

**School of InfoComm Technology**

**Predictive Analytics**

Diploma in Financial Informatics (FI)

Diploma in Information Technology (IT)

YR 2/3 (2020/21), Semester 4/6

**INDIVIDUAL ASSIGNMENT 2**

(40% of Predictive Analytics Module)

**Deadline for Submission:**

**Presentation: 31st Jan 2021 (Sunday), 2359 Hours**

**Report: 14th Feb 2021 (Sunday), 2359 Hours**

|  |  |  |
| --- | --- | --- |
| Student Name | : | Tan Zi Xu |
| Student Number | : | S10185168 |

**Penalty for late submission:**

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

**NO** submission will be accepted after 21st Jan 2021, 23:59.

Contents

[Introduction 3](#_Toc64238247)

[HR Analytics 4](#_Toc64238248)

[Problem understanding & Approaches 4](#_Toc64238249)

[Classification Model Building 5](#_Toc64238250)

[Model Evaluation & Model Tuning 6](#_Toc64238251)

[Random Forest Classifier (Grid Searching) 6](#_Toc64238252)

[Parameter Tuning (Max Depth) 8](#_Toc64238253)

[Parameter Tuning (n estimators) 9](#_Toc64238254)

[Parameter Tuning (min sample leaf) 10](#_Toc64238255)

[Finalized RFC 11](#_Toc64238256)

[Support Vector Classifier (Grid Searching) 12](#_Toc64238257)

[Parameter Tuning (C value) 13](#_Toc64238258)

[Parameter Tuning (gamma) 14](#_Toc64238259)

[Finalized SVC 15](#_Toc64238260)

[Multi-Layer Perceptron Classifier (Grid Searching) 16](#_Toc64238261)

[Finalized MLPC 17](#_Toc64238262)

[Finalized Voting Classifier 17](#_Toc64238263)

[Summary 18](#_Toc64238264)

[Airbnb Singapore 19](#_Toc64238265)

[Problem understanding & Approaches 19](#_Toc64238266)

[Regression Model Building 20](#_Toc64238267)

[Model Evaluation & Model Tuning 20](#_Toc64238268)

[Random Forest Regressor (Grid Searching) 21](#_Toc64238269)

[Parameter Tuning (Max Depth) 22](#_Toc64238270)

[Parameter Tuning (n estimators) 23](#_Toc64238271)

[Finalized RFR 24](#_Toc64238272)

[XGBoost Regressor (Grid Searching) 24](#_Toc64238273)

[Parameter Tuning (Max Depth) 26](#_Toc64238274)

[Parameter Tuning (eta) 26](#_Toc64238275)

[Finalized XGR 27](#_Toc64238276)

[Multi-Layer Perceptron Classifier (Grid Searching) 28](#_Toc64238277)

[Finalized MLPC 29](#_Toc64238278)

[Finalized Model 30](#_Toc64238279)

[Summary 30](#_Toc64238280)

[Conclusion and Further Improvement 31](#_Toc64238281)

# Introduction

In predictive modelling, problems faced are typically one of the two main types, classification or regression. What predictive modelling does is to develop models based on data from the past and make prediction with new data. On a fundamental level, classification problem is about predicting the output in the form of labels and regression problem predict values instead. Such that both problem will require a model to receive the input variable as “X” and map it to output variable as “Y”.

For classification problem, the model’s role is typically to predict a class in the target variable based on the other variables it was trained on with the historical data. The predictive capability of a model in classification is estimated on how accurate the model can classify the target in the new data. Whereas, regression problem would predict a range of values but, it measures the difference between its prediction and the actual new data. Instead of accuracy, regression problem would typically use root mean square error (RMSE) to deduce how good the model for regression problem is.

Between the two measures that the models use, accuracy is looking at the percentage of correct classification the model made amongst the prediction made. Unlike accuracy, RMSE is the reporting of error made in such prediction of models for regression problem. In a nutshell, the difference between solving classification and regression can be summarized. Such that, classification is made for predicting discrete class labels and regression predicts continuous values.

Returning back to predictive modelling, the process is typically understanding the problem followed by determining the approach for the problem. When an approach is chosen, the model will be built and it could also be multiple models being built. Once the models are built, evaluation will take place and further development will be made for chosen models. They will be further tuned until the model is at its best optimal state that the model creator intends it to be or the model can no longer be further tuned due to uncontrollable limitations.

# HR Analytics

## Problem understanding & Approaches

Human Resource (HR) is a department that can be found in any business, the role of HR departments that manage employees and employee-related operations. One of the various operations that HR is responsible for is the promotion of employees within the company. However, analytics done to identify candidates for promotion can be rather laborious to process. Promotion screening is a process of great importance as HR cannot afford to hand out promotion to undeserving employees. The screening process also takes into account of various factors. Hence, HR needs to do extensive and stringent processing of key data in their analytics, placing a huge amount of constraint on the department. Which presents an opportunity to elevate the constraints on the HR by using predictive analytics to help identify potential promotion candidates.

For this problem, there are many classification models out there to tackle this problem and they each function differently. While I want to build the most appropriate model, I am also uncertain of which model I should build upon. Such that, I want to use different models to approach this question, models that function in its own unique way. I felt that a stand-alone model I built may not cover all details or aspect in this modelling process for this problem hence, I decided to use voting classifier. Voting classifier allows me to create and train an ensemble of models and combine their prediction. I planned to feed in 3 different models into my voting classifier and run through the 2 different voting scheme it has.

