Traffic Accident Prediction with Machine Learning

Russ Limber, R.L., Limber
Bredesen Center, University of Tennessee, Knoxville, USA, rlimber@vols.utk.edu
Sanjeev Singh, S.S., Singh
Bredesen Center, University of Tennessee, Knoxville, USA, ssingh@vols.utk.edu
EonYeon, E.J., Jo
Bredesen Center, University of Tennessee, Knoxville, USA, ejo1@vols.utk.edu

The intention of this project was to model vehicular accident severity for the six largest cities in the United States (i.e. New York City, Los Angeles, Chicago, Houston, Phoenix, and Philadelphia). The data was assessed using a binary target where one indicates a severe accident (as determined by experts) and a zero indicates an accident that is not severe. Our team utilized a variety of models to classify accident severity. Those models included: logistic regression, AdaBoost, XGBoost, random forest, gradient boosting classifier and multinomial naive Bayes. The results of these models revealed that Gradient Boosting performs the best when the data has been transformed using PCA with an overall accuracy of 0.85 and a macro F1-Score of 0.77 while XGBoost performs the best on the non-transformed dataset (full dataset) with an overall accuracy of 0.88 and a macro F1-Score of 0.82.

Additional Keywords and Phrases: Vehicles, Random Forest, XGBoost, AdaBoost, Gradient Boosting, Accident Severity

1 Introduction

Vehicular accidents make up approximately 38,000 deaths in the United States each year and cause about 4.4 million hospitalizations. If you are one of the 84% of Americans licensed to be on the road, then this report is for you (ASIRT, 2021). The primary goal of this paper is to model accident severity for the six largest cities in the United States using accident: location, weather conditions, road infrastructure, etc. Accidents that are categorized as more severe (higher cost of damage and bodily harm) are far more infrequent than accidents considered less severe. This skewness in the dataset makes classification challenging. The prevailing model with the best evaluation scores is called the DAP (Deep Accident Prediction) model (Moosavi et al., 2019). DAP is able to score a standard F1-Score of 84.7%. Our intention is to see if we can create a simpler model that scores as well as or better than DAP, using the macro F1-Score, based on a subset of the original dataset. Furthermore, we intend to identify whether or not cities vary statistically with the severity of their accidents. Finally, we will identify the factors most responsible for predicting accident severity.

2 Related Work

Several published works have investigated this dataset; the approaches and performance of these analyses are described below:

Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights [1]

Moosavi and colleagues collected streaming traffic data using two APIs, "MapQuest Traffic" and "Microsoft Bing Map Traffic". These providers broadcast traffic events (accident, congestion, etc.) captured by a variety of entities - the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road networks. Because accidents occurrences are rare and because the dataset is sparse, they performed negative sampling to balance the frequency of samples between accident and non-accident classes.

For prediction, they used what they call the Deep Accident Prediction (DAP) model. It consists of 5 components. A recurrent component (LSTM with two recurrent layers, each of 128 LSTM cells), which is used on the time-series columns, and an embedding component that encodes the spatial information. They also created a description-to-vector component that utilizes the natural language description of historical traffic events using Desc2Vec and a points-of-interest component that uses a representation of the spatial characteristics with a POI vector of size 13. Finally, they created a fully connected component that is fed with the output of all the previous components to make the final

prediction. It consists of 4 layers (512-256-64-2 units), uses batch normalization after the second and the third layer, and implements Sigmoid activation in the final layers and ReLU for the previous layers.

In addition to the DAP model, three baseline models were evaluated: Logistic Regression, Gradient Boosting Classifier, and Deep Neural Network (DNN). The F-1 score was used to compare the performance of all four models for six cities: Atlanta, Austin, Charlotte, Dallas, Houston, and Los Angeles. For the weighted-average F1 score, the Gradient Boosting classifier was the best performing, with Logistic Regression being the next best performing one (Moosavi et al, 2019).

