Regression models to house price prediction

April 16, 2018

Machine Learning 1 - Professor Dr. Yuxiao Huang
Christian Braz - Toufik Bouras
Data Science, Columbian College of Arts & Sciences
George Washington University

1 Introduction

In this article we are going to discuss our work on predicting House Prices. This is a classic regression problem in which we aim to predict the selling price of a house given its attributes. The motivation in doing such analysis is to provide real estate firms accurate information about expected values of properties. After creating the seven models, we conduct an evaluation of each model by comparing its accuracy. This work is organized in the following way:

1. Methodology
2. Understanding and cleaning the data
3. Regression Models
4. Linear Regression
    1. Ordinary Least Squares
    2. Ridge
    3. Lasso
    4. Elastic Net
5. Suport Vector Machine
6. Neural Network
7. Random Forests

8. Discussion

2 Methodology

As we are dealing with a regression problem, an appropriate metric is the Root Mean Squared Error or RMSE. The methodology we followed is:

- Each method trained using 5-fold cross-validation.
- Final RMSE is calculated based on the average results of all training steps.

Using cross-validation is a common way to assess the predictive performance of the models and to judge how they perform outside the sample (and assess their generalization strength).

3 Understanding and cleaning the data

As mentioned by several authors, much of the work in making a good model is contingent on a good feature preparation process. This phase encompasses tasks such as: cleaning, feature mapping, feature selection, and feature scaling. In this section we describe the steps we did to get a better understanding of the data and to make it more appropriate for modeling analysis.

3.1 The data set

The house prices data set has 81 features and the objective is to predict SalePrice.

3.2 Exploring the dataset

Understanding the data is an important part of our project. The more we understand the data the more we will have the ability to accuratley select and manupilate the features to get the best possible results. In this section we explore possible correlations between the dependent and the independent variables (Figure 1). Linear regression models are sensitive to non-linearity, outliers and colinearity, so we are going also to check these potential problems.

3.2.1 Correlation Matrix

We can notice that some variables are strongly correlated with SalePrice. Specifically, these six features : OverallQual, GrlivArea, TotalBsmtSF, 1stFLrSF, GarageCars, and GrageArea. Moreover, some variables are strongly correlated with each other which means that we might have a multi-collinearity. Subsequently, we need to take them into consideration when selecting and preparing the features to use in our modelling. For example there is a strong correlation between Yearbuilt and GarageYrBlt which means that most Garages are built in the same time with the construction of the houses.Therefeore,we can consider that Yearbuilt and GarageYrBlt as the same variable. The correlation matrix shows only the value of the correlation but it doesn't reveal the nature of the correaltion. On the other hand scatter or some other charts can show the nature of the correlation whether it is linear or has another shape.