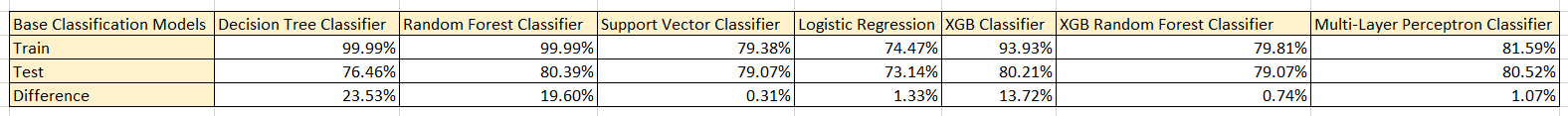
Moving on to data related manipulation for this problem, I had read the cleaned data from assignment 1 into a data frame. The data itself had already undergone stratified sampling but, I still need to scale some of the data columns. As one of the models that I am intending to build in the next section, the Support Vector Classifier, requires the data to be scaled. After scaling, the data is separated into x & y as input and output variables, the data is then split between data for training and testing. For this classification problem, the split ratio was two-third for training and the remaining portion is for testing. My reasoning for using this split ratio is that I felt in an 80-20 split, reserving 20% of the data for testing would be able to fully represent the whole data set. Considering that after stratified sampling, I am only left with 8658 data entries, which I would feel more comfortable with an 80-20 split if the data set has at least 10 thousand entries.

Lastly, I also want my models to perform consistently and as accurate as possible. Which I decide to deploy k-fold cross validation and use it average output as my scoring throughout the modelling process for both problems.

## Classification Model Building

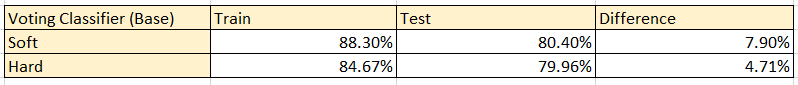
In this initial model building process, I want to identify the 3 models that I will use for my voting classifier. To do that, I have to build several classification models that I know of in their default parameter. As there many models, I lack the time and resource to manually fine tune each and every model. Hence, I decided that running the default parameters would give me a sense of how each model performed. Based on the performance, I will evaluate which models I would like in the next section.

Throughout this process, I have built 7 different classification models and recorded both of their cross validated training and testing accuracy result. The type of models that I created ranges from decision tree-based models to various ensemble models and even artificial neural network. Amongst the various ensemble models, not only bagging and boosting models were deployed, there is also boosted random forest model. To list all models that I built here, I have used Decision Tree Classifier (DTC), Random Forest Classifier (RFC), Support Vector Classifier (SVC), Logistic Regression (LG), XGBoost Classifier (XGC), XGBoosted Random Forest Classifier (XGRFC) and Multi-Layer Perceptron Classifier (MLPC).



Across the performance of the 7 base models, there are few finding based on some observations. Firstly, models that are decision tree-based models such as DTC, RFC and XGC tend to be overfitted. Next, models that were not overfitted, particularly the LG model, looks to be likely underfitted due to its poor performance. Lastly, the XGRFC model not overfitting despite being based on a decision tree, is likely due to being boosted, which might have reduced overfitting.

Starting with model selection for my voting classifier, I want use models with the best accuracy with their default parameters. However, I do not want to feed in the same type of models for my voting classifier as I find it bias to have models of the same nature to do voting. Which I can only have one of the decision tree-based models out of the 4 in DTC, RFC, XGC and XGRFC. Hence, the 3 models that I want to feed in are the RFC, SVC and MLPC. From there, I tried running the voting classifier with the 3 chosen model with default parameters to see how it would perform.

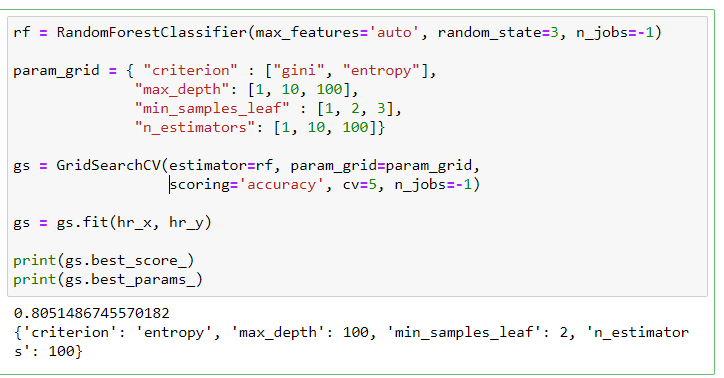


## Model Evaluation & Model Tuning

Based on the results from first voting classifier, the overfitting in RFC needs to be tuned out. However, the other 2 models are also likely to be underfitted, which would mean they require some additional tuning as well.

### Random Forest Classifier (Grid Searching)

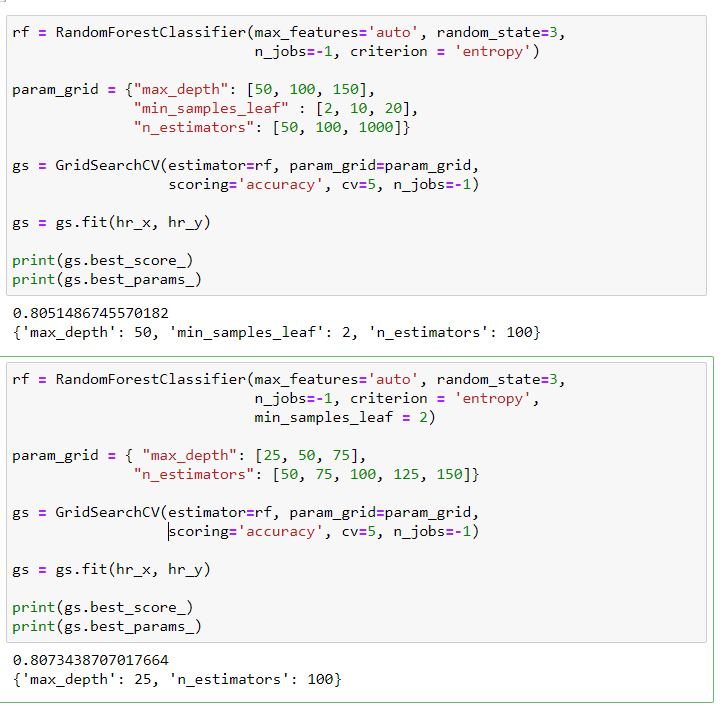
Starting with the RFC, I ran my first grid search to identify the criterion along with other hyperparameters such as max depth, n estimators and min sample leaf. In this initial grid search. I planned to keep the criterion that I got from this grid search as a constant for subsequent grid searches and parameter tuning for RFC. The brute forcing power of grid search gave me sense of where to start tuning them.



