**COMP9417 Machine Learning Project Report**

**Customer Satisfaction**

*Submitted in 2021.7.31*

|  |  |  |
| --- | --- | --- |
| Dong AO | z5305320 | z5305320@ad.unsw.edu.au |
| Jiamin Chang | z5254057 | z5254057@ad.unsw.edu.au |
| Haonan Zhong | z5289637 | z5289637@ad.unsw.edu.au |
| Naixin Li | z5376657 | z5376657@ad.unsw.edu.au |

**1.Introduction**

The rapid development of the service industry requires companies to provide higher quality services while also making consumers have more stringent requirements for service quality. In addition, from frontline support teams to C-suites, customer satisfaction has become a key measure of success. For a company, the ability to understand customer satisfaction and improve service in a timely manner is the key to retaining customers. But it is always hard to understand the customers’ dissatisfaction in advance, since unhappy customers rarely express their dissatisfaction before they leave. Considering this case, research teams have turned to machine learning to help predict the customer’s dissatisfaction in advance.

A suitable chance to apply machine learning to such a research field can be found in the <https://www.kaggle.com/c/santander-customer-satisfaction/overview> on Kaggle (2016). The aim of this competition is to construct a classification model to identify satisfied customers. And in this competition, we will need to handle hundreds of anonymized features to make a prediction. The data was provided by Kaggle. Our basic approach to address the problem is to pre-process the data first, then select the appropriate models for the dataset, finally we will train the models to achieve our goals.

We choose the problem because it is a classic classifier problem, where we can apply a lot of different models which we learnt in COMP9417. Through that, we can have a deeper knowledge over the basic concepts and the differences between the models. Meanwhile, the it is a binary classifier problem with over 300 features. The problem can also improve our understanding of dealing with dataset, which is also an important aspect in model training. It will concern with data pre-process (reasonable data split), feature selection methods and model selection in Machine Learning, and for some certain model, the adjustment of hyperparameters is also concerned.

In this project our team is focusing on constructing a binary classification model to classify whether the customer is happy with their banking experience or not. The evaluation metric of our project will be the average of the ROC AUC of the predicted result (happy/unhappy), consistent with Kaggle's evaluation metric.

**2.Implementation**

In this section we discuss those concepts for building the models, including data split for training and testing, feature selection by its importance, reviewing those learning algorithms that are suitable for this task, the important hyperparameter for those algorithms, the choice of model evaluation metric grid search, and how we use cross validation to select algorithms and models. Those implementations were done in Python 3.8. The overall implementation flow figure is in Appendix 6.1.

**2.1 Data split**

In this section, the methods we will use to split the data will be introduced and the importance of them will be fully discussed.

**2.1.1 Splitting the data to two sets**

It is necessary to divide the data into a training set and a test set to evaluate the data in machine learning. Usually, most of the data is used for training and a relatively small part of the data is used for testing. The training set will be reused to find the best model. Then, we will evaluate the best model’s performance on the test set, which has never been accessed. This will avoid the model from over-fitting or under-fitting our whole dataset. We will choose 70% of our whole dataset as the training set and the rest would be test set.

**2.1.2 Splitting the data in shuffle and maintaining distribution rate**

Using random splits can help us avoid bias when splitting data for training and testing. It can solve the problems causing by dataset with some strange data distribution. For example, the shuffle can address the problems when the data slices with the same specific target value are put together. If shuffle is not applied, we are likely to choose the test set with all the same target value, which will cause the model to fail. For the data we got, we will use the pre-built function train\_test\_split in Numpy to split the train and test set with random seed 888. Moreover, maintaining the same distribution rate of target values in train set and test set are important here. Thus, we use the function “stratify” in Numpy to achieve this.

**2.2 Feature selection**

Feature selection is to select a subset of relevant (have better importance) features from the total features of a data set before we build models. When there are large numbers of features as in this Kaggle challenge we meet, feature selection will help to improve accuracy and build simpler models that are easier to interpret, also the training time will be decreased.

Then we tried to select the features, and create those four subsets of the data set with different numbers of features using:

1. Original set

2. Data set after removing constant, Quasi-constant features, and duplicate features.

3. Data set after removing features which are highly correlated

4. Data set after using Lasso to select features

**2.2.1 Original set**

Since the original dataset retains all features, it may help models have a better fitting performance, but it may also cause the overfitting problem. Taking this into consideration, we want to keep the original set and use its fitting results as a control against other feature choices and make sure other feature selections are not underfitting.

**2.2.2 Removing constant, Quasi-constant and duplicate features.**

**2.2.2.1 Constant feature removal:**

Constant features are those features that have only one value for all outputs or target values in our data set. These are redundant data available in the data set since these functions do not provide any useful information to target prediction.

**2.2.2.2 Quasi-constant feature removal:**

Quasi-constant features are almost constant features that have the same value for very large output subsets. These features are not extremely useful for making predictions; thus, we will remove those features with variance less than 0.1.

**2.2.2.3 Duplicate feature removal:**

Duplicate features are the data sets that may contain one or more features that show the same value in all observations, which means that these two features are the same, so we will take the first column and delete the second duplicate column.

**2.2.3 Feature selection based on Pearson Correlation**

Correlation is a statistical method that can measure the relationship of 2 or more variables and show how strongly pairs of variables are linearly related. There are several different correlation techniques. Some of the most important correlation is Pearson Correlation. We will use Pearson Correlation in our implementation. The technique works well when distinguishing linear relationships: when one variable increases, the other one changes in the same trend. The value of 1 means positive correlation and -1 means negative correlation. If the value is close to 0, it means the variables do not have too much relationship with each other. We will remove those variables have strong connections with others from our dataset. As the convention, we decide to set the threshold as 0.8, which means if the absolute value of two variables’ correlation value is larger than 0.8, they will be removed.

**2.2.4 Regularization by penalty term Lasso.**

Regularization is a way to avoid overfitting, lasso is one of the regularization methods that always used in industrial.

