Predicting Injuries for Traffic Crashes in Chicago

Kristiyan Dimitrov, Joshua Khazanov, Jieda Li, Laurie Merrell, Kristian Nikolov

1 Introduction

For our project, we examined a public data set containing information about traffic accidents in the city of Chicago. Our goal was to create a model which can predict whether or not a given accident will result in injuries, based only on information which would be available before a crash occurs, for example weather, road, and signage conditions.

Our model can be used by first responders to assess the likely severity of a crash based on partial information; for example, if a crash is reported at an intersection on a rainy day vs. on a straight 2-lane road on a sunny day, first responders can better anticipate the likelihood of injuries, even if more detail about the crash itself is not yet available. We also hope that such a model might be helpful in identifying universal risk factors for crashes which can be used to more strategically deploy resources in high-risk conditions: for example, our most successful random forest model identified that accidents in daylight were less likely to have injuries than accidents in other lighting conditions. Even though this finding is obvious, it could still be used to help first responders adjust their expectations for the likelihood of injuries based on lighting conditions and plan accordingly.

We fit a variety of models, focusing primarily on tree-based methods because of the structure of our data, which was all categorical. Our most successful model was a random forest with m, the random set of predictors that the model is able to split on, equal to 2, node size equal to 3, and 200 trees.

2 Dataset Overview

Our data is from the the City of Chicago Data Portal (data.cityofchicago.org) "Traffic Crashes - Crashes" data set. The data set is updated on an ongoing basis to contain records of all traffic accidents (car crashes) in the city -- it has data starting in 2015, but has only contained reports from all police precincts since September 2017.[1] We downloaded the data in January 2020, and had an initial dataset of 377,000 observations. Each row in the data set represents a single accident, and includes features related to the conditions in which the crash occurred (location, weather, lighting, etc.), attributes of the crash itself (number of vehicles involved, type of contact between involved parties, etc.), and information about how and when the crash was reported to the police.

[1] https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if

While the dataset contains over forty features, we decided to limit our predictions to include only features which could be assessed before a crash actually occurs. This limited our predictor set to primarily environmental and location-based features, but we believe that this choice makes our model more realistic as something that could actually be deployed; if we incorporated information about the crash itself in our predictions, our models would have higher accuracy, but be of very limited use. For example, if you already know that a crash was a head-on collision between a minivan and a bike, you can assume there are likely serious injuries. However, if all you know is that a crash occurred on a specific type of road in a given set of weather and lighting conditions, a prediction of whether or not there are likely to be injuries is much more useful.

We discuss our individual predictors in more detail below. At a high level, it is important to note that all of our predictors are categorical, which impacts which models are appropriate to fit.

3 Data Cleaning and Formatting

Because all our predictors were categorical, and in the original data set many had a large number of unique categories, much of our data cleaning and formatting was focused on reducing the number of unique categories to make the data more palatable for models. Additionally, many of the categories had one class which was much more common than the others, so it made sense in many cases to binarize between the majority class and all the other classes.

The response variable chosen was presence of injuries, which was converted from the number of injuries reported (INJURIES_TOTAL in the original data) to a binary variable that only specified whether or not injuries occurred.

The first predictor we included was DEVICE_CONDITION, which indicates the condition of the traffic control device (such as a stop light) present at the site of the crash. This originally contained eight unique values, containing some granular notes about the status of any traffic controls present (for example, “WORN REFLECTIVE MATERIAL”), but it was condensed into three: DEVICE_CONDITION IMPROPER, FUNCTIONING PROPERLY, and OTHER. We included NO CONTROL (i.e., no device present) as “FUNCTIONING PROPERLY”.

Next, we included and condensed WEATHER_CONDITION into only two values, CLEAR and OTHER, from the original ten, which included values like “CLOUDY” and “BLOWING SNOW”. As a result, 83% of observations had a value of CLEAR in this column.

Similarly, LIGHTING_CONDITION was compressed into only two values, DAYLIGHT and NON_DAYLIGHT, from an original set of 6 values which included items like DAWN and DARKNESS . Around 69% of crashes occurred when the condition was DAYLIGHT.

Another variable, TRAFFICWAY_TYPE, had 20 unique values originally. This variable contains information about the type of road on which the crash occurred -- is it a two-way road with or without a divider, etc. This variable was shortened to 7 unique values: NOT_DIVIDED, DIVIDED_WITH_BARRIER, DIVIDED_NO_BARRIER, PARKING_LOT, ONE_WAY, INTERSECTION, and OTHER. NOT_DIVIDED was the most common, as 46% of observations contained this value.

We also used a variable called ALIGNMENT, although it is highly imbalanced. 97.5% of crashes occurred on a straight and level surface. The other 5 values include information about whether the crash occurred on a hill or on a curve.

ROADWAY_SURFACE_COND is another key predictor, recording whether the roadway itself was clear at the time of the crash. 74.5% of crashes occurred on a dry surface. The other values included information about whether the surface was wet, icy, snowy, or had other contamination.

ROAD_DEFECT originally contained seven unique values (for example, “WORN SURFACE”), but it was turned into a binary variable recording whether or not a defect in the road was present.

We also decided to look at the date and time of the crash, specifically the hour of the day it occurred, the day of the week, and the month. We also created two binary variables, isRush which indicates whether the crash occurred during rush hour, and isHoliday which indicates whether the crash occurred on a holiday, from the original CRASH_DATE column. More specifically, isRush has value 1 when the Day of the Week is in {1,2,3,4,5} and the Hour is in {7, 8, 9, 16, 17, 18}, 0 otherwise. isHoliday has value 1 if the day was New Year’s Eve, New Year’s Day, Martin Luther King Day, Thanksgiving, Independence Day, etc. in any of the years between 2013 and 2019. In total, we found 4,596 crashes on such holidays (which are 17 days) in the data.

Observations containing any NA values for any of the predictors above were dropped, resulting in a final dataset size of 306,572 rows.

Lastly, we had a few variables which we originally incorporated into model building, but ultimately dropped. Two were dropped because we decided they violated our assumption about only including information that would be knowable before a crash occurs: the primary contributory cause of the crash (ex. failed to yield right-of-way or following too closely) and number of units in the crash. We also considered including latitude and longitude of the crash, but ultimately decided against it, since we were not sure that including that information as numeric variables made sense (since the numeric relationship between latitude and longitude values for different crashes is of dubious relevance to the quantity we are trying to model), and including it as categorical would create an unwieldy number of categories.

4 Exploratory Data Analysis (EDA)

It is important to observe the significant class-imbalance in the response variable of the data (Presence of Injury in crash). The minority class of Injury = 1 is ~14% of the data.

In the rest of our EDA, we tried to answer this question: is there some dependency on time (hour of day, day of week, or month), of the probability of occurrence of an injury?

We generated the following plots to answer this question:
Figure 4.1: Total injury cases vs hours of day (for each day of week)
Figure 4.2: Total injury cases vs day of week (for each month)
Figure 4.3: Total injury cases vs month

The results from the exploratory analysis lead to two key conclusions:

1, The probability of an injury has a strong dependency on time of day. We see that the highest peak happens around 15:00 to 18:00 (evening rush-hour), the second highest peak happens around 7:00 to 9:00 (morning rush-hour).

2, The probability of an injury doesn't strongly depend on day of week; it has a weak dependency on month, where we see summer months (7,8,9) has the highest number of injuries.

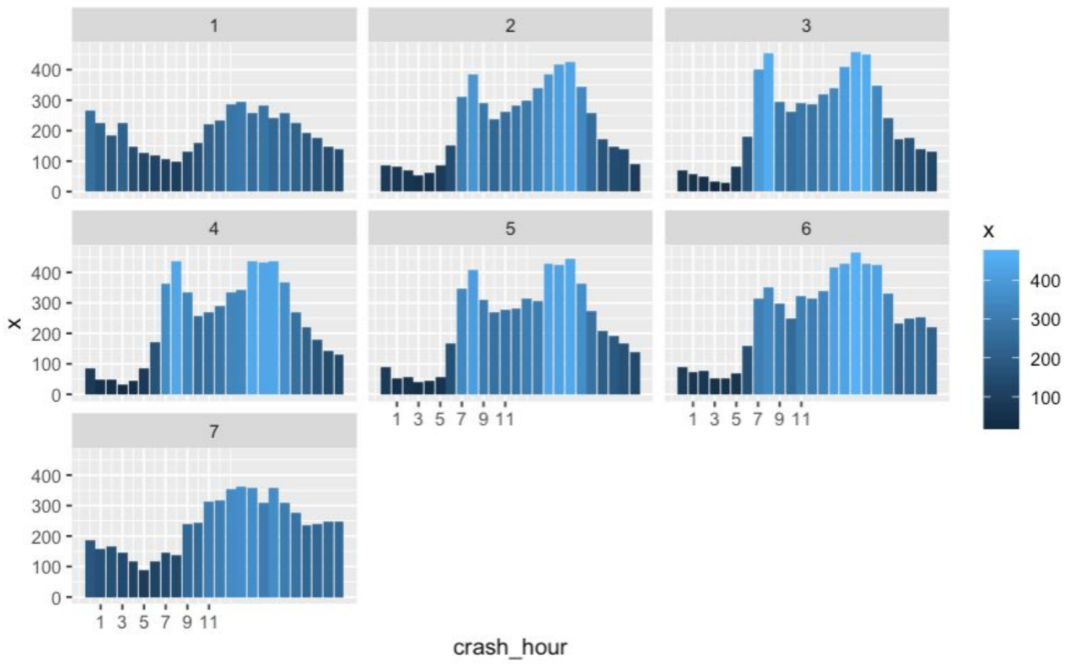


Figure 4.1 **Total injury cases** vs hours of day, for 7 days of week

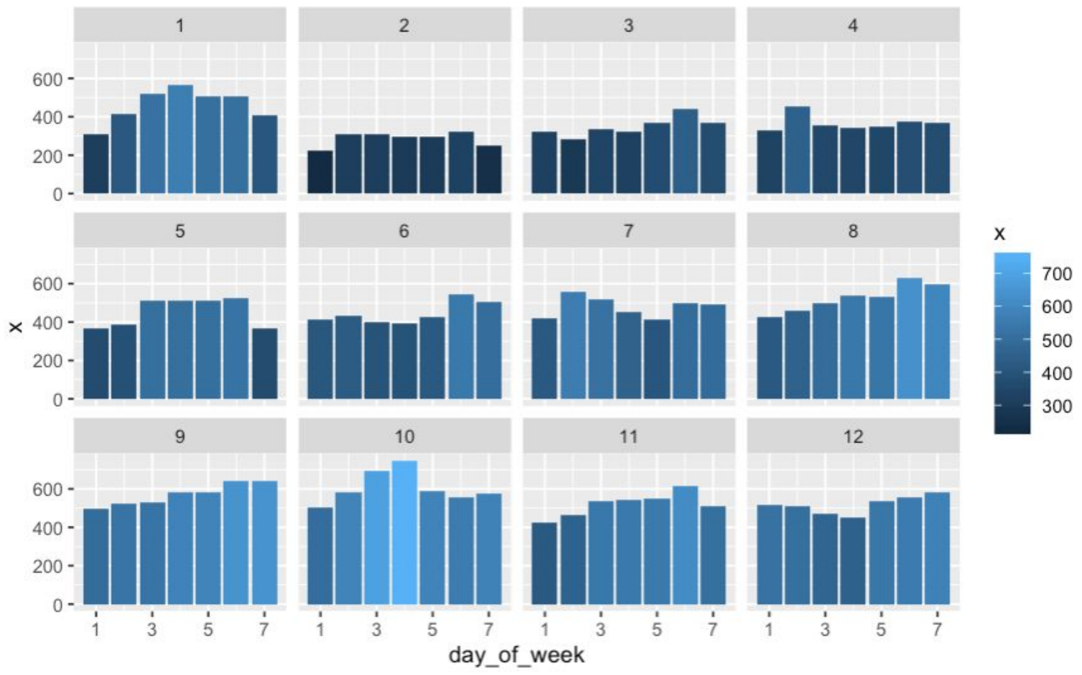


Figure 4.2 **Total injury cases** vs day of week, for 12 months

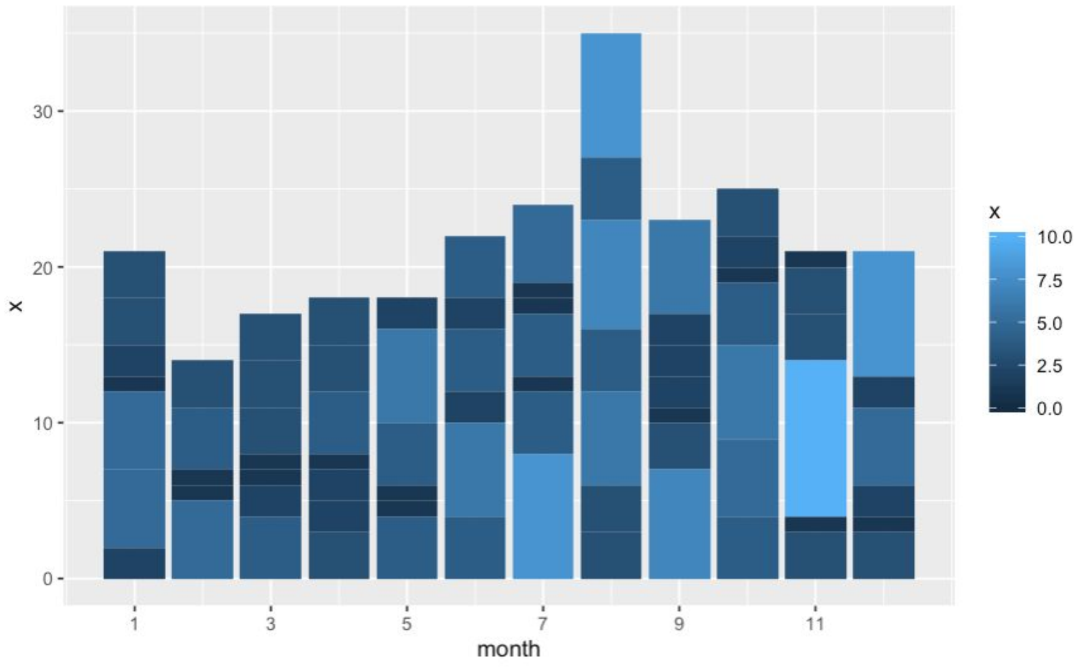


Figure 4.3 **Total injury cases** vs month (for 12 month)

5 Modeling Approach

Because the majority of our predictors are categorical variables, we anticipated that tree-based methods would perform the best. However, we experimented with a variety of methods for the sake of completeness.