RFCNN: Traffic Accident Severity Prediction Based on Decision Level Fusion of ML and Deep Learning Model [2]
Manzoor and colleagues predicted traffic accident severity by combining machine learning and deep learning techniques. Specifically, a combination of random forest and convolutional neural networks (RFCNN) was used. Model performance was compared to that of base learning models, which included AdaBoost classification, gradient boosting, random forest, extra tree, and voting classifiers. The model was applied in two phases. In the first phase, all of the features in the dataset were used to predict the outcome whereas only the top twenty features, as determined by random forest feature importance, were used in the second phase. Model performance was evaluated using accuracy, precision, recall, and F-score. The dataset was divided into 70% training data and 30% testing data. In the models using all of the dataset features, the RFCNN model had the highest accuracy, precision, recall, and F-score, all of which had values greater than 0.81. Among the models that used the twenty most important features, RFCNN outperformed all other models, with performance metric values greater than 0.97 (Manzoor et al. 2021).

Evaluating the Performance of Explainable ML Models in Traffic Accidents Prediction in California [3]
This recent report by Parra and colleagues describes the use of the U.S. vehicle accidents dataset to predict the occurrence of traffic accidents in the state of California. To extend the dataset to include non-accidents, each entry was reproduced twice, assigned a time that was either 1 hour prior to or following the accident, and labeled as a non-accident. Investigators obtained weather and climate data from an external source and joined it with these non-accident entries based upon location and time. Of the 48 features in the dataset, the investigators retained 28 relating to location, weather, and environmental conditions. The dataset was split into 70% training data and 30% validation data. After data augmentation and preparation, three tree-based algorithms (decision trees, gradient boosted trees, and random forest) were used for classification, and performance was evaluated using the F1-score. The algorithm with the best predictive performance was the gradient boosted tree model, with an F1-Score of 74% (Parra et al, 2020) .

3 Data

The dataset for this project was collected by researchers at The Ohio State University. The data collection process was a collaboration between the research team as well as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. The research team went on to publish: "Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights." The authors go on to explain how F1 Score for accidents was the most significant criteria for testing their neural network against their baseline methods (logistic regression, deep neural network and gradient boosting classifiers) and that DAP outperformed all of the baseline methods (Manzoor et al. 2021).

The dataset consists of 1.5 million observations, where each sample represents a vehicle accident that occurred in the United States between 2016 and 2020. To make the size of the dataset more manageable, our analysis will be restricted to major US cities with respect to population size, i.e. New York City, Los Angeles, Chicago, Houston, Phoenix, and Philadelphia. This dataset has forty-seven features, which can be divided into four broad categories:

1. **Event Log:** This includes information related to the time and duration of the accident, miles of traffic that were impacted by the accident, and a brief description of the event.
2. **Location:** Location features include the geographic coordinates of the accident site, as well as other information such as the Street, State, City, County, and Zip Code.
3. **Weather Conditions:** These variables describe weather conditions at the time of the accident, such as temperature, Humidity, Wind Chill, Pressure, and Visibility.

4. **Road Infrastructure:** These features specify the road infrastructure present at or near the accident site, like Bump, Crossing, Roundabout, Traffic Signal, and others.

For this project, supervised learning approaches will be used to analyze the US Accidents dataset, with accident severity (with respect to the impact on traffic) as the target variable. Accident severity is ranked from least to most severe, expressed as an ordinal variable ranging from 1 to 4. The dataset includes all the necessary information for this investigation, and therefore no external data sources will be needed. Several published works have investigated this dataset; the approaches and performance of these analyses are described below.

4 Data preprocessing

Downsampling

The U.S. Accidents dataset initially consisted of every vehicular accident in the continental United States. Prior to preprocessing, the dataset started off with ~1.5 million observations. To reduce the number of observations to a more manageable size, we decided to only analyze data from the six largest cities (Phoenix, Los Angeles, New York, Philadelphia, Houston, and Chicago). This brought the total number of observations included for analysis to 90,421.

Missing Values

Most of the forty-seven variables within our dataset did not contain missing values. The street number variable ('Number') was missing about 75% of its data. Since this variable was not necessary for the analysis, it was removed altogether. Approximately 30% of the data for the variable 'Windchill' was missing. We used Ordinary Least Squares Regression (OLS) using the wind speed and temperature variables, to create an equation to interpolate windchill: Windchill = 1.0178*Temperature - 0.3023*Wind_Speed. This model proved effective with an adjusted R^2 = 0.99. The remaining numeric variables were imputed using a kNN algorithm with k = 3. Since the missing data was all related to weather patterns, all of the weather pattern data (except wind direction) was numeric, and we were unaware of the true underlying distribution, kNN was a natural choice.

