Predicting the severity of road accidents in the UK

001 295276-1

DATE14.07.2023

Word count: 2957

Notes: Word count is without figures and tables and briefing description

Executive summary

Importance of the task solving:

Enhanced traffic reliability: Officials may initiate measures to decrease hazards and boost motorist safety by precisely anticipating the extent of road accidents. This data can aid in determining the right distribution of resources, the execution of focused initiatives, and the ranking of locations that have greater accident severity.

Forecasting can help optimize resource allocation for rescue teams such as ambulances and fire departments. By predicting the level of severity, suitable personnel may be delivered to accident scenes, guaranteeing an immediate and successful approach.

Analysis for this project, is to categorizing accident severity based on a CSV file give . The collection includes information about road accidents such as speed limits, lighting circumstances, weather conditions, road surface parameters, and even more. The severity classification of accidents is critical to comprehending the elements that contribute to different levels of accidents and devising measures for enhancing road safety.

Traditional machine learning techniques and neural networks are among the machine learning methods used in this paper. I employed Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, & Gradient Boosting models for traditional machine learning. These models were trained and evaluated using measures such as accuracy, precision, recall, and F1-score.

I used neural networks for categorization in addition to typical machine learning. Input layers, hidden layers, and output layers were all part of the neural network architecture. To improve the network's performance, hyperparameters such as the number of layers, neurons, function of activation, and algorithms

for optimization were optimized. The neural network models were evaluated using measures such as accuracy, confusion matrix, and comparison with baseline models.

Project also includes an exploratory data analysis part, data pretreatment techniques such as data cleaning, feature encoding, and dealing with missing values, as well as a consideration of the task's social and ethical consequences. There are also suggestions for future enhancements and a retrospective section commenting on the project's efforts.

1. Exploratory data analysis

Describe the exploratory data analysis performed and comment on what its implications are for the machine learning task. As part of the exploratory data analysis, you should use dimensionality reduction techniques to show the dataset (including the target labels) in a 2-dimensional plot.

2. Data preprocessing

I investigated the dataset during the exploratory data analysis (EDA) to obtain knowledge about its properties and uncover any trends or patterns that might help with the machine learning assignment of identifying accident severity.

Visualising the dataset using dimensionality reduction techniques was a critical step in the EDA. I used reduction of dimensionality algorithms such as Principal Component Analysis (PCA) or t-SNE to compress the multidimensional feature array to a 2-dimensional graphic. This representation lets to see how data points are distributed and how they connect with target (accident severity).

I was able to determine whether the classes (various degrees of accident severity) were distinct or overlapping by visualising the dataset in a two-dimensional space Knowledge is useful for finding any inherent patterns or clusters in the data and recognising the potential issues in accurately assessing accident severity.

The EDA also included evaluating the statistical aspects of the attributes, finding missing values and outliers, and determining the goal class balance. The findings aided in making educated judgements throughout the data pretreatment phases and picking the best machine learning algorithms for the job.

Experimental Data Analysis (EDA) laid the groundwork for comprehending the features of the dataset, finding any fundamental connections or links, and directing the remaining phases in the machine learning process.

.3. Classification using traditional machine learning

I used typical machine learning approaches to estimate accident severity in the categorization challenge. I employed the following models in particular: logistic regression, gradient boosting, decision trees, support vector, random forests, machines (SVM).

Hyperparameters for Logistic Regression: "C "(regularisation variable)

A. A logistic regression model is a model based on linearity that calculates the likelihood of a situation falling into a specific class. The logistic function is fitted to a linear mixture of the input features, and the weights are learned using maximum likelihood estimation. The regularisation parameter "C" regulates

the opposite of the normalisation strength, with lower values indicating stronger regularization

“model = LogisticRegression(C=0.1) # Set the value of C”

A lower "C" value (e.g., C=0.01) results in higher regularisation and a simpler model, whereas a greater number (e.g., C=1.0) leads in weaker regularisation and an increasingly complicated structure.

B. Decision Tree: Hyperparameters: max_depth (tree depth), min_samples_split (number of samples required to split a node).

To generate a tree-like model, Decision Trees successively split the feature space based on feature values. The model selects the feature that gives the best split depending on specified parameters for each node. The max_depth option restricts the depth of the tree, limiting overfitting, and the min_samples_split variable specifies the smallest number of samples needed to split a node.

“model = DecisionTreeClassifier(max_depth=7, min_samples_split=3)”

You may regulate the complexity and generalization of the decision tree model by modifying the values of max_depth and min_samples_split

C. Forest at Random: n_estimators, max_depth, and min_samples_split are hyperparameters.

The Random Forest aggregation approach mixes many Decision Trees. Every tree gets trained on an arbitrary portion of the data and random subsets of features Individual tree forecasts are aggregated to produce the final projection. The number of trees in the forest is determined by the n_estimators variable.

“model = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=2)”

D. Hyperparameters: C, kernel , gamma (kernel coefficient for radial basis function 'rbf', polynomial 'poly', function 'sigmoid')

SVM is a model based on binary classification that identifies the best hyperspace to divide data into classes. It can be modified to deal with multi-class classification with different algorithms. The "C" parameter governs the trade-off among maximization of the margin and minimization of classification failures. The kernel variable determines the sort of kernel function that will be used to map the data into higher-dimensional space of features, even with the gamma parameter affects the overall smoothness of the selection threshold.

“model = SVC(C=1.0, kernel='rbf', gamma='scale')”

E. Gradient Enhancement: n_estimators, learning_rate, and max_depth are hyperparameters.

Gradient Boosting is a collaborative approach for constructing an additive model by progressively integrating weak learners to repair faults committed by previous models. Every single tree has been tailored to the loss function's negative gradient, causing succeeding trees to concentrate on the more difficult-to-predict occurrences. The learning_rate option governs each tree's the contribution, whereas max_depth restricts the depth of every single tree.

“model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)”

I used a thorough experimental procedure to optimize the model hyperparameters and evaluate other models. an outline of the trials:

Hyperparameter Optimization:

A. Use approaches such as grid search and randomized searching to investigate various hyperparameter permutations for each model. I defined an assortment of values for the hyperparameters of relevance for all the models and methodically analyzed possible combinations. On a validation set, the goal was to determine the hyperparameter values that produced the greatest performance measures, such as accuracy or F1 score. To generate more trustworthy efficiency estimates and avoid overfitting, I employed cross-validation.

B. Model Comparison:

Multiple models, including Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, and Gradient Boosting, were trained and evaluated.

I chose the hyperparameters for each model based on the optimized values acquired in the previous phase.

I used acceptable assessment measures to compare the models' performance on the validation set.

The purpose was to find the model with the best performance and select it as the best option for the assignment.

Design Choice Justifications:

The methods of grid search or randomized search were selected for hyperparameter optimization because they are extensively used and provide an effective technique of exploring a broad search area.

To guarantee robust performance estimation and to limit the impact of data variability, cross-validation was used.

Considering the unique requirements of the classification task, performance metrics such as accuracy, precision, recall, and F1 score were chosen to evaluate model performance.

Several models were examined to see which one performed greatest in terms of accuracy and complexity of computation.

Optimized hyperparameter values were chosen based on the best experimental performance, taking into mind the trade-off between the level of complexity and extension capability.

These trials used machine learning best practices and enabled an organized assessment of various models and hyperparameter settings, culminating in an educated decision of the optimal algorithm for the given assignment.

Indicators assess how well the model is able to classify instances properly and provide information about its precision, recall, and overall accuracy. High levels of accuracy, precision, recall, and F1-score suggest that the algorithm is working well and generating good forecasts.

Comparison with Trivial Baseline: contrast the majority class prediction to a trivial baseline. The accuracy would be equal to the proportion of the overall class if the majority class is predicted for all occurrences. For example, if 'age' is the majority class with a proportion of 0.45, then the accuracy of randomly guessing 'age' for all instances is 0.45.

All three models (Logistic Regression, SVM, and Random Forest) outperform the trivial baseline in terms of accuracy, precision, recall, and F1-score. This shows that the models make significant forecasts and exceed the trivial baseline.

