



---

COMP5310 PROJECT (STAGE 2)

---

Group Name: Group 9

Activity Number: 15

Group Number: 9



Student ID	UniKey	Report ID from Table of Contents
540371073	nisa0298	II
540348673	aaln0584	III
540349681	kpoo0923	IV

MAY 7, 2024

THE UNIVERSITY OF SYDNEY  
Camperdown NSW 2050

# Table of Contents

I.	Group Component 1 .....	2
A.	Setup.....	2
1.	Topic and Research Question .....	2
2.	Dataset.....	2
3.	Modelling Agreements .....	2
B.	References.....	3
II.	Adaptive Boosting.....	4
A.	Predictive Model .....	4
1.	Model Description .....	4
2.	Model Algorithm .....	4
3.	Model Development.....	4
B.	Model Evaluation & Optimization.....	5
4.	Model Evaluation .....	5
5.	Model Optimization .....	6
C.	Appendix .....	7
III.	XGBoost.....	8
A.	Predictive Model .....	8
1.	Model Description .....	8
2.	Model Algorithm .....	8
3.	Model Development.....	8
B.	Model Evaluation & Optimization.....	9
1.	Model Evaluation .....	9
2.	Model Optimization .....	10
C.	Appendix .....	11
1.	Algorithm Pseudocode .....	11
2.	References .....	11
IV.	Random Forest .....	12
1.	Model Description .....	12
V.	Group Component 2 .....	16
A.	Discussion .....	16
1.	AdaBoost.....	16
2.	XGBoost.....	16
3.	Random Forest .....	16
B.	Conclusion.....	16

## I. Group Component 1

### A. Setup

#### 1. Topic and Research Question

Airbags are designed to increase passenger safety in the event of accidents by reducing collision impact. There are, however, some circumstances in which airbags are more effective at mitigating the risk of injury than others. Moreover, there are extreme situations in which airbags can increase the risk of injury. Children, for instance, are generally encouraged not to sit in vehicle front rows due to their lower stature and positioning relative to airbags. Understanding the effect of airbag deployment on injury severity, across various circumstances, can help increase passenger safety. The results of this analysis are informative to various stakeholders, including vehicle manufacturers, lawmakers, drivers, and passengers. Manufacturers may, for instance, revise airbag positioning and deployment thresholds to better mitigate injury risk at high vehicle speeds. Similarly, regulators may set stricter guidelines for the design and testing of airbags systems. Understanding the circumstances associated with greatest injury risk leads to more informed drivers and consumers. In this report, we investigate how airbag deployment may affect injury severity through feature selection in a machine learning model.

#### 2. Dataset

The National Highway Traffic Safety Administration (NHTSA), an agency of United States government, maintains detailed tracking of traffic accidents and related injuries. The data is reported by first responder police officers and covers car accidents from 1997-2002. Our dataset includes a subset of the variables and observations from the original NHTSA data. The dataset includes 16 attributes and 26,217 observations, with a mix of categorical and numerical variables. The data was reconstructed by Dr. Mary C. Meyer, Tremika Finney, and corrected by Dr. Charles [1]. The authors have indicated that the dataset was merged from multiple publicly available datasets on the NHTSA website. Our dataset was sourced from vincentarelbundock Rdatasets [2]. It is worth noting that the observations of our dataset go as far back as 1997 and may be outdated in the context of our research question and business need. Furthermore, the data lacks geographical attributes, driver or passenger medical history, car model and brand, airbag sensor failure rate, conditions report, crash details, impact direction, and other characteristics. The dataset has imbalanced variables, particularly underrepresentation of severe accidents, which need to be considered in the context of predictive modelling. Furthermore, the data may include human-error considering that the observation attributes are recorded by onsite police officers. We suspect that the pattern seen between airbag deployment and injury severity has a confounding variable: impact speed. Thus, the impact of airbag deployment needs to be considered for fixed impact speeds. The following adjustments have been made to address the challenges discussed above:

- 1- Class balance can be improved by grouping classes into broader categories. Specifically, we have consolidated our classes to the following: no injury, possible injury, and severe injury.
- 2- Use stratified sampling in cross validation to account for class imbalance during training.
- 3- Remove the 'deceased' attribute, as it is highly correlated to injury severity and irrelevant to our research question.

#### 3. Modelling Agreements

Our objective is to predict injury severity, categorized as *no injury*, *possible injury*, and *severe injury*. The success measurements for our predictive models are F1-Macro Score, Matthews Correlation Coefficient (MCC), and the one-versus-rest precision-recall (PR) curves. Our research question will be analysed in the context of the Feature Importance results from each model. We have decided to use F1-Macro because it considers both recall and precision and aligns with our objective of generalizing well across all classes. We have chosen F1 due to class imbalance, and we have chosen the Macro average because it appropriately penalizes the performance when a single class has a low F1 score. On the other hand, the MCC score considers all of the values of confusion matrix FP, FN, TP, and TN. Some researchers have argued that the MCC score may be a more appropriate measure than the F1 score [3]. Finally, the one-versus-rest PR curve helps us interpret how the model balances precision and recall by class. We can estimate the model's overall performance for a given class as the area between the curve and the class's respective baseline (no-skill). Like the F1-Macro score, the PR curves can be interpreted well for unbalanced data [4].

## B. References

- [1]. M. C. Meyer, T. Finney, and C. M. Farmer, “Who Wants Airbags?,” Colorado State University, 2005.  
<https://www.stat.colostate.edu/~meyer/airbags.htm>
- [2]. Arel-Bundock V (2023). Rdatasets: A collection of datasets originally distributed in various R packages. R package version 1.0.0, Retrieved from <https://vincentarelbundock.github.io/Rdatasets/>
- [3]. D. Chicco and G. Jurman, “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,” BMC Genomics, vol. 21, no. 1, Jan. 2020. doi:10.1186/s12864-019-6413-7
- [4]. “Precision-Recall,” Precision-Recall scikit-learn, [https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_precision\\_recall.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html) (accessed May 5, 2024).

## II. Adaptive Boosting

### A. Predictive Model

#### 1. Model Description

The research question was analysed by building a decision tree based Adaptive Boosting (AdaBoost) classifier to predict injury severity and evaluating the resulting importance of the `Airbag_Deploy` variable. Decision trees were particularly attractive in this context due to their interpretability, as compared to other methods. That is, our objective is not only to predict injury severity but also to understand how injury severity is impacted by deployed airbags. Alternative methods, such as neural networks, potentially offer stronger predictive accuracy at the expense of interpretability. Key drawbacks to decision trees, however, include their general sensitivity to training data and inability to represent linear relationships. In general, we do not see clear linear relationships in our dataset. Nonetheless, these risks can be mitigated by using ensembles, such as AdaBoost.

