DATA MINING
CSC 503

PROJECT REPORT
ON
HOUSE PRICES PREDICTION

SUBMITTED TO:
DR. NISHANT MEHTA

SUBMITTED BY:
Group E :
DIVYANSH BHARDWAJ (V00949736)
RAJNEESH GULATI (V00949939)
ARSHIYA GULATI (V00949938)

Introduction

The house buying/selling task is the one which is very tedious and imbibes in it the complications that are difficult to be dealt with manually. One can not just maintain a register of all the houses he is interested in and describe them feature by feature. This problem is a typical machine learning problem which has a quite simple and efficient solution by predicting the sales price of the house using various machine learning algorithms.

So, our project is all about predicting the final sales price of a house and the final price of the house depends on various features such as its location, number of rooms, bathrooms and many more. Taking all these features into consideration and estimating the final price manually is difficult for both the buyer and the seller.

And if all the features are not taken into proper consideration it may result in a loss for either of them. Financial brokers are normally always consulted while buying or selling a property, who give us an idea about the actual price. But they may always not be honest and correct (as everybody is taking care of his/her benefit first), so the buyer/seller should also be aware of price or say must have an idea about the actual price.
So, as data scientists, we have tried to implement a project that estimates the sales price of a house in Ames region located in the story county, Iowa in the United States and is approximately 30 Kms away from central Iowa. The final sales price of the house depends on some essential features like the present age of the house, no. of bedrooms and washrooms in the house, location of the house and its proximity to the major areas of the city, neighbourhood/locality of the house and many more. For an individual who is looking to buy/sell a house cannot cover each and every angle from which he should be able to predict the price of the house. Just imagine how big this problem is for a person who is new to a city/country and is looking to buy a house such that all his needs are taken care of.
Hence, the problem becomes of utmost importance for such instances and hence we, as data scientists are given the duty to make these real life hurdles look easy for the people of the society so that buying/selling a house should not be considered as a humongous task in the future.

Formal Problem

Putting it more formally, we found the Ames housing dataset which contains the details of the houses in the Ames region of the story county, Iowa in the United States. Our goal for the project was to predict the final sale price of the house based on a total of 81 features in an efficient way so that we get a higher accuracy or low error during the prediction of the sales price of the house. The dataset, in total, consisted of 1460 data samples or house samples.For achieving the goal defined, we trained and tested the dataset using three different models. Using these models we checked for the accuracy (computed from the R-Squared error) and RMSE (Root Mean Squared Error) of the sales price and checked which model will work best for this prediction process.

The models we have considered for training the dataset are linear regression, neural network and random forest.The dataset we have worked on for our project is Ames Housing Dataset compiled by Prof. Dean De Cock of Truman State University and is already divided into testing and training data sets. The dataset has in total 81 features/columns of which Sales Price is the variable whose value must be predicted. The data available to us was in the form of two CSV files, train.csv and test.csv containing training and testing data, respectively. In total we have 1460 house samples to work with.

Related work

On Kaggle, there is a competition going on under the name ‘Advanced House Price Prediction’, where teams are participating posting the approaches they took with the data set. We studied different methodologies before working on the data set and noted both ups and downs of those methodologies. We also studied different features that are given in the data set to know more about them and their relevance with the house. By doing this we were able to come up with our own approach. Our goal was to select those features that are relevant to the data set and drop those which seemed redundant. We have discussed the approach in the next section.

Approach

The dataset used for this project had 81 features in total of which **SalesPrice** being the dependent variable i.e. all other features decided or impacted the sales price of the house. After analysing the complete dataset we found out that the data needed to be cleaned. Before proceeding with the coding part, we prepared a formal documentation in which we collaboratively discussed the technique/intuition behind approaching the feature engineering part of the project. So, we came to a conclusion that all three of us should work individually on the approach to do the feature engineering and cleaning and then we will extract out all the important points that all of us would have brought in and make a hybrid version of it. We proceeded in the way described and came to a final methodology of doing the feature engineering as described below.

