Linear Regression

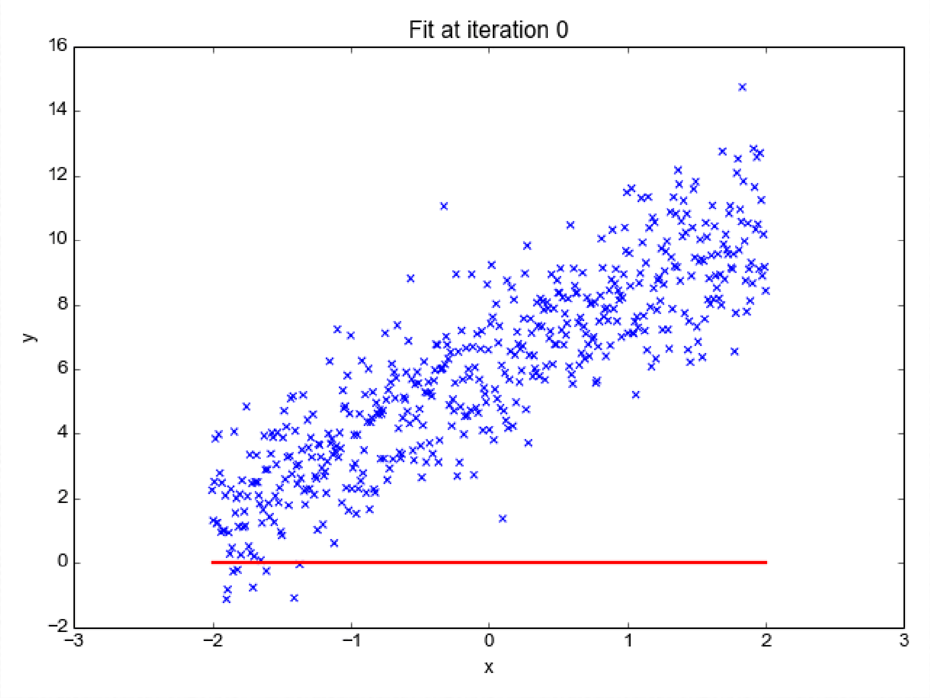
Dear Professor : Mr.Manthouri

Produced By : Ghasemi,morteza

https://github.com/Morteza-Ghasemi

January 2021

Linear Regression



Linear Regression

- There are two types of supervised machine learning algorithms: Regression and classification.
- The term "linearity" in algebra refers to a linear relationship between two or more variables.
- If we draw this relationship in a two dimensional space (between two variables), we get a straight line.

Simple Linear Regression

- We know that the equation of a straight line is basically:
  - $y = mx + b$
  - b is the intercept and m is the slope of the line
- The linear regression algorithm gives us the most optimal value for the intercept and the slope
- The values that we can control are the intercept and slope
- Basically what the linear regression algorithm does is it fits multiple lines on the data points and returns the line that results in the least error.

Multiple Linear Regression

- The same concept can be extended to the cases where there are more than two variables. This is called multiple linear regression.
- A regression model involving multiple variables can be represented as:
  - $y = b_0 + m_1b_1 + m_2b_2 + m_3b_3 + \ldots \ldots M_nb_n$
- This is the equation of a hyper plane.
- a linear regression model in two dimensions is a straight line; in three dimensions it is a plane, and in more than three dimensions, a hyper plane.

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Importing libraries**
  - import numpy as np
  - import matplotlib.pyplot as plt
  - import pandas as pd
  - import seaborn as sns
  - %matplotlib inline

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Importing the Dataset**
  - ✓ from sklearn.datasets import load_boston
  - ✓ boston_dataset = load_boston()

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Dataset**
  - ✓ We print the value of the boston_dataset to understand what it contains.
  - ✓ print(boston_dataset.keys())

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Dataset**
  - *data*: contains the information for various houses
  - *target*: prices of the house
  - *feature_names*: names of the features
  - *DESCR*: describes the dataset

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Dataset**

CRIM: Per capita crime rate by town
ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
INDUS: Proportion of non-retail business acres per town
CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX: Nitric oxide concentration (parts per 10 million)
RM: Average number of rooms per dwelling
AGE: Proportion of owner-occupied units built prior to 1940
DIS: Weighted distances to five Boston employment centers
RAD: Index of accessibility to radial highways
TAX: Full-value property tax rate per $10,000
PTRATIO: Pupil-teacher ratio by town
B: $1000(Bk — 0.63)^2$, where Bk is the proportion of [people of African American descent] by town
LSTAT: Percentage of lower status of the population
MEDV: Median value of owner-occupied homes in $1000s

