

**School of InfoComm Technology**

**Machine Learning**

Diploma in Data Science (DS)

Diploma in Information Technology (IT)

October 2023 Semester

**INDIVIDUAL ASSIGNMENT 2**

(40% of Machine Learning Module)

**Deadline for Submission:**

**Presentation: 28th Jan 2024 (Sunday), 2359 Hours**

**Report: 8th Feb 2024 (Thursday), 2359 Hours**

|  |  |  |
| --- | --- | --- |
| Student Name | : | Syed Muhammad Nuruddin Aljunied |
| Student Number | : | S10244211B |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 15th Feb 2024, 23:59.

Table of Contents

[1. Introduction 4](#_Toc158293028)

[1.1 Classification Problems 4](#_Toc158293029)

[1.2 Regression Problems 4](#_Toc158293030)

[2. HR Analytics 5](#_Toc158293031)

[2.1 Problem Statement 5](#_Toc158293032)

[2.2.1 Data Exploration 5](#_Toc158293033)

[2.2 Data Cleaning and Transformations 5](#_Toc158293034)

[2.2.2 Missing Values 6](#_Toc158293035)

[2.2.3 Categorical Encoding 6](#_Toc158293036)

[2.2.4 Numerical Transformations 7](#_Toc158293037)

[2.2.5 Feature Scaling 7](#_Toc158293038)

[2.2.6 Sampling 8](#_Toc158293039)

[2.2.7 Correlation Analysis 8](#_Toc158293040)

[2.3 Machine Learning Modelling 9](#_Toc158293041)

[2.4 Process of Optimizing Models 9](#_Toc158293042)

[2.4.1 Inspect Hyperparameters 9](#_Toc158293043)

[2.4.2 Utilizing GridSearchCV 9](#_Toc158293044)

[2.4.3 Evaluating ‘Optimal’ Parameters 10](#_Toc158293045)

[2.5 Model #1 - Random Forest Classifier 10](#_Toc158293046)

[2.5.1 Build Base Model 10](#_Toc158293047)

[2.5.2 Inspecting Hyperparameters 11](#_Toc158293048)

[2.5.3 Improve the Model 13](#_Toc158293049)

[2.5.4 Final Model Evaluation 15](#_Toc158293050)

[2.6 Model #2 – Multilayer Perceptron 16](#_Toc158293051)

[2.6.1 Build Base Model 16](#_Toc158293052)

[2.6.2 Inspecting Hyperparameters 17](#_Toc158293053)

[2.6.3 Improve the Model 19](#_Toc158293054)

[2.6.4 Final Model Evaluation 22](#_Toc158293055)

[2.7 Model #3 – ADA Boost 23](#_Toc158293056)

[2.7.1 Build Base Model 23](#_Toc158293057)

[2.7.2 Inspect Hyperparameters 24](#_Toc158293058)

[2.7.3 Improve the Model 25](#_Toc158293059)

[2.7.4 Final Model Evaluation 27](#_Toc158293060)

[2.8 Summary 28](#_Toc158293061)

[2.8.1 Model Selection 28](#_Toc158293062)

[2.8.2 ADA Boost vs. Random Forest Regressor 29](#_Toc158293063)

[3. Airbnb 30](#_Toc158293064)

[3.1 Problem Statement 30](#_Toc158293065)

[3.2 Data Cleansing and Transformations 30](#_Toc158293066)

[3.2.1 Data Exploration 30](#_Toc158293067)

[3.2.2 Missing Values 31](#_Toc158293068)

[3.2.3 Improvement Step: Natural Language Processing 31](#_Toc158293069)

[3.2.4 Categorical Encoding 31](#_Toc158293070)

[3.2.5 Outliers in Target Variable 32](#_Toc158293071)

[3.2.5 Numerical Transformations 32](#_Toc158293072)

[3.2.6 Feature Scaling 32](#_Toc158293073)

[3.2.7 Correlation Analysis 32](#_Toc158293074)

[3.3 Machine Learning Modelling 33](#_Toc158293075)

[3.4 Process of Optimizing Models 33](#_Toc158293076)

[3.4.1 Inspect Hyperparameters 33](#_Toc158293077)

[3.4.2 Utilizing GridSearchCV 33](#_Toc158293078)

[3.4.3 Evaluating ‘Optimal’ Parameters 34](#_Toc158293079)

[3.5 Model #1 – Random Forest Regressor 34](#_Toc158293080)

[3.5.1 Build Base Model 34](#_Toc158293081)

[3.5.2 Inspecting Hyperparameters 35](#_Toc158293082)

[3.5.3 Improve the Model 37](#_Toc158293083)

[3.5.4 Final Model Evaluation 40](#_Toc158293084)

[3.6 Model #2 – XG Boost 41](#_Toc158293085)

[3.6.1 Build Base Model 41](#_Toc158293086)

[3.6.2 Inspecting Hyperparameters 42](#_Toc158293087)

[3.6.3 Improve the Model 45](#_Toc158293088)

[3.6.4 Final Model Evaluation 48](#_Toc158293089)

[3.7 Model #3 – SVR 49](#_Toc158293090)

[3.7.1 Build Base Model 49](#_Toc158293091)

[3.7.2 Inspecting Hyperparameters 50](#_Toc158293092)

[3.7.3 Improve the Model 52](#_Toc158293093)

[3.7.4 Final Model Evaluation 57](#_Toc158293094)

[3.8 Summary 58](#_Toc158293095)

[3.8.1 Model Selection 58](#_Toc158293096)

[3.8.2 Random Forest Regressor vs. XG Boost 59](#_Toc158293097)

[4. Conclusion 60](#_Toc158293098)

[5. Reflection 61](#_Toc158293099)

[5.1 Improvements to ML Solution 61](#_Toc158293100)

[5.2 What could have been learnt better? 62](#_Toc158293101)

[6. Discussion Panel 63](#_Toc158293102)

[6.1 Questions 63](#_Toc158293103)

* 1. [Answers 64](#_Toc158293104)

# Introduction

In data analysis and decision making, Machine Learning (ML) has been a transformative force. It involves the development of algorithms that enable computers to learn patterns and make predictions without explicit programming. This field has revolutionized various industries by providing solutions by facilitating data-drive decision-making. In the context of ML, there are two fundamental types of problems: regression and classification.

## 1.1 Classification Problems

Classification involves categorizing input data into predefined classes or labels. This task is like sorting distinct groups based on their features. The algorithms aim to identify patterns within the data that can be used to assign new and unseen instances to one of the predefined classes. For example, classification can be used to predict email spam detection – where the algorithm categorizes incoming emails as either spam or not spam, based on certain features such as keywords and content. The challenge in classification lies in whether the models can generalize unseen data well and make accurate predictions in a wide range of scenarios.

## 1.2 Regression Problems

Regression problems in ML involve predicting a continuous output or value based on features and characteristics. Examples of such problems include predicting house prices, and forecasting stock prices using historical data. The goal in regression is to establish a relationship between input variables and the target variable (which is continuous), allowing for accurate predictions of unseen data. For example, Linear Regression assumes a linear relationship between features and the target variable, while more complex models may capture nonlinear relationships. The challenge in regression lies in the model overfitting, where the model becomes too tailored to training data and performs poorly on unseen data. Finding the balance between complexity and generalization is what creates robust regression models that can provide meaningful predictions.

# 2. HR Analytics

## 2.1 Problem Statement

Analytics plays a key role in bringing efficiency and results in the workplace, especially in the human resources department. Collection of data and processing of employee information is generally manual, presenting bottlenecks in efficiency in these HR departments, in aspects such as hiring and promotion of employees. Particularly in the context of promoting employees, **classifying** whether they will be promoted or not based on certain features they have is a challenging process of selection. To counter this classification problem, machine learning can be capitalized to better these tasks.

## 2.2 Data Cleaning and Transformations

Before Machine Learning Modelling, the data first needs to be processed and transformed in accordance to the requirements of robust machine learning models. Using the dataset hr\_data.csv, which contains information on employees such as education, past performance and key metrics, these features can be used to aid prediction.

### **2.2.1 Data Exploration** In data exploration, it was observed that a significant number of employees were not being promoted, indicating a potential class imbalance issue.

A blue and white rectangles

Description automatically generated

*Fig 1. Class Distribution*

### **2.2.2 Missing Values** In the dataset, rows with null values were removed to avoid making assumptions, for the **‘education’** variable. Null imputation was implemented for the **‘previous\_year\_rating’** variable, using the mode to maintain the distribution and ensure unbiased outcomes.

A screenshot of a computer code

Description automatically generated  
*Fig 1.1 After Null Imputation/ Removal*

### **2.2.3 Categorical Encoding** Categorical variables were encoded based on cardinality and characteristics. **One-hot encoding** was applied to features with low cardinality, and presented no order. **Ordinal Encoding** was used for ‘education’ which exhibited a hierarchy. To address the high cardinality of the ‘region’ variable, **Counts Frequency Encoding** was implemented.

