NYCU Pattern Recognition, Homework 1

311551059, 陳昱丞

Part. 1, Coding (70%):

1. (0%) Show the learning rate and epoch you choose
   learning rate: 2*1e-3
   epoch: 250000

```
batch_size = x_train.shape[0]

# TODO
# Tune the parameters
# Refer to slide page 9
lr = 2*1e-3
epochs = 250000

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: 380.1346459
   intercepts: 1382.5371478013026

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

```
Intercepts:  1382.5371478013026
Weights:  [array([380.1346459])]
```

3. (5%) What's your final training loss (MSE)?
   training loss (MSE): 139562065.48342776

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

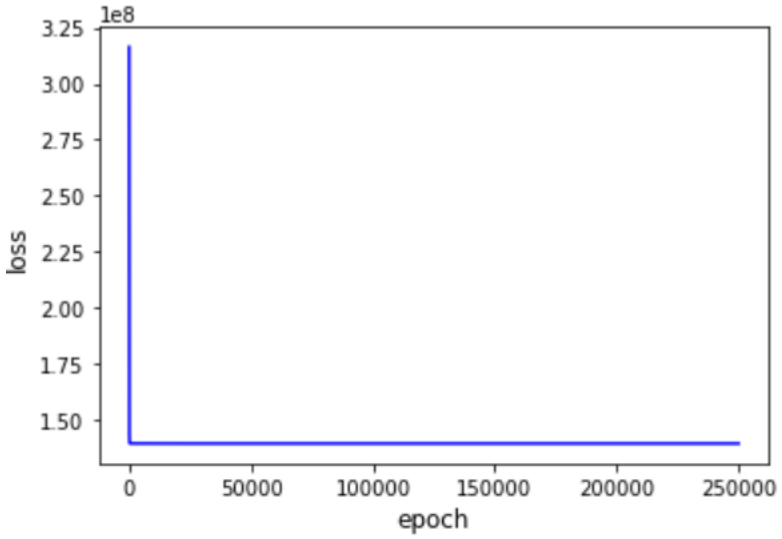
```
training loss:  139562065.48342776
```

4. (5%) What's the MSE of your validation prediction and validation ground truth?
   validation loss (MSE): 136920284.38379115

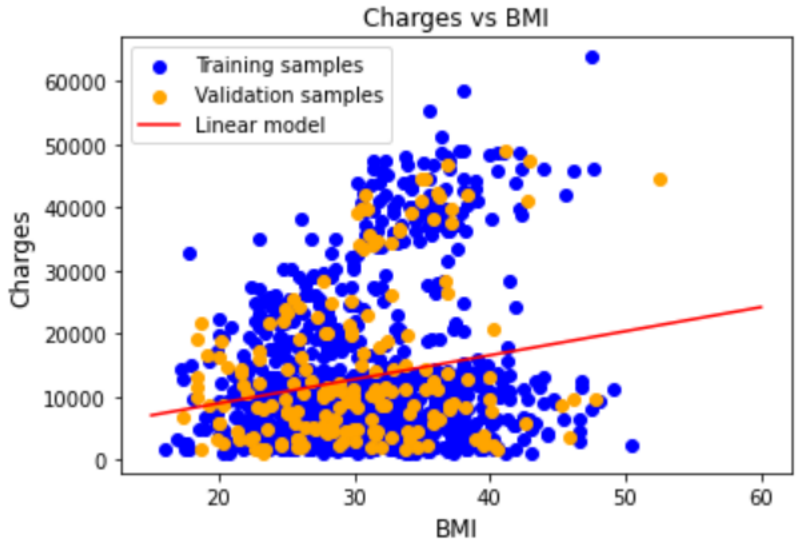
```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```

```
validation loss:  136920284.38379115
```

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: 7*1e-4
epoch: 1000000

```
batch_size = x_train.shape[0]

# TODO
# Tune the parameters
# Refer to slide page 10
lr = 7*1e-4
epochs = 1000000

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

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

weights: 259.85086418, -383.54524442, 333.33251488, 442.55747832, 24032.22099508, -416.01438631
intercepts: -11857.057350985515

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

```
Intercepts:  -11857.057350985515
Weights:  [array([  259.85086418,  -383.54524442,   333.33251488,   442.55747832,
        24032.22099508,  -416.01438631])]
```

9. (5%) What's your final training loss?
   training loss (MSE): 34697170.25351918

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

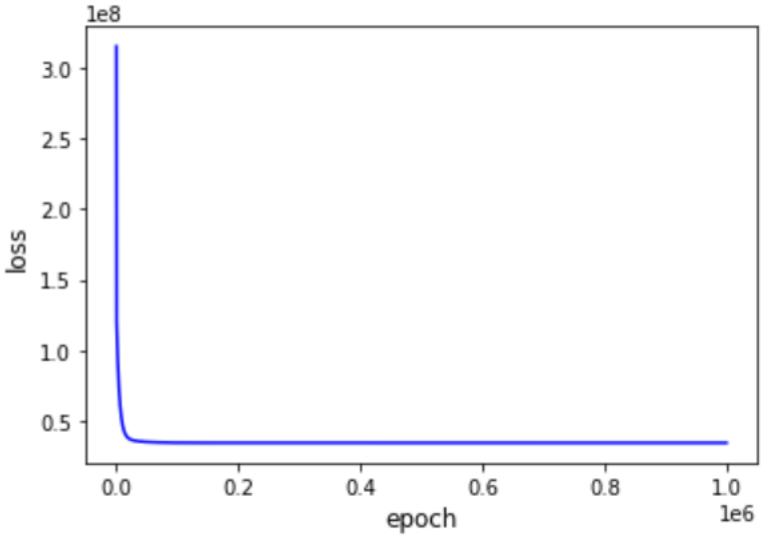
```
training loss:  34697170.25351918
```

10. (5%) What's the MSE of your validation prediction and validation ground truth?
    validation loss (MSE): 41958565.75652556

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

