Car Accident Analysis of UK

INFO-H-423 Data Mining

Group Project

Hridaya Sagar Subedi

Akash Malhotra

Haonan Jin

Yalei Li

TABLE OF CONTENT

Abstract

Millions of accidents happen in the United Kingdom, and in other countries yearly. Such accidents not only cost a lot of money as collateral damage but also human lives. A study[7] conducted at UCL found out that 50% of fatal accidents happen only at 5% of the locations in London. This shows that accidents are not random and thus it would be useful if we could find some pattern in them to prevent them. In this report, we have explored various methods to find these patterns. First, we show the hotspots in UK for accidents by using density based clustering (DBSCAN). Then we use and compare Logistic Regression, Random Forests and XGBoost to classify accidents as fatal or non-fatal. Next, we use Random Forests to rank the important factors responsible for the accidents. Lastly, using association rule mining, we study the correlation between these factors.

1. Data Preprocessing

Drop columns with missing data

There are some missing values in this datasets. Our criterion of handling missing values is that if the missing value exceeds 40%, it needs to be dropped. For instance, *Junction Detail* and *Junction Control* have to be dropped on account of plenty of missing values (details are shown in Figure 1.1). Dropping *Special_Conditions_at_Site* and *Carriageway_Hazards* is a better approach since they are string format and have a large number of negative class.

Also, we believe that *Date* and *Accident Index* can not be used in the project (Synthetic key) and are not very useful for making predictions, so we dropped them.

In addition, *Local Authority (Highway)*, *Local Authority (District)* and *LSOA of Accident Location* are some codes of location or accident, which are not related to our tasks.

Finally, *Number_of_Casualties* and *Did_Police_Officer_Attend_Scene_of_Accident* are meaningless because they characterise events that happen after the accident.

