House Sales Price Predictor

All the headings indicates to the section of code written in the Jupyter Notebook.

Refer to the code and process followed in the explore.ipynb file with the visualizations.

Separate modules are also implemented in respective directories for further usage though it is recommended to use Jupyter Notebook for understanding the workflow.

Table of Contents

Definition

Project Overview

This is a classic regression problem in which we aim to predict the selling price of a house given its attributes. After creating the four models, we conduct an evaluation of each model by comparing its accuracy.

Metrics

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).

Analysis

Data Exploration

Correlation Matrix

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

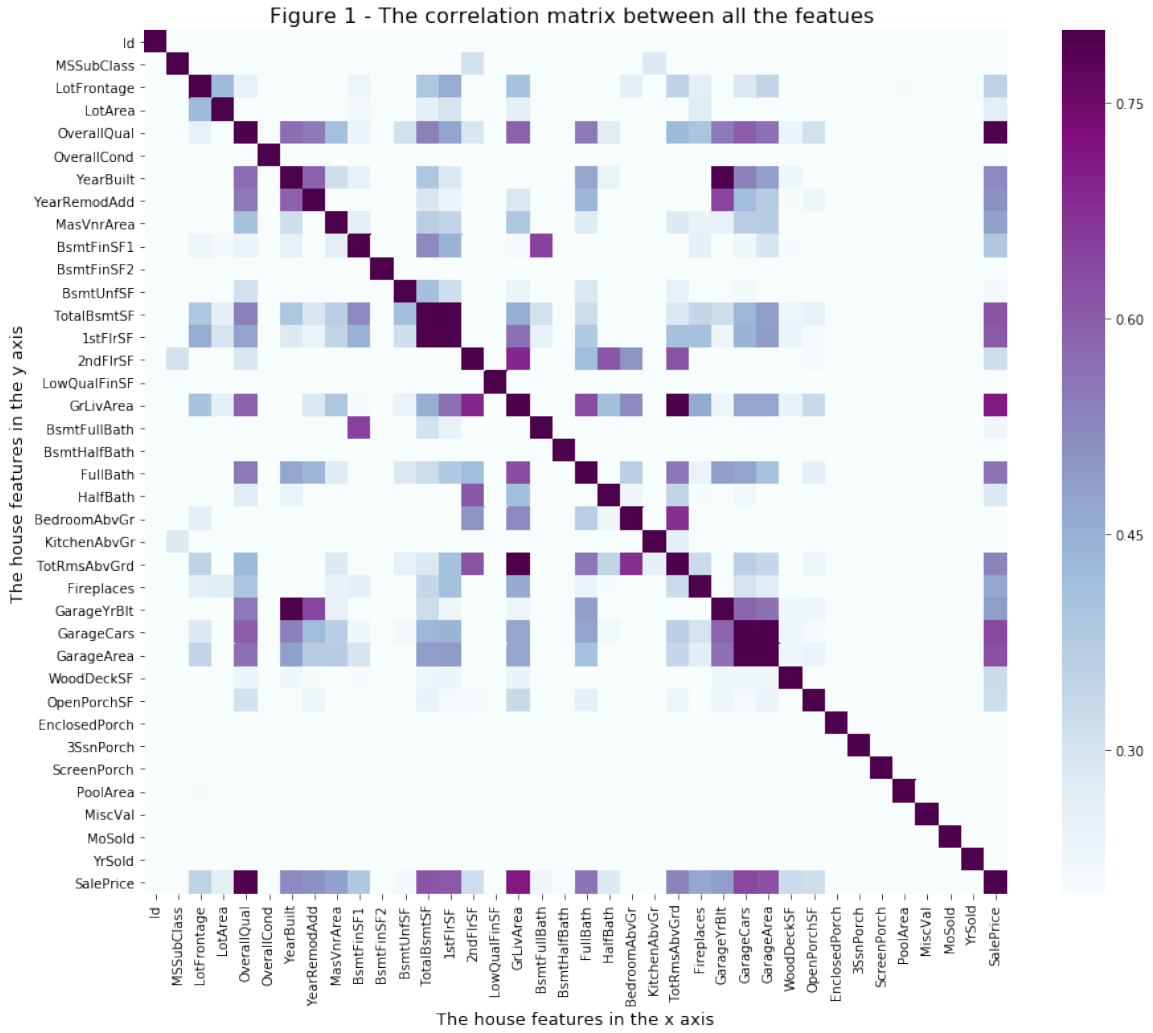


Figure1 – Correlational Matrix

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 multicollinearity. 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 correlation. On the other hand scatter or some other charts can show the nature of the correlation whether it is linear or has another shape.

