Engineering Systems & Design

Term 5

2023 TAE Kaggle Competition Report

Predicting the choice of bundle of safety features in cars

Team 14

Chow Zhe Hui 1006009

Bryan Aurelius Tanaja 1006272

Afiq Bin Zulfa 1006013

Terry Yeo Teck Meng 1006129

1 Introduction

The goal of the Kaggle Competition was to predict the choice among bundles of safety features in cars.

Our group approached this problem utilizing various models that were taught in class and other models that we found through our research.

2 Approach

Our group aimed to minimize our model's logLoss. To identify the best model with the lowest logLoss, we did the following:

1. Use the basic models taught in class (Random Forest, Multinomial Logit)
2. Tune the hyperparameters (mtry, ntree, folds etc)
3. Explore more complicated models to use (Gradient Boosting via XGBoost)

2.1 Interpreting the data

Our dataset is composed of both categorical and numerical information. The majority of our effort in the modeling process was dedicated to data wrangling, emphasizing its importance. As we delved deeper into the modeling phase, it became evident that the prerequisite format of the data varies based on the model used, underscoring the significance of this aspect.

2.2 Attempts with Basic Models

We picked the Multinomial Logit and Random Forest models because they were the most relevant to the content of the competition.

2.21 Multinomial Logit Model

For the multinomial logit model, we used a very basic model and used the importance function to pick out the least important variables. We removed them from the model and made our predictions. This first attempt got us a score of 1.5498 on the public leaderboard.

Subsequently, we realized that we accidentally predicted on the train data, instead of the test data so we corrected that mistake, resubmitted and got an improved score of 1.34164.

After removing more variables, we submitted it again but did not get a better score (1.34162). Hence, we decided to focus more on other models.

2.22 Random Forest Model

For the Random Forest model, we ran once with all predictor variables and a few times with the variables removed. We used the importance() function to determine the variables. However, we found that the former performed better in terms of accuracy so we stuck with it. This gave us our best performing score.

After these attempts, we realized that there were two ways to improve our predictions: tuning our Random Forest model and exploring other models.

2.3 Improving Random Forest

Based on what we learned in class, we fine-tuned our Random Forest model by adjusting two hyperparameters: the number of trees in the forest and the number of predictors employed in each individual tree.

Increasing the 'ntree' parameter in Random Forest can improve model performance and decrease overfitting by introducing more diversity in the ensemble. However, it also increases computational complexity. Therefore, it's a trade-off: the goal is to find a balance where the model's performance improvement justifies the extra computational cost. We ran a few models while fixing other variables to find the optimum ntree value (Table 1).

Finding Best ntree	
Constant Variables	
Fold = 3	
mtry = 9	
ntree	**logLoss**
500	1.1848
700	1.1862
1000	1.1855
2000	1.1843

Table 1

Adjusting the number of predictors (mtry) in a Random Forest model helps balance bias and variance. A higher number of predictors can potentially capture more complex patterns, improving model accuracy. However, it may also overfit the training data. Conversely, a lower number might reduce overfitting but may not capture enough complexity, leading to underfitting. Therefore, tuning the number of predictors helps find the optimal balance for best model performance. Similarly, we also ran a few models to find the optimum value of mtry (Table 2).

Finding Best mtry		
Constant Variables		
Fold = 3		
ntree = 500		
mtry	**logLoss**	**logLoss (original values)**
12	1.9777	1.977690
20	1.8761	1.876100
21	1.1895	1.189492
22	1.1892	1.189217
23	1.1893	1.189258
24	1.1892	1.189216
25	1.1859	1.185926
28	1.1860	1.186001
29	1.1848	1.184838
30	1.1863	1.186344
32	1.1891	1.189091
35	1.1894	1.189394
40	1.1905	1.190450
45	1.1970	1.197009
50	1.1993	1.199334
60	1.1991	1.199146
70	1.1999	1.19986

Table 2

A key difference with our basic models is that we ran cross-validation with our subsequent Random Forest models because we wanted to improve its performance with unseen data.

2.4 Using More Complex Models

To have better predictions, we explored other models that were not taught in class: Gradient Boosting (via XGBoost) and Keras model.

2.4.1 Gradient Boosting

We employed XGBoost, a powerful implementation of gradient boosting, to enhance the accuracy and efficiency of our machine learning model. It combines decision tree models through iterative training to create a robust predictive model. XGBoost incorporates regularization techniques, like Shrinkage to control the step size of the boosting, and max depth to limit the decision tree complexity. This effectively prevents the problem of overfitting.