Lasso regression will add a penalty equivalent to the square of the magnitude of coefficients. That could restrict the growth of weight, and those features that have less importance will have a zero coefficient.

For those characteristics, as this challenge is a binary classification problem, we chose to use the logic regression model using the penalty term l1 – lasso regression, using a small batch (first 10000), training it and recording those coefficients of features. The features which are not selected by Lasso will be removed as it is not being selected as an important feature by lasso penalty. We will use this to mask our data, to give the data set with the most important features.

**2.3 Suitable learning algorithms**

After we analysed the dataset and chose different feature selection methods, now we decide to find some appropriate algorithms for our dataset. We find that the target values of our dataset are 1 or 0 (unsatisfied or satisfied), which means they are discrete. Thus, we narrow down the possible options of algorithms to classifier models instead of regression models since it is hard to use regression models to fit a dataset which only has two labels. Considering the size of features (370 different features), we decide not to use KNN since too large dimensions in our dataset will result in meaningless distance calculation in KNN model. Also, for the learning algorithm SVM (support vector machine), we decide not to choose it because the number of our data are too large, the efficiency of SVM will be very low. Based on the lecture slides of COMP9417, we choose Logistic Regression as our first model since it is the relatively simplest algorithm in classifier models. Then, we decide to choose the Decision tree, the Random Forest and Naïve Bayes as our training models. Due to the advanced implementation of these models, we will train all of them by using the python library SKLEARN. The detailed descriptions of the models are shown in Appendix 6.2

**2.4 Hyperparameter tuning for rough model selection**

The possible values of hyperparameters need to be set before training the machine learning model. We want to determine the best set of hyperparameters that allows our model to maximize the performance. The Table 1 is a summary of the hyperparameters of each learning algorithm. Since the Gaussian Bayesian model does not have much room for improvement, we choose not to adjust his parameters and use the default one directly.

Grid search is used to implement hyperparameter tuning, we will create a model for each hyperparameter combination, it will try each value combination of this grid, then use cross-validation to evaluate the model. Finally, we will obtain the best model with the best hyperparameters.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Hyperparameter | Effect | Value test |
| Logistic  Regression | C | The parameter C is the inverse of regularization strength in Logistic Regression. Regularization is necessary as it can help reduce the complexity of the model and the risk of overfitting. The smaller the value of c, the stronger the regularization. | C values ranging from C = 0.2 to C = 3 in 5 sized increments |
| DecisionTree  Classifier | max\_depth | The maximum depth of the tree, when the default max\_depth is None, scikit-learn will expand the node until all the leaves are pure, or until all leaves contain less than min\_samples\_split samples. The deeper the tree grows, the more complex the model. Reducing the number of max\_depth can help us avoid over-fitting, but too little max\_depth also leads to under-fitting, so we need to try to find the best max\_depth. | Integers from 10 to 50. |
| RandomForest  Classifier | max\_depth | The maximum depth in each decision tree, the deeper the tree, the more it splits, and the more data it captures. We can use this parameter to limit the growth depth of each tree in the random forest. | Integers from 1 to 29. |

*Table 1: Summary of the hyperparameters of each learning algorithm.*

**2.5 Choice of model evaluation metric-> ROC AUC**

In this Kaggle challenge, the result will be evaluated on the area under the ROC curve between the probability of the predicted result and the target.

ROC is the receiver operating characteristic; it is used to determine the performance of a binary classification model and contains the true positive rate and the false positive rate.

True Positive Rate (TPR) = TP/(TP+FN)

False Positive Rate (FPR) = FP/(FP+TN)

AUC is the area under the ROC curve that provides an aggregate measurement, and it could measure the quality of the model, so this challenge chose to use this as the model evaluation metric, it is suitable for the dichotomy problem. It ranges in value from 0 to 1, and a model that makes 100% right prediction will have an auc of 1.0, 0.5 means not related at all.

In addition, we use “roc\_auc” in sklearn for quickly stimulating this.

**2.6 Algorithm selection using cross validation and grid search**

For every feature selection, we run the 4 kinds of classifier model, which are Random Forest, Decision Tree, Logistic Regression, and Naïve Bayes. By using the 5-fold cross-validation, we can get a rough estimate of accuracy for each model.

According to the performances of the models, we will choose the best model based on the cross-validation result. We will only use the most significant hyperparameter of every model to do the validation. To do the cross validation, we will use the built-in package GridSearchCV in sklearn. Based on the result, we will move on to do the detailed model optimization in **2.7**.

**2.7 Adjustment on the hyperparameter of the chosen model**

Once we select the model that performs better. We will use the same model evaluation in **2.5** and use the corresponding cross validation strategy. The detailed selection matrix is shown below in Table 2. If the best model is naive bayes, we will use the nested Cross Validation strategy since the model is relatively simple and there are not too many hyperparameters. Otherwise, we will use the normal Cross Validation method since our dataset is huge and the time cost in training our model will be unaffordable. The hyperparameters to adjust are suggested in the official sklearn website.

|  |  |  |
| --- | --- | --- |
| Models | Cross Validation Strategy | Hyperparameters to adjust |
| Random forest | Non-nested | n\_estimators: The number of trees in the forest  max\_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.  min\_samples\_split: The minimum number of samples required to split an internal node. |
| Naïve bayes | nested | var\_smoothing: Portion of the largest variance of all features that is added to variances for calculation stability.  priors: Prior probabilities of the classes. If specified, the priors are not adjusted according to the data. |
| Logistic regression | Non-nested | Penalty: The regularization expression.  C: Inverse of regularization strength.  Must be a positive float. Like in support vector machines, smaller values specify stronger regularization. |
| Decision Tree | Non-nested | max\_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.  min\_samples\_split: The minimum number of samples required to split an internal node.  max\_features: The number of features to consider when looking for the best split. |

*Table 2: Summary of the hyperparameters of the chosen model we will adjust.*

**2.8 Further try with XGBoost**

Since in every Boosting iteration, cross validation is allowed so that we can find the optimal number of Boosting iterations while using GridSearch we can only test the limited number of hyperparameters. In addition, boosting algorithm will make sure that the current base learner can meet the base condition before each turn and if the base learner cannot meet the requirement, then it will be dropped and the re-sampling method will be used to avoid the learning process terminate in advance, otherwise it will continue training on the results of the previous round which may be a huge advantage for some models. Therefore, it focuses mainly on the decrease of the bias, the training model will have a better performance score. And taking the logic of XGBoost into consideration, XGBoost constructs all subtrees from the base to the top and then does the prune in reverse direction which means that XGBoost may avoid being trapped into the local optimum.

