Regressions to Delhi house price prediction

José Roberto Londoño Sánchez - 2224263
Jhonatan David Giraldo - 2227050

ABSTRACT- The following report describes how to implement regressors that can predict the price of a house as a function of different house characteristics. From the visualization and analysis of the relationship between variables to the preprocessing and fitting of the overall dataset and the training and validation of the results. The models generated include a linear regression with ElasticNet regularization, a RandomForest, a DesicionTree and an Adaboost regressor. Finally, satisfactory results were obtained with training and validation scores of 0.75 and 0.71, 0.93 and 0.82, 0.99 and 0.67, and 0.80 and 0.72 respectively.

1. INTRODUCTION AND DESCRIPTION OF THE PHENOMENON

There are a number of characteristics that can be used to calculate the value of a house, the size of the land, the number of bedrooms, the number of bathrooms, among others. Thanks to the census we have a great variety of data from which we can extract the relationship between all these variables and the price that a house has as a consequence.

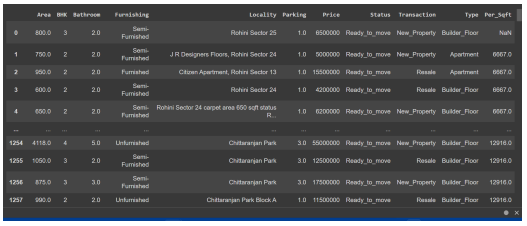
The applicability of this is reflected in the same cause of why companies in the real estate sector store this type of data, expertise and consultation. It is easy to put a price on a house if you want to sell it, especially if the objective is to get the most profit possible, however, this produces a scam in most cases because of the overvaluation that these goods suffer by their owners, and in a few other cases suffer devaluation due to ignorance. Due to the fact that a house usually has a high price for the standards of any citizen, buying one is usually a premeditated decision that critically looks for an opinion supported by truthful data and backed by the market in general. This is how companies or independent workers emerge that with the experience of several years in the work of buying and selling can objectively value the price of a house based on the characteristics mentioned above. The reason why it is important to make this type of predictions in an automatic and accessible way lies in the avoidance of fraud, the fair valuation of the property and in the long run in the regularization of inflation. The data and technology are

already in place, the next step is to implement these alternatives.

2. MACHINE LEARNING PROBLEM AND OBTAINING THE DATASET

While one could estimate the price of a house in ranges so as to focus on a classification problem, most datasets, and hence the mental scheme of data organization, place the price variable as a continuous value. This situation may have several reasons, but we believe that for the purposes of precise value and avoidance of ambiguity, it is chosen to produce point estimates. Next, due to the nature of this value, we choose to construct a regression implementation.

Originally, the alternative was to use the dataset of the example of the problem, but due to implementation problems because of the extremely low correlation between the variables, it was decided to look for a dataset that met a minimum correlation (this situation will be explained in the next section). For the moment it is important to understand that the variables to be used are the area of the land occupied by the house, the number of bedrooms, the number of bathrooms, if it is furnished, location, how many parking spaces it has, if it is a new or resale property, if it is ready to move in, the price per sqft and if it is a house or an apartment.



	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3	2.0	Semi-Furnished	Rohini Sector 25	1.0	6500000	Ready_to_move	New_Property	Builder_Floor	NaN
1	750.0	2	2.0	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1.0	5000000	Ready_to_move	New_Property	Apartment	6667.0
2	950.0	2	2.0	Furnished	Citizen Apartment, Rohini Sector 13	1.0	15500000	Ready_to_move	Resale	Apartment	6667.0
3	600.0	2	2.0	Semi-Furnished	Rohini Sector 24	1.0	4200000	Ready_to_move	Resale	Builder_Floor	6667.0
4	650.0	2	2.0	Semi-Furnished	Rohini Sector 24 carpet area 650 sqft status R...	1.0	6200000	Ready_to_move	New_Property	Builder_Floor	6667.0
...	...	...	...	...	...	...	...	...	...	...	...
1254	4118.0	4	5.0	Unfurnished	Chittaranjan Park	3.0	55000000	Ready_to_move	New_Property	Builder_Floor	12916.0
1255	1050.0	3	2.0	Semi-Furnished	Chittaranjan Park	3.0	12500000	Ready_to_move	Resale	Builder_Floor	12916.0
1256	875.0	3	3.0	Semi-Furnished	Chittaranjan Park	3.0	17500000	Ready_to_move	New_Property	Builder_Floor	12916.0
1257	990.0	2	2.0	Unfurnished	Chittaranjan Park Block A	1.0	11500000	Ready_to_move	Resale	Builder_Floor	12916.0