3.2.2 Scatter plot of the most correlated features with SalePrice

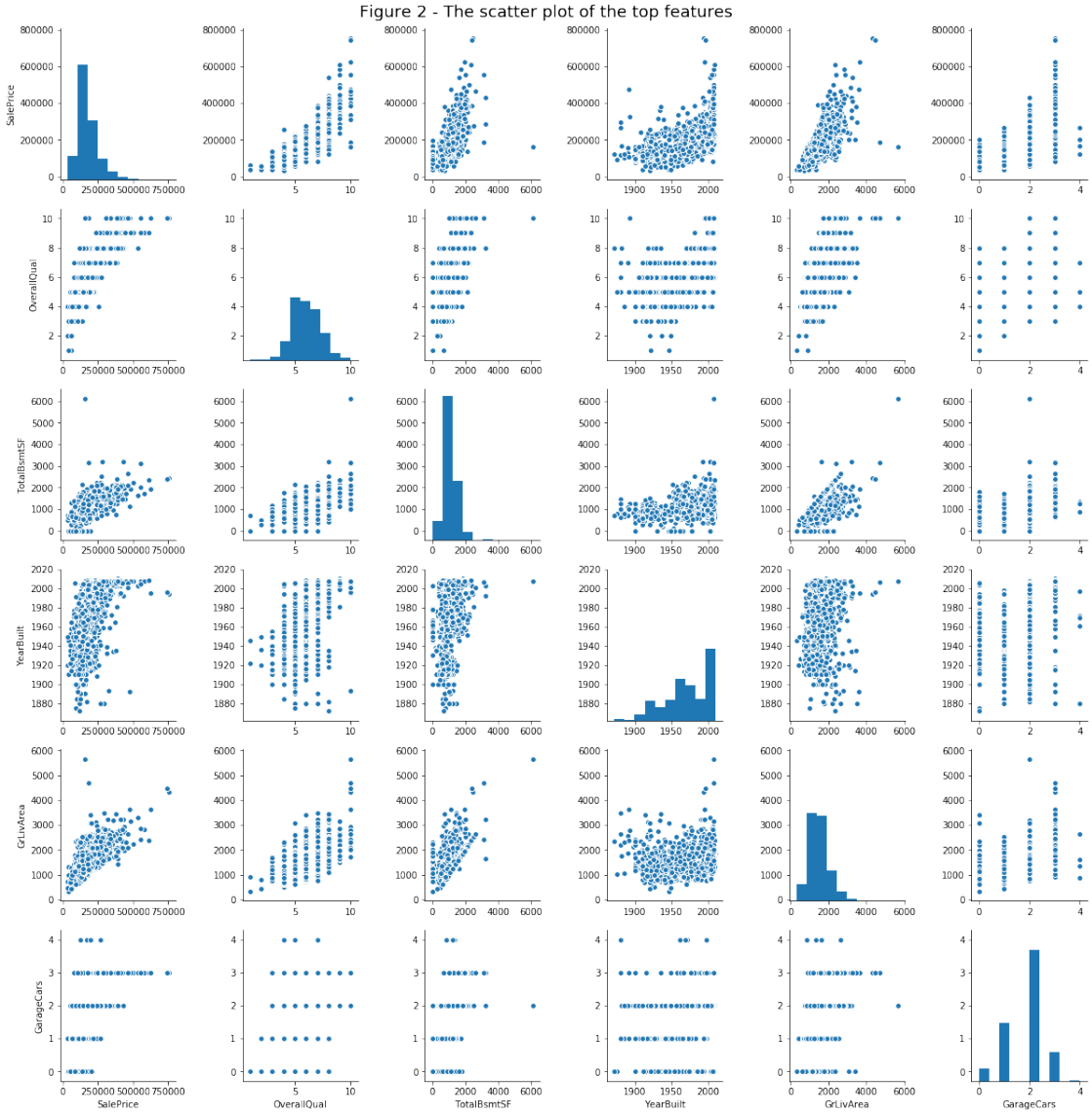