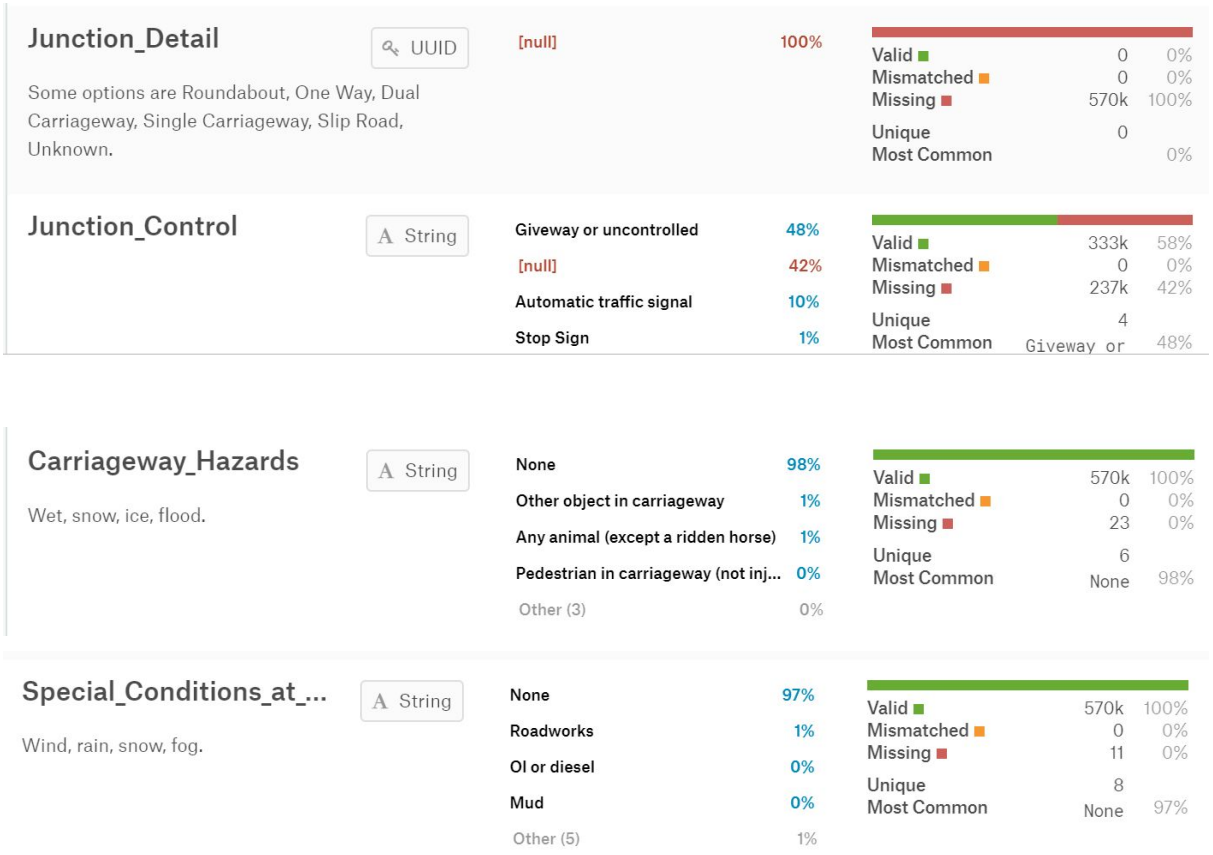


Figure 1.1. Column View of Dataset Details

Drop rows with 'Unknown' values

`Weather_Conditions` and **`Road_Type`** have a few “unknown” values (details are shown in Figure 1.2). So it is necessary for us to drop these unknown values.

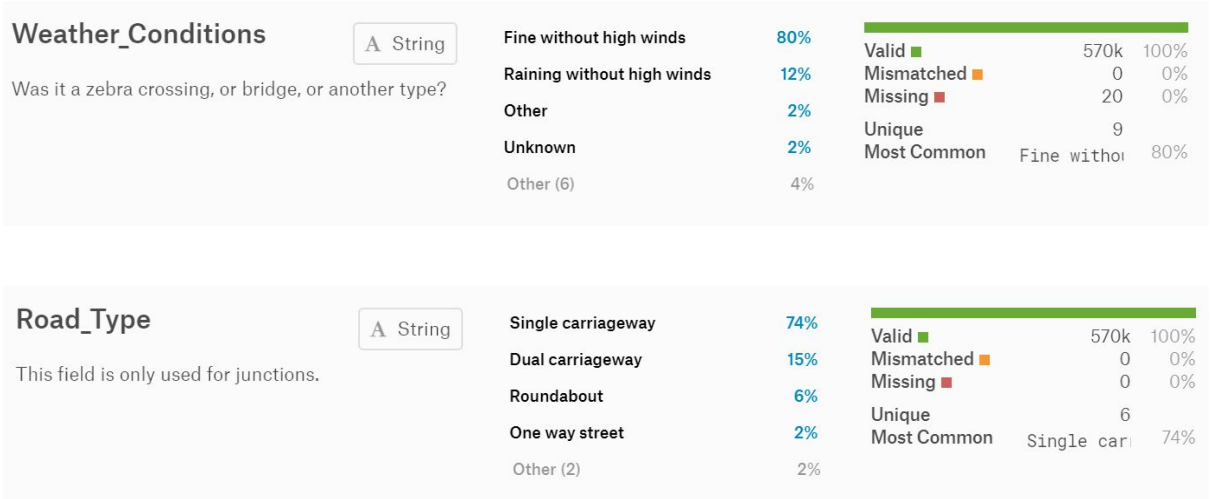


Figure 1.2. Details of Weather_Conditions & Road_Type Columns

Encode "String" Labels into "Int" Labels for easy training

In order to conduct data mining and machine learning work, encoding categorical data to numeric data is indispensable (i.e. P*edestrian_Crossing-Physical_Facilities, Light_Conditions, Weather_Conditions, Weather_Conditions, Pedestrian_Crossing-Human_Control, Pedestrian_Crossing-Human_Control).*

Creating additional columns with OneHot encoder

In order to analyze the fatal accident efficiently, a practical approach is to create new binary attributes of *Fatal*, *Severe*, and *Light* and drop *Severe* and *Light* based on the *Accident Severity*.

