NYCU Pattern Recognition, Homework 1

311552034, 林柏翰

Part. 1, Coding (70%):

You should type the answer and also screenshot at the same time. Otherwise, no points will be given. The screenshot and the figures we provided below are just examples. **The results below are not guaranteed to be correct.** Please convert it to a pdf file before submission. You should use English to answer the questions. After reading this paragraph, you can delete it.

1. (0%) Show the learning rate and epoch you choose
   learning rate: _0.019__
   epoch: __120000_

```
batch_size = x_train.shape[0]

# TODO
# Tune the parameters
# Refer to slide page 9
lr = 0.0019
epochs = 120000

linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
```

2. (5%) Show the weights and intercepts of your linear model.
   weights: _380__
   intercepts: _1382__

```
print("Intercepts: ", linear_reg.weights[-1])
print("Weights: ", linear_reg.weights[:-1])
```

```
Intercepts:  [1382.20059469]
Weights:  [[380.14513999]]
```

3. (5%) What's your final training loss (MSE)?
   training loss (MSE): _139562065.4875609__

```
print('training loss: ', linear_reg.evaluate(x_train, y_train))
```

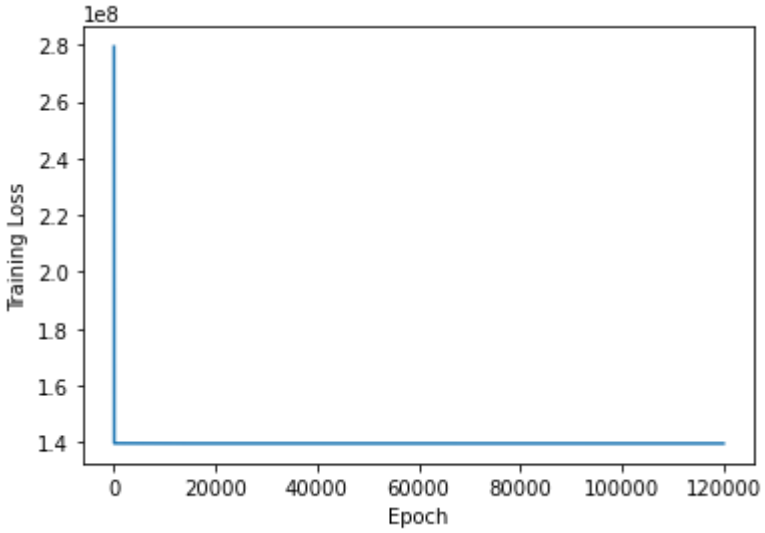
```
training loss:  139562065.4875609
```

4. (5%) What's the MSE of your validation prediction and validation ground truth?
   validation loss (MSE): _136920237.86230916__

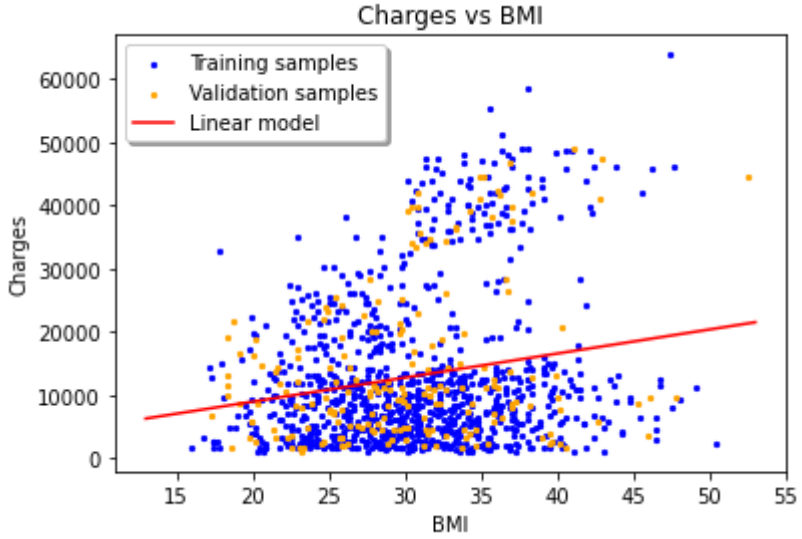
```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```