Scatter plot of the most correlated features with SalePrice

Figure 2 - The scatter plot of the top features

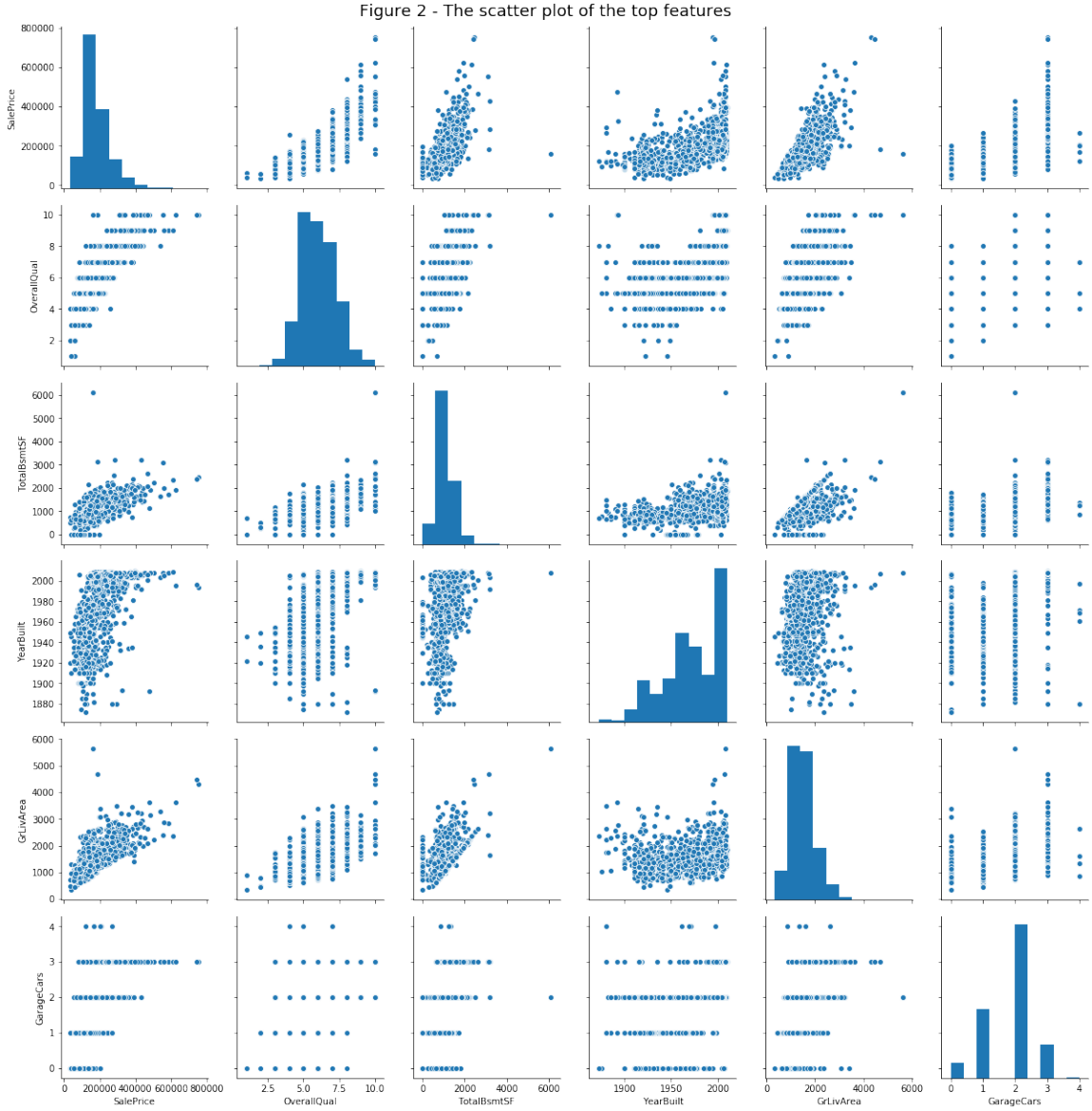


Figure2 -Scatter Plot of Features

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.