A screenshot of a computer

Description automatically generated

*Fig 1.2 Columns encoded*

### **2.2.4 Numerical Transformations**

Numerical Transformations were done on continuous variables, which helped to reduce the effect of outliers. Logarithmic Transformation was applied to ‘age’ variable to normalize and mitigate outliers, as well as Yeo Johnson Transformations on the ‘length\_of\_service’ variable for the same desired outcome.

A group of graphs and diagrams

Description automatically generatedA collage of graphs

Description automatically generated

*Fig 1.3 & 1.4 Numerical Transformations Applied*

### **2.2.5 Feature Scaling**

Robust Scaling was applied to the variables with different scales, also handling outliers.

A screenshot of a graph

Description automatically generated

*Fig 1.5 Scaling of variables*

### **2.2.6 Sampling**

To address the class imbalance identified earlier, down-sampling was done, balancing the ratio to 1:1.

A screenshot of a computer code

Description automatically generated

*Fig 1.6 Class Distribution*

### **2.2.7 Correlation Analysis**

The ‘employee\_id’ feature was dropped to prevent it from dominating other features, with other columns preserved which may be dropped later. This concludes the data cleaning and transformation process.

A screenshot of a computer screen

Description automatically generated

*Fig 1.7 Correlation of Features*

## 2.3 Machine Learning Modelling

In the context of classification, various machine learning models exist, each with its strengths and different level of complexities. However, for this specific task, the focus will be narrowed down to these three models: **Random Forest Classifier,** **Multi-Layer Perceptron (Neural Network)** and **Adaptive Boosting Classifier**.

## 2.4 Process of Optimizing Models

In most machine learning models, **hyperparameter tuning** is a crucial step in the model development process. It involves optimizing the settings of a model that are not learned from the training data but are set prior to the training process. Certain hyperparameters can significantly impact the performance of the model, and finding values that are just right can lead to improved accuracy and generalization, leading to better overall performance. The segments below outline the general process that will be adopted in optimizing the models.

### **2.4.1 Inspect Hyperparameters**

Before tuning the hyperparameters, the relationships between certain hyperparameter values and model performances are inspected. This analysis brings about a foundation for making decisions during the tuning process. It also helps to avoid high computational costs associated with exhaustively tuning hyperparameter across an entire range, by systematically evaluating a range of hyperparameter configurations where the model performs at its best. Hyperparameters that will be tuned are chosen based on whether they have significant impact on both accuracy and overfitting.

### **2.4.2 Utilizing GridSearchCV**

Grid Search Cross-Validation is a hyperparameter tuning technique used in machine learning to search through a predefined set of values. It utilizes a grid of hyperparameter values to explore, and then performs cross-validation at each grid. It exhaustively tests all possible combinations of hyperparameters, evaluating the model at each cross-validation point. The best combination of hyperparameters is chosen at the end, as the optimal set. As mentioned, the hyperparameters are inspected to find ‘best’ range, and this range of values for each parameter is defined in the grid for Grid Search.

### **2.4.3 Evaluating ‘Optimal’ Parameters**

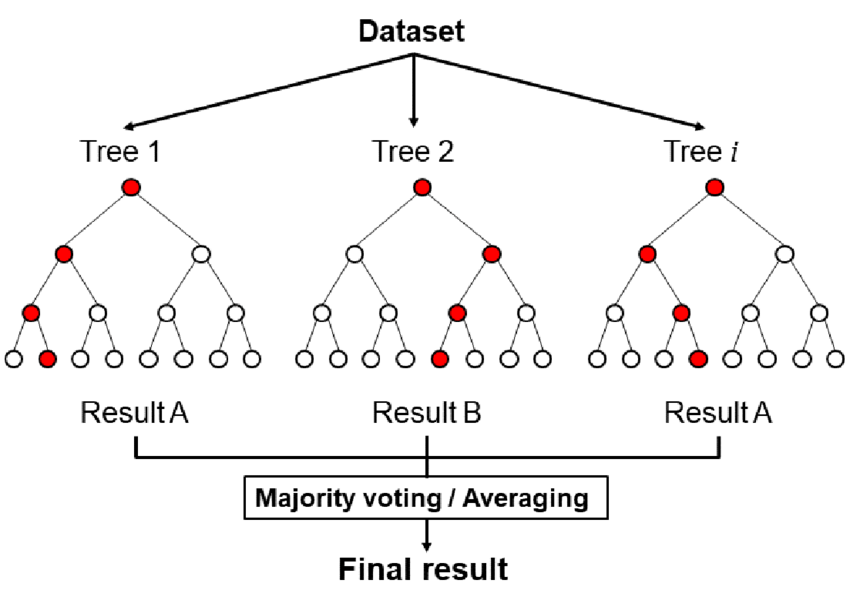
After Grid Search, the model performance is inspected each time. For example, if the model

is overfitted/ underfitted, the set of parameters is changed and placed into a new grid, and this process is repeated until the desired outcome of getting the model to have high accuracy and little to no overfitting is achieved.

## 2.5 Model #1 - Random Forest Classifier

An ensemble machine learning algorithm, the Random Forest Classifier operates by constructing multiple decision trees in training and outputs the class that is the mode of the classes of the individual trees. Each tree in the forest is trained on a random subset of training data, each making independent predictions. The final prediction is determined by the aggregation of the results of all trees.

Reading Reference: [sklearn.ensemble.RandomForestClassifier — scikit-learn 1.4.0 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)



### **2.5.1 Build Base Model**

**A screenshot of a computer

Description automatically generated**

*Fig 1.8 RFClassifier Base Model Accuracies*

Evaluating the Random Forest Classifier’s base model accuracy (meaning the model is built with no preset parameters), the train and test accuracies are relatively high, but show a large difference, indicating that the model is highly biased (overfitted), as it performs relatively poorly on the test set as compared to the train set. The model is shown to have a 100% accuracy on the train set, indicating that it captured highly detailed patterns in the training set, but unable to generalize well to new data in the test set.

A screenshot of a computer

Description automatically generated

*Fig 1.9 Test Report*

From the test report above, the F1 score, which balances the precision and recall (sensitivity), are relatively close (0.78 and 0.80 respectively). This suggests that the model is performing quite well for both classes.

### **2.5.2 Inspecting Hyperparameters**

In the Random Forest Classifier, it has key hyperparameters that control the complexity of the model. This segment covers hyperparameters that will be focused on for inspection - n\_estimators, criterion, max\_depth, min\_samples\_split and min\_samples \_leaf.

**n\_estimators** - determines the number of decision trees in the forest, increasing the number of trees generally increases the performance of the model, but too many trees may result in making the model too detailed.

A whiteboard with blue and orange dots

Description automatically generated

*Fig 2. estimators vs accuracies*

Plotted from default number of estimators (100) to 1000 estimators, the model score tends to be similar and plateaus throughout as the number of estimators increases.

**max\_depth** –defines the maximum depth of the tree in the forest, controlling it can help to prevent overfitting. For example, a shallow tree might not capture underlying patterns, but a very deep tree might lead to overfitting.

A graph of a line

Description automatically generated with medium confidence

*Fig 2.1 max depth vs accuracies*

As the depth of the tree increases, the accuracy of the train and test scores increase. However, it plateaus at around the range of 40-100, with the model being overfitted. Different ranges will be experimented on where the model performs its best.

**min\_samples\_split** – determines the number of samples required to split an internal node, with higher values leading to simpler tree structures, reducing overfitting.

A graph of a graph

Description automatically generated with medium confidence A graph of blue and orange lines

Description automatically generated

Fig 2.2 & 2.3 min samples split vs accuracies

As the number of samples increases, overfitting tends to decrease and the scores are close to each other. However, using higher samples also reduces the model score, hence this must be considered.

**Notable Optimal Ranges : (40-100)**

**min\_samples\_leaf** – sets the number of samples required to be in a leaf node, increasing it can lead to a more generalized model.

A graph of a graph with blue and orange lines

Description automatically generated

*Fig 2.4 min samples leaf vs accuracies*

Similarly here, as the number of samples for it to become a leaf increases, it generally reduced the biasness of the model.

**Notable Optimal Ranges : (10-20)**

### **2.5.3 Improve the Model**

Based on the information gathered above on the different parameters, different ranges of values are experimented on in different grids for Grid Search, to find the best parameters.

**1st iteration**

**A computer code with red text

Description automatically generated**

*Fig 2.5 1st grid*

For the first grid, a lower set of estimators is used (40-70), as previously it was identified that the scores plateau even as the number of estimators increased. For the other parameters, the ranges were set at where the train and test scores were shown to be least overfitted but exhibited high scores.