```
validation loss:  136920237.86230916
```

5. (5%) Plot the training curve. (x-axis=epoch, y-axis=loss)



6. (5%) Plot the line you find with the training and validation data.



7. (0%) Show the learning rate and epoch you choose.
   learning rate: _0.0006__
   epoch: _1000000__

```
    batch_size = x_train.shape[0]

    # TODO
    # Tune the parameters
    # Refer to slide page 10
    lr = 0.0006
    epochs = 1000000

    linear_reg = LinearRegression()
    linear_reg.fit(x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
```

✓ 30.4s

```
Don't cheat.
```

8. (10%) Show the weights and intercepts of your linear model.
   weights: ___
   intercepts: ___

```
    print("Intercepts: ", linear_reg.weights[-1])
    print("Weights: ", linear_reg.weights[:-1])
```

✓ 0.3s

```
Intercepts:  [-11857.05691462]
Weights:  [[  259.85086214]
 [ -383.54527255]
 [  333.33250502]
 [  442.55747113]
 [24032.220977  ]
 [ -416.01439469]]
```

9. (5%) What's your final training loss?
   training loss (MSE): ___

```
    print('training loss: ', linear_reg.evaluate(x_train, y_train))
```

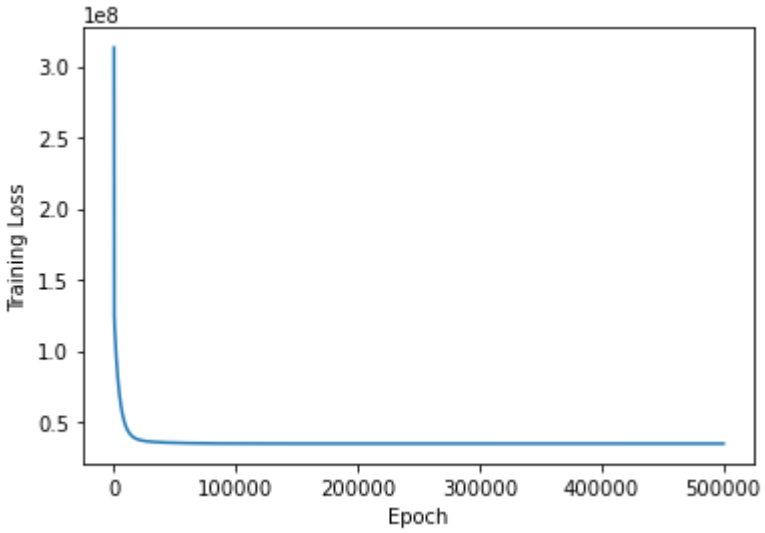
```
training loss:  34697170.25663662
```

10. (5%) What's the MSE of your validation prediction and validation ground truth?
validation loss (MSE): ___

```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```

```
validation loss:  41958541.91707702
```

11. (5%) Plot the training curve. (x-axis=epoch, y-axis=loss)



12. (20%) Train your own model and fill the testing CSV file as your final predictions.

learning rate: _0.00000027__
epoch: _10000000__
batch_size: _938__
Used features: _age , bmi ,smoker__

First, I utilized df_train.corr() to figure out the correspondence of each feature. The result as follow:

	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.053253	0.105642	0.033802	-0.004509	-0.006884	0.324213
sex	-0.053253	1.000000	0.015213	0.023992	0.068751	-0.002937	0.025908
bmi	0.105642	0.015213	1.000000	0.025009	-0.005701	0.141745	0.189973
children	0.033802	0.023992	0.025009	1.000000	-0.012050	0.000552	0.048509
smoker	-0.004509	0.068751	-0.005701	-0.012050	1.000000	-0.012810	0.789616
region	-0.006884	-0.002937	0.141745	0.000552	-0.012810	1.000000	-0.026971
charges	0.324213	0.025908	0.189973	0.048509	0.789616	-0.026971	1.000000

