Using simple linear regression techniques with gradient descent to visualise the housing dataset and make predictions

Sedar Olmez - 201276930[1]

[1]University of Leeds, Department of Geography
[1]gysol@leeds.ac.uk
[1]18/19 GEOG5995M Programming for Social Science: Core Skills

ABSTRACT

This paper was written to describe and convey my work for the second coursework of the GEOG5995M module. Linear regression is a supervised machine learning technique used to make predictions using a target variable on a continuous scale. I will adopt some popular algorithms to train a model on a housing dataset which contains information about houses in the suburbs of Boston collected by D.Harrison and D.L.Rubinfield in 1987. Using this data one can try and make forecasts and predictions about various phenomena that may possibly occur. Linear regression can only be used if data related to the experiment exists, especially if this is a supervised method. My code will first read the data, clean it then present it, finally the simple linear regression algorithm will be used to plot data against the target variable. Adding to this, various python libraries will be used to make the data more informative by adding legends to identify outliers and inlier points. Furthermore, sklearn will be used heavily to model non-linear relationships using the housing data. Finally, the goal of my project is to present simple predictions we may be interested in.

1 Introduction

The dataset used in the project can be found here: housing data. The data consists of columns filled with floating point numbers, each column corresponds to the following (left to right):

- CRIME: Crime rate per town per capita
- LAND_ZONE: Residential land zone for lots over 25,000 square feet
- BUSINESS: Non-retail business acres per town
- CHARLES_DUMMY: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NITRIC_ACID: Nitric oxide concentration
- ROOMS: Average number of rooms per dwelling
- AGE: The age of owner occupied units built prior to 1940
- DISTANCES: Weighted distances to five Boston employment centres
- RADIAL: Index of accessibility to radial highways
- PROPERTY_TAX: Full value property tax
- PUPIL_TEACHER_RATIO: Pupil-teacher ratio by town
- B: $1000(Bk-0.63)^2$, where Bk is the proportion of [people of African American Descent] by town.
- LOWER_STATUS: Percentage of lower status of the population
- MEDIAN_VALUE_PROPERTY: Median value of owner-occupied homes in $1000s

The data was cleaned using a simple algorithm to get rid of white space then added to a pandas data frame. Now we can decide which columns we want to focus on specifically as the entire data-set would be very big, less informative and more complex. The code written till now can be used with any type of data, in the realm of data scientists, using pandas to tabulate data is a common trend.

Presenting the data is the next stage, seaborn and matplotlib is used to create scatter graphs and bar charts. A heat map was created considering only the columns we focused on to see the correlation coefficient between each column. Now that we have the data ready for regression, the *simple_linear_regression* object is created, the methods provided are *fit, net_input* and *predict* (I will go into more detail regarding this object in later chapters). Furthermore, sklearn was used heavily with numpy to actually create the regression models, this includes standard scaler for both X and y. The implementation of linear regression from sklearn's linear model package was also used to compare with my *simple_linear_regression* algorithm. Adding to that, RANSACregressor was used from sklearn to detect outliers. A number of different features were adopted, these were quadratic, cubic and linear, the polynomial features package was adopted to fulfil this task, which is also from from sklearn. Finally, *decision tree regression* was used to compare DISTANCE against NITRIC_ACID. The full list of libraries and packages can be viewed in Figure 1.

Figure 1. Libraries and packages

```
# List of all the libraries I will be using.
from __future__ import print_function
import matplotlib.pyplot as plt
import seaborn as sea
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RANSACRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor as DTR
```

I studied the concepts of linear regression from reading the book[1].

2 Data presentation

Data presentation comes in many forms, well, the way in which our data is presented is dependent on various factors, one of these are: what is our data? The data type is quite important because some data types may not be presented informatively using a specific method but may be more informative when we use a different method. My data consists of rows and columns of floating point values. This makes it perfectly viable to use pandas data frame:

Figure 2. Dataset after whitespace removed

```
0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30 396.90   4.98  24.00
 0.02731   0.00   7.070  0  0.4690  6.4210  78.90  4.9671   2  242.0  17.80 396.90   9.14  21.60
 0.02729   0.00   7.070  0  0.4690  7.1850  61.10  4.9671   2  242.0  17.80 392.83   4.03  34.70
 0.03237   0.00   2.180  0  0.4580  6.9980  45.80  6.0622   3  222.0  18.70 394.63   2.94  33.40
 0.06905   0.00   2.180  0  0.4580  7.1470  54.20  6.0622   3  222.0  18.70 396.90   5.33  36.20
```

