NYCU Pattern Recognition, Homework 1

0810749 , 張君實

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: 1.5e-5
   epoch: 10000000

```
batch_size = x_train.shape[0]

# TODO
# Tune the parameters
# Refer to slide page 9
lr = 1.5e-5
epochs = 10000000
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: 380.13540619
   intercepts: 1382.5127648555201

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

```
Intercepts:  1382.5127648555201
Weights:  [380.13540619]
```

3. (5%) What's your final training loss (MSE)?
   training loss (MSE): 139562065.48344946

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

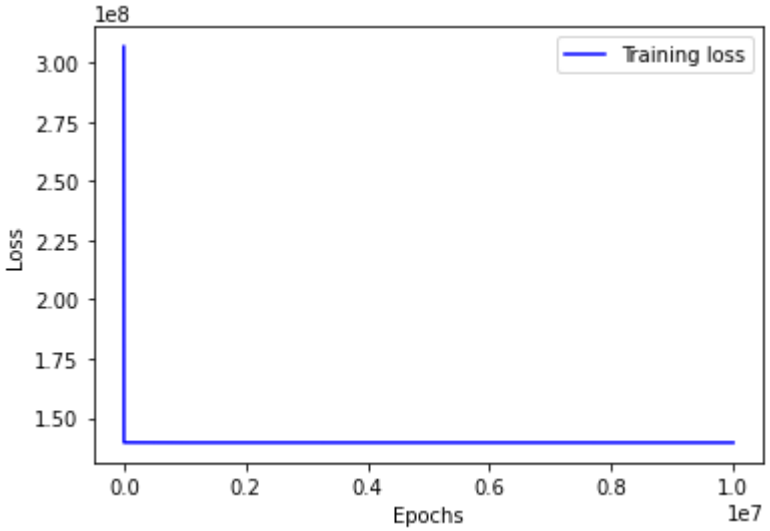
```
training loss:  139562065.48344946
```

4. (5%) What's the MSE of your validation prediction and validation ground truth?
validation loss (MSE): 136920281.01301628

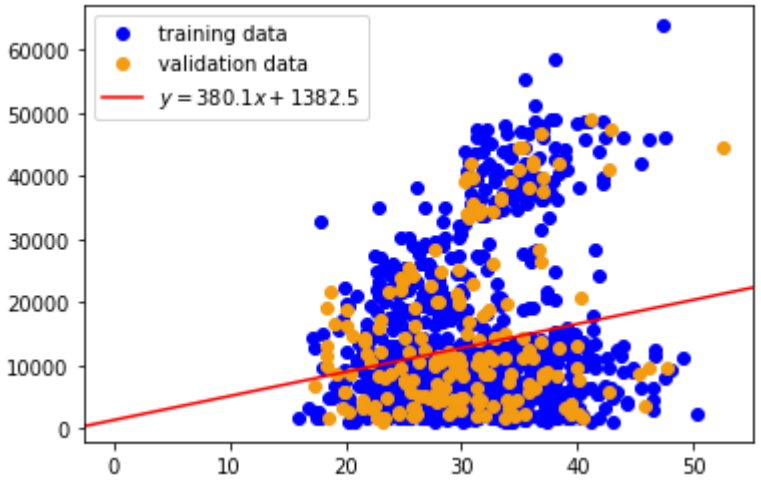
```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```

```
validation loss:  136920281.01301628
```

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: 2e-6

epoch: 100000000

```
batch_size = x_train.shape[0]

# TODO
# Tune the parameters
# Refer to slide page 10
lr = 2e-6
epochs = 100000000

linear_reg = LinearRegression()
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: [259.85021055 -383.55425656 333.3293558 442.5551735
   24032.2152028 -416.01707014]
   intercepts: -11856.917545971966

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