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Dataset**
  - ✓ We will now load the data into a pandas dataframe using pd.DataFrame.
    - ✓ boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
    - ✓ boston.head()
    - ✓ boston['MEDV'] = boston_dataset.target

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Data preprocessing**
  - After loading the data, it's a good practice to see if there are any missing values in the data. We count the number of missing values for each feature using :
    - ✓ boston.isnull().sum()

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Exploratory Data Analysis**
  - ✓ After loading the data, it's a good practice to see if there are any missing values in the data. We count the number of missing values for each feature using :
    - ✓ boston.isnull().sum()

Linear Regression

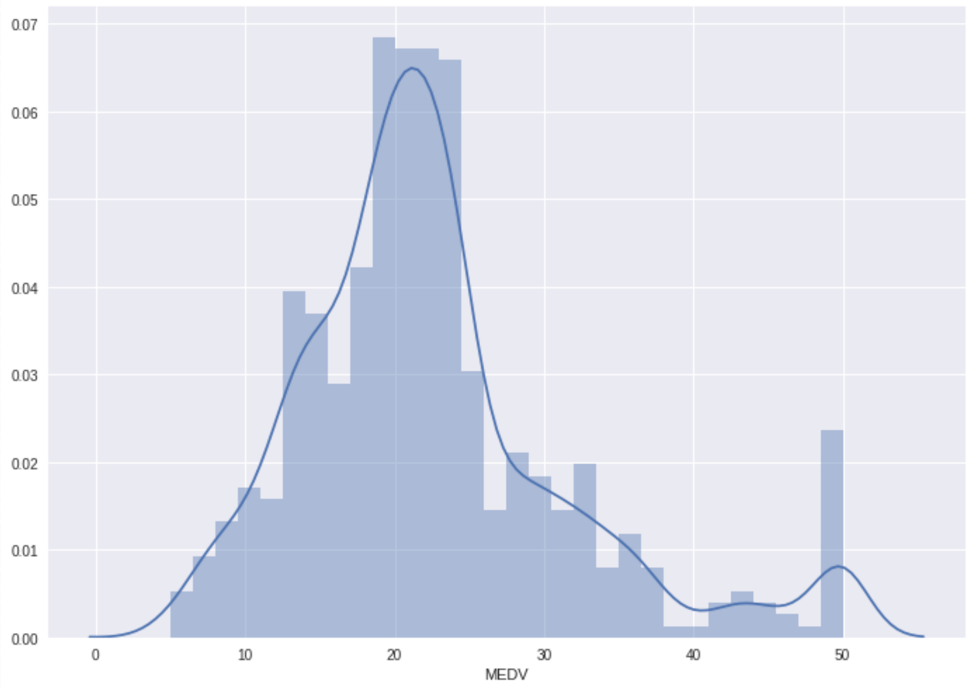
Implementing Linear Regression with Scikit-Learn

- **Exploratory Data Analysis**
  - ✓ first plot the distribution of the target variable
    - ✓ sns.set(rc={'figure.figsize':(11.7,8.27)})
    - ✓ sns.distplot(boston['MEDV'], bins=30)
    - ✓ plt.show()

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Exploratory Data Analysis**
  - We see that the values of MEDV are distributed normally with few outliers.



Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Exploratory Data Analysis**
  - create a correlation matrix that measures the linear relationships between the variables.
    - correlation_matrix = boston.corr().round(2)
    - sns.heatmap(data=correlation_matrix, annot=True)

Linear Regression

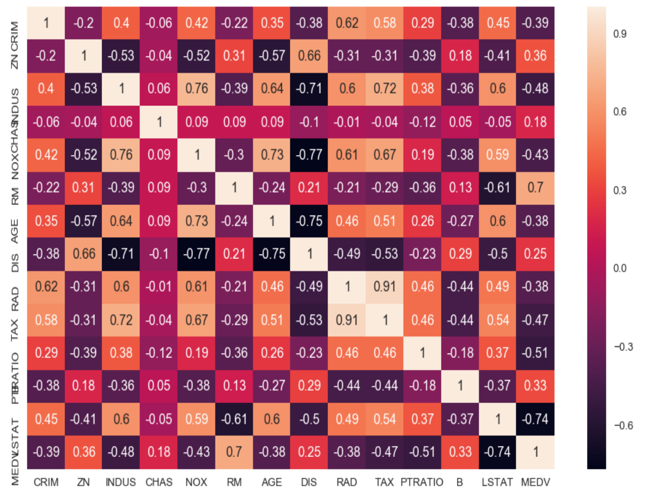
Implementing **Linear Regression** with Scikit-Learn

- **Exploratory Data Analysis**
  - The correlation coefficient ranges from -1 to 1.
  - If the value is close to 1, it means that there is a strong positive correlation between the two variables.
  - When it is close to -1, the variables have a strong negative correlation.

