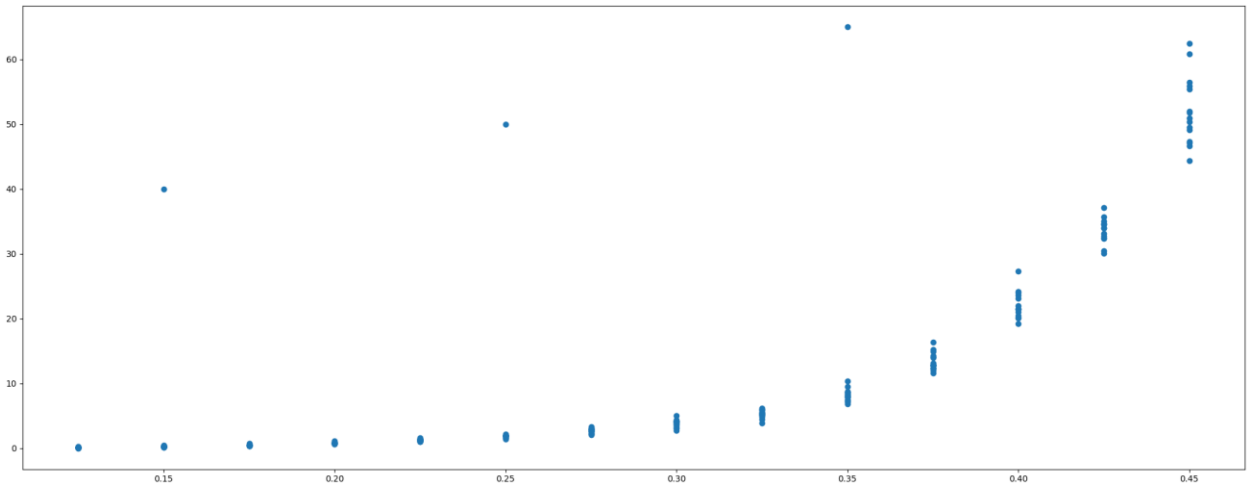
Simple linear regression

Problem

Goal of this program is to find regression model that best fits the data set, which contains X and Y values. X value represents dimensions and speed of ship, Y value represents residual resistance per weight unit.



Train data set

Right from the start we can notice data is not really linear and existance of some outliers. Before doing anything with the data we decided to implement gradient descent and normal equation to compare the results.

Gradient descent

Gradient descent is an optimization technique used to find minimum of an error function using gradient. Our error function is RSS(MSE), and our goal is to minimize it. In other words, to find parameters that give lowest value of that function. Parameters we are searching for in linear regression are m and c.

$$y = m * x + c$$

Parameters of gradient descent algorithm are learning rate, maximum number of iterations and stopping threshold.

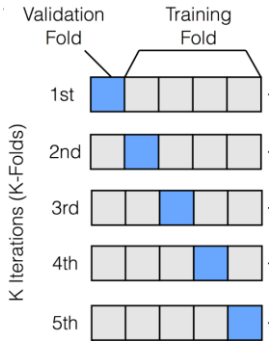
α	Max iterations	Stopping threshold
0.1	10000	1e-9

This means that we will multiply gradient with value of α in each iteration, and unless our cost decreases by less than stopping threshold compared to previous iteration, we will repeat the process until we reach max iteration number.

Using these parameters we got our results to be m = 114.25 and c = -21.85. Before talking about how we validated these values it is worth to mention that using Normal Equation formula we got almost identical values for both of these parameters. To be completelly sure we ran the test with given values here, and also used model from sklearn. Both of these tests returned respectively same parameters.

Preprocessing data

Since test set contains only 4 values and doesn't really test our model well, we decided to cross-validate our training set (K = 5).



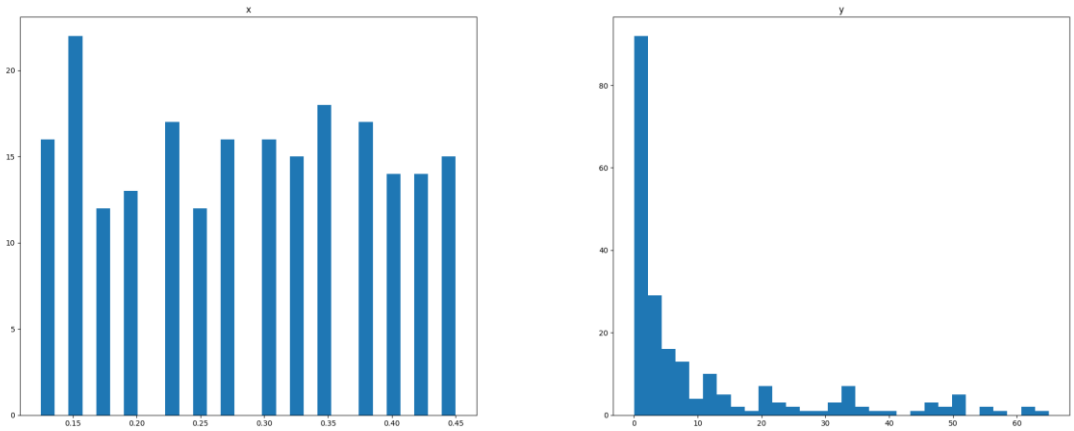
Our performance wasn't really satisfying. Lowest RMSE we got was 7.7 (8.4 on average). To improve this we had to adjust data set. First thing we did was remove outliers. From scatter plot we can notice three points that are noticibly out of place. By further inspecting the training set we can notice

X value	Y value	X value count	Average Y value for X
0.15	40	22	0.25
0.25	50	12	1.82
0.35	65	18	8.15

First and second column represent one observation of X, Y pair. Third column represents number of occurences of X value in the data set and fourth column represents average value of Y for all occurrences of X value except that one observation. Therefore we decided to remove these points from data set.

