Work 1 - Linear Regression and Alternatives

Paulo Mulotto
RA 175541
p175541@dac.unicamp.br

Vitor Aoki
RA 178474
v178474@dac.unicamp.br

I. Introduction

This work aims to test different techniques of linear regression predicting prices of diamonds from their attributes using the Diamonds dataset [1]. It contains 45.849 observations with 9 features. The input of all algorithms was the dataset treated. The categorical variables were transform in dummy variables. Respecting the specification weights were assigned to some features like quality of the cut and diamond color. Besides that we test some exponents in the dataset trying to find the best result.

We will predict the of diamonds in three ways:

- Normal equation
- Linear regression with gradient descent
- SGDRegressor by sklearn [2].

II. Method

For all this methods our parameter to choose a good model is the error function J. In this function we had the sum of all the square difference between our predict and target. This sum give to us the error. The function is described next:

$$J(\Theta_p) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2$$

In this formula, $J(\Theta_p)$ is the error considering all the features p. m is the number of instances of our data set. $h_\Theta(x^{(i)})$ is the predict value and this is calculated like next:

$$h_\Theta(x^{(i)}) = \Theta_0 + x^{(1)}\Theta_1 + x^{(2)}\Theta_2 + x^{(3)}\Theta_3 + ... + x^{(i)}\Theta_i$$

$y^{(i)}$ is the target that we are trying to achieve.

A. Normal Equation

The Normal Equation Method is one way to find the parameters on a Linear Regression. With this method, instead of using iterations to find parameters, it is used an analytic way to compute . For this we use some matrix multiplications to solve. The formula used, is the next:

$$\Theta = (\mathbf{X}^\intercal \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^\intercal \cdot \mathbf{y}$$

Where Θ is the array with parameters, $\mathbf{X}^\intercal$ is the transpose of matrix $\mathbf{X}$ (matrix with all the features) and $\mathbf{y}$ is the matrix with targets.

This method has to calculate $(\mathbf{X}^\intercal \cdot \mathbf{X})^{-1}$ and this cost $O(n)$ (n is the number of features), approximately. So, with a large n, this method can be slow, and the Gradient Descent method can be a better way to do a Linear Regression.

B. Linear Regression with Gradient Descent

Other way to do a Linear Regression, is using Gradient Descent. This method is based in the concept of gradient existing in calculus. For our work we are interested in minimize the cost function J. With Gradient Descent, using parameters Θ we can discover the minimum value of J. We use a iterative function, described next:

$$\Theta_j' = \Theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

In this equation, Θ_j' represent the new Θ that is calculated from the previous Θ_j. α is the learning rate, that is the "step" that the gradient follow to the minimum of the J function. $\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$ represents the difference between the target ($\mathbf{y}$) and the predict ($h_\theta(x^{(i)})$) that is derived from the J cost formula. To compute the minimum of J, in function of Θ, we have to iterate this function, for every Θ_j until we have the convergence in the minimum value of J.

With a little learning rate, the function can be too slow until convergence, and for this need to many iterations. Therefore, the program can be too slow and take a lot of time to converge. On the other hand, using a big learning hate, we can pass for the minimum value of J and instead of converge, we diverge. So, learning rate is the key to solve our problem using Gradient Descent.

Gradient Descent has three different types: Batch, Stochastic and Mini-Batch. The algorithm is the same for each type, the difference is in the number of data used to find Θ. In Batch, we use the whole training set to compute Θ in each iteration. In Stochastic we use only one random instance of the data set to compute Θ in each iteration. And the Mini-Batch type, we use a small random sets of instances to compute Θ in each iteration.

C. SGDRegressor by sklearn

The SGDRegressor function in sklearn, is a function that uses Stochastic Gradient Descent to do the Linear Regression. This function is a optimized function made by the group that made the library Scikit Learn.

III. Proposed Solutions

In our work we used three different ways to achieve the best model for our data set. For this, as mentioned previously, we use Normal Equation Method, Gradient Descent (Batch) and the function SGDRegressor.

First we had tried to find a good model for Normal Equation Method. After, we used this model with Batch Gradient Descent to verify if is a good method too. For this, we tested different ways with our data set and our Normal Equation Method. With these tests, we choose the test that gave to us the minimum error J possible. With this model chosen we applied our Bath Gradient Descent Method and saw that was the best model that we found.

After chose our model, we applied in the Test data set, and achieved satisfactory results. In the final we applied the model with SDGRegressor function to compare.

IV. Experiments and Discussion

For our tests, we decided to separate our training set in two parts: Training Set (70% of data) and Validation Set (30% of data). With this we used our new Training Set to training our model and obtain and used our Validation Set to evaluate if the model that we chose was a good model.

