Predicting the severity of road accidents in the UK.

Name: Raheel Shaikh

DATE: 10/04/2023

Word count: 3,238

Executive summary

To categorize the severity of the road accidents, we will be using the UK Road Accident Data to implement several machine learning classification techniques. By predicting the severity of accidents on the road, we can identify high-risk areas and better allocate resources, such as those for emergency and medical care.

In this report, five different types of traditional models were rigorously applied with various hyperparameters, and the Random Forest model emerged as the most accurate at classifying accidents as "slight," "serious," or "fatal" with an accuracy rate of 77.1% on test data. Moreover, classification using neural networks is put into practice. An accuracy of 78% on test data is what we see. To find the model that performs the best, the model was evaluated with several hyperparameters. A notable characteristic of the data set is that the accuracy of above models did not get effected after performing dimensionality reduction. This implies we got rid of insignificant fields from our data. After conducting several tests and experiments for both objectives, we got to the conclusion that the model 2 of Neural networks is the best classification algorithm since it has the highest accuracy and no overfitting.

1. Exploratory data analysis

Firstly, we use the '.describe()' functions to get a statistical insight into our data. We found that almost all the columns of our data set contain categorical values, while only 'speed_limit' and 'age_of_oldest_driver' are numerical values. This function returns some important characteristics of the data set like the count, mean, standard deviation, min, and max value in the data. We got to know that the 'speed_limit' column has a minimum value of -1 which is an anomaly and will be delt with in the data preprocessing stage. The 'age_of_oldest_driver' has a count of 25197 which is less that the total, this implies that this column contain 'NaN' values that will be handled in the next stage.

We may examine the distribution of data in each column to do further analysis on the dataset. To obtain the count of each distinct value in a column, we may use a "for" loop that iterates over all the columns using the ".value_counts()" function. It is crucial to keep in mind that many columns include missing or out-of-range data. This will need to be addressed at the data preprocessing stage. Also, we noticed that the values in the column "accident severity" have the same meanings but are written differently. For instance, the terms "fatal" and "fatal" have the same meaning. Our model may treat them as two distinct groups because of this discrepancy, which might have a detrimental effect on how well it performs.

Dimensionality reduction

Next, to check the balance of our target variable 'accident_severity' we plot a pie chart. The below pie chart in figure 1 show us that majority of our data set contains 'slight' as the accident category at 41.6% while 38.2 % and 20.2% for 'serious' and 'fatal' respectively. The data set does not seem have a very balance nature as we have 'fatal' data that is exactly half of 'slight' data category.

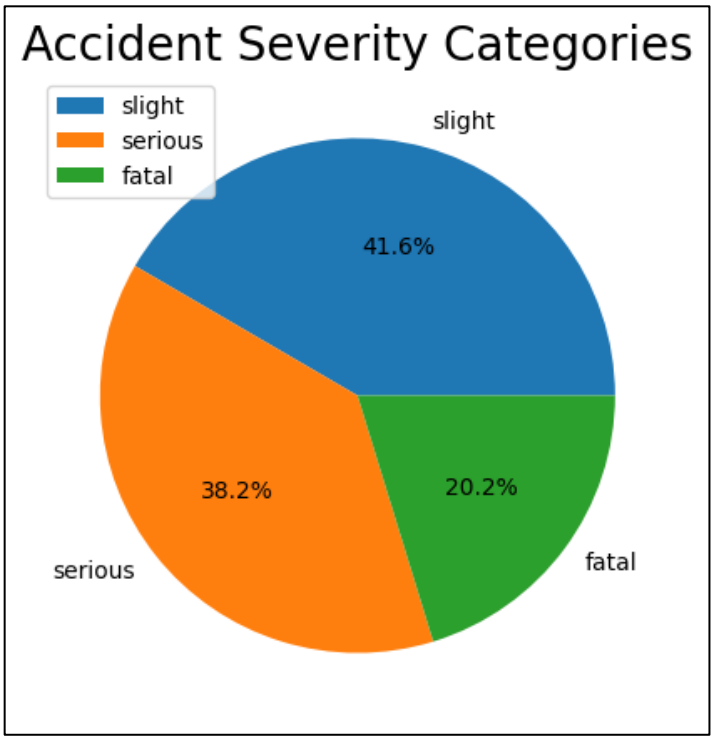

Figure 1:Pie chart of the target Variable

Principal component analysis

FAMD is used for mixed data containing categorical and numerical variables. It produces factors through PCA for numerical and MCA for categorical data, reducing dimensionality. Factors are linear combinations capturing data structure and relationships.