So, we first applied feature engineering and selection techniques that dealt with removing the features with more than 80% of the missing values and were left with 76 features to deal with. After that, we decided to list down all the features that had at least some percentage of the missing values and clean them/fill them with the appropriate methodology. Hence, we computed the percentage wise list of the features with missing values and sorted them to know which feature had the most missing values and then got an intuition out of it as to why they are missing or does these missing values signify anything about the feature.
Just a brief about all the features, the features had the nature such that they could be grouped. For example, we could group all the features related to Garage, Basement, Lot etc. So, we decided to clean the features in the group.
Now, coming on to the main approach, we divided the features in two categories: numerical and categorical. We did this to segregate the features based on their nature and then we cleaned them by taking numerical and categorical features of each group.
We found that the dataset had a feature **GarageYrBlt** which means the year the garage was built and this feature had many **'zero'** values so we decided to drop this feature as it was not making sense to keep this feature as for most of the houses, the information was not available. Also, the features **GarageType**, **GarageFinish**, **GarageQual** and **GarageCond** had NAN values at the indices where the index of GarageYrBlt was 0. This clearly meant that Garage was not there in these houses. Hence we substituted these features with **'None'** value as they all were categorical features. We dropped 'GarageYrBlt' as we assumed that the garage was either built when the house was built or when renovated. Also in the correlation graph that we analysed for all the features, this feature did not added significantly to the target variable as there were 'zero' values for many of the houses.
We followed the same approach with features related to basement and filled none instead of NAN in the features related to basement where, '**TotalBsmtSF'** (Total Basement Surface Area) was 0, concluding that there were no basements in that house. We did not remove the feature TotalBsmtSF as it had very less zero values relative to the whole size of the dataset.

There were another set of features with the same problem i.e. '**LotFrontage'** and '**LotConfig'**. But the problem for these features was that we couldn't identify whether to fill the values with 0 or none or some other value because they had missing values and no other feature existed to suggest that the value for these features should be 0 or none. In short we didn't know why they had missing values. Fortunately, after some close observations, we figured that there was another feature named **'LotShape'** which indicated that for indices where the above two features had missing values, we actually had some values for the Lot. Hence, for **'LotFrontage',** we decided to fill it with the **median** values. For **'LotConfig'**, as it was a categorical feature, we applied the forward fill method provided by pandas which does nothing but fills the value which the previous row has for that feature. As the number of missing values were not that high, it was the best suited method according to us to fill the missing values.
After this we were left with three more features that had '1' NaN value each i.e. Electrical, **'BsmtExposure'**, **'BsmtinType2**'. To fill these missing values we followed the concept of forward filling that fills the same value as that in previous value in the **row/col**.
After this part, all the missing values were dealt with and we had 0 missing values or NaN for all the features.
Now, we were looking for the features that could be merged somehow or could be combined to get a more meaningful feature out of it. We saw that there were two features **'YearRemodAd'**, which defined the date of the renovation of the house if it was done, otherwise it had the same value as the house was built which was **'YrBuilt'**. We decided to make a better and more meaningful feature out of it which was the present age of the house. Hence we made the feature **"PresentAge"**, by simply subtracting the "YearRemodAd" from the feature "YrSold". After that, we dropped the three features, "YearRemodAd" and "YrSold" and "YrBuilt".

Moving on to the next set of features, we merged **"OverallCond"** (Overall Condition of the house) and **"OverallQual"** (Overall Quality of the house) to **"OverallQC"**, removing the former two columns. Both the features, "OverallCond" and "OverallQual" had numerical values ranging from 1 to 10 to describe them. We merged both of them by taking the average of the two and extracted a new feature out of it "OverallQC" (Overall quality and condition of the house).
We had another set of features of the same type, **"ExterQual"** (External Quality of the house), **"ExterCond"** (External condition of the house). But these were categorical features. Similarly, we figured out another such pair of feature, **"GarageQual"** and **"GarageCond"**, and that's when we decided to encode the categorical variables so that we could merge them afterwards into one.
However before talking about the above features, we want to mention a feature **"MoSold"**, which meant Month in which the house was sold. This feature was dropped because the houses in the dataset had pretty good variance in terms of their ages or year built. Hence, this feature could not provide us with any important information. Also, we did not have any feature which could tell us the month in which the house was built. Hence, it was much better to discard the feature.