From here, I ran a couple more grid searches with different hyperparameter range. The decision behind running these next few grid searches was because the initial parameters may not cover the ideal range. I want to find and limit the range of these hyperparameters for subsequent manual tuning. Depending what result I get from the preceding grid search, the next one will have its range tweaked accordingly.

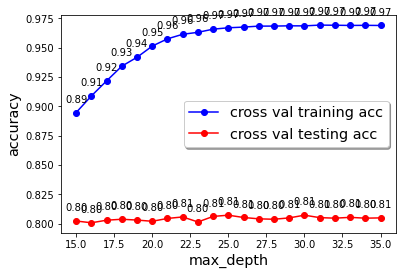
In my second iteration of grid search, I also found that the min sample leaf to be returned with the same result as the first iteration. I deduced that the ideal range could be between 1 to 5, there isn’t a need to run a grid search for it. Hence, I left out min sample leaf in the next iteration of grid search, assigning 2 in the along with the other parameters that I set as a constant.

However, I also limit to a maximum of 3 grid searches as I do not intend to use grid search to find the exact ideal value. It would not make sense to only run countless grid searches when I could get a range and fine tune each individual hyperparameters.

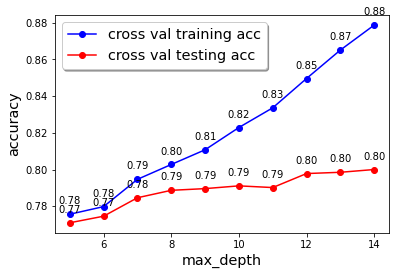


### Parameter Tuning (Max Depth)

With the grid search, I have identified which range I should look at for each hyperparameters when tuning. Starting with max depth, the range was initially between 1 to 100 but, the range was eventually limit to between 15 and 35 after a few iterations of grid searching.

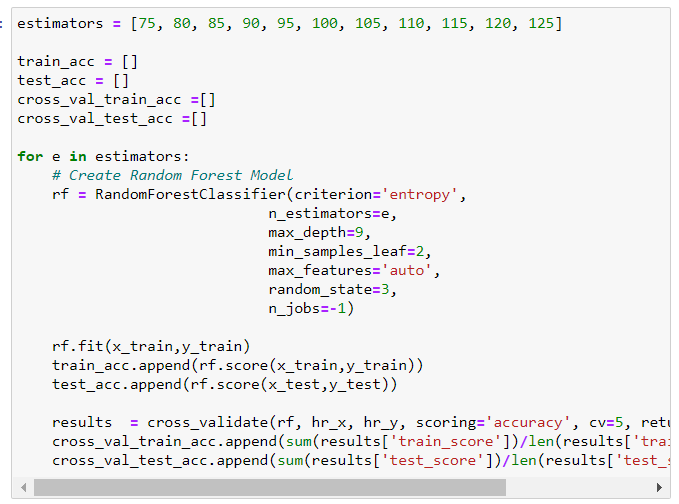
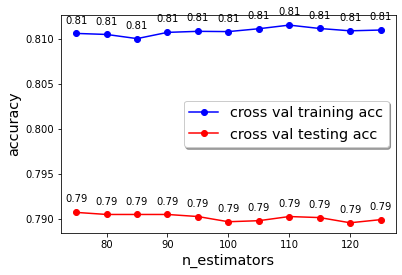


From there, I ran the first iteration of tuning for max depth but all the result that I got back are overfitted. Hence, I ran another iteration of tuning where the new range for max depth was between 5 to 14. In the second iteration, I saw much better results where I could identify 9 as the ideal max depth for my model. I could have chosen 10 but the difference between testing and training accuracy was so close to 3 percent to be considered overfitted that I determine 9 would be safer. Considering that the testing accuracy also did not improve that much between 9 and 10, I did not see the point in choosing 10 as the best depth.



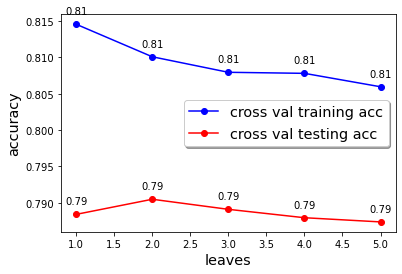
### Parameter Tuning (n estimators)

Next up, I tuned the n estimator hyperparameter and in the initial grid search, the range was between 1 to 100 again. In the second iteration, I increased the range limit to 1000 as way to check if the ideal n estimator is located way beyond 100. This would save me time from running multiple grid search with small increment to the range limit every time. After the second iteration, I noted that I should look into between 50 to 150 in my last grid search. Which narrows down the range to between 75 to 125 for parameter tuning after it return 100 as part of the best parameter in the last grid search.

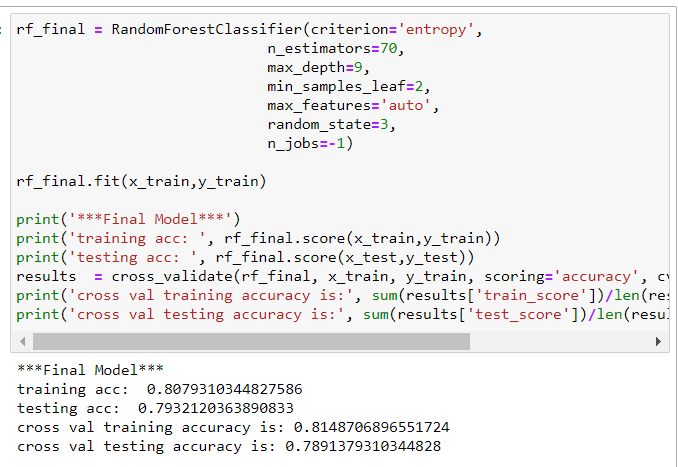
### Parameter Tuning (min sample leaf)

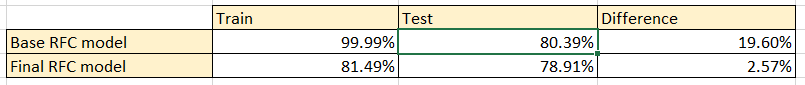
Lastly, I tuned the min sample leaf as the last hyperparameter for tuning in this model. As I already guessed the range should be between 1 to 5, I immediately ran each iterations of the model with different min sample leaf. In the results I got back, 2 was the ideal number of min sample leaf.



