School of Electrical and Information Engineering

University of the Witwatersrand, Johannesburg

ELEN4025A - Introduction to Machine Learning

Course Assignment - Individual Deliverable **Author:** **Gilad Kangisser (2367017)**

Group Members: Natan Grayman, Daron Sender, Ruth-Ann Wright

Kaggle Link: https://www.kaggle.com/giladkangisser

1. Introduction

This report encompasses a study on predicting house prices in Boston using advanced regression techniques on a plethora of distinctive features [1]. The dataset employed in this study stems from the Ames Housing dataset, a comprehensive collection amassed by Dean De Cock specifically for data science educational purposes [1]. Throughout the project, several machine learning models were explored, namely Linear Regression, K-Nearest Neighbors (KNN), and Random Forest models. The Root Mean Square Error (RMSE) was chosen as the primary evaluation metric, enabling the assessment and comparison of each model's accuracy.

This comprehensive project involved many aspects of the machine learning pipeline: data cleaning and preprocessing, feature selection, model selection and hyperparameter tuning, and at each stage, evaluation [2]. Additionally, various visualization techniques were harnessed to elucidate our findings. The objective of the investigation was to identify the most efficient regression model for predicting house prices and to identify the most impactful features and parameters contributing to their predictions.

The workload was broken down between the group partners in a coordinated method in order to enhance the efficiency of each member. As a result, the central element of my segment was implementing and the Random Forest model as well as its feature engineering and a portion of the data exploration.

2. Data Exploration

Initially, to gain a grasp on the dataset and its properties, each member independently experimented with a range of data exploration. Thereafter, the best of each exploratory tests was chosen to be included in the final version and were subsequently utilized in the feature engineering sections of the notebook.

2.1. Organizing Data

I initially classified the data into two distinct categories based on their characteristics – Numerical and Categorical. Although it was then noticed that numeric values are not a monolithic entity. The distribution of certain numeric features causes them to behave closer to the behaviour of categorical data, such as a small number of discrete possible states. Therefore, the numeric features were further split into Continuous and Ordinal. Continuous data consisted of values that can take on any value – in reality just a smoother distribution - within a certain range, such as the area of a house. Ordinal data, on the other hand, were the features that had a small number of numeric values. Lastly, Nominal data comprised of categorical data that did not have any inherent order or ranking or hierarchy.

To further simplify the categorization, a function was utilized which deemed a feature to be Ordinal if its unique numeric values were less than 5% of the total amount of samples in the dataset. All remaining features were deemed Continuous. This approach served as an initial rough classification to aid the subsequent data analysis process.

When combining with the other group members, there were certain attributes that can logically be placed into 1 of these categories but were not filtered as such based on the functional implementation. For example, the “YearBuilt” was designated as continuous by the requirements chosen, however, in reality years are more accurately described as ordinal. Although I argued that the categorization was

purely a proxy that was used to ensure effective data processing[1], the other group members chose to manually place these features into the opposing category.

2.2. Filtering Multicollinear Features

During the data exploration stage, I was also tasked with detecting and handling multicollinearity within the dataset. Multicollinearity occurs when two or more predictor variables in a regression model are highly correlated, so that one can be linearly predicted from the other at a substantial accuracy [3]. This can inflate the variance of the regression coefficients, making it unstable and challenging to interpret.

To identify multicollinearity, I examined the correlation between each pair of continuous and ordinal features in our dataset. This was performed by generating a Pearson correlation matrix – shown in Appendix: Figure 1. It a table showing correlation coefficients between variables [4]. Each cell in the table shows the correlation between two variables. The visualization of this matrix, known as a heatmap, served as a powerful tool in identifying features that were closely related to each other. A correlation threshold of 0.65 was chosen, indicating that any pair of variables with a correlation coefficient above this value was deemed to have high multicollinearity.

