PROJECT REPORT

Traffic accident Prediction

Created by - NILOY SARKAR

Date- 21.8.24

Table of Contents

1. Introduction
2. Dataset Description
    (a) Source
    (b) Dataset Description
    (c) Imbalanced Dataset
3. Dataset Pre-processing
    (a) Faults
    (b) Solutions
4. Feature Scaling
5. Dataset Splitting
6. Model Training & Testing
7. Model Selection/ Comparison analysis
8. Conclusion

INTRODUCTION

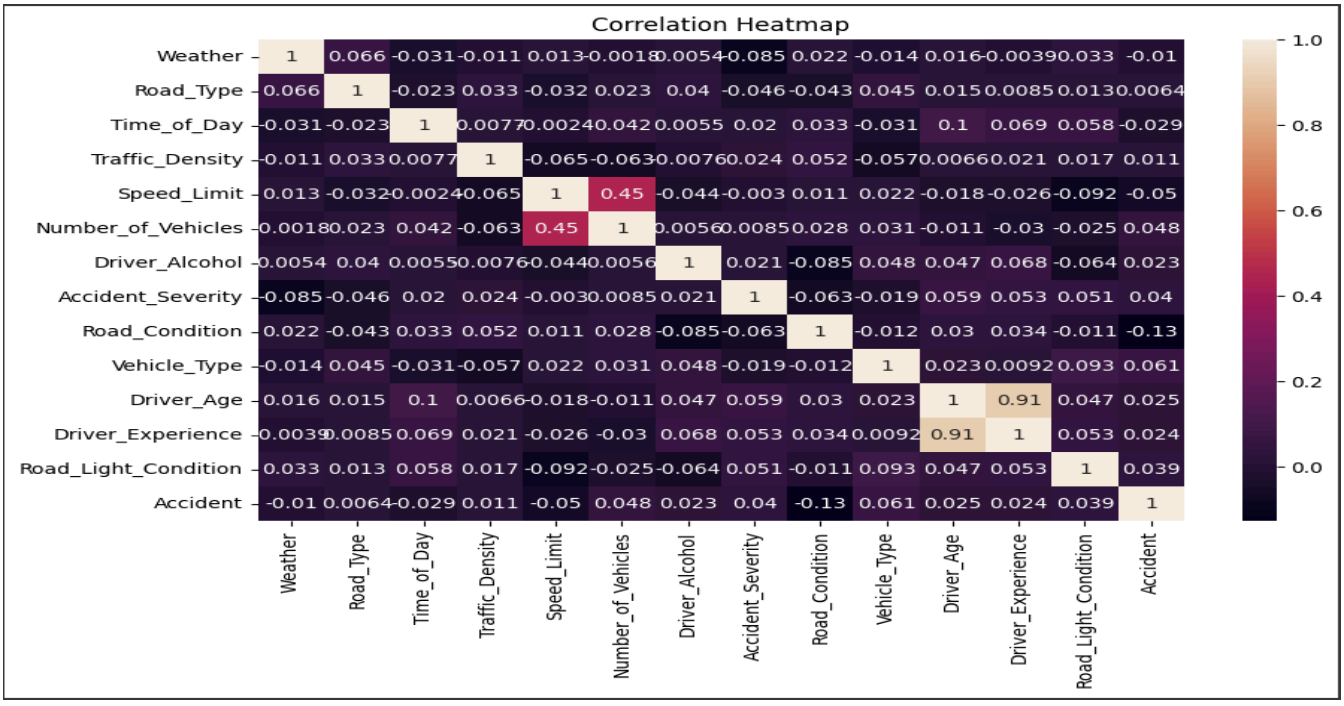
This project predicts traffic accidents and their violence based on weather, road conditions, driver behavior, and traffic levels. The goal is to find patterns that can help us to improve road safety and reduce accidents. By using this essential information, the government and drivers can take better safety to avoid risks. Actually, we are motivated by this project because we need to save lives, prevent injuries, and minimize the cost of traffic accidents.