```
Intercepts:  -11856.917545971966
Weights:  [  259.85021055  -383.55425656   333.3293558    442.5551735
 24032.2152028   -416.01707014]
```

9. (5%) What's your final training loss?
   training loss (MSE): 34697170.25408136

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

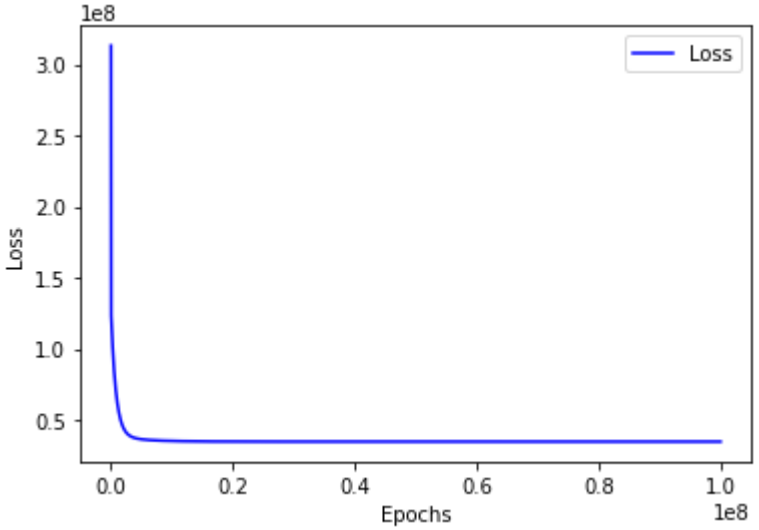
```
training loss:  34697170.25408136
```

10. (5%) What's the MSE of your validation prediction and validation ground truth?
    validation loss (MSE): 41958555.63310554

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

```
validation loss:  41958555.63310554
```

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: 2e-8
epoch: 3000000
batch_size: x_train.shape[0] (The number of the data in the regression_train.csv is 940.)
Used features: I used a total 29 features in my model, and I will explain how I choose the features in the following part.

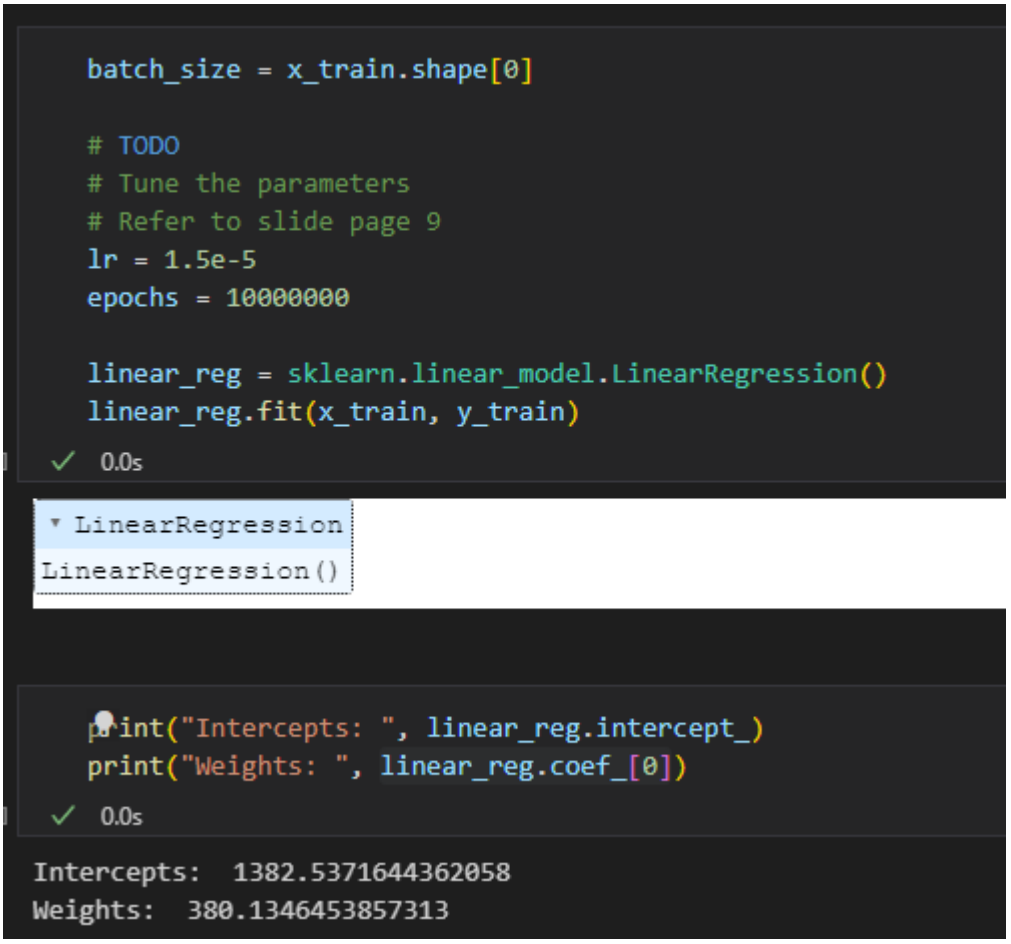
```
#training
batch_size = x_train.shape[0]
lr = 2e-8
epochs = 3000000

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

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

(1) The supportive tool for data analysis:
In the part of the homework, I need to minimize the loss of the testing dataset as low as possible. In order to minimize the loss of the testing dataset, I used to change the strategy to minimize the loss of the training dataset because there is a relation between training/validation/testing loss, to some extent.