### Finalized RFC

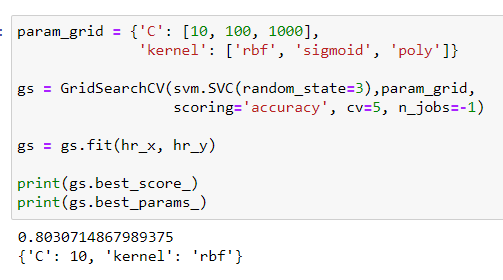
With all the hyperparameters tuned, I proceed finalized my RFC model for the voting classifier. When compared to the RFC model with default parameters, my finalized model did drop a little in terms of testing accuracy. However, the finalized model no longer overfits and its testing accuracy is consistent performing at 78.9%.



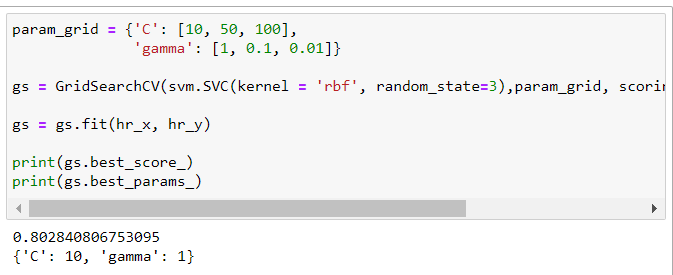


### Support Vector Classifier (Grid Searching)

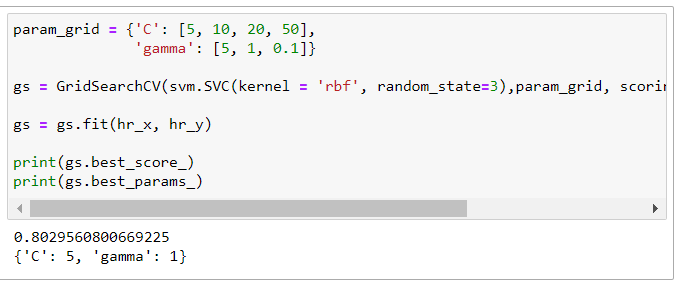
Moving onto the next model, I will be tuning the SVC model. The first thing that I want to confirm was which kernel the model should be using, which I used grid search to help me identify the kernel. Knowing that in SVC models, I can adjust the c value to change the complexity of my model, I decided to focus on this hyperparameter. As my model with default parameter could be underfitting, I should have a higher c value to increase its accuracy. Hence, I also set a big range for c value in my initial grid search.



Following the same procedure for RFC, I ran a couple more grid searches to tweak around with the coverage of the parameter grid. Now that I know my model is best with using rbf kernel, I decided to also tweak the gamma along with c value for the next grid search. Speaking of the c value, I also limited the range to 100 for the next grid search and add a middle value between 10 to 100 since I got 10 in my initial grid search.

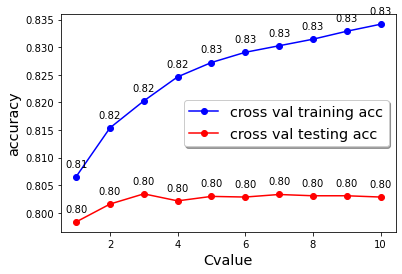


After this iteration of grid search, I ran 1 last grid search on this model with a final adjustment to range on the hyperparameters.



### Parameter Tuning (C value)

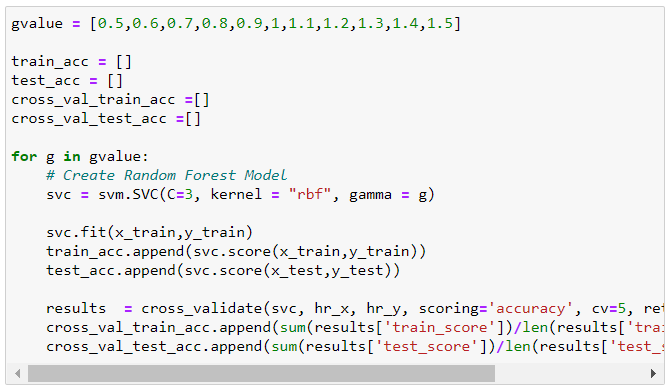
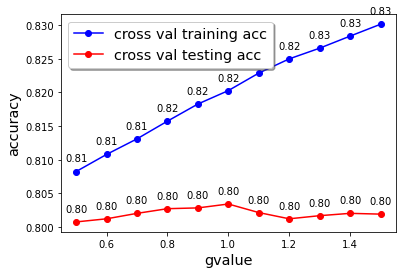
With the grid search, I have identified which range I should look at for each hyperparameters when tuning. Starting with c value, the range was initially between 1 to 1000 but, the range was eventually limit to between 5 to 50 after a few iterations of grid searching. Based on the last grid search results, I deduced the range to be between 1 to 10 for c values and ran each iterations of the model within this range.



Based the chart above, I determined the c value should be at 3 for the optimal accuracy. As it performed the best at that c value and there isn’t any overfitting.

### Parameter Tuning (gamma)

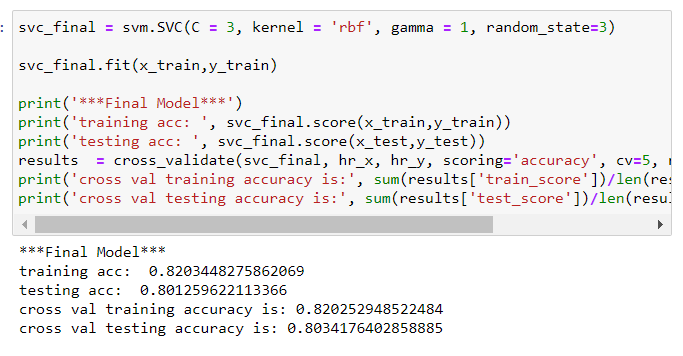
From there, I proceed to tune the gamma in a similar fashion, where range is between 0.5 and 1.5.

