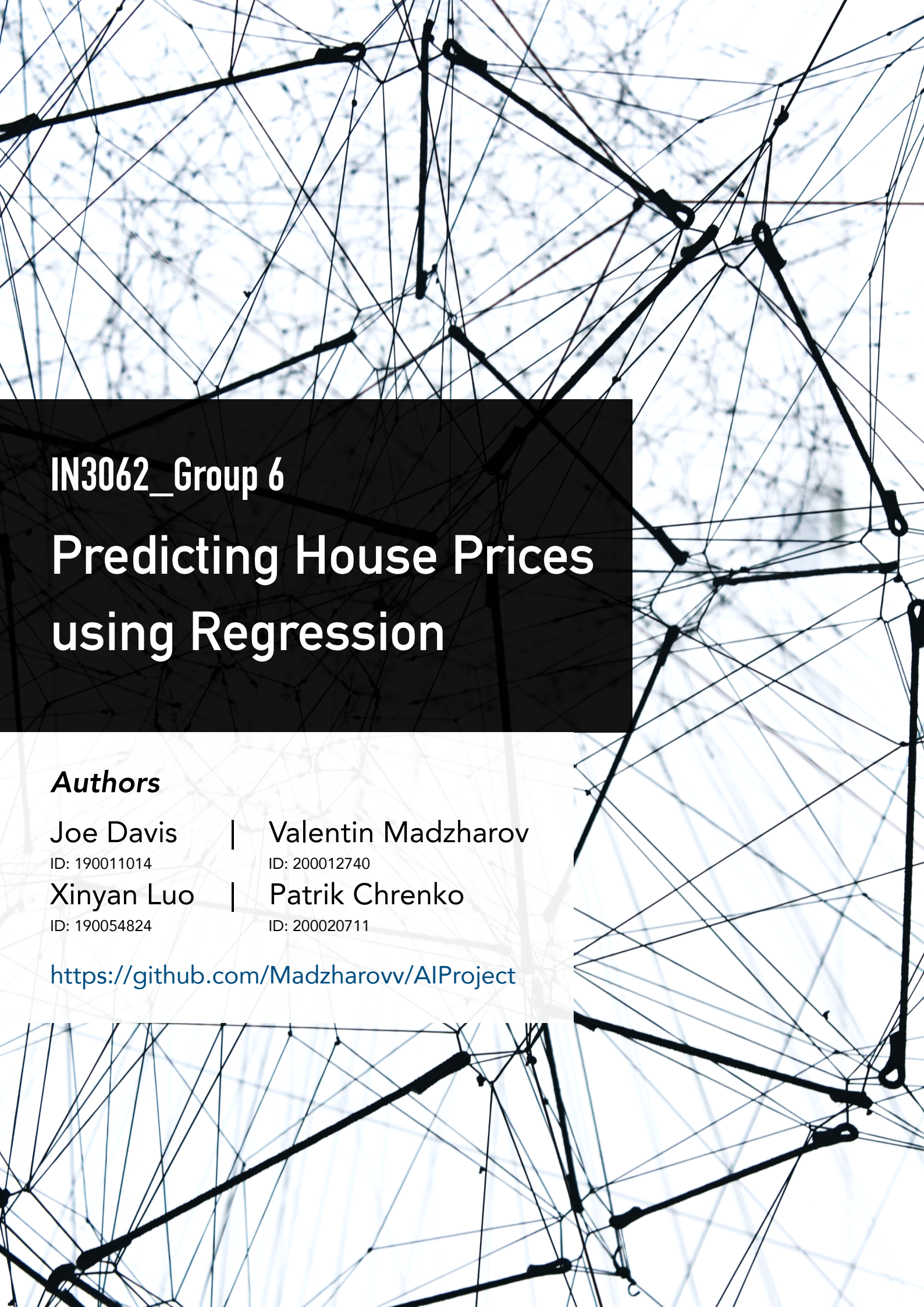


IN3062_Group 6

Predicting House Prices using Regression

Authors

Joe Davis | Valentin Madzharov

ID: 190011014 | ID: 200012740

Xinyan Luo | Patrik Chrenko

ID: 190054824 | ID: 200020711

https://github.com/Madzharovv/AIProject

Introduction

Property valuation is an important task in the real estate market, not only relied on for property sales and purchases, but also for mortgages and loan guarantees. Property valuation relies heavily on data about the property, as well as data from the rest of the market to accurately estimate a property's value. Considering this, our aim was to explore the viability of creating an accurate AI model for real estate evaluation. Real estate is a huge industry and so our exploration focused on the private housing market in Ames, Iowa, US.