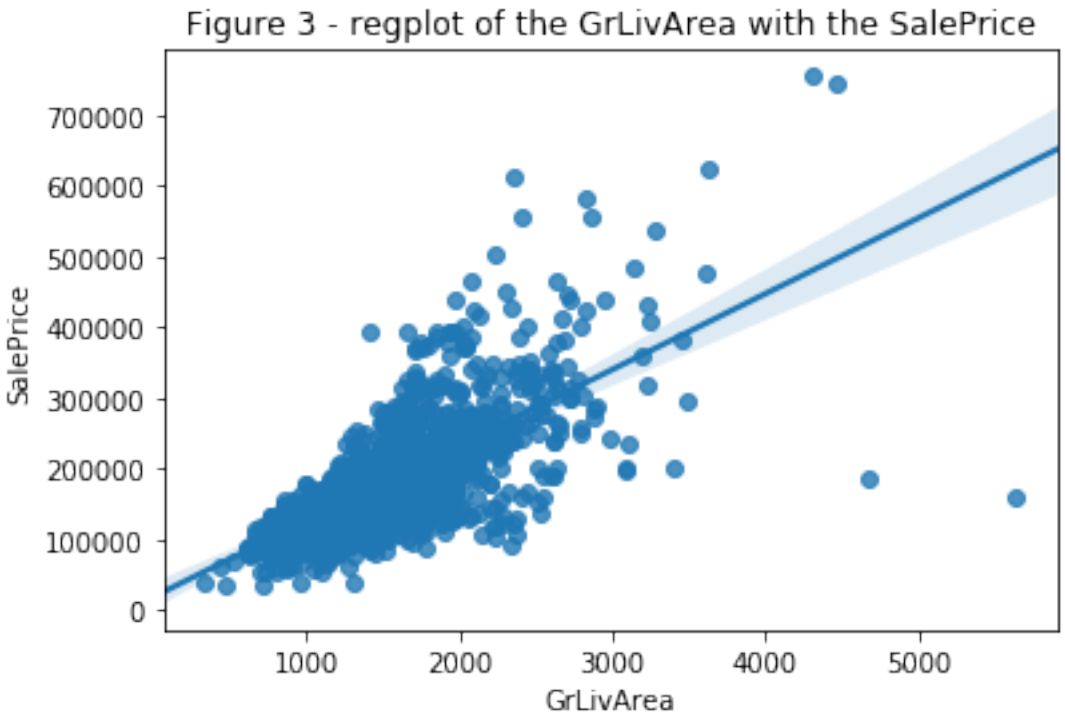


Figure2 – RegPlot of the GrLivArea

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

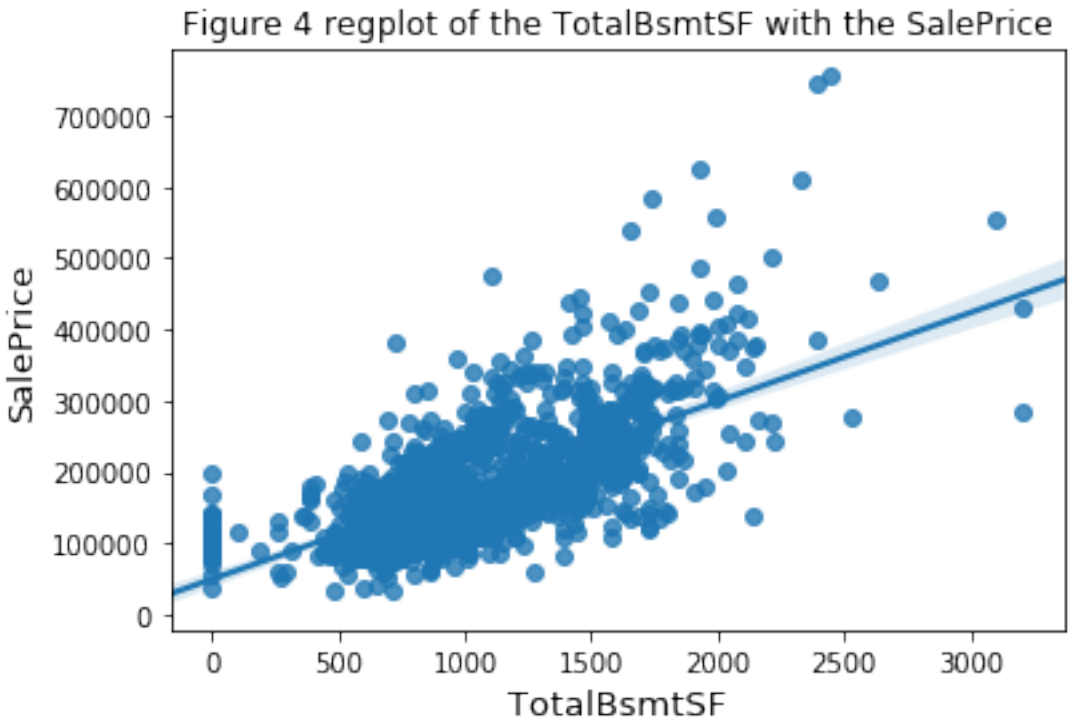


Figure4 – RegPlot of TotalBsmtSF

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

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

As explained in the data set dictionary, for a set of features NA means "No" or "0". For example, 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.

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.

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.