Now we discuss the features which were left to be combined. Once the data was cleaned from the above features, we were left with 67 features finally(including categorical and numerical). Now, we needed to encode the variables in some format. In categorical variables, we have two kinds of variables with us, Nominal and Ordinal. Nominal variables are the one's which do not have any order associated with their values where ordinal features have some kind of order associated with their values. Keeping in mind their nature, we had to encode the variables in a way that they do not lose their meaning after getting encoded. Hence, we devised three types of encoding strategies as we were not clear about the fact that which would yield us better results, namely, Label Encoding, Binary Encoding and One Hot Encoding. In Label Encoding, we treated our nominal features as ordinal features and encoded them as we did to ordinal features. After doing label encoding, the total number of features we got were 67.

FeatureX (we apply label encoding on it)	FeatureX
Ex (Excellent)	5
Gd (Good)	4
Fa (Fair)	3
Po (Poor)	2
Bd (Bad)	1
Wo (Worse)	0

In binary encoding, the first step is same as label encoding, at second step we convert the integer values of a feature to binary e.g. if a feature after label encoding takes' values as 5,4,3,2,1 then the binary representation would be 0101, 0100, 0011, 0010, 0001. So for 0101 we would have 4 different columns created in the data set. Look at the table below for better clarity

FeatureX (we apply binary encoding on it)	FeatureX_1	FeatureX_2	FeatureX_3	FeatureX_4
suppose it takes value 5 here	0	1	0	1
suppose it takes value 3 here and so on...	0	0	1	1

Continuing on the example from binary encoding, if a feature takes value 5,4,3,2,1 then in one-hot encoding 5 different columns would be created. Imagine, if for a certain row value of the feature is 5 then it is represented as 10000 (each digit in its own column) in the data. Check the table below for better clarity.

FeatureX (we apply one-hot encoding on it)	FeatureX_1	FeatureX_2	FeatureX_3	FeatureX_4	FeatureX_5
suppose it takes value 5 here	0	0	0	0	1
suppose it takes value 3 here	0	0	1	0	0
suppose it takes value 1 here and so on.	1	0	0	0	0

The number of columns generated for Label encoding were the least followed by Binary Encoding and lastly one-hot encoding. For binary encoding, like in the above example the column for '**FeatureX_1**' would always be zero and this was the case in our dataset so we were able to drop some columns further and we got 118 columns finally for binary encoding. For one-hot encoding, the final number of columns was 213. We used a library called **category_encoders** to do binary and one-hot encoding.
Finally, now we had the data all already to be fed in the models for training. We split the data in 80-20 ratio, training and testing respectively. We went on with training three models: Linear Regression, Random Forest (with KFold Cross Validation) and Neural Net (with PCA). The results obtained are mentioned in the next section.

Results

Once the pre-processing of the data was done. We split the data into 80-20 for the training and the testing purposes. We trained our dataset using three models and fed the pre-processed data into the models that are discussed above. We trained each model three times, on the data set with categorical features differently encoded each time, so as to know which approach works the best.
We calculated the accuracy (R-squared) and the RMSE (root mean square error) of each of the above mentioned models for binary encoding, label encoder and one-hot encoding respectively.

Accuracy (R-Squared)7	Random Forest Regressor	Neural Net Regressor	Linear Regression
Binary Encoding	88.49%	74.96%	86.38%
Label Encoder	87.73%	79.36%	84.28%
One-hot Encoding	88.61%	80.26%	Not Defined

Table 1: Accuracy of different models

RMSE (Root Mean Squared Error)	Random Forest Regressor	Neural Net Regressor	Linear Regression
Binary Encoding	27194.08	40115.23	29588.43
Label Encoder	28085.44	36418.80	31786.58
One-hot Encoding	27052.50	35623.03	Not Defined

Table 2: RMSE of different models

From table 1 and table 2 we can see that random forest gives the best and consistent accuracy for all the three encoders and least RMSE. Linear regression gave us some ugly numbers for one-hot encoding for accuracy and RMSE both, so we mentioned it as not defined. Neural Net gave best performance for one-hot encoding .

Since, random forest gave us the best performance we tried to improve it further using k-fold cross validation. In our case we considered k to be 5 and then calculated the accuracy for the above mentioned encoders. Hyper parameters we tuned are: **ccp_alpha** (the pruning parameter), **max_depth** (the maximum depth of the tree to which it can expand) and **n_estimators** (the number of trees in a random forest).