By considering the advantage of XGBoost we choose to apply the XGBoost classifier to training the model. And we will take the same normal cross validation method as described in 2.7, we will try to improve the model focusing on adjusting these three hyperparameters with their most possible values

|  |  |  |
| --- | --- | --- |
| min\_child\_weight | The sum weight of the minimum leaf node. By increasing the value of it, the training model can avoid local optimal value and overfitting. While, if the value of this parameter is too high, the model may be underfitting. Therefore, we need to use cross validation to adjust this parameter. | Integer from 1 to 6 increase by 1. |
| max\_depth | The maximum depth of the tree. The deeper the tree is, the more probability of the tree to be overfitting, we also need to use cross validation to find the optimal choice of this parameter. | Integer from 2 to 6 increase by 1. |
| learning\_rate | The weight and learning rate of each model constructed in each iteration.This parameter effect the speed of the training process and the accuracy of the model. To increase the fitting performance, we can decrease the learning rate, however too small value of this parameter will slow down the learning speed. | Number from 0.01 to 0.21 increase by 0.05 |

*Table 3: Summary of the hyperparameters we will adjust for XGBoost.*

**3. Experimentation**

**3.1 Exploratory data analysis and splits**

**3.1.1 Splits for training and testing data**

This Kaggle challenge only provides a labelled training set and a test set with no labelled, so we tried to split the training data and create our own testing set. The training set contains 76020 observations and includes 370 features with 1 target. We split the data in shuffle with 70% as the training set and 30% as the testing set.

As shuffle is used, the proportion for the original data set and our training and testing set will be nearly the same.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Original Set | Training Set (70%) | Testing Set (30%) |
| Total N | 76020 | 53214 | 22806 |
| Label “0” | 73012 | 51108 | 21904 |
| Label “1” | 3008 | 2106 | 902 |
| Proportion of “1” | 3.96% | 3.96% | 3.96% |

*Table 4: Summary of data split result.*

**3.1.2 The coefficient of the original dataset.**

As the data set has 370 features which means a lot, to discover the relationship between features and the importance of each feature (since it cannot be shown by the name of features), we tried to use a heatmap to see the correlation of features. From this heatmap in , we find that our dataset is very sparse, and a portion of the features are almost uncorrelated with others, and some features are purely constant. As a result, feature selection will be very important for this challenge.



*Figure 1: Heatmap for the original dataset.*

**3.1.3 Feature selection**

After feature selection, the number of features needed to be considered for each set decreased steadily. Lasso penalty will give the smallest feature set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Original** | **Selected** | **Corr** | **Lasso** |
| **Feature Num** | **370** | **251** | **167** | **111** |

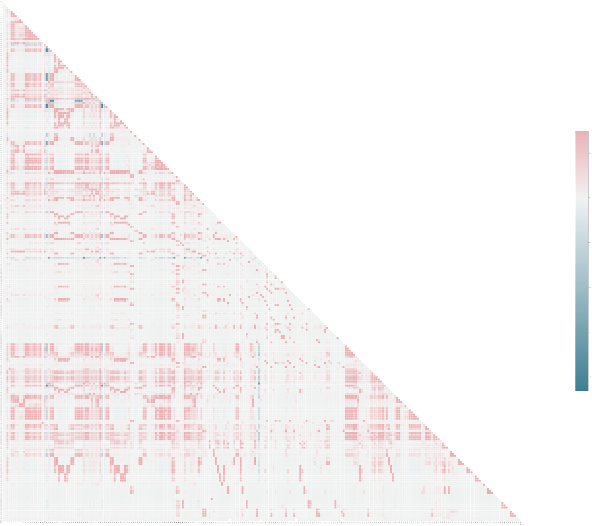
*Table 5: Summary of the feature selection result*

**3.1.3.1 Original set**

The Original set kept 370 features for comparison efficiency and correctness.

**3.1.3.2 Data set after removing constant, Quasi-constant features, and duplicate features.**

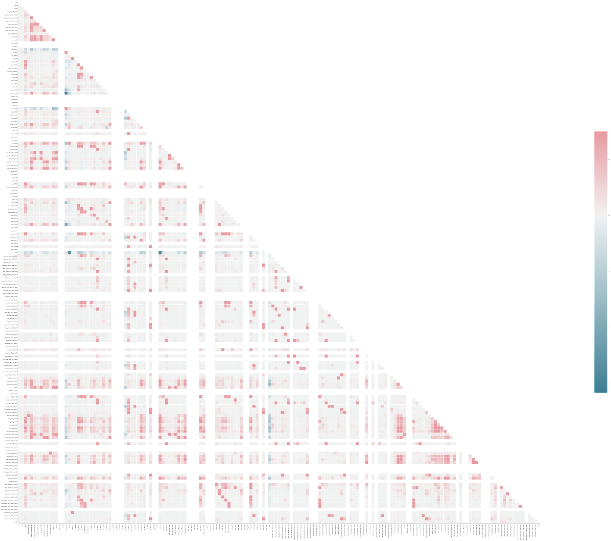
We first performed constant removal on 370 feature sets in the original data, we found there were about 40 constant features need to be removed, and then we kept 330 features, next, we did Quasi-constant features removal based on that and we deleted 61 features and kept 269 features, after that, we performed the duplicate features removals and deleted 18 features and kept 251 features as our final training set.