1 Comparison Metrics and Fitting Procedure

Our modelling approach can be summarized into three steps:

1. Tune the hyperparameters of a specific model via Cross-Validation. Sets of hyperparameter values are evaluated based on the avg. AUC of the resulting model.

2. After we have our best model, the key question is "What p-star (cutoff probability) should we use for classifying a given crash as one with expected injuries?". Making such a decision would typically involve key stakeholder knowledge and considerations based on tolerance for False Negatives (FN) & False Positives (FP). In our case, we want to minimize the number of FN, because a FN in this situation means predicting no injury when, in fact, there is one. This is much worse than a FP, which is predicting an injury and there being none. So, we decided to pick p-star such that the FN rate (FNR) is as low as possible while keeping the FP rate (FPR) below 50%. We tried different p-star values for each model in a Cross-Validation fashion and collected the results in a table.

    a. It is important to note in some cases we used downsampling on each set of folds. This was done to address the significant class imbalance in our data (~ 14% of crashes had an injury). After a model was fit on the downsampled training data and used to calculate probabilities on the validation data, we correct those probabilities using this formula:

    $$p(\mathbf{x}) = \frac{p_S(\mathbf{x})O}{O_S - p_S(\mathbf{x})(O_S - O)}$$

    Finally, the corrected probabilities are converted into binary predictions using p-star and FNR & FPR are calculated. A step-by-step code version of this CV can be found in the appendix.

3. We tested our final models, with tuned hyperparameters and optimal p-star, through one final round of evaluation to generate the final metric of evaluation: FNR. We used a few different approaches: for some models, we used 1 or 3 replicates of 3-fold CV (in these cases we ensured our folds would be the same across models by using set.seed(1) and repeating indices <- CVInd(n, 3) either one or three times -- if once, then that replicate is identical to the first of the three replicates). For some models we calculated the final FNR on the entire dataset,

or used built-in functionality within other R packages to perform cross-validation. We realize this diversity of data used to perform the final evaluation of our models is not ideal; we will take it as a lesson for future projects to ensure identical final evaluation of our models. Regardless, we believe that we have included sufficient cross-validation that our results are still dependable and allow us to draw a conclusion as to which model performs best.

The models we chose are: Logistic Regression, Neural Network, Classification Tree, Random Forest, Boosted Trees, AdaBoost, Support Vector Machines. All these and their performance is described next.

2 Logistic Regression - Baseline

To get a sense of the performance of a "basic" model, we fit two logistic regression models: one on the original data, and one using downsampled data where the data was artificially balanced (a random subsample was drawn from the majority class to have the same size as the full set of minority class observations).

2.1 Logistic Regression: Original Data

We used 10-fold cross-validation, using the "ROC" (AUC) metric in the caret package's train() function to fit a logistic regression model and get a sense for its baseline performance. The cross-validation AUC value was 0.609.

A summary of this model is included in Appendix A. As we might expect, non-clear weather and non-daylight lighting conditions were both statistically significant predictors of injuries in an accident, as was a wet roadway surface. However, an icy roadway surface was somewhat significantly associated with decreased log odds of injury; this would likely require some additional exploration. There were 2.6k rows in the data with this attribute, which is relatively small, or perhaps there is an interaction with another variable which causes this surprising outcome.

When we followed step 2 of the modeling process as described above, we performed a second round of cross-validation to test multiple p* values. This process identified the optimal threshold for this model to be 0.13 to minimize the FNR while keeping the FPR below 0.5.

Finally, we ran this model through a third round of one iterate of cross-validation on the static set of shared 3-fold CV folds to get an estimate for the final FNR and FPR based on the p* value set in step 2. Our estimate for the final FNR is 0.430, and the FPR estimate is 0.424.

2.2 Logistic Regression: Downsampled Data

We again used 10-fold cross-validation using the "ROC" (AUC) metric in the caret package's train() function to fit a logistic regression model. However, in this version, we also used the sampling = "down" parameter in the trainControl() function from caret to downsample the training data so that injury- and non-injury- cases were equally represented in the training set.[2]

The CV AUC for this model was 0.609 (slightly lower than the first logistic model before rounding). A summary of this model is included in Appendix A. We see similar coefficient signs and significance scores between the two logistic models.

Once again, when we followed step 2 of the modeling process as described above, we identified the optimal threshold for this model to be 0.13, the same as it was for the non-downsampled data. We used the expression listed in step 2a above to adjust our predicted probabilities appropriately to account for the downsampling in the model fitting process.

Finally, we ran this model through the final round of cross-validation on one replicate over the shared 3-fold CV to get an estimate for the final FNR and FPR based on the p* value set in step 2. Our estimate for the final FNR is 0.431 and for FPR it is 0.423, and for AUC it was 0.572. Therefore this model performed slightly worse than the original logistic model on the FNR (the metric on which we are focusing), but it was ever so slightly better on FPR. Downsampling therefore did not seem to be beneficial for fitting the logistic regression model on this data.

3 Neural Net

We experimented with fitting neural nets using the nnet package. We found fitting such models with cross-validation to be prohibitively slow, even when using a subset of the data (ex. 100,000 rows ≈ ⅓ of the data or 60,000 rows ≈ ⅕ of the data). We were able to go through one full round of 3-fold CV to do model selection, with models being fit on a random subset of 60,000 rows (note that we used the caret package's default accuracy metric for CV instead of AUC; because of how long these models took to fit, we chose not to re-do this with AUC instead). Because of the time involved in fitting the neural net models, we tested a limited set of size/decay parameter pairs, and we did not explore downsampling.

Working with the original (i.e., not downsampled) data, we found that the optimal neural net had 5 nodes in the hidden layer and had a decay value of 0.2. This model resulted in a CV accuracy of 0.873.

[2] More information about the caret package's built-in downsampling: https://topepo.github.io/caret/subsampling-for-class-imbalances.html

Table 5.3.1: CV Accuracy For Selected Tuning Parameter Combinations in Nnet

Size / Decay	.05	.1	.2	.25
5	0.8730667	0.8732611	0.8733222	0.8733056
9	0.8726889	0.8727056	0.8729944	0.8731111
14	0.8720222	0.8723500	Not tested	Not tested
18	0.8713334	0.8718667	Not tested	Not tested

According to ALE plots generated with the ALEPlot package, the predictors associated with the largest changes in the probability of injury were trafficway type - intersections are the single biggest risk factor, associated with an increase in log odds of probability of injury orders of magnitude larger than other predictors (which matches with what we would likely expect; intersections are risky). The other important predictors identified by the ALEPlots were hour of day that the crash occurred; alignment (slope/curvature of road); and road conditions. Surprisingly, nighttime hours were associated with a lower risk of injuries; one might anticipate that risky drunk or impaired driving would be more likely at night which would lead to more injuries.

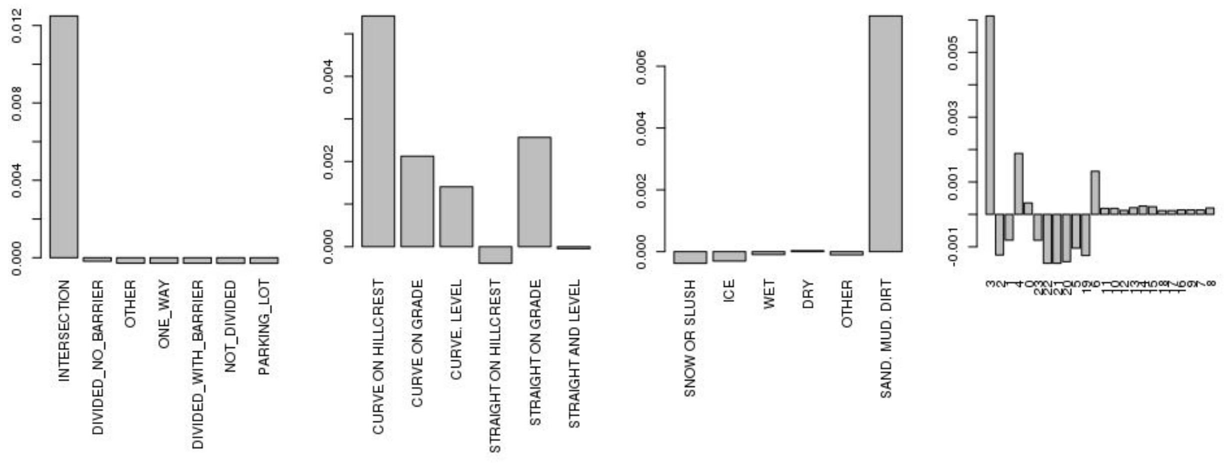


Figure 5.3.1 : ALE Plots for TRAFFICWAY_TYPE, ALIGNMENT, ROADWAY_CONDITION, and CRASH_HOUR

We used this model to proceed through steps 2 and 3 from our model fitting process as outlined above. We identified 0.13 as the optimal p* threshold once again. The final CV FNR was 0.427, FPR was 0.427, and the final CV AUC was 0.573. The neural net thus performed a little better than basic logistic regression.

4 CART

Classification trees seemed like a good fit for this problem, considering we have only categorical predictors. The first step, as in any classification problem, was to run a first version of the model with parameters which seemed like a good fit and see how the model performed overall. Using the rpart package, which has built in cross-validation, we ran a classification tree on the full dataset, the best model achieved had an R^2 of 12.4%, and never actually converged - pruning the tree with a complexity parameter greater than $2*10^{-6}$ resulted in a stump. As in most of the other models we tested, the best way to fix this was to balance our unbalanced dataset. Downsampling the majority class worked best, resulting in a model which converged at a complexity parameter of approximately 0.0005, increasing our best model's R^2 to 41.6%.

One of the great things about classification trees is that they are easily interpretable, allowing us to understand which predictors are best at predicting injuries. According to rpart's built in variable importance measure, the most important predictor is TRAFFICWAY_TYPE, followed by LIGHTING_CONDITION and CRASH_HOUR. TRAFFICWAY_TYPE very accurately classifies that most crashes that occur at parking lots or one way streets do not result in injuries.