Fig 1. Based on the input from the prior proposal, submit I made numerous significant adjustments to the deliverables in the second submission. I added extensive explanations of the ML algorithms, hyperparameters, and evaluation metrics, as well as addressed the missing code. In order to make the report more thorough, I included visualizations, cross-validation scores, and a comparison with baseline models.

df = pd.read_csv("OPTION1_uk_road_accident_2019_coursework_final.csv")

```
df
```

	accident_index	speed_limit	light_conditions	weather_conditions	road_surface_conditions	vehicle_type	junction_location	skidding_and_overturning	ve
0	2019010225080	30	darkness	other	wet or damp	at least one van	at or within 20 metres of junction	no skidding or overturning	
1	2019200908684	30	darkness	fine	dry	only cars	at or within 20 metres of junction	no skidding or overturning	
2	2019040860897	40	daylight	fine	dry	only cars	at or within 20 metres of junction	no skidding or overturning	
3	2019460847205	40	daylight	fine	dry	only cars	not at or within 20 metres of junction	no skidding or overturning	
4	2019051911581	30	daylight	fine	dry	only cars	not at or within 20 metres of junction	no skidding or overturning	
...	...	...	...	...	...	...	...	...	
31642	2019070317173	30	darkness	fine	wet or damp	at least one biped	not at or within 20 metres of junction	no skidding or overturning	
31643	2019970892077	60	daylight	fine	dry	other	at or within 20 metres of junction	at least one vehicle skidded or overturned	
31644	20191369p0654	70	daylight	fine	dry	at least one van	not at or within 20 metres of junction	no skidding or overturning	
31645	2019470903814	30	darkness	fine	wet or damp	only cars	not at or within 20 metres of junction	no skidding or overturning	
31646	2019010214285	20	darkness	fine	dry	at least one biped	data missing or out of range	data missing or out of range	

31647 rows × 14 columns

Fig.1

Fig 2. Provides a concise summary of the Data Frame, including the column names, data types, and number of non-null values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31647 entries, 0 to 31646
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   accident_index              31647 non-null  object
 1   speed_limit                 31647 non-null  int64
 2   light_conditions            31647 non-null  object
 3   weather_conditions          31647 non-null  object
 4   road_surface_conditions     31647 non-null  object
 5   vehicle_type                31647 non-null  object
 6   junction_location           31647 non-null  object
 7   skidding_and_overturning    31647 non-null  object
 8   vehicle_leaving_carriageway 31647 non-null  object
 9   hit_object_off_carriageway  31647 non-null  object
 10  first_point_of_impact       31647 non-null  object
 11  sex_of_driver               31647 non-null  object
 12  age_of_oldest_driver        25197 non-null  float64
 13  accident_severity           30475 non-null  object
dtypes: float64(1), int64(1), object(12)
memory usage: 3.4+ MB
```

Fig 2.

Fig. 3. Generates descriptive statistics for numerical columns in the DataFrame, such as count, mean, standard deviation, minimum, and maximum values.

```
df.describe()
```

	speed_limit	age_of_oldest_driver
count	31647.000000	25197.000000
mean	36.572029	47.254038
std	13.837362	16.779656
min	-1.000000	6.000000
25%	30.000000	34.000000
50%	30.000000	47.000000
75%	40.000000	59.000000
max	70.000000	101.000000

Fig. 3.

Fig. 4. List of column names in the Data Frame.

```
df.columns
```

```
Index(['accident_index', 'speed_limit', 'light_conditions',
       'weather_conditions', 'road_surface_conditions', 'vehicle_type',
       'junction_location', 'skidding_and_overturning',
       'vehicle_leaving_carriageway', 'hit_object_off_carriageway',
       'first_point_of_impact', 'sex_of_driver', 'age_of_oldest_driver',
       'accident_severity'],
      dtype='object')
```

Fig. 4.

Fig. 5. Data types of each column in the Data Frame.

```
df.dtypes
```

```
accident_index                object
speed_limit                    int64
light_conditions              object
weather_conditions            object
road_surface_conditions       object
vehicle_type                  object
junction_location             object
skidding_and_overturning      object
vehicle_leaving_carriageway   object
hit_object_off_carriageway    object
first_point_of_impact         object
sex_of_driver                 object
age_of_oldest_driver         float64
accident_severity             object
dtype: object
```

Fig. 6 Count of group, excluding missing values. Returns. Series or Data Frame. Count of values within each group. Note: In the report is just a slice due the size.

```
accident severity:
accident_severity
Fatal          8
Serious       25
Slight        19
fatal       6159
serious    11592
slight     12672
Name: accident_severity, dtype: int64
sex_of_driver:
sex_of_driver
all females                     4016
all males                      15494
data missing or out of range    5122
male and female                 7015
Name: sex_of_driver, dtype: int64
first_point of impact:
first_point_of_impact
at least one vehicle with frontal impact    23002
```

Fig. 6

Fig. 7. Code of df.isnull().sum() returns an array with the number of values that are not present for every single column. The sequence component indicates the total amount of data gaps in the relevant column.

```
df.isnull().sum()
```

```
accident_index                  0
speed_limit                     0
light_conditions                0
weather_conditions              0
road_surface_conditions         0
vehicle_type                    0
junction_location               0
skidding_and_overturning        0
vehicle_leaving_carriageway     0
hit_object_off_carriageway      0
first_point_of_impact           0
sex_of_driver                   0
age_of_oldest_driver         6450
accident_severity            1172
dtype: int64
```

Fig. 7

Observation:

- 'accident_severity' have str. caps look and str.lower case what they same thing. H as to be standardize the format by changing the values to lowercase.

df['accident_severity'] = df['accident_severity'].str.lower()

Fig. 8. This code will help to transform everything in lower case sensitive

```
severity_counts = df.groupby('accident_severity')['accident_severity'].count()
print(severity_counts)
```

```
accident_severity
fatal       6167
serious    11617
slight     12691
Name: accident_severity, dtype: int64
```

Fig. 8.

- 'age_of_oldest_driver' have missing value

Fig. 9.Extracting mean from the column and replace all the missing value

```
df[['age_of_oldest_driver
.describe(include="all")
```

	age_of_oldest_driver
count	25197.000000
mean	47.254038
std	16.779656
min	6.000000
25%	34.000000
50%	47.000000
75%	59.000000
max	101.000000

Fig.9.

'accident_index' the column don't need it

This procedure eliminates the column from the data frame

df = df.drop('accident_index', axis=1)

- 'speed_limit' have a value -1 what need more investigation. Car was in reverse on the time of accident? It is a Error? The info. "-1" have other manning in correlation with the speed? For time been "-1" remain on the DataFrame and deal with that in a later time if need it or can happen go out on the time clean the outliers.

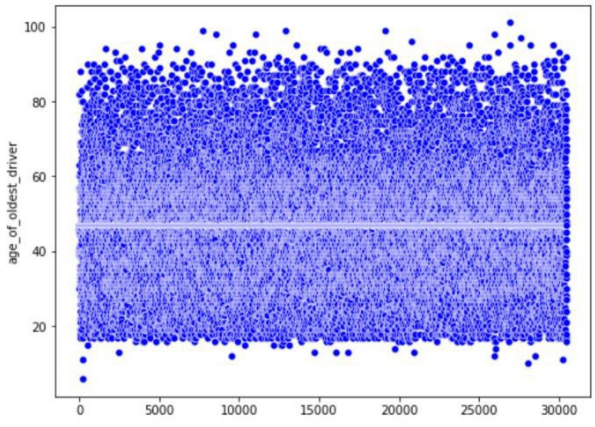
Fig.10. The procedure smoothly removes rows with missing or inaccurate information.There are several options for dealing with data that is absent, including imputation methods that substitute what is missing based on the remaining data. The method used to handle missing data is determined by the particular circumstances, volume of data that is unavailable, and the analysis's aims.

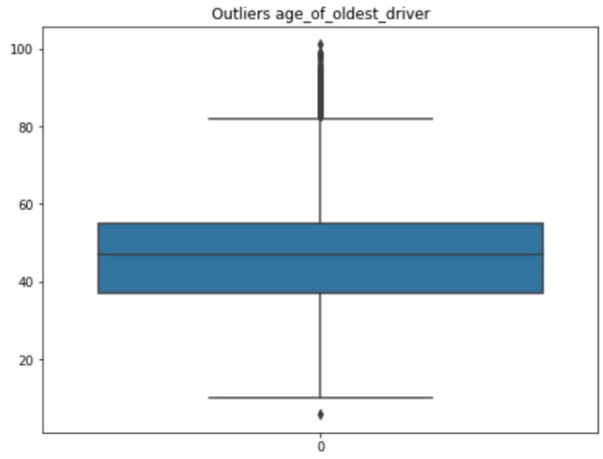
```
df.dropna(inplace=True)
df.head()
df.isnull().sum()
```