The aim of this project was to accurately predict the prices of properties using regression models with which we can consider significant property features that contribute to the value of a property as they will allow us to produce the most accurate prediction**.** In order for us to have accurate predictions we were focused on which property features did and didn't have a significant effect on the overall price of the property. Another detail which we focused on was which models would be most adequate for having the most precise predictions.

With the acceptable margin of error in housing valuations set at around 10% - 15%, we set 15% as the target accuracy rate when calculating the RMSE in comparison to the mean property price. This required making sure that the raw data is appropriately cleaned through methods such as enumeration, normalisation, removing outliers and correlation detection for the feature set. We then set out testing several training methods including linear regression, support vectors for regression, neural networks, and random forest, to then compare which model would yield the most accurate predictions for our use case

As the target of our predictions would be the house sale price, our study would be focusing on utilising regression models and techniques.

Data

When choosing a dataset we wanted to ensure that our data was not synthetic/ generated using an AI, and originating from a real-world study. We selected this dataset as it had low number of missing data. The dataset, however, only had a record size of just under 1500. We understood that this could lead to potential issues with accuracy in our models caused by an insufficient level of training data. Nevertheless, we decided that we wanted to determine whether we could produce accurate models with a limited amount of data.

When exploring the data we first had to perform data cleaning where we discarded instances where properties and their features had missing or misleading data. We decided it was most appropriate for all empty values to be replaced by 0, as empty values would only be used to indicate something non-existent, rather than a missing value. It's because of this that approaches, such as using means and medians to substitute missing values, would not be suitable. While it was logical to replace our null fields with 0, we did attempt to replace them with these averages for some features just to experiment in hopes that it would increase our models' accuracy.

Our dataset contained both categorical and numerical features, which meant that there was a good opportunity for us to look into the importance of encoding in our predictions, Both

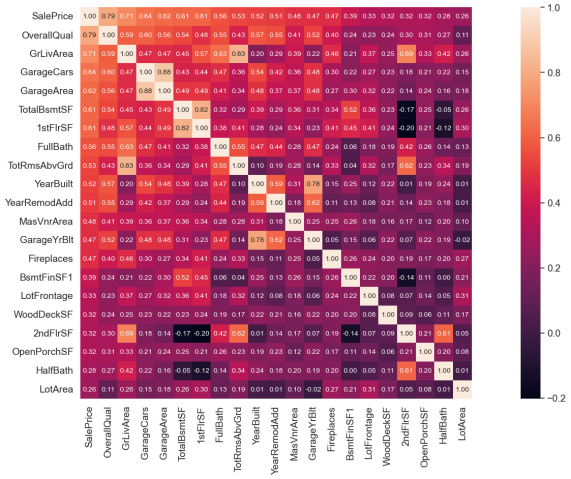
regular label and one-hot encoding were tested but we decided to stick with label encoding as it would usually have a negligible difference in testing.

We proceeded with filtering through the 79 columns containing property features and discard the ones which were misleading or insignificant to the problem that we are solving.
The columns which we realised to be the ones that had the tremendous impact on the overall property price were selected using the ".selecttype(objects)". This allowed us to find out our encodable entries.

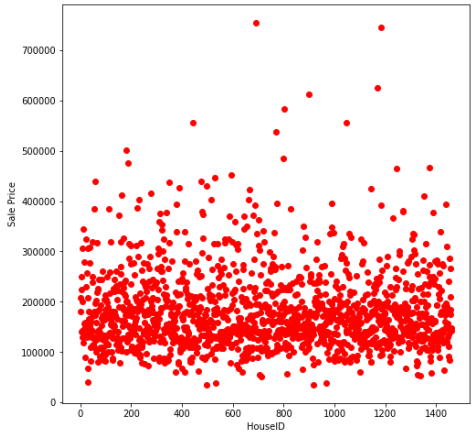
Correlation
We wanted to specify features that would be omittable when producing our models so we attempted two methods of filtering our data to find the most optimal features for our input.