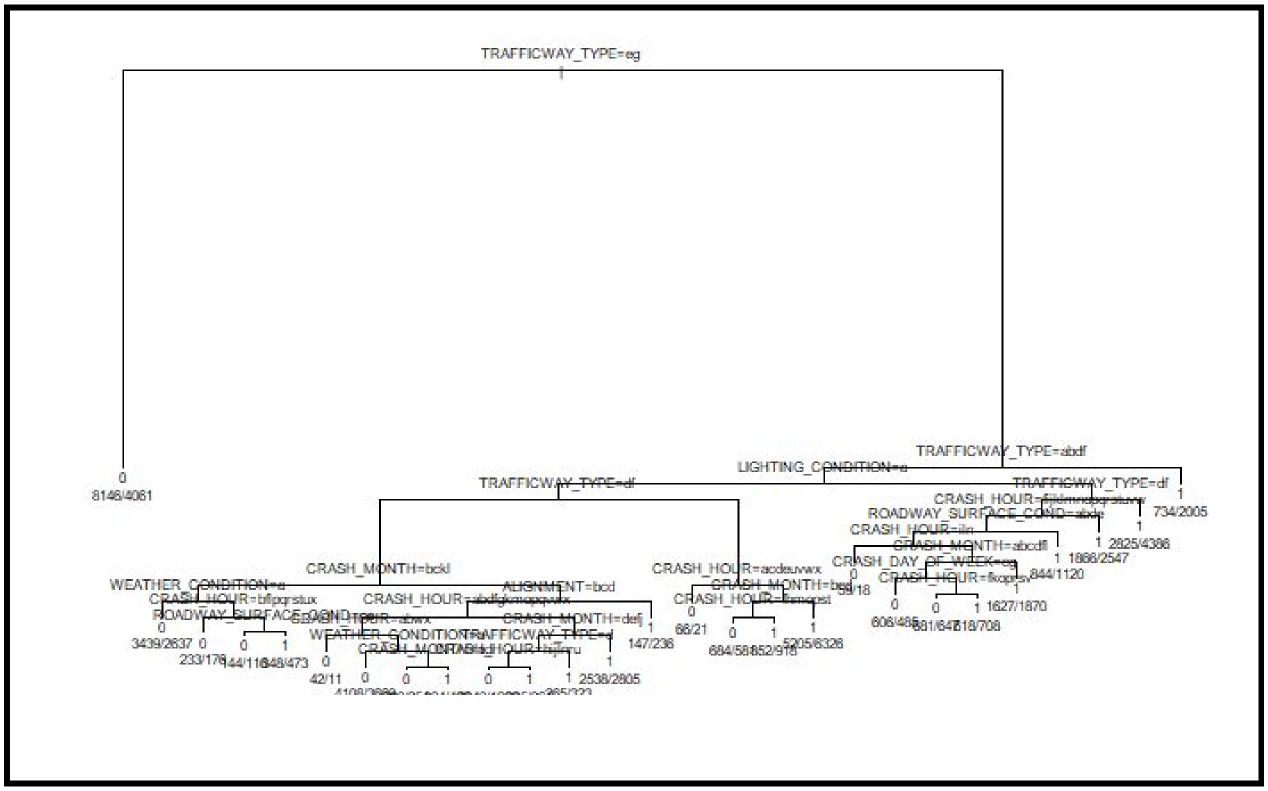


Figure 5.4.1: CART model

Now that we had cross-validated parameters for a best performing model, we needed to use them to calculate probabilities for the full dataset, and use Bayes classifier concepts to correct the probabilities, as in step 2 of our modelling approach. As in many of the other models, we used cross-validation to test multiple optimal threshold probabilities (p*) which would result in the lowest possible false negative rate (FNR). This turned out to be p* = 0.13, resulting in a FNR = 0.37, and a FPR = 0.49, right below our 0.5 threshold. To be able to compare this model to the other models, as per step 3 of our modelling approach, we ran the model on the same 3 folds the other models were using, resulting in the same FNR of 0.37, and an AUC of 0.57.

5 Random Forest

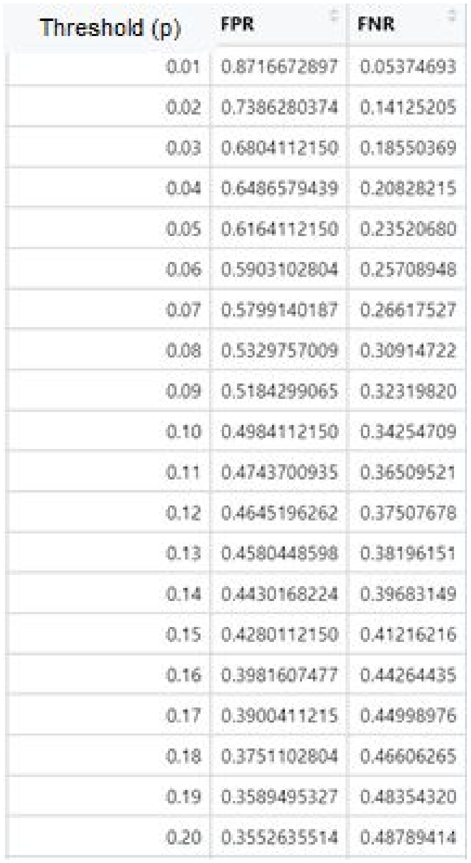
The first step in finding the best random forest model was to tune the parameters and the predictors used in the model. While predictive models like linear regression require regularization to remove unimportant variables from the model, random forests do this automatically, because unimportant variables will not be used to split data at nodes. However, there were two variables in the dataset that captured very similar time effects: CRASH_HOUR and isRush. This is because isRUSH was equal to 1 for certain CRASH_HOUR values (specifically, when it was rush hour), and 1 for the rest. Hence, we decided to tune to three variable combinations:

- All (DEVICE_CONDITION, WEATHER_CONDITION, LIGHTING_CONDITION, TRAFFICWAY_TYPE, ALIGNMENT, ROADWAY_SURFACE_COND, ROAD_DEFECT, CRASH_DAY_OF_WEEK, CRASH_MONTH, isRush, isHoliday, CRASH_HOUR, LATITUDE, and LONGITUDE)
- All but isRush
- All but CRASH_HOUR

The mtry parameter was tuned also, using values 1, 2, 3, 4, 5, and 6. Nodesize was tuned, using values 1, 2, 3, 4, and 5. Hence 150 random forests were fit using a treesize of 150. The majority class was also downsampled so that the classes were balanced when fitting the model.

Using the AUC metric on every model, the best mtry was 2, the best node size was 3, and the best variable combination was the one that excluded CRASH_HOUR. This resulted in a minority class error (FNR) of 37.6% and majority class error (FPR) of 47.3%, using the default threshold of the randomForest package on the uncorrected probabilities.

Next, the probabilities outputted by the randomForest function were corrected via Bayes classifier concepts. As aforementioned, the FNR could then be improved further by altering the classification threshold. The optimal p* turned out to be 0.1, which yielded a FNR of 34.3% while capping FPR at 50%.



Threshold (p)	FPR	FNR
0.01	0.8716672897	0.05374693
0.02	0.7386280374	0.14125205
0.03	0.6804112150	0.18550369
0.04	0.6486579439	0.20828215
0.05	0.6164112150	0.23520680
0.06	0.5903102804	0.25708948
0.07	0.5799140187	0.26617527
0.08	0.5329757009	0.30914722
0.09	0.5184299065	0.32319820
0.10	0.4984112150	0.34254709
0.11	0.4743700935	0.36509521
0.12	0.4645196262	0.37507678
0.13	0.4580448598	0.38196151
0.14	0.4430168224	0.39683149
0.15	0.4280112150	0.41216216
0.16	0.3981607477	0.44264435
0.17	0.3900411215	0.44998976
0.18	0.3751102804	0.46606265
0.19	0.3589495327	0.48354320
0.20	0.3552635514	0.48789414

Table 5.5.1: FPR and FNR or random forest model at various thresholds.

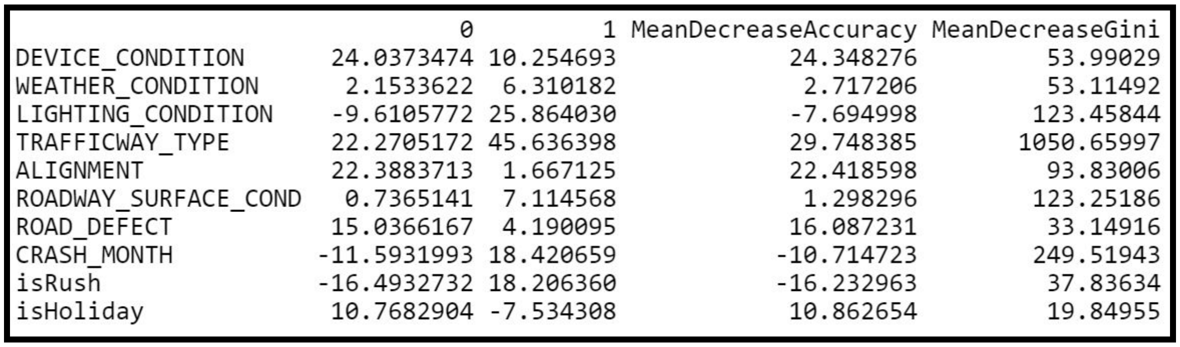
The variable importance table shows that, by far, the most important variable in predicting the presence of injuries in a crash is TRAFFICWAY_TYPE, followed by CRASH_MONTH, LIGHTING_CONDITION, and ROADWAY_SURFACE_COND. These decrease the Gini index at each split the most, on average.

We can take a look at the partial dependence plots to see what effect these predictors have on the response. A crash is more likely to result in an injury when it is darker out, which is unsurprising.

The trafficway type most likely to result in injuries is an intersection. This makes sense, because this is where head-on collisions or T-bones are more likely to happen than crashes where the vehicles are going in the same direction. By Newton's second law of motion, the latter type of crash is less forceful, and therefore is less likely to result in injury.

Wet roads are the most likely to result in injuries. This may be due to hydroplaning, during which drivers have no control over their vehicles, and are unable to change the trajectory of their vehicle even when they have time to react to a crash. Surprisingly, the plots suggest that snowy conditions are the least likely to result in injuries. Perhaps drivers exercise the most caution in these conditions, so crashes that do occur happen at slower speeds, resulting in fewer injuries.
Finally, crashes are more likely to result in injury if they occur in late summer or early autumn. This may be due to the same reason that snowy conditions result in fewer injuries.

Table 5.5.2: Variable importance of random forest model.



```
                               0          1 MeanDecreaseAccuracy MeanDecreaseGini
DEVICE_CONDITION      24.0373474 10.254693            24.348276         53.99029
WEATHER_CONDITION      2.1533622  6.310182             2.717206         53.11492
LIGHTING_CONDITION    -9.6105772 25.864030            -7.694998        123.45844
TRAFFICWAY_TYPE       22.2705172 45.636398            29.748385       1050.65997
ALIGNMENT             22.3883713  1.667125            22.418598         93.83006
ROADWAY_SURFACE_COND   0.7365141  7.114568             1.298296        123.25186
ROAD_DEFECT           15.0366167  4.190095            16.087231         33.14916
CRASH_MONTH          -11.5931993 18.420659           -10.714723        249.51943
isRush               -16.4932732 18.206360           -16.232963         37.83634
isHoliday             10.7682904 -7.534308            10.862654         19.84955
```

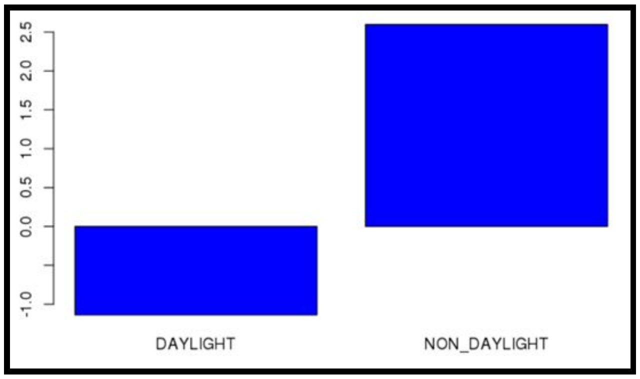


Figure 5.5.1: Partial dependence plot for LIGHTING_CONDITION

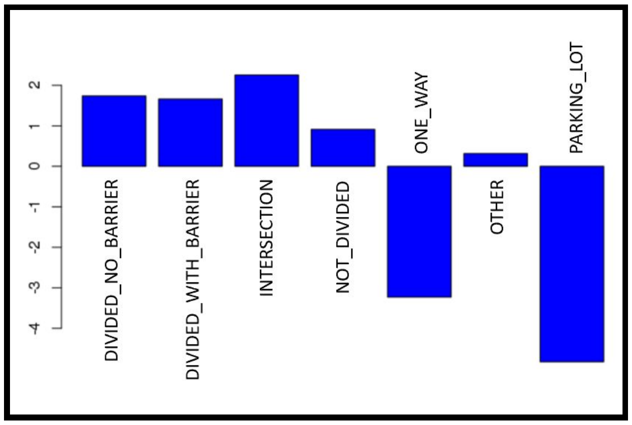


Figure 5.5.2: Partial dependence plot for TRAFFICWAY_TYPE

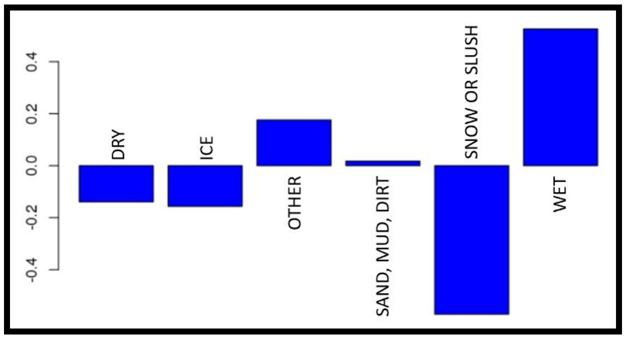


Figure 5.5.3: Partial dependence plot for ROADWAY_SURFACE_COND

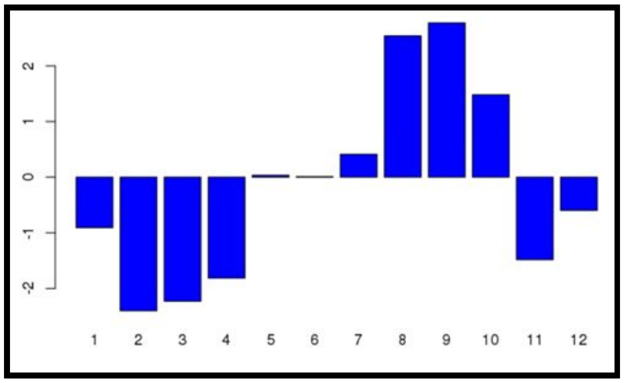


Figure 5.5.4: Partial dependence plot for CRASH_MONTH

6 Boosted Trees

We fitted a boosted tree model using the gradient boost machine (GBM) package in R; A similar modeling approach is used as we did with the other methods:

1) First downsample the majority class to have a balanced class ratio;

2) Second, use cross-validation to perform a grid search on the hyperparameters (shrinkage and interactive depth), use "AUC" as the CV measure to compare different hyperparameters; Based on CV AUC we identify the optimal hyperparameters.