**Best Parameters – Evaluation**



*Fig 2.6 best parameters*

A black text on a white background

Description automatically generated

*Fig 2.7 model scores*

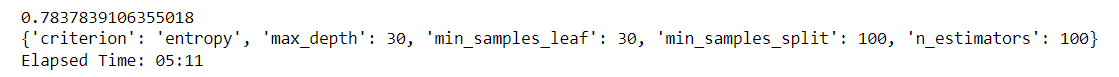
From the model built based on the best parameters obtained from Grid Search, it is slightly overfitted (0.81 train, 0.78 test) , but the overfitting is reduced.

**2nd iteration**

**A white rectangular object with red text

Description automatically generated**

*Fig 2.8 2nd Grid*

From the previous grid results, it took the highest for n\_estimators and max\_depth. Hence for the second grid, a higher range of values for all parameters was experimented on for Grid Search. With higher values for min\_samples\_split and min\_samples\_leaf, it can potentially reduce the overfitting of the model as well. 

*Fig 2.9 best params*

*A close up of a text

Description automatically generated*

*Fig 3. model score*

With the given best hyperparameters, the model train and test accuracies are relatively close to each other, with reduced overfitting. As such, this model is used for our final model.

### **2.5.4 Final Model Evaluation**

A screenshot of a computer

Description automatically generated

*Fig 3.1 Model Scores and Classification Report*

**Improvement from base model**

From the base model, the scores were now less overfitted, (previously 1 train, and 0.79 test) to now 0.79 for train and 0.78 for test, making the model less biased.

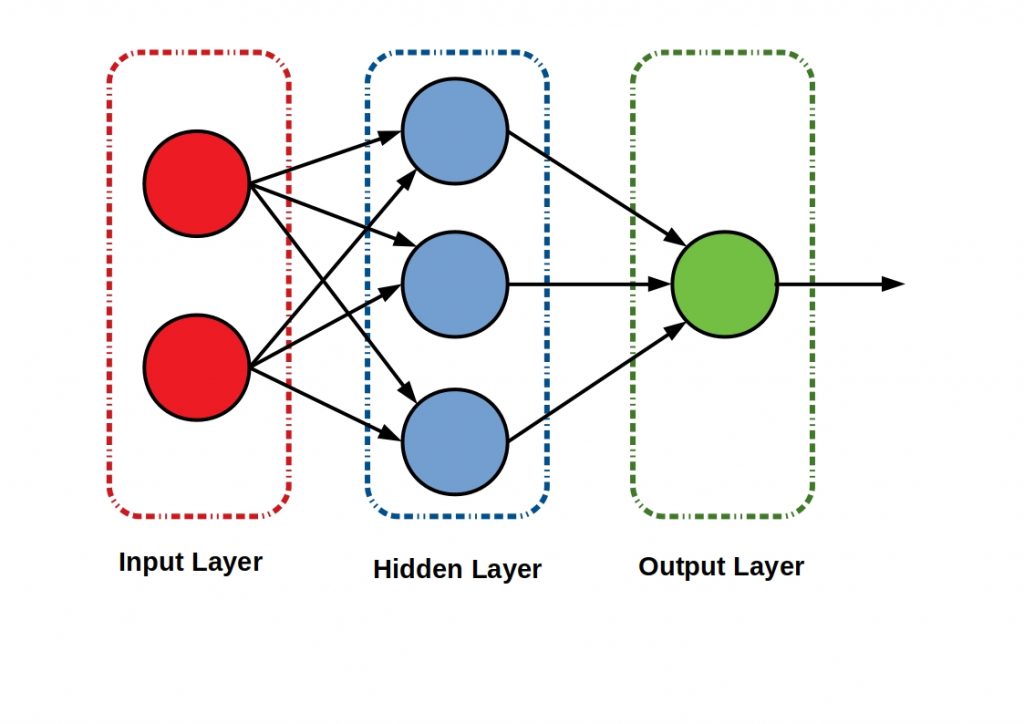
**Classification report**

In context of the classification report,the model demonstrates a reasonably good performance on train and test sets, with relatively low overfitting. The precision, recall, and F1-score for both classes show a relatively balanced performance, although there is a slight imbalance favoring class 1 (promoted) over class 2 (not promoted). Class 1 has higher recall, meaning that it is better at capturing whether an employee is promoted more than when they are not. Further tuning was experimented on for this model, but the scores did not improve/ plateaued even at trying many combinations, hence this model made final.

## 2.6 Model #2 – Multilayer Perceptron

A type of artificial neural network, a multilayer perceptron consists of multiple layers of interconnected nodes or neurons, organized into an input layer, hidden layers, and output layers. Each connection between nodes has its own weight, and the network learns by adjusting these weights during training. The input data is passed through the network’s layers from the input layer, hidden layer and then output layer, to produce a prediction or classification.

Reading reference: [sklearn.neural\_network.MLPClassifier — scikit-learn 1.4.0 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)



### **2.6.1 Build Base Model**

A screenshot of a computer code

Description automatically generated

*Fig 3.2 Base MLP Model*

Evaluating the base model for the Multilayer Perceptron, the model scores are in a satisfactory range, but can still be improved with tuning. However, the model shows no signs of overfitting, which indicates that the model is generalizing similarly in both train and test sets.

A screenshot of a graph

Description automatically generated

*Fig 3.3 Classification Reports*

From the f1 scores of the model, there is some variation between them (while not too different) for train and test, where it is favoring class 0 more than class 1.

### **2.6.2 Inspecting Hyperparameters**

In the MLP model, key hyperparameters are chosen, based on how they affect the complexity of the model. These hyperparameters will be focused on in this section – hidden layer sizes and max\_iter.

**hidden layer sizes** – determines the number of neurons in the hidden layers of the nueral network, influencing the capacity of capturing complex patterns. Too few neurons may lead to underfitting, where the model is unable to capture complex patterns, whilst too many neurons might lead to overfitting, making the model memorize training data but unable to generalize to unseen data.

A graph with blue and orange lines

Description automatically generated A graph with orange and blue lines

Description automatically generated

Fig 3.4 & 3.5 Layer Sizes Against Accuracy

As shown in the graphs above, as the number of hidden layers increases, the model scores tend to decrease, but this relationship is not consistent as at some points, the scores see improvements as show less overfitting.

**Notable Optimal Ranges : (10-40)**

**max\_iter –** the maximum number of iterations for the solver to converge, too few may result in underfitting, making it unable to converge to an optimal solution, whiele too many might lead to overfitting making the model memorize noise.

A graph with blue and orange dots

Description automatically generated A white board with blue and orange lines

Description automatically generated

Fig 3.6 & 3.7 Max Iter (Epochs) Against Accuracy

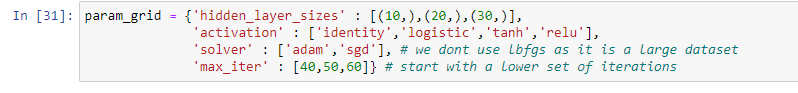
As the number of iterations increases, the model score increases, but plateaus at even a higher range of iterations.

**Notable Optimal Ranges : (40-60)**

### **2.6.3 Improve the Model**

From the information gathered above, the model is improved based on ranges where it is optimal first.

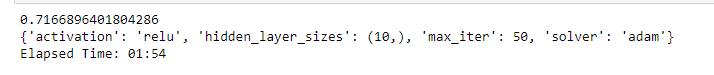
**1st iteration**

****

*Fig 3.8 1st grid*

For the first grid, a lower set of layer sizes and iterations is used for the model, where it was identified to be at its best for the base model. Different activation functions are experimented on as well, however as for solvers, only the ‘adam’ and sgd’ solver is utilized.

**Best Parameters – Evaluation**



*Fig 3.9 Best Parameters from first grid*

*A math equation with numbers and letters

Description automatically generated with medium confidence*

*Fig 4. Model Built with best params from first grid*

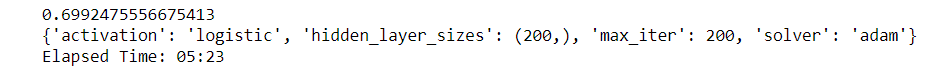
Even using the ranges identified where the model was best for both hidden layer sizes (10-30) and max iterations (40-60), the model scores did not increase significantly from the base model and showed signs of overfitting.

**2nd iteration**

****

*Fig 4.1 2nd grid*

Deviating from results from the first grid, now higher values for layer sizes and iterations are experimented on. This is with the desired effect of increasing model complexity and capacity.