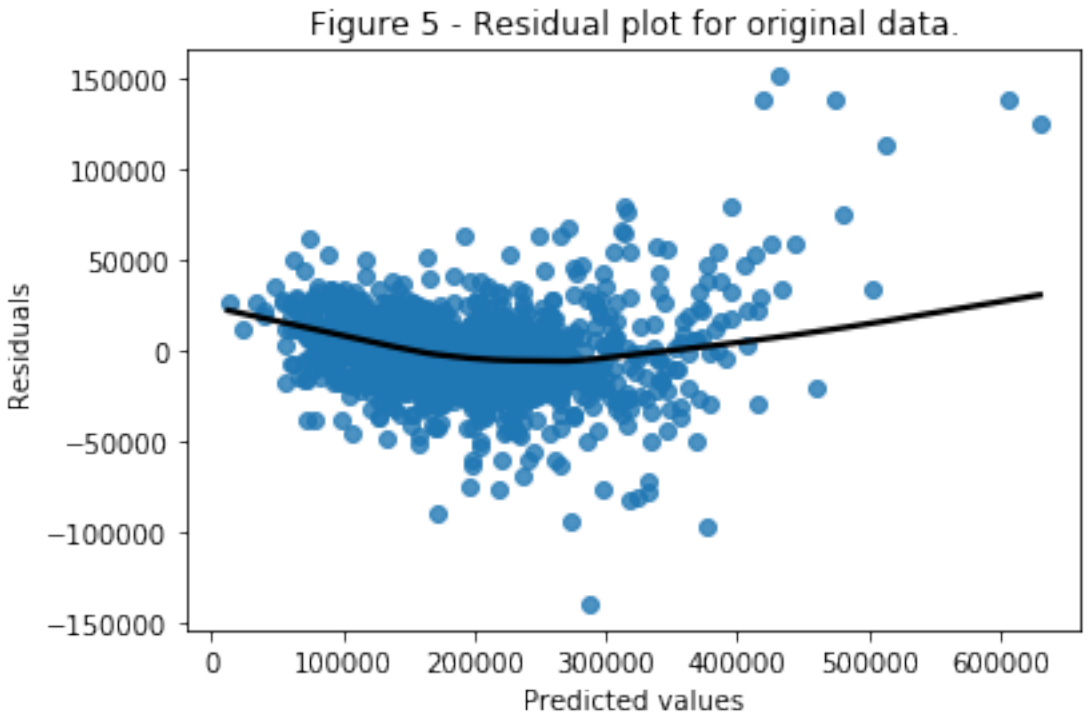


Figure5 – Residual plot for original data

Mean square error: 492970222.921

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

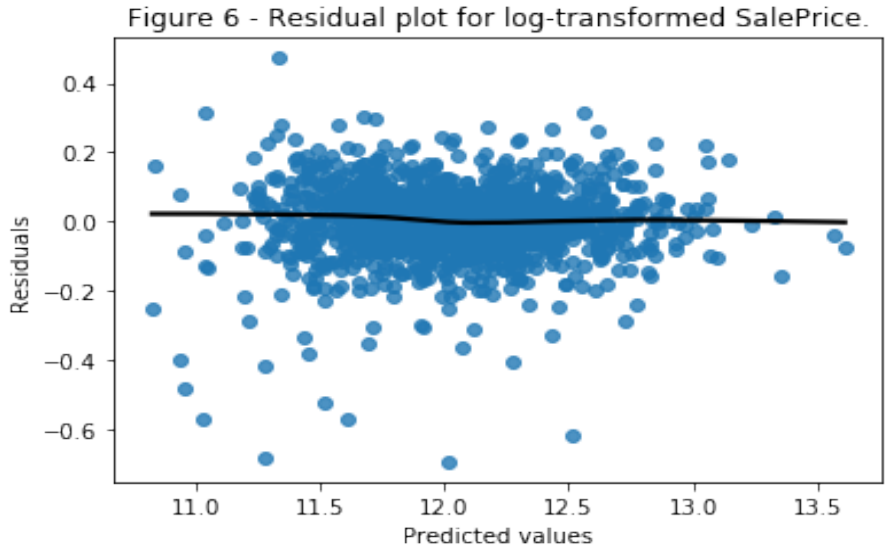


Figure6 – Residual plot for log-transformed SalePrice

```
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.

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.

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.

Feature scaling

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

Algorithms & Techniques

Linear Regression/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 analytically 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.

```
Average RMSE: 0.12535182602487316
```

Partial least squares regression

Partial least squares regression (PLS regression) is a statistical method that bears some relation to principal components regression; instead of finding hyperplanes of maximum variance between the response and independent variables, it finds a linear regression model by projecting the predicted variables and the observable variables to a new space. Because both the X and Y data are projected to new spaces, the PLS family of methods are known as bilinear factor models. Partial least squares discriminant analysis (PLS-DA) is a variant used when the Y is categorical.

