Linear Regression using Normal Equation

Khoi Duong
Prof. Chang
CS550
2/5/2023

Table of content

1. Explain and understand the problem

We are trying to implement the procedure in “Chapter 4 - Training Linear Models” in Google Colab in order to train the new dataset with the name “abalone_train.cvs”. In this lab, we will modify the source code from Google Colab to see the result of the regression process. Finally, we can conclude with the best linear regression for the dataset “abalone_train.cvs”

2. Save the “abalone_train.cvs” to Google Drive

The dataset “abalone_train.cvs” has 3320 values and they are separated into 8 attributes: Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, Shell weight, Age

My Drive > Colab Notebooks

Name	Owner	Last modified	File size
Linear Regression using Normal Equation.ipynb	me	Feb 6, 2023 me	49 KB
abalone_train.csv	me	Feb 3, 2023 me	142 KB
knn.ipynb	me	Jan 28, 2023 me	146 KB

3. Copy the source code from the demo

```python
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "training_linear_models"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

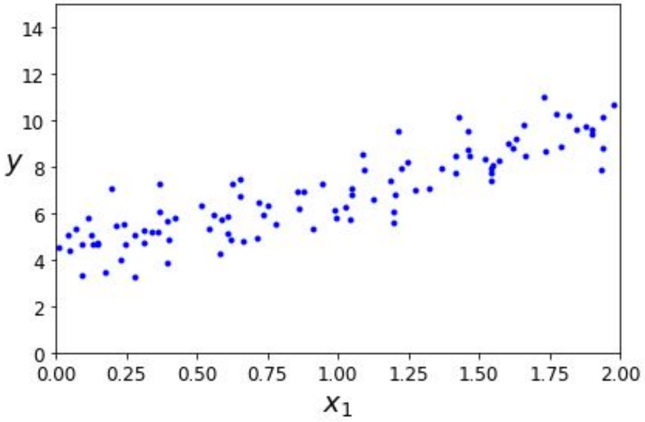
The Normal Equation

```python
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

```python
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
save_fig("generated_data_plot")
plt.show()
```

```
Saving figure generated_data_plot
```




```python
[4] X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
    theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
theta_best
```

```
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
y_predict = X_new_b.dot(theta_best)
y_predict
```

```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 1, 0, 2])
plt.show()
```

The figure in the book actually corresponds to the following code, with a legend and axis labels:

```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 1, 0, 2])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)
lin_reg.intercept_, lin_reg.coef_
```

```
lin_reg.predict(X_new)
```

The `LinearRegression` class is based on the `scipy.linalg.lstsq()` function (the name stands for "least squares"), which you could call directly:

```
[ ] theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
    theta_best_svd
```

This function computes $\mathbf{X}^{+}\mathbf{y}$, where $\mathbf{X}^{+}$ is the *pseudoinverse* of $\mathbf{X}$ (specifically the Moore-Penrose inverse). You can use `np.linalg.pinv()` to compute the pseudoinverse directly:

```
np.linalg.pinv(X_b).dot(y)
```

4. Modify the code in "Linear Regression using the Normal Equation"

Linear Regression

The Normal Equation

```
[ ] import numpy as np
    import pandas as pd

    # X = 2 * np.random.rand(100, 1)
    # y = 4 + 3 * X + np.random.randn(100, 1)


    abalone = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/abalone_train.csv",
        names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
               "Viscera weight", "Shell weight", "Age"])
    X1 = abalone["Length"]
    X2 = np.array(X1)
    X = X2.reshape(-1, 1)

    y1 = abalone["Height"]
    y2 = np.array(y1)
    y = y2.reshape(-1, 1)
```

```
[ ] plt.plot(X, y, "b.")
    plt.xlabel("$x_1$", fontsize=18)
    plt.ylabel("$y$", rotation=0, fontsize=18)
    plt.axis([0, 1, 0, 2])
    save_fig("generated_data_plot")
    plt.show()
```

5. The main task - Modify one more line to make the complete process work

First, we will try to run the source code without modifying to see if there is any error.

```
X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-bccc7d345526> in <module>
----> 1 X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
      2 theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

/usr/local/lib/python3.8/dist-packages/numpy/lib/index_tricks.py in __getitem__(self, key)
    411                 objs[k] = objs[k].astype(final_dtype)
    412 
--> 413         res = self.concatenate(tuple(objs), axis=axis)
    414 
    415         if matrix:

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0 has size 100 and the array at index 1 has size 3320
```

SEARCH STACK OVERFLOW

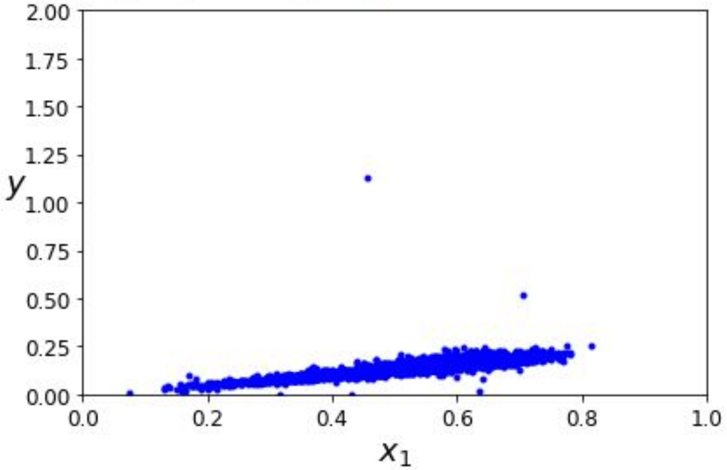
So, the error appears when the sizes of 2 arrays at index 0 and index 1 are different. Concretely, it relates to the size of the dataset of the example source code and the size of “abalone_train.cvs”. In order to make the process work, we need to change the number in the code above (from 100 to 3320, which is the size of the dataset of “abalone_train.cvs”

```
X_b = np.c_[np.ones((3320, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

6. Running the code and result