3) Using the best model identified from CV, we did a search on p* to find the optimal p* that would give us the lowest possible FNR, while capping FPR at 50%.

4) Finally, with the optimal model and optimal p* identified, we perform cross-validation to calculate the generalized error and report this as the final model performance.

There are total of three hyperparameters that needs to be tuned, interactive.depth controls the size of each constituent tree, because the nature of boosted tree is to use a smaller tree to capture the residual error, the tree size is meant not to be large, a typical value of the interactive.depth is 3-8. shrinkage is used to "shrink" the impact of each additional fitted tree, typical value of shrinkage is 0.01 - 0.1, larger shrinkage value tends to over-fit and smaller shrinkage value tends to fit very slowly, shrinkage is analogy to "learning rate" used in gradient descent. In our case we did see that when we use shrinkage values greater than 0.1, we tend to overfit. The last hyperparameter is n.tree, the maximum number of trees, this is swept using the built-in CV function.

Table 5.6.1 shows the summary of the Cross-validation results on comparing different hyperparameters; the result suggests that a shrinkage of 0.5 and interactive depth of 4 gives us the best performance in terms of AUC.

GBM Model Cross-validation AUC under different hyperparameters						
shrinkage/interactive.depth	3	4	5	6	7	8
0.01	0.6197	0.6199	0.6231	0.6228	0.6234	0.6221
0.05	0.6185	0.6184	0.6199	0.6205	0.6236	0.6223
0.1	0.6194	0.6197	0.6222	0.6201	0.6191	0.6238
0.25	0.6149	0.6151	0.6160	0.6164	0.6164	0.6163
0.5	0.6149	0.6125	0.6172	0.6140	0.6137	0.6150

Table 5.6.1: Cross-validation AUC for different hyperparameters

Figure 5.6.1 shows the CV result of the best GBM model.

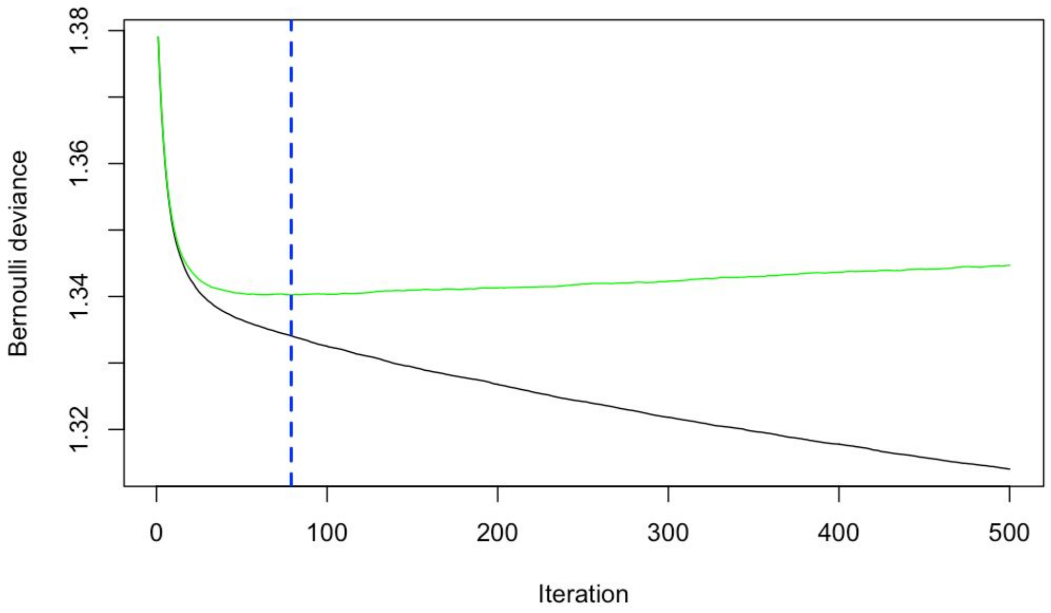


Figure 5.6.1: Cross-Validation Results for best GBM model

With the best model identified, we swapped p* and calculated FNR and FPR, the optimum FNR we achieved is 0.376 with p* of 0.5, while FPR is 0.46. (Note that our goal is to minimize FNR while capping FPR below 0.5).

With the optimum p_star identified, we perform a final cross-validation to calculate the generalized model performance in terms of FNR. Note that the final cross-validation is done using 1 replicate with 3-fold, in order to save computation time. Our final generalized FNR is 38.4%.

7 AdaBoost

As part of the project, we were supposed to choose a supervised learning model we did not cover in class. We chose Adaptive Boosting (aka AdaBoost).

First, we give a brief overview of how AdaBoost works. The easiest way is via contrasting it with Random Forests. (credits to StatQuest)

1) Random Forests grows full trees while AdaBoost grows stumps (trees with just two nodes). Individually, stumps are weak learners

2) With Random Forests, each tree is grown independently of all other trees based on bootstrap samples. AdaBoost, on the other hand, grows each subsequent tree

based on the errors of the previous tree. It does this by assigning weights to each observation in the data. These weights are then used either by calculating a weighted Gini index (description) for the effectiveness of the tree or by using the weights as a probability density function, which allows us to sample with replacement from the original dataset. In the end, the new weight for each observation is determined based on how well the current tree classifies the observation. The Adaptive part of AdaBoost's name comes from the fact that each subsequent tree adapts based on the mistakes made by the previous one. the next tree. This is similar to other boosting methods.

3) When making predictions, Random Forests assigns equal weight to all trees. AdaBoost, however, weighs the prediction of each stump based on how well it classifies the training data.

The only hyperparameter that needs tuning for AdaBoost (Step 1 of modelling approach) are the number of trees to be grown. In order to get a ballpark measure for a reasonable number of trees, we applied the adaboost() function (from the fastadaboost package) to the entire dataset of ~306k rows and grew 500 trees. Figure 5.7.1 below shows the weight each subsequent tree gets in making a prediction (weights are between 0 and 1 and constitute a measure of tree importance). More specifically, the two lines correspond to thresholds of .1 and .05.

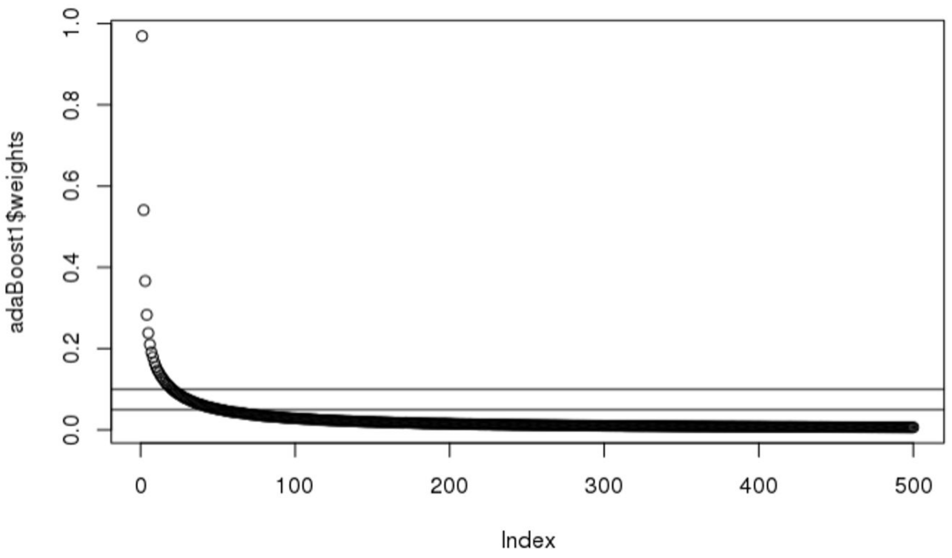


Figure 5.7.1: Tree index vs. weight in AdaBoost

Only the first 20 trees have weight > .1 and the first 51 trees have weight > .05. Results are summarized in table below:

# of Trees	Weight > X (threshold)
20	.1
51	.05
146	.02
312	.01

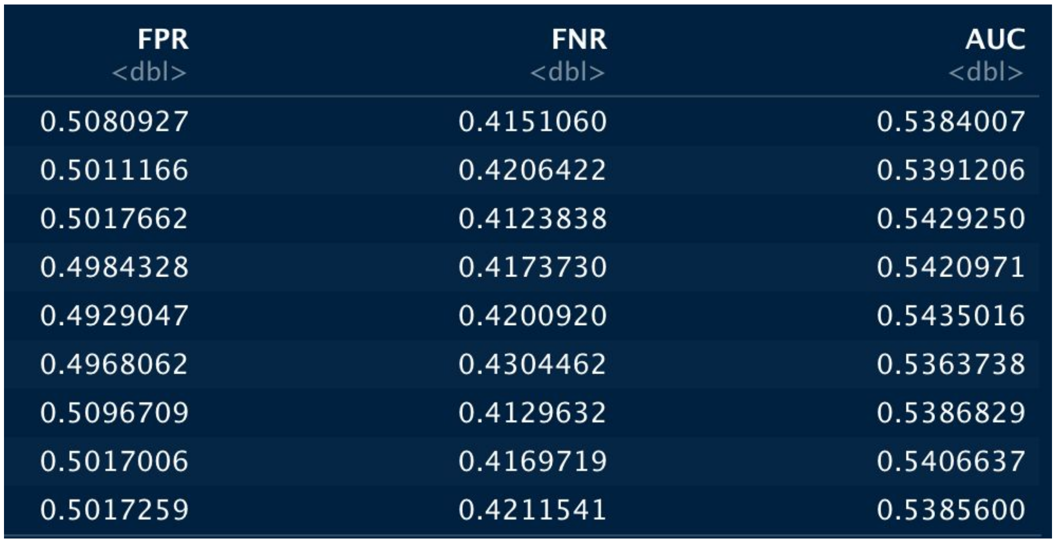
Table 5.7.1: Trees index vs. weight

Based on the above, we did not expect to need to grow more than 100 trees.

Next, we used the caret's train() function, which provides hyper-parameter tuning via cross-validation for AdaBoost with the method = 'adaboost' setting. As described in Step 1 of the modelling approach, we used AUC (the train() command requires setting the metric = 'ROC' option, but in fact it calculates AUC) as our measure of the best model. Moreover, we used downsampling to address the class imbalance (only ~14% of rows had an injury). This gave us nIter = 8 as the optimum number of trees.

In Step 2 of the modelling approach for AdaBoost, we found the optimal p* to be .12. We note again that we used the probability adjusting formula from Notes 1, page24, as described in the modelling approach to address the downsampling.

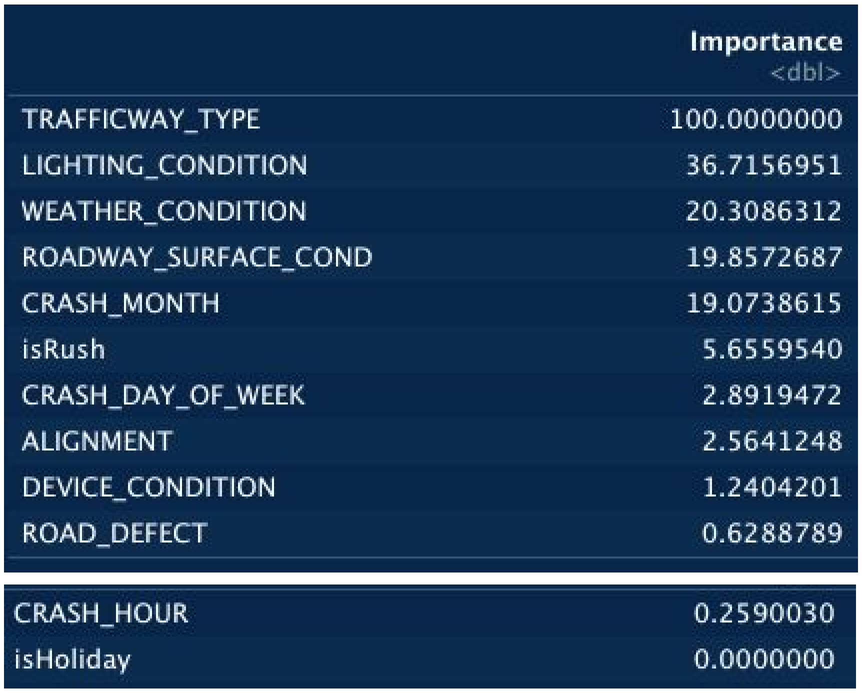
In Step 3 of the modelling approach we calculated the avg. FPR & FNR for our designated 3 fold sets (of 3 folds each) and arrived at the final results for AdaBoost: FPR = 50.13% and FNR = 41.85%. We note the avg. AUC across folds for the model is .54.



