CHAPTER 3: LINEAR REGRESSION

Theory

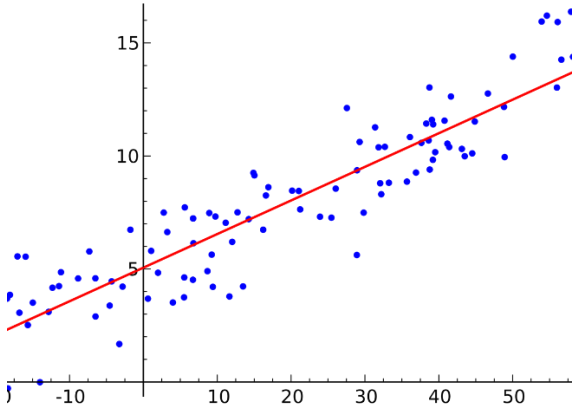
In this chapter, the basic concept and theory of linear regression which is used in machine learning is described first and consider programming method using Tensorflow and Python for several regression problems in machine learning.

What is linear regression?

Linear regression analysis is a representative method for linearly analyzing the influence of other variables on a variable.

In order to perform linear regression analysis, a linear regression model should be created first. The model referred to here is a function expressed by a mathematical expression, which consists of influencing variables and influenced variables.

The affecting variable is called an **independent variable** or an **explanatory variable**, and the affected variable is called a **dependent variable** or a **response variable**.



The dependent variable and the independent variable constituting the linear regression model may be two or more, respectively. A linear regression

model with one dependent variable is called a **univariate linear regression model** and a **multivariable linear regression model** with two or more dependent variables.

On the other hand, the linear regression model with one independent variable is called a **simple linear regression model**, and when there are two or more independent variables, it is called a **multiple linear regression model**.

Therefore, the linear regression model is called the **univariate multiple linear regression model**. When the dependent variable and the independent variable are two or more, the **multivariable multiple linear regression model** is used.

In this chapter let's consider simplest univariate simple linear regression model.

Basic of Machine Learning

Let's think about supervised learning with a student score data. These scores correspond to the hours studied.

x (hours)	y (score)
10	90
9	80
3	50
2	30

The goal should predict is the score between 0 and 100. We can know a student studied 10 hours got 90 marks and studied 9 hours got 80 marks.

We can learn these data. We call this **training**, and above data calls **training data**.

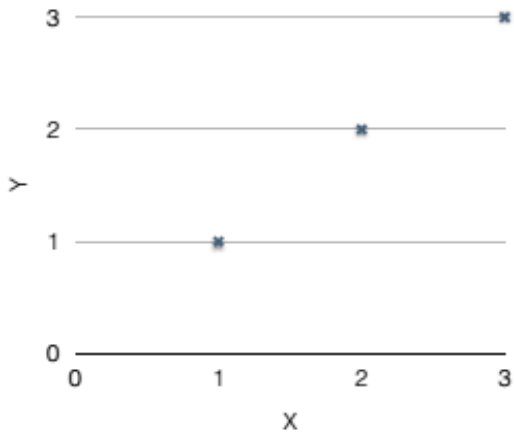
After training, we can ask trained model before the exam, "I studied 7 hours, then how much scores can I get?" If the models are well trained, maybe the answer could be 60~70.

Let’s consider how the model runs internally and how we can train this model from now.

Let’s think the simplest training data for explaining.

x	y
1	1
2	2
3	3

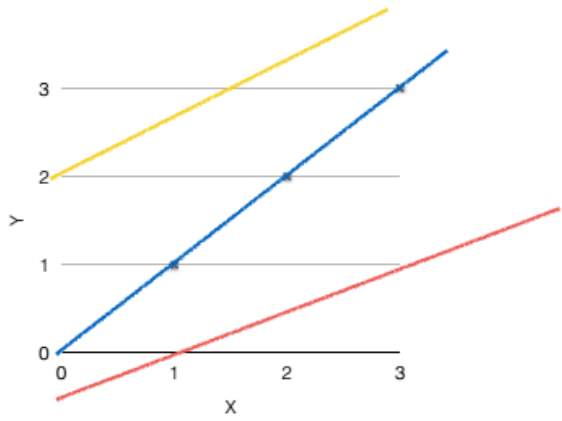
If we draw the training data then we can get the following graph.