*Fig 4.2 best params from 2nd grid*

A screenshot of a computer

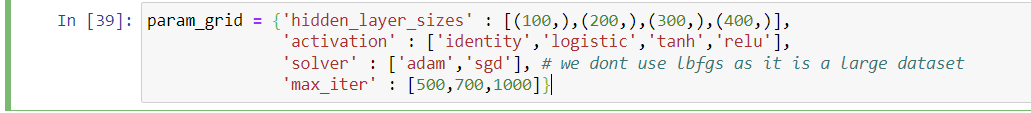
Description automatically generated

*Fig 4.3 Model Built with best params from 2nd grid*

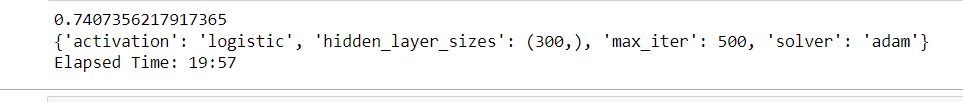
From the best parameters from the second grid, the model increased in score, and is only slightly overfitted.

**3rd iteration**

From the second grid, the highest layer sizes was taken as the best parameter. For this grid, a higher range of layer sizes is experimented on, to see if there would be any increase in train and test scores.



*Fig 4.4 3rd grid*

**

*Fig 4.5 best params from 3rd grid*

A close up of a sign

Description automatically generated

*Fig 4.6 Model Built with best params from 3rd grid*

From the model built, the train and test scores decreased from the previous grid.

### **2.6.4 Final Model Evaluation**

Out of all the grids, the one that produced the best scores was the second grid.

A screenshot of a computer

Description automatically generated

*Fig 4.7 Best MLP model*

**Improvement from base model**

From the base model, it improved from a 0.66 train and 0.66 test accuracy to 0.75 training and 0.74 testing set, with little overfitting.

**Classification Report**

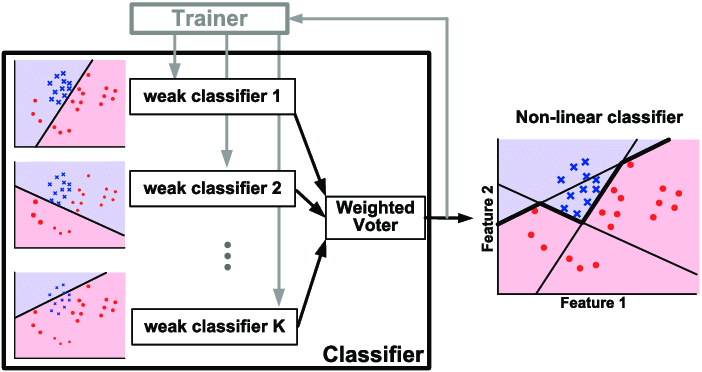
In the training set, the model achieved an overall F1 score of 0.76, indicating a good balance between precision and recall. Specifically, for class 0, the precision was 0.79, suggesting that 79% of instances predicted as class 0 were indeed correct, while recall was 0.70, indicating that the model captured 70% of the actual instances of class 0. For class 1, the precision was 0.73, demonstrating that 73% of instances predicted as class 1 were correct, and recall was 0.81, suggesting that the model effectively identified 81% of the actual instances of class 1. In the testing set, the model demonstrated similar performance trends with an overall F1 score of 0.75.

The precision and recall values for both classes were comparable between the training and testing sets, indicating that the model generalizes well to new, unseen data. These results suggest a well-balanced trade-off between precision and recall, indicative of a model that effectively classifies instances from both classes without favoring one at the expense of the other.

## 2.7 Model #3 – ADA Boost

AdaBoost, short for Adaptive Boosting, is an ensemble learning technique designed to enhance the performance of weak learners and create a robust, accurate predictive model. In the context of classification tasks, AdaBoost works by iteratively training a series of weak classifiers, assigning higher weights to instances that were misclassified in the previous iterations. In each round, the algorithm focuses on correcting the mistakes of the previous weak classifiers, thereby adapting its learning strategy. The final prediction is then made by combining the weighted votes of all weak classifiers.

Reading reference: [sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.4.0 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)



### **2.7.1 Build Base Model**

A screenshot of a computer code

Description automatically generated

*Fig 4.8 Base Model for ADA Boost*

Evaluating the base model for ADA Boost, the scores are relatively high, but is slightly overfitted with the training score being slightly higher than the test score, indicating that the model is biased.

A screenshot of a computer

Description automatically generated

*Fig 4.9 Test Report for Base ADA Boost*

For the F1 scores of the base model, the model performs equally for both train and test, at both 0.77 f1 scores, indicating that the model is performing quite well for both classes. In this segment, further improvements will be made to increase the scores of the model.

### **2.7.2 Inspect Hyperparameters**

Like the other models, ADA boost has parameters that affect the complexity of the model. These hyperparameters will be focused on – n\_estimators and the learning rate of the model.

**n\_estimators** – the number of weak learners to train iteratively, if there are too many estimators, it may lead to overfitting. The default weak learner for ADA boost is a decision tree.

A graph of blue and orange lines

Description automatically generated

*Fig 5. n\_estimators against model scores*

As the number of estimators increases, the model scores increase, but this pattern is not consistent, with some drops at a certain number of estimators. The model also becomes more overfit as there are more estimators.

**Notable Optimal Range : (200-400)**

**learning rate** – contribution of each weak learner to the final prediction, if there are more estimators, a smaller learning rate can be used.

A graph of a train set

Description automatically generated

*Fig 5.1 learning rate against model scores*

Plotted from 0 to 1, as the learning rate increases the model score increases, but this is also subject to the number of estimators later.

### **2.7.3 Improve the Model**

Based on the best ranges, the model is improved based on that first.

**1st iteration**

**A white rectangular sign with red text

Description automatically generated**

*Fig 5.2 1st grid*

For the first grid, the best range for estimators was used, with a learning rate from 0-1.

A black and white text

Description automatically generated

*Fig 5.3 Best Params for 1st grid*

*A white background with black text

Description automatically generated*

*Fig 5.4 Model built based on best params*

As compared to the base model, the scores have increased, but the model is now overfitted, with the train and test scores having a difference of 0.3.

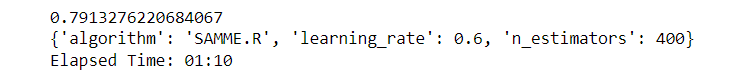
**2nd iteration**

**A white rectangular sign with red text

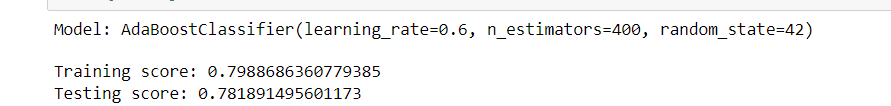
Description automatically generated**

*Fig 5.5 2nd grid*

For the second grid, a lower learning rate is used. With a relatively high number of estimators, a lower learning rate is experimented on with the aim of reducing overfitting of the model.



*Fig 5.6 best params from 2nd grid*

****

*Fig 5.7 model built*

From the results of the model built based off the second grid’s best parameters, the model did indeed have reduced overfitting, with the train and test scores being close, and the testing score increased as well from the previous grid. With reduced overfitting and relatively high scores, this can be used as the final model.

### **2.7.4 Final Model Evaluation**

A screenshot of a computer

Description automatically generated

*Fig 5.8 Best ADA Boost Model*

**Improvement from base model**

With the tuned parameters, the scores of train and test had increased from the base model, previously 0.78 train and 0.76 test, to 0.79 train and 0.78 test scores.

**Classification Report**

For both the training and testing sets, the F1 scores provide valuable insights into the model's overall performance. In the training set, the F1 score is notably high for both Class 0 (0.79) and Class 1 (0.80), indicating a well-balanced trade-off between precision and recall. This suggests that the model effectively captures true positives while minimizing false positives and false negatives.

Similarly, on the testing set, the F1 scores remain consistent, with Class 0 at 77% and Class 1 at 78%. This consistency underscores the model's ability to maintain a balanced performance across positive and negative instances in the unseen data.

## 2.8 Summary

### **2.8.1 Model Selection**

To evaluate which of the ‘best’ models built previously, cross validation is used. Cross validation involves training the model on a subset of data, and validating it on the remaining portions, iteratively rotating subsets and validating it on the remaining portions. Here, **K-folds** cross validation is used to evaluate the models, where the dataset is divided ‘k’ times and validated ‘k’ times each using a different fold for validation.

A screenshot of a computer code

Description automatically generated

*Fig 5.9 Cross Val accuracy for all models*

Observing the cross-validation scores for all the ‘best’ models, the Random Forest Regressor and the ADA Boost Classifier perform more consistently, showing that the models are more reliable as compared to the MLP model, which is not so consistent at 5 folds.

### **2.8.2 ADA Boost vs. Random Forest Regressor**

To choose between the two as they have similar scores, the key metrics from the classification reports can be compared earlier.

**ADA Boost**

* *Training Accuracy: 0.79*
* *Testing Accuracy: 0.78*
* *F1 Score: 0.78/0.79 (Test)*

Achieves similar F1 scores, indicating a balanced trade-off between precision and recall.