Dataset Description

(a) Source - Kaggle
(https://www.kaggle.com/datasets/denkuznetz/traffic-accident-prediction)

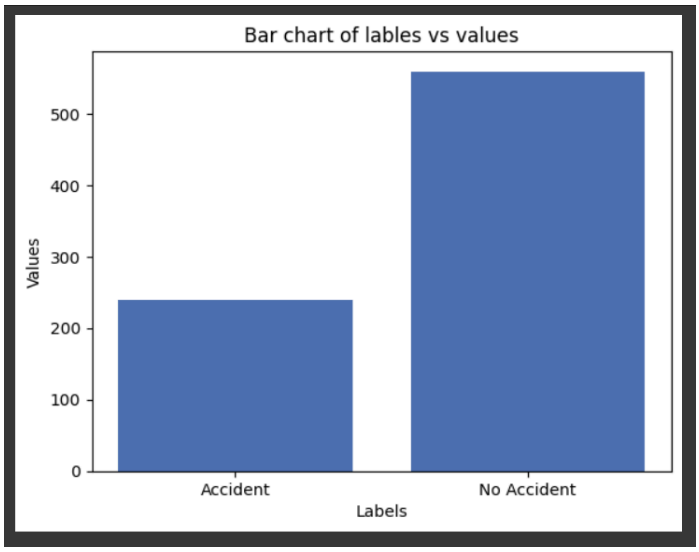
(b) Dataset description -

- There are 14 features in this dataset.
- Binary classification problem. Because we are predicting two labels from the features.
- There are 11760 data points.
- Categorical features are present in our dataset.
- Correlation of all the features (input and output features) (apply heatmap using the seaborn library) is given below -



(c) Imbalanced Dataset -

- For the output feature- No, All unique classes have not an equal number of instances .
- Represent using a bar chart of N classes (N=number of classes have in our dataset)



Dataset Pre-processing

(a) Faults -

Null Values : 435

Categorical Values:

Dealing with categorical values

```
[57]  1 from sklearn.preprocessing import LabelEncoder
      2 le = LabelEncoder()
      3
      4 for cat_col in non_numeric_column:
      5     df[cat_col] = le.fit_transform(df[cat_col])
```

```
1 df.dtypes
```

	0
Weather	int64
Road_Type	int64
Time_of_Day	int64
Traffic_Density	float64
Speed_Limit	float64
Number_of_Vehicles	float64
Driver_Alcohol	float64
Accident_Severity	int64
Road_Condition	int64
Vehicle_Type	int64
Driver_Age	float64
Driver_Experience	float64
Road_Light_Condition	int64
Accident	int64

dtype: object

(b) Solutions -

Firstly, We faced a Null value problem in our dataset.We know that if we have a 5% amount of null value in our full dataset, we can delete the null value directly. But in our dataset, there are a total 840 rows and 435 of them had null values. That's why we can not delete the null values row. If we delete the null values row then likely half of the data will be lost.

Secondly, We faced a problem in Categorical value. When we run the module, if there is any categorical value there the module will not run. Because machines are always working with numerical values. So, we just encoded the non-numerical value of our dataset. We used a label-encoder function and ran a loop for every non-numerical value to transform into a numerical value. After doing this process, we get all our data in numerical value.

Feature Scaling

In our dataset, some of the variable variance are very high and some other variable variance are very low. So, we can see that huge difference between the variable variance. If we use like as it is, the module will prioritize the high variable variance and ignore the low variable variance. In this way, we will lose the data. For this reason, we have to use feature scaling for our dataset. Now, we can use standard scaler for scaling our target features in our dataset and we will get our solution for this problem.

Feature Scaling

```
[ ] 1 y = [0, 1]
    2 std = np.std(y)
    3 mean = np.mean(y)
    4
    5 print((0-mean)/std)
    6 print((1-mean)/std)
```

```
-1.0
1.0
```

```
[70] 1 features = df.drop('Accident', axis=1).columns.to_list()
     2 print(features)
```

```
['Weather', 'Road_Type', 'Time_of_Day', 'Traffic_Density', 'Speed_Limit', 'Number_of_Vehicles', 'Driver_Alcohol', 'Accident_Severity', 'Road_Condition', 'Vehicle_Type', 'Driver_Age', 'Driver_Experience', 'Road_Light_Condition']
```