Ensembles can improve our classifier's performance by diversifying away the errors of many independent models. In other words, we can theoretically create many "independent" decision tree classifiers, aggregate their predictions (e.g., majority vote), and achieve a stronger accuracy than we would using just one decision tree. Translating this theory to practice relies on two key assumptions: 1) the base classifiers are independent from one another and 2) the base classifiers have reasonable accuracy. AdaBoost attempts to solve for assumption #1 by constructing the base classifiers from random (weighted) samples of our training set. Assumption #2 was validated by creating a single decision tree classifier and reviewing its accuracy prior to creating the ensemble.

#### 2. Model Algorithm

Entropy-based decision trees are developed by recursively partitioning our dataset according to the values of a single feature. The splitting feature is chosen as that which maximizes information gain, where information gain is the reduction in entropy. We can continue the recursive process across branches until all examples of a given branch have the same class (maximal purity). However, the process is usually terminated early (pre-pruning) or the resulting tree is post-pruned to avoid over-fitting. This depth (# of levels) can be tuned using a key hyperparameter: `max_depth`. Other hyperparameters include `min_samples_split` and `max_features`, which specify the minimum number of samples required to split a node and the number of features to consider in looking for the optimal split, respectively. The `min_samples_split` hyperparameter effectively helps us prune the model as it is developed, while `max_features` is more relevant for datasets of higher dimension.

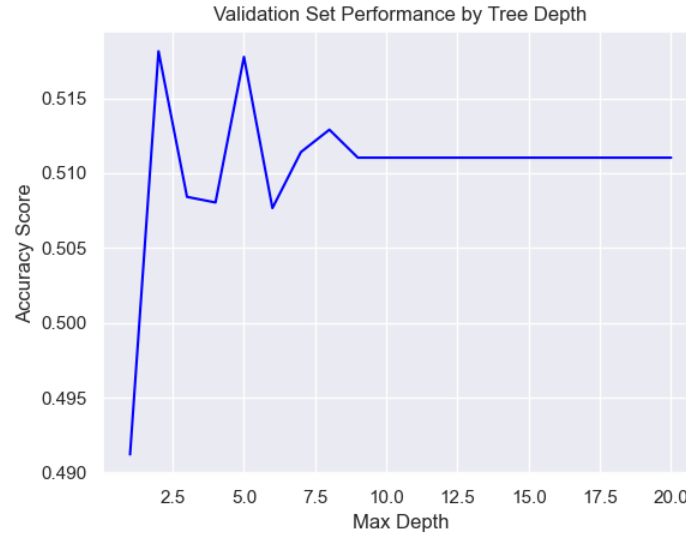
There are several alternative approaches to decision trees that use different criteria for splitting decisions. *Gain ratio*, for instance, is an approach that considers the number of branches required for a splitting decision. Attributes are penalized proportionally to the number of branches they require, thus incentivizing a narrow tree structure less prone to overfitting. Our dataset is unlikely to benefit from such an approach since our features span a relatively small set of values.

AdaBoost attempts to create independent classifiers from a subset of weighted training samples. The classifiers are created sequentially, such that samples that are misclassified by a given model are assigned a higher weight, making them more likely to appear in the subsequent model's training set. Once our classifiers are created, the classes for new examples are predicted by using a weighted vote, where the weight of a given model's prediction is determined by that model's success on its training set. The key hyperparameters of AdaBoost are `n_estimators` and `learning_rate`. In the context of our model, the `n_estimators` parameter specifies the maximum number of decision tree classifiers to construct. The `learning_rate`, on the other hand, allows us to adjust each classifier's contribution to the final prediction. There are several other boosting algorithms that resemble AdaBoost (e.g., XGBoost) but differ in how weights are updated. In contrast to most alternatives, AdaBoost has a relatively small set of hyperparameters to tune.

#### 3. Model Development

Constructing the AdaBoost decision tree classifier primarily involved feature discretization, one-hot encoding, training a base classifier, and cross-validation for parameter tuning. The decision tree classifier requires discretization of numerical features, particularly Age. All observations in our dataset were tagged as Young [16-25], Mid [26-42] or Old [43-97] by applying Pandas's `qcut()` function on Age. The remaining predictors are nominal and did not require discretization. Aside from `Impact_Level`, none of the features were ordinal and were thus one-hot encoded with new columns. The model showed improved performance from label encoding `Impact_Level` as it preserved its ordinality.

Our dataset was partitioned into training, validation, and test sets using scikit-learn’s *train\_test\_split* with stratified sampling. Stratification helps ensure that classes are fairly represented across partitions, given the class imbalance of our dataset. The next step was to create a base classifier to validate its performance as a “weak learner.” That is, we wanted to ensure that the base classifiers of our ensemble have some level of accuracy. In the case of binary classification, the base classifier should achieve accuracy over 50% (random guessing). Our multi-class base decision tree achieved accuracy and weighted-F1 score of 50% and 49%, respectively. This is likely sufficient for use in a multi-class prediction ensemble. The base classifier was built using scikit-learn’s *DecisionTreeClassifier()* with training limited strictly to the training set. As shown in **Figure 1** below, the *max\_depth* hyperparameter of the base classifier was tuned using the validation set. Based on the figure, it appears that our validation set accuracy is maximized by setting the max tree depth parameter to 5.



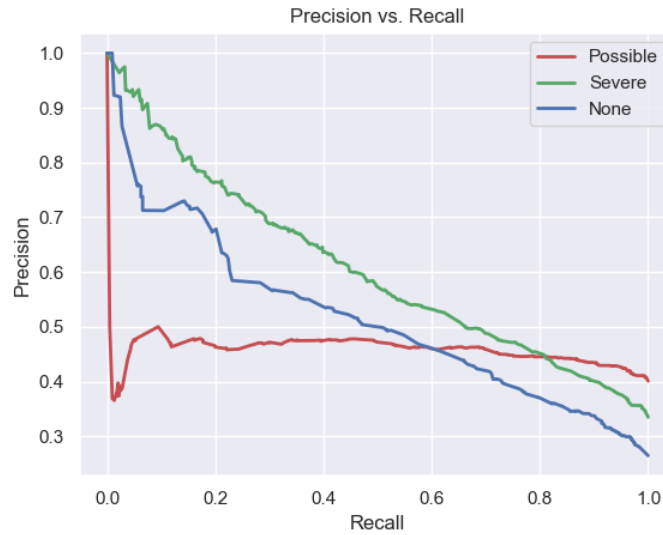
**Figure 1:** Validation Set Performance by Tree Depth

The next step was to bring the decision tree classifier into an AdaBoost ensemble and tune the AdaBoost hyperparameters. This was achieved by using scikit-learn’s *AdaBoostClassifier()* and *GridSearchCV()*. The grid search was leveraged to tune AdaBoost’s *n\_estimators* and *learning\_rate* hyperparameters and initialized to largely dispersed values. The *GridSearchCV()* was applied on the combined training and validation sets (full training set). Note that the validation set was intentionally used only for tuning the base decision tree depth, as using the test set would lead to data leakage. Cross-validation allows the grid search to determine the optimal hyperparameter combination from our grid by training multiple models using a significant portion of our full training set and comparing their accuracies on a subset of the full training set. Having left the *refit* parameter set to its default value (true), the grid search object can then be used to predict on the test set using the combined training and validation sets. The dataset contains a relatively small number of unrelated features, so most were included as predictors. Dimensionality reduction procedures, such as PCA, were largely avoided as our dataset consists of a relatively small number of unrelated features and interpretability of predictors is paramount to our research question.