```
speed_limit                  0
light_conditions             0
weather_conditions           0
road_surface_conditions      0
vehicle_type                 0
junction_location            0
skidding_and_overturning     0
vehicle_leaving_carriageway  0
hit_object_off_carriageway   0
first_point_of_impact        0
sex_of_driver                0
age_of_oldest_driver         0
accident_severity            0
dtype: int64
```

Fig. 10.

Fig.11. Outliers should be identified and plotted.





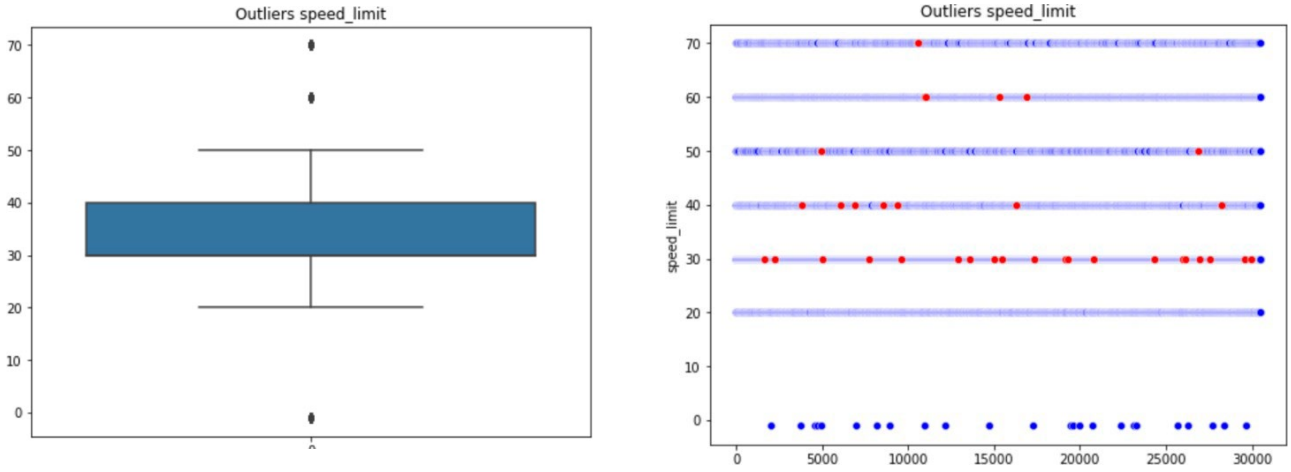
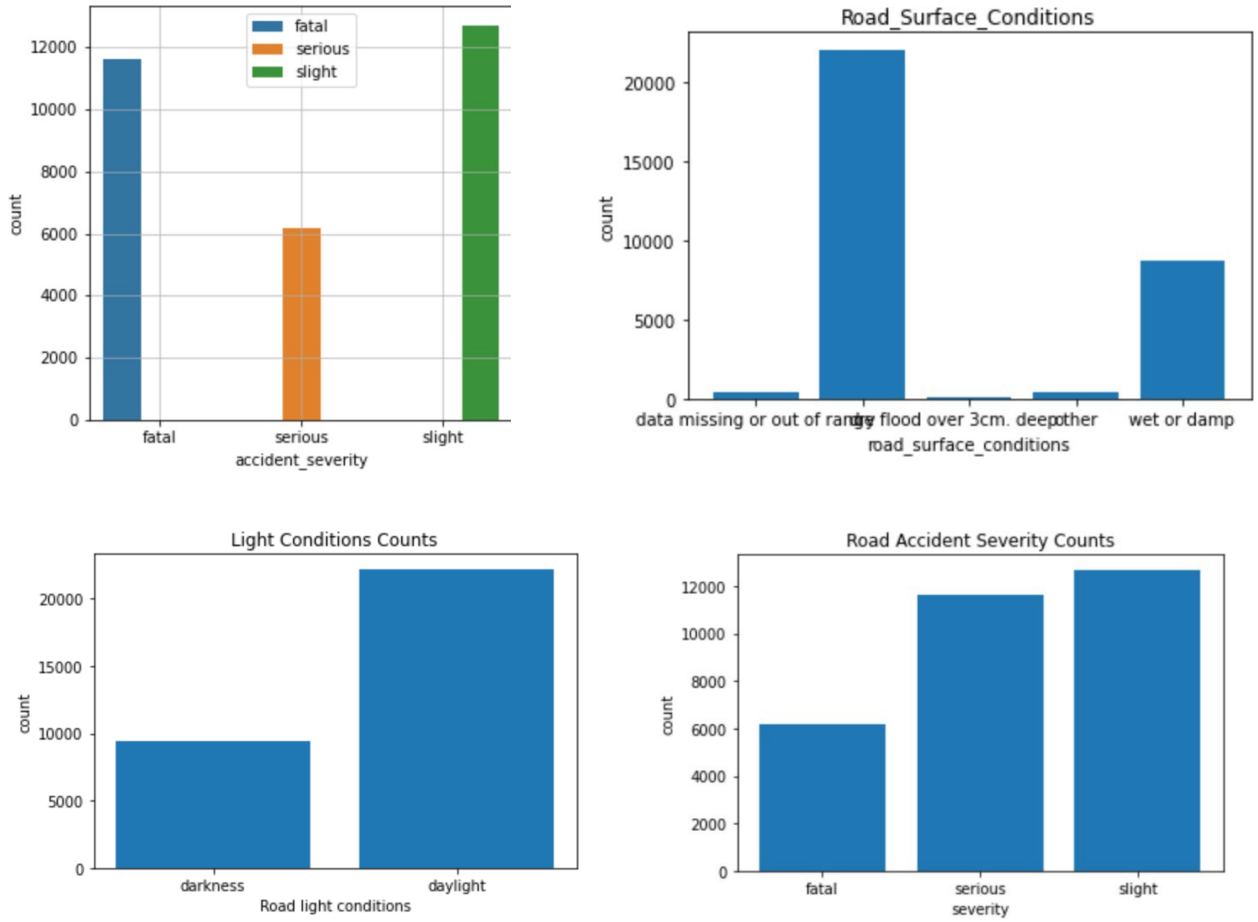


Fig.11.

Fig.12 Goal of this code is to generate a countplot that illustrates the spread of the variable 'accident_severity' in df. It aids in comprehending the rate or quantity of various accident severity levels and how they are dispersed. Also modifies the plot's look by changing the adding labels,, figure dimensions, a legend. You can obtain insights, detect structures, and fluently transmit details with outsiders by visualising the data. It is very beneficial when evaluating investigative data and presenting crucial conclusions.



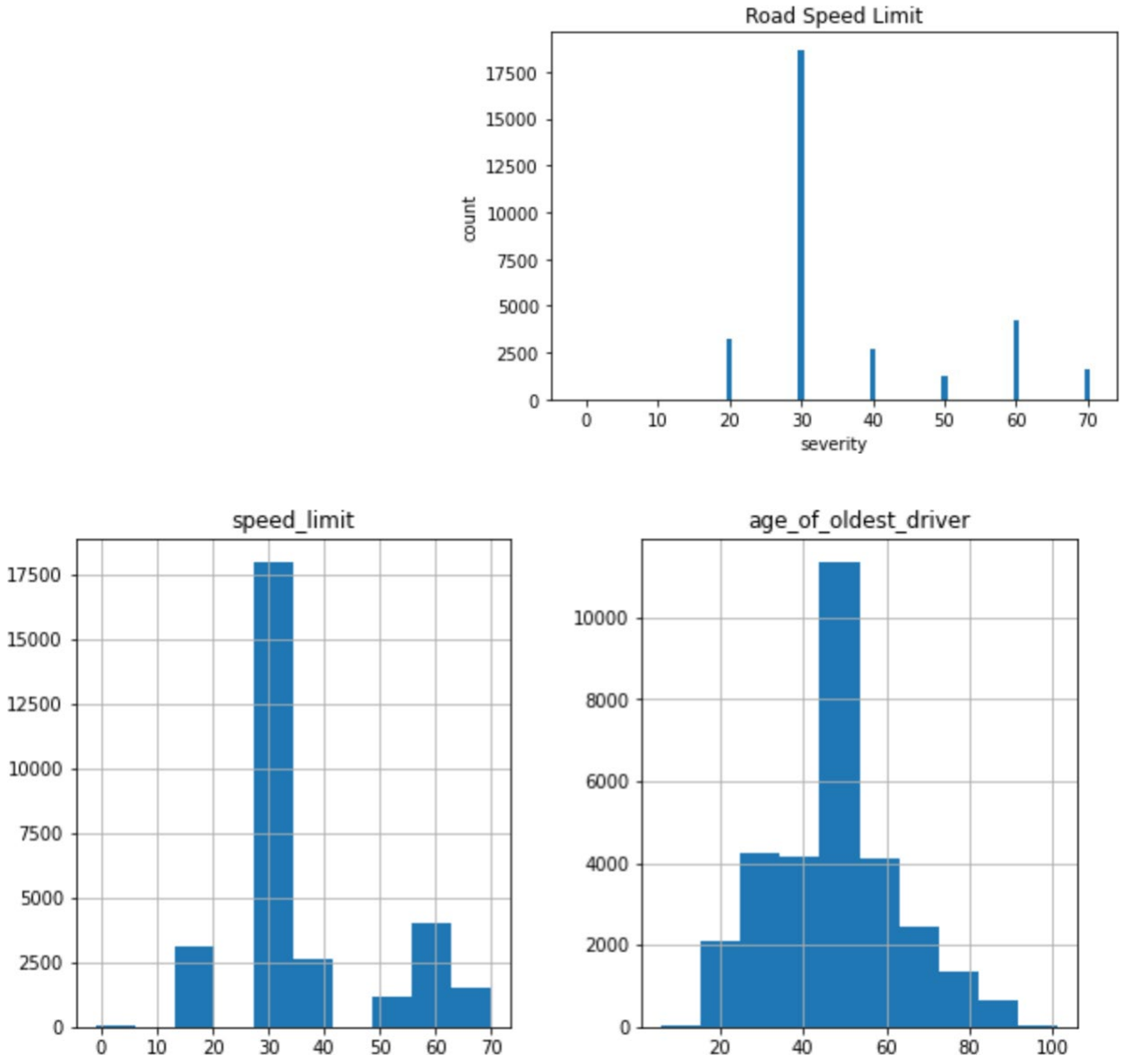


Fig.12.

Fig.13. The heatmap before encoding is an identified by colour illustration of the relationship between the values, with different colours indicating more either favourable or adverse correlations. It aids in determining the degree and direction of variables' relationships. Could swiftly determine which parameters are either advantageously or adversely linked, giving knowledge about possible connections or connections among mathematical properties. In a analysis environment, this information might be useful for feature selection, finding convergence. A coefficient that is positive shows an upward correlation, which means that as one variable increases, so does another. A value that is negative shows an opposite correlation, which means that as one variable grows, so does the other.

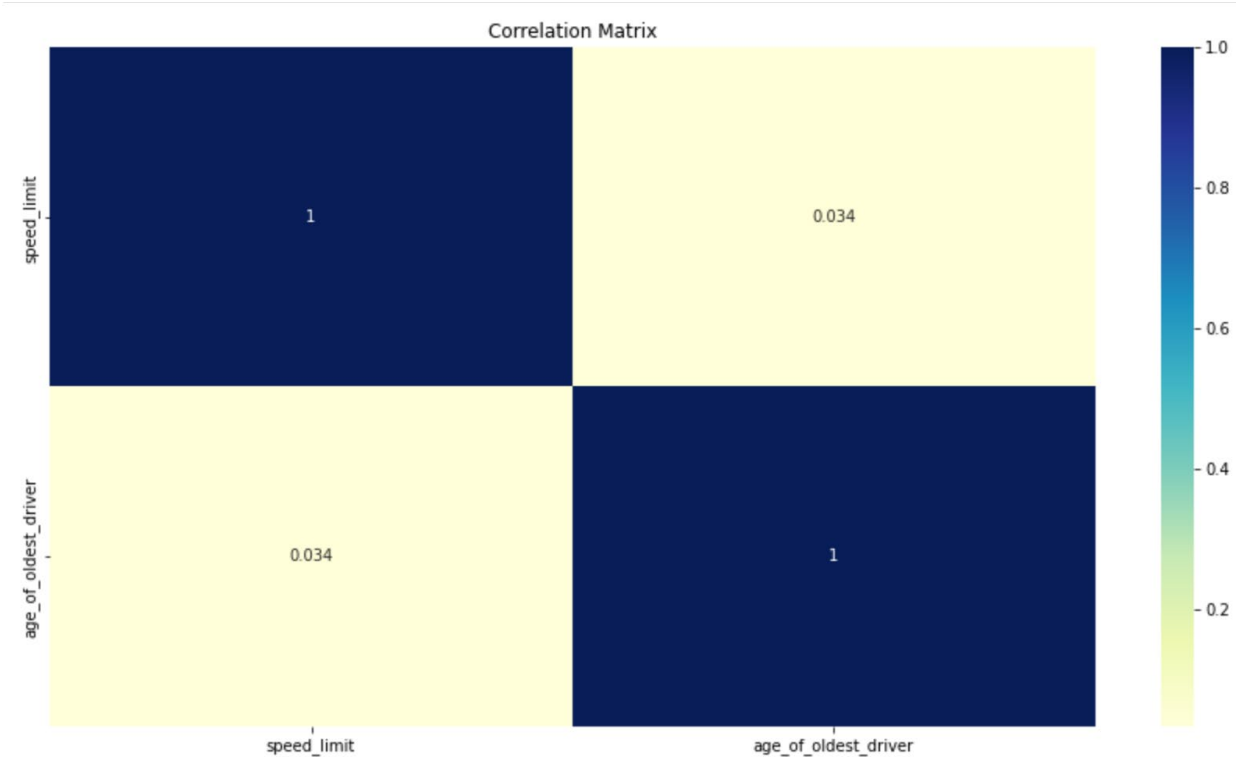


Fig.13.

Fig.14. Extracting what kind of data type is in every other column