Now I try to minimize the loss of the training dataset, and there are many strategies I can choose to modify the data pre-processing. We know that we can find a closed form solution to the linear regression model, and we can solve the linear regression and know the lower bound quickly by calculating the closed form solution. Thus, I use sklearn.linear_model.LinearRegression, which is a

library that solves the analytic solution, for saving time to understand the lower bound of the training loss.



In scikit library, there is also a handy tool that lets us know the importance of features in the regression problem quickly. We can import sklearn.feature_selection to realize the importance of every feature we use in our strategies of data pre-processing. The principle of the sklearn.feature_selection is to calculate the correlation coefficient and transform them to the f-scores. The f-scores are bigger, the features are more important.

```python
#test_pred = linear_reg.predict(x_test)
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
fs = SelectKBest(score_func=f_regression, k='all')
fs.fit(x_train_np, y_train_np)
print(len(fs.scores_))
rank = np.flip(np.argsort(fs.scores_))
for i in range(len(fs.scores_)-1):
 print('Feature %d: %f' % (i, fs.scores_[i]))
 print(x_train.columns[i])
for i in range(len(fs.scores_)-1):
    print("rank ",i,": ",rank[i])
    print(x_train.columns[rank[i]])
#print('validation loss: ', linear_reg.evaluate(x_val, y_val))
fs.fit(x_val_np, y_val_np)
print(len(fs.scores_))
rank = np.flip(np.argsort(fs.scores_))
for i in range(len(fs.scores_)-1):
 print('Feature %d: %f' % (i, fs.scores_[i]))
for i in range(len(fs.scores_)-1):
    print("rank ",i,": ",rank[i])
    print(x_train.columns[rank[i]])
```

```
Output exceeds the size limit. Open the full output data in a text editor
30
Feature 0: 109.943336
age
Feature 1: 0.628671
sex
Feature 2: 35.044649
bmi
Feature 3: 2.207758
children
Feature 4: 1550.015098
smoker
Feature 5: 0.958675
northeast
Feature 6: 1.932249
northwest
Feature 7: 3.393076
southeast
Feature 8: 2.344052
southwest
```

Of course, I still implement the homework with only numpy and python in the homework, the scikit library is only used to know the effectiveness of the strategies of data pre-processing.

(2) Label encoding:

In this lab, I modify the label encoding by TA's suggestions. At first, I thought that the region is suitable for one-hot encoding because the relationship of the space is not a linear relation, and I modified it. Additionally, in order to preserve the features for adding features, I changed the 0 to -1 in the label encoding, and it is helpful for us to add the non-linear features.

	age	sex	bmi	children	smoker	northeast	northwest	southeast	southwest
0	19	-1	27.900	0	1	1.0	1.0	1.0	-1.0
1	18	1	33.770	1	-1	1.0	1.0	-1.0	1.0
2	28	1	33.000	3	-1	1.0	1.0	-1.0	1.0
3	33	1	22.705	0	-1	1.0	-1.0	1.0	1.0
4	32	1	28.880	0	-1	1.0	-1.0	1.0	1.0

(3) Add features:

It is the most important thing to reduce the loss because we can only add the nonlinear basis function here in the homework.

As for the method of adding features, I referred to the Lotka-Volterra model, a kind of nonlinear regression method, and multiplied 2 to 3 of the features except charges as the new features. According to the table in the (2)Label encoding, we at first had 9 features. After multiplying 2 to 3 of them, we afterwards had 9+9^2+9^3 = 819 features. Of course, I had tried to multiply above 4 of the features. Unfortunately, it would make the model overfitting.