As you could see in Figure 2, after removing the whitespace the data looks readable. Now it is ready to be processed into a pandas data frame:

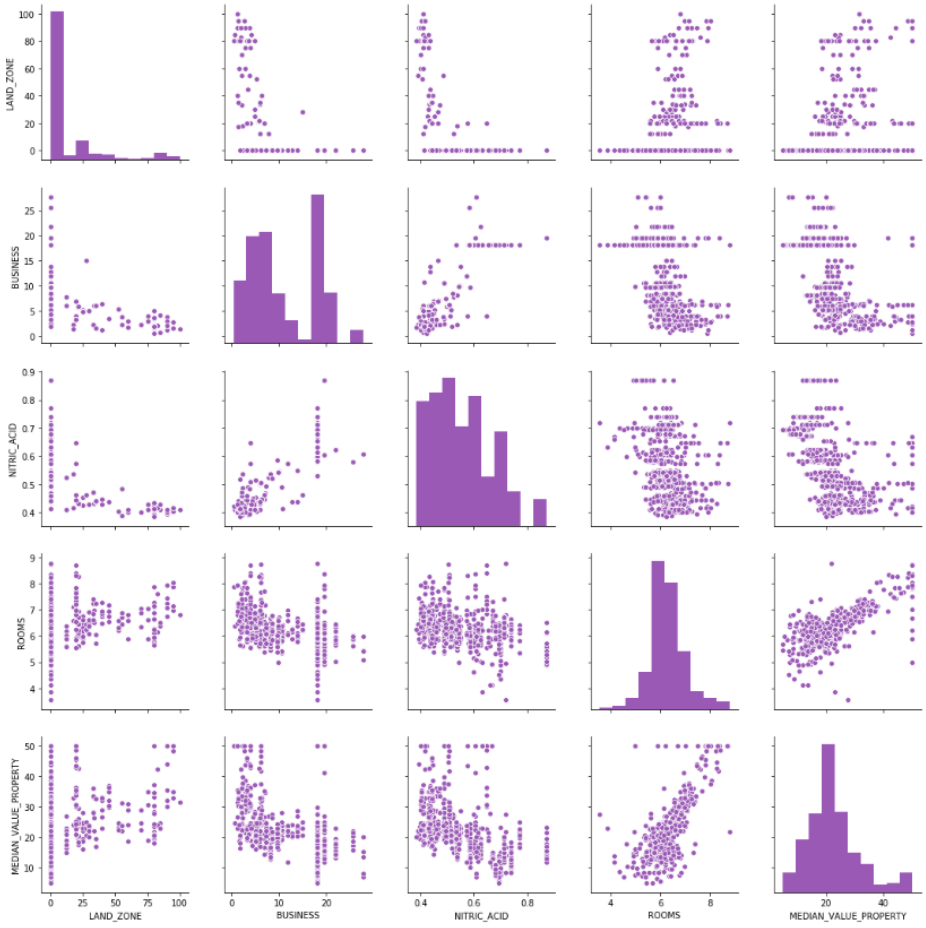
Figure 3. Dataset processed into a data frame

	CRIME	LAND_ZONE	BUSINESS	CHARLES_DUMMY	NITRIC_ACID	ROOMS	AGE	DISTANCES	RADIAL	PROPERTY_TAX	PUPIL_TEACHER_RATIO	B	LOWER_STATUS	MEDIAN_VALUE_PROPERTY
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12	5.21	28.7

Note, I only presented 5 rows, the actual dataset contains **506** rows.

