­

**Implementation of Linear regression**

**Using Numpy and Sci-kit Learn**

**Finding the linear relationship between weight of the car and carbon dioxide emission from the car using a dataset**

# Introduction

This study aims to explore the implementation of simple linear regression using the Numpy and Sci-kit libraries with a given dataset. This study critically compares Numpy and Sci-kit learn for implementing a simple linear regression model for predicting the carbon dioxide emission from cars using the feature weight and analysing their effectiveness in the implementation process.

# Methodology

# 2.1 Linear Regression

Linear regression analysis is a statistical method used to estimate the relationship between two variables. It is used to predict the value of a dependent variable based on the value of an independent variable. The dependent variable is the variable being predicted, while the independent variable is the variable used to make the prediction.

According to Yan & Su, (2009), simple linear regression model can be represented in the form

y = β0 + β1 x + ε , where the independent variable is x, the dependent variable is y, the y intercept is β0; the slope of the simple linear regression line is β1; and the random error is represented by  ε.

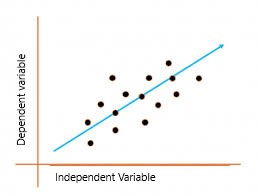


Figure 1 Linear Regression

The linear relationship between the features(X) and target labels (y) variables are illustrated in the above graph. The blue line indicates the best-fit straight line. The objective of the model is to make the best-fitting line out of the available data points.

According to Roy (2023), linear regression can be applied to categorical data with minor adjustments. The author claims that numerical data is the most typical setting for linear regression. Continuous factors like age, income, temperature, and so on are included in this. Categorical data can also be handled by linear regression with the use of strategies like one-hot encoding. One-hot encoding converts each categorical variables into binary indicators (0,1). Similarly ordinal variables are subjected into ordinal encoding for this type of transformations. Time series data can also be employed with linear regression.

## Python Libraries

In this, implementation two python libraries are used namely Numpy and Scikit.This section will briefly discuss about both libraries and its significance in the implementation of linear regression.

Numpy

Cakmak and Cuhadaroglu (2018) claim that the most essential Python package for scientific computing is called NumPy or Numerical Python which serves as the foundation for numerous other packages. NumPy is an open-source package for Python and mainly used in complex mathematical operations. The package provides several fastest predefined functions that can be used to calculate the complex mathematical operations. The primary goal of implementation is to manage arrays and matrices (Ranjani et al. 2019). Many helpful mathematical functions, such as those for Fourier transformation, random number generation, and linear algebra, are available in this library. (Idris, 2015). Although, lists in Python can replace arrays, they process data slowly. With NumPy, you can create an array object up to 50 times faster than with standard Python lists. An array object with multiple dimensions in NumPy is called ndarray, which is comprised of two components. This array object is a multidimensional homogenous array with a constant number of items. It is homogeneous because every item in it is of the same kind and size (Atharva & Shubham 2020). Each ndarray is associated with just one type of data type, which is indicated by another NumPy object called 'dtype' (data-type).The first component is the actual data and the second one is the metadata that describes the information about the data. Most array functions do not modify the raw data. Metadata is the only element which is altered (Idris, 2015). The author suggests that Numpy has many more data types than Python because Python only has three types: integer, float, and complex. This is insufficient for scientific computing. In actual use, we require even more types with various degrees of precision and, consequently, different types of memory sizes. Most NumPy numerical types have a number as their final character which shows how many bits are related to the type.

Sci-kit

Scikit-learn is a Python machine-learning package that is open source, offers a wide range of features and is considered to be one of the most comprehensive machine-learning libraries available (Hao & Ho, 2019). The author states the Scikit-learn expands upon NumPy, SciPy, and matplotlib and offers different machine learning methods, one of which is linear regression. This study claims that Scikit-learn offers a number of useful functions for carrying out various preprocessing and data manipulation techniques. The NumPy array is the primary data structure which used in Scikit-learn functions.This tool provides in-built functions and modules for building the model and for evaluation of the model’s performance.