Our data set was compose of 9 features: carat, cut, color, clarity, x, y, z, depth and table.

We started our solution, with Normal Equation Method. For this method, we tested a lot of different models, to find the best model as possible. For this we tested all different possibilities combination of exponents for our features, with the motivation to find what features had most importance in the price of a diamond. With these tests, we chose the model that gave to us the minimums values of error J. These errors was 741373.94 (Training Set) and 745713.04 (Validation Set) the model (exponents) was: (1 for carat, 1 for cut, 1 for color, 1 for clarity, 3 for x, 2 for y, 1 for z, 1 for depth and 3 for table). We tried to drop some features to obtain other models, but we achieve the best results using this models with all the data set.

After this we started to use Batch Gradient Descent. We had tried some different models, but we achieved good results using the same model than Normal Equation Method. Using the same model, and random values for initial Θ values, we had to try different learning rates and iterations numbers. After a lot of tests, that gave to us worse results, like diverge when we tried to achieve the minimum of J, we found these results for learning rate and number of iterations: $\alpha(learning\ rate) = 0.0000000000536$ and $number\ of\ iterations = 300000$.

This was the maximum value of learning rate that we could use, because any number bigger than this, the function diverged. Because this little learning rate, we had to use a bigger number of iterations.

With these values, we achieved these errors: 5214274.08 (Training Set) and 5144261.09 (Validation Set).

Together with our tests of learning rate an number of iterations we plotted graphics of cost x number of iterations, and the graphic that we obtained when the function converged and we achieved that values above, is next:

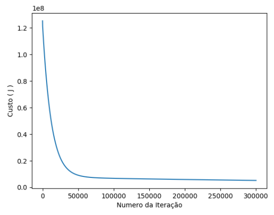

Figure 1. Cost (J) x Number of Iterations

In this graphic we can see that the function starts to converge in the range between 100000 iterations and 150000 iterations. We can see that in the beginning, the function decrease fast, and starts to decrease slower when is near to the minimum.

After chose our best model, we used it in the test model to take our final error results. And the results are the next:

- Normal Equation: 786385.65
- Gradient Descent: 5171015.07

With this, we could conclude that we didn't have over-fitting in our model.

After these tests we tried the function SGDRegressor (using the same learning rate and number of iterations that the Gradient Descent Method implemented) and obtained the next results of errors:

- Training Set: 1268947121791447.0
- Validation Set: 1248968894356939.0
- Test Set: 1575744050891067.8

We plotted graphics of price x instances (for Test Set), and obtained the next results:

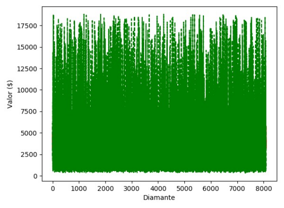

Figure 2. Gradient Descent: Price ($) x Instances

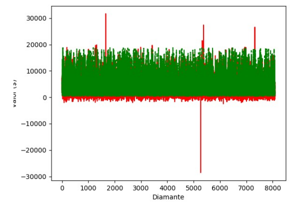

Figure 3. Normal Equation: Price ($) x Instances

The green plot represents the target price and the red plot represents our prediction. In the Gradient Descent graphic we can see that the price target overlays our predict plot. The same don't occur in our Normal Equation graphic. With this, we can see that don't occur over-fitting. In Normal Equation, we can see that in a few instances, the predict was negative. We tried to change this testing different models, but even so, we couldn't avoid that some few instances had negative predicts.

Comparing Normal Equation Method and Gradient Descent Method, we can see that the Normal Equation gave to us a better result of error. We can explain this, because Gradient Descent can converge to a local minimum value, instead of global minimum, of function J, depending on a gradient function starts. The Normal Equation Method is an analytic method and gave to us the minimum value of J possible.

The Bath Gradient Descent Method, when comparing with SGDRegressor function had better results. The explanation for this can be because this function, even using the same method for Gradient Descent, uses the Stochastic type and have different types of optimization and treatments that makes that give to us different values of Θ and consequently, errors.

V. Conclusions and Future Work

After a lot of tests and some results, we achieved the best results as possible. Analyzing all the process we can see how important is the learning rate to achieve the minimum and how much iterations we need to achieve this minimum using a little learning rate.

During the process, we realized too how important is the weight that we give to some features too. While we were testing our models, we plotted some graphics of price x feature, and we realized that some features didn't influence a lot on the final result. But in set with other features, had influence in the result. With this graphics too, we realized that the Linear Regression, and even Polynomial Regression, is not a better way to predict prices of diamonds, because the function that describes these prices is too complex and need to other tools to give good results.

References

[1] https://www.kaggle.com/shivam2503/diamonds

[2] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html

[3] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*