*Figure 2: Heatmap for the selected features*

**3.1.3.3 Data set after removing features which are highly correlated.**

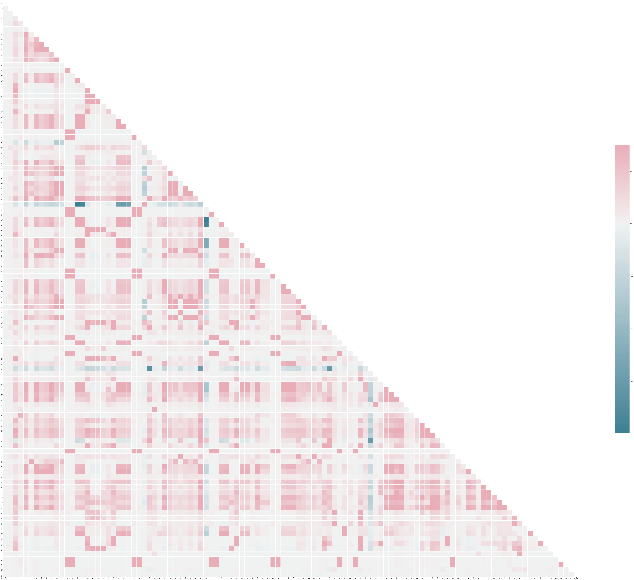
To do the feature selection by removing the highly correlated features, we set the threshold to 0.8, which means if any feature’s correlation value is larger than 0.8. It will be removed from the dataset. We end up with 167 features and get 203 features which are highly correlated.



*Figure 3: Heatmap for the correlated features*

**3.1.3.4 Data set after using lasso to select features**

We used the first 10000 data to be a batch and apply the l1 – lasso penalty for a logistic regression, using the SelectFromModel method from sklearn. After training it, we saw there were about 259 features that had coefficient downs for zero, and the SelectFromModel method returned there are 111 features that get support, those features has non-zero coefficient and seems has better importance, we mask our training set using this 111 features, the heatmap below is how those features correlated, the lasso penalty will automatically remove those features that is constant, and quasi-constant, and leave those features that affects the labels the most.

  
*Figure 4: Heatmap for the dataset after using lasso to select features*

**3.2 Cross validation results (algorithm selection) with each feature selection**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Origian | | Selected | | Corr | | Lasso | |
|  | auc | std | auc | std | auc | std | auc | std |
| tree | 0.80877 | 0.00706 | 0.80680 | 0.00756 | 0.81090 | 0.00673 | 0.81010 | 0.00688 |
| forest | 0.82172 | 0.00784 | 0.83052 | 0.00986 | 0.82683 | 0.00879 | 0.83032 | 0.00807 |
| gaussian | 0.51439 | 0.00190 | 0.51896 | 0.00241 | 0.51881 | 0.00278 | 0.69417 | 0.08295 |
| Logistic Regression | 0.57948 | 0.00795 | 0.57925 | 0.00004 | 0.57349 | 0.00637 | 0.60134 | 0.00541 |

*Table 6: Summary of the cross-validation result*

The 5-fold cross validation scoring results of four chosen learning algorithms are summarised above. The goal of the grid search in this part is trying to use the same draft hyperparameter method on different learning algorithms and basing on four learning algorithms performance scores to choose the optimal learning algorithm. All those four data sets have been experimented roughly for choosing the best parameter that will give the best score, so that we can assume the influence of the hyperparameters can be decreased when choosing the most suitable algorithm for this problem. The measurement criteria' of the performance score are the mean and standard deviation test score of the AUC among 5 folds. A most optimal learning algorithm for this data set should have a high mean test score and relatively low variance which means this algorithm can fit the data set better. From the point of view of training cost, it makes more sense to choose a model that fits the data set better for the further cross validation about hyperparameter composition selection.

From the results of our output, the optimal learning algorithm for our data set is random forest. As we can see from the above table, random forest has the highest mean score for all feature selections and at the same time it also has a lower standard deviation among those learning algorithms.

According to the analysis above, we decide to use random forest to do the detailed cross validation for all feature selections. The detailed results are shown in Appendix 6.3.2.

**3.3 Normal cross validation results (hyperparameter tuning for the selected learning methods)**

After **3.2** we found that random forest can work the best for each data set, we tried to improve the model focusing on adjusting these three hyperparameters.

|  |  |  |  |
| --- | --- | --- | --- |
| RandomForest  Classifier | n\_estimators | The number of trees in the forest, the greater the number of trees, the better the performance, but the more complex the model will slow down the code. To make our predictions more accurate and stable, we need to constantly debug to get the highest value that our processor can successfully handle. | Integers from 40 to 150, increments by 10 |
| max\_depth | The maximum depth in each decision tree, the deeper the tree, the more it splits, and the more data it captures. We can use this parameter to limit the growth depth of each tree in the random forest. | Integers from 1 to 30, increments by 1 |
| min\_samples\_split | The minimum number of data points placed in the node before the node splits. To avoid overfitting of the model, we can reduce the number of splits that occur in the decision tree by increasing the value of min\_sample\_split. | Integers from 50 to 200, increments by 10 |

*Table 7: Summary of the hyperparameters we adjusted for Random Forest Classifier*

In 3.2, we had experiments for the influence of max\_depth for the Random Forest Classifier, in this part, rough experiments have been done for the hyperparameters: n\_estimators and min\_samples\_split, the results are in the **Appendix 6.3.3**.