Encode, Standardize, and Extract Features

After removing non-relevant features, we had a mix of both categorical and numerical variables in our dataset. We also performed feature extraction and created features like Duration, Month, DayOfWeek, Hour using start_time and end_time. Also, to handle features like latitude and longitude, we transformed them to Cartesian coordinates and extracted X, Y, and Z coordinates, which would help in the feature standardization step. After feature extraction, we had 22 categorical variables and 12 numerical variables. All the categorical variables were unordered, and we performed label encoding to create binary encoding for 14 categorical variables. On the remaining 8 variables, we used one-hot encoding to create a binary column for each category under one feature; at the end, after removing unnecessary columns, we had 142 features to do downstream work. We also transformed the target labels combining severity (1, 2) as 0 and (3,4) as 1 to set the problem as a binary classification task.

Train Test Split

After encoding and feature extraction, we did a stratified random split based on Severity and created training (80% / 72336 rows), validation (10% / 9042 rows) and test (10% / 9043) datasets. The data split was done to handle data standardization and dimensionality reduction to avoid leaking any test and validation data information in training our models.

Standardization & Dimensionality Reduction

After splitting the dataset into training, validation, and testing sets we performed feature standardization using the training data so that we don't leak any information from the validation and test set while training our models. Further, we used PCA for dimensionality reduction to reduce the computational burden. By preserving 95% of the information, we ended up with 47 features to work with.

5 Methods

For performing the binary classification task, we evaluated six algorithms. We primarily relied on the Scikit-Learn library for the algorithm implementation, the only exception being XGBoost which comes as a separate package. Most of our models were ensemble methods based on trees, namely: AdaBoost, XGBoost, Random Forest, and Gradient Boosting. The other methods we tried were Logistic Regression which we used as a baseline, and finally Naive Bayes.

1. **Logistic Regression:** In its basic form, logistic regression produces weights for its covariates through gradient descent. Instead of using a linear term, a probability is first computed by feeding the value into the logit function. These are the values that gradient descent is optimizing. Once weights and a bias term are selected, we can multiply our values to the regression equation and determine whether it is 0 or 1 based on a threshold that we preset.
2. **Naive Bayes:** Naive Bayes is using Bayes Theorem which states that the posterior probability is equal to the conditional PDF, times the prior probability, divided by the normalizing PDF or the aggregate of the PDFs for both classes. The model is asking: given the data for a cell of that row and column, which posterior probability is going to be greater? Then it simply chooses the larger value.
3. **AdaBoost:** AdaBoost is an ensemble technique that uses weak learners; in our case, the learner was decision stumps (one-level decision tree). The way the algorithm works is it adds new learners to the ensemble such that they try to do better on samples that the previous learners misclassify. A given weight is assigned to each data point at each stage, which signals to the next learner how important it is to predict it correctly. In the end, it combines results from all the learners to make the final prediction.
4. **Random Forest:** The general idea is that random forest produces many decision trees and then uses bagging which is a way of grouping and averaging the results of those trees. It then moves on to eliminate certain groups of trees while favoring others (this is a very simplified definition).
5. **XGBoost:** We used another tree-based ensemble method to perform classification. XGBoost internally uses the same algorithm as Gradient Boosting, and it is designed for speed and performance. The distinguishing feature that we found in this implementation was the scope to perform regularization, it had parameters to perform L1 & L2 regularization. Since we were dealing with an imbalance dataset, we also used the parameter scale_pos_weight to tune our model further. Briefly, the way algorithms work, at each stage, adds a new learner to correct the errors made by models already added to the ensemble, and it keeps adding models to the ensemble until no further improvements can be made. Also, it borrows gradient in its name because it uses gradient descent to optimize the loss when adding the new models.
6. **Gradient Boosting:** Although we earlier used XGBoost to perform the classification task, which primarily used the same algorithm as Gradient Boosting, our primary motivation was to see how it fares against XGBoost in terms of speed and accuracy. In the next section, we'll find that with comparable results, XGBoost was faster to train.