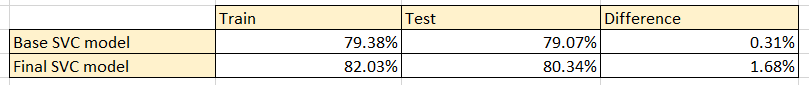
 

Here I determine gamma at 1 will give me the highest accuracy while avoid overfitting and proceed to create my finalized SVC model.

### Finalized SVC

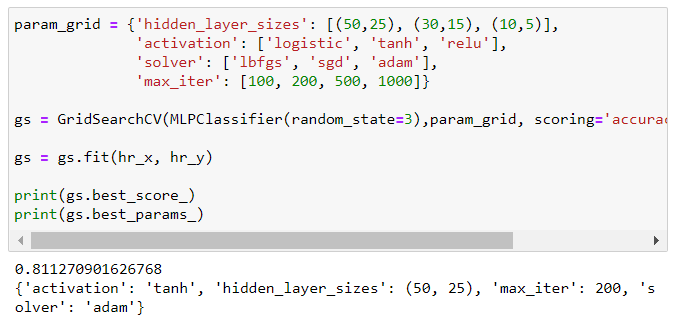
When compare to the base model with default parameters, the final model saw improvement in its accuracy. However, it is only improved marginally after tuning.



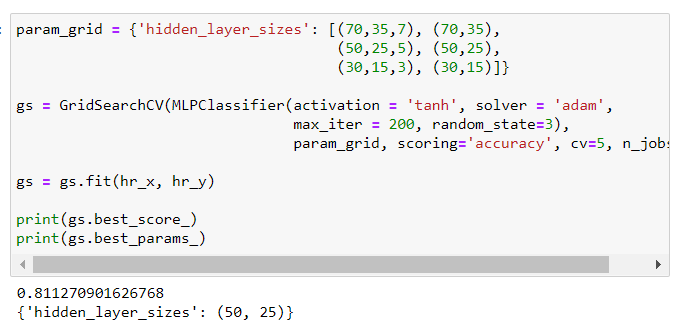


### Multi-Layer Perceptron Classifier (Grid Searching)

Lastly, I will be tuning the MLPC model. While I do not understand the MLPC too well, I do know I need to identify which solver and activation the model should be using. I also know that I am supposed to tune the hidden layer size of the MLPC. As my model with default parameter could be underfitting, I tried out the different activation and solvers while also setting other hyperparameters in my initial grid search. The additional hyperparameters include hidden layer size and max iteration.

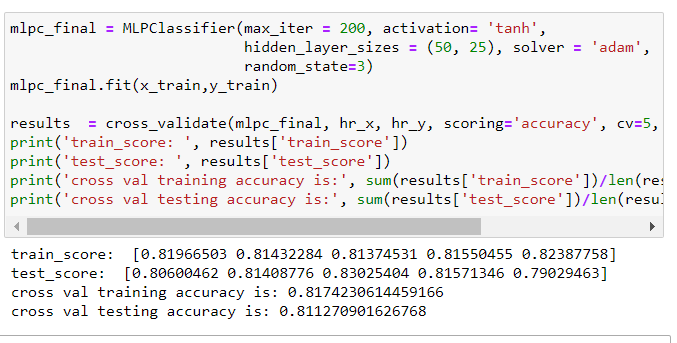


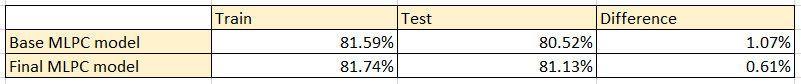
From the initial grid search, I can establish my model should be using the tanh activation, adam solver and the max iteration I should have is 200. Leaving only hidden layer size to be tune, which I decide to run an additional variation to the existing range that was set. Such that the different layer size set in the first grid search only had 2 layers but, the new variation would introduce 3 layers.



### Finalized MLPC

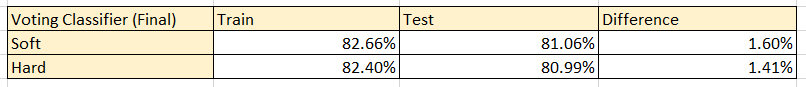
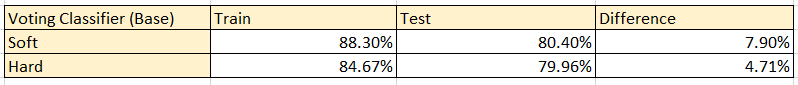
With this output from the grid search, I attempt running this suggested hidden layer size as part of my final model. Which also saw improvement to the testing accuracy when compared to the model with default parameter and once again the improvement was also marginal.





### Finalized Voting Classifier

With all models tuned out, I fed the finalized models into my finalized voting classifier and ran through the 2 different voting schemes. Under both voting schemes, the finalized voting classifier no longer overfits and there are marginal improvement that is close to 1%



## Summary

For this problem, I have built a total of 7 different types of classification model, 3 of which were chosen for improving through tuning their parameters. Amongst the 7 initial models, they were built with default parameters, some of the models had similar results. The lowest performing model was Logistic Regression at 73.14% in testing accuracy and the highest performing model was the Multi-Layer Perceptron Classifier model at 80.52% in test accuracy. An observation made was that decision tree-based models tend to overfit with the exception of XGBoosted Random Forest Classifier due to the boosting lowering its tendency to overfit.

The 3 chosen models for tuning were the Random Forest Classifier, the Support Vector Classifier and the Multi-Layer Perceptron Classifier. These 3 models were chosen to be fed into a voting classifier as the intended approach. Through the use of grid search and further parameter tuning, each of the chosen models were tuned to its highest performing condition without overfitting.