## B. Model Evaluation & Optimization

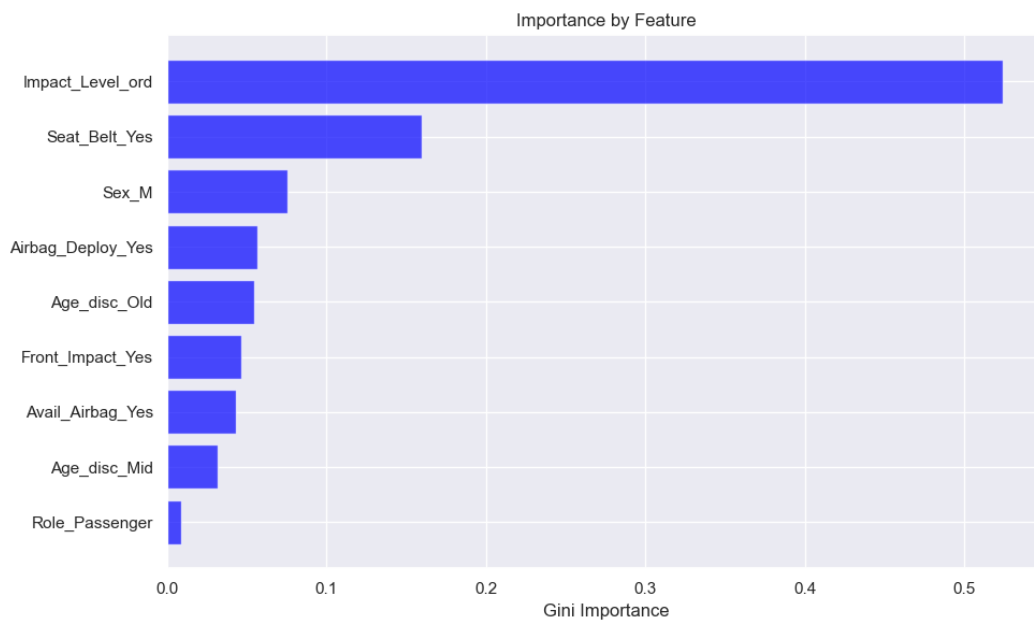
### 4. Model Evaluation

Grid search cross-validation returned optimal hyperparameter values of 250 and 0.1 for AdaBoost *n\_estimators* and *learning\_rate*, respectively. Once trained on the complete training dataset, these hyperparameters correspond to test set macro average F1 of 51%. The model also returned a Matthews Correlation Coefficient (MCC) of 0.26, suggesting weak correlation. The one-versus-rest Precision vs. Recall curves shown below corroborate these metrics. Note that the relative training set proportions for Possible, Severe, and None, are 40%, 33%, and 26%, respectively. These proportions, thought of as horizontal lines overlayed in **Figure 2**, correspond to each class’s baseline (no-skill). The limited separation between the curves and their respective baselines suggests weak performance [1].



**Figure 2:** AdaBoost Precision vs. Recall Curve

Finally, the resulting feature importance is shown in **Figure 3** below. As expected, the features most important to our classifier are impact level and whether or not the person was wearing a seat belt. Airbag deployment also appears to have moderate importance (0.05) on predicted injury severity. However, this importance should be interpreted in the context of our model's performance.



**Figure 3:** Importance by Feature


## 5. Model Optimization

The returned accuracy is very close to that of the base classifier shown in **Figure 1**, suggesting that the ensemble may be struggling to meet the independence assumption. Performance may be better with a random forest boosting model, which takes the independence assumption one step further with random feature selection across samples. Additionally, it is possible that there is some interaction between tree depth and the AdaBoost hyperparameters, yet these were tuned independently. In other words, including tree depth in the grid search may lead to a more optimal hyperparameter combination.

## C. Appendix

### 1. AdaBoost algorithm and pseudocode

Input N examples  $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ ,  $x_i \in \mathcal{X} \subseteq \mathbb{R}^n, y_i \in \{-1, 1\}$   
T: number of hypotheses in the ensemble  
Initialize  $D_1(i) = 1/N, i = 1, 2, \dots, N$

- 1: **for**  $t = 1$  to  $T$  **do**
- 2: Create a sample  $D_t$  by sampling  $D$  with replacement by taking into consideration the data points weights (as given in subsection 3.1)
- 3: Train a Weak Learner using  $D_t$  and obtain the hypothesis  $h_t : \mathcal{X} \rightarrow \{1, -1\}$  
- 4: Computed weighted error  $\epsilon_t = \sum_{i=1}^N D_t(i)_{\{h_t(x_i) \neq y_i\}}$
- 5: If  $\epsilon_t \leq 0.5$  continue else go to step (2)
- 6: Compute hypothesis weight  $\alpha_t = \frac{1}{2} \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 7: If  $t < T$ , update the data points weights:
$$D_{t+1}(i) = \frac{D_t(i)e^{-y_i\alpha_t h_t(x_i)}}{\sum_{i=1}^N D_t(i)e^{-y_i\alpha_t h_t(x_i)}}$$
- 8: **end for**
- 9: Final vote  $H(x) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$  is the weighted sum.

**Figure 1** – AdaBoost pseudocode, quoted from [2] by Sumitra, S.

### 2. References

- [1] Saito T, Rehmsmeier M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS One*. 2015 Mar 4;10(3):e0118432. doi: 10.1371/journal.pone.0118432. PMID: 25738806; PMCID: PMC4349800.
- [2] Sumitra, S. (n.d.). *AdaBoost*. Indian Institute of Space Science and Technology, Department of Mathematics.



### III. XGBoost

#### A. Predictive Model

##### 1. Model Description

XGBoost is an efficient, powerful, and flexible algorithm supervised machine learning algorithm for classification and regression. It is known for its high accuracy with tabular data, regularization ability to prevent overfitting [1], and interpretability through feature extraction. Furthermore, it can handle various datatypes, missing data [1], scaling data, and understand non-linear relationships. Not all impressive models are perfect, and neither is XGBoost. XGBoost is a complex algorithm because it has many hyperparameters, memory-intensive run due to deep trees, and sensitive to data size and its distribution. I have chosen it mainly to investigate the research question to find if the airbag deployment variable is affecting the injury severity variable.