3. Model Experimentation

3.1. Feature Engineering

Implementing the successful filtering of collinearity was decided as a part of the feature engineering. However, the mere existence of multicollinearity does not warrant the removal of features. In the context of our predictive model, we wanted to identify which among the highly correlated pairs contribute more to the prediction of our target variable, 'SalePrice'.

Hence, we compared the correlation of each member of the highly correlated pair with the target variable. The intuition behind this step is that the variable with a higher correlation with 'SalePrice' would be a better predictor for our model, and thus should be retained. Conversely, the variable with a lower correlation with 'SalePrice' contributes less to our prediction and can be removed to simplify the model without significant loss of information. This procedure helped us to reduce multicollinearity and avoid its potential problems, resulting in a more reliable and interpretable set of predictor variables. Therefore, we ended up with an optimized set of numerical features in our dataset, providing a solid foundation for our model building process.

3.2. Model Investigation

This comparative analysis examines Random Forest Regression in relation to the other regression techniques, specifically Linear Regression, KNN, Decision Trees, Ridge, and Lasso.

Linear Regression: A staple regression technique, it assumes a linear relationship between independent and dependent variables. This method requires data to meet several assumptions, including linearity, normality, and homoscedasticity, which may not always be practical [3]. Random Forest, on the other hand, makes no such assumptions and can handle non-linearities in the data efficiently.
Ridge and Lasso: These regression techniques are extensions of linear regression that include regularization to prevent overfitting. Ridge regression performs L2 regularization, while Lasso employs L1 regularization, which can lead to feature selection by reducing certain coefficients to zero [5][6].

K-Nearest Neighbors (KNN): KNN regression is based on feature similarity, predicting the value of a new observation by calculating the mean of the K most similar instances. Unlike other techniques, KNN may perform poorly with high-dimensional data due to the curse of dimensionality [7].

Random Forest is a popular ensemble machine learning method that incorporates the decision tree algorithm [8]. Its application in regression tasks provides an intuitive, non-parametric, and effective means of handling complex data structures, making it a favored choice for many data science

[1] The purpose of separating the data was to differently handle features depending on their distribution such as using correlation or ANOVA or the different methods of finding the category's variance. Therefore, the categories were only proxies and not indicative of an actual attribute of the data. Thus, manually placing them into a category whose distribution is not accurate in describing the data, is an error in my opinion.

applications. Random Forest Regression operates by constructing numerous decision trees during training and outputting the mean prediction of individual trees. Its power resides in its ability to control overfitting without requiring extensive data preparation or feature engineering. The method's non-parametric nature allows it to model intricate, non-linear relationships.

3.3. Hyper Parameter Tuning

3.3.1. Random Forest Parameters

- `n_estimators`: This is the number of trees you want to build before taking the maximum voting or averages of predictions. A higher number of trees gives you better performance but also makes your code slower. In this model, two values (50 and 100) were tested [9].
- `max_depth`: This indicates how deep the tree can be. The deeper the tree, the more splits it has, and it captures more information about the data. Here, the maximum depth is limited to 20 for some trees to avoid overfitting, while other trees are allowed to fully expand with None as the depth [9].
- `min_samples_split`: This represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. Here, it's between 2 and 5 [9].
- `max_features`: These are the maximum number of features Random Forest is allowed to try in individual trees. Here, auto means that all features are considered when looking for the best split.

3.3.2. Grid Search Cross-Validation:

Grid search is a hyperparameter tuning method that will methodically build and evaluate a model for each combination of the hyperparameters specified in a grid [10]. This method exhaustively searches through the hyperparameter grid, performing a 5-fold cross-validation for each combination of hyperparameter values. In 5-fold cross-validation, the training set is split into 5 subsets, and the model is trained 5 times, each time using 4 of the subsets as the training set and the remaining subset as the validation set. The final model's performance is the average performance across the 5 iterations. This cross-validation method helps ensure that the model's performance is robust and not overly reliant on a particular arrangement of the training and validation sets.

