In-Depth Analysis – Binary Classification Using LightGBM Supervised Model

## Feature Selection

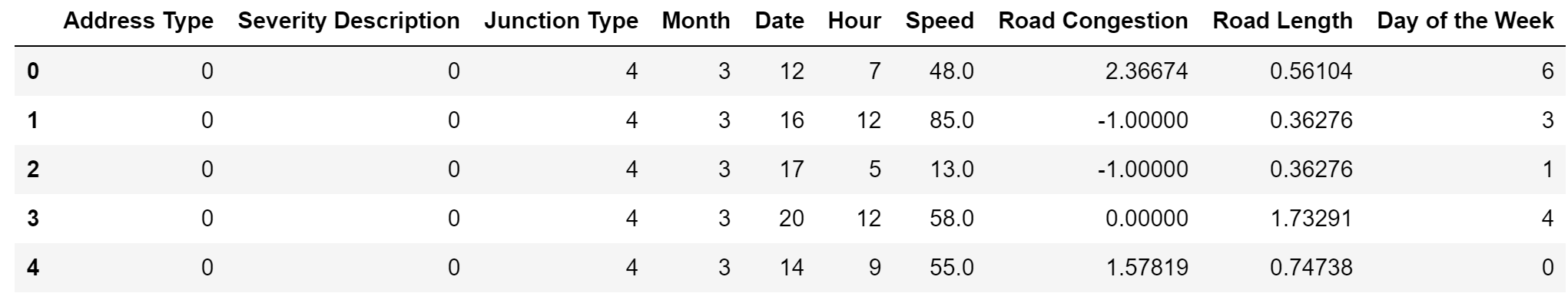
Variables that only provide information about the collision incident once it has occurred were dropped from the dataset as they could not be used for predicting the severity of future collisions. These are variables such as *Number of People Involved* and *SDOT Collision Description*.

*Weather, Light Condition, Road Condition* and *Neighborhood* were also dropped following initial modeling as they did not necessarily improve the model. In fact, removing the *Neighborhood* feature improved the model performance. The following features were retained:

* *Address Type*
* *Junction Type*
* *Month*
* *Hour*
* *Day of the Week*
* *Speed*
* *Road Congestion*
* *Road Length*
* *Severity Description*

## Pre-processing

The categorical variables were converted to discrete integers using the LabelEncoder function in Python. Features that were encoded are *Address Type, Junction Type, Month, Hour, Day of the Week* and *Severity Description*. The encoded dataframe is shown in *Fig 35*.



**Fig 35.** Snippet of dataframe with label encoding for categorical variables

Note that feature scaling was not implemented initially as our selected model was *LightGBM* which did not require the features to be scaled as opposed to an algorithm such as *Logistic Regression* which is a linear model and hence required scaling for optimum performance.

The dataset was then split into the explanatory variables (*X*) and the target variable (*Y*) for modeling and evaluation.

## Train test split

The datasets are further split into train and test samples. The train sample size was chosen as 70% of the original size. During the optimization phase, the test sample will be considered a validation sample. What we mean by this is that the train sample would be used for parameter tuning and cross-validation where it will be split into several random train and test samples in order to determine the best average score.

## Oversampling of minority class

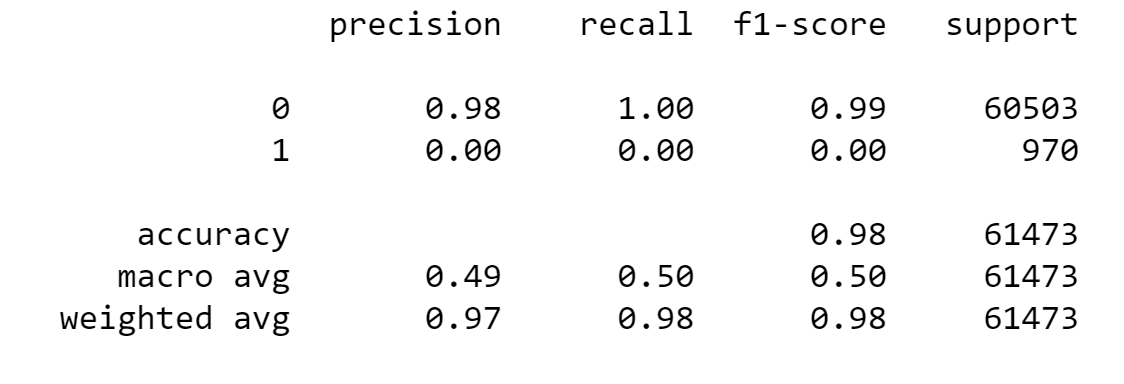
In the case where the weather and neighborhood variables were included, *Synthetic Minority Oversampling Technique (SMOTE)* improved the model performance (ROC AUC score). SMOTE creates synthetic data points like the minority instances, making the representation of the two classes nearly the same whereby fixing the imbalance issue to an extent.