Method 1:
While modelling the data it became apparent that there was significant noise in the dataset due to the relatively high amount of features for every record. We, therefore, wanted to identify which columns could be omitted to improve model accuracy. To do this, we looped through each column and compared what effect on the RMSE its omission from the training would have. This was repeated until the RMSE could not be lowered. (This was tested using a basic Linear Regression Model)

Method 2:
By creating a correlation matrix which is a heatmap between all numerical columns, we can find out that nearly half of the columns only have a correlation rate with the Sale Price that is lower than 0.25. And also there is no column that has a strong negative correlation with it. So we dropped the columns that had a correlation rate lower than 0.25 to decrease the data set complexity. Then we create a more specific heat map for the Sale Price.



~Figure 1: Correlation Heatmap of our datasets top features.



~Figure 2: Scatter Graph of all House Sale Prices

In our dataset, we understood that house prices can fluctuate greatly, with some houses having higher valuations than our models would be able to justify from the provided data. (We used a scatter graph to display the pattern of house pricing, see Figure 2, it displays a huge difference in the potential sale prices) So we decided to remove these outliers in many of the models. We found that the accuracy would increase when removing all values 1 standard deviation away from the mean (though usually 2 standard deviations would be used in maths, but we found this had a greater effect on the accuracy).

Method

Beginning development

For each of the models we implemented, we initially tested using the raw data but dropped all rows with null inputs and all non-numeric features. Afterwards, we built four preset inputs to benchmark the accuracy of our models using certain features. These four presets were; the first preset, using the same features listed in their own benchmark linear regression model hosted on Kaggle. The second preset was using all numerical features available. The third preset was using only encoded categorical features. And the fourth was the combination of all features including encoded categorical features. (In most models, the fourth preset was the most accurate). We then transitioned to using the version of our dataset that had converted all null values to zero for further testing.

Altering our inputs

The data we were inputting generally didn't have a huge difference in scale, with most numerical features ranging between hundreds to thousands. So we didn't expect the normalisation of our data to produce a significant improvement on our data, however we still attempted to apply normalisation across all of our features in hopes that it may increase the accuracy by any amount. We also noticed from the scatter graph that the sale prices don't closely fit a gaussian curve (as expected in a dataset obtained from a real dataset), therefore we didn't expect the standardisation of our data to be able to work as effectively with our models. We did however test their effects on each model regardless by using standard scalers.

Our measurement of accuracy

As our models would be predicting regression problems, we decided that using the RMSE would be our way of evaluating model accuracy. We decided earlier that a 15% error margin would be an acceptable result. So we calculated that the mean house price was 180921.20 (2 d.p) and therefore our 15% margin for the RMSE would be 27138.18 (2 d.p).

Linear regression model:

As our first model, we decided to test both test-train split and K-fold techniques for allocating our test and training data. Generally, a Linear Regression Model has very little hyperparameters for us to alter and setting positive or fit_intercept to "True" both had detrimental effects on the model's accuracy.

Findings for our splits

We found that test-train split would produce models with RMSE values lower than in our K-fold experiments (When using a number of folds that would yield our most accurate models). We tested a range of fold values (1-30) but found that it was most accurate with a lower number of folds (3-7). It did have the potential to produce lower RMSE values than a regular split however it was far more unstable in the results and outputs provided. We decided to use a test_size of 0.2 for our split in the rest of our models, as our dataset's size was limited and wanted to ensure that a majority of potential values were accessible in the training set. Our Random_State values are different in many of these models, as we tested many values to evaluate how accurate the models were.

