CSE 5523 Project: Road Accident Severity Classification Using US Accidents Dataset

Yuvraj Singh

Abstract—**Fewer drivers were on the roads during the onset of COVID-19 pandemic because of the stay-at-home mandates and workplace policies aimed at curbing the viral spread. These stay-at-home policies led to a significant reduction in car traffic, hence reducing urban traffic congestion. This study aims to classify the severity of road accidents using the US Accidents dataset link here covering road accidents from 49 contiguous US states from February 2016 through December 2020. The road accidents can be qualitatively described to come from one of these severity classes: low, medium and high severity. This is an exploratory study employing several machine learning approaches including ensemble methods: random forests, heuristic multi-class classifiers: One vs All and One vs One strategies for perceptron and support vector machine binary classifiers and multi-layer neural networks. Model evaluation is based on confusion matrices and the models are validated using cross-validation methods and binomial significance testing. GitHub Link: link**

I. Introduction: Dataset Description

Note: This report describes data cleanup procedure, the base model (logistic regression), random forest with adaptive boosting, heuristic-based multi-class classification using perceptron binary classifier, various neural network architectures, evaluation metrics based on confusion matrix and cross validation. The remaining portion of the study is described in [1].

The dataset described in [2][3] is a large scale dataset created using several APIs that stream real-time data related to traffic events collected by US Department of Transportation and Law enforcement agencies through traffic cameras and traffic sensors. The original dataset has 47 variables that can be categorized under traffic (severity, accident start and end time, distance affected); geography (street, city, etc.); weather (temperature, wind, humidity, etc.); Points of Interest (POI) such as cafes and train stations; and time of day (sunrise, sunset, civil twilight, nautical twilight, and astronomical twilight). In this study, severity class is used as the target variable and the rest of variables as feature variables. The data creation process as followed by the original paper is shown in Figure 1.

II. Feature Selection: Data Cleanup

Script: `cleanData.py`

The data cleanup process using the `cleanData()` function involves processing `datetime` variables to compute the duration of the incident by subtracting the valid ISO format time stamps for start and end time of an incident. A `bool` variable called `preCovid` is generated such that `preCovid = True` if the incident occurred before February 2020, else it is zero. Certain variables like street number, zipcode, country (since only US data is collected), airport code, natural language description (word embeddings are not explored in this study), latitude and longitude, etc. that don't lead to meaningful quantitative data are eliminated. The `NaN` values are dropped from the dataset. The variables being considered after data cleanup are shown in the Appendix in Figure 2. After converting all categorical predictors to one-hot variables, $K = 207$ variables are available. Out of over 1.5 million observations, 60.02% of the observations are retained after removing missing data. The predictor features are selected using the following Pearson correlation threshold:

$$X = \{X_k \text{ such that } |corr(X_k, y))| > 0.05 \\ \text{where } k \in \{1, 2, 3, \cdots, K\}\}$$

III. Methodologies

Script: `regressionLib.py`

The confusion matrix computed using `confusionMatrix()` function was used as the scoring criterion for the classifiers and the metrics: overall accuracy, user's accuracy (precision), producer's accuracy (recall) and kappa coefficient were derived from the confusion matrix using the `metrics()` function. The overall accuracy is the ratio of sum of diagonal values to total number of cell counts in the confusion matrix. The user's accuracy for a certain class A is the proportion of predictions classified as class A that were actually class A. The producer's accuracy for class A is the proportion of actual class A observations predicted correctly as class A. The kappa coefficient is a measure of the overall agreement of the confusion matrix. If confusion matrix has diagonals significantly higher than off-diagonals, kappa coefficient is high. More details on these metrics can be found in remote sensing literature [4].

A. *Base Model: Logistic Regression*

The logistic regression, being one of the most simple multi-class classifiers is chosen as the base model. The selected predictors are used to model accident severity and the model achieves an overall accuracy of 88.5% (refer Figure 4), however the confusion matrix and the metrics computed from it show very low producer's accuracy for high severity accident predictions.