However, oversampling was not used for the final baseline model as it decreased the scoring metric. This is not surprising as oversampling only works well when the initial model is robust to begin with. In our case, the class imbalance was severe making oversampling ineffective.

## Modeling, prediction and evaluation

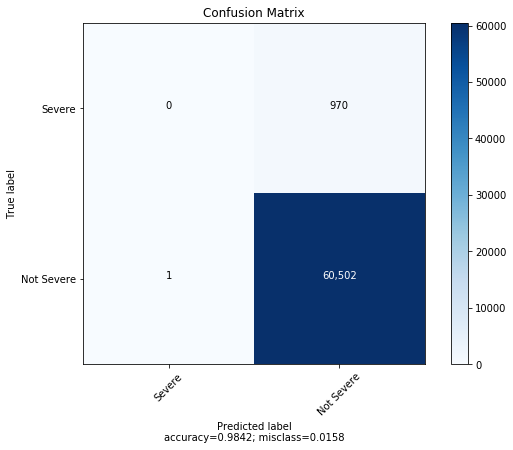
We fit the training data with the *LightGBM* classifier and tested the model by making predictions on the x\_test sample. For evaluating the model, **accuracy**, **precision**, **recall**, and **ROC AUC** were used as the main metrics, all of which were imported from the *sklearn.metrics* library. A first look at the results reveals the **accuracy score to be 0.9842**. The precision and recall scores values are same.

Another way of quickly visualizing these values is by using the *classification\_report* function. Although, it does not show the accuracy and roc\_auc scores. Looking at the classification report reveals the bigger picture, shown in *Fig 36*.



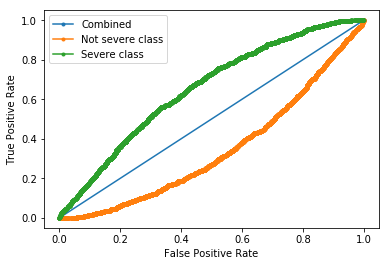
**Fig 36.** Classification report for baseline model

The table above shows that the model is only predicting the majority class. This can be observed from the individual precision or recall value for the ‘1’ (severe) class which is a 0. In fact, if the predicted values are categorized into severe and non-severe values, we notice that only one case was predicted as ‘severe’ or ‘1’ which is actually a wrong prediction. This can be further analyzed using a confusion matrix, shown in *Fig 37*.



**Fig 37.** Confusion matrix for the baseline model.

As seen in *Fig 37*, 60,502 instances were correctly predicted as being non-severe (true negative), 970 were incorrectly predicted as non-severe (false negative), 1 was incorrectly predicted as severe (false positive) and none were predicted correctly as severe (true positive). In order to explore this behavior further, we looked at the ROC AUC curve, shown in *Fig 38*.



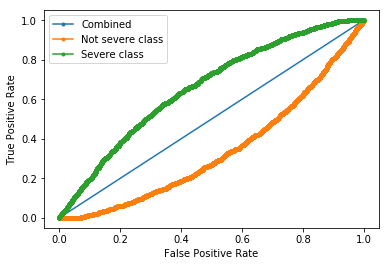
**Fig 38.** ROC AUC curve for combined, non-severe and severe classes.

The calculated **ROC AUC score for this model was 0.654**. This value is low mostly due to the large class imbalance in the data. The plot shown in *Fig 38* displays the ROC AUC curves for both the severe and non-severe classes as well as the combined ROC AUC. Moving along the curve corresponds to changing our threshold value for the predictions. Therefore, depending on the application, different thresholds can be set in order to achieve a certain specification.

## Model selection and optimization

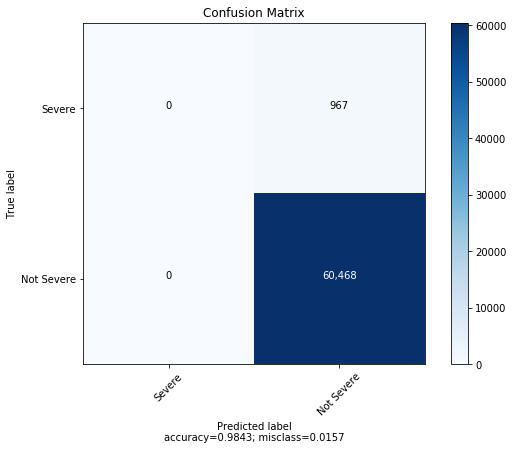
Grid search was performed in order to tune the hyperparameters of the model. Cross-validation was used within the grid search algorithm with 5 folds in order to determine the best average ROC AUC score. This ensured that the optimization would not cause the obtained parameters to overfit the model.