The contribution of each category to the first two components of FAMD is depicted in Figure 2 as a stacked bar chart. The height of each bar denotes the contribution to each component, and each bar represents a category divided into two halves. Component 1 is heavily influenced by the fields "hit_object_off_carriageway" and "vehicle_leaving_carriageway," but "sex_of_driver" has a minimal impact. In component 2, "road_surface_conditions" and "weather_conditions" provide significant contributions, whilst "age_of_oldest_driver" makes a negligible one. This shows that, in comparison to the former variables, the latter are less connected to the data variability in component 2. As "age_of_oldest_driver","sex_of_driver" and "first_point_of_impact" have vey small bar thus contributing the least to data variability hence we drop them.

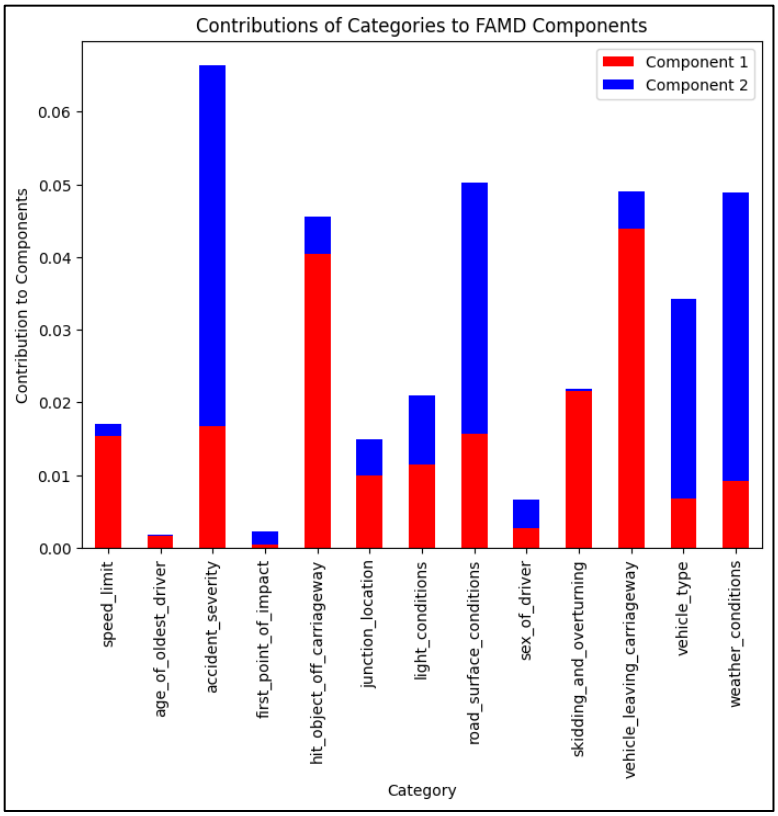

Figure 2: FAMD graph showing contribution to variation of each column in the data.

2. Data preprocessing

While performing EDA we got to know that the target variable has values that have been entered in different was and yet mean the same i.e., we have values that have the first letter that is an upper-case letter, so we change the to lower case by locating the values with the uppercase letter and replacing it with the same word in lower case, this is to maintain consistency and avoid model misinterpretation.