B. *Data Resampling*

This abysmally low producer's accuracy for high severity class is found for other multi-class classifiers as well (refer Figures 3,4). Upon closer inspection, it was revealed that the data has very high class representation of low severity accidents and very low representation of high severity accidents leading the model to learn parameters optimal for low severity class at the expense of high severity class. As a result, the training data is resampled using the function `resampleData()` to achieve equal class representation for every target class. The confusion matrix is shown in Figure 5, which results in a reasonably good producer's accuracy for each class at the expense of overall accuracy.

IV. Models Implemented

A. Random Forest with Adaptive Boosting

Random Forest with adaptive boosting implements multiple decision tree classifiers. The main idea is to train multiple decision trees (weak learners) on randomly selected observations. Subsequent weak learners compensate for training error of predecessors (Boosting).

B. Perceptron: One vs All and One vs One

In this approach, heuristic meta-algorithms are used to train multiple binary classifiers using two different schemes to select binary classifiers. The multi-class classification is achieved by using the most confident classification (One vs All) or the most popular classification (One vs One). The two heuristic approaches are described in Figure 8. In this report, the binary classifier used is perceptron. Support vector machine is implemented in [1]. The heuristic-based classifier failed to provide very good results as compared to other multi-class classifiers (refer Figure 6), hence it is not explored further.

C. Fully-Connected Multi-Layer Neural Networks

A fully connected neural network with 3 hidden layers with each layer having 50 hidden neurons is implemented. The model is compared with the other candidate models and the performance is explained in the next section. A neural network with higher complexity (refer Figure 9) is also implemented using TensorFlow/Keras framework, which however fails to provide adequate performance and hence is not considered for further analysis.

V. Model Performance Evaluation

Out of the above models, the best model is the Random Forest using adaptive boosting. The confusion matrix for the candidate models is shown in Figure 10.

A. Metrics based on Confusion Matrix

The credible intervals of test set accuracy and other metrics is computed using the code provided in homework assignments. The credible intervals of the candidate models' producer's accuracy metrics for each class is shown in Figure 11. Hence, it can be concluded that the The binomial significance testing (Figure 12) shows that the best model is superior in terms of overall accuracy as compared to the base logistic regression and other candidate models by a probability of 1. This probability can be explained by observing the beta distribution of test set performance (refer Figure 7), which is a very narrow distribution. Since, the test set has a lot of observations, the binomial significance test gives very confident results. The best model's performance is further verified using Monte-Carlo cross validation in the next section.

B. Cross Validation

The `class splitCV` (coded in the library script `regressionLib.py`) is used to implement test-train split, K-Fold splits and monte-carlo splits for cross validation of the best model. Figure 13 shows a schematic of these splitting procedures. Out of these approaches, the Monte-Carlo cross validation procedure is selected as the validation approach to supplement the model performance metrics obtained from credible intervals testing. Cross-validation using 10 Monte-Carlo splits shows consistent model performance (refer Figure 14). K-Fold and Monte-Carlo are comparable in terms of performance. However, Monte-Carlo tends to be more repeatable (less variance) since any desired amount of statistically different splits can be generated because splits are performed independently. However, it tends to have higher bias than the corresponding value for K-Fold [5]

VI. Code Integration

The script `main.py` Implements all the models. The code is executed in an iPython Jupyter notebook environment using some required command-line arguments to control plot generation. All plots are generated in `yuvraj.ipynb`. The scripts `regressionLib.py` and `cleanData.py` are library scripts that are never executed on their own, but contain various helper functions.

VII. Discussion: Summary/Conclusion

Unbalanced class representation can lead to models that learn features related to the dominant class at the expense of another class that occurs rarely, but may have catastrophic effects if the class label is encountered in the data. Such datasets are very common in domains like safety systems engineering, where the task is usually to detect rare, but catastrophic events like faults. As a result, more robust evaluation methods were applied in this project than simply using accuracy. The data was resampled to achieve even class representation. Since, the class membership of each class in the training dataset is ensured to be similar, the overall accuracy can be used as a model selection criterion. Out of all the candidate models, the random forest with adaptive boosting was the best performing model. The model's classification was consistent across test sets as evidenced by performance on 10 Monte-Carlo splits.

REFERENCES

[1] A. Dahir, "CSE 5523 project: Road accident severity classification using US accidents dataset," 2021.