Parameters **maximum depth**, **number of leaves**, **learning rate**, and **minimum data in each leaf** were tuned to values of **5, 10, 0.1 and 20** respectively while the **best score was 0.6397**. Applying the above parameters to the *LightGBM* model improved the baseline score from 0.654 to 0.659, which was further improved to 0.66 after applying scaling for the features using the *MinMaxScaler* function from *sklearn*. The resulting ROC AUC curve is shown in *Fig 39*.

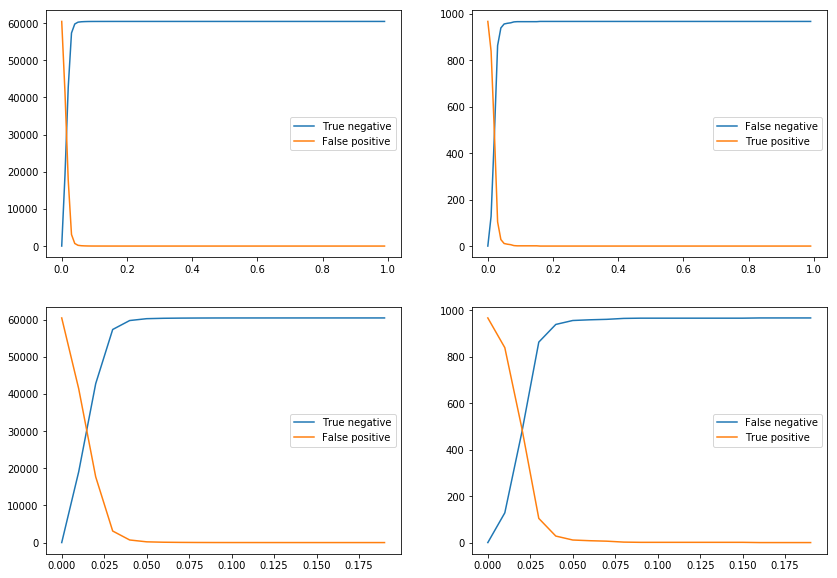


**Fig 39.** ROC AUC curve for combined, non-severe and severe classes after grid search and feature scaling.

The confusion matrix in *Fig 40* shows that even now, none of the severe cases are being predicted due to class imbalance. Hence, the thresholds were tuned to increase the number of TP results and decrease the number of FN results. *Fig 41* shows a sweep of threshold values from 0 to 1 with a step size of 0.01 to visualize the change in the above metrics.