**Random Forest**

* *Training Accuracy: 0.79*
* *Testing Accuracy: 0.78*
* *F1 Score: 0.75/0.81 (Test)*

Has a higher F1 score for class 1 (promoted) as compared to class 0 (not promoted)

In context of these metrics, the best model for companies to utilize is the **ADA Boost Classifier**, as it has a more balanced performance across both classes in the testing set. This is crucial in the context of promotion prediction, as a model that achieves high precision ensures that when it predicts promotions, it is accurate and reliable, avoiding unnecessary promotions for those who may not be deserving. Simultaneously, high recall (for both train and test) ensures that the model identifies a significant portion of actual promotions, minimizing the risk of overlooking deserving candidates.

# 3. Airbnb

## 3.1 Problem Statement

Airbnb is a platform for tourists to stay in many different countries. It presents a new personalized way to experience the world. The problem lies within accurately pricing listings, based on information such as the condition of listings, reviews etc. To counter this **regression** problem, machine learning can be utilized for better gauges of prices for hosts and users alike.

## 3.2 Data Cleansing and Transformations

Before machine learning modelling, the data needs to first be cleansed and transformed in accordance to the nature of machine learning models. From 2013-2019, the data file **listings.csv** contains information on hosts’, condition of listings, reviews etc. The problem lies within accurately priced listings based on this information, to provide better gauges for hosts and guests alike.

### **3.2.1 Data Exploration**

In the data exploration phase, it was found that most listings were in the ‘Central Region’, which was 79% of the dataset. The dataset was subset based on this region.

A pie chart with different colored triangles

Description automatically generated

*Fig 6. Distribution of the Regions*

### **3.2.2 Missing Values**

For the listings that had 0 reviews, the ‘last\_review’ and ‘reviews\_per\_month’ was null. For the last review, the datetime features were extracted and imputed as 0 as a null identifier, and for the ‘reviews\_per\_month’ it was replaced with 0 as there were no reviews.

A screenshot of a computer program

Description automatically generated

*Fig 6.1 Nulls imputed*

### **3.2.3 Improvement Step: Natural Language Processing**

To extract meaningful features from the listing names, TF-IDF was used. TF-IDF, or Term Frequency-Inverse Document Frequency, is a numerical representation used in natural language processing to evaluate the importance of a word in a document relative to a collection of documents. It assigns weight to each word based on its frequency in the document and its rarity across the entire dataset. This allows TF-IDF to highlight significant terms, emphasizing words that are distinct to a particular document while downplaying commonly occurring words. The 'spacious,' 'mrt,' and 'private' columns were retained due to their indicative nature, providing valuable insights and capturing relevant features that contribute significantly to predicting Airbnb prices based on the dataset characteristics.

A screenshot of a computer

Description automatically generated

*Fig 6.2 TF-IDF columns*

### **3.2.4 Categorical Encoding**

Since the data was subset, there were only 2 variables to encode. Counts Frequency Encoding was use for the ‘neighbourhood’ column, and Ordinal Encoding was used for the ‘room\_type’ column.

A screenshot of a computer

Description automatically generated

*Fig 6.3 after encoding*

### **3.2.5 Outliers in Target Variable**

Performed winsorisation, including the target variable. This was done to remove bias and provide more accuracy in predictions. It improved the target, but as for other variables the outliers were not removed but the scales were changed, reducing its effect.

### **3.2.5 Numerical Transformations**

Done to remove outliers and normalize variables. **Power transformation** for reviews\_per\_month and **Yeo Johnson** transformation for minimum\_nights and number\_of\_reviews.

A diagram of a graph

Description automatically generated with medium confidence

*Fig 6.4 after numerical transformations*

### **3.2.6 Feature Scaling**

Standardisation was done for the numerical columns to put the variables on the same scale, so that they do not dominate other features.

A screenshot of a computer screen

Description automatically generated

*Fig 6.5 after scaling*

### **3.2.7 Correlation Analysis**

Despite highly correlated features, there is still variation in the data. For example, higher availability rates may not necessarily indicate lower/higher prices. Most features were maintained, except for the listing\_id, listing\_name, and names of hosts (dropped prior).

## 3.3 Machine Learning Modelling

For regression problems, there are many algorithms offering different approaches to predicting continuous outcomes. However, for this task in specific, these models will be focused on: **Random Forest Regressor**, **XG Boost (eXtreme Gradient Boosting)** and **SVR** (Support Vector Regression).

## 3.4 Process of Optimizing Models

In most machine learning models, **hyperparameter tuning** is a crucial step in the model development process. It involves optimizing the settings of a model that are not learned from the training data but are set prior to the training process. Certain hyperparameters can significantly impact the performance of the model, and finding values that are just right can lead to improved accuracy and generalization, leading to better overall performance. The segments below outline the general process that will be adopted in optimizing the models.

### **3.4.1 Inspect Hyperparameters**

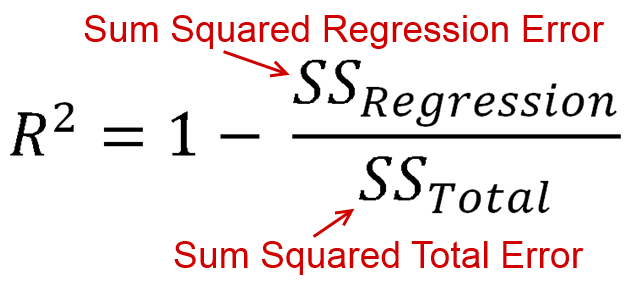
Before tuning the hyperparameters, the relationships between certain hyperparameter values and model performances are inspected. This analysis brings about a foundation for making decisions during the tuning process. It also helps to avoid high computational costs associated with exhaustively tuning hyperparameter across an entire range, by systematically evaluating a range of hyperparameter configurations where the model performs at its best. Hyperparameters that will be tuned are chosen based on whether they have significant impact on both accuracy and overfitting.

### **3.4.2 Utilizing GridSearchCV**

Grid Search Cross-Validation is a hyperparameter tuning technique used in machine learning to search through a predefined set of values. It utilizes a grid of hyperparameter values to explore, and then performs cross-validation at each grid. It exhaustively tests all possible combinations of hyperparameters, evaluating the model at each cross-validation point. The best combination of hyperparameters is chosen at the end, as the optimal set. As mentioned, the hyperparameters are inspected to find ‘best’ range, and this range of values for each parameter is defined in the grid for Grid Search.

### **3.4.3 Evaluating ‘Optimal’ Parameters**

After Grid Search, the model performance (using **R-squared** metric) is inspected each time. **R-squared** values range from 0 to 1, where 0 indicates that the model does not explain any variability in the target variable, and 1 signifies that the model perfectly explains all the variability. In practical terms, an R-squared value closer to 1 suggests that the model provides a good fit to the data, capturing a significant portion of the variability.



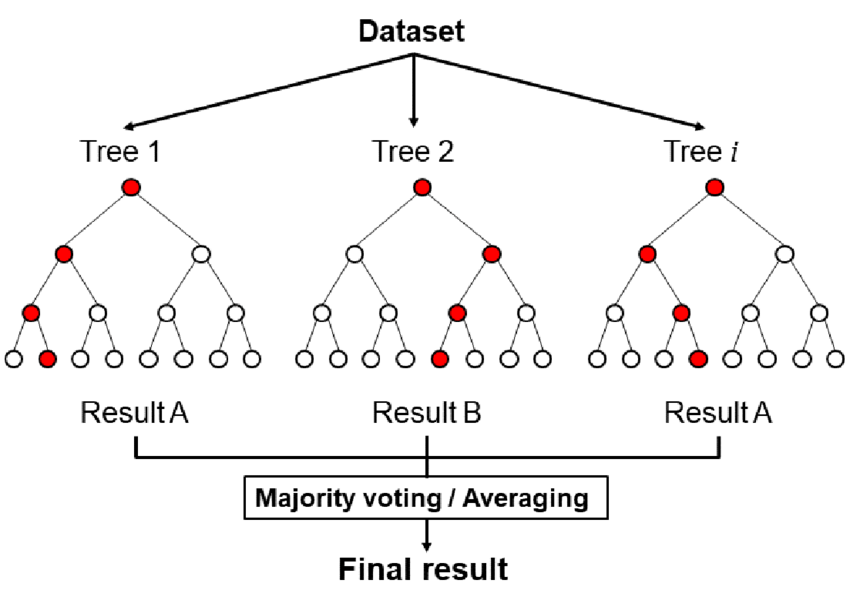
*Fig 6.6 r-squared formula*

If the model is overfitted/ underfitted, the set of parameters is changed and placed into a new grid, and this process is repeated until the desired outcome of getting the model of resolving overfitting is achieved.