[2] S. Moosavi, M. H. Samavatian, S. Parthasarathy, and R. Ramnath, "A countrywide traffic accident dataset," *CoRR*, vol. abs/1906.05409, 2019. [Online]. Available: http://arxiv.org/abs/1906.05409

[3] S. Moosavi, M. H. Samavatian, S. Parthasarathy, R. Teodorescu, and R. Ramnath, "Accident risk prediction based on heterogeneous sparse data: New dataset and insights," *CoRR*, vol. abs/1909.09638, 2019. [Online]. Available: http://arxiv.org/abs/1909.09638

[4] G. Banko, "A review of assessing the accuracy of classifications of remotely sensed data and of methods including remote sensing data in forest inventory," IIASA, Laxenburg, Austria, IIASA Interim Report, November 1998. [Online]. Available: http://pure.iiasa.ac.at/id/eprint/5570/

[5] "K-fold and montecarlo cross-validation vs bootstrap: a primer," https://nirpyresearch.com/kfold-montecarlo-cross-validation-bootstrap-primer/, accessed: 2021-12-13.

APPENDIX: FIGURES

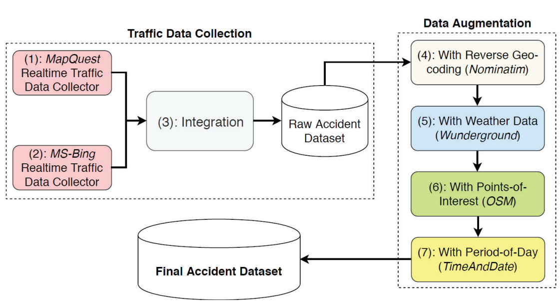

Fig. 1. Process of creating traffic accident dataset. Courtesy: Sobhan Moosavi [2]

```
Severity                    int64
Distance(mi)              float64
Side                       object
State                      object
Timezone                   object
Temperature(F)            float64
Wind_Chill(F)             float64
Humidity(%)               float64
Pressure(inches)          float64
Visibility(mi)            float64
Wind_Direction             object
Wind_Speed(mph)           float64
Precipitation(inches)     float64
Weather_Condition          object
Amenity                      bool
Bump                         bool
Crossing                     bool
Give_Way                     bool
Junction                     bool
No_Exit                      bool
Railway                      bool
Roundabout                   bool
Station                      bool
Stop                         bool
Traffic_Calming              bool
Traffic_Signal               bool
Turning_Loop                 bool
Sunrise_Sunset             object
Civil_Twilight             object
Nautical_Twilight          object
Astronomical_Twilight      object
incidentTimeLength_seconds float64
preCovid                     bool
```

Fig. 2. Output from `cleanData()` function: shows all variables before one-hot conversion

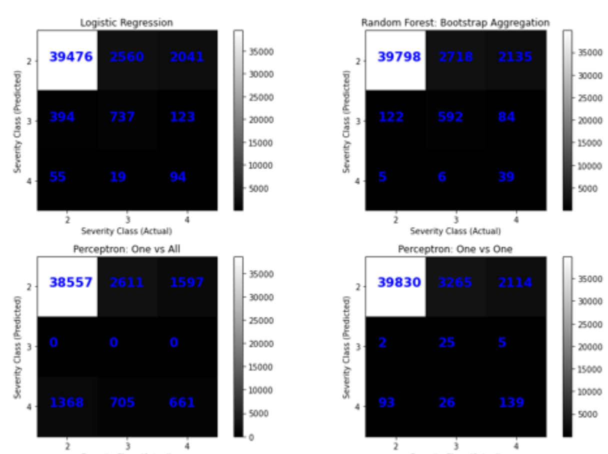

Fig. 3. Confusion matrix for models trained on datasets with highly uneven class representation

```
Overall Accuracy: 0.885
User's Accuracy: [0.895 0.587 0.5  ]
Producer's Accuracy: [0.988 0.228 0.04 ]
Kappa Coefficient: 0.232713
```

Fig. 4. Evaluation metrics for models trained on datasets with highly uneven class representation