##### 2. Model Algorithm

XGBoost implements an end-to-end scalable gradient boosting algorithm. Unlike Classification and Regression Trees (CART) models that just make a decision based on satisfied conditions in a tree, it uses ensemble methods to make a prediction. Ensemble methods in machine learning refers to the concept of using multiple models, combining their result, and outputting a single decision. Gradient Boosting creates multiple sequential trees, and it uses the error of previous tree to optimize the new tree. Hence, XGBoost combines many weak models to make a strong learner. Figure 2 shows the error propagates through the ensemble model, and the model computes a single value at the end. Not only that, XGBoost implements regularization techniques through L1, Lasso, L2, Ridge, to prevent overfitting. The crucial hyperparameters are learning rate, gamma, max\_depth, subsample. There are many other hyperparameters, but these are the ones I found out effective when optimizing for this problem. Learning rate, known as step size shrinkage (eta), is used to prevent overfitting by shrinking the features weight. Max\_depth is a parameter to set the depth of a tree, increasing it making the model prone to overfitting. Gamma is another regularization parameter that is applied through the whole tree, and it can make the tree reach higher depths without overfitting. Subsample is the ratio of training observation, and it is used to prevent overfitting. A pseudo-code of the algorithm is shown in Figure 5 that illustrates its execution step-by-step. Other recent advancements in the field of gradient boosting are CatBoost and LightGBM algorithms, and alternatively Neural Networks. CatBoost is known for its categorical features processing support out-of-the-box, usage of powerful gradient boost Ordered Boosting, and needless of extensive parameter tuning. On the other hand, LightGBM is known to be very fast because of its histogram-based algorithm to construct trees, categorical features support, gradient-based one-side sampling, and growing leaf-wise instead of tree/level-wise. I avoided neural networks because they lack interpretability compared to XGBoost.

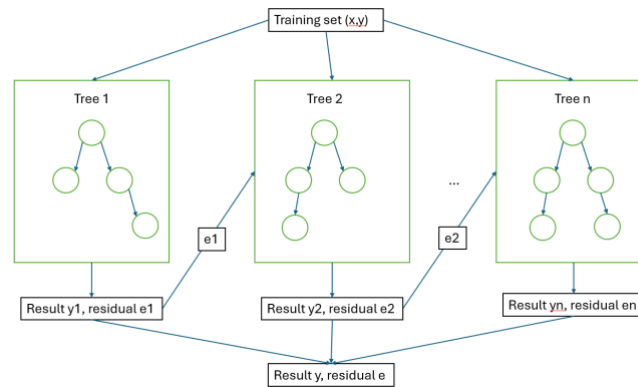


Figure 2 - Simple Iteration of XGBoost

##### 3. Model Development

First thing I did before modelling is feature engineering. I reduced the data imbalance in Year\_Vehicle variable by grouping the low count years values into one group (2001-2003), and I dropped the low count observations that have injury severity level 5 or 6 since 5 and 6 are not the focus of research due to them being “Unknown” and “Prior Death”. We have decided as a group to combine level 1 and 2 of injury severity into one group since they both indicate possible injury to reduce data imbalance. Furthermore, we have grouped level 3 and 4 into one level called severe injury. Hence, the model predicts three classes, no injury, possible injury, and severe injury. Next, since Age variable is a categorical variable, I converted it to balanced categorical groups using pandas qcut function. According to my testing, I found grouping the Age variable makes the model better. Next, I dropped the irrelevant or made-up columns, and only kept the cars that have available airbag. Before modelling, I have encoded injury severity to labels using LabelEncoder, Impact level to ordered variable using OrdinalEncoder, and other variables to one-hot dummy variable using pandas get\_dummies function. I have done this based on the advice in Scikit-learn documentation and because the model only accepts nu-

-merical values. However, I found later that XGBoost supports categorical variables, so I stopped doing one-hot encoding and OrdinalEncoder. XGBoost categorical handling is better than manual encoding according to my testing. As for the data, I have split it into training and test datasets using `train_test_split` function. I decided to split 20% of the data for testing based on a recommendation in a paper about imbalanced multiclassification [2]. I used the `stratify` flag to keep the original distribution in both sets to help with data imbalance. Furthermore, I have used `StratifiedKFold` function to make stratified Cross Validation training sets. Therefore, 1 K-Fold is going to be held for validation each time, and the others are used for training. It is called stratified because it keeps the original data distribution in the folds and this helps with data imbalance, avoiding overfitting, and model generalization. The recommended K-Fold value is 5 or more, and I experimented with different values and settled at 5. Moreover, I created a `classes_weightage` variable to use in `fit` function to tell the model to do a balanced training on all classes. On a sidenote, I avoided PCA because we do not have many observations and variables, so we did not need PCA for data processing. I used `XGBClassifier` from XGBoost library to configure the model before fitting. For my first model fit, I kept it simple and used the default settings of Scikit-learn and XGBoost. I just added parameters such as number of classes, enabled categorical data reading, limited the categorical one-hot encoding to be 8, used `mlogloss` evaluation metric, and put early stopping value of 10. I limited it to 8 because it was the best value during my testing, and 8 means one variable maximum may be left to not be one-hot encoded, which is logical for our data. I used `logloss` evaluation metric because it is the recommended for XGBoost multiclassification, and I put early stopping to stop training if no improvement happens in 10 steps. An early stopping value of 10 is more than enough and it speeds-up the training process. Finally, I used `eval_set` parameter in model fit function so the model stores the values of training loss and validation loss.

## B. Model Evaluation & Optimization

### 1. Model Evaluation

The result I got for the measurements are: F1-Macro is 0.49, Matthews Correlation Coefficient (MCC) is 0.239, and One-vs-the-Rest (OvR) Macro ROC is 0.686. For the feature importance, the most important features are Impact speed with %29.2, seatbelt with %16.4, and airbag deployment with %15. An F1-Macro score of 0.49 is considered to be a good performance, meaning it can balance precision and recall for some classes, but it misses some. A MCC value of 0.239 shows that the model is better than a random prediction. ROC value of 0.686 shows that the model is good at distinguishing a class from the others, and it is not guessing at random. The graphs in Figure 3 shows that severe injury class is the most troublesome class for the model to predict because it has low recall value of 0.351, which indicate the model has a high false negative rate. We can see that in Figure 3 precision and recall curve, all the classes have a good result except one class, which is severe injury. The poor performance of recall in severe injury class indicates that the data may be lacking, or the model is struggling with the data. However, since the model did an acceptable job with other classes, we expect it to do that as well with this class, yet we fail to achieve this. I believe the data is missing more important variables and it has a lot of noise from human-factor measurement. The current set of variables are not significance since the model only concentrate on 3 variables to take a decision. Hyperparameters optimization can help a little bit with this problem, and we are going to see next how to improve the overall performance of model.