Next, we replace the value 'missing or out of range' with NaN in all the columns in the data set. We do this by setting up a list of the acceptable values and then using a lambda expression to set all the values that do not match the excepted values to NaN. The '.isna().sum()' will return the count of all the NaN values in the data set for each column. To handle the NaN value for the categorical data we replace it with the most frequent value within the column, this can be done by using the SimpleImputer() function from the imputer library, by doing so we avoid the tendency of biasing the analysis towards any specific value. We are effectively presuming that the missing values are most likely to be the same as the most frequent value in the dataset when we replace missing values with the most frequent value.

SimpleImputer() is also used for replacing the NaN in the numerical columns with the median of the column because it is less sensitive to outlier as compared to mean. Using the mean to fill in missing values from a numerical column that contains a few outliers or extreme values might distort the data's distribution and produce biased findings. On the other hand, the median offers a more reliable assessment of the data's central tendency since it is unaffected by outliers (Audigier et al.,2016).

The target variable 'accident_severity' contains 1172 NaN values so we drop the row from our data set to avoid

adding a bias to the data. In the following step we drop all the duplicate row we do this while still having the 'accident_index' in the data set since, it is the attribute that uniquely differentiates each data point in the data set. This is an important step in preprocessing to prevent the model from being trained on redundant data, which might have a detrimental effect on the model's performance. This could lead to model overfitting on the duplicated data, which means that it is discovering patterns unique to that sample rather than those that are prevalent throughout the data set (white et al., 2010).

Next, we create a two data frames X and Y. Where X contains the feature set and Y contains the target variable we want to predict. The columns 'accident_index', 'age_of_oldest_driver', 'first_point_of_impact' and 'sex_of_driver' were not considered in X as they are insignificant fields. Before performing any modeling, the data set must be split into training, validation, and testing data set. This is done by using the 'train_test_split' function from the 'model_selection' library which splits the data in an 80:10:10 ratio. Here, the class proportions in the target variable Y are preserved in both the training and testing sets by setting 'stratify' equal to Y. When dealing with classes that are unbalanced when one class has much less samples than the other, this is very crucial.

To implement a model, the data should be in numeric format. Hence, we use the one hot encoder from the 'preprocessing' package to encode the categorical values in the X and Y data frames. It converts each categorical value into a new column and adds zeros or ones if that value exists for the row of data. Additionally for traditional models we encode the Y using label encoding. As for numerical columns we perform standardization by using the StandardScaler() function. This is crucial because machine learning algorithms frequently presume that the characteristics have a comparable range of values and are scaled on the same scale. We can guarantee that the characteristics have a mean of 0 and a standard deviation of 1 by normalizing them. This guarantees that the algorithm evaluates all features equally and makes it simpler to assess the relative value of various features.

3. Classification using traditional machine learning

In this section different traditional ML models like SGD Classifier, Logistic Regression, Decision Tree, Random Forest, and K- Nearest Neighbors have been implemented by performing rigorous experimentation. After comparing the accuracy and performance Random Forest proves to be the best model.

The algorithm used builds a "pipe_rf" pipeline to create a random forest classifier. A RandomForestClassifier estimator with the hyper parameters "n_estimators" and "max_depth" is included in the pipeline. Where "max_depth" regulates the maximum depth of each tree and "n_estimators" determines the total number of trees in the forest. Table 1 provides an extensive description of this model's hyperparameters. In our case an "n_estimator" of 100 and a "max_depth" of 10 give us an accuracy of 78.2% on training data and a validation accuracy of 77.5%, while having a test accuracy of unseen data of 77.1%. The difference between the two is not very significant this implies that there is no possible overfitting in the model. which is the highest amongst all the other models.

Hyper parameters	Potential parameters	Best parameter
n_estimators	100,500	100
max_depth	None,5,10	10

Table 1: Hyper parameter of best model(Random Forest).