With chosen hyperparameters that gave the five best scores for each, we generated a Grid Search to find the best components for the three parameters. As a result, we did 5-fold cross validation for all the 125 components (5 for each parameter), measured the performance by the score of the AUC and took the standard deviation into consideration. The first 10 results for those four features are shown in Table 8.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature Selection | max\_depth | min\_samples\_split | n\_estimator | mean\_test\_score | std\_test\_score | rank |
| Original | 24 | 160 | 150 | 0.83059448 | 0.0089132 | 1 |
| 24 | 150 | 130 | 0.83040517 | 0.00814123 | 2 |
| 24 | 170 | 120 | 0.83025269 | 0.00864942 | 3 |
| 24 | 190 | 130 | 0.83017189 | 0.00837022 | 4 |
| 24 | 150 | 150 | 0.82998103 | 0.00876895 | 5 |
| 22 | 160 | 110 | 0.82991374 | 0.00792897 | 6 |
| 24 | 180 | 140 | 0.82982629 | 0.00886008 | 7 |
| 24 | 190 | 120 | 0.82978718 | 0.00927248 | 8 |
| 24 | 190 | 150 | 0.82978102 | 0.00881512 | 9 |
| 24 | 160 | 110 | 0.82965803 | 0.00996877 | 10 |

*Table 8: Summary of the first 10 result for original data set*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature Selection | max\_depth | min\_samples\_split | n\_estimators | mean\_test\_score | std\_test\_score | rank |
| Selected | 23 | 150 | 100 | 0.83159144 | 0.00841704 | 1 |
| 23 | 160 | 150 | 0.83147652 | 0.00820917 | 2 |
| 22 | 160 | 120 | 0.83119791 | 0.00841697 | 3 |
| 23 | 120 | 140 | 0.83119554 | 0.00774572 | 4 |
| 23 | 150 | 150 | 0.83099603 | 0.00879021 | 5 |
| 23 | 190 | 150 | 0.83097492 | 0.00852567 | 6 |
| 22 | 120 | 150 | 0.83096911 | 0.00778803 | 7 |
| 22 | 120 | 110 | 0.8309412 | 0.00988918 | 8 |
| 23 | 190 | 140 | 0.8306895 | 0.00930986 | 9 |
| 23 | 160 | 100 | 0.83065418 | 0.00909802 | 10 |

*Table 9: Summary of the first 10 result for the selected features*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature Selection | max\_depth | min\_samples\_split | n\_estimators | mean\_test\_score | std\_test\_score | rank |
| Correlation | 24 | 150 | 155 | 0.83462522 | 0.00865463 | 1 |
| 24 | 130 | 154 | 0.83455573 | 0.00934363 | 2 |
| 24 | 180 | 146 | 0.83452434 | 0.00904411 | 3 |
| 24 | 150 | 151 | 0.83439135 | 0.0087102 | 4 |
| 24 | 170 | 155 | 0.83432897 | 0.00898611 | 5 |
| 23 | 170 | 151 | 0.83430129 | 0.00804885 | 6 |
| 24 | 150 | 152 | 0.83429832 | 0.00758221 | 7 |
| 23 | 190 | 148 | 0.83418274 | 0.00833147 | 8 |
| 23 | 150 | 145 | 0.83417398 | 0.00877743 | 9 |
| 24 | 180 | 154 | 0.83413655 | 0.00769474 | 10 |

*Table 10: Summary of the first 10 result for correlation features*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature Selection | max\_depth | min\_samples\_split | n\_estimators | mean\_test\_score | std\_test\_score | rank |
| Lasso | 22 | 150 | 140 | 0.835528 | 0.00907951 | 1 |
| 21 | 130 | 140 | 0.83475878 | 0.0081908 | 2 |
| 21 | 170 | 150 | 0.83464521 | 0.00797539 | 3 |
| 22 | 180 | 120 | 0.83462172 | 0.00814862 | 4 |
| 21 | 130 | 150 | 0.83460011 | 0.00826773 | 5 |
| 20 | 190 | 130 | 0.83455604 | 0.00852211 | 6 |
| 21 | 150 | 130 | 0.83454621 | 0.00850438 | 7 |
| 22 | 190 | 130 | 0.83453172 | 0.00752379 | 8 |
| 22 | 190 | 150 | 0.83443761 | 0.00856228 | 9 |
| 22 | 180 | 140 | 0.83439772 | 0.00908414 | 10 |

*Table 11: Summary of the first 10 result for lasso features*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rank | mean test score | std test score | max depth | min samples split | n\_estimators | Feature | Train\_AUC | Test\_AUC |
| 1 | 0.83462522 | 0.008655 | 24 | 150 | 155 | Corr | 0.916606 | 0.831029 |
| 2 | 0.835528 | 0.00908 | 22 | 150 | 140 | Lasso | 0.912587 | 0.830551 |
| 3 | 0.83159144 | 0.008417 | 23 | 150 | 100 | Selected | 0.913363 | 0.828318 |
| 4 | 0.83059448 | 0.008913 | 24 | 160 | 150 | Original | 0.911112 | 0.826416 |

*Table 12: Summary of the best result after feature selection*

The Table 11 shows that data set after removing features which are highly correlated will perforsmed the best. Between the datasets with different feature selection methods, we noticed that the differences between the scores are small. Meanwhile, the hyperparameters of the best models are close, which is the same as what we expect. Random Forest Classifier is doing random feature selection during the training time. Thus, no matter how we select the features beforehand, the features that the Random Forest classifier are using will be similar. However, the smaller number of features will decrease the training time without decreasing correctness, so that feature selection is still a good method for training in random forest.

**3.4 Further XG Boost experimentation result**

Taking the advantages of XGBoost as what has been discussed in 2.8, we experimented with three common parameters with higher impact. The results are listed below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| feature\_selection | max\_depth | min\_child\_weight | learning\_rate | mean\_test\_score | std\_test\_score | rank |
| original | 4 | 4 | 0.11 | 0.83970293 | 0.00763948 | 1 |
| 4 | 5 | 0.11 | 0.83969916 | 0.00731794 | 2 |
| 4 | 2 | 0.11 | 0.8394387 | 0.00800385 | 3 |
| 4 | 1 | 0.11 | 0.8393537 | 0.00657072 | 4 |
| 6 | 4 | 0.06 | 0.83919908 | 0.00721108 | 5 |
| 5 | 4 | 0.11 | 0.83913972 | 0.00725944 | 6 |
| 3 | 5 | 0.11 | 0.83908195 | 0.00718697 | 7 |
| 5 | 3 | 0.11 | 0.83906551 | 0.00696831 | 8 |
| 3 | 2 | 0.11 | 0.83898985 | 0.00696775 | 9 |
| 3 | 5 | 0.16 | 0.83896537 | 0.00672339 | 10 |

