Diamonds In-Depth Analysis: Linear Regression

Farzin Shams, farzinshams95@gmail.com

Introduction

In this work, we explore different alternatives of linear regression on the Kaggle dataset *Diamonds In-Depth Analysis* and compare their performances. The dataset contains the prices and other attributes of almost 54,000 diamonds. The features and their categories are:

- **Price (numerical, target)**: in US dollars ($326–$18,823)
- **Carat (numerical)**: weight of the diamond (0.2–5.01)
- **Cut (categorical)**: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- **Color (categorical)**: diamond colour, from J (worst) to D (best)
- **Clarity (categorical)**: a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- **x (numerical)**: length in mm (0–10.74)
- **y (numerical)**: width in mm (0–58.9)
- **z (numerical)**: depth in mm (0–31.8)
- **Depth (numerical)**: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- **Table (numerical)**: width of top of diamond relative to widest point (43–95)

Feature Cleaning and Engineering

Let's take a first look at the data and its description. Fig. 1 displays the data's first 10 datapoints and Fig. 2 describes the whole dataset.

	carat	cut	color	clarity	x	y	z	depth	table	price
0	1.62	Ideal	I	VS2	7.53	7.58	4.69	62.1	55.0	10501
1	0.30	Premium	G	SI1	4.39	4.35	2.57	58.8	60.0	574
2	1.37	Ideal	F	VS1	7.28	7.22	4.32	59.6	57.0	11649
3	0.33	Fair	H	VVS2	4.40	4.32	2.84	65.1	59.0	922
4	0.32	Very Good	F	VS2	4.43	4.48	2.62	58.8	62.0	602
5	0.51	Premium	E	SI2	5.15	5.12	3.22	62.7	52.0	1205
6	1.50	Good	H	VS2	7.22	7.27	4.61	63.6	58.0	10291
7	0.85	Ideal	G	VS1	6.09	6.11	3.77	61.8	55.0	4373
8	0.33	Ideal	F	VS1	4.46	4.49	2.74	61.2	56.0	723
9	1.10	Ideal	G	VS1	6.69	6.65	4.09	61.3	54.0	6535

Figura 1: Dataset's first 10 datapoints

	carat	x	y	z	depth	table	price
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	5.731157	5.734526	3.538734	61.749405	57.457184	3932.799722
std	0.474011	1.121761	1.142135	0.705699	1.432621	2.234491	3989.439738
min	0.200000	0.000000	0.000000	0.000000	43.000000	43.000000	326.000000
25%	0.400000	4.710000	4.720000	2.910000	61.000000	56.000000	950.000000
50%	0.700000	5.700000	5.710000	3.530000	61.800000	57.000000	2401.000000
75%	1.040000	6.540000	6.540000	4.040000	62.500000	59.000000	5324.250000
max	5.010000	10.740000	58.900000	31.800000	79.000000	95.000000	18823.000000

Figura 2: Dataset Description

First, let's remove the datapoints where either x, y or z are zero, since it doesn't make any sense for a diamond to have a dimension of length zero. We'll also add a constant feature to make bias implementation easier. Next, let's deal with the categorical variables. The usual options are label encoding or one-hot encoding. However, label encoding isn't usually the best option when using a non-tree based model since the categories might not be perfectly ordinal. Moreover, there are only three categorical variables, with a total of 20 different categories, so it makes sense to one-hot encode them, since the increase in number of features won't be too big.

Finally, let's deal with the numerical variables. Fig. 3 shows their distributions. It might be a good idea to try to get their distributions to be as normal as possible so as to preserve heteroscedasticity, which will prevent the model from overprioritizing datapoints with high *price* value. In order to do this, we applied $log(x + 1)$ to all of these features (later, one need only apply $exp(x) - 1$ to recover the correct value), and their final distribution is shown in Fig. 4.