From this graph, we can guess that there will be some linear relationship between x and y. The reason why linear regression is widely used in machine learning is that many data in the world satisfy the linear relationship.

Then we can make some hypothesis which can show the relationship between x and y.

It is just learning to find the hypothesis closest to the learning data among these hypotheses.



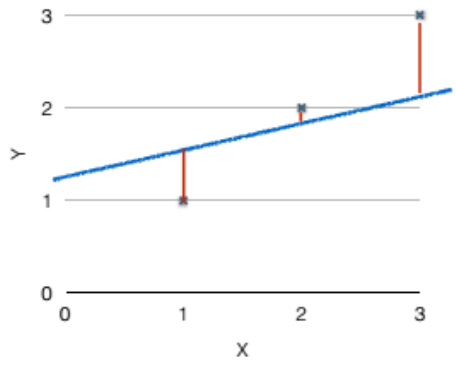
Let's represent mathematical formulas. We can represent the hypothesis as below linear equation.

$$H(x) = Wx + b$$

The shape of the line will depend on the value of W and b.

Then which line is the closest to training data?

We can simply calculate the errors using the distance between hypotheses and training data.



In linear regression, we call it **cost function** or **loss function**.

In above case, the cost function could be such as:

$$\cos t = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right)-y^{(i)}\right)^2$$

In here, m is counts of training data. H is a function of W and b, so $\cos t$ is also become the function of W and b.

$$\cos t(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right)-y^{(i)}\right)^2$$

Our goal is to find the value of W and b which minimize the cost function.

$$\underset{W,b}{\text{minimize}}\, cost(W,b)$$

AIM

The aim of the following lab exercise is to implement the simple linear regression model with Tensorflow.

Following steps are required.

Task 1: Build graph using TF operations

Task 2: Run and update graph and get results

Task 3: Using Placeholder

LAB EXERCISE 2: IMPLEMENTATION OF LINEAR REGRESSION

1. **Build graph using TF operations**
2. **Run and Update graph and get results**
3. **Using Placeholder**

Task 1: Build graph using TF operations

We can implement the linear regression learning easily using Tensorflow.

STEP 1: Import tensorflow

First, import the tensorflow package in order to use tensorflow in Python script.

```
import tensorflow as tf
```

STEP 2: Define or load training data

Let's define the training data as follows.

```
# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]
```

STEP 3: Define model weights

And define W and b as tensorflow variable.

```
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
```

We should define W and b as tf.variable since W and b could be updated during training. W and b are given as random value initially.

As we can know from training data, W=1, b=0 is the optimal solution of this linear regression problem.

The training is working well means the value of W converge to the 1 and the value of b converge to the 0.

STEP 4: Define hypothesis

Next, let's define hypothesis.

```
# Our hypothesis XW+b
hypothesis = x_train * W + b
```

We can know above hypothesis is the implementation of below equation we discussed before.

$$H(x) = Wx + b$$

STEP 5: Define cost function

And cost function could be defined as follows.

```
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

Of course this is the implementation of the below-cost function.

$$\cos t(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right) - y^{(i)}\right)^2$$

This is the only definition of cost function operation, not means calculation. The value will be calculated and updated during training.

STEP 6: Define learning algorithm

The next part is the definition of the learning algorithm.

```
# Minimize
a = tf.Variable(0.1)
optimizer =tf.train.GradientDescentOptimizer(learning_rate=a)
train = optimizer.minimize(cost)
```

This means we will use Gradient Descent Optimization algorithm for learning. About this and another algorithm will be discussed later. We can use it as black box here.

Task1 is completed.

Task 2: Run and Update graph and get results

STEP 7: Initialize the tensorflow variables

Initialize the tensorflow variables such as W and b.

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())
```

If we ignore initialize command, tensorflow occurred error message about variable initialization. When we create the session, we should initialize the variables first.

STEP 8: Minimize the cost function

Then minimize the cost function.

```
# Fit the line
for step in range(2001):
  sess.run(train)
  if step % 20 == 0:
    print(step, sess.run(cost), sess.run(W), sess.run(b))
```

The minimization of the cost function is performed with only one line command sess.run(train).