So I got 819 features. However, the features are more, the convergence time is longer. Thus, we need to pick up the useful features to reduce the computation. To be frank, we actually can not intuitively see which features are useful. Thus, I used sklearn.feature_selection, which is mentioned in the (1)The supportive tool for data analysis, and selected 20 of the most important features and 9 features in the table of Label encoding. Thus, I ultimately used 29 features in my model.

```
    #Multiply 2~3 feature(except charge) to get some new feature.
        #Multiply 2 numbers
two_mul_index = np.array([[2,4],[0,4],[3,4],[4,6],[4,5],[0,2]])
for i,j in two_mul_index:
    df_train[str(i)+","+str(j)] = np.nan
    df_train[str(i)+","+str(j)] = df_train.iloc[:,i] * df_train.iloc[:, j]
    df_val[str(i)+","+str(j)] = np.nan
    df_val[str(i)+","+str(j)] = df_val.iloc[:,i] * df_val.iloc[:, j]
    df_test[str(i)+","+str(j)] = np.nan
    df_test[str(i)+","+str(j)] = df_test.iloc[:,i] * df_test.iloc[:, j]

        #Multiply 3 numbers
three_mul_index = np.array([[4,8,8],[4,5,5],[4,7,7],[4,6,6],[1,1,4],[4,4,4],[2,2,4],[0,2,4],[0,0,4],[2,3,4],[2,4,6],[2,4,5],[0,3,4],[2,4,8]])
for i,j,k in three_mul_index:
    df_train[str(i)+","+str(j)+","+str(k)] = np.nan
    df_train[str(i)+","+str(j)+","+str(k)] = df_train.iloc[:,i] * df_train.iloc[:, j] * df_train.iloc[:, k]
    df_val[str(i)+","+str(j)+","+str(k)] = np.nan
    df_val[str(i)+","+str(j)+","+str(k)] = df_val.iloc[:,i] * df_val.iloc[:, j] * df_val.iloc[:, k]
    df_test[str(i)+","+str(j)+","+str(k)] = np.nan
    df_test[str(i)+","+str(j)+","+str(k)] = df_test.iloc[:,i] * df_test.iloc[:, j] * df_test.iloc[:, k]
```

After adding features, now we can easily check the training/validation loss of closed form by sklearn.linear_model.LinearRegression(), and the result indicates the selection of the features is good enough.

```
    linear_reg = sklearn.linear_model.LinearRegression()
    linear_reg.fit(x_train_np, y_train_np)
```

```
▾ LinearRegression
LinearRegression()
```

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

```
Intercepts:  -13232.302565362781
Weights:  [ 2.43460331e+02 -2.71086704e+02  7.36601759e+02  4.67484213e+02
 -1.64635109e+03 -5.68045336e+02 -4.69095641e+01  3.61604845e+02
  2.53350055e+02  6.79188155e+02  1.53525273e+02  4.67439225e+02
 -1.22180430e+03 -1.67640054e+03  6.65320510e-01 -1.64635109e+03
 -1.64635109e+03 -1.64635109e+03 -1.64635109e+03 -1.64635109e+03
 -1.64635109e+03 -9.43752810e-02  1.93904783e+00 -2.78746170e+00
 -1.75336878e+01  2.61553009e+01  3.79052521e+01 -1.89547827e+00
 -1.35136028e+01]
```

```
    inferred_body_mass = linear_reg.predict(x_train_np)
    model_error = mean_squared_error(y_train_np, inferred_body_mass)
    print(f"The mean squared error of the optimal model is {model_error:.2f}")
    #print('training loss: ', linear_reg.evaluate(x_train, y_train))
```

```
The mean squared error of the optimal model is 20955205.27
```

```
    #print('validation loss: ', linear_reg.evaluate(x_val, y_val))
    inferred_body_mass = linear_reg.predict(x_val_np)
    model_error = mean_squared_error(y_val_np, inferred_body_mass)
    print(f"The mean squared error of the optimal model is {model_error:.2f}")
```

```
The mean squared error of the optimal model is 28172730.75
```

(4) Tuning the lr/epochs/batch:

According to my experience in the class DLP, in order not to overfit the model and maintain the robustness of the model, we don't need to train the model until closed form. Thus, I tuned the parameters according to my experience and made my training loss close to the training loss of the optimal model.

```python
#training
batch_size = x_train.shape[0]
lr = 2e-8
epochs = 3000000

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

#evaluation
print("Intercepts: ", linear_reg.weights[-1])
print("Weights: ", linear_reg.weights[:-1])
print('training loss: ', linear_reg.evaluate(x_train, y_train))
print('validation loss: ', linear_reg.evaluate(x_val, y_val))

#prediction
test_pred = linear_reg.predict(x_test)
print("test_pred shape: ", test_pred.shape)
assert test_pred.shape == (200, 1)
```

```
Intercepts:  5.623406621178636
Weights:  [-3.60739144e+01 -2.62080394e+01  3.22197797e+02  4.73190858e+01
  3.67542050e+00 -3.15045678e+01  3.67458837e+00  2.34250678e+01
  1.56517249e+01  7.37520299e+01  4.17712007e+01  1.54129190e+00
 -8.55432431e-01 -1.62487405e+00  9.65392040e+00  3.67542050e+00
  3.67542050e+00  3.67542050e+00  3.67542050e+00  3.67542050e+00
  3.67542050e+00  6.23069544e+00  7.16160455e+00 -3.41521247e+00
 -1.48083957e+01 -6.91039774e+00  2.04183190e-01  9.83472535e-02
 -1.40927927e+01]
training loss:  21885413.112560567
validation loss:  27462889.769601174
test_pred shape:  (200, 1)
```

Part. 2, Questions (30%):

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

The batch size of Gradient Descent is the number of the training data.

The batch size of Stochastic Gradient Descent is 1.

The batch size of Stochastic Mini-Batch Gradient Descent is not 1 or the number of the training data. According to the experience, we usually set the batch size as 16, 32 ,64.

The advantage of Gradient Descent is that the updated path of the parameter is smooth, but the disadvantage is that the updatinging time is long because we need to calculate all the data before updating the parameter.

The advantage of Stochastic Gradient Descent is that the updatinging time is short because we need to calculate 1 data before updating the parameter, but the disadvantage is that the updated path of the parameter is oscillating.

The effect of Mini-Batch Gradient Descent is between the effect of Gradient Descent and Stochastic Gradient Descent.

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

Of course yes.

If we use a very big number as a learning rate, it would move too far in every epoch, and it cannot converge.

Additionally, about the method of gradient descent, it only converges to the local minima. In other words, it may not converge to the global minima.

On the contrary, if we use a very small number as a learning rate, it would move so small that it would converge to the local minima, but it cannot escape from local minima to the global minima.

Thus, we only choose a moderate number to the learning rate by our experience. Additionally, if we can use optimizer, we can use adam to increase the probability to escape from local minima.

(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 I can guess, find out the nonlinear relationship between X and Y, and add new features to linear regression, I can still use linear regression to analyze this data.

Otherwise, I cannot use linear regression to analyze this data.

All I can do is, I can pre-process the data, transforming X and Y to the nonlinear basis and adding features. Maybe I can guess and find out the relationship between X and Y and analyze the data.

However, in the real world, we can find many examples that we cannot intuitively find the relationship between X and Y. For example, let X as a face picture and Y as facial attractiveness. In this case, I highly recommend using Resnet50 instead of linear regression. Guessing the relationship and adding the new features for linear regression is not easy.

(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?

(1)

We assume that A have r features, and B have S features, where $s \le r$

The traing loss of A is

$$Loss_A = \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_{A0} - w_{A1}x_{1i} \cdots - w_{Ar}x_{ri}\right)^2 = \frac{1}{n}\cdot(Y - W_A^T X)^T (Y - W_A^T X)$$

And the training loss of B is

$$Loss_B = \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_{B0} - w_{B1}x_{1i} \cdots - w_{Bs}x_{si}\right)^2 = \frac{1}{n}\cdot(Y - W_B^T X)^T (Y - W_B^T X)$$

$$= \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_{B0} - w_{B1}x_{1i} \cdots - w_{Bs}x_{si} - 0x_{s+1} - 0x_{s+2} \cdots - 0x_{ri}\right)^2$$

$$= \frac{1}{n}(Y - W_{B'}^T X)^T (Y - W_{B'}^T X)$$

, where $W_A, W_{B'}$ are length r+1 vector, and W_B is length s+1 vector.