The best hyperparameters are selected based on the performance on the validation set, as quantified by the negative mean squared error (neg_mean_squared_error). The best_params_ attribute of the grid search object contains these hyperparameters. Finally, these parameters are used to train a final Random Forest model, which is evaluated on the test set to get an unbiased estimate of its predictive performance. In conclusion, with careful hyperparameter tuning via Grid Search Cross-Validation, the Random Forest regression model saw an improvement in performance by ensuring robustness through cross-validation. The Final code used to generate and test the Random Forest Model is available in the appendix.

4. Results and Analysis

The following table delineates the RMSE values for each model at each stage of feature engineering, serving as a quantitative measure of the models' performances. Subsequent stages incorporated specific feature engineering processes, carried out sequentially. At each stage, a hyperparameter optimization – either grid or random search – was used to ensure the models were being examined at their ideal state.

Table 1: Showing the performance of the models at various stages of feature selection.

Step	Description	Linear	KNN	Random Forest	Ridge	Lasso
1	No feature engineering	38062.08	47679.64	27355.38	25657.45	28164.97
2	Fixing skewness	0.1496	0.2626	0.1478	0.1236	0.1354
3	Filtering high-NaN values	0.1464	0.2629	0.1495	0.1207	0.1360
4	Adding features	212069.55	0.2596	0.1509	0.1206	0.1362
5	Removing legacy features and Filtering highly dominant features	0.1314	0.2594	0.1473	0.1204	0.1250
6	Dropping weak-correlated categories	0.1280	0.2471	0.1485	0.1275	0.1272
7	Dropping weak-correlated data	0.1272	0.2398	0.1487	0.1266	0.1265
8	Dropping multicollinear features	0.1270	0.2427	0.1468	0.1263	0.1263

4.1. Analysis

Table 1 shows how each regression technique reacted to each step of the feature engineering process. Additional results have been added in the Appendix. The initial 'no feature engineering' stage established the baseline performance of the models. At this stage, the Ridge and Random Forest models displayed the best performance. On the contrary, the KNN model, which has a more simplistic approach to data interpretation, performed the worst. The disparity in these results underlined the importance of matching the complexity of the model with the complexity and the nature of the problem space.

As the skewness in the dataset was fixed, all models saw an improvement, indicating that transformations catering to the underlying statistical assumptions of the models can be instrumental in improving model performance.

The subsequent steps involving filtering high-NaN features, dominant categories, and weakly correlated categories progressively stripped the dataset of potentially misleading or irrelevant features. Each step was designed to reduce noise in the data and help models focus on the most significant features. The slight improvements in RMSE across models, excluding a minor increase in KNN after removing high-NaN features, supported the efficacy of these steps.

An intriguing outcome was observed during the 'feature addition and removal of legacy features step'. While all models, excluding KNN, benefited from this stage, the Linear Regression model displayed a significant spike in RMSE. This highlighted the model's vulnerability to increased feature dimensionality and multicollinearity, potentially due to overfitting. This observation reiterated the careful balance required in feature engineering to optimize a model's performance.

During feature engineering, Ridge and Lasso Regression models displayed greater stability than basic Linear Regression, demonstrating their robustness to multicollinear, high dimensional data due to its inherent regularization properties. Random Forest, however, maintained consistently strong performance across all stages, demonstrating the robustness inherent to its ensemble nature. Despite the slightly better results and lower computational cost of Linear, Ridge, and Lasso Regression models, Random Forest's performance suggests that it may be better suited for complex real-world datasets.

Overall, the entire process from raw, unprocessed data to refined data undergoing various regression techniques gave us valuable insights. The iterative process showcased the interplay between feature engineering and model performance, stressing the need for an integrated understanding of both. It further emphasized the importance of selecting an appropriate model based on its inherent characteristics and the nature of the data at hand, to effectively leverage its strengths and mitigate its weaknesses. The experiment underscored the dynamism and adaptability required in predictive modeling, from tailoring the model to interpreting the results.