FPR <dbl>	FNR <dbl>	AUC <dbl>
0.5080927	0.4151060	0.5384007
0.5011166	0.4206422	0.5391206
0.5017662	0.4123838	0.5429250
0.4984328	0.4173730	0.5420971
0.4929047	0.4200920	0.5435016
0.4968062	0.4304462	0.5363738
0.5096709	0.4129632	0.5386829
0.5017006	0.4169719	0.5406637
0.5017259	0.4211541	0.5385600

*Table 5.7.2: 3 replicates of 3 folds with the optimal p**

Finally, we note the variable importance of the variables in the AdaBoost model on downsampled dataset.



	Importance <dbl>
TRAFFICWAY_TYPE	100.0000000
LIGHTING_CONDITION	36.7156951
WEATHER_CONDITION	20.3086312
ROADWAY_SURFACE_COND	19.8572687
CRASH_MONTH	19.0738615
isRush	5.6559540
CRASH_DAY_OF_WEEK	2.8919472
ALIGNMENT	2.5641248
DEVICE_CONDITION	1.2404201
ROAD_DEFECT	0.6288789
CRASH_HOUR	0.2590030
isHoliday	0.0000000

Tables 5.7.3 and 5.7.4: Variable importance in AdaBoost

Support Vector Machines (SVM)

SVM did not lead to any meaningful results for this dataset. First of all, it took very long to fit the model due to the large size of the dataset (~306k rows). To address this, we tried fitting the model on just 20k rows. However, any and all SVMs we fitted, regardless of choice of kernel (radial, polynomial, linear) and other hyperparameters, always predicted the majority class. Our initial hypothesis for this was the sheer number of dummy variables created from our data. The product of the number of levels for all categorical variables (minus 1) is ~455k, which is much more than the number of observations. Finally, after further deliberation, we realized SVM doesn't make sense for our data, because *all* our variables are categorical and therefore there is no meaningful definition of distance between observations. As a result, it is expected behavior that SVM wouldn't work and somewhat interesting how it still degenerated into always predicting the majority class.

8 Table Of Final Results

Recall that the values in this table are results from models which will vary somewhat from what was presented in the sections above, because these value were generated from a separate round of cross-validation (i.e., the FNR listed here is the average across folds from CV, so it is the result of 10 different models with variable coefficients, for example, though the hyperparameters are the values which were identified above in the tuning sections).

Model	**Final CV False Negative Rate**	**AUC**
Logistic Regression - No downsampling*	43.0%	0.57
Logistic Regression - Downsampling*	43.1%	0.57
Neural Net - No downsampling	42.7%	0.57
CART	37.2%	0.57
Random Forest	34.3%	0.58
Boosted Trees - DownSampling	38.4%	0.62
AdaBoost - DownSampling	41.9%	0.54
SVM	N/A	N/A

6 Conclusion

After running 8 different models, the random forest performed best in terms of our target metric, the false negative rate. As a whole, this level of accuracy was only accomplished since we capped the false positive rate at 50%, and our CV AUC score is only 0.58, performing just slightly better than guessing. Considering that our end goal was to build a model which could help the relevant institutions proactively prevent injuries from occurring, we cannot say that this model would actually help, considering the sheer number of false positives that go along with it, and the fact that most government institutions are understaffed as is.

However, running multiple different models on this dataset and testing different techniques helped us solidify our understanding of the data and, more importantly, helped us gain some interesting insights, which would certainly be of use. Knowing the types of roads, weather and lighting conditions which lead to more injuries can be used to better coordinate emergency units to dangerous areas on risky days and conditions.

We believe there may be ways to improve upon our models. One area to consider would be experimenting more with the bucketing of the categorical variables; we only really tried one or two approaches, but perhaps with some more experimentation with which categories to combine and keep separate additional insights could be possible. A second area of exploration would be quantifying the expected severity of a given crash i.e. the number of injuries and possible fatalities.

As with most data science initiatives, the gain to be made might not seem like too much, but if this information is able to decrease traffic injuries by even 1% at no additional cost, then we would consider our efforts worth it.

Appendix A: Model Output

Logistic Regression without Downsampling

Coefficients:	Estimate	Std. Error	z value	Pr(>|z|)	Signif.
(Intercept)	-0.79258	0.136674	-5.799	6.67E-09	***
DEVICE_CONDITIONFUNCTIONING_PROPERLY	-0.320022	0.047056	-6.801	1.04E-11	***
DEVICE_CONDITIONOTHER	-0.070863	0.075025	-0.945	0.344903	
WEATHER_CONDITIONOTHER	0.146247	0.022009	6.645	3.04E-11	***
LIGHTING_CONDITIONNON_DAYLIGHT	0.181194	0.020254	8.946	< 2e-16	***
TRAFFICWAY_TYPEDIVIDED_WITH_BARRIER	-0.155618	0.022294	-6.98	2.95E-12	***
TRAFFICWAY_TYPEINTERSECTION	0.661152	0.032875	20.111	< 2e-16	***
TRAFFICWAY_TYPENOT_DIVIDED	-0.328736	0.020655	-15.916	< 2e-16	***
TRAFFICWAY_TYPEONE_WAY	-0.813899	0.026422	-30.803	< 2e-16	***
TRAFFICWAY_TYPEOTHER	-0.274725	0.028426	-9.664	< 2e-16	***
TRAFFICWAY_TYPEPARKING_LOT	-1.659251	0.041135	-40.336	< 2e-16	***
`ALIGNMENTCURVE ON HILLCREST`	-0.052316	0.240113	-0.218	0.827523	
`ALIGNMENTCURVE, LEVEL`	-0.37533	0.131316	-2.858	0.00426	**
`ALIGNMENTSTRAIGHT AND LEVEL`	-0.455451	0.118566	-3.841	0.000122	***
`ALIGNMENTSTRAIGHT ON GRADE`	-0.009668	0.125543	-0.077	0.938614	
`ALIGNMENTSTRAIGHT ON HILLCREST	-0.191735	0.145571	-1.317	0.187797	
ROADWAY_SURFACE_CONDICE	-0.121106	0.06322	-1.916	0.055412	.
ROADWAY_SURFACE_CONDOTHER	0.398827	0.108751	3.667	0.000245	***
`ROADWAY_SURFACE_CONDSAND, MUD, DIRT	` 0.370131	0.230708	1.604	0.108642	
`ROADWAY_SURFACE_CONDSNOW OR SLUSH	-0.23647	0.037997	-6.223	4.86E-10	***
ROADWAY_SURFACE_CONDWET	0.086447	0.022279	3.88	0.000104	***

ROAD_DEFECT1	-0.184142	0.035822	-5.14	2.74E-07	***
CRASH_HOUR1	-0.10354	0.056104	-1.846	0.064963	.
CRASH_HOUR2	-0.066176	0.057232	-1.156	0.247566	
CRASH_HOUR3	0.015986	0.059423	0.269	0.78791	
CRASH_HOUR4	-0.093187	0.061896	-1.506	0.132189	
CRASH_HOUR5	-0.121762	0.058568	-2.079	0.037619	*
CRASH_HOUR6	-0.074245	0.051737	-1.435	0.151278	
CRASH_HOUR7	-0.097275	0.05221	-1.863	0.062442	.
CRASH_HOUR8	-0.162081	0.051712	-3.134	0.001722	**
CRASH_HOUR9	-0.156002	0.052296	-2.983	0.002854	**
CRASH_HOUR10	-0.226002	0.049779	-4.54	5.62E-06	***
CRASH_HOUR11	-0.151137	0.048878	-3.092	0.001987	**
CRASH_HOUR12	-0.230931	0.048656	-4.746	2.07E-06	***
CRASH_HOUR13	-0.143955	0.048139	-2.99	0.002786	**
CRASH_HOUR14	-0.162523	0.047683	-3.408	0.000653	***
CRASH_HOUR15	-0.199461	0.04694	-4.249	2.14E-05	***
CRASH_HOUR16	-0.196221	0.04906	-4	6.34E-05	***
CRASH_HOUR17	-0.228399	0.047949	-4.763	1.90E-06	***
CRASH_HOUR18	-0.178773	0.047851	-3.736	0.000187	***
CRASH_HOUR19	-0.130462	0.045522	-2.866	0.004158	**
CRASH_HOUR20	-0.161407	0.047358	-3.408	0.000654	***
CRASH_HOUR21	-0.102048	0.047916	-2.13	0.033194	*
CRASH_HOUR22	-0.142042	0.048757	-2.913	0.003577	**
CRASH_HOUR23	-0.050845	0.050243	-1.012	0.311546	
CRASH_DAY_OF_WEEK2	-0.032265	0.021963	-1.469	0.14182	
CRASH_DAY_OF_WEEK3	-0.025102	0.021788	-1.152	0.249277	
CRASH_DAY_OF_WEEK4	-0.001611	0.021731	-0.074	0.9409	

CRASH_DAY_OF_WEEK5	-0.034606	0.021773	-1.589	0.111976	
CRASH_DAY_OF_WEEK6	-0.060251	0.022865	-2.635	0.008411	**
CRASH_DAY_OF_WEEK7	-0.039707	0.022385	-1.774	0.076095	.
CRASH_MONTH2	-0.10634	0.0306	-3.475	0.00051	***
CRASH_MONTH3	-0.032847	0.029345	-1.119	0.263005	
CRASH_MONTH4	-0.004683	0.029059	-0.161	0.87196	
CRASH_MONTH5	0.093684	0.027907	3.357	0.000788	***
CRASH_MONTH6	0.090007	0.028113	3.202	0.001367	**
CRASH_MONTH7	0.166267	0.027628	6.018	1.77E-09	***
CRASH_MONTH8	0.202754	0.027066	7.491	6.83E-14	***
CRASH_MONTH9	0.192731	0.026457	7.285	3.22E-13	***
CRASH_MONTH10	0.126751	0.02591	4.892	9.98E-07	***
CRASH_MONTH11	0.023435	0.026324	0.89	0.37333	
CRASH_MONTH12	0.015701	0.026284	0.597	0.550262	
isRush1	-0.017171	0.025464	-0.674	0.500096	
isHoliday1	0.065738	0.050873	1.292	0.196289	

Signif. codes: 0 ‘***’ 0.001‘ **’ 0 .01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 233918 on 36571 degrees of freedom

Residual deviance: 227844 on 36508 degrees of freedom

AIC: 227972

Number of Fisher Scoring iterations: 5

Logistic Regression With Downsampling

Coefficients:	Estimate	Std. Error	z value	Pr(>|z|)	Signif.
(Intercept)	1.205645	0.202335	5.959	2.54E-09	***
DEVICE_CONDITIONFUNCTIONING_PROPERL	-0.393037	0.068974	-5.698	1.21E-08	***

Y					
DEVICE_CONDITIONOTHER	-0.089731	0.10742	-0.835	0.403532	
WEATHER_CONDITIONOTHER	0.166918	0.029493	5.66	1.52E-08	***
LIGHTING_CONDITIONNON_DAYLIGHT	0.173097	0.027675	6.255	3.98E-10	***
TRAFFICWAY_TYPEDIVIDED_WITH_BARRIER	-0.131802	0.031195	-4.225	2.39E-05	***
TRAFFICWAY_TYPEINTERSECTION	0.686498	0.051493	13.332	< 2e-16	***
TRAFFICWAY_TYPENOT_DIVIDED	-0.287769	0.028877	-9.965	< 2e-16	***
TRAFFICWAY_TYPEONE_WAY	-0.796027	0.034934	-22.787	< 2e-16	***
TRAFFICWAY_TYPEOTHER	-0.259713	0.038997	-6.66	2.74E-11	***
TRAFFICWAY_TYPEPARKING_LOT	-1.621751	0.048459	-33.466	< 2e-16	***
`ALIGNMENTCURVE ON HILLCREST`	-0.123876	0.34699	-0.357	0.721091	
`ALIGNMENTCURVE, LEVEL`	-0.463942	0.193965	-2.392	0.016762	*
`ALIGNMENTSTRAIGHT AND LEVEL`	-0.521255	0.177343	-2.939	0.00329	**
`ALIGNMENTSTRAIGHT ON GRADE`	-0.070356	0.187406	-0.375	0.707348	
`ALIGNMENTSTRAIGHT ON HILLCREST`	-0.224937	0.21448	-1.049	0.294291	
ROADWAY_SURFACE_CONDICE	-0.113633	0.082898	-1.371	0.170451	
ROADWAY_SURFACE_CONDOTHER	0.303188	0.149712	2.025	0.042853	*
`ROADWAY_SURFACE_CONDSAND, MUD, DIRT	` 0.193238	0.309135	0.625	0.53191	
`ROADWAY_SURFACE_CONDSNOW OR SLUSH`	-0.179854	0.049323	-3.646	0.000266	***
ROADWAY_SURFACE_CONDWET	0.080305	0.029897	2.686	0.00723	**
ROAD_DEFECT1	-0.193081	0.047133	-4.097	4.19E-05	***
CRASH_HOUR1	-0.139122	0.078194	-1.779	0.075209	.
CRASH_HOUR2	0.002005	0.081416	0.025	0.98035	
CRASH_HOUR3	-0.045128	0.08351	-0.54	0.588924	
CRASH_HOUR4	-0.230082	0.084852	-2.712	0.006696	**
CRASH_HOUR5	-0.192551	0.080969	-2.378	0.017403	*