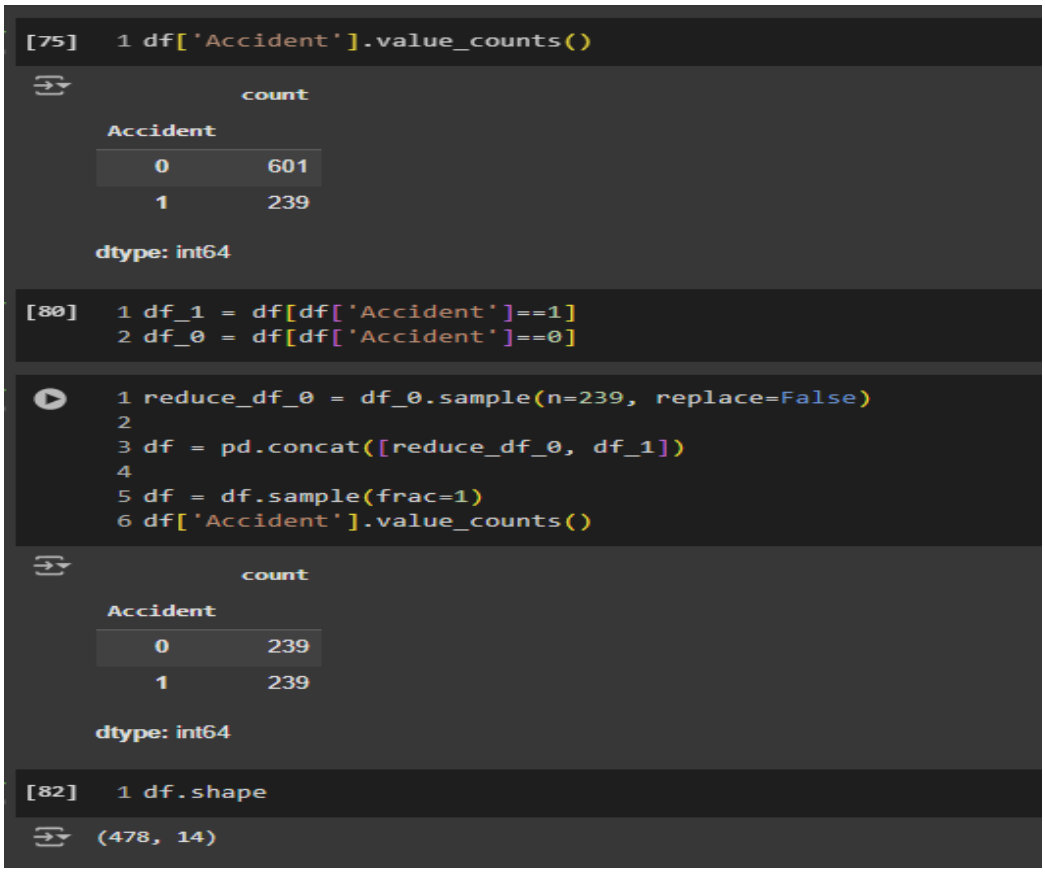
```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 df[features] = scaler.fit_transform(df[features])
4 df.head()
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicles	Driver_Alcohol	Accident_Severity	Road_Condition	Vehicle_Type	Driver_Age	Driver_Experience	Road_Light_Condition	Accident
0	0.665306	-1.122817	0.835725	-0.001557	0.942145	0.878649	-0.423999	-0.328934	1.760777	-0.434882	0.526107	0.606155	-0.859781	0
1	-0.949888	2.017582	1.785670	-0.001557	1.580837	-0.138670	-0.423999	1.376650	1.760777	2.375127	0.390400	0.270069	-0.859781	0
2	0.665306	-0.076018	-0.114220	-0.001557	-0.335237	0.369989	-0.423999	-0.328934	0.045933	-0.434882	0.729668	0.875024	-0.859781	0
3	-0.949888	-1.122817	-1.064164	1.306418	-0.335237	-0.138670	-0.423999	-0.328934	0.903355	-1.839887	-0.627403	-0.536537	0.670337	0
4	0.665306	-0.076018	0.835725	-0.001557	3.975929	3.930606	-0.423999	-0.328934	-0.811489	-0.434882	1.272496	1.076675	-0.859781	1

Dataset Splitting