Therefore, I decided to use features 'age', 'bmi', and 'smoker' to train my model. To find out the best performance can be reached using these features. I use the

LinearRegression from sklearn.linear_model to verify the performance in advance. The result as follow:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
y_train = y_train.reshape( (  y_train.shape[0] , 1) )
model.fit(x_train, y_train)

print("Model slope:    ", model.coef_)
print("Model intercept:", model.intercept_)

y_pred = model.predict( x_train )
y_train = y_train.reshape( (  y_train.shape[0] , 1) )
loss = np.mean( np.square( y_pred - y_train )  )
print('training loss: ', loss)

y_pred_val = model.predict( x_val )
y_val = y_val.reshape( (  y_val.shape[0] , 1) )
loss_val = np.mean( np.square( y_pred_val - y_val )  )
print('validation loss: ', loss_val )
```

✓ 0.3s

```
Model slope:     [[  262.47945427   323.48891417 23997.45835145]]
Model intercept: [-12006.42140669]
training loss:  35216032.9029364
validation loss:  42125448.102460936
```

We can find that it's impossible to beat the baseline using only three features with the linear regression model assigned by TAs. So I need to create more features around the three features I chose. I used the polynomial expansion to create 6 more features. Beside the normal feature the sklearn.preprocessing PolynomialFeatures would create. I also create more features around 'smoker' because it has the highest correspondence to 'charges' . The code is as follow:

```
def poly_expand(dataframe):
    dataframe['age^2']=dataframe['age']**2
    dataframe['age bmi']=dataframe['age']*dataframe['bmi']
    dataframe['age smoker']=dataframe['age']*dataframe['smoker']
    dataframe['bmi^2']=dataframe['bmi']**2
    dataframe['bmi smoker']=dataframe['bmi']*dataframe['smoker']
    dataframe['smoker^2']=dataframe['smoker']**2
    dataframe['age^2 smoker']=dataframe['age']**2*(dataframe['smoker'])
    dataframe['bmi^2 smoker']=dataframe['bmi']**2*(dataframe['smoker'])
    dataframe['age*bmi*smoker']=dataframe['bmi']*dataframe['age']*dataframe['smoker']
    return dataframe
```

After verifying the performance of expanded features with sklearn, I found that the we can beat the baseline with expanded features :

```
Model slope:     [[ 1.15360221e+02  3.41810423e+02 -1.11390646e+04  2.40454923e+00
  -1.02851307e+00 -2.15231405e+01 -4.72190797e+00  1.62655082e+03
  -1.11390646e+04 -2.39101592e+00 -6.42789732e+00  6.18589961e+00]]
Model intercept: [-5063.12576823]
training loss:  21847038.619785804
validation loss:  27685409.285232175
```

And my remaining job is to adjust the parameters to beat the baseline.

```
    batch_size = x_train.shape[0]

    # TODO
    # Tune the parameters
    # Refer to slide page 10
    #6個零
    lr = 0.00000027
    epochs = 10000000

    linear_reg = LinearRegression()
    linear_reg.fit(new_x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
 ✓  5m 47.7s

Don't cheat.
```

After several trials, I reach the performance below:

```
    print('training loss: ', linear_reg.evaluate(new_x_train, y_train))
    print('validation loss: ', linear_reg.evaluate(new_x_val, y_val))
    linear_reg.plot_curve()
 ✓  2.6s

training loss:  22147095.947670132
validation loss:  28563304.530444108
```

Hope it works on the ground truth testing set :)

What data analysis have you done? Why choose the above setting? Other strategies? (please explain in detail; otherwise, no points will be given.)

Part. 2, Questions (30%):

(7%) 1. What's the difference between Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent?

Ans:

These three methods are all optimization algorithms commonly used in machine learning to train models.The followings are detailed descriptions:

Gradient descent works by iteratively adjusting the model parameters in the opposite direction of the gradient of the cost function. It moves the model parameters in the direction that minimizes the cost function. It computes the gradient of the cost function with respect to the model parameters using the entire training dataset, which can be computationally expensive for large datasets.

To solve the above shortcomings, Mini-Batch gradient descent works by computing the gradient of the cost function with respect to the model parameters on small subsets of the training dataset, that's why it's called mini-batches.This reduces the computational cost of computing the gradient on the entire dataset and leads to faster convergence than using the full dataset.

Stochastic Gradient Descent is another variation of gradient descent. It computes the gradient of the cost function with respect to the model parameters on a single randomly selected data point from the training dataset. The randomness helps the model avoid local minima and can lead to faster convergence , but can also make the training process noisy and require more iterations to converge.

To sum up , when computing gradient , Gradient Descent use the whole data set, Mini-Batch Gradient Descent use the small subset of dataset, and Stochastic Gradient Descent computes the gradient on a single randomly selected data point from the dataset. Each of these methods has its own advantages and disadvantages, so we must carefully consider the timing of their use.

(7%) 2. Will different values of learning rate affect the convergence of optimization? Please explain in detail.

Ans:

The learning rate is a crucial hyperparameter that can significantly affect the convergence of the optimization algorithm. If the learning rate is too large, the algorithm may take steps that are too large and miss the optimal solution. Therefore the algorithm may diverge. And if the learning rate is too small, the algorithm may converge too slowly and has the risk of getting stuck in local minima and failing to find the global minimum of the cost function. That's why finding the optimal learning rate is crucial for efficient and effective model training.

(8%) 3. Suppose you are given a dataset with two variables, X and Y, and you want to perform linear regression to determine the relationship between these variables. You plot the data and notice that there is a strong nonlinear relationship between X and Y. Can you still use linear regression to analyze this data? Why or why not? Please explain in detail.

Ans:

Linear regression assumes that there is a linear relationship between the independent variable(s) (X) and the dependent variable (Y). If the relationship is nonlinear, the linear model will not be able to capture the underlying pattern in the data, resulting in a poor fit and inaccurate predictions.

There are some approaches to do regression on non-linear data. The first one is to transform the data so that the relationship becomes linear. For instance, if the relationship between X and Y is quadratic, we could create a new variable Z = X^2 and use linear regression with Z as the independent variable. If the result is still not satisfying, you should try nonlinear regression.

In summary, linear regression may not be the best choice for analyzing data when there is a strong nonlinear relationship between the variables. It may be necessary to transform the data or use a nonlinear regression model to figure out the underlying pattern in the data.

(8%) 4. In the coding part of this homework, we can notice that when we use more features in the data, we can usually achieve a lower training loss. Consider two sets of features, A and B, where B is a subset of A. (1) Prove that we can achieve a non-greater training loss when we use the features of set A rather than the features of set B. (2) In what situation will the two training losses be equal?

Ans:

(1) Assume the lowest loss we achieve in set A is L(A). And that uses only the features in set B and achieves a training loss of L(B). Because is a subset of A, we can write A as a union of B and some other features that are not in A, i.e., $A = B \cup C$, where C is a set of additional features that not in set B but belong to set A.

Now, we can use the model that was trained on A to predict the target variable using only the features in B. Since the model was trained on more features than are present in B, it will have access to more information, which can only improve its predictions. Therefore, the model that was trained on A will have a training loss that is lower or equal to the model trained on B, i.e., $L(A) \leq L(B)$.

(2) The two training losses will be equal when the additional features in set A do not contribute to the prediction of the target variable. That is, the additional features do not contain any additional information that helps the model to better fit the training data. In this case, the model trained on set B is capable of performing as well as the model trained on set A, which contains additional non-informative features.