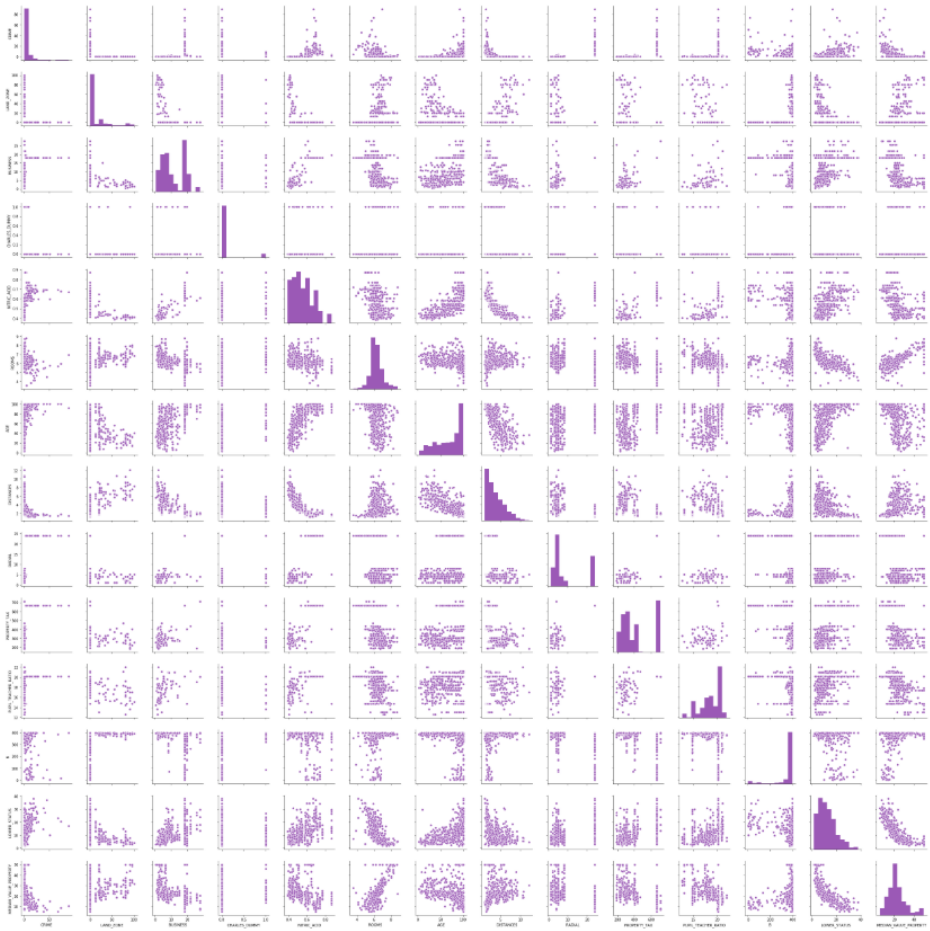
The data is now ready to be used, the seaborn library allows us to perform pair plot which then allows us to draw statistical plots for data column against other columns. The columns we are interested in are: LAND_ZONE, BUSINESS, NITRIC_ACID, ROOMS and MEDIAN_VALUE_PROPERTY. These columns can of course be anything our hearts desire, however, I chose these randomly.

Figure 4. Custom columns presented in pair-plot matrix



As you can see in Figure 4, a 5*x*5 matrix presenting the pair-plot of the columns we focused on, now we can deduce the linear relationships between columns. Now I would like to show you the pair-plot of the entire dataset, this presentation should be enough of a reason as to why I used only a subset:

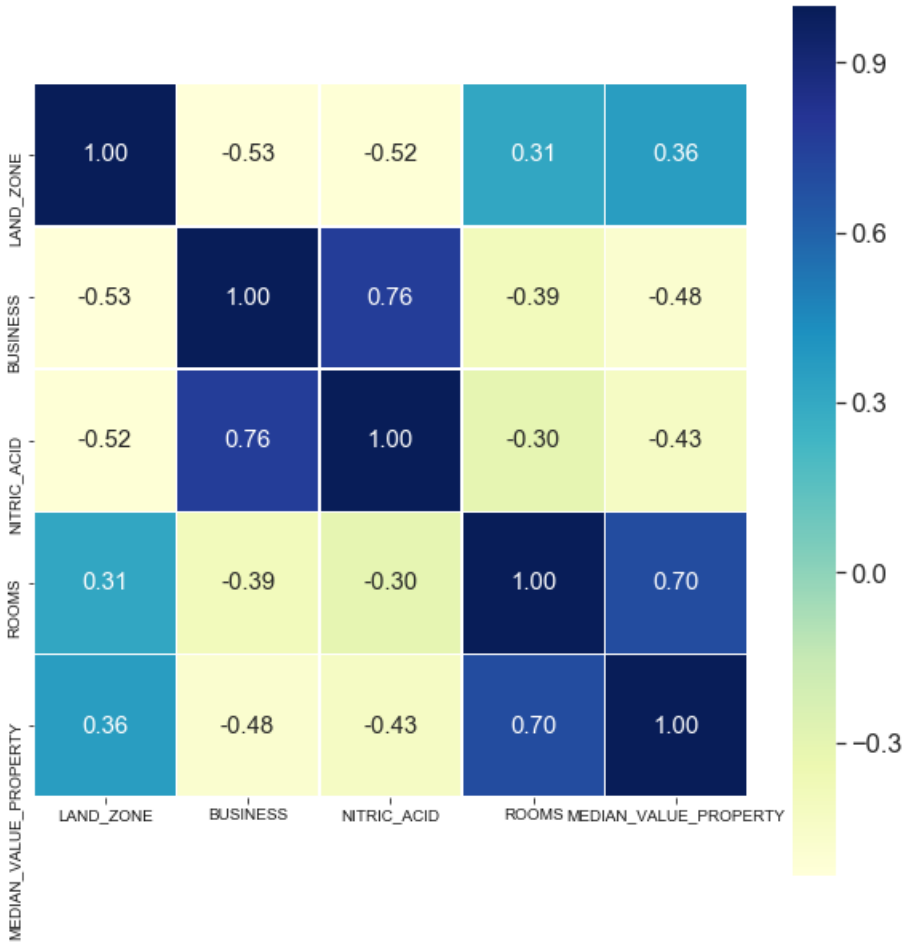
Figure 5. 14 x 14 matrix pair-plot



It is quite clear looking at Figure 5 that when working with a 14x14 pair-plot, things can get a little messy.

Now is the right time to produce a heat map of our selected features to see the correlation coefficient, this will allow us to pick features based on their linear correlations.

Figure 6. Heat map of selected features



In Figure 6, we can see that there is a strong correlation between NITRIC_ACID and BUSINESS and another between ROOMS and MEDIAN_VALUE_PROPERTY.

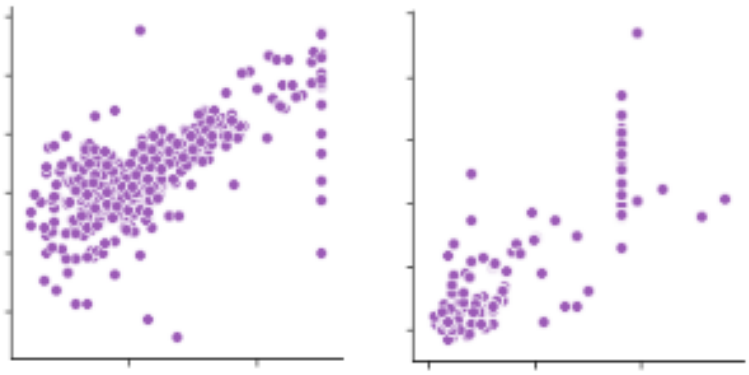
Next we will identify our *target variable* using the material above.

3 Identifying target variables

We expect our *target variable* to have a linear relationship with other features, of course it is perfectly acceptable to have outliers. Now when we look at Figure 6 (as mentioned above) two sets of variables have strong correlation, so our target variable could be: $X = NITRIC\_ACID$ and $y = BUSINESS$ (where y is the responsive variable) or the other way around. Furthermore, for our other pair of features, we could have $X = MEDIAN\_VALUE\_PROPERTY$ and $y = ROOMS$ (X being our target variable).

Let us take a look at the pair-plots of both sets of features:

Figure 7. Linear relationship between features



On the left we have $X = ROOMS$ and on the right we have $X = NITRIC\_ACID$. For my first experiment I will be setting ROOMS as my target variable to try calculate how much house prices cost as the number of rooms are increased.

4 Simple linear regression

The simple linear regression algorithm is mentioned in various text books, as these algorithms are just mathematical formula converted to programmable algorithms there could be various interpretations. I studied two interpretations of the algorithm one from[1] (which was my main source of study) and also[2].