From our dataset, we can see that the dataset is imbalanced because accidents occur 239 times but no accident happens 601 times. If we run a module normally with this data without balancing, the module will be biased and the module will not predict correctly in new data. Now, we will reduce the number of accidents value and make both accident and no accident value same. So, we need to use a stratified dataset for building the perfect module.

For example, if we have 100 data, we will use 70 for training and 30 for testing purpose.



Model Training & Testing

Our dataset is a classification problem. So, we selected KNN, Decision Tree, Logistic Regression and Random Forest for our module. We train our module with this 4 models.

After trained our module, we also test the model with dataset and get a prediction result.

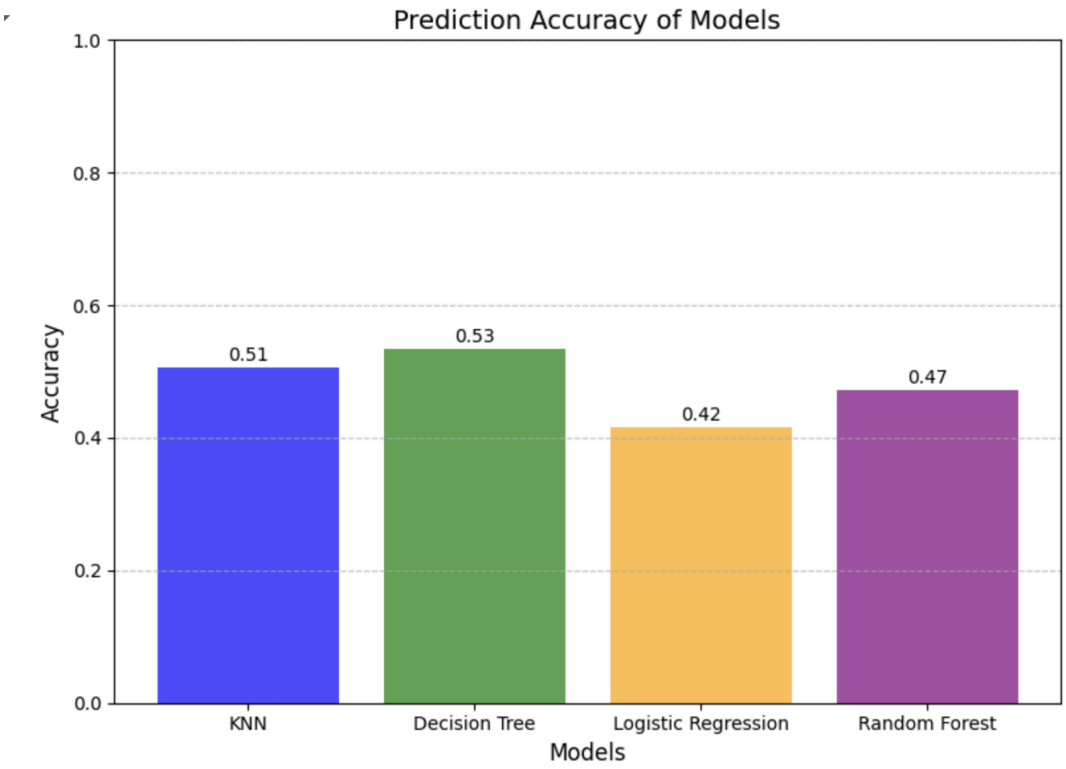
Here is the Model Accuracy result(approx) given below -

MODEL NAME	MODEL ACCURACY(approx)	
KNN	0.506	
Decision Tree	0.534	
Logistic Regression	0.416	
Random Forest	0.4722	

Model Selection/ Comparison analysis

Bar Chart showcasing prediction accuracy of all models we used -