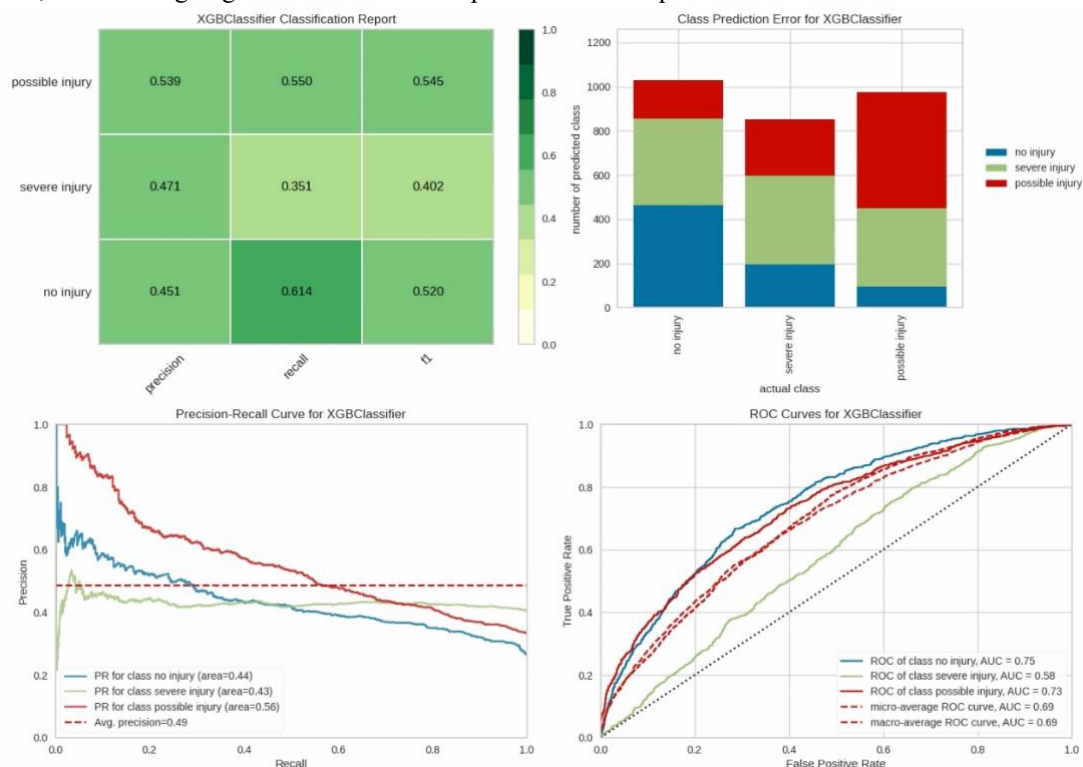


Figure 3 - XGBoost Evaluation through Confusion Matrix, Correct Predictions, PR Curve, and ROC Curve

## 2. Model Optimization

I have experimented hyperparameters tuning with two approaches. Firstly, RandomizedSearchCV, and secondly, Optuna. Both do a random search of optimal hyperparameters after specifying the hyperparameters, their range, and their distribution. I preferred to go with a random search because I have many parameters, and they can create infinite combinations. Doing an extensive search is not practical for my case as it computationally expensive, takes very long time, and does not converge to local extrema unless you search all the combinations. Both functions implement a fit and scoring methods, by that I mean model training, and optimizing the model based on a metric like F1-score. After many experiments, I found that Optuna is way better for my problem than RandomizedSearchCV because it is faster, more efficient, and can reach local minima with ease compared to RandomizedSearchCV. Optuna implements state-of-the-art search algorithms to sample hyperparameters more efficiently than just a random search algorithm [3]. First, I have looked online on articles and research related to selecting starting point of hyperparameters and I followed the suggestions in [4] and [5]. I have searched a large grid of values and I run Optuna multiple times for 30-100 trials. Every time Optuna finishes, I record the best hyperparameters and run it again. I compare around 4 runs of Optuna to reduce the search space. For example, if I have seen a hyperparameter that only converges to a number around 10, I reduce the search space of Optuna to be from 8-12. I tuned the parameters `max_depth`, `subsample`, `learning_rate`, and `gamma`. I experimented with others, but they decreased model performance. Furthermore, I fixed `n_estimators`, max created trees number, to 2000 but the model stops at around 100, so it does not really matter for this data because the max is never reaching 2000. The optimized model has an F1 increase from 0.49 to 0.5, matthews\_score pump from 0.239 to 0.251, and ROC change from 0.686 to 0.692. The optimized model relies less on impact speed, going down from 29.2% to 22.9%, and more on other variables like seatbelt, front impact, sex, age, and airbag deployment. Finally, the confusion matrix in Figure 4 shows a good increase in all values, and PR curve reflects that change. No big change is observed for ROC and prediction error graphs.