As we are dealing with a single target variable, our formula looks like this: $y = W_0 + W_1 x$ where W_0 is the y-axis intercept and W_1 is the weight coefficient of the target variable. After reading both sources, I came up with an object:

Figure 8. Simple linear regression algorithm using gradient descent

```
class Simple_Linear_Regression_SLR:


    def __init__(self, eta = 0.001, n_iteration = 20):
        # eta = the learning rate
        # n_iteration = the number of times passed over thee training dataset
        self.eta = eta
        self.n_iteration = n_iteration

    # y = target values
    def fit(self, X, y):
        # The weights after fitting
        self.w_ = np.zeros(1 + X.shape[1])
        # The cost values
        self.cost_ = []
        for i in range(self.n_iteration):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors ** 2).sum() / 2.0
            self.cost_.append(cost)
        return self


    def net_input(self, X):
        # Calculates the net input.
        return np.dot(X, self.w_[1:]) + self.w_[0]


    def predict(self, X):
        # Returns class label after unit step.
        return self.net_input(X)
```

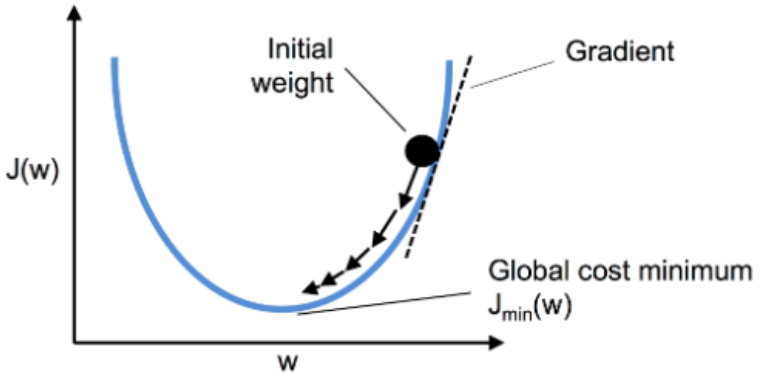
The gradient descent(GD) optimisation technique used alongside linear regression is depicted in Figure 8.

I added comments throughout the algorithm, however, I would like to make it more informative. Some of the variables used are:

- eta: learning rate between 0.0 and 1.0
- n_iteration: passing over the training set
- w_: weights after fitting
- errors: number of misclassifications in each epoch
- X: training vectors
- y: target values

For our method to be supervised learning, an objective function that is to be optimised during the learning process must be adopted and this function is often a cost function that we want to minimise. There is a really good depiction of gradient descent in[1], which I would like to share:

Figure 9. Gradient descent

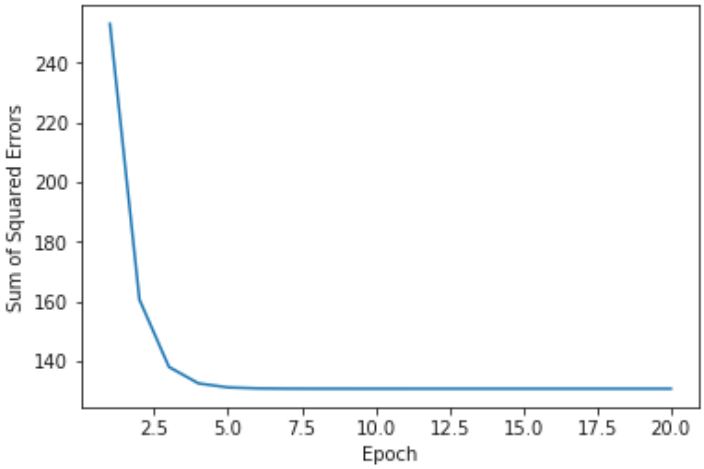


As depicted in Figure 9, we can think of gradient descent as climbing down a hill until a local or global cost minima is reached. For each iteration, a step is taken in the opposite direction of the gradient where the step size is determined by the value of the learning rate.

5 Visualizing regression

Now we have all the tools to see when our simple linear regression algorithm using gradient descent converges. It is always a good idea to plot the cost as a function of the number of epoch passes over the training set to see if it really does converge to a cost minimum. These techniques are normally adopted when dealing with optimisation techniques such as gradient descent, stochastic gradient descent and so forth.