If I were to recommend the single best model for this problem, I would recommend the Multi-Layer Perceptron Classifier model. The MLP is an artificial neural network, one that resembles our human brain works but simpler. Which sort of replicates how we would think and classify inputs that were given. When compared to a decision tree-based models, it is not strictly limiting its decision based on a feature. Which enables a level of flexibility or maybe complexity when classifying the input, giving more leeway for MLP to predict better than the other models. Hence, MLP classifier performs better other models that I have built in this problem.

# Airbnb Singapore

## Problem understanding & Approaches

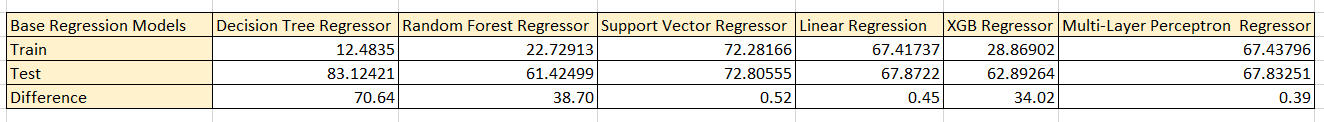
Airbnb is widely known as an online platform that connects travelers with host looking to rent out their properties. It is a popular tool for travelers or organizers that are looking for accommodation during their trip. While providing an opportunity for travelers to have a unique and personalized experience, the host gets to earn some rental revenue, making it a win-win situation for both travelers and hosts alike. In this problem, the task is to predict the rental price of the listing in Airbnb Singapore.

For this problem, there are many regression models out there to tackle this problem and they each function differently. Unlike the approach I used for the classification problem, I want to get straight to the point, using only the best model with the lowest error. I plan to build a set of basic models, train the top 3 and pick the best out of those 3. In a problem like this, I want to minimize as much error that my model can make as possible.

Moving on to data related manipulation for this problem, I had read the cleaned data from assignment 1 into a data frame. In this data set, I still need to scale most of the data columns except for the target column here, price. As one of the models that I am intending to build in the next section, the Support Vector Regressor, requires the data to be scaled. After scaling, the data is separated into x & y as input and output variables, the data is then split between data for training and testing. For this regression problem, the split ratio was two-third for training and the remaining portion is for testing. My reasoning for using this split ratio is the same reason found in the classification problem. Considering that this data set already has lesser entries, I would feel more comfortable following the same steps.

## Regression Model Building

Following the same procedure as the classification problem, I have to build several regression models that I know of in their default parameter. The reasoning behind it is also same where I am lacking in time and resource to manually tune each model I make. This time round I have built 6 different regression models and recorded both of their cross validated training and testing RMSE. The type of models that I created ranges from decision tree-based models to various ensemble models and even artificial neural network. Amongst the various ensemble models, not only bagging and boosting models were deployed. To list all models that I built here, I have used Decision Tree Regressor (DTR), Random Forest Regressor (RFR), Support Vector Regressor (SVR), Linear Regression (LR), XGBoost Regressor (XGR) and Multi-Layer Perceptron Regressor (MLPR).



Across the performance of the 6 base models, there are few finding based on some observations. Firstly, models that are decision tree-based or ensembles models such as DTR, RFR and XGR tend to be overfitted. Lastly, models that were not overfitted, particularly the SVR model, looks to be likely underfitted due to its poor performance when compared to the others.

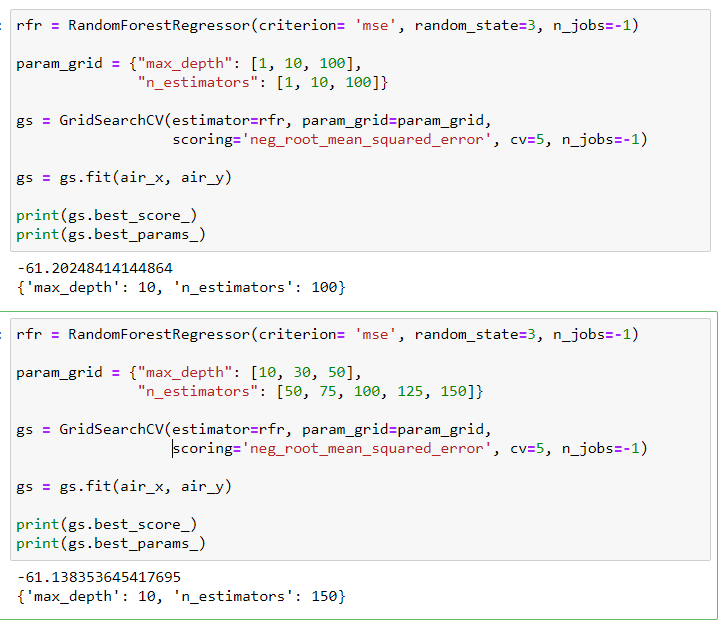
Looking at the results, the top 3 models that performed well here are RFR, XGR and MLPR. Coincidentally, I get to match 2 ensemble models against each other in RFR’s bagging and XGR’s boosting. The MLPR also serves as a baseline comparison between the 2 ensemble models and through this I get find out whether a bagging, boosting or neural network performs better in a regression problem after the tuning them.

## Model Evaluation & Model Tuning

Referring back to how the 3 models faired, the RFR and XGR was overfitted, which requires tuning to reduce the model’s complexity. Whereas, the MLPR could be underfitted but, I will never know until I try tuning it.

### Random Forest Regressor (Grid Searching)

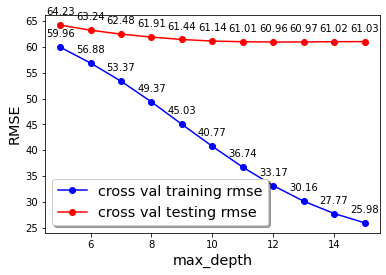
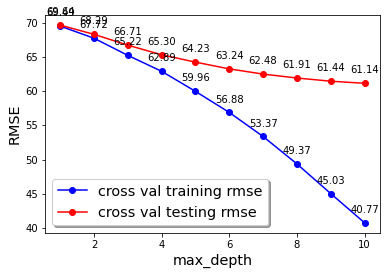
Following the same process that I did in the classification problem, I ran a couple of grid searches to narrow down the range of the hyperparameters for tuning. However, there is 2 key differences. Firstly, my criterion is already set as mse as I am using RMSE. With random forest regressor, I have either mse or mae to use as the criterion, I would have to use the former if I were to use RMSE as my measurement for regression models. Lastly, I also decided to use the default min sample leaf as I notice once I tune variables such as max depth, changes in min sample leaf were negligible.