**Fig 40.** Confusion matrix for the optimized model.

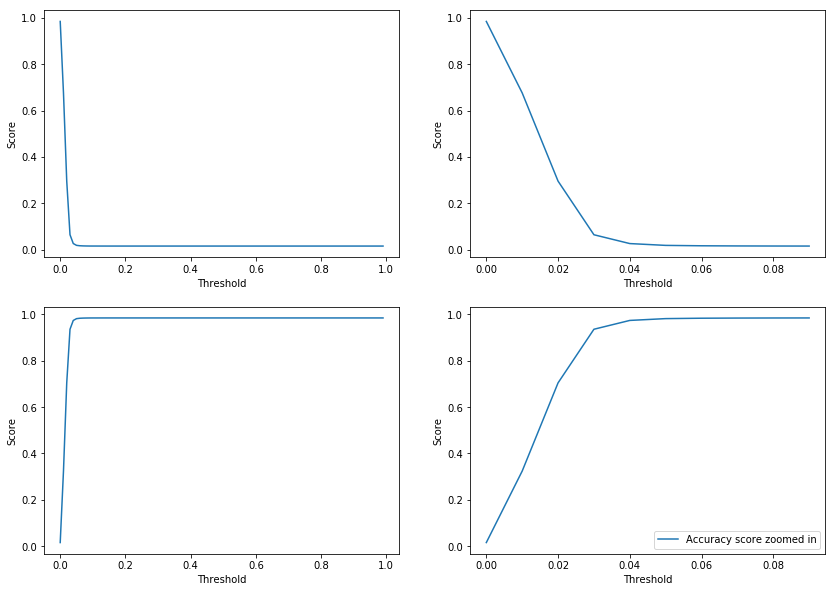


**Fig 41.** Confusion matrix metrics across thresholds.

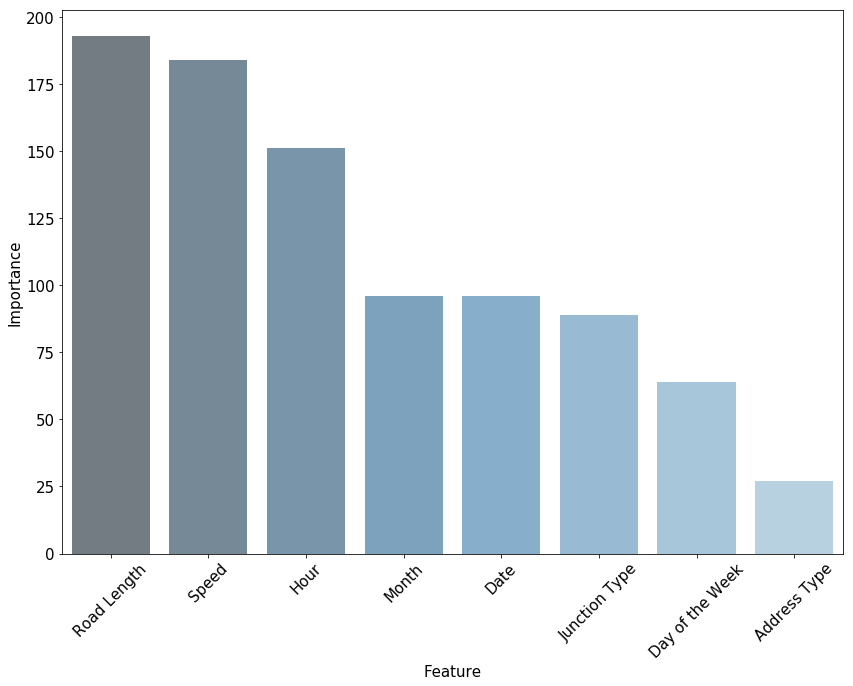
Looking at the plots above, the number of TP increases and FN decreases with decreasing threshold. We start to see significant increases in TP values only at a threshold of less than 0.05. At about 0.025, the TP and FN are the same.

The trend is same for TN and FP. The interesting thing to note here is that as we decrease the threshold, our accuracy decreases while our AUC remains the same as we are just moving along the AUC curve. As threshold decreases, events go from TN to FP whereby decreasing the accuracy and increasing the misclassification error which can be seen in *Fig 42*. Similarly, values go from FN to TP for decreasing threshold.

In our application where detecting severe collisions is the most important criteria, this is a fair tradeoff to consider.

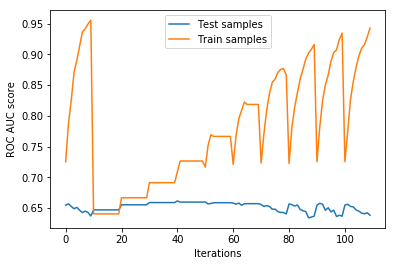


**Fig 42.** Accuracy and misclassification error across thresholds.



**Fig 43.** Important model features.

*Fig 43* shows the most important features for the model based on the training data. *Road Length* and *Speed* are the best estimators followed closely by *Hour*. This makes sense as we had observed some association between these variables and severity during EDA. The least important features are *Day of the Week* and *Address Type*. For practical purposes, the importance numbers are more relative than absolute. For example, *Address Type* is not necessarily a bad feature, it just has less predictive power than a variable like *Road Length*.



**Fig 44.** ROC AUC scores for train and test samples across hyperparameters.

*Fig 44* shows ROC AUC scores for train and test samples across the *number of leaves* and *max depth* hyperparameters. Since the number of leaves is the outer loop, the points at 0, 10, 20 and so on represent different number of leaves while the points between 0-10, 10-20 and so on represent different max depth values. The test sample performance remains fairly flat as compared to the train sample. Hence, higher train sample performance means greater overfitting. Based on this, points between 10-20 offer the best model in terms of overfitting. From point 20, each increase in test performance increases train performance by a higher rate, the ideal points being between 10-40. Anything above that would cause more and more overfitting.

## Other models

Other baseline models that were considered were *Logistic Regression* and *Random Forest* which yielded **ROC AUC scores of 0.629 and 0.545** respectively. For the logistic regression model, the meeting point between the TP and FN values during the threshold sweep was much earlier. Therefore, it would also be a reasonable model for the application of collision severity prediction as the threshold would only need to be adjusted by a small value.