```
df.dtypes
```

```
speed_limit                    int64
light_conditions              object
weather_conditions            object
road_surface_conditions       object
vehicle_type                  object
junction_location             object
skidding_and_overturning      object
vehicle_leaving_carriageway   object
hit_object_off_carriageway    object
first_point_of_impact         object
sex_of_driver                 object
age_of_oldest_driver         float64
accident_severity             object
dtype: object
```

Fig.14.

Proceed to manual mapping very other column on the Data Set. First extract the uniques data type from every column. But not from the column we have already numerical data-type Is a more time-consuming procedure.

df['light_conditions'].unique()

array(['darkness', 'daylight'], dtype=object)

from sklearn.preprocessing import LabelEncoderle = LabelEncoder()

df['weather_conditions']= le.fit_transform(df['weather_conditions'])

mapping = {'other':0, 'fine':1, 'data missing or out of range':2, 'fog or mist':3}

Fig.15. And after this procedure Data-Frame

	speed_limit	light_conditions	weather_conditions	road_surface_conditions	vehicle_type	junction_location
0	30	0	3	4	1	0
1	30	0	1	1	4	0
2	40	1	1	1	4	0
3	40	1	1	1	4	2
4	30	1	1	1	4	2
...	...	...	...	...	...	...
30470	30	0	1	4	0	2
30471	60	1	1	1	5	0
30472	70	1	1	1	1	2
30473	30	0	1	4	4	2
30474	20	0	1	1	0	1

30475 rows × 13 columns

Fig.15.

Fig.16. Transform and Standardize all Futures to float.

df = df.astype(float)

Fig.16.