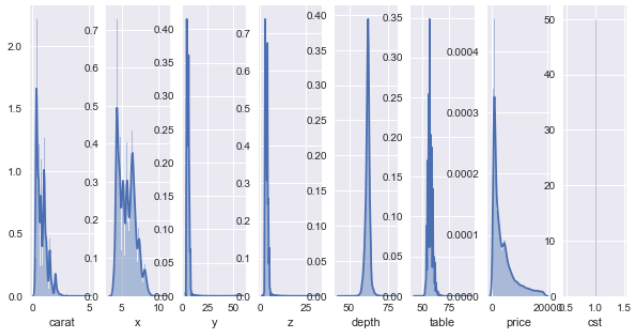

Figura 3: Numerical variable's initial distributions

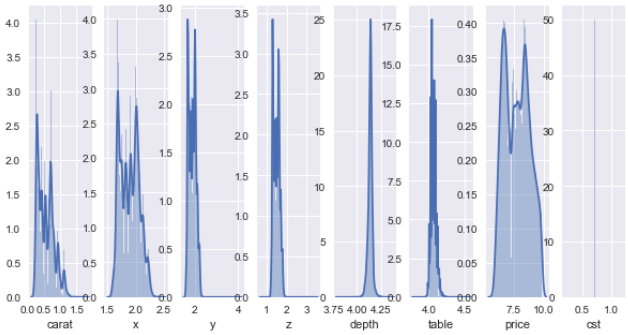

Figura 4: Numerical variable's final distributions

Fig 5 shows the resulting dataset, which has 24 features, ready to be fed into the linear model.

	carat	x	y	z	depth	table	price	cst	cut_Good	cut_Ideal	...	color_H	color_I	color_J	clarity_IF	clarity_SI1	cl
0	0.963174	2.143589	2.149434	1.738710	4.144721	4.025352	9.259321	0.693147	0	1	...	0	1	0	0	0	
0	0.963174	2.143589	2.149434	1.738710	4.144721	4.025352	9.259321	0.693147	0	1	...	0	0	0	0	0	
0	0.438255	1.834180	1.838961	1.444563	4.135167	4.043051	7.705713	0.693147	0	1	...	0	1	0	0	0	
0	0.438255	1.834180	1.838961	1.444563	4.135167	4.043051	7.705713	0.693147	0	1	...	0	0	0	0	0	
1	0.262364	1.684545	1.677097	1.272566	4.091006	4.110874	6.354370	0.693147	0	0	...	0	0	0	0	1	
1	0.262364	1.684545	1.677097	1.272566	4.091006	4.110874	6.354370	0.693147	0	0	...	1	0	0	0	0	
1	0.792993	2.041220	2.036012	1.669592	4.182050	4.043051	8.525558	0.693147	0	0	...	0	0	0	0	1	
1	0.792993	2.041220	2.036012	1.669592	4.182050	4.043051	8.525558	0.693147	0	0	...	1	0	0	0	0	
2	0.862890	2.113843	2.106570	1.671473	4.104295	4.060443	9.363061	0.693147	0	1	...	0	0	0	0	0	
2	0.862890	2.113843	2.106570	1.671473	4.104295	4.060443	9.363061	0.693147	0	0	...	0	0	0	0	1	

Figura 5: Resulting dataset

Linear Regression

We'll be testing three different linear regression methods: one using gradient descent (GD), one using closed-form equations, and *sklearn's* ready-made *SGDRegressor* (SGD). 45849 datapoints ($\sim$ 85%) will be used for training, and 8091 ($\sim$ 15%) for testing. The train set is used for providing the necessary data to train the model; whereas the test set is necessary to determine the point beyond which the model starts to overtrain. Thus, the optimal parameters are the ones where test error is minimized.

Gradient Descent

Fig. 6 shows four different instances of GD optimization with different learning rates. In all four cases, test error minimum occurs early on and is clearly defined. As indicated by both training and testing set learning curves, the bigger the learning rate, the less epochs is needed to converge to the local minimum. However, learning rates > 0.02 were highly unstable and would not converge, probably due to the fact that the large step would make the optimization process leave the initial basin of attraction of the cost function and get lost in unknown territory. Conversely, very small learning rates did nothing to improve the final score and took a larger number of training epochs to converge. The minimum $RMSE$ obtained was equal to 1953.49.