Tensorflow optimize the train with GradientDescentOptimizer algorithm as we see above. And display the value of cost, W, b every 20 steps.

The coding is complete here.

Then the full code is below.

```
import tensorflow as tf

# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Our hypothesis XW+b
hypothesis = x_train * W + b

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))

# Minimize
a = 0.01
```

```
optimizer =tf.train.GradientDescentOptimizer(learning_rate=a)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
  sess.run(train)
  if step % 20 == 0:
    print(step, sess.run(cost), sess.run(W), sess.run(b))
```

Let's run this code.

Then we can see the results such as below.

```
'''
0 2.82329 [ 2.12867713] [-0.85235667]
20 0.190351 [ 1.53392804] [-1.05059612]
40 0.151357 [ 1.45725465] [-1.02391243]
...

1920 1.77484e-05 [ 1.00489295] [-0.01112291]
1940 1.61197e-05 [ 1.00466311] [-0.01060018]
1960 1.46397e-05 [ 1.004444] [-0.01010205]
1980 1.32962e-05 [ 1.00423515] [-0.00962736]
2000 1.20761e-05 [ 1.00403607] [-0.00917497]
'''
```

As we can know from results, the cost value is decreased from 2.82 to 1.2e-05, and the value of W and b is converged from 2.128, -0.852 to 1.0, 0.0.

Task2 is completed.

Task 3: Using Placeholder

We can use tf.placeholder for define training data.

Let's consider an example.

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
```

```
adder_node = a + b # + provides a shortcut for tf.add(a, b)

print(sess.run(adder_node, feed_dict={a: 3, b:4.5}))
print(sess.run(adder_node, feed_dict={a: [1, 3], b: [2, 4]}))
```

We defined a and b as placeholder float32 type. And define adder_node as an additional operation of a and b. Numerical value calculation isn't performed until now.

In last 2 lines, we set the value into a and b using feed_dict and calculate the adder_node using sess.run command.

The result is displayed as follows.

```
7.5
[3.  7. ]
```

We can use placeholder in our previous linear regression code.

Then why we should use placeholder?

Because we can reuse the linear regression model if we use placeholder.

STEP 8: Update code using placeholder

Let's update the code using placeholder and add prediction command.

```
import tensorflow as tf
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

X = tf.placeholder(tf.float32, shape=[None])
Y = tf.placeholder(tf.float32, shape=[None])

# Our hypothesis XW+b
hypothesis = X * W + b
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
a = 0.01
optimizer =tf.train.GradientDescentOptimizer(learning_rate=a)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
```

```
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
  cost_val, W_val, b_val, _ = sess.run([cost, W, b, train],
             feed_dict={X: [1, 2, 3], Y: [1, 2, 3]})
  if step % 20 == 0:
    print(step, cost_val, W_val, b_val)

# Testing our model
print(sess.run(hypothesis, feed_dict={X: [5]}))
print(sess.run(hypothesis, feed_dict={X: [2.5]}))
print(sess.run(hypothesis, feed_dict={X: [1.5, 3.5]}))
```

We can get results when running the code.

```
...
1980 1.32962e-05 [ 1.00423515] [-0.00962736]
2000 1.20761e-05 [ 1.00403607] [-0.00917497]

[ 5.0110054]
[ 2.50091505]
[ 1.49687922  3.50495124]
```

The last 3 lines are for prediction x value using trained linear regression model.

LAB CHALLENGE

Challenge

In this task what is more important is use the proper tensorflow functions depend on its version.

For example, the variable initialization function is

```
tf.global_variables_initializer()
```

But below the 0.10 version, the function is

```
tf.initialize_all_variables()
```

SUMMARY

Linear regression is the simple and widely used learning method and we can implement the linear regression model using tensorflow.

We can know the work flow of machine learning programming with tensorflow using this exercise.

REFERENCES

- https://en.wikipedia.org/wiki/TensorFlow
- https://en.wikipedia.org/wiki/Machine_learning
- https://en.wikipedia.org/wiki/Supervised_learning

INDEX