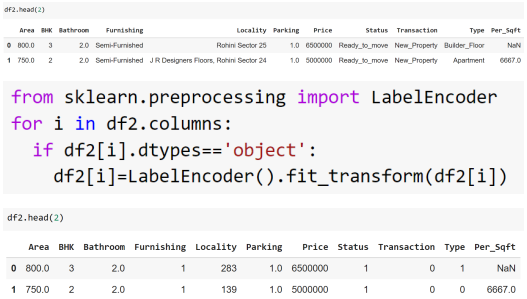
Figure 1. Some examples of the dataset.

As can be seen, there are 5 categorical variables (3 binary) and 6 numerical variables.

3. PREPROCESSED DATASET

Label encoder

The first thing that was done for the data preprocessing was the label encoding for the categorical variables, which is a method that convert a group of categorical features into a number between 0 and the total number of features, so that all the variables could be analyzed numerically.



```
df2.head(2)
```

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3	2.0	Semi-Furnished	Rohini Sector 25	1.0	6500000	Ready_to_move	New_Property	Builder_Floor	NaN
1	750.0	2	2.0	Semi-Furnished	J R Designers Floors, Rohini Sector 24	1.0	5000000	Ready_to_move	New_Property	Apartment	6667.0

```
from sklearn.preprocessing import LabelEncoder
for i in df2.columns:
  if df2[i].dtypes=='object':
    df2[i]=LabelEncoder().fit_transform(df2[i])
```

```
df2.head(2)
```

	Area	BHK	Bathroom	Furnishing	Locality	Parking	Price	Status	Transaction	Type	Per_Sqft
0	800.0	3	2.0	1	283	1.0	6500000	1	0	1	NaN
1	750.0	2	2.0	1	139	1.0	5000000	1	0	0	6667.0

Figure 2. Label encoder

Correlation matrix

The next step was to evaluate the behavior of all the variables with the output by elaborating the correlation matrix of the numerical values in order to analyze the connection between the price and the other features, as a disqualifying

parameter we decided to delete the characteristics that had an extremely low or negative correlation.