2.3 Critical Analysis

In the project, a linear regression model has been implemented to find the linear relationship between weight of the car and carbon dioxide emission from a given dataset cars.csv. The implementation has been carried out using two tools - Numpy and Sci-kit. The first program demonstrates the implementation of linear regression using Numpy, while the second program illustrates the same implementation using the Sci-kit library. To implement simple linear regression using a cars dataset, following steps need to be followed.

To implement the model using the numpy tool, the essential package called Numpy is imported. For visualization purposes, Matplotlib is imported. Additionally, the dataset is loaded as a dataframe using pandas and timeit module is also imported to calculate the execution time.

import numpy as np

import matplotlib.pyplot as plt

import timeit

The Numpy uses a custom linear regression implementation that involves a manual optimization loop.

class SimpleLinearRegression:

    def fit(self, X, y):

        #//code adapted from Batuhan, 2019

        X\_b = np.c\_[np.ones((len(X), 1)), X]  # Add a bias term

        params = np.linalg.inv(X\_b.T.dot(X\_b)).dot(X\_b.T).dot(y)

        #//end of the code

        return params

    def predict(self, X, params):

        X\_b = np.c\_[np.ones((len(X), 1)), X]

        return X\_b.dot(params)

In numpy, a custom implementation of linear regression is created as the library does not have any in-built function to perform the task. The ordinary least squares (OLS)approach is used to implement simple linear regression in the SimpleLinearRegression class.In the above code,the fit() add a bias term and calculate the parameters using OLS and return it.The predict() returns the predicted values.

df = pd.read\_csv('data/cars.csv')

    df\_new = df.rename(columns={'Weight': 'wgt','CO2': 'CD'})

    df\_new = df\_new.dropna()

    X = df\_new['wgt'].values.reshape(-1, 1)

    y = df\_new['CD'].values.reshape(-1, 1)

The code snippet above loads the dataset as a data frame, and extracts the weight as features and CO2 emission(CD) as target value.Then some preprocessing steps are done before training the model as it is important for making the data suitable for analysis.

The dropna() drop the rows containing missing values and then reshape both feature and target values into column vectors.

Alternatively, the same linear regression model can also be implemented using the sci-kit library, a machine learning package.The packages listed below are imported first. The library performs linear regression to forecast new values, assess model performance, and fit a linear model to a dataset using the built-in LinearRegression class.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

from sklearn.metrics import r2\_score

from sklearn.preprocessing import StandardScaler

import timeit

Then the same preprocessing steps that are taken in numpy is also used in sklearn as shown below.

# Load the data

df = pd.read\_csv('data/cars.csv')

df\_binary = df.rename(columns={ 'Weight': 'wgt','CO2': 'CO'})# Renaming the columns for easier writing of the code

df\_binary.head()

df\_binary.info()

#//code adapted from Jolly K, 2018

print(df\_binary.isnull().any())

df\_binary = df\_binary.dropna()

#//end of adapted code

The DataFrame's initial few rows are displayed by the head() method, and its details, such as data types and whether any values are missing, are provided by the info() method.

Sci-kit's train\_test\_split() function splits a dataset into 80% for training and 20% for testing. Using the fit() method, a model is built and trained. Finally, predictions are made with the test data from the dataset, which is illustrated given below.

# Splitting into Training/Testing

#//code adapted from Douglass, 2020

test\_dataset\_size = 0.2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, random\_state=1, test\_size=test\_dataset\_size)

# Building the Regression Model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Assess the Accuracy

predictions = model.predict(X\_test)

#//end of adapted code

Although Numpy does not provide any functions for data splitting, a splitting ratio spl is used for splitting the data manually..Here in both implementation some manual predictions are made to test the model.