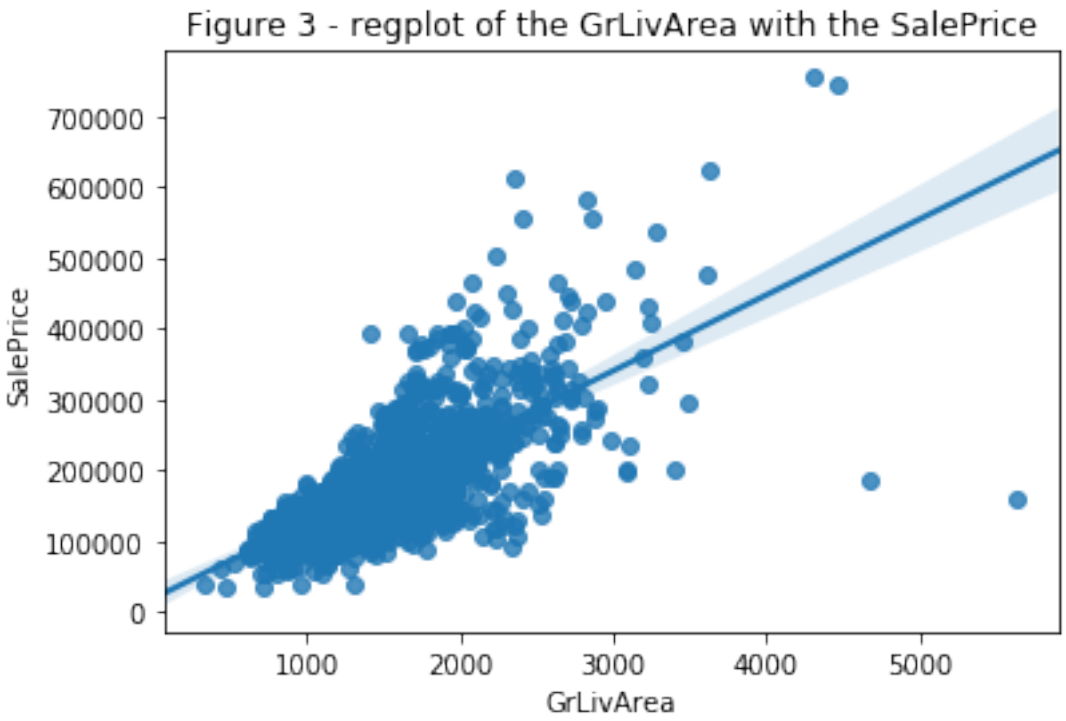
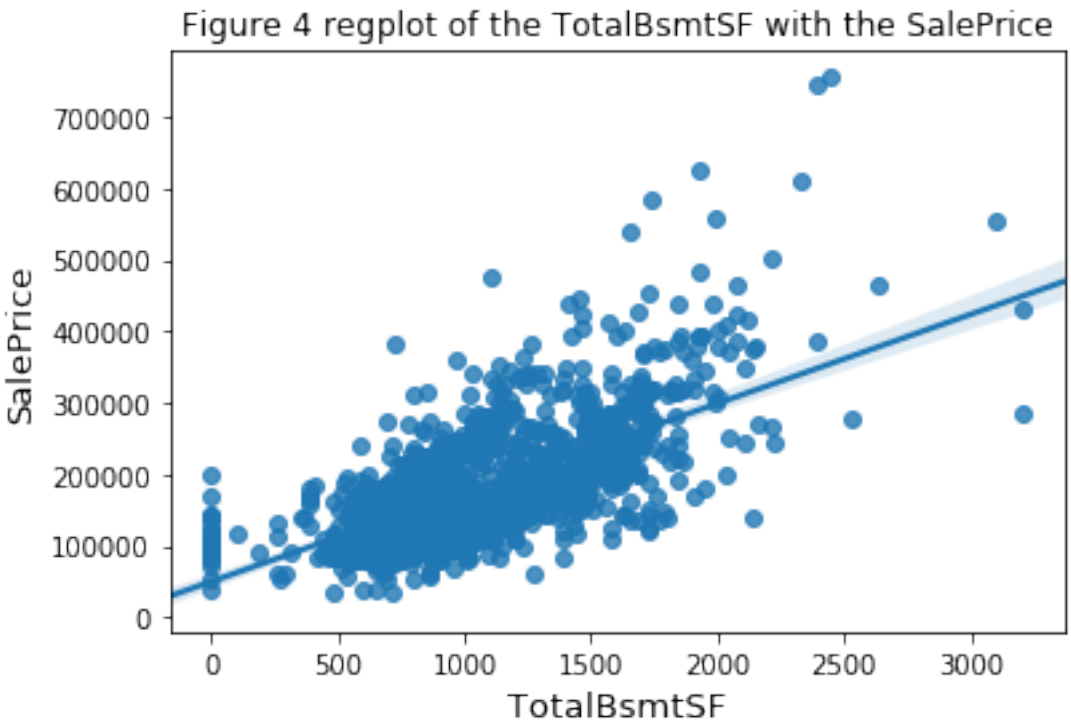