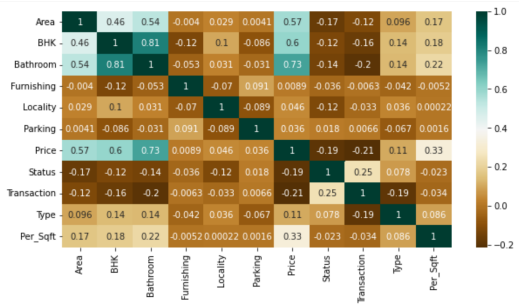


Figure 3. Correlation matrix

As we can see in the next image, the features selected to work with the model were 'Price', which would be the target of the model according to which the regression will be made; 'Per_Sqft', is the price per square foot of the lot; 'Type', which is the type of property; 'Locality', which is the locality where the house is located; 'Bathroom', is the amount of bathrooms that the house has; 'BHK', is the number of bedrooms along with 1 Hall and 1 kitchen; finally 'Area', which is Area of the Property in square feet.

```
Dataset with all layers
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1005 entries, 1 to 1258
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Area         1005 non-null   float64
 1   BHK          1005 non-null   int64
 2   Bathroom     1005 non-null   float64
 3   Furnishing   1005 non-null   int64
 4   Locality     1005 non-null   int64
 5   Parking      1005 non-null   float64
 6   Price        1005 non-null   int64
 7   Status       1005 non-null   int64
 8   Transaction  1005 non-null   int64
 9   Type         1005 non-null   int64
 10  Per_Sqft     1005 non-null   float64
dtypes: float64(4), int64(7)
memory usage: 94.2 KB
None
Dataset without bad correlation values
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1005 entries, 1 to 1258
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Price     1005 non-null   int64
 1   Per_Sqft  1005 non-null   float64
 2   Type      1005 non-null   int64
 3   Locality  1005 non-null   int64
 4   Bathroom  1005 non-null   float64
 5   BHK       1005 non-null   int64
 6   Area      1005 non-null   float64
dtypes: float64(3), int64(4)
memory usage: 95.1 KB
None
```

Figure 4. Non-delete variables

Delete null values

To eliminate the null values, the 'dropna()' method was used, with which there was a reduction of 254 missing data.

```
dataset shape before delete null values: (1259, 11)
Area            0
BHK             0
Bathroom        2
Furnishing      0
Locality        0
Parking        33
Price           0
Status          0
Transaction     0
Type            0
Per_Sqft      241
dtype: int64
Area          0
BHK           0
Bathroom      0
Furnishing    0
Locality      0
Parking       0
Price         0
Status        0
Transaction   0
Type          0
Per_Sqft      0
dtype: int64
dataset shape after delete null values: (1005, 11)
```

Figure 5. Null values

Delete outliers

To eliminate outliers, the 'IQR' was calculated, and only those above the 75% quartile 3 were eliminated, thus reducing the number of data from 1005 to 846.

```
IQR
Price       21870000.0
Per_Sqft       11636.0
Type               1.0
Locality         165.0
Bathroom           1.0
BHK                1.0
Area             930.0
dtype: float64
Data shape before delete outliers(1005, 7)
Data shape after delete outliers(846, 7)
```

Figure 6. Outliers.

Train test split

Due to the small amount of data in the dataset, it was decided to implement a separation of 80% of the data for training and 20% for validation. The split was done using sklearn's train-test split algorithm with a random state of 42 to control the shuffling applied to the data before applying the split. Finally, 676 data were used for training and 170 for validation.

```
y = df4[['Price']]
x = df4.drop([ 'Price'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print('x_train shape: ' + str(x_train.shape) + ' y_train shape' + str(y_train.shape)
  + ' x_test shape' + str(x_test.shape) + ' y_test shape' + str(y_test.shape))
```

```
x_train shape: (676, 6) y_train shape(676, 1) x_test shape(170, 6) y_test shape(170, 1)
```

Figure 7. Train-test split

4. TRAINING AND EVALUATION OF MODELS

Three different methods were used to evaluate the model, the first is the r2 score, which measures the amount of variability in the dependent variable that the model can explain, generating a value between 0 and 1, the closer to 1 the better the fit between the predicted and actual values.

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Another metric used is the root mean square error, which is a metric that scales with actual values by averaging the difference between predictions and actual data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

The last metric used was mean absolute error, which finds the average absolute distance between the predictions and the real data; being absolute, it does not take into account whether the prediction error is positive or negative and is more stable in the presence of outliers than MSE.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