# Splitting into Training/Testing

    #//code adapted from Rotem 2019

spl = 0.2

    N = len(X)

    sample = int(spl\*N)

    X\_train, X\_test,  = X[:-sample], X[-sample:]

    y\_train, y\_test   = y[:-sample],y[-sample:]

   #//end of the code

    # Building the Regression Model

    reg = SimpleLinearRegression()

    params = reg.fit(X\_train, y\_train)

    # Assess the Accuracy

    predictions = reg.predict(X\_test, params)

Scikit-learn's metrics module has functions like mean\_absolute\_error(), r2\_score(), and mean\_squared\_error() which are used to evaluate model performance.

Obtaining the Mean Absolute Error

    score = metrics.mean\_absolute\_error(y\_test, predictions)

    print('\nMAE:', round(score, 3))

# Calculate R-squared

    r\_squared = r2\_score(y\_test,predictions)

#r\_squared = model.score(X\_test, y\_test)

    print('R-squared (Efficiency):', round(r\_squared, 3))

#//  code adapted from Greg,2013

mse

    mse = metrics.mean\_squared\_error(y\_test, predictions)

    print('MSE:', round(mse, 3))

#rmse

    rmse = np.sqrt(mse)

    print('RMSE:', round(rmse, 3))

#// end of adapted code

In contrast to sklearn, Numpy uses array operations to calculate evaluation metrics which is clearly evident in the code snippet below.

# Calculate R-squared

    mean\_y = np.mean(y\_test)

    total\_variation = np.sum((y\_test - mean\_y) \*\* 2)

    explained\_variation = np.sum((predictions - mean\_y) \*\* 2)

    r\_squared = 1 - (explained\_variation / total\_variation)

    #//end of the code

    print('R-squared (Efficiency):', round(r\_squared, 3))

    # Calculate MSE, MAE, RMSE

    #//code adapted from siammi 2013

    mse = np.mean((predictions - y\_test) \*\* 2)

    #//end of the code

    mae = np.mean(np.abs(predictions - y\_test))

    rmse = np.sqrt(mse)

Then Numpy, creates a scatter plot with a linear regression line and predictions using Matplotlib. The below code generates a scatter plot for actual data and plot a red linear regression line and to show predicted values, with legend, title, and labels adjusted for better comprehension.

plt.scatter(X, y, color='b', label='Actual')

    plt.plot(test\_data, predictions, color='r', marker='x', label='Predicted')

    plt.xlabel('Weight')

    plt.ylabel('CO2 Emission')

    plt.title('Linear Regression\_Numpy:  Weight vs CO2 Emission')

    plt.legend()

    plt.show()

The sci-kit learn make manual predictions using the model same as like Numpy as shown below.

# Using the model

    Xnew = np.array([1200,110,1005,1600]).reshape(-1, 1)

# Feature Scaling for new data

# make a prediction

    ynew = model.predict(Xnew)

# show the inputs and predicted outputs

    for i in range(len(Xnew)):

        print(f"X={Xnew[i][0]}, Predicted={ynew[i][0]}")

Numpy also uses the model and make some predictions with the feature values given below.

# Using the model

    test\_data = np.array([100,110,500,1600,1000,200,1750]).reshape(-1, 1)

    predictions = reg.predict(test\_data, params)

    print(predictions)

Both implementation uses Matplotlib for visualization of the model.The sklearn visualize the result of regression using the code below.

  plt.scatter(X, y, color='b', label='Actual')

    plt.plot(test\_data, predictions, color='r', marker='x', label='Predicted')

    plt.xlabel('Weight')

    plt.ylabel('CO2 Emission')

    plt.title('Linear Regression\_sci-kit:  Weight vs CO2 Emission')

    plt.legend()

    plt.show()

Finally this implementation tries to find the execution time of both models by using timeit() function as given below.

execution\_time = timeit.timeit(numpy\_linear\_regression, number=1)

execution\_time = timeit.timeit(scikit\_linear\_regression, number=1)

This function shows that sklearn takes less time for executing the code as compared to Numpy.