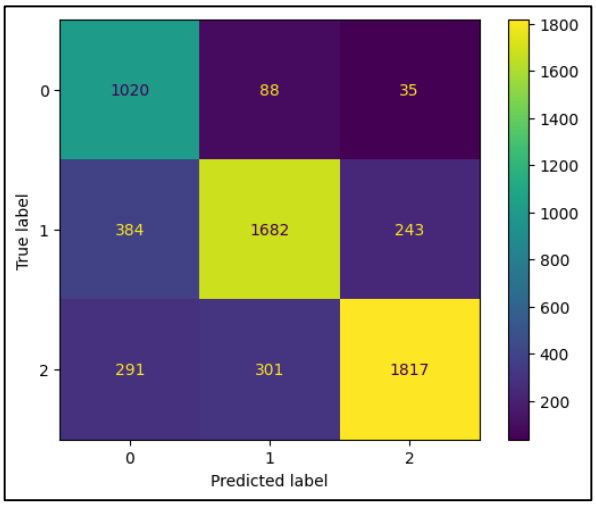

Figure 3: Confusion Matrix of the best model (Random Forest)

For a set of test data with known real values, the Random Forest model performs as shown in Figure 3's confusion matrix. Classes are used to categorise accidents according to their severity; class 0 corresponds to "fatal," class 1 to "severe," and class 2 to "slight" incidents. The model correctly classified 1020 out of 1143 occurrences as belonging to class 0, but incorrectly assigned 88 and 35 examples to classes 1 and 2, respectively. In class 1, 1682 out of 2309 occurrences were correctly recognised, whereas 384 and 243 examples, respectively, were incorrectly categorised as classes 0 and 2. In class 2, 1817 out of 2409 cases were correctly recognised, whereas 291 and 301, respectively, were incorrectly categorised as classes 0 and 1.

	Precision	Recall	F1-score	Support
0 (Fatal)	0.60	0.89	0.72	1143
1 (Serious)	0.81	0.73	0.77	2309
2 (Slight)	0.87	0.74	0.81	2409

Table 2: Classification report of best model (Random Forest)

From table 2 we know that the effectiveness of the Random Forest model on the severity classes 0, 1, and 2 was evaluated using precision and recall. Recall is the percentage of real positive cases accurately detected, whereas precision is the percentage of correctly categorized positive occurrences. For classes 0, 1, and 2, the corresponding precision values are 0.60, 0.81, and 0.86, while the corresponding recall values are 0.60, 0.81, and 0.86. With an accuracy of 0.60 and a recall of 0.88, the model performed well on classes 1 and 2 but badly on class 0, suggesting the categorization of examples from classes 1 and 2 as class 0.

Experimenting different hyper parameters

To implement various models in an efficient manner we make use of a pipeline along with grid search to perform an exhaustive search for all the possible hyper parameter for each model.

An algorithm is implemented in which four pipelines are defined for each model, each pipeline consists of the classifier for each model. The pipelines are defined first, and then each classifier's parameter grids are defined in the code. The different hyperparameters in these grids will be tuned using GridSearchCV.

Then, a dictionary of models is constructed, with a pipeline and hyperparameter grid for each model. The optimal hyperparameters, validation accuracy and test accuracy are store in the table 3 below. A 5-fold cross-validation is carried out, and the optimal hyperparameters are found using GridSearchCV. The dictionary of models has the best estimator.

Model Name	Best Parameters	Accuracy of test data	Precision	Recall	F1-Score
SGD Classifier	{'eta0': 0.005623413251903491, 'learning_rate': 'constant', 'max_iter': 20000}	76.1%	0- 0.60 1- 0.81 2- 0.85	0- 0.88 1- 0.70 2- 0.76	0- 0.71 1- 0.75 2- 0.80
Logistic regression	{'clf__C': 10.0, 'clf__penalty': 'l2'}	75.1%	0- 0.61 1- 0.80 2- 0.80	0- 0.78 1- 0.72 2- 0.78	0- 0.68 1- 0.75 2- 0.79
Decision Tree	{'clf__max_depth': 5, 'clf__min_samples_split': 5}	76.2%	0- 0.60 1- 0.82 2- 0.84	0- 0.89 1- 0.70 2- 0.76	0- 0.72 1- 0.78 2 – 0.80
Random Forest	{'clf__max_depth': 10, 'clf__n_estimators': 100}	77.1%	0- 0.60 1- 0.81 2- 0.87	0- 0.89 1- 0.73 2- 0.75	0- 0.72 1- 0.77 2- 0.81
KNN Classifer	{'clf__n_neighbors': 9}	74.8%	0- 0.60 1- 0.76 2- 0.83	0- 0.79 1- 0.72 2- 0.75	0- 0.68 1- 0.74 2- 0.79