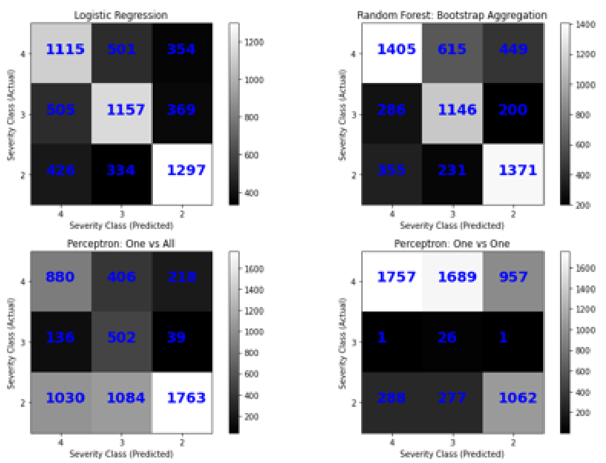

Fig. 5. Confusion matrix for models trained on datasets with highly even class representation

```
Perceptrons
One vs All
Overall Accuracy: 0.512
User's Accuracy: [0.664 0.462 0.531]
Producer's Accuracy: [0.238 0.729 0.579]
Kappa Coefficient: 0.270475

One vs One
Overall Accuracy: 0.493
User's Accuracy: [0.587 0.839 0.433]
Producer's Accuracy: [0.584 0.05  0.842]
Kappa Coefficient: 0.239509
```

Fig. 6. Performance metrics for heuristic-based multi-class classifiers: data has balanced class representation

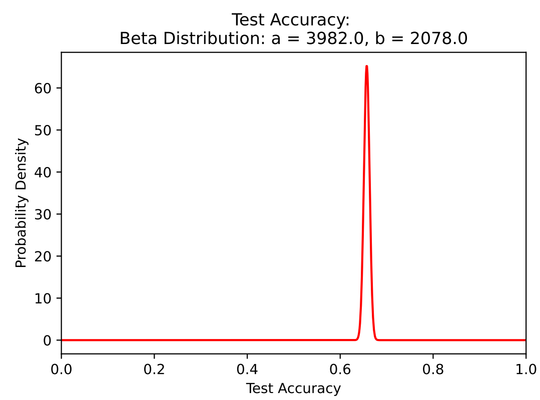

Fig. 7. Beta distribution of overall accuracy on test set: Random forest with AdaBoost (best model)