3. Result Analysis

From the implementation section, it is evident that sklearn is more user-friendly and provides a greater level of standardization and modularity, which makes it easier to comprehend and maintain.In contrast Numpy needs more manual implementations using array operations.Both models are subjected to the same dataset.A comparison of evaluation metrics is given in the table below.

Table I Comparison of Performance Evaluation

|  |  |  |
| --- | --- | --- |
| **Evaluation Metrics** | **Sklearn Implementation** | **Numpy Implementation** |
| R\_squared | 0.494 | -3.359 |
| Mean Absolute Error(MAE) | 4.785 | 10.768 |
| Mean Squared Error(MSE) | 26.409 | 135.701 |
| Root Mean Squared Error(RMSE) | 5.139 | 11.649 |
| Execution time of Implementation | 3.06 seconds | 6.34 seconds |

This table shows that Based on the R-squared value of 0.494 from Scikit-learn, it appears that the model is responsible for some of the data's variability. Conversely, the negative R-squared figure of -2.035 provided by NumPy is insignificant in terms of efficiency and may indicate that the model does not fit the data well.

It is evident from the reduced MAE, MSE, and RMSE values that Scikit-learn's forecasts align more closely with actual values. On the other hand, NumPy's predictions appear to deviate more from reality due to its higher error metrics.

In terms of execution time, Scikit-learn outperforms NumPy by completing the task in 3.062 seconds compared to NumPy's 6.656 seconds.

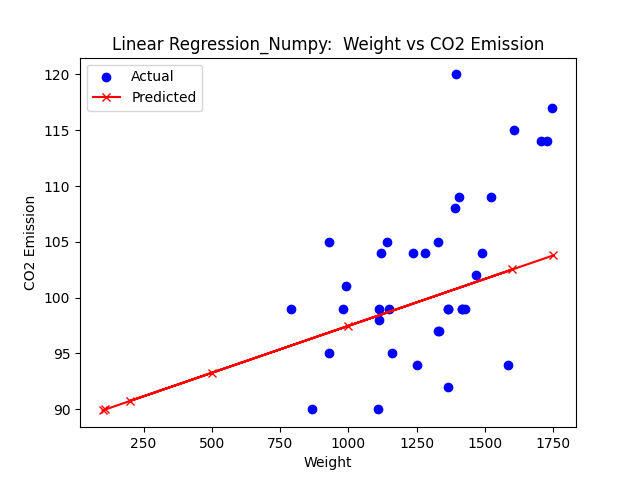


Figure 2 Regression line of the dataset using Numpy

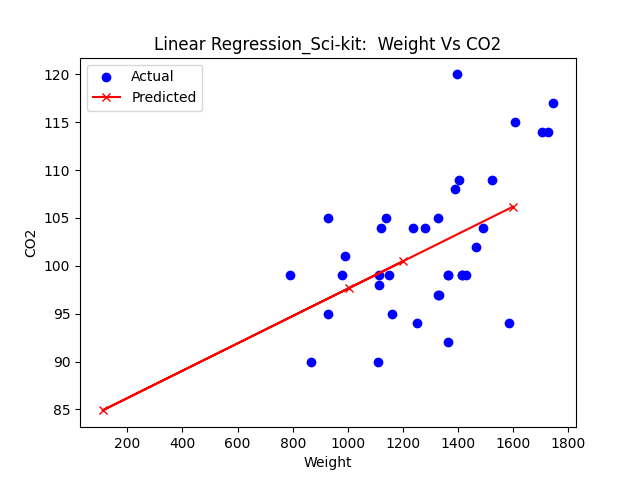


Figure 3 Regression line of the dataset using Sci-kit learn

4. CONCLUSION

Python libraries are crucial for statistical analysis. Scikit-learn's linear regression model provides better prediction accuracy and faster execution time than NumPy's implementation. Scikit-learn is user-friendly and provides exceptional implementations of various machine learning algorithms. NumPy is versatile but may not work well with complex datasets. Sklearn provides a simple and standardized interface for a variety of machine learning techniques and has been designed for high performance.