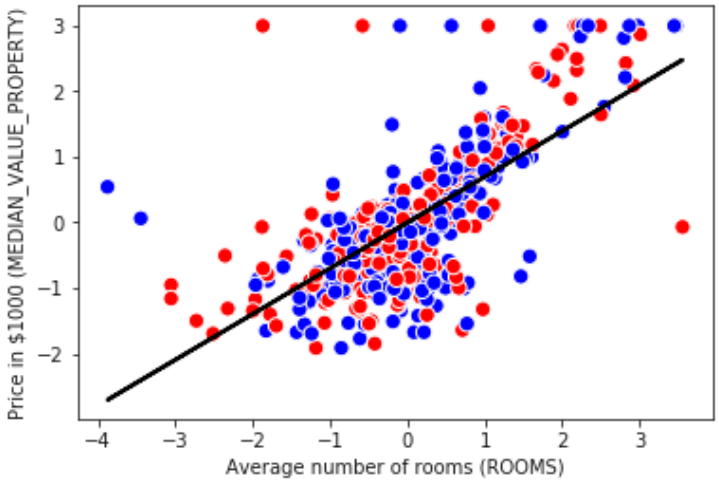
Figure 10. Global cost minima convergence



Analysing Figure 10, we can see that our simple linear regression algorithm converged at the 5th epoch. An epoch is a single presentation of the training dataset.

Let us now visualise how well the linear regression line fits the training data. The number of rooms [ROOMS] will be plotted against the house prices [MEDIAN_VALUE_PROPERTY]:

Figure 11. Average number of rooms compared to house prices



If we look closer at Figure 11, it is clear that the number of rooms alone doesn't explain the house prices very well, however, it is clear that as the number of rooms increase so does the house prices increase.

Lets make a simple prediction with the data we have, lets predict how much a house with 10 rooms could potentially cost:

Figure 12. Average number of rooms compared to house prices

```
# Now lets make some simple predictions. If our house has 10 rooms, what is the predicted price
number_of_rooms = standard_scalar_x.transform(10.0)
price = linear_regression.predict(number_of_rooms)
print("Price in $1000s: %.3f" % \
      standard_scalar_y.inverse_transform(price))
```

```
Price in $1000s: 37.706
```

In Figure 12, we create a *number_of_rooms* variable and instantiate it to the function *standard_scaler_x* performing the function *transform* on the value 10. Then the *price* variable is instantiated to the prediction model's output. Finally, we print a string of the value predicted given our input 10, the cost according to our model would be roughly $37,706.

In the next chapter we will use the RANSAC package from the sklearn library to identify inliers and outliers.

6 Inliers and outliers

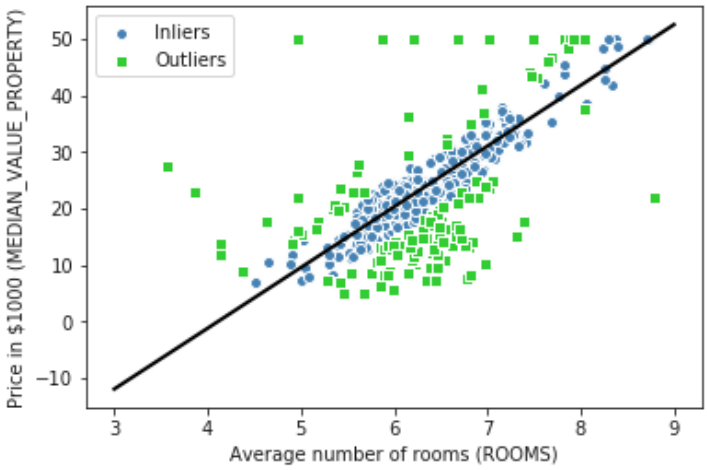
Outliers are very important when it comes to linear regression. The results of linear regression can be heavily impacted by the presence of outliers. In some situations a subset of our data can have a big effect on the estimated model coefficients. The removal of outliers always requires our own judgement as well as our domain knowledge. We will be demonstrating the use of the RANSAC algorithm (**RAN**dom **SA**mple **C**onsensus) to identify outliers and inliers. The psuedocode of the algorithm is as such[1]:

1. Select a random number of samples to be inliers and fit the model.
2. Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers.
3. Refit the model using all inliers.
4. Estimate the error of the fitted model versus the inliers.

5. Terminate the algorithm if the performance meets a certain user-defined threshold or if fixed number of iterations were reached; go back to step 1 otherwise.