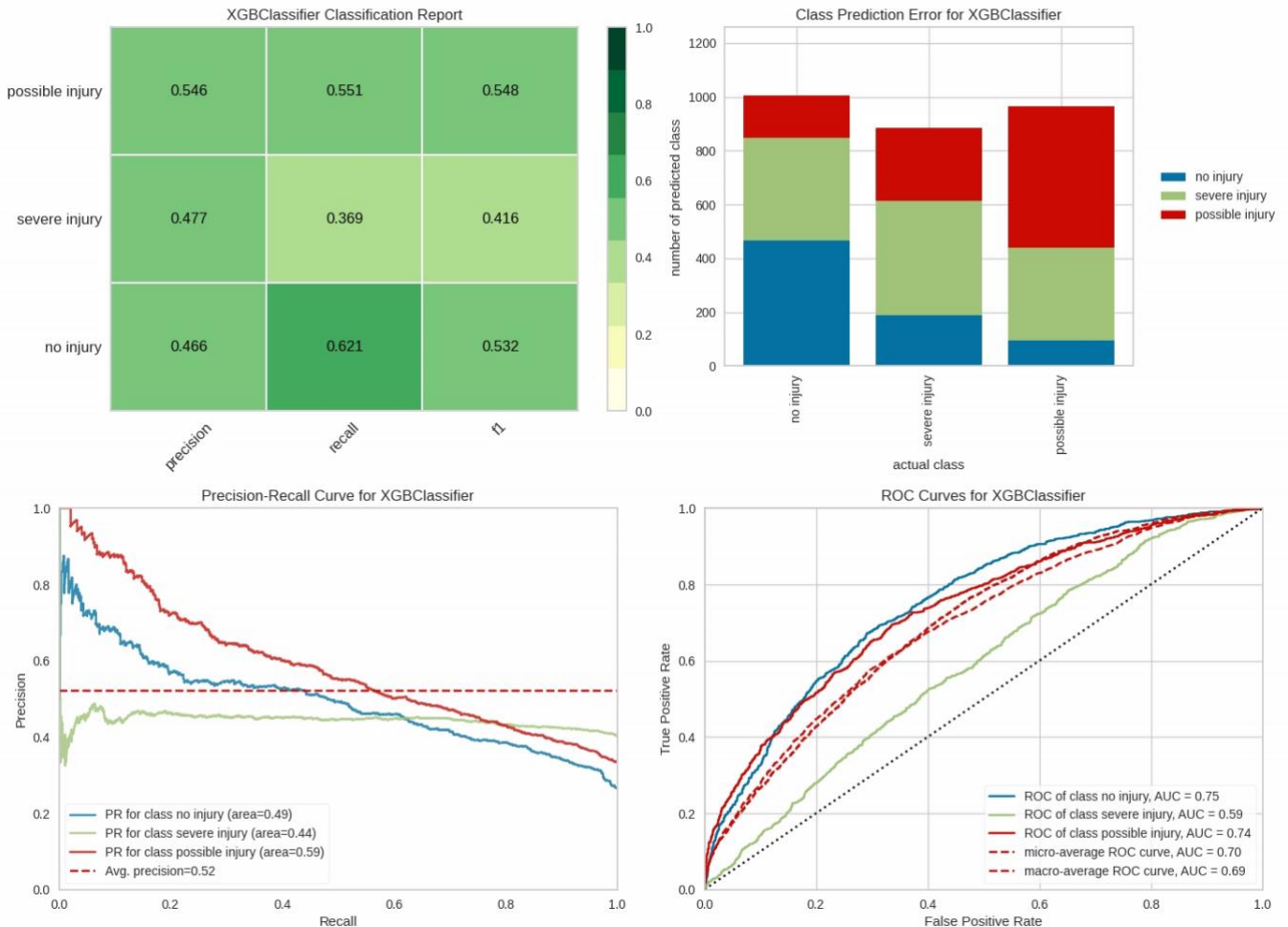


Figure 4 - Optimized XGBoost Evaluation through Confusion Matrix, Correct Predictions, PR Curve, and ROC Curve

## C. Appendix

### 1. Algorithm Pseudocode

#### XGBoost Algorithm

Input:  $\{(x_j, y_j)\}_{j=1}^M$  training set

Input: N tree (weak learners)

Input:  $L(y, F(x))$  differentiable loss function

Input:  $\alpha$  learning rate

Model initialization by using a constant value:  $F_i^{(0)}(x) = \underset{\theta}{\operatorname{argmin}} \sum_{j=1}^M L(y_j, \theta)$

for  $k = 1$  to  $N$

1. Compute gradients and Hessians:

$$g_k(x_j) = \left[ \frac{\partial L(y_j, f(x_j))}{\partial f(x_j)} \right]_{f(x)=f_{k-1}(x)}$$

$$h_k(x_j) = \left[ \frac{\partial^2 L(y_j, f(x_j))}{\partial f(x_j)^2} \right]_{f(x)=f_{k-1}(x)}$$

2. Fit a tree by means of the training set  $\left\{ \left( x_j, -\frac{g_k(x_j)}{h_k(x_j)} \right) \right\}_{j=1}^M$  by solving the following optimization problem:

$$\phi_k = \underset{\phi \in \Phi}{\operatorname{argmin}} \sum_{j=1}^M \frac{1}{2} h_k(x_j) \left[ -\frac{g_k(x_j)}{h_k(x_j)} - \phi(x_j) \right]^2$$

$$f_k(x) = \alpha \phi_k(x)$$

3. Update the model:

$$F_i^{(k)}(x) = F_i^{(k-1)}(x) + f_k(x)$$

end

Output:  $F_i^{(N)} = \sum_{k=0}^N f_k(x)$

Figure 5 – Unregularized XGBoost Pseudocode quoted from [6] by A. Panarese, G. Settanni, V. Vitti, and A. Galiano

### 2. References

- [1]. T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016. doi:10.1145/2939672.2939785
- [2]. A. Rácz, D. Bajusz, and K. Héberger, “Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification. *Molecules*,” *Molecules*, vol. 26, no. 4, p. 1111, Feb. 2021. doi:10.3390/molecules26041111
- [3]. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul. 2019. doi:10.1145/3292500.3330701
- [4]. E. Luellen, “Mastering XGBoost,” *Medium*, <https://towardsdatascience.com/mastering-xgboost-2eb6bce6bc76> (accessed May 4, 2024).
- [5]. Laurae, “xgboost: ‘Hi I’m Gamma. What can I do for you?’ — and the tuning of regularization,” *Medium*, <https://medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6> (accessed May 4, 2024).
- [6]. A. Panarese, G. Settanni, V. Vitti, and A. Galiano, “Developing and Preliminary Testing of a Machine Learning-Based Platform for Sales Forecasting Using a Gradient Boosting Approach,” *Applied Sciences*, vol. 12, no. 21, p. 11054, Oct. 2022. doi:10.3390/app122111054

## IV. Random Forest

### (A) Predictive Model

#### 1. Model Description

I have chosen random forest as a predictive model for analyzing the research problem. The random forest model is a supervised machine learning technique that makes predictions by integrating multiple decision trees. A significant advantage of random forest is that it can handle skewed and multi-class data without assuming any formal distribution and is therefore considered a non-parametric model. One of the most essential features of random forest is that it can deal with dataset with categorical data. The aim of this study was to assess the effect of airbag deployment on injury severity, where the results contained three classes. Random Forest is a fitting choice due to its robustness in handling categorical data and its suitability for multiclass classification tasks. Random forest is known for mitigating overfitting problem in decision trees, thus enhancing model accuracy while lowering prediction variance. Moreover, random forest values feature importance which provides information about the importance of each feature in the data, contributing to a better understanding of the pattern. Additionally, random forest demonstrates scalability, making it proficient at handling large and high-dimensional dataset. Random forest indeed possesses huge predictive power; however, it has its weakness too. The low interpretability makes it challenging to understand as it combines with multiple decision trees. Nevertheless, due to ensemble approach, training a random forest is time-consuming, especially dealing with more trees. Given the class imbalance and high-dimensional features of our dataset, the use of random forests allows for efficient processing of the dataset and construction of a highly accurate model.

#### 2. Model Algorithm

Random forest algorithm is a supervised learning algorithm with a direct relationship between the number of trees in the forest and the result generated from it. It contains a certain number of decision trees and predicts a precise result by voting in the classification case. Each node in the decision tree works on a random subset of features to obtain an output. Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual method. Random forest first selects a number of random subsets from the training set, then trains a decision tree for each subset. For each individual tree, generate a candidate in the test set respectively for prediction. The final prediction is depending on the majority vote for the class. To reach the optimal result of performance, hyperparameter is necessary for the process of modelling. By choosing the appropriate hyperparameter values, we can improve model accuracy. In random forest, there are several hyperparameters, '*n\_estimators*', '*max\_features*', '*min\_samples\_split*', '*min\_samples\_leaf*' and '*max\_depth*'. The key hyperparameter is '*max\_depth*', it calculated the longest path between the root node and the leaf node. '*max\_features*' indicates the number of features to consider during the splitting process. '*min\_samples\_split*' decide the minimum number of samples required to split a node in a decision tree. '*min\_samples\_leaf*' set the minimum number of samples required to be in a leaf node. '*n\_estimators*' decide the number of decision trees in the forest. The node impurity responsible for splitting the nodes into branches, in random forest, Gini impurity, Entropy and Log Loss are often used to measure the quality of the split. It can be varied according to different circumstances. Gini impurity calculates the probability of misclassifying an observation, while Entropy measures the disorder level in a set. Log Loss indicates how close the prediction probability is comparing to the corresponding actual value with penalizing inaccurate predictions with higher value. The use of three different measurements depends on the characteristic of the dataset and performance metrics of each measurement.