*Table 13: Summary of the best result for the original dataset*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| feature\_selection | max\_depth | min\_child\_weight | learning\_rate | mean\_test\_socre | std\_test\_score | rank |
| Selected | 4 | 4 | 0.11 | 0.83970293 | 0.00763948 | 1 |
| 4 | 5 | 0.11 | 0.83969916 | 0.00731794 | 2 |
| 4 | 2 | 0.11 | 0.8394387 | 0.00800385 | 3 |
| 4 | 1 | 0.11 | 0.8393537 | 0.00657072 | 4 |
| 6 | 4 | 0.06 | 0.83919908 | 0.00721108 | 5 |
| 5 | 4 | 0.11 | 0.83913972 | 0.00725944 | 6 |
| 3 | 5 | 0.11 | 0.83908195 | 0.00718697 | 7 |
| 5 | 3 | 0.11 | 0.83906551 | 0.00696831 | 8 |
| 3 | 2 | 0.11 | 0.83898985 | 0.00696775 | 9 |
| 3 | 5 | 0.16 | 0.83896537 | 0.00672339 | 10 |

*Table 14: Summary of the best result for the selected features*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| feature\_selection | max\_depth | min\_child\_weight | learning\_rate | mean\_test\_score | std\_test\_score | rank |
| Correlation | 4 | 5 | 0.11 | 0.83995951 | 0.00749984 | 1 |
| 3 | 5 | 0.11 | 0.83961024 | 0.00703222 | 2 |
| 4 | 3 | 0.11 | 0.83959971 | 0.00644701 | 3 |
| 4 | 4 | 0.11 | 0.83954314 | 0.00685336 | 4 |
| 5 | 4 | 0.06 | 0.83932772 | 0.00719501 | 5 |
| 5 | 3 | 0.06 | 0.83929935 | 0.00671488 | 6 |
| 5 | 5 | 0.06 | 0.83915605 | 0.00699618 | 7 |
| 4 | 1 | 0.11 | 0.8391471 | 0.00746448 | 8 |
| 6 | 5 | 0.06 | 0.83904704 | 0.0072227 | 9 |
| 3 | 3 | 0.11 | 0.83904285 | 0.00744932 | 10 |

*Table 15: Summary of the best result for the corrected features*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| feature\_selection | max\_depth | min\_child\_weight | learning\_rate | mean\_test\_score | std\_test\_score | rank |
| Lasso | 4 | 5 | 0.11 | 0.84068076 | 0.00788544 | 1 |
| 4 | 4 | 0.11 | 0.84032059 | 0.00713836 | 2 |
| 5 | 2 | 0.11 | 0.83992686 | 0.00723865 | 3 |
| 5 | 5 | 0.06 | 0.83954903 | 0.00729346 | 4 |
| 5 | 4 | 0.11 | 0.83952956 | 0.00712294 | 5 |
| 4 | 2 | 0.11 | 0.83945007 | 0.0075662 | 6 |
| 5 | 4 | 0.06 | 0.83936805 | 0.00677918 | 7 |
| 3 | 4 | 0.11 | 0.83935175 | 0.00702467 | 8 |
| 5 | 1 | 0.06 | 0.83925883 | 0.00710545 | 9 |
| 4 | 1 | 0.11 | 0.83910802 | 0.00704123 | 10 |

*Table 16: Summary of the best result for feature after selected by lasso*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| feature\_selection | max\_depth | min\_child\_weight | learning\_rate | mean\_test\_score | std\_test\_score |
| Original | 4 | 4 | 0.11 | 0.83970293 | 0.00763948 |
| Selected | 4 | 4 | 0.11 | 0.83970293 | 0.00763948 |
| Correlation | 4 | 5 | 0.11 | 0.83995951 | 0.00749984 |
| Lasso | 4 | 4 | 0.11 | 0.83972069 | 0.00803305 |

*Table 17: Summary of the best result after feature selection for XGboost*

From the XGBoost results above we can see that using XGBoost classification promote the fitting results of each feature selection. And the same as Random Forest hyperparameter adjusting, the feature selection according to correlation has the best performance score in the end. From the table above we can find that, the performance of selected data set and original data set are the same which means that during the XGBoost learning process the constant, Quasi-constant and duplicate features may be removed automatically. Besides, the time cost of using XGBoost method is lower than Random Forest. From the aspects of both model fitting results and time cost, XGBoost has excellent ability in machine learning area, and it has become the most used method in Kaggle’s challenges. Because of the limitation of the computer hardware and time, we have not done further exploration of XGBoost, but in the further we may do more experiment on it and try more hyperparameters adjusting methods.

**4. Conclusion, learning and further work**

In this project, we used an important method which is feature selection. We used 4 kinds of dataset which came from different feature selection method to train four selected learning algorithms by adjusting several hyperparameters. The aim was to find the optimal model solution to the classification problem of Kaggle’s Customer Satisfaction Challenge.

After the hyperparameter tuning for rough model selection, Random Forest performed much better than the other three models in each dataset. The final best AUC of four feature selections is listed above. The best AUC we got was 0.830318. Thus, we decided to use Random Forest as our learning algorithm and trained the model by adjusting the hyperparameters. The adjustment of three hyperparameters: min\_samples\_split, max\_depth, n\_estimators gave us a model with corresponding best parameters. With the best model, the four datasets after feature selection produced high marks on the train and test set. The best final AUC of our model is the Random Forest model with correlation feature selection and some specific hyperparameters. The score hit 0.83724 at the Kaggle website, which made us the rank around 2800 among over 5000 groups, which is reasonable, and we are satisfied with this given the computing power we had and the size of training dataset we got.