Instead of running 3 or more grid searches, I realized that with the grid search above, the range I have to explore for tuning is already narrow enough. Hence, I did not see the need to do anymore grid searches and moved on with tuning the first hyperparameter.

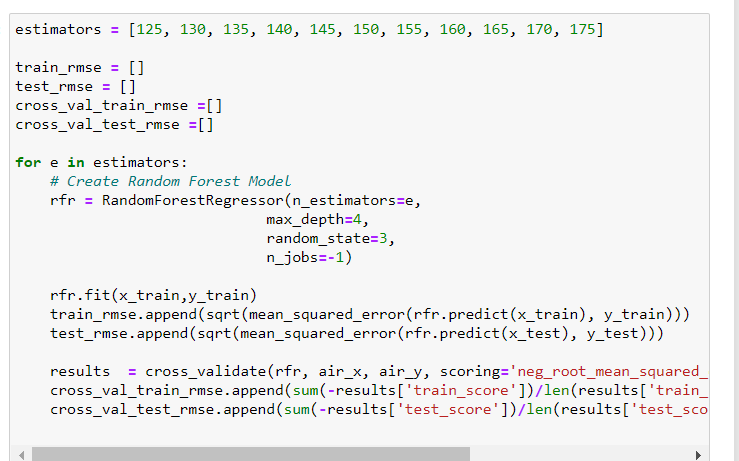
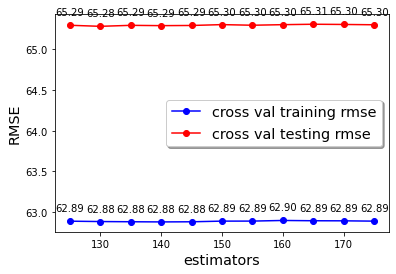
### Parameter Tuning (Max Depth)

Based on the grid search, I have identified the range for max depth I should look at is between 5 to 15. After running the various iteration of RFR at different max depth, I found that at very point within the range resulted in an overfitted model. Hence. I adjusted the range to be between 1 to 10 and ran through the various iterations again. Where this time round, I got 4 as the ideal max depth for lowest error while not pushing to be overfitted.

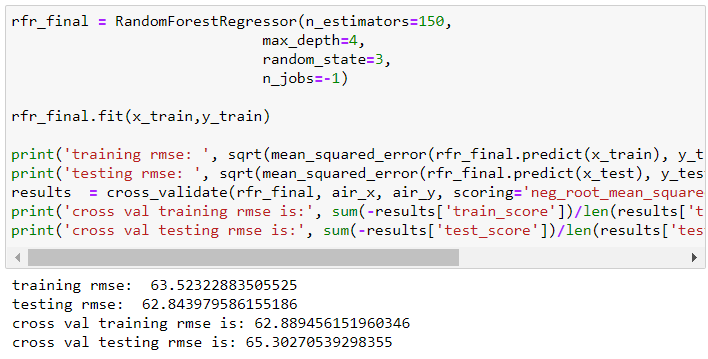
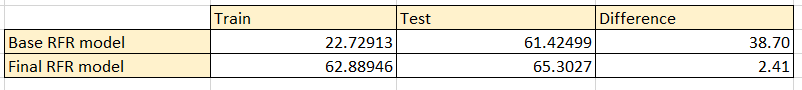
### Parameter Tuning (n estimators)

Before finalizing the RFR model, I am left with 1 more hyperparameter to tune, which is the n estimators. Based on the grid search earlier, I estimate the range that I should explore to be between 125 to 175. After running the various iteration of RFR at different n estimators, I found that there is hardly any difference between each iteration. Hence, I set the best estimator to be 150 as per what the grid search suggested.

### Finalized RFR

With all the hyperparameters tuned, I proceed finalized my RFR model. When compared to the RFR model with default parameters, my finalized model increased in its error rate. However, the finalized model no longer overfits and its testing RMSE is consistent performing at 65.30.

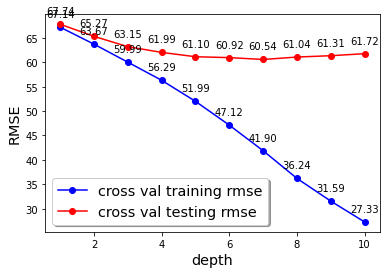
### XGBoost Regressor (Grid Searching)

For XGR, I start by grid searching which booster I should used for my model and the subsequent grid searches was to find the range is should explore when tuning. Due to the booster returning as dart, my chosen parameter for tuning are max depth and eta. As it will be a tree-based model, I will have to tweak with its depth range and the eta is the learning of the model.



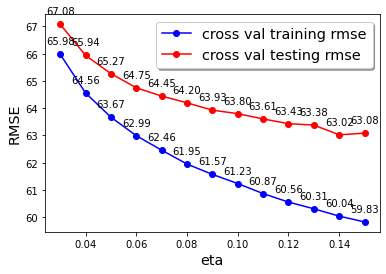
### Parameter Tuning (Max Depth)

Similar to tuning max depth in RFR, the depth range I will be exploring is from 1 to 10 as suggested from the grid search. After running the various iteration of XGR at different max depth, I found that 2 was the ideal range for lowest RMSE while not overfitting.