6 Hyperparameter Tuning

Computation Speed

For each model we opted to use a different number of hyperparameter combinations or fits. This was done to save on runtime since some models such as naive Bayes can handle thousands of fits in only a few minutes while others such as gradient boosting classification requires one and a half hours just to perform ten fits. You will notice in the Table 1 below that when used on the PCA transformed dataset the naive Bayes was the fastest algorithm by far, taking only 0.015 seconds per fit. The gradient boosting classifier was the slowest, taking 630.44 seconds per fit. Also, the same can be said for the Table 2 using the full dataset where we see that naive Bayes is again the fastest taking 0.038 seconds/fit and gradient boosting classifier the slowest taking 128.05 seconds per fit.

Tuning Logistic Regression

We used a fairly wide variety of parameters to tune SKLearn's logistic regression model. Our final model chose to not use warm start meaning that it did not use the previous solution to initialize the weights for the upcoming solution (this was a fairly arbitrary dismissal). The solver was selected to be LBFGS. Basically this is a method for approximating the second derivative of the function used in gradient descent. It is also the most commonly used and is a default in

SKLearn. The model opted to not use a penalty term for creating the weights but did choose to calculate an intercept. Calculating an intercept and adding it to the weights for logistic regression will often improve the model accuracy. Lastly, the model opted to use C = 0.73 which means that there was some regularization (or smoothing) to prevent overfitting. This may explain why logistic regression didn't seem to overfit the model.

Table 1: Hyperparameter tuning time on the PCA dataset

PCA Transformed Dataset			
Classifier	Time to Tune (s)	Hyperparameter Combinations (fits)	Time to Tune per Fit
Multinomial Naive Bayes PCA	232.44	16000	**0.015**
Logistic Regression with PCA	148.86	375	0.397
Random Forest with PCA	3580.03	300	11.933
XGBoost with PCA	1256.64	96	13.09
AdaBoost with PCA	8286.32	28	295.94
Gradient Boosting PCA	6304.4	10	**630.44**

Table 2: Hyperparameter tuning time on the PCA dataset

Full Dataset			
Classifier	Time to Tune (s)	Hyperparameter Combinations (fits)	Time to Tune per Fit
Multinomial Naive Bayes	603.46	16000	**0.038**
Logistic Regression	500.99	375	1.336
XGBoost	3376.76	116	29.11
Random Forest	13325.97	300	44.42
AdaBoost	1811.04	28	64.68
Gradient Boosting	1280	10	**128.05**

Tuning Naive Bayes

This algorithm is incredibly easy to tune because it takes very few hyperparameters to begin with and can produce results very quickly. The alpha parameter indicates smoothing of the model. This algorithm chose to use a very small alpha of 0.005. Similar to the C parameter in logistic regression, the alpha is responsible for regularizing the model to prevent overfitting by using the Laplace-Lidstone method. Interestingly, while logistic regression had better accuracy

though smoothing, naive Bayes chose to almost not smooth at all. Furthermore, the naive Bayes model chose to not utilize class priors but rather treat the initial probabilities as a uniform distribution.

Tuning Random Forest

Tuning random forest is fairly straightforward but time consuming. One problem that you tend to run into is overfitting which we found was most likely to occur if the number of trees is too high or if the maximum number of levels per tree is too high. While the algorithm favored more trees to improve accuracy on the training dataset (366), we purposefully lowered that value to 100 in order to curb overfitting (although it still overfit on us). Otherwise the model selected standard parameters such as setting the Gini index for its criteria. Also the minimum number of samples to split at a node was set to two which tells us that the model is highly discerning.

Tuning Gradient Boosting Classifier

This is a challenging model to train because of how long it takes to fit. Therefore, one cannot simply produce a large number of fits unless they have the time and or computing resources to do so. We started off trying to produce 96 fits and assumed that it would converge if we let it run all night but even that proved to not work. So as a result we really only have a rough estimate of what the optimal parameters are. We selected 100 decision trees as our number of estimators and found 20 levels per tree to be the most effective from our sample size of 10 fits. We also found that a learning rate of 1.0 seemed to produce good results even compared to lower learning rates (i.e. 0.1 and 0.01).

Tuning XGBoost

To tune XGBoost we mainly tried changing the parameter such as n_estimator which controls how many stages to add; learning rate, which controls the weighting factor for each tree added to the ensemble; max_depth that determines how deep we want to grow the tree; scale_pos_weight that handles the imbalance in the class, an optimal value for this parameter was to take a ratio number of samples in the larger class to the smaller one. Lastly, to deal with overfitting we used reg_alpha that provides L1 regularization and reg_lambda that provides L2 regularization. Overall, for XGBoost, the tuning performance per fit was better than other tree-based ensemble methods that we used. While tuning the model, we found that large max_depth values gave us a good validation score, but at the same time, the model suffered from overfitting.