Random Forest
Random Forest's main hyperparameter would be the number of n_estimators used, therefore to identify an optimal number of estimators to use, we created a graph to portray the RMSE of the model up to 50 estimators. Changing the criterion would cause us errors for "absolute_error", and less accurate results other than the default "squared_error"

SVR
We developed two separate SVR models, one using a "linear" Kernel, and another using an "rbf" kernel. They both produced similar accuracy, with the "rbf" svr being slightly more accurate, we chose to keep the gamma value as "auto" to try and prevent us from causing overfitting when setting a high gamma in pursuit of accuracy. For the Linear kernel, we kept the C value as default. Increasing it would cause minimal increases in accuracy at the cost of a far greater computation time. We expected our SVR's to have the potential to be highly accurate as they work well with high dimensionality.

Neural Network
Our neural network (NN) was the only model we kept standardisation enabled, as researching online recommended it for NN models as something that would only be beneficial, and in testing proved so slightly. Our NN model consisted of three hidden layers with 80 units each. We did this to make sure it would not cause underfitting when predicting, and adding more than three hidden layers would not significantly increase the accuracy. Each layer used the "relu" activation function, (including our output layer) as all other activation functions would produce RMSE values upwards of 100,000.

We also put an activity_regularizer (AR) and a kernel_regularizer (KR) on the first hidden layer. We used L2 for the AR with a value of 0.01 which appeared to help reduce overfitting, however the use of L1 on our KR appeared to make no difference, we assume that the model found all features to be important enough to prevent them from being dropped, preventing sparsity from affecting our layers. We used the "Adam" optimizer for our model, as it performed the best (only marginally better than "rmsprop,adamax") with a learning rate of 0.0015. We found that a value close to the default learning rate worked best with our model. We also used a validation_split of 0.25 when running our models for 250 epochs, as this gave the model enough time to run to it's peak performance. By adding an early stopping monitor (monitoring loss) with a min-delta of 1e-1 and a patience of 7, it kept the model from overfitting itself.

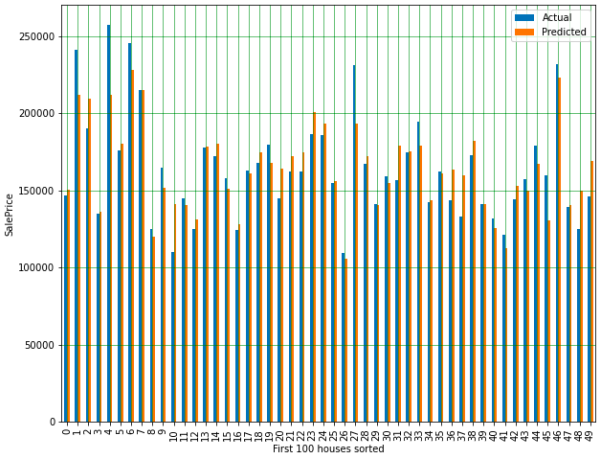
Results

General Results

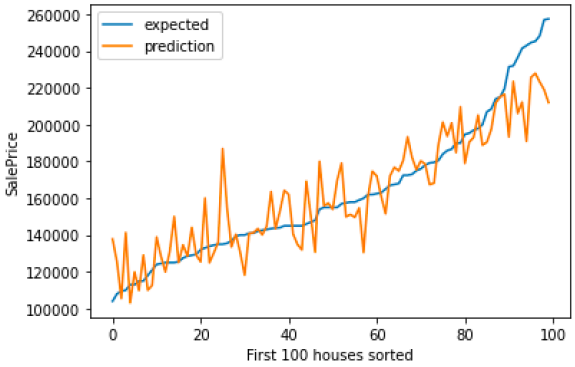
Our fourth preset for the inputs would tend to be the most accurate for each of our models. Even after our research through feature correlation we couldn't drop features without sacrificing accuracy or it having no effect on the model. This proved to us that the encoding of categorical features is important in the creation of model predictions. However the method of encoding used (Label or One-hot) had no effect.

Linear Regression Model (LNR)
While a simple model to implement, it would be one of our most accurate models producing an RMSE of 16907.24 (2 d.p) which sits comfortably in our error margin. We found that removing the outliers using only a single standard deviation (SD) from the mean managed to

approximately half RMSE produced. Unfortunately, standardisation and normalisation had no effect on our results. As you can see from figure 4 (Line graph of the first 100 houses in sorted order), our model has a high tendency to fluctuate between overestimating and under-estimating the value of houses, which we may have been able to correct with some more training data. In figure 3 (bar chart of the first 50 house results produced) We see that the trends from the Line graph are visible here too.



