# Supervised Learning using SciKit Learn

Part 1: Linear Regression

Micheș Mihnea Bogdan 30431

Iacob Andrei Ștefan 30431

***Overview & main functionalities***

Supervised learning is a powerful tool used to classify and process data . It infers a function from labeled training data consisting of a set of training examples. Each one of this examples is actually a pair consisting of an input object and a desired output value. The training data provided will help the supervised learning algorithm to infer a function which will later be used for mapping new examples .

In order words, the training data provides the examples of situations and for each example it specifies an outcome. Then the machine will use the training data to build the model which can predict the outcome of the new,useen data that it will be provided afterwards.

It doesn’t act like unsupervised learning, where the model deals more with recognizing patterns without knowing the specific class to which they belong. In supervised learning we have to ’guide’ our system more, giving him from the start a list of the target output values .

Supervised learning problems can be grouped in 2 main categories : regression and classification problems.

* **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
* **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

***Description of one algorithm***

Regression is a method of modelling a target value based on independent predictors. Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable.

While training the model we are given :  
**x:** input training data (univariate – one input variable(parameter))  
**y:** labels to data (supervised learning)

In order to better understand the workings of the algorithm , it is better to look at a small example . 

In the figure above, X (input) is the work experience and Y (output) is the salary of a person. And here is where the linear regression is needed . The regression line is basically the best fit line for our model. So, this regression technique finds out a linear relationship between x (input) and y(output).

One important concept to be mentioned here is the hypothesis function for Linear Regression:

**hθ = θ1 + θ2.x**

And our problem will come down to this: the model gets the best regression fit line by finding the best θ1 and θ2 values( where **θ1:** intercept ;  
**θ2:** coefficient of x).

But in order for us to know how to update the values **θ1 and θ2 accordingly we need to take into consideration the Cost Function J . The** Cost function(J) is the **Root Mean Squared Error (RMSE)** between the predicted y value (pred) and the true y value (y). It is important to minimize as much as possible the error difference between this two values .(the predictions should be closer to the actual values).

And for achieving the required updates the Gradient Descent is used. The idea behind this is quite simple (but harder to make it efficient) : we start with random θ1 and θ2 values and then iteratively update the values, reaching a minimum cost in the end.

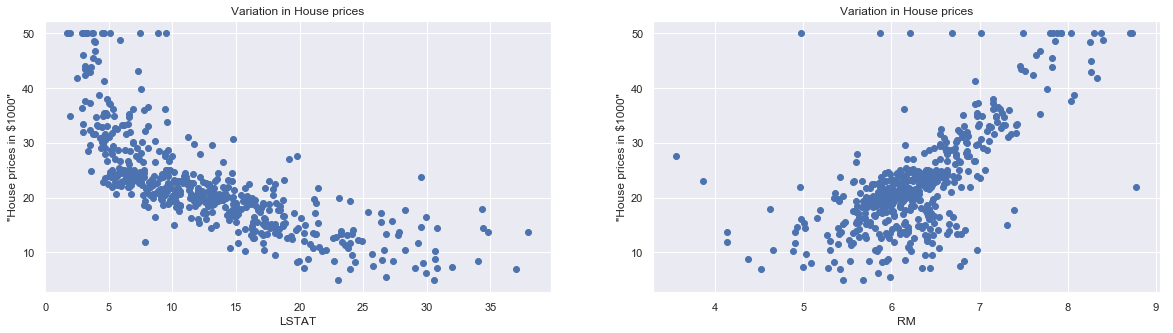
***Example of Supervised Learning***

For this project, we have used Jupyter in order to run the code.

Supervised Machine Learning means training a model using known data in order to be able to obtain predictions about unknown data. One example of this is linear regression, which models the relationship between a response and one or multiple variables, and it is a common statistical data analysis technique.

Let us take an example. The most common one is the prediction of housing prices. For this, we use the Boston data set, included in sklearn.

We have these plots:



They describe the house prices in terms of the status of the population, and the number of rooms, respectively.

We want to feed this data to our model and then obtain decent predictions from it. We start by initializing the data set:

In:

import pandas as pd

import numpy as np

from sklearn import linear\_model

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

import matplotlib.pyplot as plt

from sklearn.datasets import load\_boston

boston = load\_boston()

print(boston.DESCR)

Out:

.. \_boston\_dataset:

Boston house prices dataset

---------------------------

\*\*Data Set Characteristics:\*\*

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

:Attribute Information (in order):

- 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 oxides 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 centres