Linear regression

The first alternative solution was to implement a common linear regressor from the sklearn library, however,

problems of overfitting were occurring, so following the behavioral process to eliminate it, tests were performed using the 3 types of regularization L1, L2 and Elastic Net, thus, of the 3 the results were very similar and we opted to stay with what was obtained by Elastic Net, presenting a score value of 0. 75 and 0.71 for training and validation respectively, likewise these magnitudes can be seen in the MSE value, where the average variation was of 6 million approximately, being a small value for the one that is handled in the house price output variable. Likewise, the MAE was calculated where a value of approximately 4 million was obtained in both, also a minuscule value considering the output.

```
from sklearn import linear_model
lr = linear_model.ElasticNet(alpha=.00001).fit(x_train,y_train)
val = lr.predict(x_train)
print('Train')
print("score: " + str(r2_score(y_train,val)))
print("MSE: "+ str(mse(y_train,val,squared=False)))
print("MAE: " + str(mean_absolute_error(y_train,val)))
#Validation
print('Validation')
val2 = lr.predict(x_test)
print("score: " + str(r2_score(y_test,val2)))
print("MSE: "+ str(mse(y_test,val2,squared=False)))
print("MAE: " + str(mean_absolute_error(y_test,val2)))
```

```
Train
score: 0.7580690717252164
MSE: 6192330.226505143
MAE: 4512599.6036294
Validation
score: 0.7128844985223033
MSE: 6257234.772981188
MAE: 4435707.204387748
```

Figure 8. Linear model

Random Forest Regressor

Random forest is a method that uses a group of decision trees that each one generate a prediction that is finally averaged to produce the final prediction of the model.

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor()
rf_model.fit(x_train, y_train)
val = rf_model.predict(x_train)
print('Train')
print("score: " + str(r2_score(y_train,val)))
print("MSE: "+ str(mse(y_train,val,squared=False)))
print("MAE: " + str(mean_absolute_error(y_train,val)))
#Validacion
print('Validation')
val2 = rf_model.predict(x_test)
print("score: " + str(r2_score(y_test,val2)))
print("MSE: "+ str(mse(y_test,val2,squared=False)))
print("MAE: " + str(mean_absolute_error(y_test,val2)))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launch
  This is separate from the ipykernel package so we can
Train
score: 0.9737639478164839
MSE: 2039189.8256902904
MAE: 1294183.8675729302
Validation
score: 0.8248288857296877
MSE: 4887484.307133149
MAE: 3187172.205604981
```

Figure 9. Random Forest.

Decision Tree

A decision tree was then performed which yielded a score value of 0.99 and 0.67 respectively for training and validation, the worst result of the 4 models.

```
from sklearn.tree import DecisionTreeRegressor
h_dat_model = DecisionTreeRegressor()
h_dat_model.fit(x_train, y_train)
val = h_dat_model.predict(x_train)
print('Train')
print("score: " + str(r2_score(y_train,val)))
print("MSE: "+ str(mse(y_train,val,squared=False)))
print("MAE: " + str(mean_absolute_error(y_train,val)))
#Validacion
print('Validation')
val2 = h_dat_model.predict(x_test)
print("score: " + str(r2_score(y_test,val2)))
print("MSE: "+ str(mse(y_test,val2,squared=False)))
print("MAE: " + str(mean_absolute_error(y_test,val2)))
```

```
Train
score: 0.999442622649373
MSE: 297223.4500830634
MAE: 36317.56756756757
Validation
score: 0.6754515159406882
MSE: 6652638.4592086
MAE: 4162847.7690288713
```

Figure 10. Decision Tree

Adaboost

Finally, we implemented an adaboost type regressor, where 50 estimators were

used, so that 50 times the weighting of the importance of separate segments within the data was done, which using an alpha of 1 (best value found for the hyperparameter) achieved scores of 0.80 and 0.71 with MSE DE approximately 5 and 6 million, again, a minimum value. Similarly for the MAE.