Table 3: Comparison of different traditional models

Table 3 shows that four distinct machine learning models, including K-nearest neighbours (KNN), Random Forest, Decision Tree, and Logistic Regression, were trained and tested. With an accuracy score of 0.771 on the test set, the random forest model had the highest performance in terms of accuracy. The accuracy ratings for the other models varied from 0.75 to 0.76. For each class, the Random Forest model had the best accuracy, recall, and F1-score.

Finally, predicting the majority class for each occurrence is a simple baseline for a multi-class classification issue. In this instance, a majority class baseline would forecast class 2(slight) for all cases since class 2(slight) has the greatest number of occurrences. From table 2 the percentage of occurrences that fall under class

2(slight) can be used to calculate the accuracy of the majority class baseline. This baseline can be used as a standard against which to compare the effectiveness of the random forest model. Class 2(slight) is the dominant class here, accounting for 2409 of the 5861 occurrences in the test set. As a result, the accuracy of the baseline for the majority class is 2409/5861 = 0.411. The accuracy of the random forest model is 0.771, which is much better than the majority class baseline and can capture patterns in the data.

4. Classification using neural networks

In this section we have implemented neural network using the Keras library in TensorFlow for multi-class classification. The same preprocessed data i.e X containing the feature data frame and Y containing the target variable is used.

A 'Sequential' model is used to build a simple neural network with 2 hidden layers. The number of features in the input data is used to specify the input layer. 20 nodes in the hidden layer having activation functions. The output layer utilises the 'softmax' activation function and has three nodes, one for each of the three classes.

After that, the model is created using the 'CategoricalCrossentropy' loss function, which is appropriate for multi-class classification, and the 'Adam' optimizer with a learning rate of 0.01 is used. The accuracy score is used as an assessment metric.

The 'compute_class_weight' function from scikit-learn is used to calculate the class weights into account for data imbalance. The fit method is then used with the class weights as an input to give the minority classes extra training weight.

Using a batch size of 500 and 100 epochs, the 'fit' method is used to train the model. The model's performance during training is tracked using a 10% validation split. The 'fit' method provides a history object that details the training procedure and includes the loss and accuracy values for the training and validation sets at each epoch. An overall accuracy of 78 % is achieved by the neural network model.

Learning rate	Activation function	optimizer	epochs	No. of Hidden layer	No of hidden layer nodes	Epochs	Batch size	Accuracy
0.001	Softmax,relu	Adam	100	2	20	100	500	78%

Table 4: Best model (Model 2) hyper parameters, accuracy, and balance accuracy.

Table 4 contains all the hyper parameter used for training the above explained model. 20 hidden layer nodes. A learning rate of 0.001 is used, this establishes the step size at which the neural network's optimizer changes its weights during training. An Epoch of 100 and batch size of 500 is used it is the quantity of training samples the optimization method uses during each forward and backward pass. The first and third hidden layer uses 'softmax' and 'relu' as activation functions respectively while the second applies dropout with a rate of 0.2.

