Linear regression

Puteaux, Fall/Winter 2020-2021

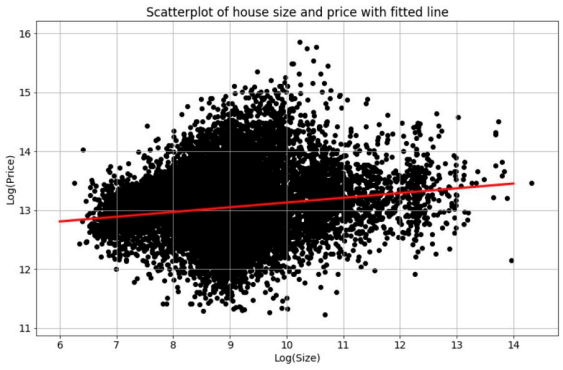
```
################################
##                            ##
##  Deep Learning in Python   ##
##                            ##
################################
```

`§2 Introduction to TensorFlow in Python`

`§2.2 Linear models`

1 Linear regression

1.1 What is linear regression?



1.2 How to make a linear regression model?

- A linear regression model assumes a linear relationship:
  - e.g., for the example of king county house sales dataset,
    - $price = intercept + size \times slope + error$
- Univariate regression models have only one feature,
  - e.g., for the example above, there could be only one feature, `size`
- Multiple regression models have more than one feature,
  - e.g., for the example above, there also could be two feature, `size` and `location`

1.3 Code of linear regression in TensorFlow:

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf

housing = pd.read_csv('ref1. King county house sales.csv')
```

[2]:
```
# Define the targets and features
price = np.array(housing['price'], np.float32)
size = np.array(housing['sqft_living'], np.float32)

# Define the intercept and slope
intercept = tf.Variable(0.1, np.float32)
slope = tf.Variable(0.1, np.float32)
```

[3]:
```
# Define a linear regression model
def linear_regression(intercept, slope, features=size):
    return intercept + features * slope
```

[4]:
```
# Compute the predicted values and loss
def loss_function(intercept, slope, targets=price, features=size):
    predictions = linear_regression(intercept, slope)
    return tf.keras.losses.mse(targets, predictions)
```

[5]:
```
# Define an optimization operation
opt = tf.keras.optimizers.Adam()
```

[6]:
```
# Minimize the loss function and print the loss
for j in range(50):
    opt.minimize(lambda: loss_function(intercept, slope),
                 var_list=[intercept, slope])
    print(loss_function(intercept, slope))
```

```
tf.Tensor(426196570000.0, shape=(), dtype=float32)
tf.Tensor(426193880000.0, shape=(), dtype=float32)
tf.Tensor(426191160000.0, shape=(), dtype=float32)
tf.Tensor(426188400000.0, shape=(), dtype=float32)
tf.Tensor(426185700000.0, shape=(), dtype=float32)
tf.Tensor(426182930000.0, shape=(), dtype=float32)
tf.Tensor(426180280000.0, shape=(), dtype=float32)
tf.Tensor(426177500000.0, shape=(), dtype=float32)
tf.Tensor(426174800000.0, shape=(), dtype=float32)
tf.Tensor(426172060000.0, shape=(), dtype=float32)
tf.Tensor(426169340000.0, shape=(), dtype=float32)
tf.Tensor(426166650000.0, shape=(), dtype=float32)
tf.Tensor(426163930000.0, shape=(), dtype=float32)
tf.Tensor(426161180000.0, shape=(), dtype=float32)
```