#### 3. Model Development

To construct a random forest classifier, the initial step is to transform raw data into a format that suitable for machine learning, with feature engineering being the priority. In this research, we are focusing on the investigation of relationship between airbag deployment and the level of injury severity. Therefore, based on this assumption, we decided to eliminate observations where avail airbags were not deployed. For random forest classifier, it is necessary to encode categorical variables into numeric representations. Initially, addressing our outcome variable, “*Injury severity*”, which originally have 8 unique values, including missing values. Given that injury severity level 5 and 6 were not determinable, we excluded them alongside the missing data. We grouped severity level 0 into “None”, combined severity level 1 and 2 into “Possible” and severity level 3 and 4 into “Severe”, successfully grouping and store the result in “*Injury\_Severity\_level*”. Additionally, as ‘Acc\_Year’ and ‘Veh\_Year’ are in string format and do not hold any predictive value to the model, they were transformed from string to integer format, with the years extracted and the difference calculated, creating a new variable, ‘Vehicle Age.’ Subsequently, I use *OrdinalEncoder* from scikit-learn to encode the ordinal variable “*Impact\_Level*”. Following the definition of the dictionary named “*Impact\_Level*” which mapped each interval of the ordinal variable to its corresponding numerical value. I applied *ordinal\_encoder:fit\_transform* to encode all the numerical value stored into ordinal categorical values to preserve the ordinal nature of the feature. Similarly, we discretize “Age” into categories, using “*pd.qcut()*” to separate “Age” data into three labelled as “Young”, “Mid” and “Old”. Once all the important features are transformed into categories that ready for the random forest classification. Then we implement one-hot encoding to convert categorical variables into binary vectors using *pd.get\_dummies()*. By using one-hot encoding, it preserves data in a simple way, and it avoids bias when the model misinterprets of ordinality in categorical variables. Then I construct the feature matrix, “X” and the target variable “y”. Using *LabelEncoder* from sklearn.preprocessing for encoding our “*Injury\_Severity\_level*” form categorical to numerical label, which is also known as class. In order to maintain the balance of training and testing dataset, I use *StratifiedKfold* from sklearn.model\_selection during the cross-validation process. Given the imbalanced class distribution in our data, *StratifiedKfold* can effectively deal with the issue as it ensures that each fold has the same distribution of class labels as the entire dataset, it allows we gain the convincing estimation of model performance across all the folds. Also employing *shuffle* to prevent bias in the cross-validation process. Finally, I split the dataset using scikit-learn’s *train\_test\_split* function to split the training and testing set with a training size of 80% and testing size of 20%. Moreover, I employ *stratify* parameter to ensure the class distribution in the target variable “y” is maintained across both sets.

The next step is to perform random forest classifier using *RandomForestClassifier* by scikit-learn, meanwhile optimizing its hyperparameter through grid search using *GridSearchCV*. The hyperparameters are defined within a dictionary “*params\_to\_test*”, with hyperparameter such as *n\_estimators*, *max\_features*, *min\_samples\_split*, *min\_samples\_leaf*, *max\_depth* and *max\_leaf\_nodes*. These hyperparameter bring determinate factors to the model that directly influencing the accuracy and optimizing. During grid search, the random forest model is presented with the hyperparameter, the scoring metric and the CPU cores. We chose f1 macro-averaged F1 score as the scoring matric because it particularly useful in handling multiclass classification problem with imbalance class. The macro-averaged F1 scores were calculated for each category and then they were averaged to ensure that the performance of each category was weighted equally, independent of its prevalence in the dataset. This helps to provide a more balanced assessment of the model's performance across all categories which is a useful scoring metric for the classification. Furthermore, by setting CPU core as “*n\_job=-1*” can effectively speed up computations that “-1” basically means *GridSearchCV* can detect the number of available CPU cores on the computer and assign tasks for all cores. Following this, X\_train and y\_train are fitted into *grid.search* to search through specified hyperparameter to obtain the best combination according to the scoring metric. PCA are not considered in this research due to its potential loss of interpretability. As random forest values feature importance, using PCA may obscure the interpretability and ignore the importance of the original feature.

## (B) Model evaluation and optimization

### 1. Model evaluation

Evaluation metrics are used for evaluating the random forest model performance. I got the macro average F1 score measuring at 0.4946 suggesting a moderately fair performance of the random forest model. It indicates that the classifier indeed achieves a certain level of accuracy and predictability across all classes, even though it is not optimal. The reason for it can be the imbalance of class distribution, in such cases, the classifier cannot predict the minority classes precisely, leading to a low F1 score. Another evaluating measurement is Matthews Correlation Coefficient (MCC), I got the MCC result be 0.2325, suggesting a weak correlation between the predicted and actual classification. It indicates that random forest classifier prediction is not significantly better than a random guess classifier, also struggling to present the pattern of the data effectively.

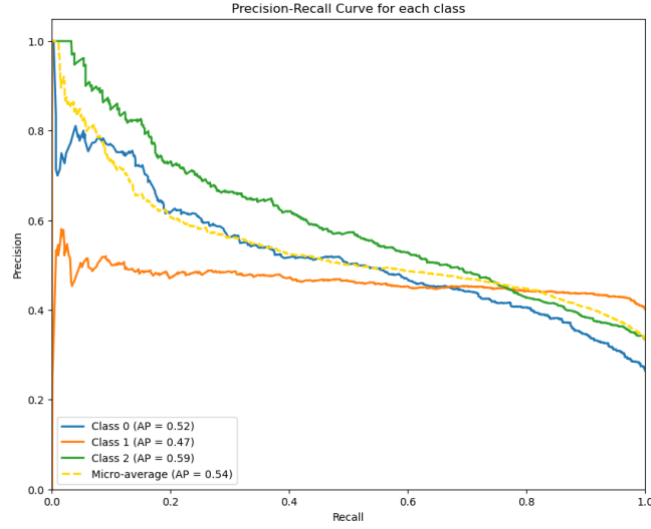


Figure 1: Precision-Recall curve for each class

Moreover, we choose one-vs-rest (OvR) Precision-Recall (PR) curves as the measurement of evaluation. For class 0, the average precision (AP) score is 0.52, indicating an acceptable performance in precision-recall trade-off. For class 1, the AP score is 0.47, indicating the worst trade-off performance. For class 2, the AP score is 0.59, indicating the best overall performance compared to other classes. Micro-average AP scores is 0.54, providing a decent performance of average precision across all classes by taking account of each prediction equally regardless of class frequency.