Since we had a large dataset, XGBoost proved efficient for predicting training and data sets. We conducted cross-validation to determine the optimal number of boosting rounds (nrounds) and trained our final model accordingly for the final prediction on the test1.csv dataset.

Despite achieving a logLoss of 1.04 on our training and test datasets, our predictions on test1.csv unexpectedly resulted in a higher logLoss of 1.55, which indicated suboptimal performance. Even though we conducted thorough research and experiments on the train dataset to improve logLoss values, we concluded that this was not the best approach for predicting the target variables.

2.4.2 Sequential Model

Since this is a classification problem, the Sequential keras model is appropriate because it allows for a linear stack of layers to be easily created. This model is particularly useful for classification problems as it enables the straightforward construction of deep learning models layer by layer.

The simplicity and ease-of-use of the Sequential model make it a good choice for not only binary, but also multi-class classification problems.

Furthermore, Keras also provides a variety of options for configuring layers, optimizers, and activation functions, allowing for fine-tuning and optimization of the model for better performance on specific classification tasks.

We used the libraries ‘reticulate’, ‘tensorflow’, and ‘keras’. The keras_model_sequential() function is used to initialize a sequential model, a linear stack of layers. This model then gets three dense layers and two dropout layers. The dense layers have a ReLU activation function (except the last one, which uses sigmoid), and the dropout layers have rates of 0.4 and 0.3, meaning 40% and 30% of the input units will be dropped during training to prevent overfitting.The model is compiled with binary cross-entropy as the loss function, RMSProp as the optimizer, and accuracy as the metric.The model is then trained (or fit) on the training data for 20 epochs with a batch size of 32.

After training, the model is evaluated on the validation set. However, due to our unfamiliarity of this model, we were not able to obtain a good score on Kaggle.

3 Results

Overall, we submitted a few models to Kaggle : Random Forest, Gradient Boosting, Multinomial Logit and Keras.

	Accuracy	
Model	**Public**	**Private**
Random Forest	1.29062	1.28325

Multinomial Logit	1.34681	1.34164
Gradient Boosting	1.55099	1.5219
Keras Sequential Model	1.32923	1.34812

From Table 3, we can see that our best model was our Random Forest model, using all predictor variables, ntree = 2000, mtry = 29 and using 10 fold cross validation. This was only possible after tuning the parameters and doing cross validation for the model.

As for other models, we were less familiar with the tuning so we were not able to optimize the predictions.

4 Interpretability

Interpretability varied for the different models.

For instance, we were able to use the importance() function to pick up the most significant variables for the multinomial logit model, but we were not able to do that for gradient boosting.

Nonetheless, the models were able to give predictions in terms of percentages for each choice, instead of a blanket binary prediction for each choice. This means that readers can look at the data and draw their own conclusion as different stakeholders may have a different threshold of prediction (i.e. some people may require a probability of 0.5 for a particular choice to execute a strategy while others may require up to 0.7).

The goal of this Kaggle competition was to minimize logLoss, in other words, they reward confident and accurate predictions while penalizing confident and wrong predictions. This incentivises the models we build to predict better but is not focused on empowering us to interpret the relationship between the predictor variables and response variable. Hence, accuracy was prized over interpretability in this context.

5 Limitations

5.1 Computing power

The training process of our random forest model was notably time-consuming due to the constrained computing power and our available hardware. The complexity of our model, that consists of over 2000 trees, set it apart from the other models that we developed. Moreover,

implementing cross-validation further extended the training time, particularly with k-fold cross-validation which we applied 3 times in our model.

5.2 Limited Time

In our approach, we opted for cross-validation instead of a single train-test split in order to enhance the robustness of our model, even though it suggested a longer execution time. By involving multiple training iterations and evaluations, cross-validation yields more reliable results and estimates, which were crucial for predictions that we conducted after building our model.

We suspect that the significant data size is a contributing factor to this situation as well. The dataset we employed contained 113 variables and 21,565 rows of data. When using cross-validation, the training time for these rows is multiplied by the number of folds, which highlights the intensive and time-consuming nature of the training of our model.

The extended computational time can also be attributed to the multiple train-test splits used to divide the dataset into smaller subsets. This procedure involves training the model on k-1 folds and then testing it on the remaining fold, which is repeated three times in our case. Considering the dataset's size, repeating this process indicates the substantial time required to run and test the model. However, since our goal was to identify the best model for predicting the target variables, we had to incur a long waiting period to thoroughly assess the overall performance of our model.