5. Submission - The Submission can be seen in figures 3-4 in the appendix.

6. Conclusion

This project emphasized key learnings for real-world predictive machine learning applications. Primarily, it highlighted the crucial role of feature engineering in model performance, impacting the accuracy of predictions significantly. It also underscored the importance of understanding each regression model's strengths and limitations, particularly illustrating overfitting and multicollinearity effects on Linear Regression compared to the resilience of Ridge and Lasso regressions with their regularization capabilities. The project further showcased the benefits of model blending, mitigating individual model shortcomings and reducing overfitting risk. Conclusively, the project affirmed the importance of iterative model building, meticulous feature engineering, and model characteristic comprehension for successful real-world machine learning applications.

Additionally, the individual work exploring data exploration and Random Forest regression was an illuminating experience and I believe it provided me a comprehensive view of the machine learning project pipeline, illuminating the iterative and dynamic nature of the field. The group nature of the project also taught me about the collaborative nature of machine learning, and how to best adapt to situations in which interoperability and dependencies are present.

References

[1] “House Prices - Advanced Regression Techniques | Kaggle,” *Kaggle.com*, 2023. https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview/description (accessed Jun. 02, 2023).

[2]“Root Mean Square Error (RMSE),” *C3 AI*, Sep. 28, 2021. https://c3.ai/glossary/data-science/root-mean-square-error-rmse/#:~:text=What%20is%20Root%20Mean%20Square,true%20values%20using%20Euclidean%20distance. (accessed Jun. 02, 2023).

[3] A. Bhandari, “Multicollinearity | Causes, Effects and Detection Using VIF (Updated 2023),” Analytics Vidhya, Mar. 19, 2020. https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/#:~:text=Multicollinearity%20occurs%20when%20two%20or%20more%20independent%20variables%20have%20a,variable%20on%20the%20dependent%20variable. (accessed Jun. 02, 2023).

[4] “Correlation matrix : A quick start guide to analyze, format and visualize a correlation matrix using R software - Easy Guides - Wiki - STHDA,” Sthda.com, 2020. http://www.sthda.com/english/wiki/correlation-matrix-a-quick-start-guide-to-analyze-format-and-visualize-a-correlation-matrix-using-r-software (accessed Jun. 02, 2023).

[5] Rashmi Manwani, “Lasso and Ridge Regularization - A Rescuer From Overfitting,” *Analytics Vidhya*, Sep. 15, 2021. https://www.analyticsvidhya.com/blog/2021/09/lasso-and-ridge-regularization-a-rescuer-from-overfitting/ (accessed Jun. 02, 2023).

[6] S. Bhattacharyya, “Ridge and Lasso Regression: L1 and L2 Regularization,” *Medium*, Sep. 26, 2018. https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b (accessed Jun. 02, 2023).

[7] T. Srivastava, “A Complete Guide to K-Nearest Neighbors (Updated 2023),” *Analytics Vidhya*, Mar. 25, 2018. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/ (accessed Jun. 02, 2023).

[8] Chaya, “Random Forest Regression - Level Up Coding,” *Medium*, Jun. 08, 2020. https://levelup.gitconnected.com/random-forest-regression-209c0f354c84 (accessed Jun. 02, 2023).

[9] P. Vadapalli, “Random Forest Hyperparameter Tuning: Processes Explained with Coding | upGrad blog,” *upGrad blog*, Sep. 23, 2022. https://www.upgrad.com/blog/random-forest-hyperparameter-tuning/ (accessed Jun. 02, 2023).

[10] R. Shah, “Tune Hyperparameters with GridSearchCV,” *Analytics Vidhya*, Jun. 23, 2021. https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/ (accessed Jun. 02, 2023).