Evaluation

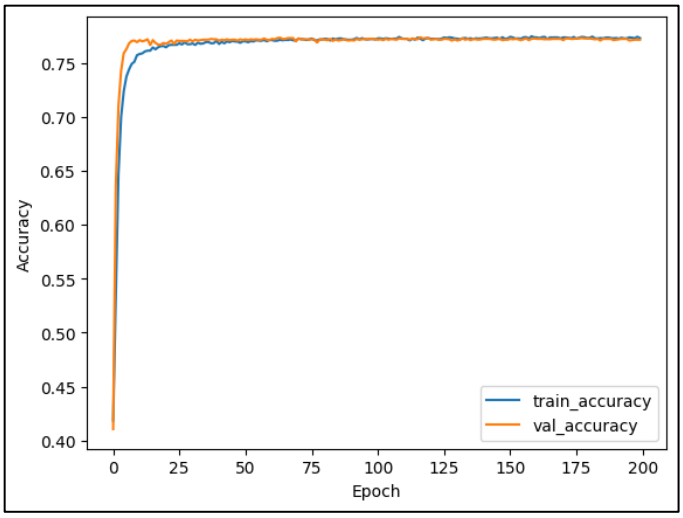

Figure 4: Epoch Vs Accuracy (to check for overfitting)

The figure 4 is a plot of accuracy for each epoch for train and validation data. we can see that validation line follows the training line without deviation and thus shows that the model is not overfitting the data.

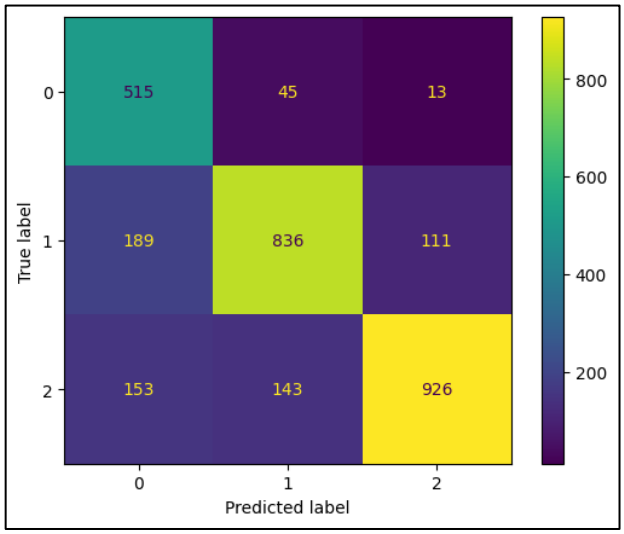

Figure 5: Confusion Matrix for model 2.

In figure 5 the confusion matrix contrasts the actual (real) classes with the classes that the neural network predicted. Classes "fatal," "severe," and "slight" are denoted by the labels 0, 1, and 2, respectively. Correct predictions for classes 0, 1, and 2 totaled 515, 836 and 927, respectively.

	Precision	recall	F1-score	Balance Accuracy
0	0.60	0.90	0.72	0.797
1	0.82	0.74	0.77	
2	0.88	0.76	0.82	

Table 5: Classification report of Model 2 (best model)

From the above table 5 given that there are three classes, the model's accuracy of 0.78 is a good result. The model is doing well across all three classes, according to the balanced accuracy score of 0.797. The model struggles a bit while predicting the class 0 but performs well for the other two.

Experimenting with different hyper parameters.

Two different Neural network models were experimented. The model 1 makes use of 1 hidden layer with 'relu' as the activation function, and 50 nodes and a batch size of 130 and resulted with an accuracy of 77%. While the Model 2 has two hidden layers with 'softmax' and 'relu' in the 1st and 3rd layer, while a dropout is applied to the 2nd layer to avoid overfitting the model. Both the models make use of 'Adam' as the optimizer. The hyper para meters and the results of the above implementations are shown in Table 6

	Learning rate	Activation function	optimizer	No. of Hidden layer	No of hidden layer nodes	Epochs	Batch size	Accuracy
Model 1	0.01	relu	Adam	1	50	50	130	77%
Model 2	0.001	Softmax,relu	Adam	2	20	100	500	78%

Table 6: Comparison of the Neural Network models

Additionally in model 2 different learning rates where experimented. Figure 5 shows that the best learning rate is 0.001 and has the following accuracy.