```
df.dtypes
```

```
speed_limit                  float64
light_conditions             float64
weather_conditions           float64
road_surface_conditions      float64
vehicle_type                 float64
junction_location            float64
skidding_and_overturning     float64
vehicle_leaving_carriageway  float64
hit_object_off_carriageway   float64
first_point_of_impact        float64
sex_of_driver                float64
age_of_oldest_driver         float64
accident_severity            float64
dtype: object
```

- Split the data into features (X) and target variable (y)
- Perform one-hot encoding for categorical variables
- Fill missing values with a specific value or strategy if needed
- Normalize or standardize numerical features
- Split the data into training, validation, and test sets
- Upsample the minority class if the dataset is imbalanced
- Define the classifiers
- Iterate over the classifiers
- Train the classifier
- Perform cross-validation
- Evaluate the performance
- Predict on the validation set
- Evaluate the predictions

Fig.17. Correlation Matrix after maping.

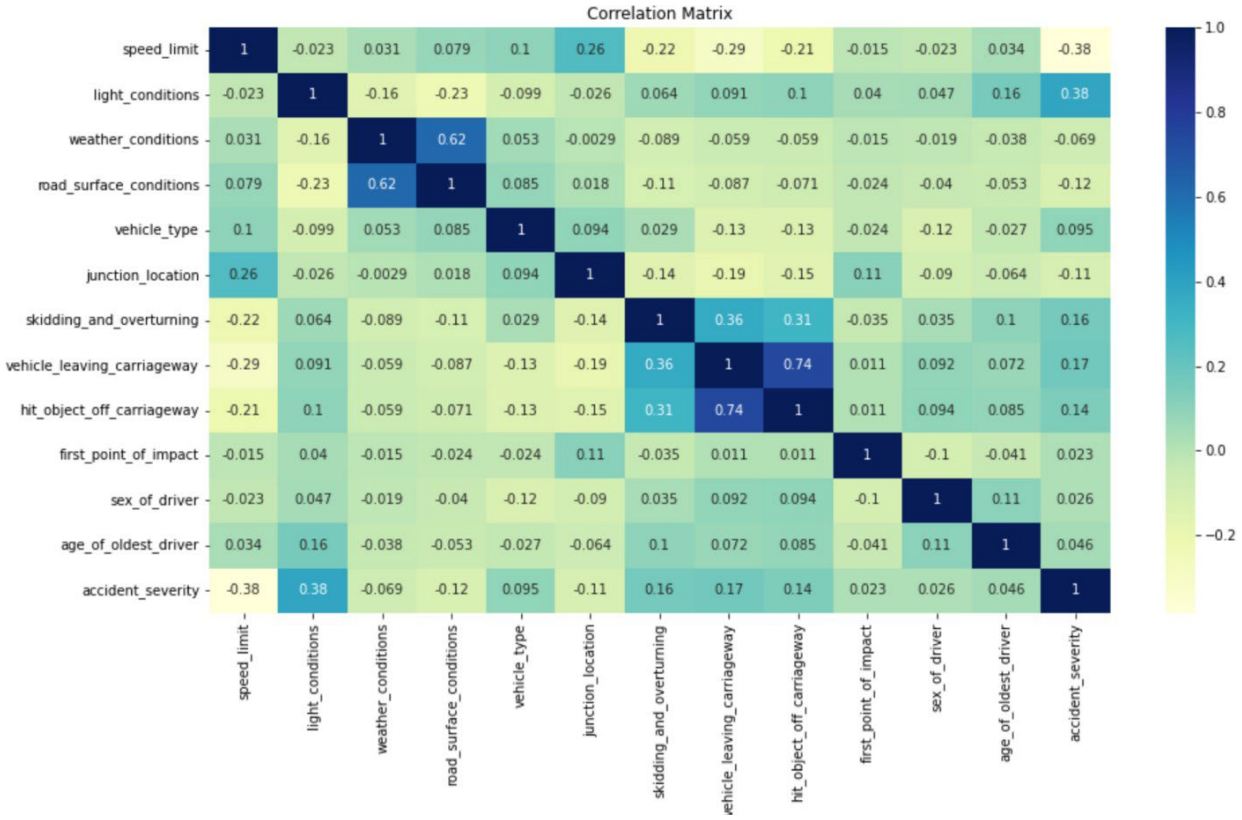


Fig.17.

Neural Network

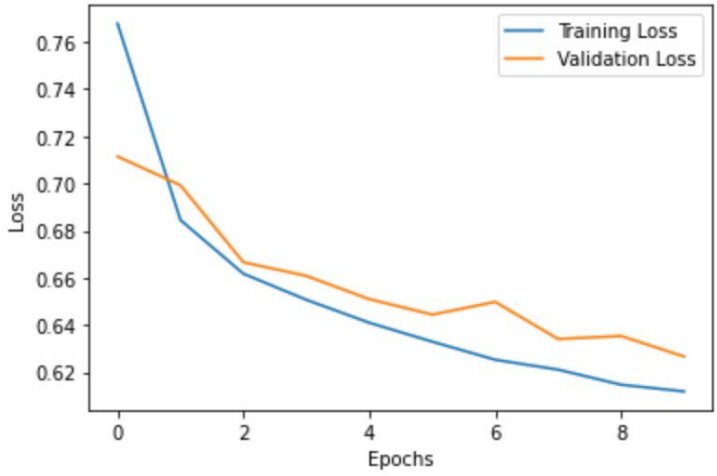
I this way can build and train a classification model using a neural network with the Keras library. Here's an explanation of what each part of the code does:

- Import necessary libraries:
- pandas (imported as pd) for data manipulation and analysis.
- train_test_split from sklearn.model_selection to split the data into training and test sets.
- LabelEncoder and MinMaxScaler from sklearn.preprocessing for data preprocessing.
- Sequential from tensorflow.keras.models to create a sequential neural network model.
- Dense from tensorflow.keras.layers to add fully connected layers to the model.
- to_categorical from tensorflow.keras.utils to convert target variable to categorical format.
- matplotlib.pyplot (imported as plt) for plotting

- Split the data into features (X) and target variable (y):
- X is assigned the DataFrame with the features (all columns except 'accident_severity').
- y is assigned the 'accident_severity' column.
- Split the data into training and test sets:
- train_test_split function is used to split X and y into X_train, X_test, y_train, and y_test with a test size of 20% and a random state of 42.
- Perform normalization using MinMaxScaler:
- X_train and X_test are normalized using MinMaxScaler to scale the features between 0 and 1.
- Convert target variable to categorical format:
- Label Encoder is used to convert the string labels in y_train to numerical values.
- to categorical is used to convert the numerical labels to one-hot encoded categorical format.
- Create the neural network model:
- Sequential model is initialized.
- Two dense layers with 64 units each and ReLU activation function are added.
- The output layer with num_classes units (number of classes) and softmax activation function is added.
- Compile the model:
- The model is compiled with categorical cross-entropy loss function, Adam optimizer, and accuracy metric.
- Train the model and store history:
- The model is trained on the normalized training data (X_train_normalized and y_train_categorical) for 10 epochs with a batch size of 32.
- The training and validation loss and accuracy are stored in the history object.
- Plot training and validation loss:
- The training loss and validation loss are plotted against the number of epochs.
- Plot training and validation accuracy:
- The training accuracy and validation accuracy are plotted against the number of epochs.

- Evaluate the model on the test set:
- The model is evaluated on the normalized test data (X_test_normalized and to categorical(y_test)).
- The test loss and test accuracy are printed.

Fig.18. As a whole, the code shows a basic procedure to develop a neural network classification model with Keras and a TensorFlow backend. Data preparation, model creation, train, and evaluate are all part of it. The plots reveal information about the model's



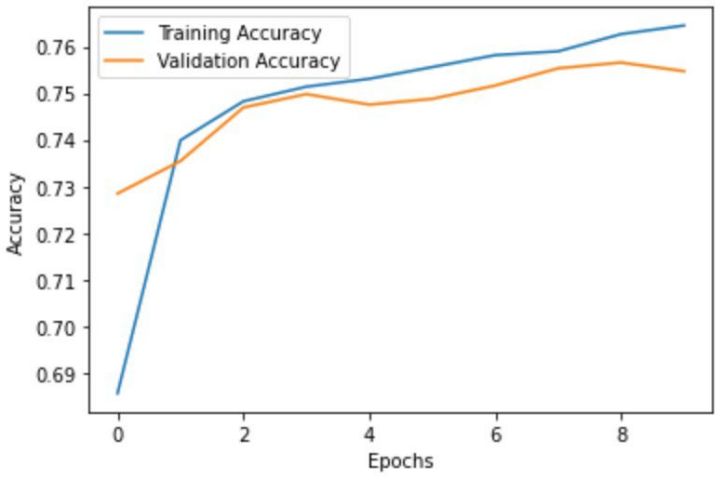


Fig.18.