Accuracy (R-Squared)	Random Forest	
Binary Encoding (ccp_alpha = 1, max_depth = 50, n_estimators = 100)	Validation Set	Test Set
	Best Score: 83.5%	Best Score: 88.49%
Label Encoding (ccp_alpha = 0.1, max_depth = 25, n_estimators = 50)	Validation Set	Test Set
	Best Score: 83.41%	Best Score: 88.09%
One-Hot Encoding (ccp_alpha = 1, max_depth = 50, n_estimators = 100)	Validation Set	Test Set
	Best Score: 83.68%	Best Score: 88.61%

Table 3: accuracy (R-squared) of random forest classifier after 5 fold CV

When comparing performance of random forest from table 1 with table 3 we see no significant increase in performance except a little increase for label encoding. We think that the default parameters we were working were already the best parameters and which was further confirmed by K Fold cross validation. We chose the value of k as 5 because the size of the data set is small (1460 data points with further 80-20 split), choosing a value any greater would have impacted the performance because the size of the data set at each iteration of K Fold would have been too small.

We further tried to reduce the number of features using feature extraction. We extracted those features which had 95% information on them. We fed the data set after applying PCA to the neural net.
With label encoding we got 49 features out of 69 with 95%variance.
With binary encoding we got 79 features out of 118 with 95%variance.
With one-hot encoding we got 134 features out of 213 with 95%variance.

Accuracy (R-Squared)	Neural Net
Binary Encoding	84.85% (9.89% ^)
Label Encoding	82.67 (3.31%^)
One- Hot Encoding	83.94% (3.68%^)

Table 4: accuracy (R-squared) of neural network

RMSE (Root Mean Square Error)	Neural Net
Binary Encoding	31221.64 (8893.59 Dec)
Label Encoding	33392.05 (3026.75 Dec)
One-Hot Encoding	32146.93 (3476.1 Dec)

Table 5: RMSE of neural network

Comparing Table 2 and Table 4 for neural net's accuracy score, we see considerable improvement in the performance. With PCA, our goal was to further reduce the size of that feature set, to reduce the training time. But, to retain 95% variance not much features were dropped by PCA (e.g. for label encoding we expected final number of features would be in the range 20-30, but we got 49), this suggests that contribution of each feature is quite significant. We thought after doing PCA, we would see a minor drop in performance but instead got a significant increase in performance for neural nets. We were not able to figure out the reason for this increase in performance, as the neural net is a black box and we don't know what is going on inside of it.

Conclusion

There are many observations and conclusions which play a crucial role in accordance with any machine learning/data mining project. First of all, the feature engineering part should be done very carefully as this is the part which if done in a right way can lead to significant improvement in the performance of the model. While combining or merging the features, care should be taken regarding the information loss which could happen there and then. Also, while encoding the variables, we identified that we just can not simply use the library and encode the variables in any unorderly fashion. Rather, sufficient care should be taken while encoding them and one has to encode some features manually as we can not just give the same range of values to any categorical value. What we concluded during this stage is to make groups of the similar features and then encode them. If the feature engineering part is done right, not much is left as the main performance of the model is highly dependent on the features that we are using. Secondly, we analysed the performance of binary encoding and one hot encoding. For every model, binary encoding has worked much better than the one hot encoding. Moreover, one hot encoding generates a lot of features and results in a sparse matrix of features which again causes trouble as it takes a lot of time to train them. Also, when we trained Linear regression with the One hot encoding, it gave us some very ugly numbers. So our experience with one hot encoding was not good. Another important observation that we made was, there was a significant improvement in the performance of the Neural network when we applied PCA on the dataset. As mentioned above, the neural network is a black box and we do not know what is happening inside the black box. Hence, it is much harder to say what caused this. Lastly, Random forest has given the best performance out of all the models and worked remarkable well for all the three encodings. The reason behind it is that random Forest is an ensemble method of training the model and takes the wisdom of the crowd to get to the final decision. Although the K-Fold cross validation in our case did not give any surprising improvements in the model, yet it is a very crucial part of any model training.

References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[4] https://medium.com/@mandava807/cross-validation-and-hyperparameter-tuning-in-python-65cfb80ee485

[5] https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd

[6] https://towardsdatascience.com/encoding-categorical-features-21a2651a065c

[7] https://pypi.org/project/category-encoders/