- Training accuracy = 77.33%
- Validation accuracy = 77.64%
- Test accuracy = 78.1 %

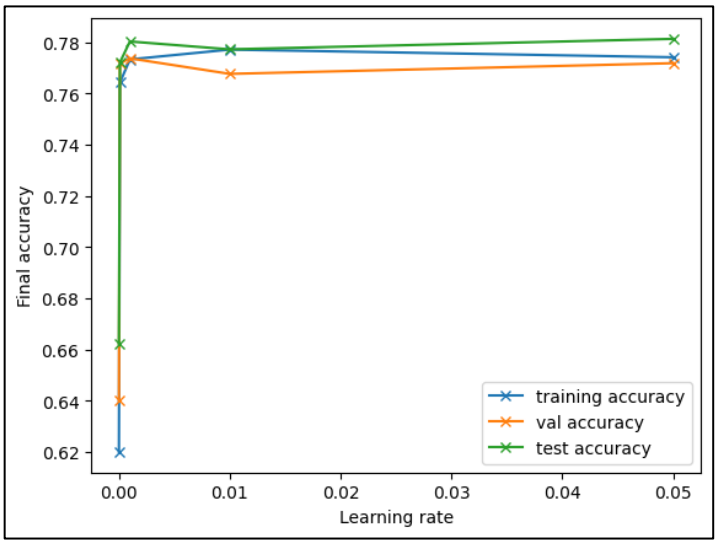

Figure 6: Final Accuracy Vs Learning rate.

5. Ethical discussion

I will make use of the Ethical OS Toolkit, which offers a framework for identifying and resolving ethical challenges in the development and deployment of AI systems, to identify and analyze the social and ethical implications of the task I have chosen.

Human Rights: The data gathered and processed for the accident severity prediction task may include sensitive data, such as identifying information and medical information, which may be a violation of people's right to privacy and data protection.

Bias: The task's dataset may contain biases that have an impact on the fairness and accuracy of the machine learning predictions. For instance, there could be biases in the reporting or recording of incidents, which could result in erroneous or insufficient data. Moreover, using historical data to train the model may reinforce biases from earlier choices or behaviors, resulting in unfair results for some groups.

Accountability: If the machine learning predictions are used to guide crucial judgements like allocating resources or responding to emergencies, it may be unclear who is in charge of ensuring their fairness and accuracy. Because to the transparent nature of the machine learning algorithms, it may be challenging to hold people or organizations' responsible for any unfavorable outcomes that result from the predictions.

Transparency: Those who are not professionals in the field of machine learning may find the methods utilized for the assignment to be obscure or challenging to comprehend. It can be challenging to find and correct any biases or mistakes in the data or model because of this lack of transparency, which might undermine confidence in the forecasts.

Effect on Society and the Environment: By precisely forecasting the severity of accidents and identifying high-risk zones, the machine learning algorithm can increase road safety. This can lower the number of deaths, manage resources more effectively, and stop accidents from causing environmental harm.

6. Recommendations

- The Model 2 of Neural Network is the best classification algorithm for this task since it has the highest accuracy in classifying the classes correctly without overfitting.
- The Neural Network model is successful in predicting the classes correctly based on the pattern it identified in our data. Hence it can be used to be put in to practice
- In the future the model could be experimented with more hidden layer as this requires higher processing power and hence couldn't be implemented. Additionally, more complete and accurate data could be for the task.

7. Retrospective

Since we deal with real world data. The dataset is bound to have fields that have either missing value or values that behave like anomalies. An area of interest to experiment would be to implement various method for handling these types of values and then comparing the effect of this method on the efficiency of the model.

8. References

Audigier, V., Husson, F., & Josse, J. (2016). A principal component method to impute missing values for mixed data. *Advances in Data Analysis and Classification*, *10*(1), 5–26. https://doi.org/10.1007/s11634-014-0195-1

White, I. R., Daniel, R., & Royston, P. (2010). Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical variables. *Computational Statistics & Data Analysis*, *54*(10), 2267–2275. https://doi.org/10.1016/j.csda.2010.04.005