TC3020 Machine Learning (Gpo 2)

Prof. Rafael Pérez Torres
Team #04

Luis Felipe Álvarez Sánchez	A01194173
Jesús Omar Cuenca Espino	A01378844
Juan José González Andrews	A01194101
Rodrigo Montemayor Faudoa	A00821976

Linear regression is a mathematical model used to approximate the relationship dependence between a dependent variable Y, and m independent variables X. This model is used to predict continuous valued output. One easy example can be estimating the price(Y) of a house by using the size in sq feet (X).

In this document we will go through the elaboration of the Linear Regression algorithm in python and give a brief analysis of the several variables that we can modify to improve the Linear Regression model. In order to implement the LR and Gradient Descent algorithm, we will be working on the functioning requirements for the Linear Regressor and making sure that it is returning the correct dimensions of the arrays, as well as debugging to understand possible issues. Afterwards, there will be tests performed with different hyperparameters to evaluate the behaviour of the LR when there isn't a mean normalization applied.

Cases where the code fails (if any).

1. There may be some cases where the code crashes eventually due to the learning rate being too big and thus making the cost increase exponentially and making the function divergent. In this case for example with a learning rate of 0.1, the cost explodes and python eventually is unable to handle numbers that large and returns values "nan".

Impact of changing the number of iterations

1. Making the algorithm much slower
2. Making the function without mean normalization to reach convergence eventually. Such is the case with 5'000,000 epochs where as shown in picture #1, it begins to approach similar results as 50,000 epochs using mean normalization, yet it takes much longer, as shown in picture #2.

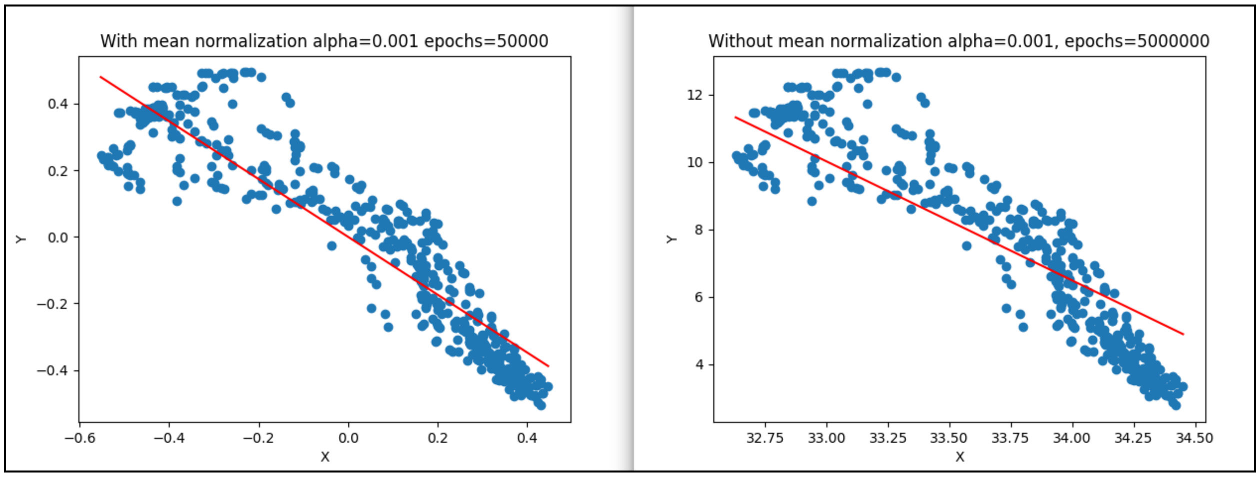
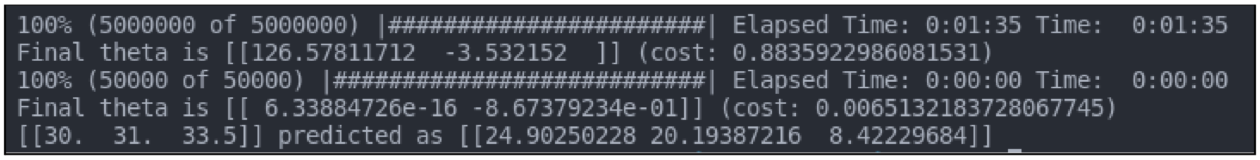


Figure 1. Results obtained for LR Implementation.



```
100% (5000000 of 5000000) |########################| Elapsed Time: 0:01:35 Time:  0:01:35
Final theta is [[126.57811712  -3.532152  ]] (cost: 0.8835922986081531)
100% (50000 of 50000) |############################| Elapsed Time: 0:00:00 Time:  0:00:00
Final theta is [[ 6.33884726e-16 -8.67379234e-01]] (cost: 0.0065132183728067745)
[[30.  31.  33.5]] predicted as [[24.90250228 20.19387216  8.42229684]]
```

Figure 2. Prediction outputs obtained from LR.

Alpha value (learning rate) large vs small.

1. A sufficiently big LR value will make the algorithm crash.
2. Making use of epochs and LR are entwined. And using enough epochs will make even the most inefficient algorithms eventually converge. However, given a small enough LR, will make an efficient algorithm useless as shown in the picture. In contrast to the previous scenario, now the Learning Rate is more of a handicap for the mean normalized GD.

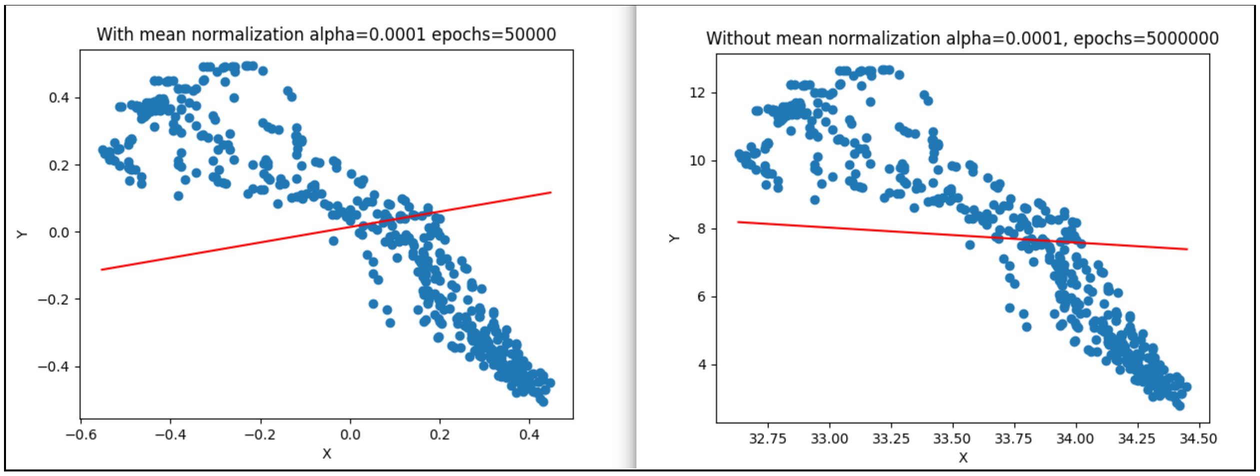


Figure 3. Results obtained from changes in learning rate.

How GD is influenced when mean normalization over the input dataset features is/is not performed.

Mean normalization helps the algorithm in figuring out faster the values for a correct generalization, as it impacts significantly more the Theta values if these are smaller. The good thing is due to the values being meanly distributed, the theta values will remain correct regardless.