## 3.5 Model #1 – Random Forest Regressor

The Random Forest Regressor is an ensemble learning algorithm used for regression tasks. It operates by constructing multiple decision trees during training and outputs the average prediction of the individual trees for regression problems. Each tree is trained on a random subset of the dataset, and the randomness is introduced by considering a random subset of features at each split. This randomness and the combination of multiple trees enable the Random Forest Regressor to capture complex relationships in the data, handle noise, and produce robust predictions for continuous target variables.

Reading reference : [sklearn.ensemble.RandomForestRegressor — scikit-learn 1.4.0 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)



### **3.5.1 Build Base Model**

A white background with black text

Description automatically generated

*Fig 6.7 RF base model r-square values*

Based on the r-square values for train and test, the model is performing relatively well, but is very overfitted (train at 0.94, and test at 0.6) and hence the parameters must be tuned to reduce it.

### **3.5.2 Inspecting Hyperparameters**

For the model, there are key parameters that affect the model’s complexity resulting in the effect of overfitting. These are the hyperparameters that will be focused on – **n\_estimators,** **max\_depth**, **min\_samples\_leaf** and **min\_samples\_split.**

**n\_estimators** – the number of trees in the forest, generally having more trees increases the performance of the model but might lead to overfitting as it the trees become too detailed.

A screen shot of a graph

Description automatically generatedA white board with blue and orange dots

Description automatically generated

*Fig 6.8 & 6.9 n\_estimators vs model accuracies*

As the number of estimators increases, the model scores increase, but it plateaus as it reaches 200-1000 estimators, with the model constantly overfit.

**Notable Ranges : (20-40), (100-400)**

**max\_depth** – the maximum depth of the tree, the deeper the tree is, the more details it captures but it might lead to overfitting.

A graph with blue and orange dots

Description automatically generated

*Fig 7. max depth vs model accuracies*

As the depth of the tree increases, the model score increases, but shows a plateau in scores from the range of 40-100.

**Notable Range: (20-40)**

**min\_samples\_split** – the number of samples required to split an internal node, increasing the number of samples required for a split helps to reduce overfitting.

A graph of a line

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidence

*Fig 7.1 & 7.2 samples split vs model accuracies*

As the number of samples to split increases, the model becomes less overfit, especially at higher ranges. However, when the requirement for a split is too high, both train and test scores also decrease.

**Notable Range: (40-100)**

**min\_samples\_leaf** – the minimum number of samples for it to be a leaf node, similar to the number required to split, for the higher the number, it gives a more generalized model reducing overfitting.

A graph of a line

Description automatically generated with medium confidence

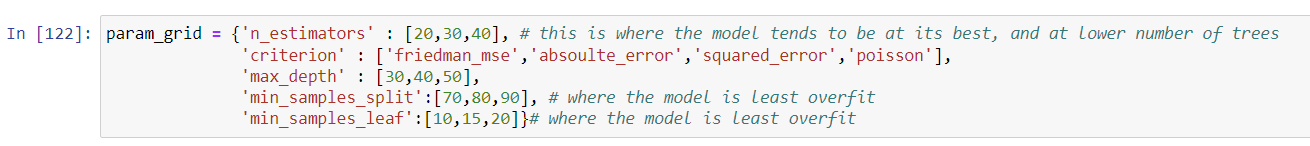
*Fig 7.2 samples leaf vs train and test accuracies*

As the requirement increases, the model becomes less overfitted, but it also reduces both train and test scores.

**Notable Range: (10-20)**

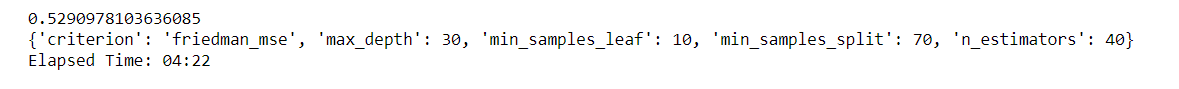
### **3.5.3 Improve the Model**

**1st iteration**



*Fig 7.4 1st grid*

For the first grid, the ranges where the model is the best is used.



*Fig 7.5 1st grid best parameters*

*A screenshot of a computer code

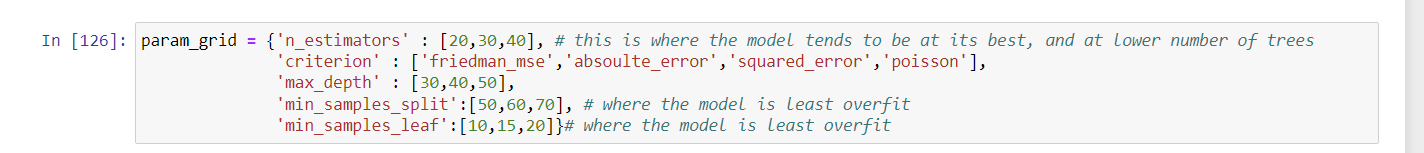
Description automatically generated*

*Fig 7.6 model build based on 1st grid best parameters*

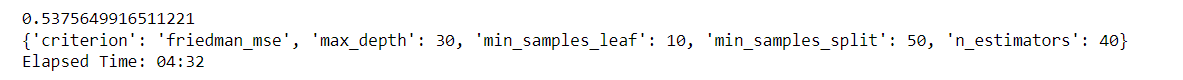
Evaluating the new model, the overfitting has been reduced from the base model by a substantial amount.

**2nd iteration**

In the first grid, the overfitting is reduced, but the scores could be improved by changing the range for the samples split – slightly increasing the complexity of the model.



*Fig 7.7 2nd grid*

**

*Fig 7.8 2nd grid best params*

*A black text on a white background

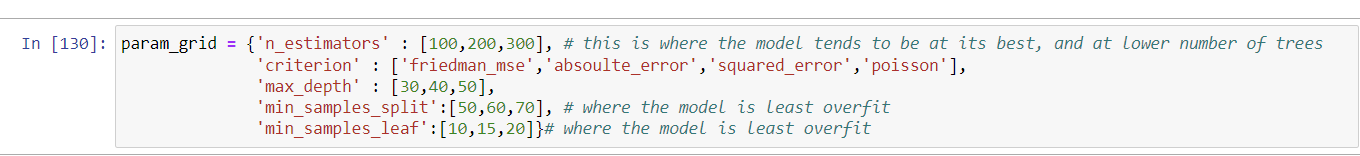
Description automatically generated*

*Fig 7.9 model built based on 2nd grid best params*

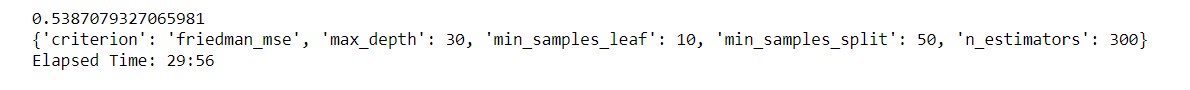
From the model built, the train and test scores slightly increased, but is slightly more overfitted.

**3rd iteration**

Again, to potentially increase the scores for both train and test, a new grid is experimented on for a higher number of estimators – to increase model complexity slightly.



*Fig 8. 3rd grid*

**

*Fig 8.1 Best params 3rd grid*

*A white background with black text

Description automatically generated*

*Fig 8.2 model built from best params*

There was no significant increase in the scores for both train and test, even at the attempt to increase model complexity with more trees.

### **3.5.4 Final Model Evaluation**

A screenshot of a computer

Description automatically generated

*Fig 8.3 all model improvements*

For the final ‘best’ model after improvements, the best was from the second grid, with relatively high training and testing scores.

*A black text on a white background

Description automatically generated*

*Fig 8.4 best model for RF regressor*

**Improvements from base model**

The model’s train and test r-square values addressed overfitting, with it being less overfitted than the base model.

## 3.6 Model #2 – XG Boost

XGBoost (eXtreme Gradient Boosting) Regressor is a powerful and efficient machine learning algorithm specifically designed for regression tasks. It belongs to the family of gradient boosting algorithms and is known for its speed and performance. XGBoost builds an ensemble of weak learners (typically decision trees) sequentially, with each new tree aiming to correct the errors of the combined ensemble so far. The algorithm uses a gradient descent optimization technique to minimize a specified loss function, resulting in a highly accurate and robust predictive model. XGBoost incorporates regularization techniques to prevent overfitting and is widely used in various regression problems due to its effectiveness and scalability.

Reading reference : [Introduction to Boosted Trees — xgboost 2.0.3 documentation](https://xgboost.readthedocs.io/en/stable/tutorials/model.html)

### **3.6.1 Build Base Model**

A screenshot of a computer code

Description automatically generated

*Fig 8.5 base model for XGB regressor*

Evaluating the base model for XGB, it is very overfitted, with the train r-square almost being close to 1, and the test only at about 0.6. Improvements can be made to reduce the overfitting through tuning the hyperparameters.