Figure 2 shows that there are two plots wich exhibit some kind of outliers: TotalBamtSF and GrLivArea. So we need to take a close look into them and decide whether we have outliers or not.



In Figure 3 it is Obvious that there are two points which can be considered as outliers. At thses points the ground living area is large but the sale price is low which is not logical. So we are going to remove them.

Next we are going to check the regplot of TotalBsmtSF and SalePrice in Figure 4.



Generally speaking there aren't obvious outliers, which is great because we don't need to delete more rows.

3.3 Missing values

Maybe the most common problem in real datasets is the missing values. Sometimes, instead of a blank space, a wildcard is used like '?' or 'NA'. In this project, in addition to the missing values we also have some 'NAs' used to denote the absence of features. To solve this issue, whenever appropriate we replace 'NA' by the value 'No'. Here is an example of an NA used instead of the absence to an alley access:

```
    Alley: Type of alley access to property

       Grvl Gravel
       Pave Paved
       NA   No alley access

Shape of training set:  (1458, 82)
Missing values before remove NA:
Index(['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
       'MiscFeature'],
```

```
      dtype='object')
```

As explained in the data set dictionary, for a set of features NA means "No" or "0". For exmple, for a house which does not have a Fence we would find "NA" as its fence value. So, for these categorical variables we replace NA by No whenever appropriate.

```
Missing values after insert No, i.e., real missing values:
Index(['LotFrontage', 'MasVnrType', 'MasVnrArea', 'Electrical', 'GarageYrBlt'], dtype='object')
```

For the numeric features, we imputed each missing value with a zero. We did this because mostly the missing value means that the house does not have that feature.

For some categorical features we used an interpolation technique, inserting the mode of each column. For these features the dataset description didn't mention anything about the missig values, so they are indeed missing so we needed to decide how to fill them. The most appropriate way to do that in our case is to replace NAs with the mode.

```
After we imputed the missing values, the status of the data set is:
Index([], dtype='object')
```

We can see that there are no missing values.

3.4 Encoding

Encoding is important to deal properly with different types of values: nominal, ordinal and numeric. In the house prices data set we can find all of them. Here is an example of ordinal feature:

```
ExterQual: Evaluates the quality of the material on the exterior

       Ex   Excellent
       Gd   Good
       TA   Average/Typical
       Fa   Fair
       Po   Poor
```

Thus, we are going to perform the following steps:

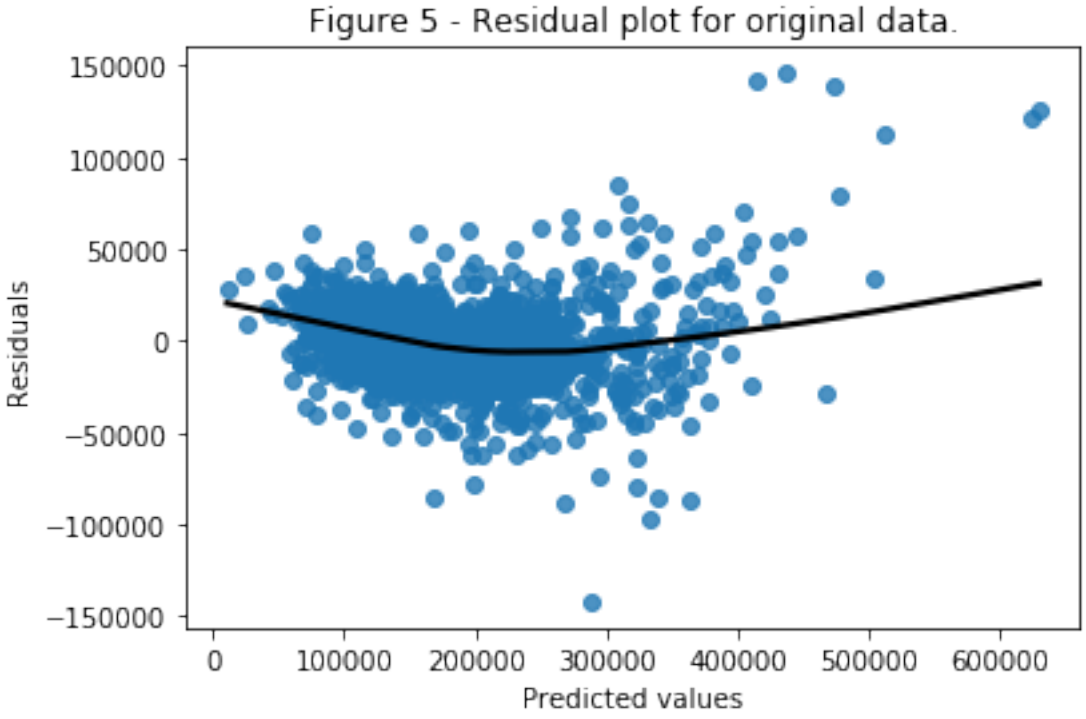
- Mapping the ordinal fields which are strings to the corresponding meaningful codes.
- Transforming numeric categorical features to string.
- Applying one-hot encoding.

```
New  shape after one-hot encoding: (1458, 229)
```

We now have a new data set with no missing values and a well defined set of features. As expected, after the one-hot encoding process we jumped from 81 to 229 features.