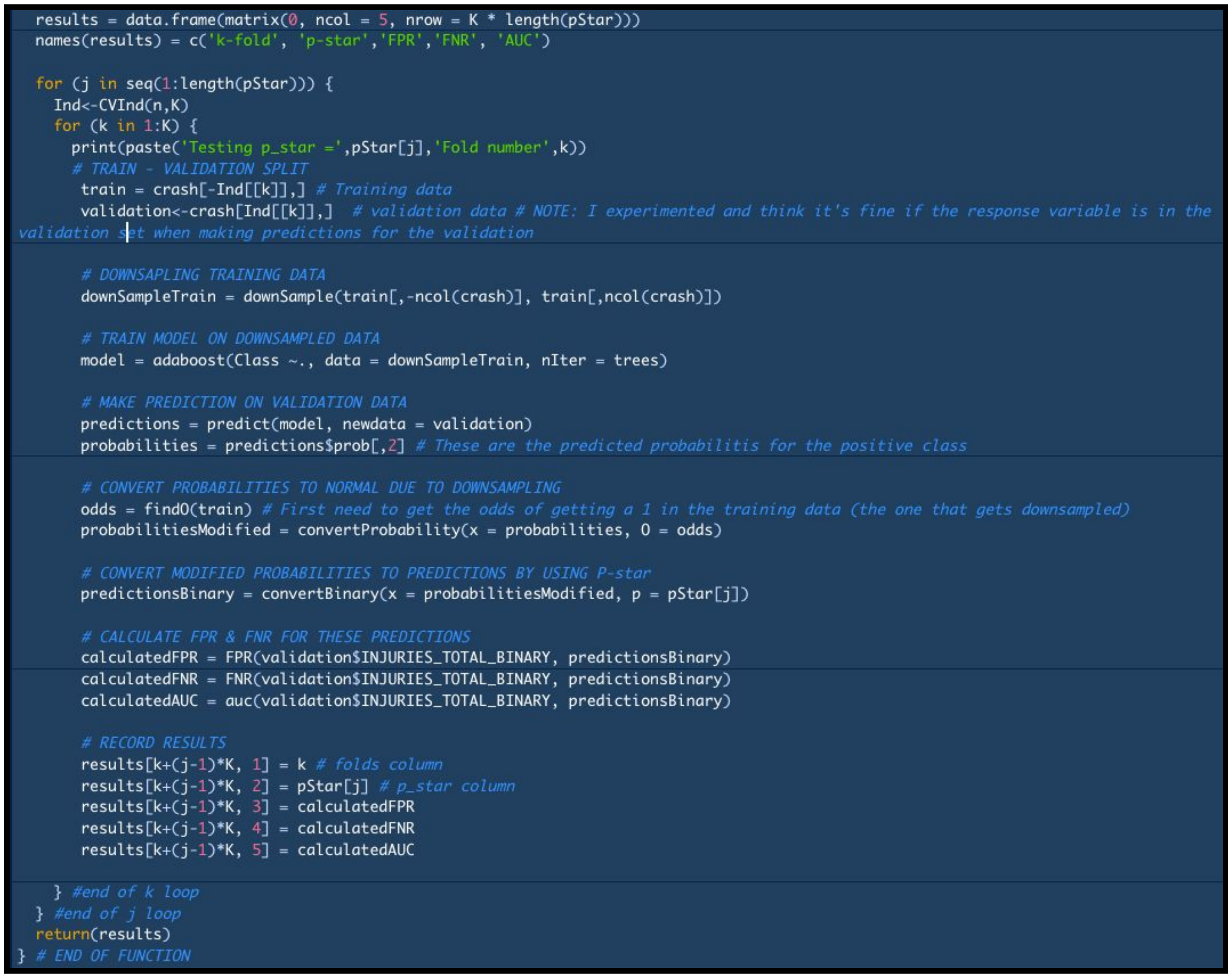
CRASH_HOUR6	-0.186651	0.07169	-2.604	0.009225	**
CRASH_HOUR7	-0.120343	0.072637	-1.657	0.097567	.
CRASH_HOUR8	-0.209878	0.071729	-2.926	0.003434	**
CRASH_HOUR9	-0.178927	0.072416	-2.471	0.01348	*
CRASH_HOUR10	-0.255735	0.068919	-3.711	0.000207	***
CRASH_HOUR11	-0.176655	0.068028	-2.597	0.009409	**
CRASH_HOUR12	-0.257765	0.067539	-3.817	0.000135	***
CRASH_HOUR13	-0.171459	0.067061	-2.557	0.010565	*
CRASH_HOUR14	-0.208979	0.066378	-3.148	0.001642	**
CRASH_HOUR15	-0.230423	0.065451	-3.521	0.000431	***
CRASH_HOUR16	-0.268178	0.068127	-3.936	8.27E-05	***
CRASH_HOUR17	-0.279674	0.066735	-4.191	2.78E-05	***
CRASH_HOUR18	-0.211996	0.066696	-3.179	0.00148	**
CRASH_HOUR19	-0.120415	0.063965	-1.883	0.059768	.
CRASH_HOUR20	-0.200097	0.066143	-3.025	0.002485	**
CRASH_HOUR21	-0.062095	0.067811	-0.916	0.359816	
CRASH_HOUR22	-0.120916	0.068681	-1.761	0.078317	.
CRASH_HOUR23	-0.1002	0.070627	-1.419	0.155978	
CRASH_DAY_OF_WEEK2	0.005195	0.029363	0.177	0.859565	
CRASH_DAY_OF_WEEK3	0.001282	0.029076	0.044	0.964818	
CRASH_DAY_OF_WEEK4	0.053977	0.029174	1.85	0.064286	.
CRASH_DAY_OF_WEEK5	0.014646	0.029133	0.503	0.615149	
CRASH_DAY_OF_WEEK6	-0.009721	0.030578	-0.318	0.750568	
CRASH_DAY_OF_WEEK7	-0.013383	0.030028	-0.446	0.65582	
CRASH_MONTH2	-0.060211	0.039982	-1.506	0.132079	
CRASH_MONTH3	-0.062376	0.038244	-1.631	0.102887	
CRASH_MONTH4	0.027181	0.038367	0.708	0.478677	

CRASH_MONTH5	0.123513	0.03727	3.314	0.00092	***
CRASH_MONTH6	0.105839	0.037394	2.83	0.00465	**
CRASH_MONTH7	0.207626	0.03712	5.593	2.23E-08	***
CRASH_MONTH8	0.219973	0.036318	6.057	1.39E-09	***
CRASH_MONTH9	0.214183	0.035482	6.036	1.58E-09	***
CRASH_MONTH10	0.146219	0.034642	4.221	2.43E-05	***
CRASH_MONTH11	0.066538	0.035044	1.899	0.057601	.
CRASH_MONTH12	0.015065	0.034788	0.433	0.664976	
isRush1	0.018178	0.03382	0.537	0.590924	
isHoliday1	0.072255	0.069006	1.047	0.295065	

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.0 5 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 108331 on 78143 degrees of freedom
Residual deviance: 104791 on 78080 degrees of freedom
AIC: 104919

Appendix B: Code

Cross-Validation for Selecting P (referred to in the modelling approach section)*
Note: Example code is for AdaBoost, but could vary.



```
  results = data.frame(matrix(0, ncol = 5, nrow = K * length(pStar)))
  names(results) = c('k-fold', 'p-star','FPR','FNR', 'AUC')

  for (j in seq(1:length(pStar))) {
    Ind<-CVInd(n,K)
    for (k in 1:K) {
      print(paste('Testing p_star =',pStar[j],'Fold number',k))
      # TRAIN - VALIDATION SPLIT
       train = crash[-Ind[[k]],] # Training data
       validation<-crash[Ind[[k]],]  # validation data # NOTE: I experimented and think it's fine if the response variable is in the
validation set when making predictions for the validation

       # DOWNSAPLING TRAINING DATA
       downSampleTrain = downSample(train[,-ncol(crash)], train[,ncol(crash)])

       # TRAIN MODEL ON DOWNSAMPLED DATA
       model = adaboost(Class ~., data = downSampleTrain, nIter = trees)

       # MAKE PREDICTION ON VALIDATION DATA
       predictions = predict(model, newdata = validation)
       probabilities = predictions$prob[,2] # These are the predicted probabilitis for the positive class

       # CONVERT PROBABILITIES TO NORMAL DUE TO DOWNSAMPLING
       odds = findO(train) # First need to get the odds of getting a 1 in the training data (the one that gets downsampled)
       probabilitiesModified = convertProbability(x = probabilities, O = odds)

       # CONVERT MODIFIED PROBABILITIES TO PREDICTIONS BY USING P-star
       predictionsBinary = convertBinary(x = probabilitiesModified, p = pStar[j])

       # CALCULATE FPR & FNR FOR THESE PREDICTIONS
       calculatedFPR = FPR(validation$INJURIES_TOTAL_BINARY, predictionsBinary)
       calculatedFNR = FNR(validation$INJURIES_TOTAL_BINARY, predictionsBinary)
       calculatedAUC = auc(validation$INJURIES_TOTAL_BINARY, predictionsBinary)

       # RECORD RESULTS
       results[k+(j-1)*K, 1] = k # folds column
       results[k+(j-1)*K, 2] = pStar[j] # p_star column
       results[k+(j-1)*K, 3] = calculatedFPR
       results[k+(j-1)*K, 4] = calculatedFNR
       results[k+(j-1)*K, 5] = calculatedAUC

    } #end of k loop
  } #end of j loop
  return(results)
} # END OF FUNCTION
```

Logistic Regression - Baseline

```
crash_ROC = crash

levels(crash_ROC$INJURIES_TOTAL_BINARY) <- c("False", "True")

fitControlROC <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated ten times
  repeats = 10, verboseIter = TRUE, summaryFunction =
twoClassSummary, classProbs = TRUE)

logistic_ROC <- train(INJURIES_TOTAL_BINARY ~ ., data =
crash_ROC,
                  method = "glm",
                  trControl = fitControlROC, family =
"binomial", metric = "ROC")

#CV - ROC - with downsampling
fitControlROC2 <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated ten times
  repeats = 10, verboseIter = TRUE, sampling = "down",
  summaryFunction = twoClassSummary, classProbs = TRUE)

logistic_downsample_ROC <- train(INJURIES_TOTAL_BINARY ~ ., data
= crash_ROC,
                             method = "glm",
                             trControl = fitControlROC2, family
= "binomial", metric = "ROC")
```