- 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 blacks by town

- LSTAT % lower status of the population

- MEDV Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

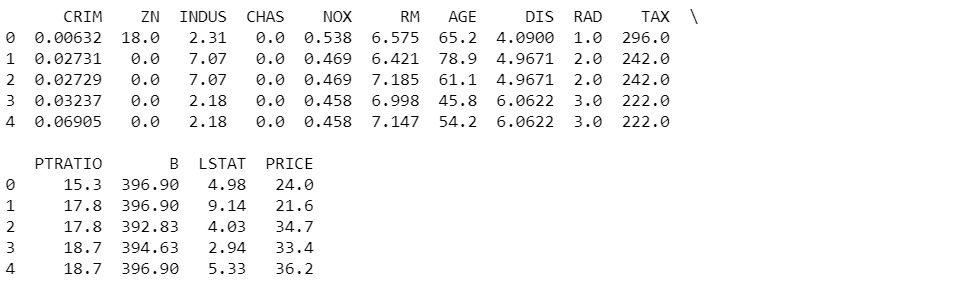
:Creator: Harrison, D. and Rubinfeld, D.L.

Now we have to convert this data into a pandas dataframe:

In:

|  |
| --- |
| Bos= pd.DataFrame(boston.data, columns = boston.feature\_names) |
| bos['PRICE'] = boston.target |  |
| print(bos.head()) |  |

Out:



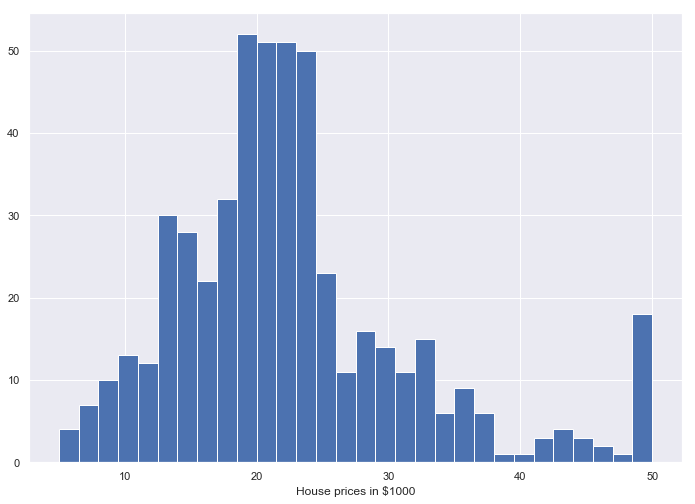
Exploratory Data Analysis is a very important step before training the model. Here, we will use visualizations to understand the relationship of the target variable with other features.

Let’s first plot the distribution of the target variable. We will use the histogram plot function from the matplotlib library.

In:

sns.set(rc={'figure.figsize':(11.7,8.27)})  
plt.hist(bos['PRICE'], bins=30)  
plt.xlabel("House prices in $1000")  
plt.show()

Out:



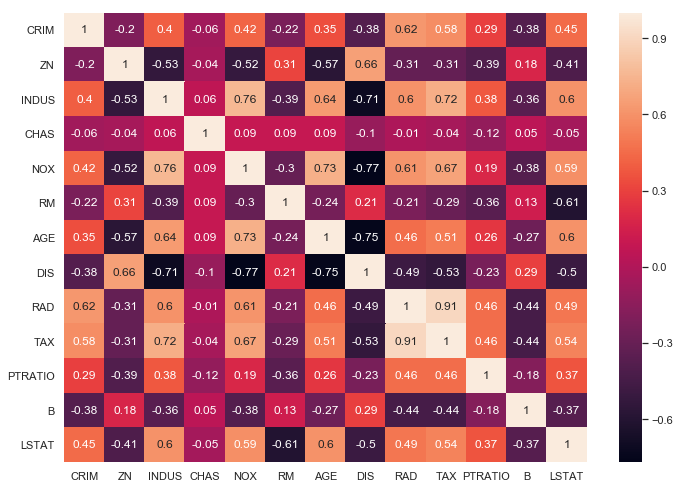
We can see from the plot that the values of **PRICE** are distributed normally with few outliers. Most of the houses are in the 20–24 range (in $1000 scale)

Now, we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be formed by using the corr function from the pandas dataframe library. We will use the heatmap function from the seaborn library to plot the correlation matrix.

In:

|  |
| --- |
|  |
|  | bos\_1 = pd.DataFrame(boston.data, columns = boston.feature\_names) |
|  |  |
|  | correlation\_matrix = bos\_1.corr().round(2) |
|  | sns.heatmap(data=correlation\_matrix, annot=True) |

Out:



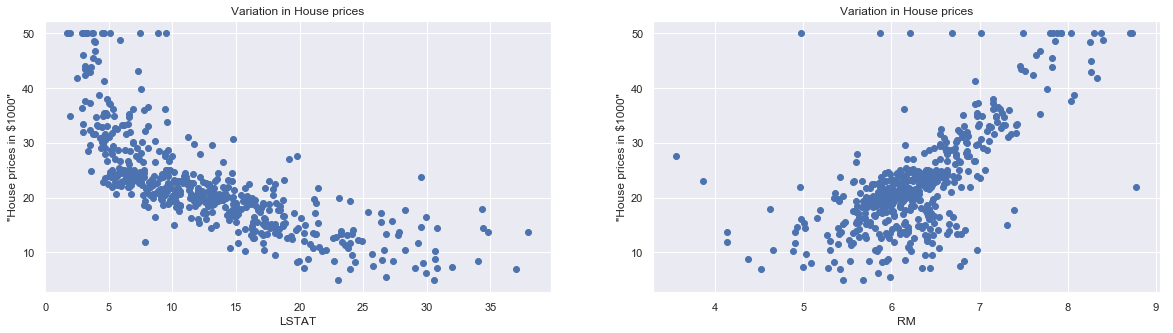
The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation. Notice:

1. By looking at the correlation matrix we can see that RM has a strong positive correlation with PRICE **(0.7)** where as LSTAT has a high negative correlation with PRICE **(-0.74)**.
2. An important point in selecting features for a linear regression model is to check for multicolinearity. The features RAD, TAX have a correlation of **0.91**. These feature pairs are strongly correlated to each other. This can affect the model. Same goes for the features DIS and AGE which have a correlation of -0.75.

But for now we will keep all the features.

In:

|  |
| --- |
| plt.figure(figsize=(20, 5)) |
|  |  |
|  | features = ['LSTAT', 'RM'] |
|  | target = bos['PRICE'] |
|  |  |
|  | for i, col in enumerate(features): |
|  | plt.subplot(1, len(features) , i+1) |
|  | x = bos[col] |
|  | y = target |
|  | plt.scatter(x, y, marker='o') |
|  | plt.title("Variation in House prices") |
|  | plt.xlabel(col) |
|  | plt.ylabel('"House prices in $1000"') |



This is the graph that was presented at the beginning of this document.

We can notice two things here:

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

We want to use this data in order to obtain predictions out of our model. That means we need to draw a line to fit all these points.

Since we need to test our model, we split the data into training and testing sets. We train the model with 80% of the samples and test with the remaining 20%. We do this to assess the model’s performance on unseen data.

To split the data we use train\_test\_split function provided by scikit-learn library. We finally print the shapes of our *training* and *test* set to verify if the splitting has occurred properly.

In:

|  |
| --- |
| X\_train\_1, X\_test\_1, Y\_train\_1, Y\_test\_1 = train\_test\_split(X\_rooms, y\_price, test\_size = 0.2, random\_state=5) |
|  |  |
|  | print(X\_train\_1.shape) |
|  | print(X\_test\_1.shape) |
|  | print(Y\_train\_1.shape) |
|  | print(Y\_test\_1.shape) |

|  |
| --- |
| reg\_1 = LinearRegression() |
|  | reg\_1.fit(X\_train\_1, Y\_train\_1) |
|  |  |
|  | y\_train\_predict\_1 = reg\_1.predict(X\_train\_1) |
|  | rmse = (np.sqrt(mean\_squared\_error(Y\_train\_1, y\_train\_predict\_1))) |
|  | r2 = round(reg\_1.score(X\_train\_1, Y\_train\_1),2) |
|  |  |
|  | print("The model performance for training set") |
|  | print("--------------------------------------") |
|  | print('RMSE is {}'.format(rmse)) |
|  | print('R2 score is {}'.format(r2)) |
|  | print("\n") |

Out:



And now, we want to try the test data as well:

In:

|  |
| --- |
| # model evaluation for test set |
|  |  |
|  | y\_pred\_1 = reg\_1.predict(X\_test\_1) |
|  | rmse = (np.sqrt(mean\_squared\_error(Y\_test\_1, y\_pred\_1))) |
|  | r2 = round(reg\_1.score(X\_test\_1, Y\_test\_1),2) |
|  |  |
|  | print("The model performance for training set") |
|  | print("--------------------------------------") |
|  | print("Root Mean Squared Error: {}".format(rmse)) |
|  | print("R^2: {}".format(r2)) |
|  | print("\n") |

Out:



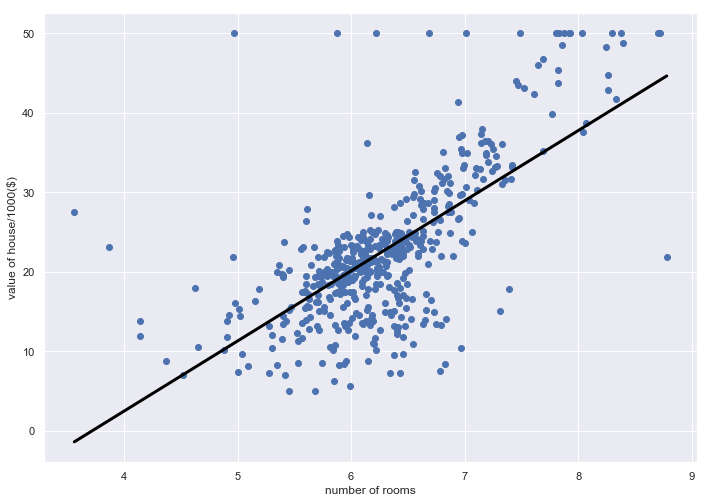
So, the root mean squared error is larger when the model had to handle the test data. It is obvious that the worst case scenario for the predictions is predicting a value for the first time, which is not the case for the training data set.