Tuning AdaBoost

To tune the AdaBoost, the scikit-learn library didn't provide many parameters to tune. The parameter that we selected were similar to other tree-based methods like n_estimator, which governs the maximum number of learners we can add to the model, and the learning rate, which controls the weight applied to each classifier. Similar to what we observed with Gradient Boosting, AdaBoost took more time to train on the PCA dataset than the entire dataset. The latter was counterintuitive as with PCA, we had fewer dimensions to handle, and ideally, the algorithm should have taken less time.

7 Results

As shown in the Table 3 the random forest and gradient boosting classifier tied for the best overall accuracy at 0.85 for the PCA transformed dataset. The best recall value was recorded by the naive Bayes classifier at 0.73 and the best specificity recorded by random forest at 0.96. The highest value for precision was also the random forest with 0.78. Lastly the highest macro F1 Scores were tied between XGBoost and gradient boosting classifier at 0.77.

Pertaining to the Table 4 XGBoost had the best overall accuracy at 0.88 for the full dataset. The best recall value was recorded by the naive Bayes classifier at 0.67 and the best specificity recorded by random forest at 0.95. The highest value for precision was also the random forest with 0.79. Lastly the highest macro F1 Scores was produced by XGBoost with a score of 0.82.

Table 3: Classifier results on the PCA dataset.

Final Results PCA Transformed Dataset					
Classifier	**Overall Accuracy**	**Recall**	**Specificity (tn/n)**	**Precision**	**F1 Score (macro)**
Logistic Regression with PCA	0.81	0.36	0.94	0.65	0.67
Random Forest with PCA	**0.85**	0.49	**0.96**	**0.78**	0.75
AdaBoost with PCA	0.82	0.43	0.94	0.67	0.7
XGBoost with PCA	0.83	0.7	0.87	0.61	**0.77**
Multinomial Naive Bayes PCA	0.74	**0.73**	0.75	0.46	0.69
Gradient Boosting PCA	**0.85**	0.56	0.94	0.74	**0.77**

Table 4: Classifier results on the Full dataset.

Final Results Full Dataset					
Classifier	**Overall Accuracy**	**Recall**	**Specificity (tn/n)**	**Precision**	**F1 Score (macro)**
Logistic Regression	0.82	0.41	0.94	0.67	0.7
Random Forest	0.87	0.58	**0.95**	**0.79**	0.79
AdaBoost	0.84	0.55	0.93	0.69	0.76
XGBoost	**0.88**	0.66	0.94	0.78	**0.82**
Multinomial Naive Bayes	0.77	**0.67**	0.8	0.5	0.71
Gradient Boosting	0.87	0.66	0.94	0.76	0.81

8 Discussion

The naive Bayes model chose to not utilize class priors but rather treat the initial probabilities as a uniform distribution. This is perhaps the strangest take-away from the entire project (at least to Russ) because the dataset is strongly skewed with the actual priors being 23% for class 1 and 77% for class 0. Yet for some reason this model opted to treat them as 50/50. Also it had a wide landscape to choose from with 16,000 fits which tells us that this is likely to be the "true best model."

We noticed that the AdaBoost and random forest classifiers took longer to tune on the PCA dataset than they did on the full dataset. To make matters even more curious we note that random forest and XGBoost models were the opposite, and tuned faster on the PCA than they did on the full dataset despite also being based on decision trees. We are unclear about this result as well.

We selected XGBoost as our model based on our results as it gave the highest accuracy on the test dataset. To dive deeper, we looked at the feature importance scores (Figure 1) to know what features are considered more important in predicting the severity of accidents. Although we didn't see much difference in the mean duration across the severity, the distance feature turned out to be the most important. Next was coord_z, which is directly related to the location of the accident and along with coord_x (8th important feature), and coord_z (7th important feature) can provide the information that can help classify the severities. The next set of features like Distance(mi), Sunrise_Sunset, and City_Chicago were important as, based on our EDA, they showed patterns that can help in classification. For example, Chicago had more severe one accident. At night, the gap between severe one accident and severe 0 accidents is low, meaning there are more chances of having a severe one accident at night than in the daytime. We also looked at the feature importance scores (figure y) of Random Forest and saw that there was a consensus in the top 4 features which further substantiate our understanding, i.e., what factors we can use to predict the severity of an accident.