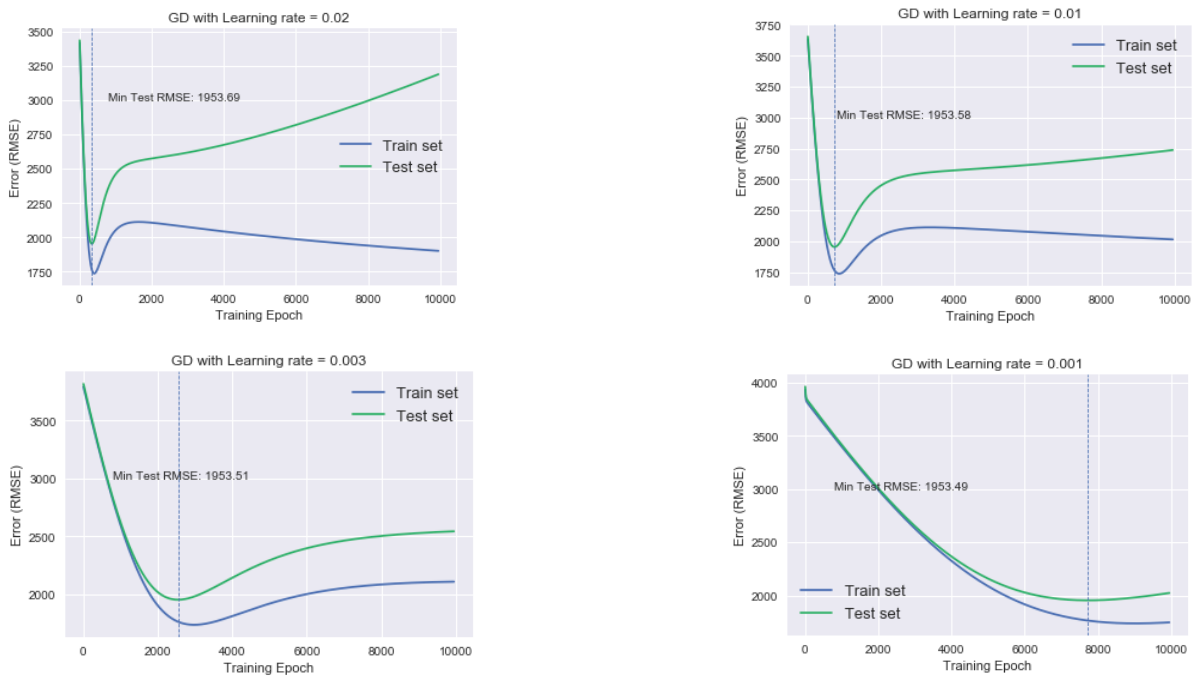

Figura 6: Root Mean Squared Error (RMSE) vs. Training Epoch with gradient descent optimization of a linear model

Closed-form equations

The closed-form linear regression is a non-iterative form of linear regression in which the error is minimized by matrix operations. The optimal solution is

$$w^* = X^\dagger y \tag{1}$$

where $X^\dagger$ is the pseudoinverse of the data matrix, y is the target array, and w^* is the optimal parameter array. The minimum test split RMSE obtained was equal to 1287.52.

SGDRegressor

Lastly, the SGDRegressor provided by *sklearn* is a linear regressor with stochastic gradient descent optimization. The minimum test split RMSE obtained was equal to 2635.40.

Final Analysis and Conclusion

Judging by test set prediction error, the best performance was obtained by the closed-form equation model, which also had a quick convergence. Thus, in this particular problem, it's the best model. However, its solution involves computing inverse matrices, which can be computationally infeasible when the dataset is large enough. The GD model outperformed *SGDRegressor* probably due to the fact that it had early stopping, which prevented the model from overtraining. However, GD's convergence is much slower because it uses all datapoints to determine the next optimization step. A fairer comparison approach would be to use SGD with early stopping.