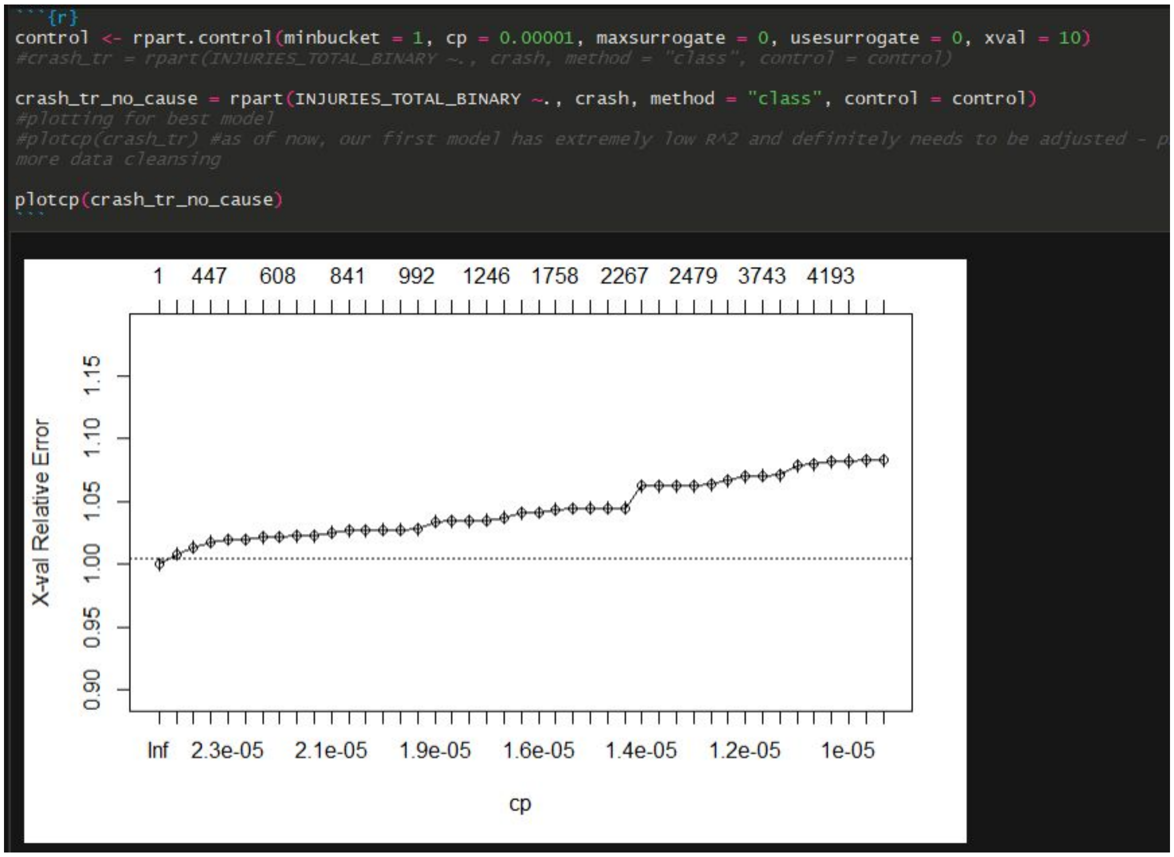
Neural Net

```
library(tidyverse)
library(caret)
library(nnet)

train_subset = crash %>% sample_n(60000)

#use caret package to tune parameters automatically

fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated 3 times
  repeats = 3, classProbs = TRUE)

levels(train_subset$INJURIES_TOTAL_BINARY) <- c("No", "Yes")

nnetGrid <-  expand.grid(size = c(5, 9, 14, 18, 22),
                        decay = c(0.001, 0.005, 0.01, 0.05,
0.1))
```

```
nnet_fit_test <- train(INJURIES_TOTAL_BINARY ~ ., data =
train_subset,
                 method = "nnet",
                 trControl = fitControl,
                 tuneGrid = nnetGrid)

#run again - increase num weights for the large size ones, would
not fit previously

nnetGrid2 <-  expand.grid(size = c(18, 22),
                         decay = c(0.001, 0.005, 0.01, 0.05,
0.1))

nnet_fit_test2 <- train(INJURIES_TOTAL_BINARY ~ ., data =
train_subset,
                       method = "nnet",
                       trControl = fitControl,
                       tuneGrid = nnetGrid2, MaxNWts = 2300)

#try for a third time - larger decay values
nnetGrid3 <-  expand.grid(size = c(5, 9),
                          decay = c(0.2, 0.25))

nnet_fit_test3 <- train(INJURIES_TOTAL_BINARY ~ ., data =
train_subset,
                        method = "nnet",
                        trControl = fitControl,
                        tuneGrid = nnetGrid3)
```

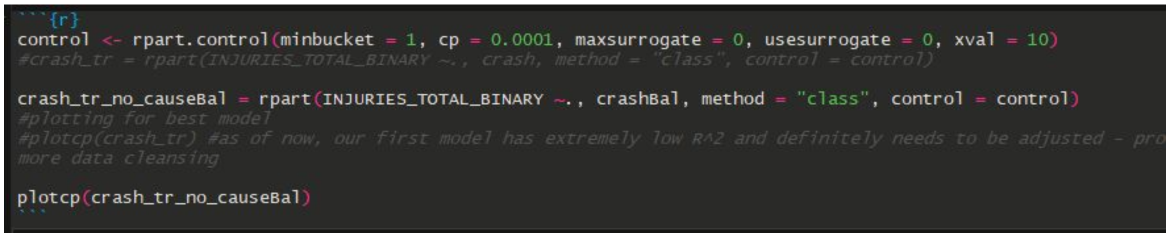
CART

1. Testing classification tree on full dataset

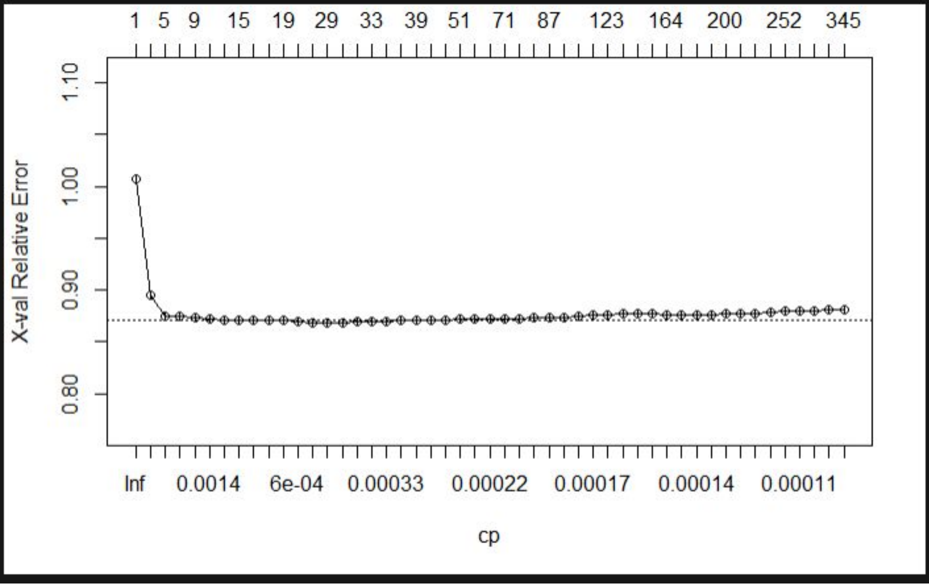


~~~
```{r}
control <- rpart.control(minbucket = 1, cp = 0.00001, maxsurrogate = 0, usesurrogate = 0, xval = 10)
#crash_tr = rpart(INJURIES_TOTAL_BINARY ~., crash, method = "class", control = control)

crash_tr_no_cause = rpart(INJURIES_TOTAL_BINARY ~., crash, method = "class", control = control)
#plotting for best model
#plotcp(crash_tr) #as of now, our first model has extremely low R^2 and definitely needs to be adjusted - p
more data cleansing

plotcp(crash_tr_no_cause)
```
~~~

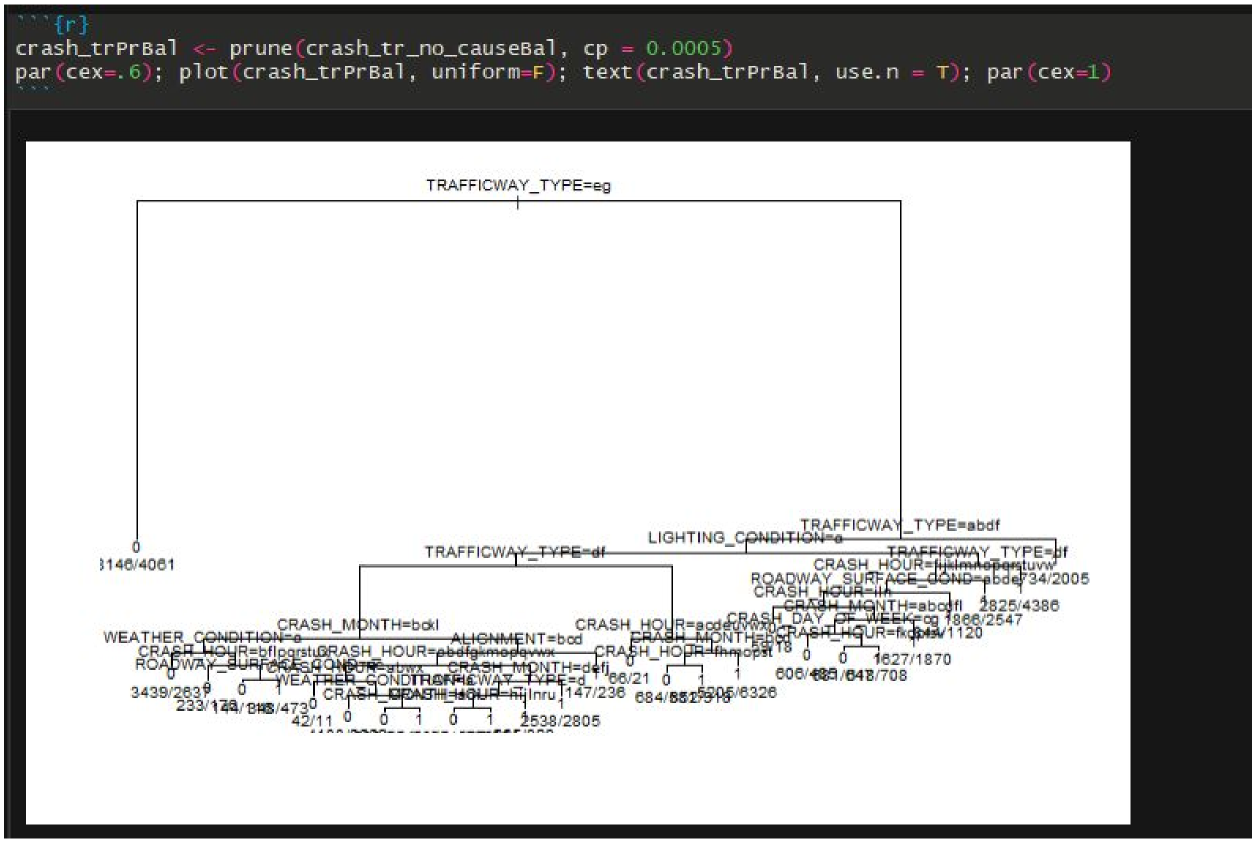
2. Modeling on downsampled dataset



~~~
```{r}
control <- rpart.control(minbucket = 1, cp = 0.0001, maxsurrogate = 0, usesurrogate = 0, xval = 10)
#crash_tr = rpart(INJURIES_TOTAL_BINARY ~., crash, method = "class", control = control)

crash_tr_no_causeBal = rpart(INJURIES_TOTAL_BINARY ~., crashBal, method = "class", control = control)
#plotting for best model
#plotcp(crash_tr) #as of now, our first model has extremely low R^2 and definitely needs to be adjusted - pro
more data cleansing

plotcp(crash_tr_no_causeBal)
```
~~~



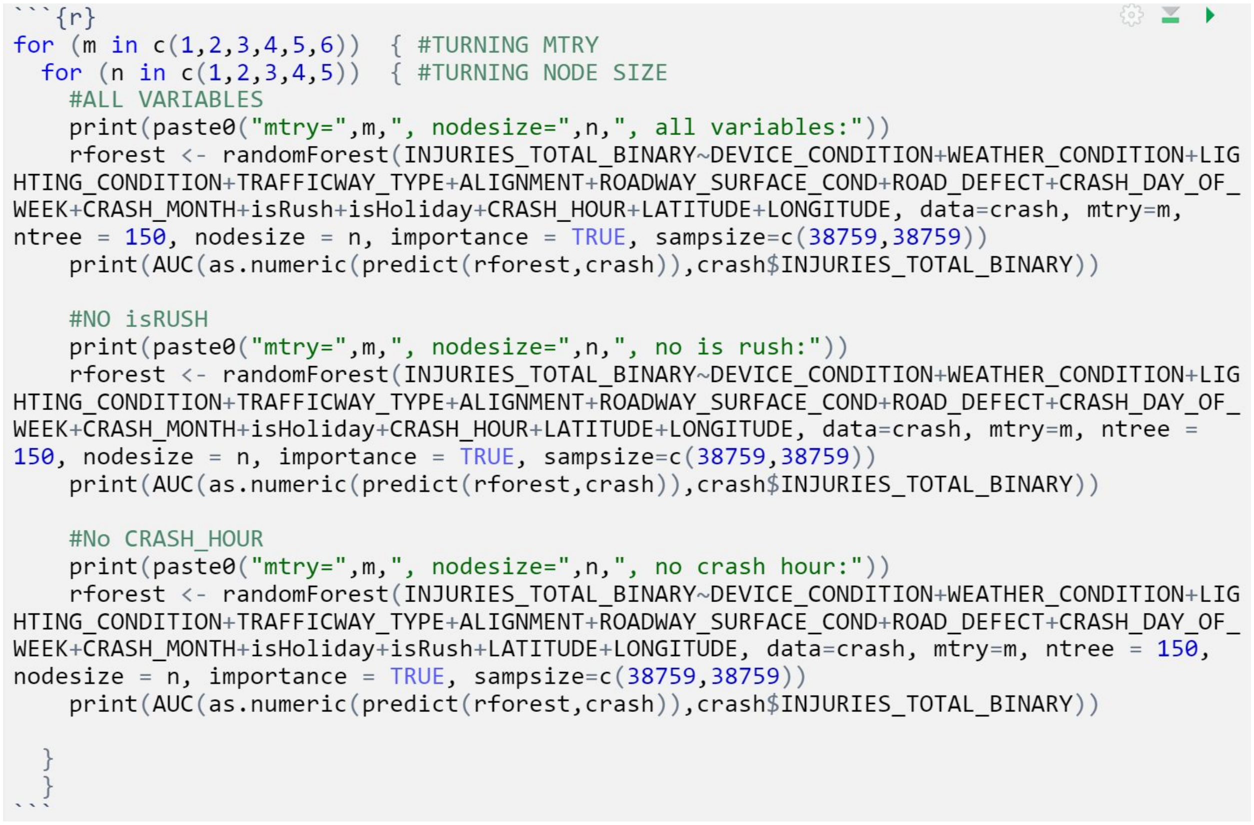
3. Pruning the optimal tree based on R^2 and analyzing variable importance using the pruned tree



```{r}
crash_trPrBal <- prune(crash_tr_no_causeBal, cp = 0.0005)
par(cex=.6); plot(crash_trPrBal, uniform=F); text(crash_trPrBal, use.n = T); par(cex=1)
```