```
tf.Tensor(426158520000.0, shape=(), dtype=float32)
tf.Tensor(426155770000.0, shape=(), dtype=float32)
tf.Tensor(426153050000.0, shape=(), dtype=float32)
tf.Tensor(426150300000.0, shape=(), dtype=float32)
tf.Tensor(426147580000.0, shape=(), dtype=float32)
tf.Tensor(426144900000.0, shape=(), dtype=float32)
tf.Tensor(426142170000.0, shape=(), dtype=float32)
tf.Tensor(426139420000.0, shape=(), dtype=float32)
tf.Tensor(426136730000.0, shape=(), dtype=float32)
tf.Tensor(426134080000.0, shape=(), dtype=float32)
tf.Tensor(426131300000.0, shape=(), dtype=float32)
tf.Tensor(426128600000.0, shape=(), dtype=float32)
tf.Tensor(426125850000.0, shape=(), dtype=float32)
tf.Tensor(426123130000.0, shape=(), dtype=float32)
tf.Tensor(426120400000.0, shape=(), dtype=float32)
tf.Tensor(426117660000.0, shape=(), dtype=float32)
tf.Tensor(426114940000.0, shape=(), dtype=float32)
tf.Tensor(426112320000.0, shape=(), dtype=float32)
tf.Tensor(426109530000.0, shape=(), dtype=float32)
tf.Tensor(426106780000.0, shape=(), dtype=float32)
tf.Tensor(426104060000.0, shape=(), dtype=float32)
tf.Tensor(426101370000.0, shape=(), dtype=float32)
tf.Tensor(426098600000.0, shape=(), dtype=float32)
tf.Tensor(426095900000.0, shape=(), dtype=float32)
tf.Tensor(426093200000.0, shape=(), dtype=float32)
tf.Tensor(426090500000.0, shape=(), dtype=float32)
tf.Tensor(426087780000.0, shape=(), dtype=float32)
tf.Tensor(426085100000.0, shape=(), dtype=float32)
tf.Tensor(426082300000.0, shape=(), dtype=float32)
tf.Tensor(426079620000.0, shape=(), dtype=float32)
tf.Tensor(426076900000.0, shape=(), dtype=float32)
tf.Tensor(426074140000.0, shape=(), dtype=float32)
tf.Tensor(426071460000.0, shape=(), dtype=float32)
tf.Tensor(426068740000.0, shape=(), dtype=float32)
tf.Tensor(426066020000.0, shape=(), dtype=float32)
tf.Tensor(426063330000.0, shape=(), dtype=float32)
```

1.4 Practice exercises for linear regression:

▶ Package pre-loading:

[7]:
```
import pandas as pd
import numpy as np
from tensorflow import add, multiply, keras
```

▶ Data pre-loading:

[8]:
```
housing = pd.read_csv('ref1. King county house sales.csv')
price_log = np.log(np.array(housing['price'], np.float32))
size_log = np.log(np.array(housing['sqft_lot'], np.float32))
```

▶ **Linear regression set-up practice:**

[9]:
```
# Define a linear regression model
def linear_regression(intercept, slope, features=size_log):
    return add(intercept, multiply(features, slope))


# Set loss_function() to take the variables as arguments
def loss_function(intercept, slope, features=size_log, targets=price_log):
    # Set the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Return the mean squared error loss
    return keras.losses.mse(targets, predictions)


# Compute the loss for different slope and intercept values
print(loss_function(0.1, 0.1).numpy())
print(loss_function(0.1, 0.5).numpy())
```

```
145.44653
71.866
```

▶ **Package re-pre-loading:**

[10]:
```
from tensorflow import Variable
import matplotlib.pyplot as plt
```

▶ **Data re-pre-loading:**

[11]:
```
intercept = Variable(5, dtype=np.float32)
slope = Variable(0.001, dtype=np.float32)
```

▶ **Code pre-loading:**

[12]:
```
def plot_results(intercept, slope):
    size_range = np.linspace(6, 14, 100)
    price_pred = [intercept + slope * s for s in size_range]
    plt.scatter(size_log, price_log, color='black')
    plt.plot(size_range, price_pred, linewidth=3.0, color='red')
    plt.xlabel('log(size)')
    plt.ylabel('log(price)')
    plt.title('Scatterplot of data and fitted regression line')
    plt.show()
```

▶ **Linear model training practice:**

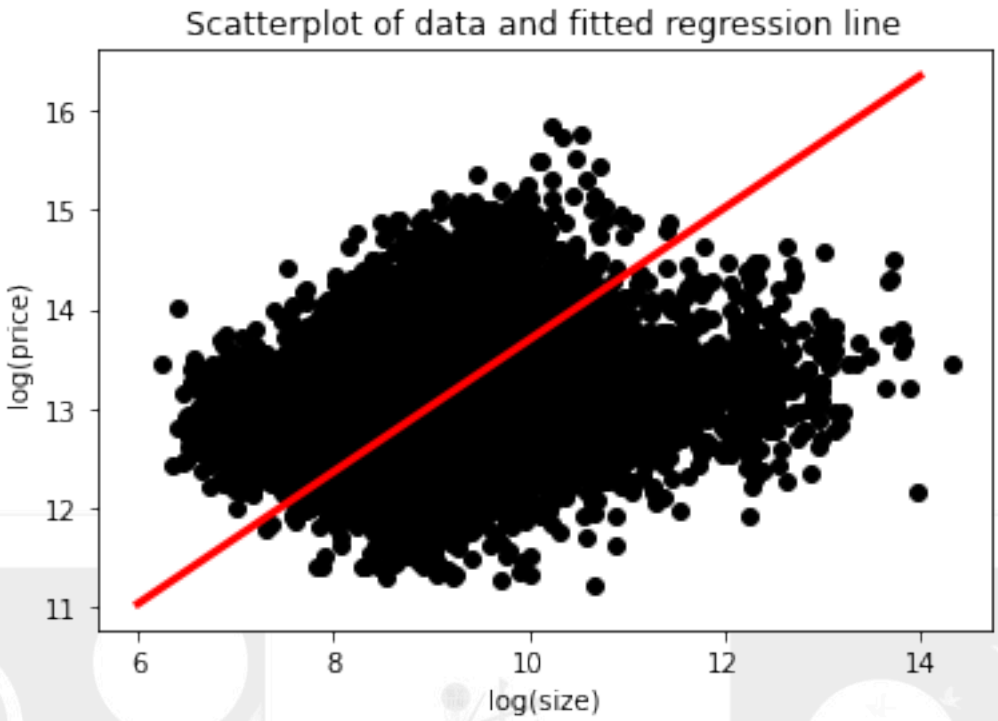
[13]:
```
# Initialize an adam optimizer
opt = keras.optimizers.Adam(0.5)