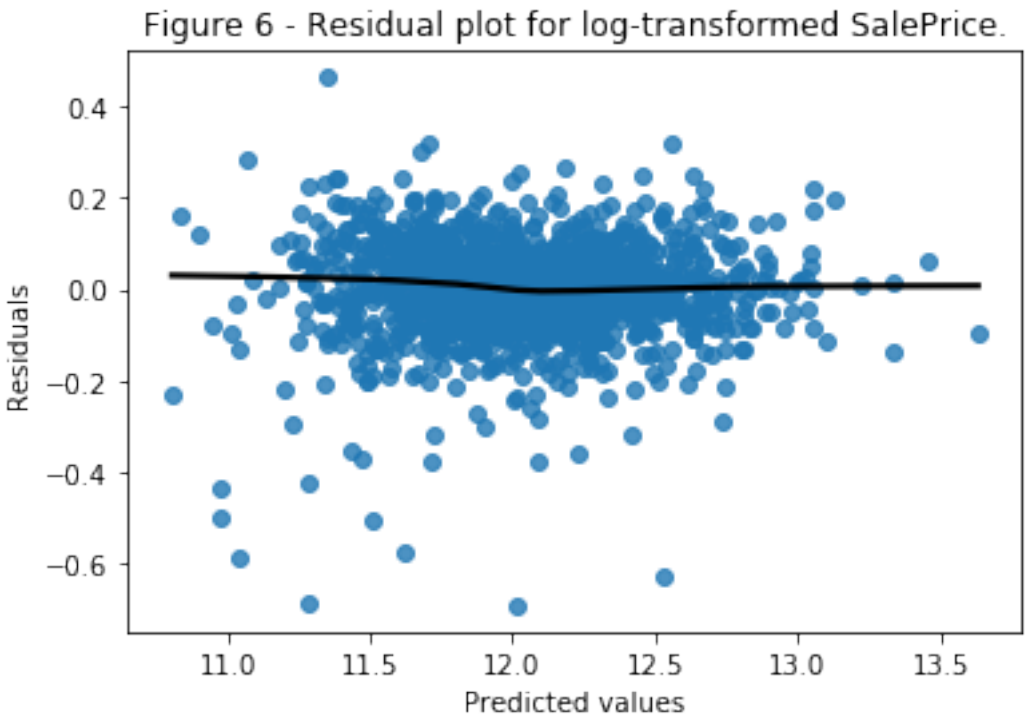
3.4.1 Transforming non-linear data into linear data

We are going to start with verifying whether the data is linear or not. To do this in a multiple linear regression model we plot the residuals against the predicted values. The results are shown in Figure 5.




```
Mean square error:  492970222.921
```

We can note some evidence of non-linearity and also a fairly high error. Hence, as recommended by the specilailists, we applied log transformation on the dependent variable SalesPrice. The new result is shown in Figure 6.




```
Mean square error:  0.0101646358751
```

As a result of applying a log transformation on the values of SalePrice, we get a much more linear residual plot and a impressive decline in the mean square error.

3.4.2 Engineering new features

By Analyzing the data we can observe that there is no feature with the total size of the house. This information can be obtained from other variables, specificaly from: TotalBsmtSF, 1stFlrSF, 2ndFlrSF and GarageArea. Hence, it seems to be a good idea to create new feature (TotalSF) with this information.