All in all, the general information of dataset after pre-processing is exhibited as Figure 1.3:

```
Data columns (total 26 columns):
Location_Easting_OSGR                        454809 non-null int64
Location_Northing_OSGR                       454809 non-null int64
Longitude                                    454809 non-null float64
Latitude                                     454809 non-null float64
Police_Force                                 454809 non-null int64
Accident_Severity                            454809 non-null int64
Number_of_Vehicles                           454809 non-null int64
Number_of_Casualties                         454809 non-null int64
Day_of_Week                                  454809 non-null int64
Time                                         454809 non-null int32
1st_Road_Class                               454809 non-null int64
1st_Road_Number                              454809 non-null int64
Road_Type                                    454809 non-null int32
Speed_limit                                  454809 non-null int64
2nd_Road_Class                               454809 non-null int64
2nd_Road_Number                              454809 non-null int64
Pedestrian_Crossing-Human_Control            454809 non-null int32
Pedestrian_Crossing-Physical_Facilities      454809 non-null int32
Light_Conditions                             454809 non-null int32
Weather_Conditions                           454809 non-null int32
Road_Surface_Conditions                      454809 non-null int32
Urban_or_Rural_Area                          454809 non-null int64
Year                                         454809 non-null int64
Fatal                                        454809 non-null uint8
```

Figure 1.3. General Information of Dataset After Pre-processing

2. Hotspots for Car Accidents

The UK traffic data from 2012 and 2014 data source consists of nearly half a million accidents in the process. The dataset contains 35 attributes. To identify the hotspots for car accidents, we processed the data for finding patterns in accidents with respect to location. As the dataset is large, plotting the accident zones directly on the map will show highly populous cities as accident hotspots, which does not convey meaningful information. Hence, we have divided the data into smaller blocks based on coordinates. We have only considered the Longitude and Latitude of each accident for our process as hotspot refers to the frequency of accidents. Then DBSCAN clustering [2] is employed on such small blocks to get the clusters. All the cluster coordinates - longitude and latitude are plotted in Google Maps to get a visual representation of hotspots.

As the location of accidents and their total count determines the hotspot. We first trimmed the data with columns Longitude and Latitude. Out of the dataset, we calculated the minimum and maximum value of Longitude and Latitude to define the boundary (length and breadth) of our data. Then finding the boundary, we divide the data into smaller blocks of equal length and breadth. These blocks are individually read again and clustered using DBSCAN. After computing the DBSCAN, all the core and boundary points coordinates are plotted on the map whereas noise points are disregarded.

Figure 2.1 demonstrates the clustering heatmap results of traffic accidents based on the interactive GoogleMap API. When zooming into the city-wise scale (Figure 2.2), it can be concluded that the cross-sections in the roads are the hotspots of accidents.