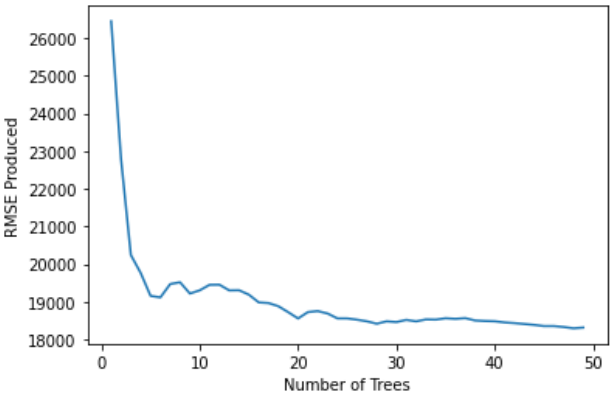
~Figure 3: Bar Chart of the first 100 house sale



~Figure 4: Line Graph of the first prices comparing predicted and actual values 100 house prices comparing predicted and actual values

Random Forest (RF)

We used a line graph in figure 5 to display how our RMSE value changed with an increasing number of estimators. We found that between 20 and 30 estimators produced the best RMSE for this model, but as observed in the graph we can see that the RMSE value would jump up on occasion, meaning our model must be overfitting to some degree. Our optimal RMSE value found was 18291.50 (2 d.p). While only slightly less accurate than our LNR, the RF was still comfortably under our margin of error. Neither standardisation or normalisation would benefit our results, we believe that only the training size is holding back this model, and that our hyper parameters are ideal. This model also benefited most from removing all outliers one SD away from the mean, on average shaving 3,000 from the RMSE.

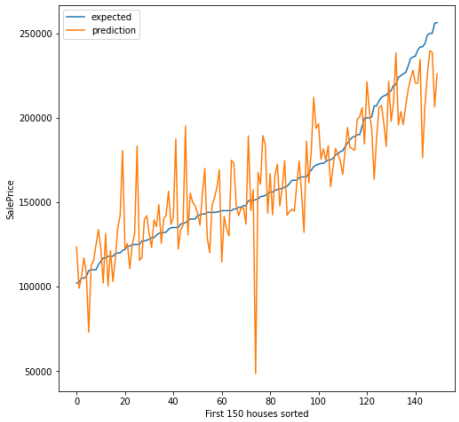


~Figure 5: Line Graph displaying the effect of N_estimators on the RMSE produced

SVR

For our linear kernel, the SVR also gained the most accuracy when removing outliers, a single SD away from the mean. In Figure 6 (Line graph of first 150 house prediction results compared to their actual values) we can see it follows the same behaviour as the LNR, but produces greater peaks of inaccuracy. We would get an extremely erroneous result (House ID approx 75*) consistently. We believe that this is due to the nature of linear kernel SVR's, which work optimally when the data isn't noisy and linearly separable. Due to our data's

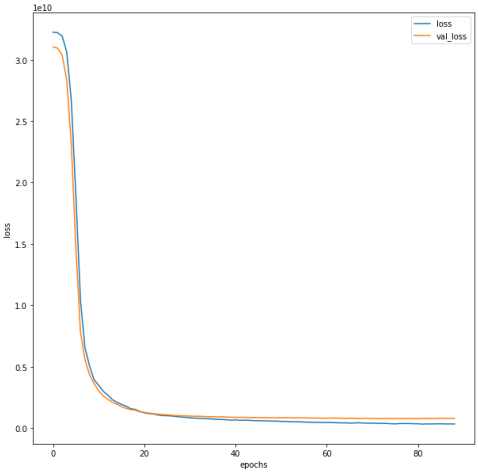
noise, it is struggling to produce an accurate result. It would produce an RMSE of 18757.32 (2 d.p) which also fits in our margin of error. But marginally less accurate than our other models. We did try to use the rbf kernel, and the results showed that the accuracy of the rbf kernel is way worse than the linear kernel. At low gamma value(gamma < 0.8) it is very easy to cause overfit in the case of higher gamma value(gamma >= 0.8). The best RMSE we can get on rbf kernel without overfitting is about 30000. When altering the model, we found that replacing our nulled numeric fields with mean values enabled it to compete with the linear kernel more closely.