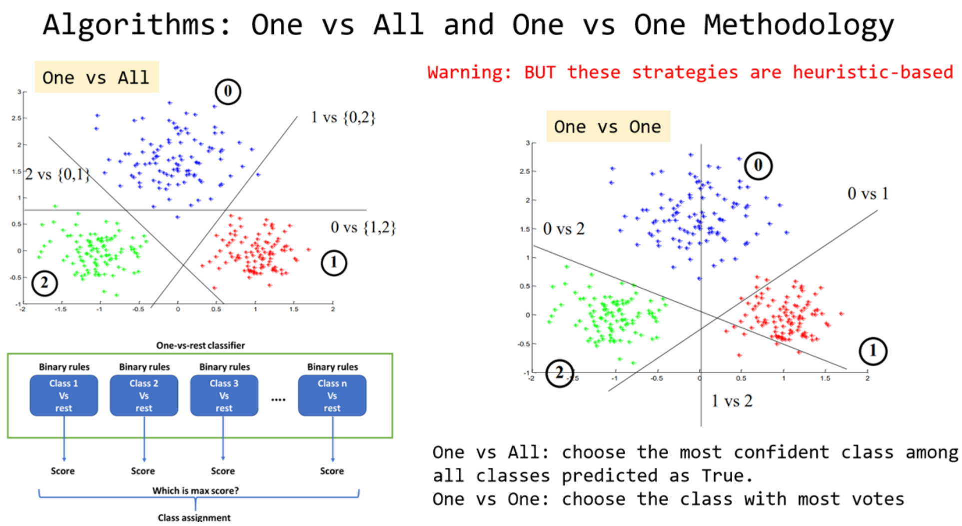

Fig. 8. One vs All and One vs One heuristic multi-class classification strategy.
Source: https://people.cs.pitt.edu/ milos/courses/cs1675-Spring2019/Lectures/class15b.pdf

```
h = 100    ## number of hidden units
N = XTrain.shape[0]
numEpochs = 10

X0 = tf.constant( XTrain , dtype=tf.float32 ) ## tensorflow format
Y0 = tf.constant( YTrainBinary , dtype=tf.float32 )
X0Test = tf.constant( XTest , dtype=tf.float32 ) ## tensorflow format
Y0Test = tf.constant( YTestBinary , dtype=tf.float32 )

model = tf.keras.models.Sequential([tf.keras.layers.InputLayer(input_shape=(XTrain.shape[1],)),
                                    tf.keras.layers.Dense(h, activation='softmax'),
                                    tf.keras.layers.Dense(h, activation='softmax'),
                                    tf.keras.layers.Dense(h, activation='softmax'),
                                    tf.keras.layers.Dense(h, activation='softmax'),
                                    tf.keras.layers.Dense(h, activation='softmax'),
                                    tf.keras.layers.Dense(YTrainBinary.shape[1],
                                    ↪  activation='softmax')
                                   ])
print(model.summary())
model.compile(loss='categorical_crossentropy',optimizer=tf.keras.optimizers.SGD(1.0))

from tensorflow.keras.callbacks import ModelCheckpoint,ReduceLROnPlateau,EarlyStopping
callbacks = [
    ModelCheckpoint("model.h5",verbose=1,save_best_model=True),
    ReduceLROnPlateau(monitor='val_loss',patience=3,factor=0.1,verbose=1,min_lr=1e-6),
    EarlyStopping(monitor='val_loss',patience=5,verbose=1)
]

model.fit(X0,Y0, validation_data = (X0Test,Y0Test), epochs=numEpochs)
```

Fig. 9. Neural network with high complexity implemented in TensorFlow/Keras

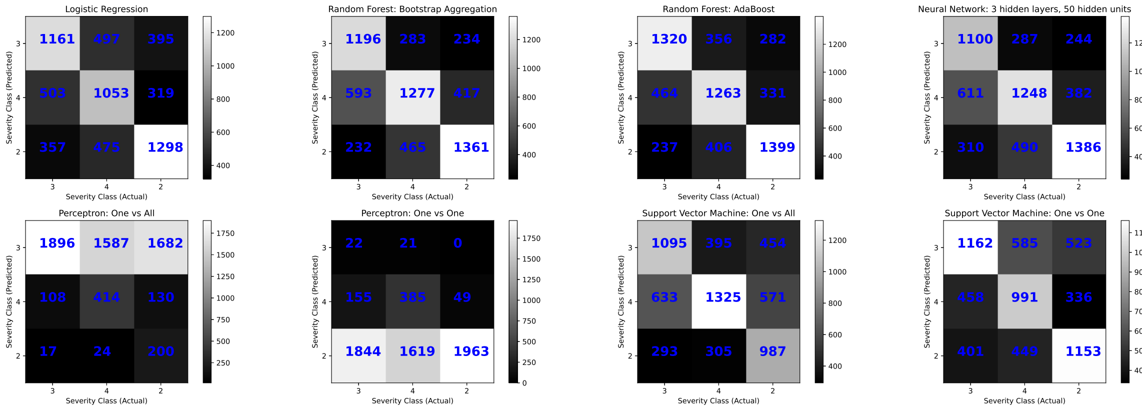

Fig. 10. Confusion matrix for candidate models

```
Logistic Regression
Credible interval for producer's accuracy
  Class: 2       Credible Interval:     mean: 0.744556427853841 +/- 0.010782878423567177
  Class: 3       Credible Interval:     mean: 0.7105307565778302 +/- 0.011297253710394473
  Class: 4       Credible Interval:     mean: 0.7040207089534526 +/- 0.011441248418083805
Credible interval for overall accuracy  mean: 0.5795712946697323 +/- 0.012648627341258378

Random Forest: Bootstrap Aggregation
Credible interval for producer's accuracy
  Class: 2       Credible Interval:     mean: 0.7772984123549281 +/- 0.010441314289261094
  Class: 3       Credible Interval:     mean: 0.7783798769453784 +/- 0.010444503330318211
  Class: 4       Credible Interval:     mean: 0.7094883835103649 +/- 0.01067849028330059
Credible interval for overall accuracy  mean: 0.6326982503725748 +/- 0.012562548694474596

Random Forest: AdaBoost
Credible interval for producer's accuracy
  Class: 2       Credible Interval:     mean: 0.7927396899585472 +/- 0.0104410336896541
  Class: 3       Credible Interval:     mean: 0.7790084254055665 +/- 0.009964324118019507
  Class: 4       Credible Interval:     mean: 0.7428173745724604 +/- 0.011793504535819732
Credible interval for overall accuracy  mean: 0.6575521349205048 +/- 0.011366299587973194

Neural Network: 3 hidden layers, 50 hidden units
Credible interval for producer's accuracy
  Class: 2       Credible Interval:     mean: 0.7646473319700947 +/- 0.010647385992344272
  Class: 3       Credible Interval:     mean: 0.7601719546709861 +/- 0.010357278515790047
  Class: 4       Credible Interval:     mean: 0.707774634945423 +/- 0.011782019678442723
Credible interval for overall accuracy  mean: 0.6165993230827137 +/- 0.011782910553360004
```