Now, to draw that fitted line we talked about.

In:

prediction\_space = np.linspace(min(X\_rooms), max(X\_rooms)).reshape(-1,1)   
plt.scatter(X\_rooms,y\_price)  
plt.plot(prediction\_space, reg\_1.predict(prediction\_space), color = 'black', linewidth = 3)  
plt.ylabel('value of house/1000($)')  
plt.xlabel('number of rooms')  
plt.show()

Out:



Now, this plot only takes into account one input variable: the number of rooms the house has. We can already see some serious errors, even for this simple case.

To create a model that takes into account all the features:

In:

|  |
| --- |
| X = bos.drop('PRICE', axis = 1) |
|  | y = bos['PRICE'] |
|  |  |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=42) |
|  |  |
|  | reg\_all = LinearRegression() |
|  | reg\_all.fit(X\_train, y\_train) |
|  |  |
|  | # model evaluation for training set |
|  |  |
|  | y\_train\_predict = reg\_all.predict(X\_train) |
|  | rmse = (np.sqrt(mean\_squared\_error(y\_train, y\_train\_predict))) |
|  | r2 = round(reg\_all.score(X\_train, y\_train),2) |
|  |  |
|  | print("The model performance for training set") |
|  | print("--------------------------------------") |
|  | print('RMSE is {}'.format(rmse)) |
|  | print('R2 score is {}'.format(r2)) |
|  | print("\n") |

Out:



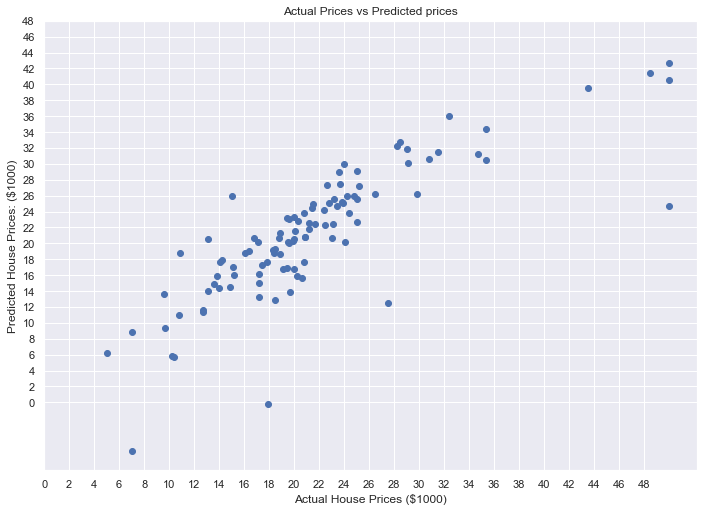
The error is obviously much larger when multiple features are considered.

This time, we can not plot a line in 2D space to illustrate the predictions. But we can see how our model is predicting by plotting a scatter plot between the original house price and predicted house prices.

In:

|  |
| --- |
| plt.scatter(y\_test, y\_pred) |
|  | plt.xlabel("Actual House Prices ($1000)") |
|  | plt.ylabel("Predicted House Prices: ($1000)") |
|  | plt.xticks(range(0, int(max(y\_test)),2)) |
|  | plt.yticks(range(0, int(max(y\_test)),2)) |
|  | plt.title("Actual Prices vs Predicted prices") |

Out:



***Proposed Problem***

Classifying DNA sequences: exploring the world of bioinformatics by using Markov models, K-nearest neighbor algorithms, support vector machines, and other common classifiers to classify short E Coli DNA sequences. This project will use a dataset from the UCI Machine Learning Repository that has 106 DNA sequences, with 57 sequential nucleotides (”base pairs”) each.

During this task, we will:

* Import data from the UCI repository
* Convert text inputs to numerical data
* Build and train classification algorithms
* Compare and contrast classification algorithms

The models we will use: Nearest Neighbors, Gaussian Process, Decision Tree, Random Forest, Neural Net, AdaBoost, Naïve Bayes, SVM Linear, SVM RBF, SVM Sigmoid.

Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/molecular-biology/promoter-gene-sequences/promoters.data