Linear Regression II: Scikit-learn and Google Colab

Mikel Ignacio Barajas Martínez

Ingeniería en Sistemas Inteligentes, 2021, 336483, 202102300012

Machine Learning, 281601

January 28th, 2024

1 Abstract

This paper serves as a proof-of-work for using the Scikit-Learn module in Python for simple linear regression tasks. Three tests are conducted: the first using the example provided in Scikit's documentation; the second modifies the given example to use an 80-20 split of the diabetes dataset (instead of only using 20 samples); and the third using the accident dataset provided by the professor.

2 Introduction

Scikit-learn is a popular data analysis tool for Python that provides simple and efficient prediction tools. Linear regression is one of the included models, using the Ordinary Least Squares (OLS) approach.

3 Test 1

The first test uses the example code present in the documentation, loading the diabetes data directly from Scikit's toy datasets.

3.0.1 Implementation

```
# Code source: Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
import numpy as np

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

3.0.2 Results

- $\mathbf{a_0}$: 938.237
- $\mathbf{a_1}$: 152.918
- **Mean squared error**: 2548.07
- **Coefficient of determination**: 0.47

4 Test 2

Same as the first test, using an 80/20 training-testing split.

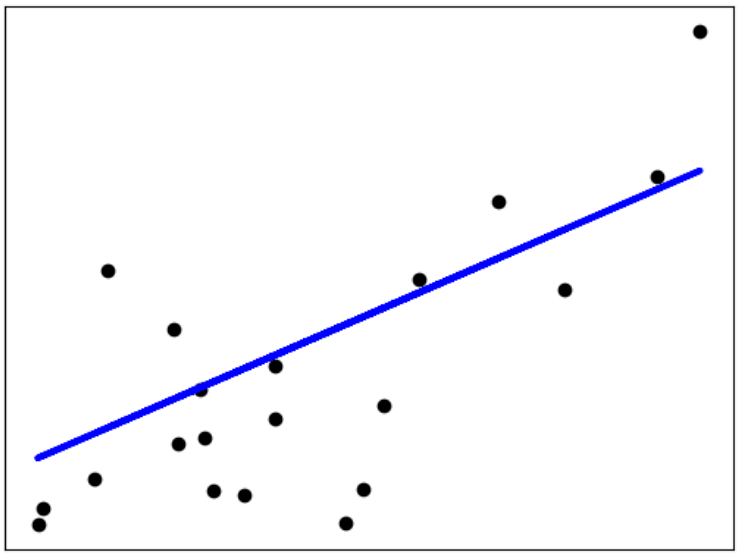

Figure 1: Test 1.

4.0.1 Implementation

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

#split
diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test = train_test_split(

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n \n", regr.coef_, regr.intercept_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
```

```
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.show()
```

4.0.2 Results

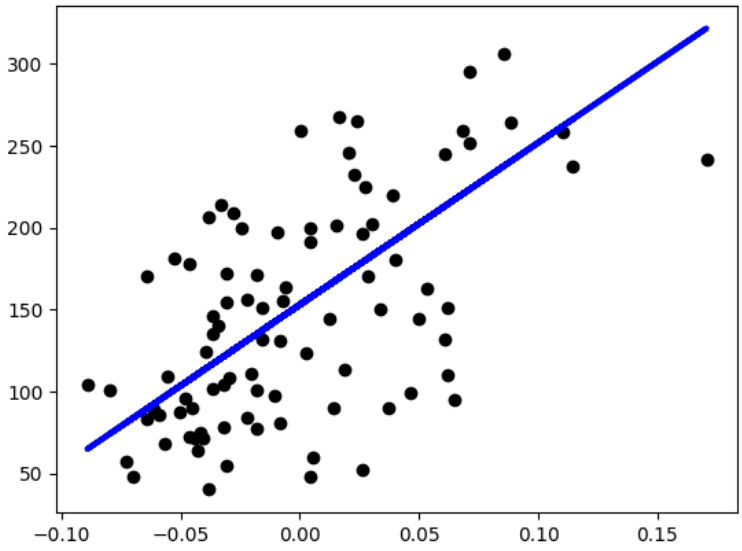

Figure 2: Test 2.

- $\mathbf{a_0}$: 989.10
- $\mathbf{a_1}$: 152.98
- **Mean squared error**: 3089.23
- **Coefficient of determination**: 0.31

5 Test 3

Same approach, using the accidents dataset.

5.0.1 Implementation

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn import linear_model
```

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the accidents dataset
df = pd.read_csv("/content/gdrive/My Drive/ML/accidents.csv", header=None)
x = df[0]
y = df[1]

population_x = x.values.reshape(-1, 1)
accidents_y = y.values.reshape(-1, 1)

# Split the data into training/testing sets
x_train, x_test, y_train, y_test = train_test_split(population_x, accidents_y, test_size

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)

# The coefficients
print("Coefficients: \n \n", regr.coef_, regr.intercept_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(y_test, y_pred))

# Plot outputs
plt.scatter(x_test, y_test, color="black")
plt.scatter(x_train, y_train, color="gray")
plt.plot(x_test, y_pred, color="blue", linewidth=3)

plt.show()
```

5.0.2 Results

- $\mathbf{a_0}$: 9.542e-08665
- $\mathbf{a_1}$: 240.
- **Mean squared error**: 459955.83
- **Coefficient of determination**: 0.78

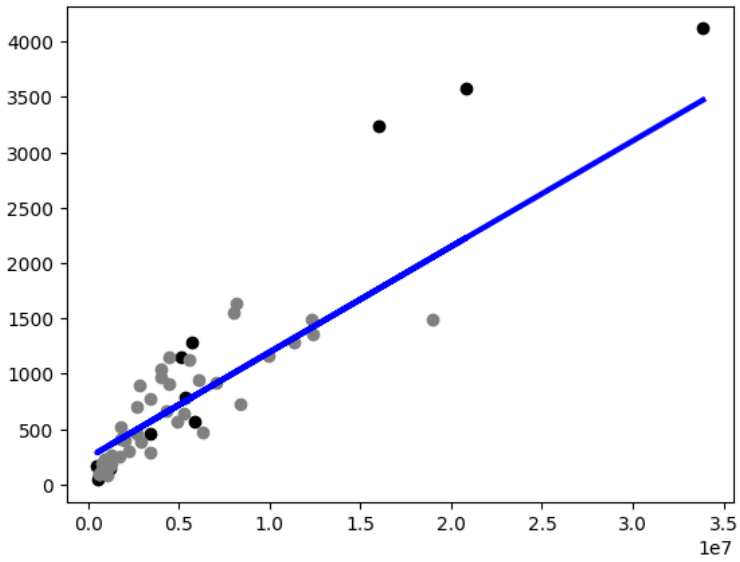

Figure 3: Test 3 (training data in black).

6 Conclusions

I accidentally did most of the assignment for the previous submission, so I had already encountered and google-searched my way through the basic troubleshooting process (such as loading the .csv file, indicating that it does not contain headers, and dimensionality issues). Still, testing and learning about the conventions of any tool you plan on using is not superfluous.

7 References

Cuevas, J. (2020). Handouts on Classification Algorithms. DOI:10.13140/RG.2.2.23597.03043/1