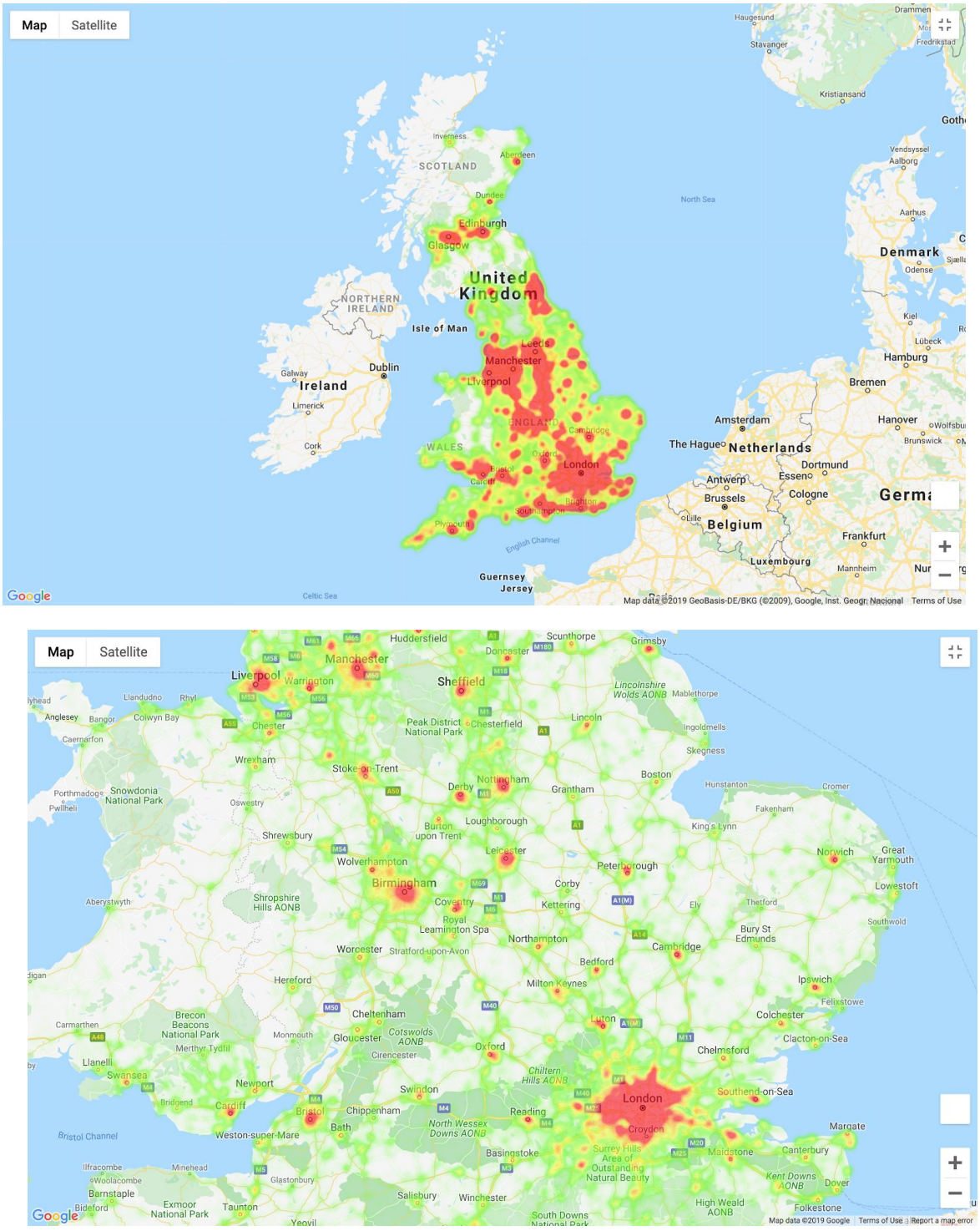


Figure 2.1. Overall DBSCAN Heatmap Results of UK Traffic Accident

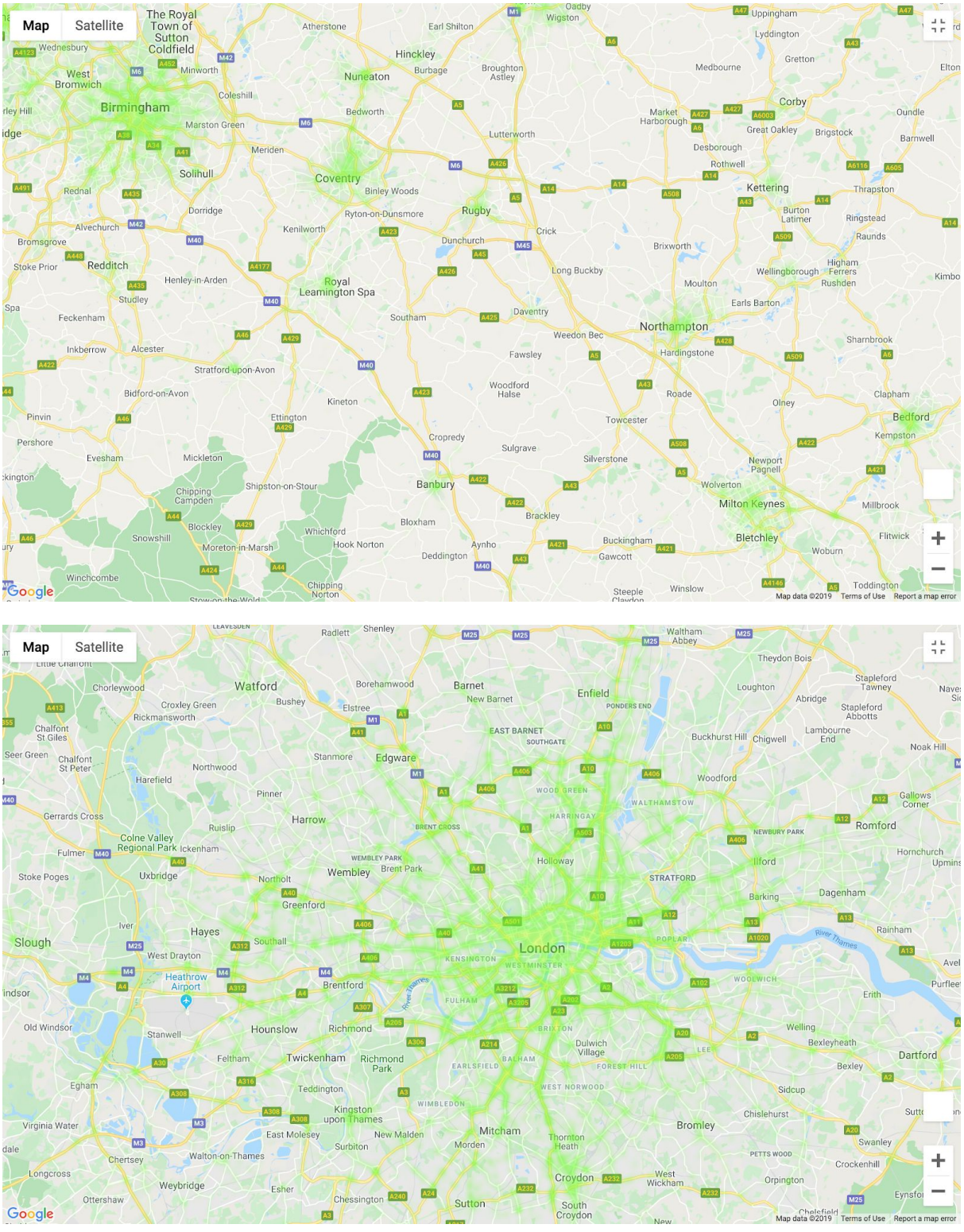


Figure 2.2. Zoomed-in Heatmap Result of Traffic Accidents

3. Fatal Accidents Prediction

3.1. Original Prediction

After the pre-processing, three related algorithms will be tested so as to compare the different approaches.

Logistic Regression

Logistic regression [3] is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labelled as "0" and "1".

Accuracy 98.92

	precision	recall	f1-score	support
0	0.989171	1.000000	0.994556	89977
1	0.000000	0.000000	0.000000	985
accuracy			0.989171	90962
macro avg	0.494586	0.500000	0.497278	90962
weighted avg	0.978460	0.989171	0.983786	90962

Predicted	0	All
Actual		
0	89977	89977
1	985	985
All	90962	90962

Random forest

Random forests [4] or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Accuracy 99.05

	precision	recall	f1-score	support
0	0.990511	1.000000	0.995233	89977
1	1.000000	0.124873	0.222022	985
accuracy			0.990524	90962
macro avg	0.995255	0.562437	0.608627	90962
weighted avg	0.990613	0.990524	0.986860	90962

Predicted	0	1	All
Actual			
0	89977	0	89977
1	862	123	985
All	90839	123	90962

XGBOOST

XGBoost [5] is an open-source software library which provides a gradient boosting framework for C++, Java, Python, R, and Julia. It works on Linux, Windows, and macOS. From the project description, it aims to provide a "Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library". It runs on a single machine, as well as the distributed processing frameworks Apache Hadoop, Apache

Spark, and Apache Flink. It has gained much popularity and attention recently as the algorithm of choice for many winning teams of machine learning competitions.

Accuracy 98.92

	precision	recall	f1-score	support
0	0.989171	1.000000	0.994556	89977
1	0.000000	0.000000	0.000000	985
accuracy			0.989171	90962
macro avg	0.494586	0.500000	0.497278	90962
weighted avg	0.978460	0.989171	0.983786	90962

Predicted Actual	0	All
0	89977	89977
1	985	985
All	90962	90962

Although the accuracy of three different algorithms is high, the recall and F1 score are not good on account of lacking more fatal accidents to support the models. This is because there are less fatal accidents in our datasets. Therefore, it is necessary to handle the class imbalance problem in the dataset.