```
from sklearn.ensemble import AdaBoostRegressor
regr = AdaBoostRegressor( n_estimators=100)
regr.fit(x_train, y_train)
val = regr.predict(x_train)
print('Train')
print("score: " + str(r2_score(y_train,val)))
print("MSE: "+ str(mse(y_train,val,squared=False)))
print("MAE: " + str(mean_absolute_error(y_train,val)))
#Validacion
print('Validation')
val2 = regr.predict(x_test)
print("score: " + str(r2_score(y_test,val2)))
print("MSE: "+ str(mse(y_test,val2,squared=False)))
print("MAE: " + str(mean_absolute_error(y_test,val2)))
```

```
Train
score: 0.8045532674908084
MSE: 5565736.21982495
MAE: 4586513.248613496
Validation
score: 0.7181905218528276
MSE: 6199146.895478325
MAE: 4850394.813762219
```

Figure 11. AdaboostRegressor.

Finalmente, se realizó un ejercicio de análisis, la comparación de estos modelos en el contexto de usar outlier y no usarlos:

```
LienarRegression
score: 0.7582184786972409
score: 0.7046624016221024
RandomForest
score: 0.9763370651237138
score: 0.8103075512831678
DecisionTree
score: 0.9994490188295592
score: 0.7039047264435727
adaboost
score: 0.8079031569506414
score: 0.7580106752910702
```

Figure 12. Scores without outliers.

```
LienarRegression
score: 0.6182166168382918
score: 0.5645308493285732
RandomForest
score: 0.9807907025972492
score: 0.8636841652728177
DecisionTree
score: 0.9997292672008897
score: 0.7397892183224251
adaboost
score: 0.8673089492198218
score: 0.7390782836239875
```

Figure 13. Scores with outliers.

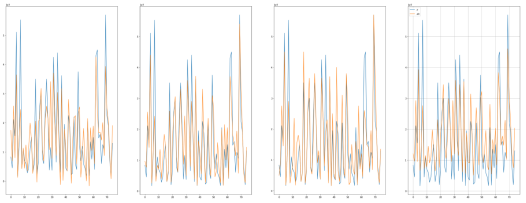


Figure 14. Comparison between the prediction of 4 models without outliers.

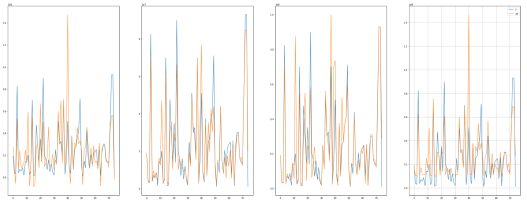


Figure 15. Comparison between the prediction of 4 models with outliers.

As can be seen for conventional linear regression, better results are obtained by removing the outliers; however, a small improvement is noted when outliers are left out when using models of the nature of decision trees, which contrasts with the original problem of using outliers in these models, the difficulty of separating the data. This indicates that these variations of the variables were beneficial to the dataset, demonstrating that the assumption does not apply in all cases.

5. CONCLUSIONS

- The correlation between variables in a model tends to be determinant, so that no matter how many attempts are made to change the hyper-parameters of a model, change the regression model or try an endless number of methods of categorical data processing, numerical, dimensionality reduction or normalization, the main point of a problem of this nature may lie in the poor construction of the dataset or in the behavior between variables that does not have a sufficient connection to produce a linear relationship or a predictive relationship in general.

- Although a categorical variable can have a fairly large number of variations, the effect it can have on the output is not a magnitude to be measured by that appearance, but directly when this characteristic has become a manageable value (coding) and the model is trained.

- Use One hot encoding on the features affect the results of validation for linear regression because, if one of those variables wasn't used in the train data, when we try to predict the test data it becomes a problem for the model, that's why we use label encoder instead of one hot.

6. BIBLIOGRAPHY

[1] Skitlearn developers. "sklearn.linear_model.ElasticNet". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet

[2] Skitlearn developers. "sklearn.ensemble.AdaBoostRegressor". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor

[3] Skitlearn developers. "sklearn.linear_model.LinearRegression". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[4] Skitlearn developers. "sklearn.metrics.mean_squared_error". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[5] Skitlearn developers. "sklearn.tree.DecisionTreeRegressor". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor

[6] Skitlearn developers. "sklearn.ensemble.RandomForestRegressor". 2020. skit-learn.org. online: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html