```
[10] plt.plot(X, y, "b.")
     plt.xlabel("$x_1$", fontsize=18)
     plt.ylabel("$y$", rotation=0, fontsize=18)
     plt.axis([0, 1, 0, 2])
     save_fig("generated_data_plot")
     plt.show()
```

```
Saving figure generated_data_plot
```




```
X_b = np.c_[np.ones((3320, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

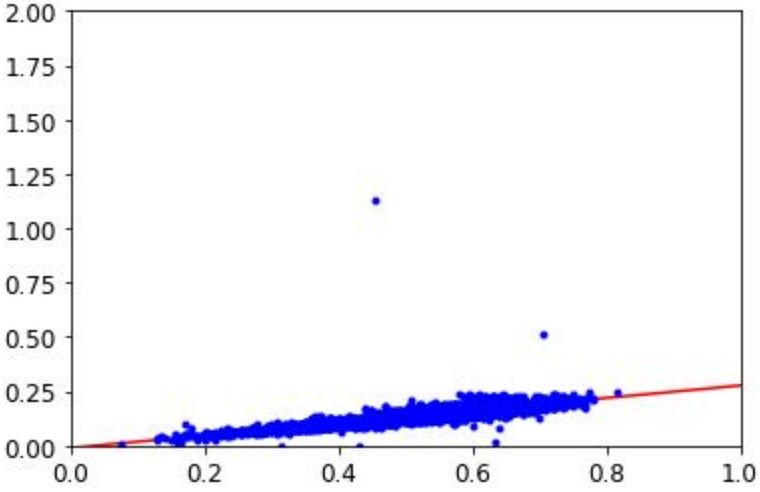
```
[12] theta_best
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

```
[13] X_new = np.array([[0], [2]])
     X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
     y_predict = X_new_b.dot(theta_best)
     y_predict
```

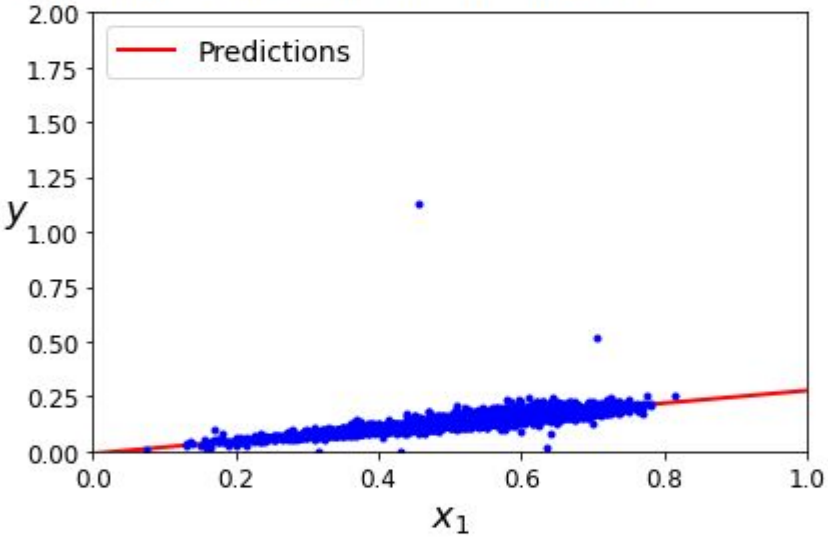
```
array([[-0.0108267 ],
       [ 0.56349837]])
```

```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 1, 0, 2])
plt.show()
```




```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 1, 0, 2])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
Saving figure linear_model_predictions_plot
```



```
[16] from sklearn.linear_model import LinearRegression

     lin_reg = LinearRegression()
     lin_reg.fit(X, y)
     lin_reg.intercept_, lin_reg.coef_
```

```
(array([-0.0108267]), array([[0.28716253]]))
```

```
[17] lin_reg.predict(X_new)
```

```
array([[-0.0108267 ],
       [ 0.56349837]])
```

The `LinearRegression` class is based on the `scipy.linalg.lstsq()` function (the name stands for "least squares"), which you could call directly:

```
[18] theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
     theta_best_svd
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

This function computes $\mathbf{X}^+\mathbf{y}$, where $\mathbf{X}^+$ is the *pseudoinverse* of $\mathbf{X}$ (specifically the Moore-Penrose inverse). You can use `np.linalg.pinv()` to compute the pseudoinverse directly:

```
[19] np.linalg.pinv(X_b).dot(y)
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

7. References

- Linear Regression
  - The Normal Equation
- Three Basic Machine Learning Algorithms
  - Linear Regression
- Colab
  - Get start with Colab
- Linear Regression on Housing.csv Data (Kaggle)
  - handson-ml/datasets/housing/housing.csv
- Learning The TensorFlow Way of Linear Regression - Loading IRIS data