3.5 Feature selection

Feature selection is the task of trying to discover the smallest set of features highly correlated with the dependent variable. It is important for the interpretability of the model but also to get a better fit, and consequently a better performance. We employed an automatic feature selection technique using a tree-based learning algorithm, and then used the tree structure produced to select the best features.

```
Ten most important features selected with tree-based selection:

+----+----------+--------------+------------+----------------+-----------+
```

```
|    |   TotalSF |   OverallQual |   YearBuilt |   YearRemodAdd |   LotArea |
|----+-----------+---------------+-------------+----------------+-----------|
|  0 |  0.547792 |      0.243313 |   0.0193983 |      0.0187881 | 0.0122713 |
+----+-----------+---------------+-------------+----------------+-----------+
+----+-------------+-------------+--------------+------------+------------+
|    |   GrLivArea |   BsmtUnfSF |   BsmtFinSF1 |   1stFlrSF |   BsmtQual |
|----+-------------+-------------+--------------+------------+------------|
|  0 |   0.0105283 |  0.00862509 |   0.00770424 | 0.00651909 | 0.00542328 |
+----+-------------+-------------+--------------+------------+------------+

 New shape for train after tree-based feature selection: (1458, 41)

 Features selected :
Index(['TotalSF', 'OverallQual', 'YearBuilt', 'YearRemodAdd', 'LotArea',
       'OverallCond', 'GrLivArea', 'BsmtUnfSF', 'BsmtFinSF1', '1stFlrSF',
       'BsmtQual', 'CentralAir_Y', 'CentralAir_N', '2ndFlrSF', 'FireplaceQu',
       'TotalBsmtSF', 'GarageYrBlt', 'GarageArea', 'OpenPorchSF', 'ExterQual',
       'LotFrontage', 'GarageFinish', 'Id', 'index', 'WoodDeckSF',
       'KitchenAbvGr', 'TotRmsAbvGrd', 'MasVnrArea', 'ExterCond',
       'SaleCondition_Abnorml', 'SaleCondition_Family', 'HeatingQC',
       'GarageType_Detchd', 'Fireplaces', 'Neighborhood_OldTown',
       'EnclosedPorch', 'MSSubClass_30', 'Neighborhood_Crawfor',
       'Neighborhood_IDOTRR', 'FullBath', 'BedroomAbvGr'],
      dtype='object')

 End of the process of selecting best features.
```

The final process resulted in 40 features. It is important to note that the field we created before **TotalSF** was chosen as one of the most relevant feature.

3.6 Feature scaling

Our last step in the pre-processing phase would be standardizing the data. This will be usefull for all the models. As we are using cross-validation, the scaling has to be done independently for the training and the testing sets.

4 Regression Models

In this section we are going to present the evaluation of different scikit-learn modeling algorithms. We aim to mesure the performence of each model and compare it with the other models.

4.1 Linear Regression

In this section we are going to experiment with a set of regression methods in which the target value is expected to have a linear relationship with the input variables.

4.1.1 Ordinary Least Squares

Least Square Error is a well known mathematical measure of the performance of a linear regression model. LSE works by changing the coefficients of the model in a way that minimize the sum of squares between the true values and the predicted values. It solves a problem of the form: $\min_w ||Xw - y||_2{}^2$ and can be solved analyticaly by the equation

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Where X is a matrix of the independents features, y is the actual response and $\hat{\beta}$ the estimated weights w.

```
Linear Regression
```

```
Average RMSE: 0.12352798289953427
```

4.2 Ridge

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The new equation of a penalized residual sum of square is $\min_w ||Xw - y||_2{}^2 + \alpha ||w||_2{}^2$.

Here, $\alpha \geq 0$ is called the regularization parameter (L2) and controls the amount of shrinkage. Higher the values of alpha, bigger is the penalty and therefore the magnitude of coefficients are reduced. It is worth to note that we used RidgeCV which does an implicit leave-one-out cross-validation to choose the best alpha.

```
Ridge
```

```
Average RMSE: 0.12330292068107444
```

4.3 Lasso

The Mathematics behind lasso regression is quiet similar to that of ridge. the only difference is instead of adding squares of theta, we add the absolute value of w: $\min_w \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha ||w||_1$. Lasso is a linear model that estimates sparse coefficients, i.e., it reduces the number of variables upon which the given solution is dependent. It does a kind of feature selection and thus lead to a less complex final model. For instance, when there are correlated features it will choose one and set the coefficient of the other to zero. The regularization parameter alpha (L1) controls the degree of sparsity of the coefficients estimated and we employed once more the version of the algorithm that automatically chooses the best value.

```
Lasso
```

```
Average RMSE: 0.12390289951654518
```