Appendix

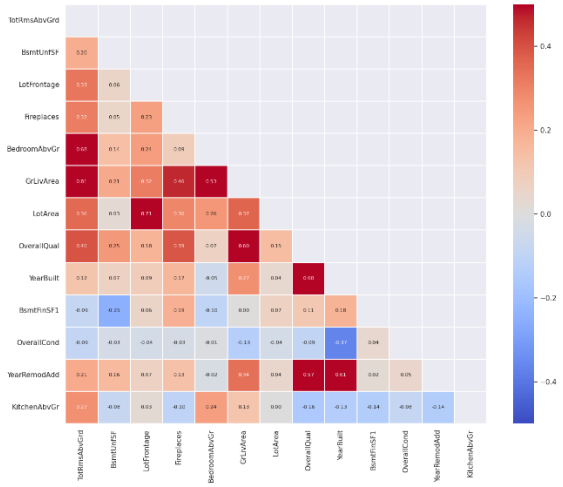

Figure 1: Showing the Pearson Correlation Matrix.

Snippet of code that creates the Random Forest Regression Model:

```
from sklearn.ensemble import RandomForestRegressor

def random_forest_regression(X_train, X_test, t_train, t_test):

    rf = RandomForestRegressor(random_state=42)

    param_grid = {
        'n_estimators': [50, 100],
        'max_depth': [None, 20],
        'min_samples_split': [2, 5],
        'max_features': ['auto']
    }

    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=
5, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

    grid_search.fit(X_train, t_train)

    best_params = grid_search.best_params_

    best_rf = RandomForestRegressor(**best_params)
    best_rf.fit(X_train, t_train)

    r_sq = best_rf.score(X_train, t_train)
    print(f"coefficient of determination: {r_sq}")

    t_pred = best_rf.predict(X_test)

    # Calculate the RMSE
    RMSE = np.sqrt(np.mean((t_pred - t_test) ** 2))
    print(f"RMSE: {RMSE}")
    return RMSE
```

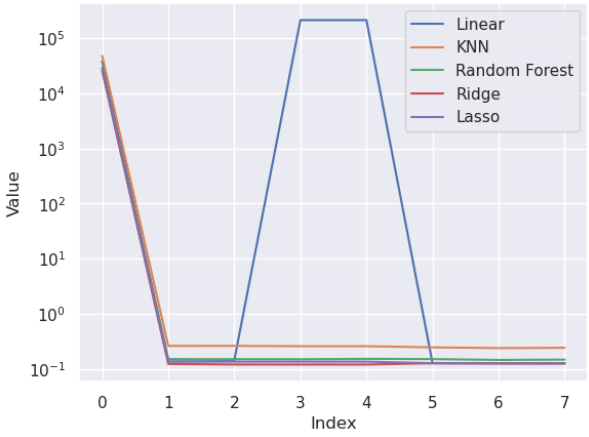

Figure 2: Showing the performance (RMSE) of each model at each stage of feature engineering.

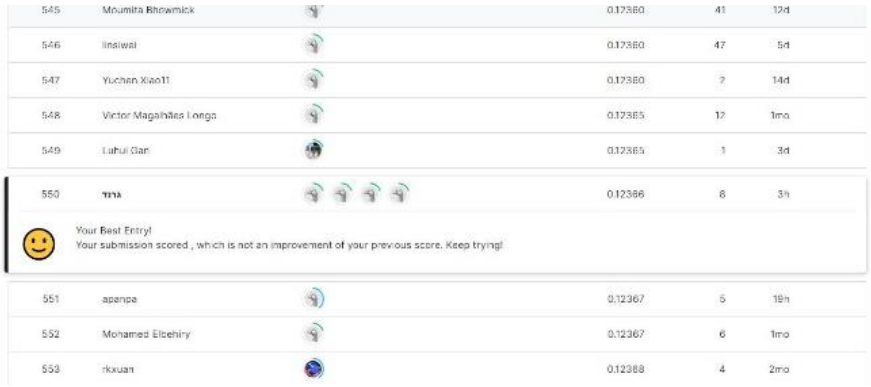

Figure 3: Showing the project submission on kaggle.

Figure 4: Showing the performance of the submissions.