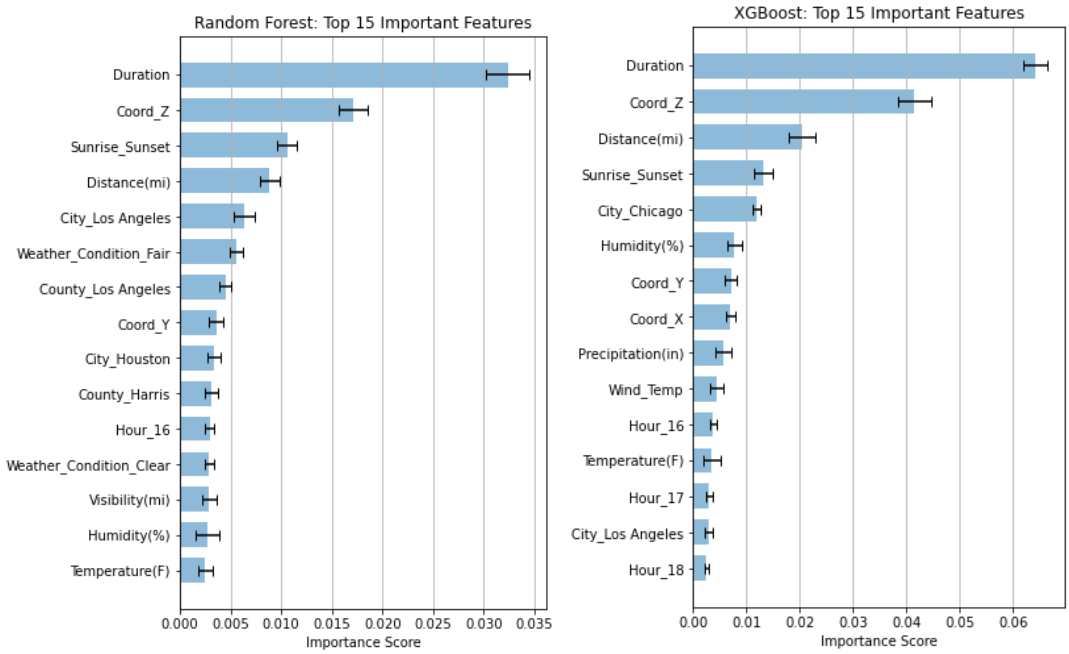


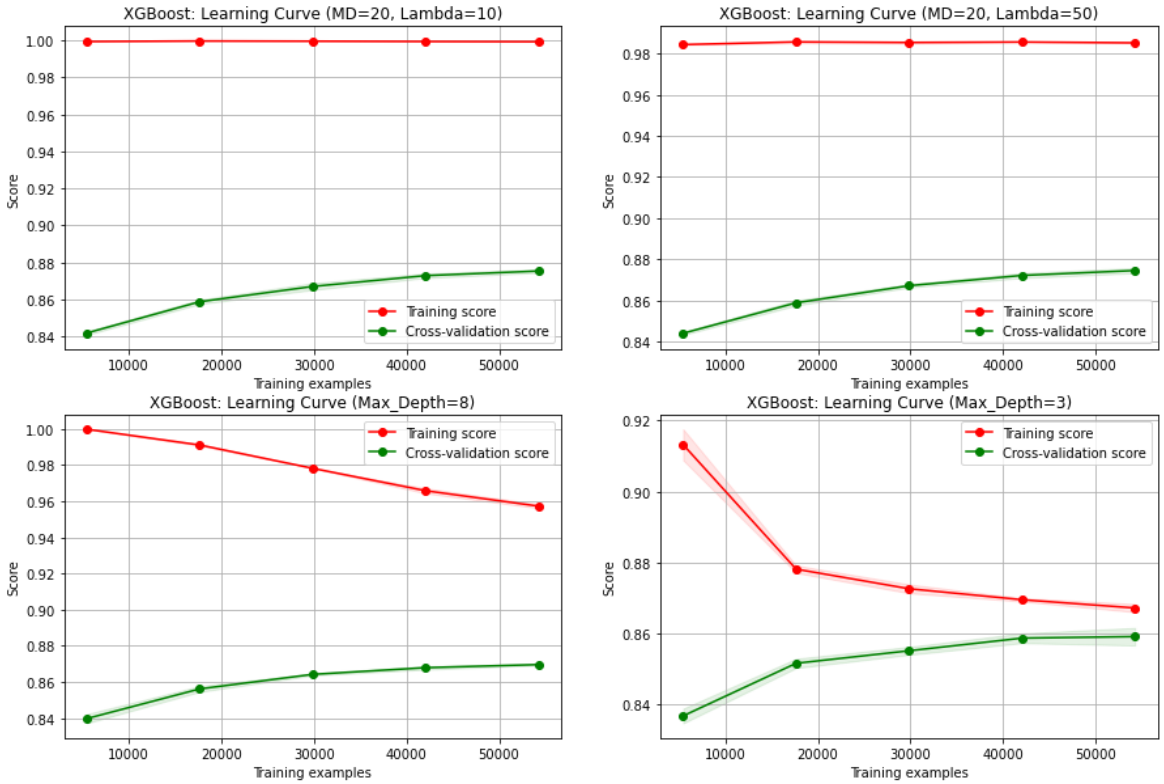
Figure 1: Feature importance from Random Forest and XGBoost.