```
validation loss:  41958565.75652556
```

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: 1e-7
    epoch: 1000000
    batch_size: 938
    Used features: 16

```
batch_size = x_train.shape[0]
linear_reg = MyLinearRegression(16)
lr = 1e-7
epochs = 1000000
linear_reg.fit(x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
```

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

```
training loss:  23301836.36248132
validation loss:  28305788.92847915
```

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

Firstly, I would like to know if all features are helpful for the model's prediction. Therefore, I dropped each feature separately and compared the validation loss with the model using all six features. The following table shows the results:

	Using all	Drop age	Drop sex	Drop bmi	Drop children	Drop smoker	Drop region
Training loss	34697170	48180706	34733626	38584969	34975610	125302326	34905983
Validation loss	41958565	52010169	41810115	47132990	41772259	132347522	42453172

Results showed that dropping the "sex" and "children" features reduced the validation loss. So, I tried dropping both features at the same time, and the loss was as follows:
Training loss: 35006988
Validation loss: 41663153
The loss was even smaller than before. So, I decided to only use the following four features: "age", "bmi", "smoker", and "region".

Next, I tried to add some nonlinear features to the model to increase its complexity. I referred to PolynomialFeatures with interaction_only=True in scikit-learn. Its behavior is that given four existing features written as X1, X2, X3, and X4, the feature set is expanded to 1, X1, X2, X3, X4, X1X2, X1X3, X1X4, X2X3, X2X4, X3X4, X1X2X3, X1X2X4, X1X3X4, X2X3X4, X1X2X3X4, which are a total of 16 features. Since scikit-learn cannot be used for this assignment, I have to implement this using numpy:

```
# features: age, sex, bmi, children, smoker, region
x_train = df_train.drop(['charges', 'sex', 'children'], axis=1)
y_train = df_train['charges']
x_val = df_val.drop(['charges', 'sex', 'children'], axis=1)
y_val = df_val['charges']
x_test = df_test.drop(['charges', 'sex', 'children'], axis=1)

x_train = np.array(x_train)
y_train = np.array(y_train)
x_val = np.array(x_val)
y_val = np.array(y_val)
x_test = np.array(x_test)

def poly_feature(a):
    Len = a.shape[0]
    buffer = []
    li0 = [1] * Len
    li1 = a[:,0].tolist()
    li2 = a[:,1].tolist()
    li3 = a[:,2].tolist()
    li4 = a[:,3].tolist()
    li5 = (a[:,0] * a[:,1]).tolist()
    li6 = (a[:,0] * a[:,2]).tolist()
    li7 = (a[:,0] * a[:,3]).tolist()
    li8 = (a[:,1] * a[:,2]).tolist()
    li9 = (a[:,1] * a[:,3]).tolist()
    li10 = (a[:,2] * a[:,3]).tolist()
    li11 = (a[:,0] * a[:,1] * a[:,2]).tolist()
    li12 = (a[:,0] * a[:,1] * a[:,3]).tolist()
    li13 = (a[:,0] * a[:,2] * a[:,3]).tolist()
    li14 = (a[:,1] * a[:,2] * a[:,3]).tolist()
    li15 = (a[:,0] * a[:,1] * a[:,2] * a[:,3]).tolist()

    buffer.append(li0)
    buffer.append(li1)
    buffer.append(li2)
    buffer.append(li3)
    buffer.append(li4)
    buffer.append(li5)
    buffer.append(li6)
    buffer.append(li7)
    buffer.append(li8)
    buffer.append(li9)
    buffer.append(li10)
    buffer.append(li11)
    buffer.append(li12)
    buffer.append(li13)
    buffer.append(li14)
    buffer.append(li15)

    buffer = np.array(buffer, dtype=float)
    return buffer.T


x_train = poly_feature(x_train)
x_val = poly_feature(x_val)
x_test = poly_feature(x_test)
```

After the feature expansion described above, the validation loss is 28305788, which meets the homework requirement of loss < 30000000.

Part. 2, Questions (30%):

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

The difference between Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent is the amount of data used to update the model parameters at each iteration. Gradient Descent uses the entire dataset, Mini-Batch Gradient Descent uses small batches, and Stochastic Gradient Descent uses a single sample at each iteration.

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

Yes, a large value of learning rate will lead to fast convergence. But its more likely to overshoot the optimal values and fail to converge. While a small value of learning rate converges slowly, but its more likely to converge to the optimal value successfully.

(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.

If the relationship between X and Y is nonlinear, using linear regression to analyze the data may result in a poor model fit. However, you can use transformation of the variables to linearize the relationship, then you can still perform linear regression on it.

(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?

Write weight vector of A as w_A, B as w_B, C as w_C. Since B is a subset of A, we can write w_A as [w_B; w_C], where w_C is the weight vector for the additional features in set A that are not in set B.

(1) If we fit with set A, then ya_pred = X*w_A = X*[w_B; w_C]. If we fit with set B, then yb_pred = X*[w_B; 0]. Then after training, Loss(y, ya_pred) <= Loss(y, yb_pred) since we will always have zeros in the weight vector.

(2) The two training losses will be equal if the additional features (set C) in set A do not contribute to the prediction of the target variable. Because when it happens, we will get w_C~=0 after training.