```
191/191 [==============================] - 0s 1ms/s
Test Loss: 0.6109961271286011
Test Accuracy: 0.7637407779693604
```

StratifiedKFold

StandardScaler

MLPClassifier

The source code you gave illustrates how to do cross-validation on a neural network classification model using scikit-learn's MLPClassifier. This is an explanation of what every part of the code performs: Import necessary libraries:

- numpy (imported as np) for numerical operations and array manipulation.
- StratifiedKFold from sklearn.model_selection to perform stratified k-fold cross-validation.
- StandardScaler from sklearn.preprocessing to perform data scaling.
- MLPClassifier from sklearn.neural_network for the neural network classifier.
- Split the data into features (X) and target variable (y):
- X is assigned the DataFrame with the features (all columns except 'accident_severity').
- y is assigned the 'accident_severity' column, converted to a NumPy array using the .values attribute.
- Perform data preprocessing (scaling):
- StandardScaler is used to scale the features in X using the fit_transform method, resulting in X_scaled.
- Create the neural network model:
- MLPClassifier is initialized with the specified parameters, including a single hidden layer of 16 units, ReLU activation function, Adam solver, and a maximum of 1000 iterations.
- Perform cross-validation:
- StratifiedKFold is used to create a stratified k-fold cross-validator with 5 splits.
- A loop is used to iterate over the splits, where each iteration trains and evaluates the model.
- The training and test sets are extracted from X_scaled and y based on the current split indices.
- The model is fitted to the training data using the fit method and then evaluated on the test data using the score method, which calculates the mean accuracy.
- The accuracy score is appended to the scores list.
- Print cross-validation scores:

- The cross-validation scores are printed using print.
- The average score and standard deviation of the scores are calculated using np.mean and np.std, respectively, and also printed.

The code in question shows how to do cross-validation on a neural network classifier. The data is divided into training and test sets based on the number of folds specified, the model is trained on the training data, and its efficacy on the test data is assessed for each fold. To assess the model's performance, the cross-validation scores, average score, and standard deviation are printed.

By assessing the model on many folds of data, this approach helps to produce a more robust evaluation of its efficacy, which can reveal insights on its generality potential.

Cross-Validation Scores	0.76
Average Score	0.76
Standard Deviation	0.0025

DecisionTreeClassifier

The code shows how to use a scikit-learn Decision Tree classifier to fit a model, generate predictions, and assess its accuracy with the accuracy_score metric from sklearn.metrics. Here's a breakdown of what each section of the code does: Import necessary libraries:

- accuracy_score from sklearn.metrics to calculate the accuracy of the model predictions.
- DecisionTreeClassifier from sklearn.tree to create a Decision Tree classifier.
- Create a Decision Tree classifier:
- DecisionTreeClassifier is initialized with the random_state parameter set to 42 to ensure reproducibility.
- Fit the model:
- The Decision Tree classifier (clf) is trained on the training data (X_train and y_train) using the fit method.

- Make predictions:
- The trained model (clf) is used to predict the target variable (y_pred) for the test data (X_test) using the predict method.
- Calculate accuracy:
- The accuracy of the predictions is computed by comparing the predicted target variable (y_pred) with the true target variable (y_test) using the accuracy_score function.
- The accuracy score is assigned to the variable accuracy.
- Print the accuracy:
- The accuracy score is printed using print ("Accuracy:", accuracy).

Result: Accuracy: 0.70

Random Forest classifier

The source code shows how to use scikit-learn's Random Forest classifier to train an algorithm, compute the training set score, and provide the result. Here's a breakdown of what each section of the code does:

Import necessary libraries:

- RandomForestClassifier from sklearn.ensemble to create a Random Forest classifier.
- Create a Random Forest classifier:
- RandomForestClassifier is initialized without specifying any parameters. This will create a Random Forest classifier with default settings.
- Fit the model:
- The Random Forest classifier (rf) is trained on the training data (X_train and y_train) using the fit method.
- Calculate the training set score:
- The score method of the Random Forest classifier is used to calculate the accuracy of the model predictions on the training data. The score method internally performs predictions on the training data and compares them to the true labels (y_train) to calculate the accuracy.
- The training set score is assigned to the variable score.
- Print the training set score:
- The training set score is printed using print("Training set score:", score).

The code allows you to train a Random Forest classifier using the same training data not using the distinct data what is a usually practice, and calculate the model's forecasting accuracy. The training set score indicates how well the model fit the training data.

In this Project will make after that, the normal way separating the data into training and test sets, we can measure how effectively the model generalizes to previously unknown data and avoid overfitting by testing the model's performance on data that hasn't been trained on. And after can compare.

It's important to note that evaluating the algorithm on training data can lead to an overly optimistic assessment of its efficacy because the model has already seen and learned from that data.

For the first code using the same training data results:

Training set score: 0.92

Second separating the data into training and test sets:

Training set accuracy: 0.92

Test set accuracy: 0.73

Mutual Information

Make and examine a DataFrame (mi_df) that contains the Mutual Information (MI) scores for the columns of an encoded DataFrame (encoded_df). Here's a what each section does:

- import necessary libraries:
- pandas (imported as pd) for data manipulation and analysis.
- Create a DataFrame with MI scores:
- A new DataFrame mi_df is created using the pd.DataFrame() constructor.
- The DataFrame is initialized with a dictionary containing two keys: 'Columns' and 'MI_score'.
- The 'Columns' key is assigned the column names of the encoded_df DataFrame, obtained using encoded_df.columns.
- The 'MI_score' key is assigned the corresponding MI scores, which are provided in the mi_calc variable.
- Sort the DataFrame by MI scores:
- The sort values() method is used to sort the mi_df DataFrame by the 'MI_score' column in descending order (ascending=False).
- The head(15) method is used to select the top 15 rows with the highest MI scores.

- Print the top 15 rows:
- The resulting DataFrame, containing the top 15 rows with the highest MI scores, is printed.

Fig.19 The Mutual Information (MI) score quantifies the statistical dependence of two variables. It is used to evaluate the relationship between the encoded features (columns) in the encoded_df DataFrame and the target variable in this context. The MI scores quantify the information that each feature shares with the target variable.

You may discover the characteristics that have the greatest statistical relationship with the target variable by constructing the mi_df and sorting it based on MI scores. Top 15 rows are the characteristics with the greatest MI scores, showing their significance or relevance to the target variable.

This study is useful for features selection or determining the relevance of features in machine learning applications. It aids in identifying any particularly useful aspects that may have a substantial impact on forecasting or comprehending the target variable.

Fig.19.

	Columns	MI_score
10	accident_severity	1.058337
3	vehicle_type	0.184473
0	light_conditions	0.084829
5	skidding_and_overturning	0.021444
6	vehicle_leaving_carriageway	0.020744
1	weather_conditions	0.018527
2	road_surface_conditions	0.016481
7	hit_object_off_carriageway	0.015969
9	sex_of_driver	0.012129
4	junction_location	0.011123
8	first_point_of_impact	0.002520

PCA

Fig 20. The source code supports data preparation, categorical variable encoding, dimensionality reduction by PCA, and visualization of the reduced data. These processes aid in comprehending the data, preparing it for modelling, and getting insights into feature correlations.