4.4 Elastic Net

Is a hybrid method that trains with L1 and L2 as regularizers. This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge. It is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while Elastic Net is likely to pick both. The objective function to minimize is in this case

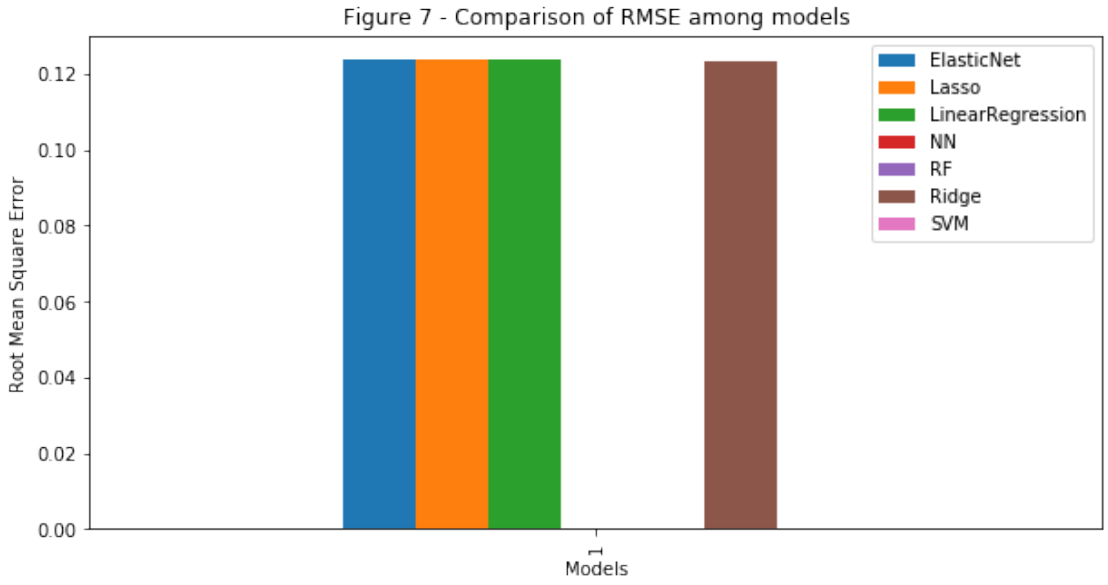
$$\min_{w} \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$$

```
Elastic Net
```

```
Average RMSE: 0.12379041525319807
```

Let's compare the results we have got so far. From Figure 7 we can observe that all methods have a fairly similar performance. OLS and Ridge have slight lower RMSE. In any case, it seems that regularization is not an issue for this particular problem. All the models were trained using their default parameter values.

```
Out[66]: Text(0.5,0,'Models')
```



4.5 Support Vector Machine

SVM is a large margin classifier. The rationale behind having decision boundaries with large margins is that they tend to have a lower generalization error. Whereas, models with small margins are more prone to overfitting.

We have used the linear kernel as it gave far the best result when comparing to rbf or sigmoid, set the regularization parameter C = 1 as according to the literature this is a reasonable choice and kept the other parameters in their default values.

```
SVM

 Average RMSE: 0.12539720824278738
```

4.6 Multilayer Perceptron

Multi-layer Perceptron (MLP) is a supervised learning algorithm. Given a set of features X = {x_1, x_2, ..., x_m} and a target y, it can learn a non-linear function for either classification or regression.

The class we are using in our work is the MLPRegressor which implements a multi-layer perceptron that trains using backpropagation with no activation function in the output layer (uses the identity function as activation function). Therefore, it uses the square error as the loss function, and the output is a set of continuous values.

A one hidden layer MLP has a function in the form:

$$f : R^D -> R^L$$

where D is the size of input vector x, and L is the size of the output vector f(x).