### **3.6.2 Inspecting Hyperparameters**

XG Boost has a wide range of parameters however for this segment, there will be a focus on the Tree Based parameters that XG Boost utilizes to control the decision trees within the ensemble. These hyperparameters are: **eta**, **subsample**, **lambda**,**alpha,n\_estimators and max\_depth.**

**eta** – also known as the learning rate, it is the step size shrinkage used for each boosting iteration, which can help prevent overfitting.

A graph with a line and a line

Description automatically generated with medium confidence

*Fig 8.6 learning rate vs train and test scores*

The learning rate is set at a **range from 0-1** for XG Boost. As the learning rate increases, the t train score shows an increase, but the testing score decreases.

**subsample** – subsample ratio of training instances, it controls the fraction of training data to be randomly samples for growing trees which reduces overfitting.

A graph of a line

Description automatically generated with medium confidence

*Fig 8.7 subsample vs train and test scores*

For XG Boost, the subsample **range is specified from 0-1**. As shown here, as the subsample ratio increases, the train and test scores show an increase.

**lambda** – l2 regularization term for weights, it adds a penalty to squared weights, controlling model complexity and helps to reduce overfitting.

A graph of blue and orange lines

Description automatically generated

*Fig 8.8 lambda vs train and test scores*

As the regularization strength increases, the model gradually becomes less overfitted. However, it results in the scores decreasing as well.

**Notable range: 600-1000**

**alpha** – l1 regularization term, which adds a penalty to the absolute weights preventing overfitting.

A screen shot of a graph

Description automatically generated

*Fig 8.9 alpha vs train and test scores*

Like the l2 regularization term, the model becomes less overfitted, but at the cost of reduced accuracy.

**Notable Range: 600-1000**

**n\_estimators –** the number of boosting rounds or trees to run, increasing the number of trees improves model performance but might introduce overfitting.

A graph with blue and orange dots

Description automatically generated

*Fig 9. n\_estimators vs train and test scores*

As the number of estimators increases, the accuracy increases, but plateaus as it increases as well.

**Notable Range: (400-800)**

**max\_depth –** depth of the tree, overly complex trees might lead to overfitting.

A screenshot of a computer

Description automatically generated

*Fig 9.1 max\_depth vs train and test scores*

As the depth of the tree increases the scores increase, but plateau as they increase as well.

**Notable Range: (10-50)**

### **3.6.3 Improve the Model**

**1st iteration**

For the first grid, the model is tuned without adding regularization terms, to see if overfitting can be reduced without reducing model complexity.

A white background with red and green text

Description automatically generated

*Fig 9.2 1st grid*

*A screenshot of a computer

Description automatically generated*

*Fig 9.3 best params for 1st grid*

*A computer screen shot of a computer code

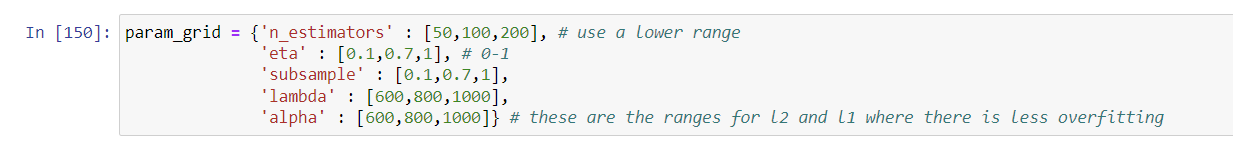
Description automatically generated*

*Fig 9.4 model built based on best params for 1st grid*

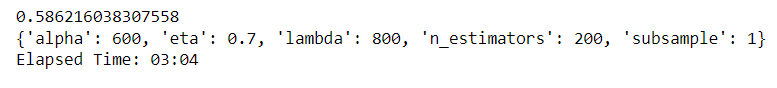
Evaluating the model created, even though the r-squared values increased slightly for the train and test sets, the overfitting was not reduced. This addressed the need to make the model more generalized which can be done through adding the regularization terms.

**2nd iteration**

From the first grid, the need to add regularization terms was addressed, and the best ranges were used for the parameters.

**

*Fig 9.5 2nd grid*

**

*Fig 9.6 best params from 2nd grid*

*A screenshot of a computer code

Description automatically generated*

*Fig 9.7 model built based best params from 2nd grid*

Evaluating the model built based on the best parameters, the overfitting is now reduced, with the r-squared values slightly closer to each other than the base model.

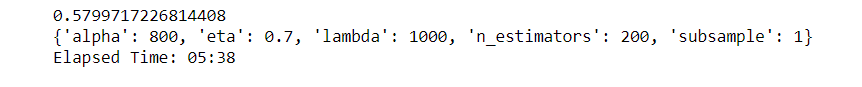
**3rd iteration**

To reduce model complexity and overfitting, lower values for learning rate are experiment on, with also with higher regularization penalty.

A screen shot of a computer code

Description automatically generated

*Fig 9.8 3rd grid*

**

*Fig 9.9 3rd grid best params*

*A screen shot of a computer code

Description automatically generated*

*Fig 10. model built based on 3rd grid best params*

For the r-squared values, the train score increased while the test score decreased slightly.

### **3.6.4 Final Model Evaluation**

A screenshot of a computer

Description automatically generated

*Fig 10.1 models built*

Based on the all the improvements implemented, the ‘best’ model that was built was from the second grid, which had the added regularization terms at a relatively high learning rate.

A screenshot of a computer code

Description automatically generated

*Fig 10.2 best XGB model*

**Improvements from base model**

The model was least overfit, with satisfactory training and test scores. Previously, the base model had higher overfitting at a 0.92 train r-squared and 0.6 test r-squared, the model is now less overfitted and less biased.

## 3.7 Model #3 – SVR

A Support Vector Regressor (SVR) is a machine learning algorithm used for regression tasks. It is part of the support vector machines (SVM) family, which is initially designed for classification. SVR, however, extends the concept to predict continuous rather than discrete outcomes. The goal of SVR is to find a hyperplane that best fits the data points while minimizing the margin violation. It does so by mapping the input data into a higher-dimensional space and determining a hyperplane that represents the best-fit regression line. SVR is particularly effective in capturing non-linear relationships and is robust against outliers, providing accurate predictions in various regression scenarios.

Reading reference: [sklearn.svm.SVR — scikit-learn 1.4.0 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html)

### **3.7.1 Build Base Model**

A black text on a white background

Description automatically generated

*Fig 10.3 base model SVR*

From the base model built, the model r-squared values are relatively low for both train and test sets, with it being slightly overfitted. This can be improved with tuning in subsequent steps.

### **3.7.2 Inspecting Hyperparameters**

SVR has several parameters that significantly impact its performance and its ability to generalize well with data. These are the hyperparameters that will be focused on in this segment – **gamma**, **C**, **epsilon** and **max\_iter**.

**gamma** – kernel coefficient, which is only applicable for the ‘rbf’,’poly’ and ‘sigmoid’, determining the influences of a training point in the decision boundary. A high gamma might lead to a more complex decision boundary, while a low gamma (which produces a smoother boundary), might lead to underfitting.

A graph of a line

Description automatically generated with medium confidenceA graph on a white board

Description automatically generated

*Fig 10.4 & 10.5 gamma against train and test scores*

As shown here, as the gamma value increases, the train and test accuracies become less overfitted, but also at the cost of reduced accuracy.

**Notable ranges: 10-30**

**C** – the regularization parameter, it influences the width of margin and penalty for misclassification. Higher C might lead to overfitting, while lower C may lead to underfitting.

A graph of a curve

Description automatically generatedA graph of a curve

Description automatically generated

*Fig 10.6 & 10.7 C against train and test scores*

**Notable ranges: (0-1),(8-10)**

**max\_iter** – number of iterations for the solver to converge, when appropriately set, the model will converge finding the balance between fitting the data well.

A graph of a graph

Description automatically generated with medium confidence

*Fig 10.8 max\_iter against train and test scores*

As the number of iterations is increased, the train and tests scores also increase, and plateaus at about 3000-4000 iterations.

**epsilon** – margin of tolerance for errors, sets the width of the epsilon sensitive tube within which no penalty is associated with errors. With a narrow rube, it is more sensitive to errors, presenting higher risks of overfitting. Conversely, a wide tube allows for more tolerance for errors, which may lead to underfitting.

A whiteboard with blue and red lines

Description automatically generated

*Fig 10.9 epsilon against train and test scores*

### **3.7.3 Improve the Model**

**1st iteration**

For the first grid, the ranges used are where the parameters for C and max\_iter are found to be most optimal.

A screen shot of a computer code

Description automatically generated

*Fig 11. 1st grid*

*A screenshot of a computer code

Description automatically generated*

*Fig 11.1 best params for 1st grid ]*

A screenshot of a computer program

Description automatically generated

*Fig 11.2 model built based on best params for 1st grid*

From the model built, the train and test r-squared values were found to be negative, hence the model had to be tuned further and more appropriately.