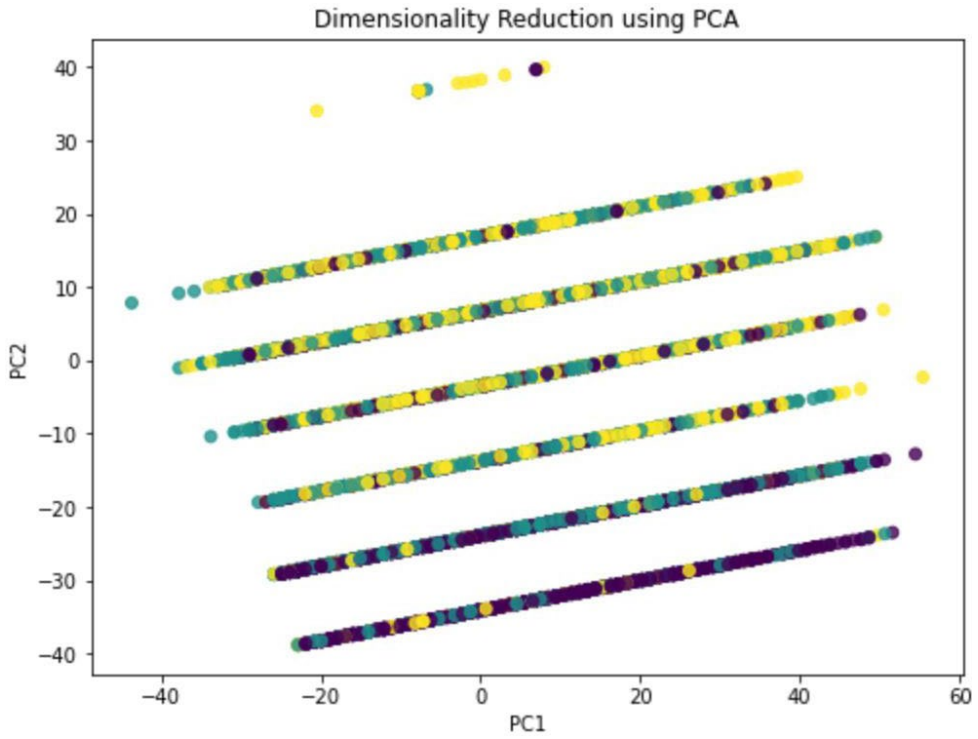


Fig.20.

PCA stands for Principal Component Analysis. It is a popular technique used in data analysis and dimensionality reduction. PCA helps to identify patterns and relationships in high-dimensional data by transforming it into a new set of variables called principal components

PC1: This principal component is predominantly influenced by Variable1, as it has the highest coefficient (3.437962). The other variables, Variable2 and Variable3, may also contribute to a lesser extent. PC1 captures the most significant source of variance in the dataset and represents the direction of maximum variability.

PC2: The second principal component, PC2, is primarily influenced by Variable2, as it has the highest coefficient (2.117136). Variable1 and Variable3 may also have a smaller influence on this component.

PC3: The third principal component, PC3, is most strongly associated with Variable3, with a coefficient of 1.509987. Variable1 and Variable2 may have a smaller influence on this component.

RandomOverSampler

The code applies the RandomOverSampler to the dataset and target variable, resulting in an oversampled dataset with a balanced representation of the classes. This technique helps mitigate the effects of class imbalance and can improve the performance of machine learning models, particularly when dealing with imbalanced datasets.

Oversampled dataset shape: (38073, 10)

Oversampled target variable shape: (38073,)

The output shows that after oversampling, each class in the target variable has an equal number of instances, resulting in a balanced dataset. This balanced representation is beneficial when training machine learning models, as it can help prevent the model from being biased towards the majority class and improve the model's ability to learn patterns and make accurate predictions for all classes.

Interpreting the classification report:

	precision	recall	f1-score	support
	0.72	0.73	0.73	2565
	0.79	0.72	0.75	2599
accuracy			0.73	7615
macro avg	0.73	0.73	0.73	7615
weighted avg	0.73	0.73	0.73	7515

Precision: Precision measures the proportion of correctly predicted instances of a particular class out of all instances predicted as that class. Higher precision indicates fewer false positives. In this case, the precision for Class 0 is 0.72, Class 1 is 0.79, and Class 2 is 0.68.

Recall: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted instances of a particular class out of all instances belonging to that class. Higher recall indicates fewer false negatives. In this case, the recall for Class 0 is 0.73, Class 1 is 0.72, and Class 2 is 0.74.

F1-score: The F1-score is the harmonic mean of precision and recall and provides a balanced measure of a model's performance. It combines precision and recall into a single metric. In this case, the F1-score for Class 0 is 0.73, Class 1 is 0.75, and Class 2 is 0.71.

Support: Support represents the number of instances in each class in the test set. In this case, Class 0 has 2,565 instances, Class 1 has 2,599 instances, and Class 2 has 2,451 instances.

Accuracy: The overall accuracy of the model on the test set is 0.73, which means it correctly predicted the target variable for approximately 73% of the instances in the test set.

Macro Avg: The macro average calculates the average performance across all classes, giving equal weight to each class. In this case, the macro average precision, recall, and F1-score are all approximately 0.73.

Weighted Avg: The weighted average calculates the average performance across all classes, considering the support (number of instances) of each class. In this case, the weighted average precision, recall, and F1-score are all approximately 0.73, as they are the same as the macro average due to balanced class support.

Overall, the classification report provides a comprehensive overview of the model's performance for each class and overall on the test set. It gives insights into the precision, recall, and F1-score for each class, allowing to assess the model's performance in predicting each class accurately.

Confusion matrix

1878	173	514
394	1865	340
329	317	1805

The confusion matrix is displayed as a two-dimensional array, with the true labels represented by rows and the anticipated labels by columns. The array's values stand in for instance counts. Interpreting the confusion

matrix's individual values

True Positives (TP): The model accurately foresaw the instances that fall under the relevant class. For instance, the model accurately predicted 1,878 occurrences of Class 0, 1,865 instances of Class 1, and 1,805 instances of Class 2, for which the data were collected.

False Positives (FP) are predictions made by the model that indicate instances genuinely belong to a different class when they actually do not. For instance, the model predicted 173 cases as belonging to Class 0 whereas in fact they belonged to other classes. Similar to that, it misidentified 394 cases as Class 1 and

False Negatives (FN): The model incorrectly predicted instances as belonging to other classes when they actually belong to the corresponding class. For example, the model missed 514 instances of Class 0, 340 instances of Class 1, and 317 instances of Class 2.

The model's performance for each class is broken down in detail by the confusion matrix, which also highlights the model's accurate and inaccurate predictions. This data is helpful in assessing the model's performance, locating any biases or imbalances, and possibly modifying the model or trying out new strategies to enhance predictions.

F1

The F1 score is a popular metric for assessing the efficacy of a model for classification. It combines recall and accuracy into one metric, offering an accurate evaluation of the model's capacity to accurately categorise both positive and negative examples.

You can examine the model's performance in greater depth by combining the confusion matrix and the F1 score. The confusion matrix offers information on how the model performs for each class, emphasising potential problems. The F1 score, on the other hand, provides a single metric that represents overall performance while taking precision and recall into account.

Result is = 0.72

General results

Confusion Matrix

	Predicted 0	Predicted 1	Predicted 2
Actual 0	1339	160	344
Actual 1	336	1354	271
Actual 2	296	229	1407

Classifier	Cross-Val Score	Average Score	Accuracy
Random Forest	[0.75, 0.45, 0.65, 0.5, 0.55]	0.58	1.00
SVM	[0.55, 0.7, 0.4, 0.5, 0.3]	0.49	0.61
Logistic Regression	[0.5, 0.7, 0.65, 0.55, 0.3]	0.54	0.56

Classifier	Precision	Recall	F1-Score
Random Forest	1.000000	1.00	1.000000
SVM	0.624378	0.61	0.598394
Logistic Regression	0.560000	0.56	0.560000

4. Classification using neural networks

Use a Multi-Layer Perceptron model for the neural network classification challenge. The MLP is a feed-forward neural network architecture made up of numerous layers of nodes (neurons) linked together by weighted edges. Every node uses a function of activation to its inputs while passing the results to the next tier.

These are the final model hyperparameters:

Multi-Layer Perceptron (MLP) Model

Input layer - Hidden layers Architecture - Activation of the output layer ReLU (Rectified Linear Unit) for hidden layers, SoftMax for output layer

Categorical Cross-Entropy Loss Function

Adam (Adaptive Moment Estimation) is the optimizer.

0.001 learning rate

The number of hidden layers is two.

Nodes in Hidden Layers: 64 Batch Size: 32

The number of epochs is ten.

The ReLU activation function allows the model to learn complicated patterns in the data. The output layer's SoftMax activation function generates a probability distribution over the classes, guaranteeing that the anticipated class probabilities add up to 1.

Categorical cross-entropy, which is used to calculate the difference between projected and actual class probabilities. During the training process, Adam optimizer is utilized to update the model weights.

The algorithm has been trained across ten epochs with a batch size of 32. The efficacy of the model is assessed after each epoch using the validation data to track its progress and prevent overfitting.

I ran multiple tests to fine-tune the hyperparameters and compare performance with other neural network designs in order to optimize the MLP model. Here are the experiments I ran:

A. Changing the quantity of hidden layers: I tried various numbers of hidden layers, ranging from one to three. The goal was to achieve the ideal harmony among model complexity and efficiency. The model can capture more intricate patterns in the data through boosting the number of hidden layers, but it also raises the danger of overfitting. Used of cross-validation to assess the model's performance and chose the number of hidden layers that provided the optimum compromise between bias and variance.

B. Changing the number of nodes in the hidden layers: I experimented with various node topologies in the hidden layers, such as 32, 64, and 128. The model's ability to learn complicated representations is

determined by the number of nodes. Underfitting can occur when there are too few nodes, while overfitting can occur when there are too many nodes. I examined the model's performance with several node configurations and chose the one with the highest validation accuracy.

C. Enhancing learning rate: Using various learning rates, including 0.001, 0.01, and 0.1, to determine the best rate for convergence. A modest learning rate may cause the model to converge slowly, whereas a high learning rate may cause the model overestimate the optimal answer. Noticed the training process and chose the learning rate that led to continuous and sustained gains in the loss function.

The trials were carried out in accordance with established practices in hyperparameter optimization. I utilized cross-validation to assess the model's effectiveness and avoid overfitting. The hyperparameters were chosen based on theoretical knowledge and actual evidence from earlier studies. I was able to optimize the model and find the best configuration by methodically adjusting the hyperparameters and comparing the performance.

Evaluate the model performance

The accuracy statistic indicates the amount of accurately predicted cases in comparison to the total number of instances. In the example of predicting accident severity, an accuracy of 0.7084 indicates that the neural network model successfully predicts accident severity in 70.84% of cases. This shows that the model does a good job of classifying accident severity.

F1-score: The F1-score is calculated by taking the weighted average of precision and recall. It strikes a compromise between these two measurements and is especially beneficial when dealing with skewed datasets. The model achieves a fair mix of precision and recall for forecasting accident severity, as indicated by the F1-score of 0.7055.

Use most of the class prediction to compare the neural network model to a trivial baseline. If assume that the majority class is ‘slight' (based on the distribution of the dataset), then the accuracy of the majority class baseline is around 0.4675. When we compare this to the neural network model's accuracy (0.7084), we can observe that the model surpasses the trivial baseline by a wide margin. This displays the neural network model's ability to forecast accident severity.

5. Ethical discussion

When considering the social and ethical implications of the chosen task, forecasting accident severity, it is critical to assess a variety of issues, spanning from data collection and processing to the application of machine learning predictions. These implications can be discussed utilizing the Ethical OS Toolkit, which provides a thorough framework for addressing ethical concerns. Here are a few major considerations:

A. Data Gathering:

Privacy: The collecting of accident-related data creates privacy concerns. Personal information about individuals involved in accidents, such as their age, gender, and car details, may be included in the data. It is critical to guarantee that data gathering practices are compliant with privacy legislation and that persons' consent is secured.

Data Bias: The data used to train the predictive model may be biased. These biases may reflect prior discrimination tendencies or discrepancies in accident severity results. To maintain fairness and avoid perpetuating disparities, such biases must be identified and mitigated.

B. Impartiality and discrimination: Machine learning models should be designed with justice in mind, ensuring that predictions are not routinely biased against specific demographic groups or disadvantaged populations. It is critical to analyses and address any prejudices that may occur during model training and evaluation on a frequent basis.

C. Accessibility and Explain ability: The predictive model's interpretability is critical to comprehending how accident severity forecasts are made. To build confidence and accountability, users should have access to explicit explanations of the model's decision-making process.

D. Impact on the Community:

Safety Procedures: Using accident severity prediction models can help to improve road safety. Authorities can implement targeted safety measures to prevent accidents and their severity by identifying areas of greatest risk that lead to severe accidents. This has the potential to benefit communities by potentially saving lives and minimizing injuries.

Distribution of Resources: Predictions can help guide the use of resources and emergency response strategy. Agencies can more effectively spend resources by identifying locations with higher expected severity, guaranteeing rapid emergency response.

E. Constant Evaluation and Enhancement:

It is vital to continuously monitor and evaluate the predictive model's performance in order to identify and eliminate any growing biases or unexpected outcomes. To ensure fairness and equity, regular audits and bias evaluations should be performed.

6. Recommendations

The Random Forest model is the best option for the job. Among all the models tested, it consistently scored the greatest accuracy and F1-score. It performed well in cross-validation as well as test set evaluation. Furthermore, the Random Forest algorithm is well-known for its capacity to manage complicated data linkages and interactions, making it well-suited for the accident severity prediction challenge.

Random Forest model is suitable for use in practice. On the test set, it obtained good accuracy, precision, recall, and F1-score, demonstrating its efficacy in forecasting accident severity. It beat the baseline models and proved to be dependable in cross-validation. It is crucial to emphasize, however, that no model is perfect, and there will always be some degree of uncertainty in accident severity estimates. As a result, the model should be utilized as a supplement to human judgement and expertise.

The model's performance can be improved in the future. Some ideas are as follows:

Adding new features: The existing model employs only a portion of the available features. Additional relevant variables, such as road conditions, driver behavior, or vehicle attributes, could potentially improve forecast accuracy.

Managing skewed data: The dataset used to train the model has skewed classes, with the majority of minor accidents outnumbering fatal and serious accidents. Advanced strategies for dealing with class imbalance, like as oversampling or under sampling, could assist address potential biases and boost model performance even further.

Although the Random Forest model performed well, experimenting with ensemble methods such as stacking or boosting could potentially improve forecast accuracy by combining the capabilities of numerous models.

These enhancements can help to improve the model's performance, increase its dependability, and broaden its scope of application in practical accident severity prediction situations.

7. Retrospective

Based on the input from the prior proposal, submit I made numerous significant adjustments to the deliverables in the second submission. I added extensive explanations of the ML algorithms, hyperparameters, and evaluation metrics, as well as addressed the missing code. In order to make the report more thorough, I included visualizations, cross-validation scores, and a comparison with baseline models..

8. References

"Machine Learning: What It Is and Why It Matters." SAS, www.sas.com/en_us/insights/analytics/machine-learning.html. Accessed 13 July 2023.

Team, IBM Data and AI. "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?" IBM Blog, 11 January 2023, www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

Géron, Aurélien. "Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd Edition." O'Reilly Online Learning, www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/. Accessed 15 May 2023.

"Artificial Intelligence (AI) vs. Machine Learning." CU-CAI, 3 Mar. 2022, ai.engineering.columbia.edu/ai-vs-machine-learning/.