Lab program no 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

- Part 1: Simple Linear Regression and Multiple Linear Regression(Trial version)
- Part 2 : Github repository

Part 1: Simple Linear Regression

Types of Linear Regression

There are two main types of linear regression:

- **Simple linear regression:** This involves predicting a dependent variable based on a single independent variable.
- **Multiple linear regression:** This involves predicting a dependent variable based on multiple independent variables.

Simple Linear Regression

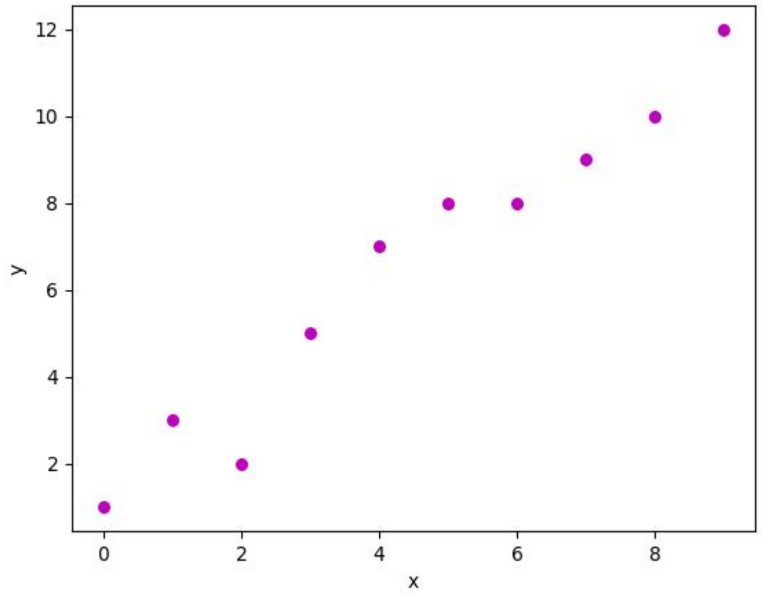
Simple linear regression is an approach for predicting a **response** using a **single feature**. It is one of the most basic machine learning models that a machine learning enthusiast gets to know about. In linear regression, we assume that the two variables i.e. dependent and independent variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x). Let us consider a dataset where we have a value of response y for every feature x:

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

For generality, we define:

*x as **feature vector**, i.e x = [x_1, x_2, …., x_n],*

*y as **response vector**, i.e y = [y_1, y_2, …., y_n]* for **n** observations (in the above example, n=10). A scatter plot of the above dataset looks like this:-



Scatter plot for the randomly generated data

Now, the task is to find a **line that fits best** in the above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in a dataset) This line is called a **regression line**. The equation of the regression line is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here,

- h(x_i) represents the **predicted response value** for i^{th} observation.
- b_0 and b_1 are regression coefficients and represent the **y-intercept** and **slope** of the regression line respectively.

To create our model, we must "learn" or estimate the values of regression coefficients b_0 and b_1. And once we've estimated these coefficients, we can use the model to predict responses!
In this article, we are going to use the principle of **Least Squares**.

Now consider:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

Here, e_i is a **residual error** in ith observation. So, our aim is to minimize the total residual error. We define the squared error or cost function, J as:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} \varepsilon_i^2$$

And our task is to find the value of b_0 and b_1 for which $J(b_0, b_1)$ is minimum! Without going into the mathematical details, we present the result here:

$$\beta_1 = \frac{SS_{xy}}{SS_{xx}}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Where SS_{xy} is the sum of cross-deviations of y and x:

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

And SS_{xx} is the sum of squared deviations of x:

$$SS_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n(\bar{x})^2$$

Estimating Coefficients Function

This function, `estimate_coef()`, takes the input data `x` (independent variable) and `y` (dependent variable) and estimates the coefficients of the linear regression line using the least squares method.

- **Calculating Number of Observations:** `n = np.size(x)` determines the number of data points.
- **Calculating Means:** `m_x = np.mean(x)` and `m_y = np.mean(y)` compute the mean values of `x` and `y`, respectively.
- **Calculating Cross-Deviation and Deviation about x:** `SS_xy = np.sum(y*x) - n*m_y*m_x` and `SS_xx = np.sum(x*x) - n*m_x*m_x` calculate the sum of squared deviations between `x` and `y` and the sum of squared deviations of `x` about its mean, respectively.
- **Calculating Regression Coefficients:** `b_1 = SS_xy / SS_xx` and `b_0 = m_y - b_1*m_x` determine the slope (`b_1`) and intercept (`b_0`) of the regression line using the least squares method.
- **Returning Coefficients:** The function returns the estimated coefficients as a tuple `(b_0, b_1)`.