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Observations :**
  - To fit a linear regression model, we select those features which have a high correlation with our target variable MEDV
  - we can see that RM has a strong positive correlation with MEDV(0.7).
  - LSTAT has a high negative correlation with MEDV(-0.7)



Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Observations :**
  - An important point in selecting features for a linear regression model is to check for multi-co-linearity.
  - The features RAD and TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. We should not select both these features together for training the model.

Same goes for the features DIS and AGE which have a correlation of -0.75.

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Observations :**
  - Based on the above observations we will RM and **LSTAT** s our features.
  - Using a scatter plot let’s see how these features vary with MEDV

Linear Regression

Implementing Linear Regression with Scikit-Learn

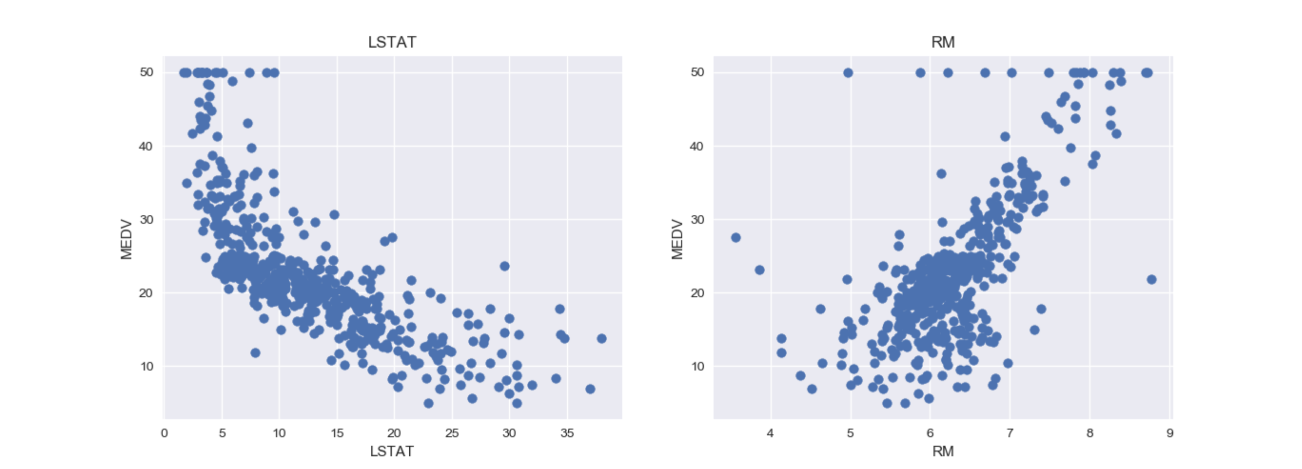
- plt.figure(figsize=(20, 5))
- features = ['LSTAT', 'RM']
- target = boston['MEDV']
- for i, col in enumerate(features):

```
    plt.subplot(1, len(features) , i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Observations :**
  - Using a scatter plot let's see how these features vary with MEDV



Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Observations :**
  - The prices increase as the value of RM increases linearly. There are few outliers and the data seems to be capped at 50.
  - The prices tend to decrease with an increase in LSTAT. Though it doesn't look to be following exactly a linear line.

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Preparing the data for training the model :**
  - We concatenate the LSTAT and RM columns using np.c_ provided by the numpy library.
    - X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
    - Y = boston['MEDV']

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Splitting the data into training and testing sets:**
  - from sklearn.model_selection import train_test_split
  - X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Training and testing the model :**
  - from sklearn.linear_model import LinearRegression
  - from sklearn.metrics import mean_squared_error
  - lin_model = LinearRegression()
  - lin_model.fit(X_train, Y_train)

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Model evaluation :**
  - We will evaluate our model using RMSE and R2-score.
    - y_train_predict = lin_model.predict(X_train)
    - rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
    - r2 = r2_score(Y_train, y_train_predict)

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Model evaluation :**
  - Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as: $\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$
  - Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

    $$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$
  - Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

    $$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

Linear Regression

Implementing **Linear Regression** with Scikit-Learn

- **Model evaluation :**

```
The model performance for training set
--------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701

The model performance for testing set
--------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186952
```

Linear Regression

Implementing Linear Regression with Scikit-Learn

- **Model evaluation :**
  - There are many factors that may have contributed to this inaccuracy, a few of which are listed here:
    - Need more data: Only one year worth of data isn't that much, whereas having multiple years worth could have helped us improve the accuracy quite a bit.
    - Bad assumptions: We made the assumption that this data has a linear relationship, but that might not be the case. Visualizing the data may help you determine that.
    - Poor features: The features we used may not have had a high enough correlation to the values we were trying to predict.

The End