Even after selecting our final model, we wanted to know how we could tweak the model parameters to see if we could improve the cross-validation score. We plotted our best model's learning curves (Figure-2) and found that it suffered from high variance and almost overfitted the training dataset. To leverage the wide gap between training and validation scores, we tried to introduce a higher regularization parameter lambda; it lowered the training error a bit increasing the bias, but we didn't observe much improvement on the cross-validation score and the model still had a high variance. Next, we tried regularization by using shallow trees to have a more generalizable model. Still, even at the expense of increasing the model bias, there was no significant lift in the cross-validation score. Finally, the training and cross-validation scores almost converged with a very shallow tree, but even with an expense of higher bias, the cross-validation score didn't improve, but it dropped. Thus, with our selected model, the regularization seemed to have little effect on improving the cross-validation score. One way to improve the results is to provide more data at the training stage, but the way the cross-validation curve is flattening out, we are quite not sure if it's even going to work out.

9 Conclusion

The intention of this project was to model vehicular accident severity for the six largest cities in the United States (i.e. New York City, Los Angeles, Chicago, Houston, Phoenix, and Philadelphia). The data was assessed using a binary target where one indicates a severe accident (as determined by experts) and a zero indicates an accident that is not severe. Our team utilized six classification models: logistic regression, AdaBoost, XGBoost, random forest, gradient boosting classifier and multinomial naive Bayes. The results of these models revealed that Gradient Boosting performs the best when the data has been transformed using PCA whereas XGBoost performs the best on the full dataset. We believe that our results were comparable but not quite as strong as DAP. The team at The Ohio State University were able to classify severity with an F1 Score of 0.874. We were not able to attain macro F1 Scores that high however we were able to produce a score of 0.82 on a much smaller subset of the data.

Figure 2: Learning curves for the XGBoost model.



REFERENCES

[1] Moosavi S, Samavatian MH, Parthasarathy S, Teodorescu R, Ramnath R. Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights. GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems. 2019 Sep 19;33–42. Available from: http://arxiv.org/abs/1909.09638

[2] Manzoor M, Umer M, Sadiq S, Ishaq A, Ullah S, Madni HA, et al. RFCNN: traffic accident severity prediction based on decision level fusion of machine and deep learning model. IEEE Access [Internet]. 2021;9:128359–71. Available from: http://www.safetylit.org/citations/index.php?fuseaction=citations.viewdetails&citationIds[]=citjournalarticle_696455_34

[3] Parra C, Ponce C, Rodrigo SF. Evaluating the Performance of Explainable Machine Learning Models in Traffic Accidents Prediction in California. Proceedings - International Conference of the Chilean Computer Science Society, SCCC. 2020 Nov 16; Available from: https://ieeexplore.ieee.org/abstract/document/9281196.

GITHUB: https://github.com/DSE511-Project3-Team/DSE511-Project-3-Code-Repo