for j in range(100):
    # Apply minimize, pass the loss function, and supply the variables
    opt.minimize(lambda: loss_function(intercept, slope),
                 var_list=[intercept, slope])

    # Print every 10th value of the loss
    if j % 10 == 0:
        print(loss_function(intercept, slope).numpy())

# Plot data and regression line
plot_results(intercept, slope)
```

```
9.681214
11.737101
1.1297756
1.6701097
0.80904144
0.81259125
0.6220206
0.6118439
0.5933767
0.57028323
```



▶ **Data re-pre-loading:**

[14]:
```
bedrooms = np.array(housing['bedrooms'], np.float32)
params = Variable([0.1, 0.05, 0.02], dtype=np.float32)
```

▶ **Code pre-loading:**

[15]:
```
def print_results(params):
    return print(
        'loss: {:0.3f}, intercept: {:0.3f}, slope_1: {:0.3f}, slope_2: {:0.3f}'
        .format(
            loss_function(params).numpy(), params[0].numpy(),
            params[1].numpy(), params[2].numpy()))
```

▶ **Multiple linear regression practice:**

[16]:
```
# Define the linear regression model
def linear_regression(params, feature1=size_log, feature2=bedrooms):
    return params[0] + feature1 * params[1] + feature2 * params[2]


# Define the loss function
def loss_function(params,
                  targets=price_log,
```

```
                  feature1=size_log,
                  feature2=bedrooms):
    # Set the predicted values
    predictions = linear_regression(params, feature1, feature2)

    # Use the mean absolute error loss
    return keras.losses.mae(targets, predictions)


# Define the optimize operation
opt = keras.optimizers.Adam()

# Perform minimization and print trainable variables
for j in range(10):
    opt.minimize(lambda: loss_function(params), var_list=[params])
    print_results(params)
```

```
loss: 12.418, intercept: 0.101, slope_1: 0.051, slope_2: 0.021
loss: 12.404, intercept: 0.102, slope_1: 0.052, slope_2: 0.022
loss: 12.391, intercept: 0.103, slope_1: 0.053, slope_2: 0.023
loss: 12.377, intercept: 0.104, slope_1: 0.054, slope_2: 0.024
loss: 12.364, intercept: 0.105, slope_1: 0.055, slope_2: 0.025
loss: 12.351, intercept: 0.106, slope_1: 0.056, slope_2: 0.026
loss: 12.337, intercept: 0.107, slope_1: 0.057, slope_2: 0.027
loss: 12.324, intercept: 0.108, slope_1: 0.058, slope_2: 0.028
loss: 12.311, intercept: 0.109, slope_1: 0.059, slope_2: 0.029
loss: 12.297, intercept: 0.110, slope_1: 0.060, slope_2: 0.030
```