**2nd iteration**

For the second grid, to improve the model accuracies, a higher set of regularization strength was experimented on.

A graph of a curve

Description automatically generated

*Fig 11.3 higher regularization strength*

With higher regularization strength the train and test scores are set to increase.

A screen shot of a computer code

Description automatically generated

*Fig 11.4 2nd grid*

*A black and white text

Description automatically generated*

*Fig 11.5 best params 2nd grid*

*A black text on a white background

Description automatically generated*

*Fig 11.6 best params 2nd grid*

As expected, with higher regularization strength, the train and test r-squared values increased.

**3rd iteration**

To increase model accuracy, the gamma hyperparameter was introduced, to bring more complexity to the model.

A white background with red and green text

Description automatically generated

*Fig 11.7 3rd grid*

*A black and white text

Description automatically generated*

*Fig 11.8 best params from 3rd grid*

*A black text on a white background

Description automatically generated*

*Fig 11.9 Model built based on best params from 3rd grid.*

From the model built with gamma introduced, the r-squared values were very overfitted, with the train r-squared value almost near 1. For consecutive grids, the gamma value will be removed, to simplify the model.

**4th iteration**

For the fourth grid, the epsilon hyperparameter is introduced, to set a margin of tolerance for the model.

A screen shot of a computer code

Description automatically generated

*Fig 12. 4th grid, with epsilon*

*A black and white text

Description automatically generated*

*Fig 12.1 4th grid best params*

*A black text on a white background

Description automatically generated*

*Fig 12.2 model built based on 4th grid best params*

With epsilon introduced, the model scores went back to normal, but did not improve significantly.

**5th iteration**

For the fifth grid, a higher epsilon is experimented on to set a higher margin of tolerance for the model.

A computer code with red and blue text

Description automatically generated

*Fig 12.3 5th grid*

*A screenshot of a computer

Description automatically generated*

*Fig 12.4 5th grid best params*

*A white background with black text

Description automatically generated*

*Fig 12.5 model built based on 5th grid best params*

From the model built, the train and test r-squared values slightly increased, with the model being slightly overfit.

### **3.7.4 Final Model Evaluation**

A table with numbers and letters

Description automatically generated

*Fig 12.6 Models Created*

Evaluating all the models, the ‘best’ model that was optimized for SVR was the final grid, with introduced epsilon.

**A black text with white text

Description automatically generated with medium confidence**

*Fig 12.7 ‘best’ SVR model*

**Improvements from base model**

The model’s train and test r-squared values improved from the base model (0.23 Train and 0.2 Test). However, the model is slightly overfitted – indicating that it may have high bias and low variance.

## 3.8 Summary

### **3.8.1 Model Selection**

To evaluate which of the ‘best’ models built previously, cross validation is used. Cross validation involves training the model on a subset of data, and validating it on the remaining portions, iteratively rotating subsets and validating it on the remaining portions. Here, **K-folds** cross validation is used to evaluate the models, where the dataset is divided ‘k’ times and validated ‘k’ times each using a different fold for validation.

A screenshot of a computer code

Description automatically generated

*Fig 12.8 cross validation scores for ‘best’ models*

From the cross validation for each model, the SVR model can be discarded, as the model’s train and test r-squared values are presented to be negative, indicating that the model is unable to capture any meaningful patterns in different folds.

### **3.8.2 Random Forest Regressor vs. XG Boost**

A white background with black numbers

Description automatically generated

*Fig 12.8 cross validation scores for RF and XGB*

**Random Forest Regressor**

*Cross Val Train R-square - 0.695, moderate fit of training data*

*Cross Val Test R-square - 0.507, moderate level of generalization*

*Comments – model is slightly overfit with higher train r-squared values.*

**XG Boost**

*Cross Val Train R-square - 0.860, higher fit to training data*

*Cross Val Test R-square - 0.547, moderate level of generalization*

*Comments – model is slightly more overfit with higher train r-squared values.*

**Final Decision**

Considering overfitting and model generalization, the XG Boost model is better, demonstrating higher train and testing scores compared to the RF regressor. In context of predicting listing prices for Airbnb’s platform, the better generalization is crucial. The consistent performance across different subsets shows that the XG Boost model has to capacity to make relatively accurate predictions. This benefits Airbnb and its users, where a reliable model ensures hosts receive fair and competitive rates for their listings, contributing to an improved overall user experience within the Airbnb platform.

# Conclusion

The same process of tuning was applied for both classification and regression tasks.

For classification tasks, the initial step of inspecting hyperparameter relationships ensures that parameters impacting accuracy and overfitting are carefully selected. The examination of relationships between hyperparameter values and model performance is crucial for understanding their impact on classification metrics like precision, recall, and F1-score. The systematic evaluation of hyperparameter configurations helps in identifying the optimal set that maximizes the model's ability to correctly classify instances while avoiding overfitting.

Similarly, in regression tasks, the same process is adapted to the continuous nature of the target variable. The inspection of hyperparameter relationships considers the impact of the R-Squared metric. The goal remains to find the hyperparameter configuration that optimally balances accuracy and minimizes overfitting in the context of predicting continuous values.

Utilizing Grid Search Cross-Validation is particularly valuable for both classification and regression. The exhaustive exploration of hyperparameter values through cross-validation ensures robust model evaluation, providing insights into how well the model generalizes to unseen data. The iterative evaluation and adjustment of hyperparameters post-Grid Search are beneficial in refining the model's performance for both classification and regression scenarios, contributing to the overall effectiveness of the machine learning models across different tasks.

# Reflection

## Improvements to ML Solution

**Tuning Process**

The tuning process for machine learning models plays a pivotal role in optimizing their performance. While GridSearchCV offers an automated approach, manual tuning allows for a more granular exploration of hyperparameters. Employing various search methods or manual adjustments provides a nuanced understanding of model behavior. Additionally, utilizing cross-validation throughout the tuning process, rather than just for final evaluation, offers a more reliable estimate of model performance, ensuring robustness against overfitting or underfitting issues.

**ML Model Performance**

The quality of data profoundly influences model performance, with feature engineering and dataset characteristics being critical factors. However, there are inherent limitations to improving model performance solely through hyperparameter tuning. It is not uncommon to encounter situations where the training accuracy improves, yet the test accuracy plateaus, signifying potential overfitting or insufficient generalization.

**Scope of Problem Statement**

Predicting prices in the context of the Airbnb dataset presents a unique challenge due to the user-defined nature of pricing on the platform. The arbitrary and subjective nature of Airbnb pricing makes it inherently complex for machine learning models to accurately predict. Despite challenges, exploring innovative approaches and considering domain-specific features may offer insights into enhancing prediction accuracy.

**Ensemble Methods**

Ensemble methods, such as Voting and Stacking, present powerful techniques to improve model performance by combining multiple models. Leveraging different weak learners for ensemble methods, specifically for algorithms like XG Boost and ADA Boost, enhances diversity and strengthens the ensemble's predictive capabilities. Incorporating ensemble methods contributes to building more robust and reliable models, harnessing the strengths of individual models to collectively achieve superior predictive performance.

## What could have been learnt better?

**How to tune hyperparameters**

Tuning hyperparameters is a nuanced task with no singular approach, often lacking a predefined process. The optimization of a model's hyperparameters involves exploring a range of values through methods like manual tuning, grid search, or randomized search. The iterative nature of this process requires a thoughtful and adaptive strategy to navigate the complex parameter space effectively.

**Machine Learning Process**

Machine learning modeling presents a distinctive challenge due to its inherent ambiguity, setting it apart from the more visually intuitive realm of data engineering. Unlike the structured nature of data manipulation, modeling involves navigating a complex landscape with multiple algorithms, hyperparameters, and intricate interactions. Visualization of the process is elusive, as it is not always straightforward to comprehend the inner workings of algorithms. Rather, it relies heavily on experiential knowledge. The more familiarity an individual possesses with a specific model or algorithm, the more adept they become at interpreting its nuances and steering it towards optimal outcomes.

The process is steeped in experiential learning, where practitioners draw upon their understanding of algorithms, data patterns, and domain-specific knowledge to guide the modeling journey. Trial and error play a significant role as modelers iteratively experiment with different configurations, hyperparameter choices, and feature engineering strategies. This experiential base enables them to make informed decisions and adapt their approach based on observed model behavior. In essence, the art of machine learning modeling is a dynamic, learning-intensive process, where expertise and a deep understanding of the data and algorithms intertwine to navigate the complexities of predictive modeling successfully.

# Discussion Panel

## Questions

A screenshot of a web page

Description automatically generated

A screenshot of a white paper

Description automatically generated

A screenshot of a computer

Description automatically generated

## Answers

A screenshot of a computer

Description automatically generated

A screenshot of a search

Description automatically generated

A screenshot of a computer

Description automatically generated