The algorithm can be read in the jupyter notebook script which contains all of my compiled code (submitted with this report). Let us now see the outliers from our plot in Figure 11:

Figure 13. Average number of rooms compared to house prices

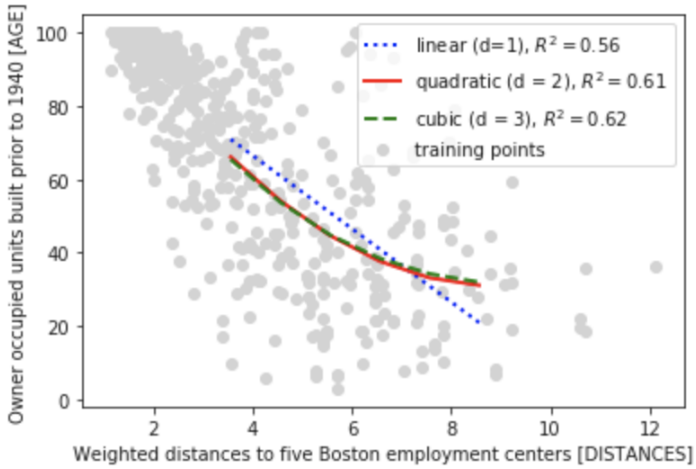


Next, I will add polynomial terms to fit different features like quadratic, cubic and linear fit.

7 Polynomial features

In this section, I chose DISTANCES as my target variable compared against AGE. The library used is sklearn.preprocessing Polynomial Features. This will be used to model non-linear relationships in the housing data:

Figure 14. Polynomial features linear, cubic and quadratic



It is clear from Figure 14, that when the age is increased, less distance can be traveled, and as the age is decreased the person can walk further distances.

At this point, my initial analysis was finished, however, I studied decision tree regression and thought it would be perfectly applicable to induce this method with my dataset.

8 Decision tree regression

One of the main reasons for using decision tree regression was that the data used, did not have to be linear. We grow a decision tree by splitting each node until the leaf nodes have been met. When decision trees are used for classification, we define the entropy as a measure of impurity to determine which feature split maximises the Information Gain(IG) defined as:

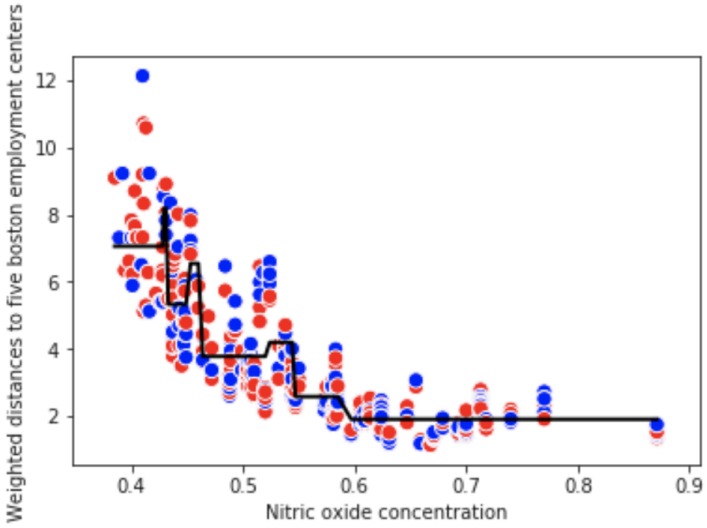
$$IG(D_p, \quad X_i) = I(D_p) - \frac{N_l}{N_p} I(D_l) - \frac{N_r}{N_p} I(D_r)$$

Where:

- X = feature to perform split on.
- N_p = number of samples.
- I = impurity function.
- D_p = subset of training samples at the left and right.
- D_l = subset of training samples at the left.
- D_r = subset of training samples at the right.
- N_l = number of left samples
- N_r = number of right samples

I compared NITRIC_ACID against DISTANCES:

Figure 15. Decision tree regression



In Figure 15, the scatter plots are the observation and the line is the prediction.

9 Conclusion

In this project, I wanted to convey various regression techniques and data visualisation to make predictions. My script can be applied to any dataset not just the housing dataset. My work can be used to teach people starting out in machine learning as it is a very simplified approach to the concept of regression and optimisation.

References

1. Raschka, S. *Python Machine Learning* (2015).

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to Algorithms, Third Edition* (2009).