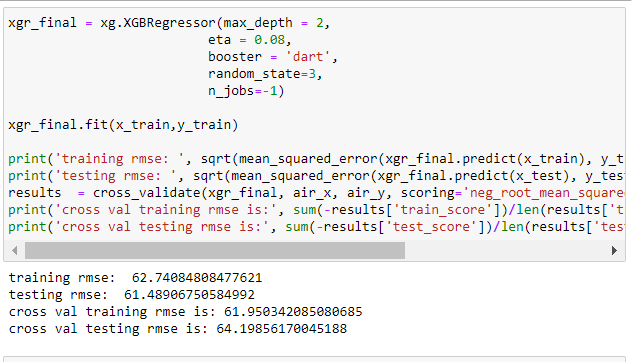
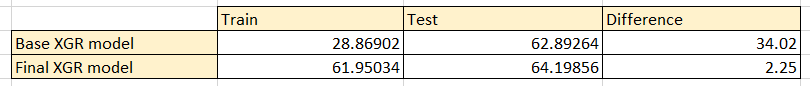
### Parameter Tuning (eta)

As for the eta, I set a range between 0.03 and 0.15 to be explored as part of tuning based on the last grid search. After running the various iteration of XGR at different eta, I found that 2 was the ideal range for lowest RMSE while not overfitting.



### Finalized XGR

With all the hyperparameters tuned, I proceed finalized my XGR model. When compared to the XGR model with default parameters, my finalized model increased in its error rate. However, the finalized model no longer overfits and its testing RMSE is consistent performing at 64.20.

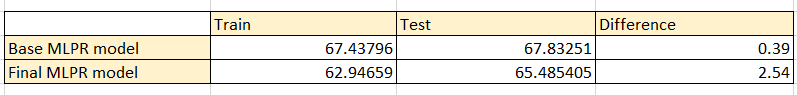
### Multi-Layer Perceptron Classifier (Grid Searching)

Lastly, the final model for tuning the MLPR model. Similar to dealing with the MLPC model, I know I need to identify which solver and activation the model should be using first. Then I should tune the hidden layer size of the MLPR, as my model with default parameter could be underfitting. Following the same steps as I did for MLPC, I replicate the things I did for this model.



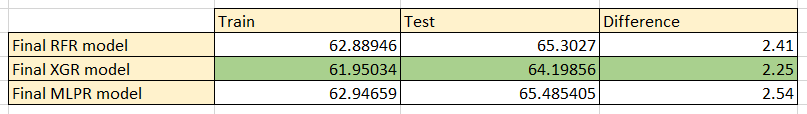
### Finalized MLPC

However, I found out that the hidden layer suggested by the grid search would give me an overfitted model just as I was about to finalized the model. From there, I tried variation with small weight in the hidden layer size and found (20,10) gave the optimal result. With finalized model, I saw improvement to the testing RMSE when compared to the model with default parameter and the improvement was also marginal.

### Finalized Model

With all models tuned out, I had a hard deciding which model is the best as all 3 of the models have very similarly close results in their testing RMSE. However, the tiebreaker that help me made my final decision to break the stalemate was their difference between training and testing RMSE. Using consistency of their performance as factor, the model with the smallest difference would be deemed as the best model and used my final model. Thus, the XGR was my final model to use in this regression problem.



## Summary

For this problem, I have built a total of 6 different types of regression model, 3 of which were chosen for improving through tuning their parameters. Amongst the 6 initial models, they were built with default parameters, some of the models had similar results. The lowest performing model was Support Vector Regression at 72.805 in testing RMSE. The highest performing model was the Random Forest Regressor at 61.425 in testing RMSE, although it was overfitted. An observation made was that decision tree-based models tend to overfit.

The 3 chosen models for tuning were the Random Forest Regressor, the XGBoost Regressor and the Multi-Layer Perceptron Regressor. Through the use of grid search and further parameter tuning, each of the chosen models were tuned to its highest performing condition without overfitting.

If I were to recommend the single best model for this problem, I would recommend the XGBoost Regressor model. The XGR is boosting ensemble model. As a collective string of weak models, it is one that improves by learning from mistakes of the preceding model. When compared to a decision tree-based models, it does not in parallel with the weak models instead the series models learn from mistakes. Which set itself apart by improving though its own self. Hence, XGBoost Regressor performs better other models that I have built in this problem.

# Conclusion and Further Improvement

As a quick summary, I have applied to 2 different approaches to deal with problems, one for classification and another for regression. The models that I have built in both problems varied in how they function. I have tuned the models that seems promising or it was part of my intended approach. I believed models that I have finalized as my solution to the problem has been tuned to its best state that I could bring it to.

As to detail what I did in the classification problem, I established the problem understanding and approach of using voting classifier as my solution. From there, I have built a set of 7 basic models in their default parameters that are meant for classification problems. Next, I shortlisted the models to the top 3 models that performed the best and they are models that I Intend to feed into my voting classifier. Then I ran through one round of the voting classifier with the selected models. Based on the results from the preliminary screening, selected models are then repeatedly evaluated and improved through iterations of grid searches and manually fine tuning each parameter. Lastly, the tuned models will run through a final voting classifier as my final solution.

On the other hand, I established the problem understanding and approach of using the model with the lowest RMSE as my solution for the regression problem. From there, I have built a set of 6 basic models in their default parameters that are meant for regression problems. Next, I shortlisted the models to the top 3 models that performed the best. The selected models are then repeatedly evaluated and improved through iterations of grid searches and manually fine tuning each parameter. Lastly, a comparison will be made between the tuned models to be my final model.

Summarizing and reflecting back on the performance of final models, I find that my model could not break though certain thresholds. For example, my models in the classification problem could not achieve a testing accuracy of 85% and beyond. On the other hand, my regression models are unable to get a testing RMSE below 60. In my opinion, it felt that something could have been done or improved to achieve those goals and I can think of plausible reasons and areas to improve on. Firstly, I think both datasets could be the problem. As there could be a lot of noise in both datasets, which there isn’t anything I can do about it. However, if it wasn’t noise, it is likely due to how I cleaned and prepare my data. Which is an area I can work to improve on with more experience. The other plausible reason is that the decisions I made in this assignment was not the optimal or right choice. There are still a lot of things that I have yet to experience or understand in the predictive modelling world. If I were to have more knowledge or experience, I could have perhaps made better decision and create a better modelling process. To work on both of these technical problems, I will need more exposure in crafting more predictive model solutions or work on more similar problems that are found in this assignment.