```
21 plt.show()
```



This bar chart compares the prediction accuracy of our selected four classification models: KNN, Logistic Regression, Decision Tree, and Random Forest. Decision tree achieves the highest accuracy at approximately 0.53.KNN follows with an accuracy of around 0.51. Random Forest has a slightly lower accuracy of 0.47. Logistic Regression performs the least accurately, with an accuracy of 0.42.

The differences indicate that Decision Tree is the best-performing model among all four models for this dataset, while Logistic Regression is the least effective. The other two models fall in between with moderate performance.

Showcasing Precision and Recall comparison , Confusion Matrix For Each Model Below:

KNN MODEL -

Modeling KNN

```
[ ]  1 from sklearn.neighbors import KNeighborsClassifier
     2 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
     3 from sklearn import neighbors, datasets
     4
     5 model_1 = KNeighborsClassifier(n_neighbors=5)
     6 model_1.fit(x_train, y_train)
     7
     8
     9 y_pred = model_1.predict(x_test)
    10 conf_matrix_1 = confusion_matrix(y_test, y_pred)
    11 report = classification_report(y_test, y_pred)
    12 acc1 = accuracy_score(y_test, y_pred)
    13 print(acc1)
    14 print(conf_matrix_1)
    15 print(report)
```

```
0.5069444444444444
[[32 41]
 [30 41]]
              precision    recall  f1-score   support

           0       0.52      0.44      0.47        73
           1       0.50      0.58      0.54        71

    accuracy                           0.51       144
   macro avg       0.51      0.51      0.51       144
weighted avg       0.51      0.51      0.50       144
```

Here, Confusion Matrix -

True Positives: 41

True Negatives: 32

False Positives: 41

False Negatives: 30

Decision Tree -

Decision Tree

```
[ ]  1 from sklearn.tree import DecisionTreeClassifier
     2
     3
     4 model2 = DecisionTreeClassifier()
     5 model2.fit(x_train, y_train)
     6
     7 y_pred = model2.predict(x_test)
     8 conf_matrix_2 = confusion_matrix(y_test, y_pred)
     9 report = classification_report(y_test, y_pred)
    10 acc2 = accuracy_score(y_test, y_pred)
    11 print(acc2)
    12 print(conf_matrix_2)
    13 print(report)
```

```
0.5347222222222222
[[39 34]
 [33 38]]
              precision    recall  f1-score   support

           0       0.54      0.53      0.54        73
           1       0.53      0.54      0.53        71

    accuracy                           0.53       144
   macro avg       0.53      0.53      0.53       144
weighted avg       0.53      0.53      0.53       144
```

Here, Confusion Matrix -

True Positives: 38

True Negatives: 39

False Positives: 34

False Negatives: 33

Logistic Regression -

Logistic Regression

```
[ ]  1 from sklearn.linear_model import LogisticRegression
     2
     3 model1 = LogisticRegression()
     4 model1.fit(x_train, y_train)
     5
     6
     7 y_pred = model1.predict(x_test)
     8 conf_matrix_3 = confusion_matrix(y_test, y_pred)
     9 report = classification_report(y_test, y_pred)
    10 acc3 = accuracy_score(y_test, y_pred)
    11 print(acc3)
    12 print(conf_matrix_3)
    13 print(report)
```

```
0.4166666666666667
[[33 40]
 [44 27]]
              precision    recall  f1-score   support

           0       0.43      0.45      0.44        73
           1       0.40      0.38      0.39        71

    accuracy                           0.42       144
   macro avg       0.42      0.42      0.42       144
weighted avg       0.42      0.42      0.42       144
```

Here, Confusion Matrix -

True Positives: 27

True Negatives: 33

False Positives: 40

False Negatives: 44

Random Forest -

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

model3 = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42 , criterion='entropy')
model3.fit(x_train, y_train)


y_pred = model3.predict(x_test)
conf_matrix_4 = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
acc4 = accuracy_score(y_test, y_pred)
print(acc4)
print(conf_matrix_4)
print(report)
```

```
0.4722222222222222
[[29 44]
 [32 39]]
              precision    recall  f1-score   support

           0       0.48      0.40      0.43        73
           1       0.47      0.55      0.51        71

    accuracy                           0.47       144
   macro avg       0.47      0.47      0.47       144
weighted avg       0.47      0.47      0.47       144
```

Here, Confusion Matrix -

True Positives: 39

True Negatives: 29

False Positives: 44

False Negatives: 32

Now, Based on the model evaluation matrix, we can say that Decision Tree has fewer false positives compared to all other models we selected, improving classification balance. Decision Tree shows us slightly higher precision for both classes, indicating better positive predictive value among other models. Decision Tree provides more consistent recall across both classes compared to other models.

CONCLUSION

In this module, we measure the accuracy of four models for predicting traffic accidents. The Decision Tree performed the best, with an accuracy of 53.4%, suggesting it may effectively capture non-linear patterns in the data. KNN and Random Forest showed moderate performance, with accuracies of 50.6% and 47.2%, respectively. However, Logistic Regression had the lowest accuracy, at 41.6%, possibly due to its limitations in handling complex or non-linear relationships in the dataset.

To improve these results, further steps should use more essential methods to enhance predictive performance.