~Figure 6: Line graph displaying first 150 house price predictions inc. anomalous result

Neural Network (NN)

This was our only model that benefitted when removing outliers that were 2 SD away from the mean. This could have been because this model had the complexity to perceive a greater distribution of sale prices more accurately. Using the Standard Scaler on this model reduced our average RMSE's by approximately 3,000. While running our models, it would often utilise the callback to our Early_Stopping monitor at epochs between 80 - 130 saving us some time and resources but indicating that the model had converged to a solution without risking continuous overfitting. In figure 7 (line graph of our loss plotted against val_loss) we can see that our "val_loss" is plateauing while the regular loss continues to decrease minimally. This is a good sign as it would indicate overfitting if our val_loss increased at all. Overall our NN with three hidden layers produced an RMSE of 22004.25 (2 d.p). It was our least accurate model but we believe it could become our most accurate with a larger training data set as well as some more experimentation with applying techniques to our data.



~Figure 7: Line Graph plotting Loss against Val_Loss

Evaluation and Conclusion

Overall, the models we managed to create were all able to perform within our designated margin of error, which means that we can confidently say that they all have a good degree of accuracy. It was unexpected that our NN model would produce the worst output out of our experiment, however there are techniques that we could have applied to the data, as well as other parameters we could have tested which may have changed this. For the rest of our models, they performed as planned when declaring their hyper parameters, for example we knew that 'relu' would be our most effective activation function as our model utilised multiple hidden layers (which sigmoid and tanh will struggle with) and our data's noise and complexity would be better suited to this architecture. We also can say that in a scenario with limited amounts of data, using a simpler model like LNR can prove greatly effective at

producing an accurate result, as long as the right techniques are applied to the data. It would have been beneficial if it was possible to also evaluate the effectiveness of unsupervised learning techniques, but as a regression problem it cannot be done. The most prominent issue in our dataset however, was the size of our usable data. With its size it greatly limited the potential of our models, and with the removal of outliers it further reduced the available training data. Luckily our outliers only summed up to approximately 200-300 records.

Our original aims of creating models that could be used in the prediction of property value (at least in relation to the area the dataset was collected from) within an acceptable margin were met. However we cannot faithfully recommend the use of these models in a professional environment without further training and testing data. Ultimately our experimentation to find the most important features for these models concluded that all features had a high enough weight in the models to yield a positive result when included, or a negligible effect when removed (we kept them in use incase of their potential positive effect on accuracy).

References

Stack Overflow. *(pandas - df.iloc[:,:-3])*. [online] Available at: https://stackoverflow.com/questions/53608653/how-to-select-all-but-the-3-last-columns-of-a-dataframe-in-python [Accessed 19 Dec. 2022]. // Used in code to retrieve specifically located features

scikit-learn (2018). *3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. // Used to research how to implement a Random Forest model for Regression problems. Provided greater clarity about all properties and parameters that we could modify to increase the accuracy of our model.

scikit-learn. (n.d.). *Support Vector Regression (SVR) using linear and non-linear kernels*. [online] Available at: https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html. // Used to research how to implement a Support Vector Machine for a regression problem. Like the Random Forest Regressor it allowed us to deepen our understanding of the model and see how it is affected by certain parameters.

Team, K. (n.d.). *Keras documentation: Optimizers*. [online] keras.io. Available at: https://keras.io/api/optimizers/. // Used to find all Optimizers that could be used in the Neural Network model.

Reference to all Tutorials and Lectures posted on Moodle. A majority of the understanding of our models is attributed to the Lectures explanations of the models, as well as The demonstrated code in the tutorials. They were used as templates for us to build our own models from and provided suggestions into ways that we could represent the data produced as well as how to measure the accuracy of our models.