
"""
Build the gradient descent function

"""
# N denotes the number of training examples here,
# not the number of features
def Gradient_Descent_Algo(x, y, beta, alpha, N,
numIterations):
    xTrans = x.transpose()
    for i in range(0, numIterations):
        # predicted values from the model
        model_0 = np.dot(x, beta)
        loss_temp = model_0 - y
        # calculte the loss function
        loss = np.sum(np.square(loss_temp)) / (2 * N)
        # save all the loss function values at each step
        loss_total[i]= loss
        #You can check iteration by
        #print("Iteration: {0} | Loss fucntion: {1}".format(i, 
loss))
        # calcualte the gradient using matrix representation
        gradient = np.dot(xTrans, loss_temp) / N
        # update the parameters simulteneously with learning 
rate alpha
        beta = beta - alpha * gradient
        # save all the estimated parametes at each step
        beta_total[i,:]= beta.transpose()
    return beta
```

Questions:

(1) Find out which statements are implementing the gradient descent updating rule?

(2) Which statement gives the derivatives of loss function with respect to the coefficients (in its matrix form). Please refer to Lecture 2 slides

Step 2: Generate synthetic data from a linear model by using the following assumption:

$$f(\boldsymbol{x};\boldsymbol{\beta}) = \beta_0 + \beta_1 x$$

where $\beta_0=4$, $\beta_1=1.5$.

Then add random noise to the $f(\boldsymbol{x};\boldsymbol{\beta})$, that is:

$$t = f(\boldsymbol{x};\boldsymbol{\beta}) + \varepsilon$$

where ε is generated from a normal distribution with mean=0, standard deviation = 0.1.

Produce a scatter plot of $\boldsymbol{t}$ against $\boldsymbol{x}$.

```
# Initialise RNG to generate the same random numbers each time
np.random.seed(0)

m = 50 #number of training examples
x = np.linspace(0.0, 1.0, m)

# Function true coefficients/parameters
beta0 = 4
beta1 = 1.5

# true values from linear model
f = beta0 + beta1 * x
# Add noisy
sigma2 = 0.1
```

```
y = f + np.random.normal(0, np.sqrt(sigma2), m)

# reshape
y = np.reshape(y, (len(y), 1))

x_data_1 = np.reshape(x, (len(x), 1))
x = np.column_stack((np.ones(len(x)), x_data_1))

fig0 = plt.figure()
plt.scatter(x[:,1],y)
```

Step 3: Fitting model by calling the function "Gradient_Descent_Algo", (with learning rate *α*=0.0005 and training iterations numIterations =10,000).

```
"""
Add your code to call Gradient_Descent_Algo function for this
data set.  Think about what you need

Ask tutors for helps if needed

"""
```

Step 4: Visualise your results by using the following code:

```
fig1 = plt.figure()
plt.plot(loss_total, label = "Loss fucntion")
plt.plot(beta_total[:,0], label = "Beta0")
plt.plot(beta_total[:,1], label = "Beta1")
plt.legend(loc="upper right")
plt.xlabel("Number of iteration")
#fig1
```