And that the minimum loss A and B as

$$Loss^*A = \min Loss A = \frac{1}{n}(Y - W_A^{*T} X)^T (Y - W_A^{*T} X)$$

$$Loss^*B = \min Loss B = \frac{1}{n}(Y - W_B^{*T} X)^T (Y - W_B^{*T} X)$$

We assume that $Loss^*B < Loss^*A$. But according to the definition of $Loss\ B = \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_0 - w_1 x_{1i} \cdots - w_s x_{si}\right)^2$

$$= \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_0 - w_1 x_{1i} \cdots - w_s x_{si} - 0x_{s+1} - 0x_{s+2} \cdots - 0x_{ri}\right)^2$$

It means that we find a vector $W_B^* = [w_{B0}, w_{B1} \cdots, w_{Bs}, 0, \cdots, 0]^T$ with length r+1 such that $Loss^*B = \frac{1}{n}(Y - W_{B'}^{*T} X)^T (Y - W_{B'}^{*T} X) < Loss^*A$

However, $Loss^*A = \min LossA$, we cannot find a length r+1 vector W such that $\frac{1}{n}(Y-W^TX)^T(Y-W^TX) < Loss^*A$, so it is contradict.

Thus, we prove $Loss^*A \leq Loss^*B$, in other words, we can achieve a non greater training loss when we use the features A instead of B.

(2)

The training loss of B is

$$Loss_B = \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_{B0} - w_{B1}x_{1i} \cdots - w_{Bs}x_{si}\right)^2 = \frac{1}{n}(Y-W_B^TX)^T(Y-W_B^TX)$$

The traing loss of A is

$$Loss_A = \frac{1}{n}\sum_{i=1}^{k}\left(y_i - w_{A0} - w_{A1}x_{1i} \cdots - w_{Ar}x_{ri}\right)^2 = \frac{1}{n}(Y-W_A^TX)^T(Y-W_A^TX)$$

We assume that $\min Loss_B = \frac{1}{n}(Y-W_B^{*T}X)^T(Y-W_B^{*T}X) = Loss^*B$

$$\min LossA = \frac{1}{n}(Y-W_A^{*T}X)^T(Y-W_A^{*T}X) = Loss^*A$$

Case 1: If $\det(X^TX) \neq 0$

It is trivial if $W_{Ai}^* = W_{Bi}^*$ for $i \in [0, s]$

and if $W_{Ai}^* = 0$ for $i \in [s+1, r]$

, then $Loss^*_A$

$$= \frac{1}{n}\sum_{i=1}^{k} \left(y_i - w_{A0}^* - w_{A1}^* x_{1i} - \cdots - w_{Ar}^* x_{ri}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{k} \left(y_i - w_{A0}^* - w_{A1}^* x_{1i} - \cdots w_{As}^* x_{si}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{k} \left(y_i - w_{B0}^* - w_{B1}^* x_{1i} - \cdots - w_{Bs}^* x_{si} - 0x_{s+1\,i} \cdots - 0x_{ri}\right)^2$$

$$= Loss^*_B$$

And the closed form solution of linear regression

$$W_{ML} = (X^T X)^{-1} X^T Y$$

implies that if $(X^T X)^{-1}$ is solvable, then

W_{ML} is the only solution $= \underset{W}{\operatorname{argmin}}$ Loss

So I prove that $W_A^* = [W_B^*, 0, 0, 0 \cdots 0]$

is the only solution when $Loss^*_A = Loss^*_B$ and $\det(X^T X) \neq 0$.

Case (2): If $\det(X^T X) = 0$, where $X = \begin{bmatrix} x_{01} & \cdots & x_{M-1\,1} \\ x_{02} & & \vdots \\ \vdots & & \vdots \\ x_{0N} & \cdots & x_{M-1\,N} \end{bmatrix}$

$\Rightarrow$ There exists $k \in [0, M-1]$ such that $X_k^T X_i = 0$, for all $i \in [0, M-1]$

$\Rightarrow X_k^T X_k = 0 \Rightarrow \|X_k\| = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

It implies that there exists a feature filled with zeros, and the corresponding W_k could be any number. Thus, we remove the X_k column, and check the $\det(X^TX)$ again, if $\det(X^TX) \neq 0$, go to the case(1), else, go to the case(2).

Ultimately, we get the conclusion.

If A contains feature $1 \sim$ feature r,

and B contains feature $1 \sim$ feature s.

Unless feature $k \in [1, r]$ are all zero, where W_{Ak}^* and W_{Bk}^* could be an arbitrary number, $W_{Ak}^* = W_{Bk}^*$ for all $k \in [1, s]$, and

$W_{Ak}^* = 0$ for all $k \in [s+1, r]$