This is one of the models with more hyperparameters to set. We have tested some configurations and the best set of parameters was: learning rate of 0.001, a ReLu activation function for the hidden layers as this can speed up the learning process, a small regularization parameter alpha of 0.002 and a net of three hidden layers with 80, 50 and 20 neurons respectively.

```
Neural Network

 Average RMSE: 0.725288899787116
```

4.7 Random Forest

Random Forest is one of the most versatile and precise classifier. It does not need the data being scaled and can deal with any number of features. In order to test the random forest algorithm a little further than the others, we decided to train it using different train sets. First, with a data set containing all the features and no scaling. Second, in the reduced train set with scaled features.

As we can see in the results bellow, it did well in both cases. This test was also interesting to validate our set of selected features. Seems they are really good options once the performance using the the full set is not significatively better than using the restricted set.

After some experimentation, the configuration we chose was: the number of trees is 500 as any greater value did not enhance the accuracy, MSE as the function to measure the quality of

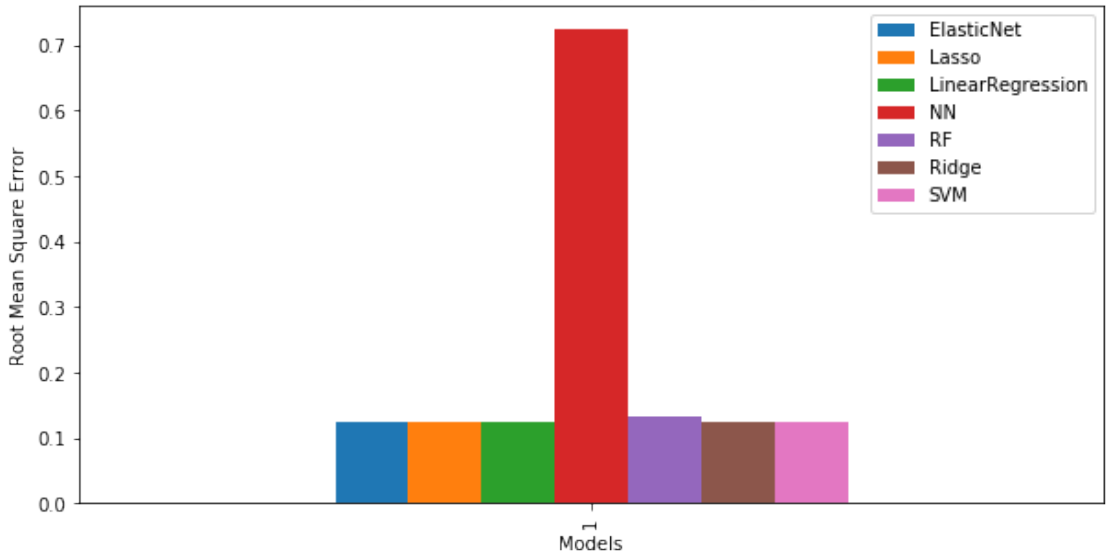
a split, max_depth with None which means the nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples, max_features = auto to consider all the features when looking for the best split and bootstrap = True for replacement as it gave better results during the tests. All the others parameters were maintained in their default values.

```
Random Forests

Full Features

Average RMSE using all features: 0.1341187954172455

Less features

Average RMSE less features: 0.13369220187027012
```

Done! We have tested seven different algorithms in the house prices prediction problem. Lets now compare the performance of each one (Figure 8).

```
Out[70]: Text(0.5,0,'Models')
```

Figure 8 - Comparison of RMSE among models



5 Discussion

Most models exhibited similar performance except for MLP. As this model has several parameters to set, a reasonable approach would be to do an extensive exploration for best values. As this can be a tedious and error prone task, one can make use of some automated search/evaluation process, like the GridSearch resource available in the scikit-learn package.

Data preprocessing have been proven to be a crucial part of our work, for instance adressing the non-linearity problem with log transformation improved the performance dramatically. Moreover, removing the outliers also yield better results. Encoding the features according to their type: nominal, ordinal, and numerical is also critical to our work.

One way of improving our results is creating an ordinal version of the location, because, as we know, location is quite important factor in most housing prices. We can also improve our model doing more feature engineering, as we did when we create the TotalSF, or doing more useful transformations such as Box-Cox or log transformation to other variables to reduce their variability.