The results we got corresponded with what we expected. Since the Random Forest can select features, the result of constant removing method did not make too much difference with the original dataset. However, dataset with removing highly correlated features or Lasso penalty did give us a better performance in Random Forest and XG Boost, which could be treat as an efficient way in multivariate binary classifier problem. Since it could remove most features which are constant and highly correlated but at the same time, it will keep more features than the dataset with Lasso feature selection. However, model (using Lasso) and dataset (using Correlation) produced some similar outcomes but in different test datasets (private and public) in Kaggle, the score showed some little differences, which can be explained as the feature selection methods fit different dataset. We treat this problem as a future work: we will combine the different feature selection methods to avoid the overfitting and underfitting problem.

In terms of reality benefits, our final model can address the problem of analysing whether the customer is satisfied with Santander bank based on customer’s experience. The process is automatically done, and we can extend this to any circumstance if that is a classifier problem. We can apply our model easily in some other problems and help with their prediction.

For future work, first we will try some new advanced methods in feature selection like information gain. Moreover, we will try more hyperparameters in Radom Forest and XG Boost method and try more combinations of different hyperparameters if computing power is supported. In addition, we can also try to implement Elastic Net method while it can regularize regression models by using the penalties from the lasso and ridge, and this technique can be seen as the combination of lasso and ridge regression methods to improve the regularization of model by learning their shortcomings. Elastic Net has 2 hyperparameters (Alpha and L1 ratio) can be turned. For instance, we can try to use the Grid search method, set the value of alpha between 1e-5 and 100 in the log-10 range and set the values of l1\_ratio from 0 to 1 with steps 0.1 or 0.0, and aim to find the hyperparameters that could provide the best result for our dataset.

**5. Reference**

Corporate Finance Institute. 2021. *Elastic Net - Overview, Geometry, and Regularization*. [online] Available at: <https://corporatefinanceinstitute.com/resources/knowledge/other/elastic-net/> [Accessed 30 July 2021].

Brownlee, J., 2021. *How to Develop Elastic Net Regression Models in Python*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/elastic-net-regression-in-python/> [Accessed 30 July 2021].

De Ville, Barry Hoboken, USA: John Wiley & Sons, Inc, peer\_reviewed Wiley interdisciplinary reviews. Computational statistics, 2013-11, Vol.5 (6), p.448-455

Engineering Education (EngEd) Program | Section. 2021. *Using Grid Search to Optimize Hyperparameters*. [online] Available at: <<https://www.section.io/engineering-education/grid-search/#grid-search>> [Accessed 26 July 2021].

Google.com. 2021. *Google*. [online] Available at: <<https://www.google.com/>> [Accessed 30 July 2021].

Hosmer, David W., Jr., et al. *Applied Logistic Regression*, John Wiley & Sons, Incorporated, 2013. *ProQuest Ebook Central*, <<http://ebookcentral.proquest.com/lib/unsw/detail.action?docID=1138225>> [Accessed 26 July 2021].

Medium. 2021. *How to tune a Decision Tree?* [online] Available at: <<https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680>> [Accessed 28 July 2021].

Medium. 2021. *Hyperparameter Tuning the Random Forest in Python*. [online] Available at: <<https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74>> [Accessed 30 July 2021].

Medium. 2021. *In Depth: Parameter tuning for Random Forest*. [online] Available at: <<https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d>> [Accessed 30 July 2021].

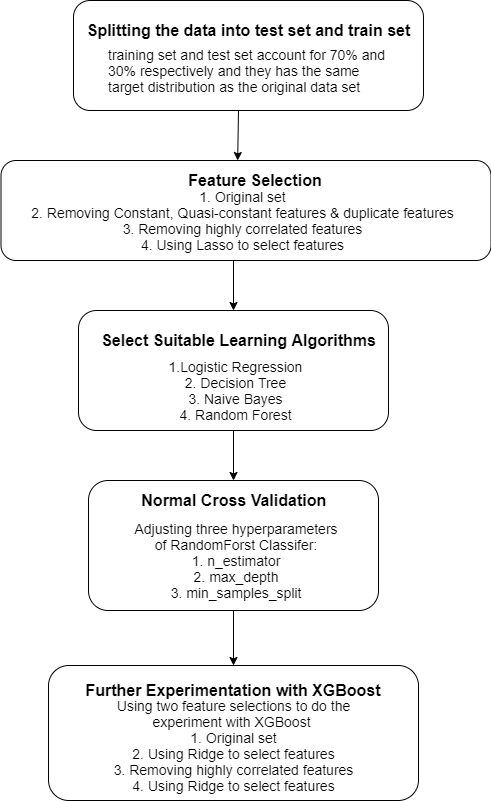
Oleszak, M., 2019. *Regularization: Ridge, Lasso and Elastic Net.* [online] Available at: <https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net> [Accessed 29 July 2021].

Scikit-learn.org. 2021. *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation*. [online] Available at: <<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>> [Accessed 29 July 2021].

Tuning, A., 2021. *Random Forest Hyperparameter Tuning in Python | Machine learning*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/ <https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/>> [Accessed 29 July 2021].

**6. Appendix**

**6.1** **The overall implementation**



*Figure 5: Summary of the overall progress for this project*

**6.2 The model selection matrix**

|  |  |
| --- | --- |
| Machine learning algorithm | Description |
| Logistic Regression | To ﬁnd the best ﬁtting and most parsimonious, clinically interpretable model to describe the relationship between an outcome (dependent or response) variable and a set of independent (predictor or explanatory) variables. The independent variables are often called covariates. The most common example of modelling, and one assumed to be familiar to the readers of this text, is the usual linear regression model where the outcome variable is assumed to be continuous (Hosmer, David W.). |
| Decision tree | The main characteristic of decision trees is a recursive subset of a target field of data according to the values of associated input fields or predictors to create partitions and associated descendent data subsets (called leaves or nodes), that contain progressively similar intra-leaf (or intra-node) target values and progressively dissimilar inter-leaf (or inter-node values) at any given level of the tree (De Ville, Barry Hoboken). |
| Random forest | Random forest is a commonly used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems. |
| Naïve Bayes | A Naive Bayes classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem. |

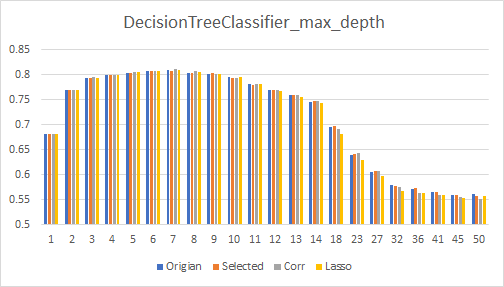
*Table 18: Summary of the model we chosen for this project*

**6.3 Results from preliminary experimentation**

In the early stage, we tried to test those models we selected by testing some different values for those important hyperparameters, selecting the best result for the model to decrease the effect of hyperparameters.