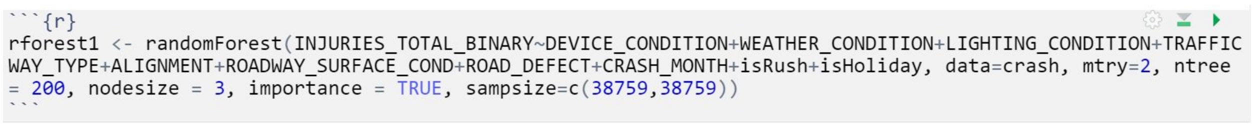
Random Forest

1. *Tuning mtry, node size, and variable selection*



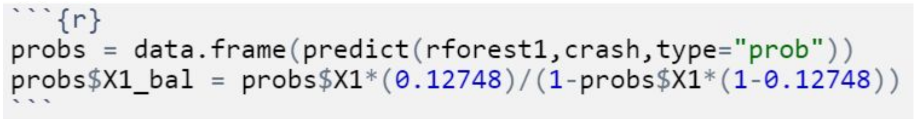
~~~
```{r}
for (m in c(1,2,3,4,5,6))  { #TURNING MTRY
  for (n in c(1,2,3,4,5))  { #TURNING NODE SIZE
    #ALL VARIABLES
    print(paste0("mtry=",m,", nodesize=",n,", all variables:"))
    rforest <- randomForest(INJURIES_TOTAL_BINARY~DEVICE_CONDITION+WEATHER_CONDITION+LIG
HTING_CONDITION+TRAFFICWAY_TYPE+ALIGNMENT+ROADWAY_SURFACE_COND+ROAD_DEFECT+CRASH_DAY_OF_
WEEK+CRASH_MONTH+isRush+isHoliday+CRASH_HOUR+LATITUDE+LONGITUDE, data=crash, mtry=m,
ntree = 150, nodesize = n, importance = TRUE, sampsize=c(38759,38759))
    print(AUC(as.numeric(predict(rforest,crash)),crash$INJURIES_TOTAL_BINARY))

    #NO isRUSH
    print(paste0("mtry=",m,", nodesize=",n,", no is rush:"))
    rforest <- randomForest(INJURIES_TOTAL_BINARY~DEVICE_CONDITION+WEATHER_CONDITION+LIG
HTING_CONDITION+TRAFFICWAY_TYPE+ALIGNMENT+ROADWAY_SURFACE_COND+ROAD_DEFECT+CRASH_DAY_OF_
WEEK+CRASH_MONTH+isHoliday+CRASH_HOUR+LATITUDE+LONGITUDE, data=crash, mtry=m, ntree =
150, nodesize = n, importance = TRUE, sampsize=c(38759,38759))
    print(AUC(as.numeric(predict(rforest,crash)),crash$INJURIES_TOTAL_BINARY))

    #No CRASH_HOUR
    print(paste0("mtry=",m,", nodesize=",n,", no crash hour:"))
    rforest <- randomForest(INJURIES_TOTAL_BINARY~DEVICE_CONDITION+WEATHER_CONDITION+LIG
HTING_CONDITION+TRAFFICWAY_TYPE+ALIGNMENT+ROADWAY_SURFACE_COND+ROAD_DEFECT+CRASH_DAY_OF_
WEEK+CRASH_MONTH+isHoliday+isRush+LATITUDE+LONGITUDE, data=crash, mtry=m, ntree = 150,
nodesize = n, importance = TRUE, sampsize=c(38759,38759))
    print(AUC(as.numeric(predict(rforest,crash)),crash$INJURIES_TOTAL_BINARY))

  }
  }
```
~~~

2. *Fitting final model*



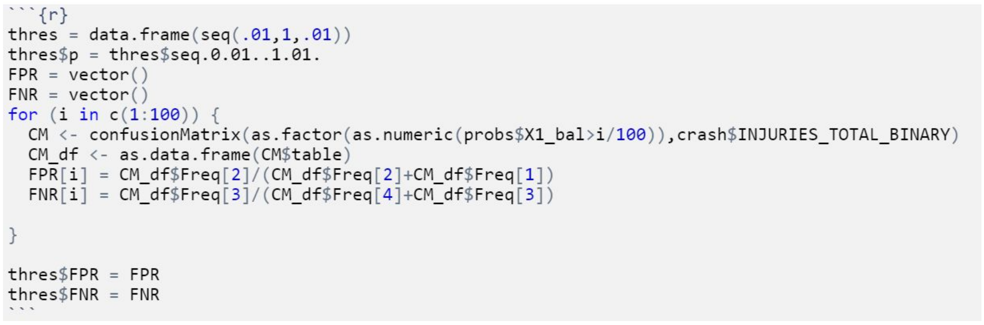
~~~
```{r}
rforest1 <- randomForest(INJURIES_TOTAL_BINARY~DEVICE_CONDITION+WEATHER_CONDITION+LIGHTING_CONDITION+TRAFFIC
WAY_TYPE+ALIGNMENT+ROADWAY_SURFACE_COND+ROAD_DEFECT+CRASH_MONTH+isRush+isHoliday, data=crash, mtry=2, ntree
= 200, nodesize = 3, importance = TRUE, sampsize=c(38759,38759))
```
~~~

3. *Balancing probabilities using Bayes*



~~~
```{r}
probs = data.frame(predict(rforest1,crash,type="prob"))
probs$X1_bal = probs$X1*(0.12748)/(1-probs$X1*(1-0.12748))
```
~~~

4. *Finding the optimal threshold*



~~~
```{r}
thres = data.frame(seq(.01,1,.01))
thres$p = thres$seq.0.01..1.01.
FPR = vector()
FNR = vector()
for (i in c(1:100)) {
  CM <- confusionMatrix(as.factor(as.numeric(probs$X1_bal>i/100)),crash$INJURIES_TOTAL_BINARY)
  CM_df <- as.data.frame(CM$table)
  FPR[i] = CM_df$Freq[2]/(CM_df$Freq[2]+CM_df$Freq[1])
  FNR[i] = CM_df$Freq[3]/(CM_df$Freq[4]+CM_df$Freq[3])

}

thres$FPR = FPR
thres$FNR = FNR
```
~~~

Boosted Trees

1) *Hyperparameter tuning for GBM*

```
shrink_vec<-c(0.01,0.05,0.1,0.25,0.5)
depth_vec<-c(3,4,5,6,7,8)
cv_auc_list = list("")
overall_index=0
for (sh_index in 1:5)
{
  for (depth_index in 1:6)
  {
    overall_index = overall_index+1
    gbm1<-gbm(Class~., data=down_crash, distribution="bernoulli", n.trees=3000,
shrinkage=shrink_vec[sh_index], interaction.depth=depth_vec[depth_index],
bag.fraction = .5, train.fraction = 1, n.minobsinnode = 10, cv.folds = 10,
keep.data=TRUE, verbose=F)
    print("I finished the following number of iteration:")
    print(overall_index)
    best.iter <- gbm.perf(gbm1,method="cv")
    yhat <-predict(gbm1,down_crash,n.trees = best.iter,type="response")
    # yhat_rounded<-round(yhat)
    y<-down_crash$Class
    # mis.class.rate = sum(y!=yhat_rounded)/length(y)
    # auc
    auc <-gbm.roc.area(y, yhat)
    print("AUC:")
    print(auc)
    cv_auc_list[overall_index]=auc
  }
}
print("AUC list is:")
cv_auc_list
```

2) *P_star sweep to identify the optimum p_star*

```
#
index=0
fnr_list = list("")
fpr_list = list("")
pstar_list = list("")
for (p_star in seq(from=0.4,to=0.6,by=0.01)){
  index=index+1
  yhat_binary<-ifelse(yhat>p_star,1,0)
  confusion=table(y,yhat_binary)
  fnr=confusion[2,1]/(confusion[2,1]+confusion[2,2])
  fpr=confusion[1,2]/(confusion[1,1]+confusion[1,2])
  pstar_list[index]=p_star
  fnr_list[index]=fnr
  fpr_list[index]=fpr
}
#
combined=cbind(pstar_list,fnr_list,fpr_list)
combined
```

3) Final CV to calculate the generalized FNR

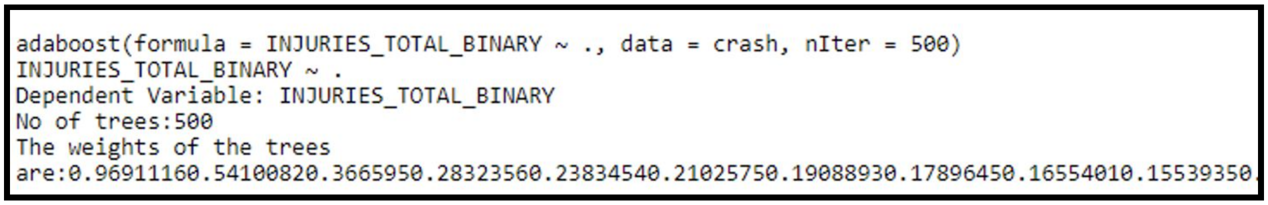
```
## use CV to find the best neural network model
set.seed(123)
Nrep<-1 #number of replicates of CV
K<-3  #K-fold CV on each replicate
n.models = 1 #number of different models to fit
n=nrow(down_crash)
y<-down_crash$Class
yhat=matrix(0,n,n.models)
FNR<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K)
  {
    train<-down_crash[-Ind[[k]],]
    test<-down_crash[Ind[[k]],]
    ##
    out<-gbm(Class~., data=train, distribution="bernoulli", n.trees=500, shrinkage=0.1,
interaction.depth=8, bag.fraction = .5, train.fraction = 1, n.minobsinnode = 10, cv.folds =
10, keep.data=TRUE, verbose=F)
    best.iter <- gbm.perf(out,method="cv")
    yhat[Ind[[k]],]<-predict(out,test, n.trees = best.iter,type="response")

  } #end of k loop
  # calculate the FNR rate for each model
  yhat_binary<-ifelse(yhat>0.5,1,0)
  confusion=table(y,yhat_binary)
  fnr=confusion[2,1]/(confusion[2,1]+confusion[2,2])
  FNR[j,]=fnr
} #end of j loop
FNR_avg<- apply(FNR,2,mean) #averaged mean square CV error
```

print("average FNR:")
FNR_avg

AdaBoost

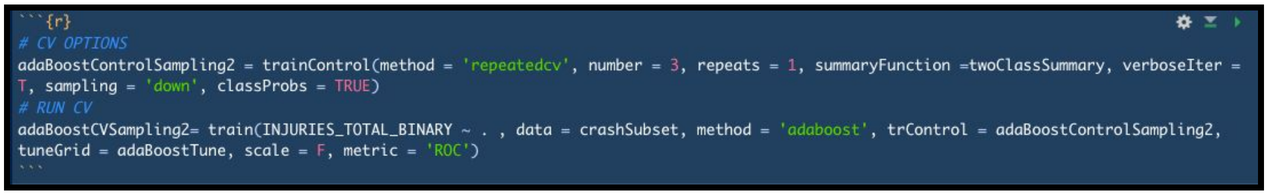
1) *Fitting AdaBoost with 500 trees on entire dataset:*

adaBoost1 = adaboost(INJURIES_TOTAL_BINARY ~ . , data = crash, nIter = 500)



```
adaboost(formula = INJURIES_TOTAL_BINARY ~ ., data = crash, nIter = 500)
INJURIES_TOTAL_BINARY ~ .
Dependent Variable: INJURIES_TOTAL_BINARY
No of trees:500
The weights of the trees
are:0.96911160.54100820.3665950.28323560.23834540.21025750.19088930.17896450.16554010.15539350.
```

2) *Tuning with the caret package on a subset of 100k rows, using ROC/AUC as measure of best model and downsampling to address class imbalance*



~~~
```{r}
# CV OPTIONS
adaBoostControlSampling2 = trainControl(method = 'repeatedcv', number = 3, repeats = 1, summaryFunction =twoClassSummary, verboseIter =
T, sampling = 'down', classProbs = TRUE)
# RUN CV
adaBoostCVSampling2= train(INJURIES_TOTAL_BINARY ~ . , data = crashSubset, method = 'adaboost', trControl = adaBoostControlSampling2,
tuneGrid = adaBoostTune, scale = F, metric = 'ROC')
```
~~~