```
Average RMSE: 0.13130323879877792
```

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 can control the width

of the margin using the regularization parameter C. Lower values of C give smaller margins and vice versa.

We have used the linear kernel as it gave far the best result when comparing to rbf or sigmoid. It is a sign that a linear fit is well adjusted to the true data and also can explain that no regularization was necessary. The best value for C was 1. We kept the other parameters in their default values.

`Average RMSE: 0.1263581422495872`

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.

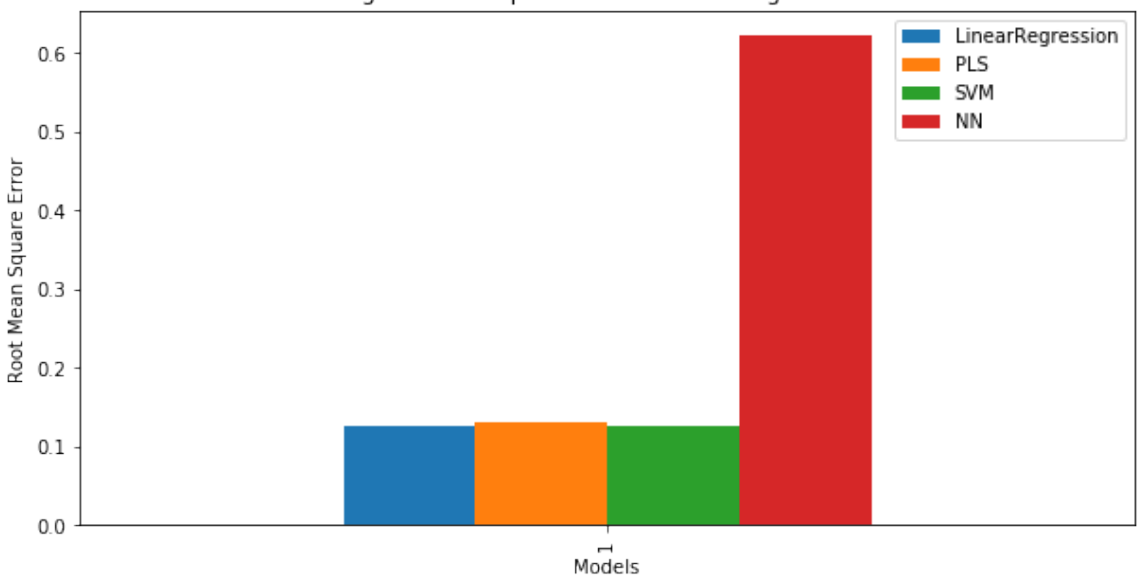
A one hidden layer MLP has a function in the form:$f : R^D -> R^L$ where D is the size of input vector x, and L is the size of the output vector f(x).

We think this model the most difficult to find a good adjustment of complexity. 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 value for the regularization parameter alpha of 0.0001 which means an agressive regularization and a net of three hidden layers with 80, 50 and 20 neurons respectively. Even so, it was not enough to get comparable results with the other models.

`Average RMSE: 0.6236172983183979`

Result

Figure 7 - Comparison of RMSE among models



Comparison/Discussion

Most models exhibited similar performance except for NN. 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.

SVMs are non-parametric models (i.e., the complexity grows as the number of training samples increases). Training a non-parametric model can thus be more expensive, computationally, compared to a generalized linear model. Also, we can end up with a lot of support vectors in SVMs; in the worst-case scenario, we have as many support vectors as we have samples in the training set. Although, there are multi-class SVMs, the typical implementation for multi-class classification is One-vs.-All; thus, we have to train an SVM for each class – in contrast, decision trees or random forests, which can handle multiple classes out of the box. In

SVMs, we typically need to do a fair amount of parameter tuning, and in addition to that, the computational cost grows linearly with the number of classes as well.

PLS showed good results as to the nature of the model as it runs on many variables that are often correlated with each other and for this model we provided nearly 40 variables. In PLS regression, the emphasis is on developing predictive models. Therefore, it is not usually used to screen out variables that are not useful in explaining the response. PLS can calculate as many components as there are predictors; often, cross-validation is used to identify the smaller set of components that provide the greatest predictive ability. If you calculate all possible components, the resulting model is equivalent to the model you would obtain using least squares regression. PLS can fit multiple response variables in a single model. PLS regression fits multiple response variables in a single model. Because PLS regression models the response variables in a multivariate way, the results can differ significantly from those calculated for the response variables individually.

As to why we saw not so great results with Neural Networks is because they require “relatively” large datasets to work well, and we also need the infrastructure to train them in reasonable time. Also, deep learning algorithms require much more experience: Setting up a neural network using deep learning algorithms is much more tedious than using an off-the-shelf classifiers such as random forests and SVMs. On the other hand, deep learning really shines when it comes to complex problems such as image classification, natural language processing, and speech recognition. Another advantage is that you have to worry less about the feature engineering part.

Data preprocessing have been proven to be a crucial part of the work, for instance addressing 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.