**6.3.1 Decision tree**

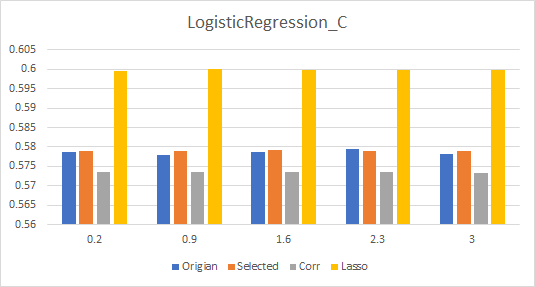
For decision tree, we choose to test the hyperparameter max\_depth, we set the length of the max\_depth from 1 to 50, we can see when max\_depth is too small, the model will be under fitting because those features does not have enough depth to classifier, however, if the depth is too large, the tree will be overfitting. In our result, most of the datasets will be trained best with max\_depth in range (6,10), we keep the best solution to compare with other models.



*Figure 6: Summary of the max\_depth we adjusted for the DecisionTreeClassifier*

**6.3.2 Logistic regression**

For logic regression, we chose to test the hyperparameter C: the inverse of regularization strength in Logistic Regression valued from 0.2 to 3. In this case, the change of this parameter is not very significant. The model: logistic regression will work better with the data set that remove features by lasso, that is caused by it has already done the penalty for weight and avoid overfitting in some place.



*Figure 7: Summary of the C we adjusted for LogisticRegression*

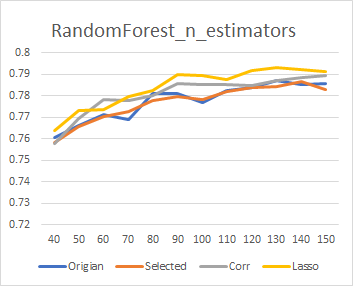
**6.3.3 Random Forest**

For the random forest, we choose the following hyperparameter: n\_estimator, max\_depth, min\_samples\_split. They limit the number of trees in the forest, the maximum depth of the tree and the minimum number of samples required to split an internal node, respectively.

As those three hyperparameters are the top three most influential hyperparameters as to the random forests

**6.3.3.1 N\_estimator**

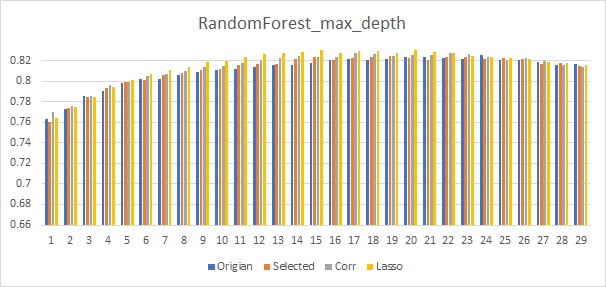
We choose to test the hyperparameter n\_estimator, we set the range of n\_estimator from 40 to 150. As we can see from the graph, the larger n\_estimator is, the better fitting performance is. However, when n\_estimator is greater than 80, the difference between scores is less than 0.01 which means that we can assume that when our n\_estimator is greater than 80, the fitting performance tends to be stable. In our result, most of datasets will be train best with the best five n\_estimators according to the feature selections.



*Figure 8: Summary of the n\_estimators we adjusted for the RandomForest*

**6.3.3.2 Max\_depth**

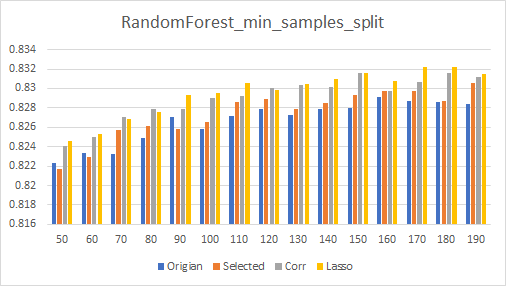
We choose to test the hyperparameter max\_depth, we set the range of max\_depth from 1 to 30. When max\_depth is small compared with data size, more leaf nodes in the tree will be pruned which can prevent some overfitting cases, however, too small max\_depth may also cause underfitting problems. Similarly, when max\_depth is too large, the model may have better performance score, but it is overfitting. Since various data selections have different data sizes, the data size from the original set, removing constant, Quasi-constant & duplicate features and selection based on Pearson Correlation to the regularization by penalty term lasso is decreasing, the optimal max\_depth has a decreasing trend. Considering this case, in our result, most of datasets will be trained with the best five max\_depth according to the feature selections.



*Figure 9: Summary of the max\_depth we adjusted for the RandomForest*

**6.3.3.3 Min\_sample\_split**

We choose to test the hyperparameter min\_samples\_split, we set the range of min\_samples\_split from 50 to 200 and the step is 10. As we can see from the graph, when min\_samples\_split is not large enough, the performance score cannot find the optimal solution, since our data size is large, if min\_samples\_split is too small when compared to the data set, it may cause the overfitting problem. However, when min\_samples\_split is too large, the performance score begins to decrease which means as the min\_samples\_split increases the model may have some underfitting problem. Since different data selections have different data sizes, the best min\_samples\_split number various, taking this into account, in our result, most of datasets will be train with the best five n\_estimators according to the feature selections.



*Figure 10: Summary of the min\_samples\_split we adjusted for the RandomForest*