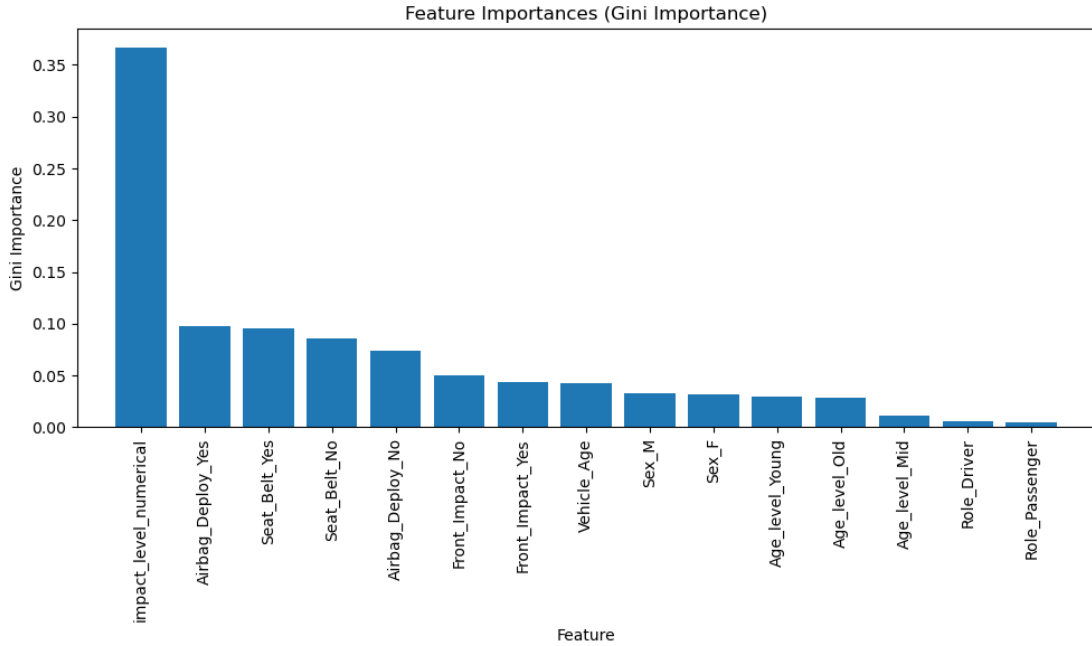


Figure 2: Bar chart showing feature importances

For random forest, a feature importance graph helps examining the relative importance of different features in the making model. **Figure 2** shows that “*impact\_level\_numerical*” has the highest Gini importance with 0.36, suggesting that impact level has a significant impact on the model, meanwhile, “*Role\_Driver*” and “*Role\_Passenger*” have the lowest Gini importance, indicating a low significance level for the model.

## 2. Model optimization

There are various approaches to optimize Random Forest models. One approach is leveraging hyperparameter optimization through grid search. However, further improvement in the model can be achieved by selecting better hyperparameters. According to a report investigating the optimal hyperparameter for random forest to predict leakage current alarm on premises, it used grid search with different hyperparameters optimization, it showed a significant increase in hyperparameters with a larger value [2]. However, the value I used in the code is relatively small compared to the report as larger the parameter value, the longer processing time. The model can be optimized with more and larger value include in the hyperparameters’ dictionary.



Another effective strategy involves feature selection, which enhances the quality and relevance of features during model training. **Figure 2** illustrates feature importance, it proves that "Role" and "Age" receive relatively low importance scores. Hence, given their low significance, excluding these features from the model could be beneficial. By prioritizing the selection of more influential features and omitting those with low relevance, it optimizes the performance of the model.

## (c) Appendix

### I) Random forest algorithm

1. For  $b = 1$  to  $B$ :
  - (a) Draw a **bootstrap sample**  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  **$m$  variables at random** from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.

2. Output the ensemble of trees  $\{T_b\}_1^B$ .

To make a prediction at a new point  $x$ :

*Regression:*  $\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$ .

*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{rf}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

Figure 3: Algorithm and Pseudocode for random forest [2] by Jo. J

### II) References

- [1]. Yokoyama, A., & Yamaguchi, N. (2020). Optimal hyperparameters for random forest to predict leakage current alarm on premises. In *E3S Web of Conferences* (Vol. 152, p. 03003). EDP Sciences.
- [2]. Jo, J. (2022, July 20). *Overview Of Random Forest*. Medium. <https://medium.com/@wjj1019/overview-of-random-forest-36a9d51f4b57>



## V. Group Component 2

### A. Discussion

#### 1. AdaBoost

The AdaBoost model performed relatively well and returned intuitive results. XGBoost, on the other hand, showed similar performance but differed in terms of feature importance. As discussed in section II, the AdaBoost ensemble showed minor performance improvement compared to the base decision tree classifier. This suggests that the model may be struggling to create sufficient independence between decision trees. In other words, the random sampling may benefit from random feature selection (i.e., random forest). The random forest classifier from section IV, however, did not show improved performance. Note that we excluded vehicle age from predictors in the AdaBoost model, since the mean imputation conducted in Stage 1 for vehicle year can lead to a negative vehicle age. Nonetheless, the model may benefit from this additional information if a different approach is taken to missing values.

#### 2. XGBoost

XGBoost is an algorithm known for high accuracy in classification problems. It has demonstrated success on many Kaggle competitions, outperforming similar models. XGBoost appears to be a suitable solution for this problem since the dataset includes enough observations and is structured as tabular data. Furthermore, its interpretability and ability to conduct feature extraction are valuable to our research question. The model development and hyperparameter tuning processes appropriately deal with imbalanced data; however, the predictive performance is moderate. Though overall performance was close to AdaBoost, the XGBoost model placed more importance on airbag deployment. Both models agree, however, on impact level (speed) and seatbelts as being the main contributors to injury severity.

#### 3. Random Forest

Random forest is a powerful ensemble learning algorithm known for its effectiveness in handling multiclass classification. A key strength of random forest is its speed, enabled by fitting the base classifiers parallelly, as opposed to sequentially (e.g., AdaBoost). However, random forest requires considerable hyperparameter tuning and benefits from a large feature set. A large feature set allows random forest classifiers to effectively create independent decision trees that can, in aggregate, reduce prediction errors. The overall performance of the random forest classifier is comparable, in terms of Macro F1 and MCC, to AdaBoost and XGBoost. Additionally, the feature importance results closely resemble those of XGBoost.

### B. Conclusion

In conclusion, we recommend AdaBoost as the most effective predictive model for our research question because it showed the strongest overall performance. Specifically, AdaBoost achieved an F1-macro score of 0.51 and MCC score of 0.26, compared to 0.5 and 0.25 of XGBoost, and 0.49 and 0.23 of random forest. All three models suggest that impact level and seatbelts are the primary factors impacting injury severity, not airbag deployment. Furthermore, the model architectures are appropriate for this problem because all three are recognized as state-of-the-art approaches to classification problems dealing with tabular data lacking obvious linear relationships. Nonetheless, even in AdaBoost, performance across all measures is moderate. All three models could benefit from additional attributes, such as geographical data, car details, condition report, crash details, observer bias, and many others. One advantage of AdaBoost is that it has relatively few parameters. However, XGBoost and random forest could likely be refined from more extensive parameter tuning.