Plotting Regression Line Function

This function, `plot_regression_line()`, takes the input data `x` (independent variable), `y` (dependent variable), and the estimated coefficients `b` to plot the regression line and the data points.

1. **Plotting Scatter Plot:** `plt.scatter(x, y, color = "m", marker = "o", s = 30)` plots the original data points as a scatter plot with red markers.
2. **Calculating Predicted Response Vector:** `y_pred = b[0] + b[1]*x` calculates the predicted values for `y` based on the estimated coefficients `b`.
3. **Plotting Regression Line:** `plt.plot(x, y_pred, color = "g")` plots the regression line using the predicted values and the independent variable `x`.
4. **Adding Labels:** `plt.xlabel('x')` and `plt.ylabel('y')` label the x-axis as `'x'` and the y-axis as `'y'`, respectively.

Main Function

The provided code implements simple linear regression analysis by defining a function `main()` that performs the following steps:

1. **Data Definition:** Defines the independent variable (x) and dependent variable (y) as NumPy arrays.
2. **Coefficient Estimation:** Calls the `estimate_coef()` function to determine the coefficients of the linear regression line using the provided data.
3. **Printing Coefficients:** Prints the estimated intercept (`b_0`) and slope (`b_1`) of the regression line.

Code

```
def estimate_coef(x, y):
  # number of observations/points
  n = np.size(x)

  # mean of x and y vector
  m_x = np.mean(x)
  m_y = np.mean(y)

  # calculating cross-deviation and deviation about x
  SS_xy = np.sum(y*x) - n*m_y*m_x
  SS_xx = np.sum(x*x) - n*m_x*m_x

  # calculating regression coefficients
  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x

  return (b_0, b_1)
```

```
def plot_regression_line(x, y, b):
  # plotting the actual points as scatter plot
  plt.scatter(x, y, color = "m",
        marker = "o", s = 30)

  # predicted response vector
  y_pred = b[0] + b[1]*x

  # plotting the regression line
  plt.plot(x, y_pred, color = "g")

  # putting labels
  plt.xlabel('x')
  plt.ylabel('y')

def main():
  # observations / data
  x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
  y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

  # estimating coefficients
  b = estimate_coef(x, y)
  print("Estimated coefficients:\nb_0 = {} \
    \nb_1 = {}".format(b
```

Multiple Linear Regression

- **Importing Libraries**
- **Loading the Boston Housing Dataset**
- **Preprocessing Data**
- **Splitting Data into Training and Testing Sets**
- **Creating and Training the Linear Regression Model**
- **Evaluating Model Performance**
- **Plotting the error**

dataset

- http://lib.stat.cmu.edu/datasets/boston

```python
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
```

```python
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+",
                     skiprows=22, header=None)
```

```python
X = np.hstack([raw_df.values[::2, :],
               raw_df.values[1::2, :2]])
y = raw_df.values[1::2, 2]
```

```python
X_train, X_test,\
  y_train, y_test = train_test_split(X, y,
                    test_size=0.4,
                    random_state=1)
```

```python
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
```

```python
# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

```python
# plot for residual error

# setting plot style
plt.style.use('fivethirtyeight')

# plotting residual errors in training data
plt.scatter(reg.predict(X_train),
            reg.predict(X_train) - y_train,
            color="green", s=10,
            label='Train data')

# plotting residual errors in test data
plt.scatter(reg.predict(X_test),
            reg.predict(X_test) - y_test,
            color="blue", s=10,
            label='Test data')

# plotting line for zero residual error
plt.hlines(y=0, xmin=0, xmax=50, linewidth=2)

# plotting legend
plt.legend(loc='upper right')

# plot title
plt.title("Residual errors")

# method call for showing the plot
plt.show()
```

Part 2 : Github repository

- https://github.com/shuv50/Data-Science/blob/main/Simple_Linear_Regression.ipynb

- https://github.com/shuv50/Data-Science/blob/main/Multiple_Linear_Regression.ipynb