Fig. 11. Credible intervals of producer's accuracy and overall accuracy
Best model is found to be Random Forest with Adaptive Boosting

Binomial Significance Matrix: Classifier B (rows), Classifier A (columns)

Values: Probability of Classifier B having overall accuracy >A

	Logistic Regression	Random Forest: Bootstrap Aggregation	Random Forest: AdaBoost	Neural Network: 3 hidden layers, 50 hidden units
Logistic Regression	NaN	0.000	0.000	0.000
Random Forest: Bootstrap Aggregation	1.0	NaN	0.001	0.975
Random Forest: AdaBoost	1.0	0.997	NaN	1.000
Neural Network: 3 hidden layers, 50 hidden units	1.0	0.033	0.000	NaN

Fig. 12. Binomial significance testing

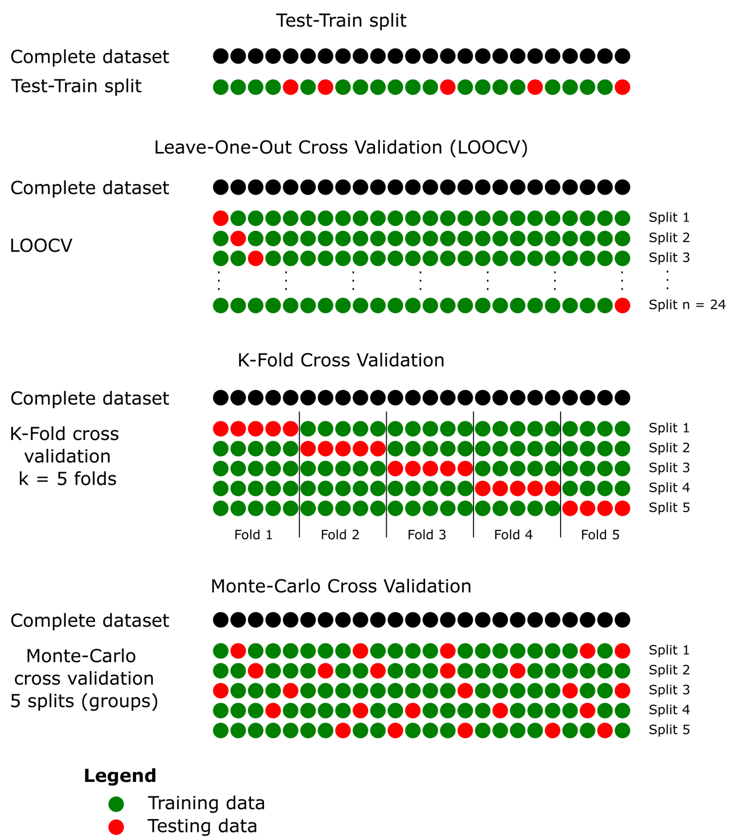

Fig. 13. Various Cross-Validation Strategies

```
Overall Accuracy for all cross validation splits:
[0.6571476  0.6541763  0.66391546 0.64328164 0.6467481  0.6559921
 0.64806867 0.6619346  0.6449323  0.6630901 ]

Mean Overall Accuracy for all cross validation splits: 0.6539286375045776
Stdev Overall Accuracy for all cross validation splits: 0.007366388104856014
```

Fig. 14. Overall accuracy scores on 10 Monte-Carlo Splits