Box cox

Next thing we can notice by looking at our data is that it doesn't really fit any linear solution. To find the best way to transform our data we need information about skewness and kurtosis.

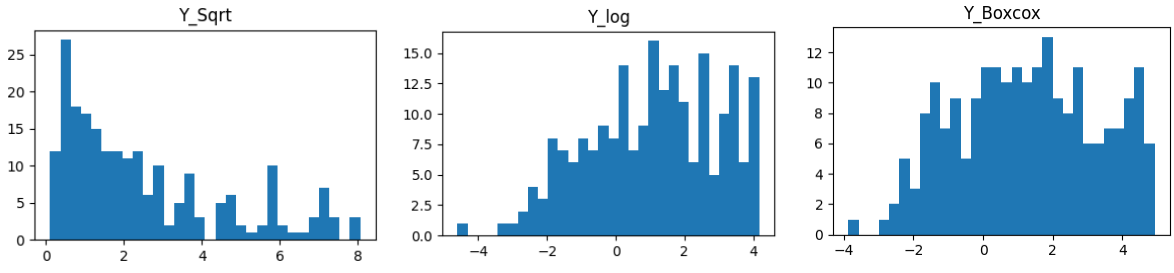


Train data set histograms

	Skewness	Kurtosis
x	-0.014	-1.226
y	1.79	2.257

	Skewness
Fairly Symmetrical	-0.5 to 0.5
Moderate Skewed	-0.5 to -1.0 and 0.5 to 1.0
Highly Skewed	< -1.0 and > 1.0

From this we can deduce that our Y values are highly right skewed so we need to apply some sort of transformation to make it fairly symmetrical. We compared results of three transformation techniques: square root transformation, log transformation and box cox transformation.



	Skewness	Kurtosis
Y_sqrt	1.014	-0.041
Y_log	-0.304	-0.645
Y_box_cox	-0.043	-0.879

We can see that using box cox transformation we managed to transform our skewed data to fairly symmetrical distribution. Box cox only works for positive data(which is fine since in our training set all

values are greater than 0). Using box_cox method from scipy library we found out that best value for our lambda is 0.07755.

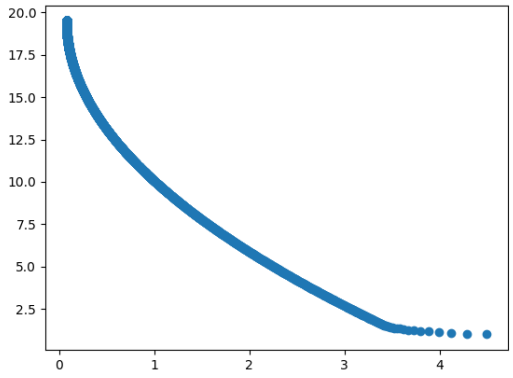
$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, \lambda \neq 0 \\ \log(y), \lambda = 0 \end{cases}$$

Here we have formula for box cox transformation. We can see that it depends on lambda parameter, so we need to find best lambda for our data set. Lambda is calculated by finding maximum log-likelihood. Complete code that we used to test the lambda was based on this.

Final steps

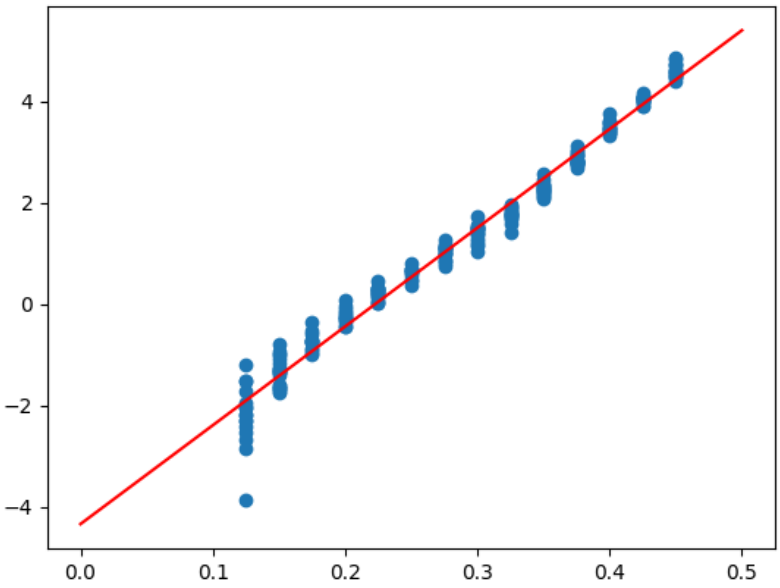
Now that we have transformed our data lets take a look at our gradient descent algorithm once more.

Error function:



Our cost difference is lower than stopping threshold and we manage to break from for loop.

Lets see how our model fits the data after transformations:



Final thing we need to do is test our model. Since we cannot transform our testing values we need to create inverse box cox transformation.

$$y(\lambda) = \begin{cases} e^{\frac{\log(\lambda * y + 1)}{\lambda}}, \lambda \neq 0 \\ \mathrm{e}^{\mathrm{y}}, \lambda = 0 \end{cases}$$

Now we have correct predictions of Y values and we can evaluate our model with RMSE

$$RMSE = \sqrt{\frac{\Sigma\ (Predicted_i - Actual_i)}{N}}$$

With these steps we achieved RMSE of 2.98.

Authors:

Filip Volarić SW54-2018

Svetozar Vulin SW57-2018