3.2. Final Prediction with Oversampling Adjustment

There are a number of methods available to oversample a dataset used in a typical classification problem (using a classification algorithm to classify a set of images, given a labelled training set of images). The traditional techniques are to copy the fatal accidents or generate random fatal accidents, but they will make a great effect

on the accuracy and performance of models. The most common technique is known as SMOTE: Synthetic Minority Over-sampling Technique.[1]

```
#Before SMOTE
y_train.value_counts()
```

```
0    359595
1      4252
Name: Fatal, dtype: int64
```

```
#Count after SMOTE
pd.Series(y_resampled_train).value_counts()
```

```
1    359595
0    359595
dtype: int64
```

Logistic Regression after SMOTE data augmentation

```
Accuracy 54.66
              precision    recall  f1-score   support

           0   0.990974  0.546640  0.704606     89977
           1   0.012993  0.545178  0.025382       985

    accuracy                       0.546624     90962
   macro avg   0.501984  0.545909  0.364994     90962
weighted avg   0.980383  0.546624  0.697251     90962

Predicted      0      1    All
Actual
0          49185  40792  89977
1            448    537    985
All        49633  41329  90962
```

Random Forest after SMOTE data augmentation

	precision	recall	f1-score	support
0	0.9904	0.9997	0.9950	89977
1	0.8156	0.1168	0.2043	985
accuracy			0.9901	90962
macro avg	0.9030	0.5582	0.5997	90962
weighted avg	0.9885	0.9901	0.9865	90962

Predicted	0	1	All
Actual			
0	89951	26	89977
1	870	115	985
All	90821	141	90962

XGBOOST after SMOTE data augmentation

	precision	recall	f1-score	support
0	0.9914	0.8971	0.9419	89977
1	0.0302	0.2924	0.0547	985
accuracy			0.8905	90962
macro avg	0.5108	0.5947	0.4983	90962
weighted avg	0.9810	0.8905	0.9323	90962

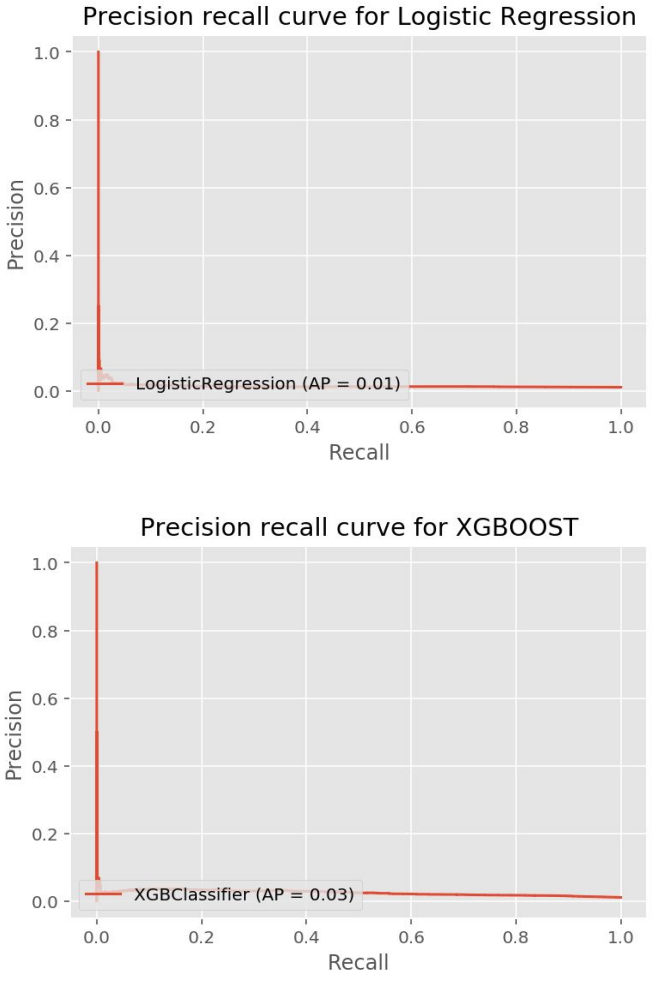
Predicted	0	1	All
Actual			
0	80718	9259	89977
1	697	288	985
All	81415	9547	90962

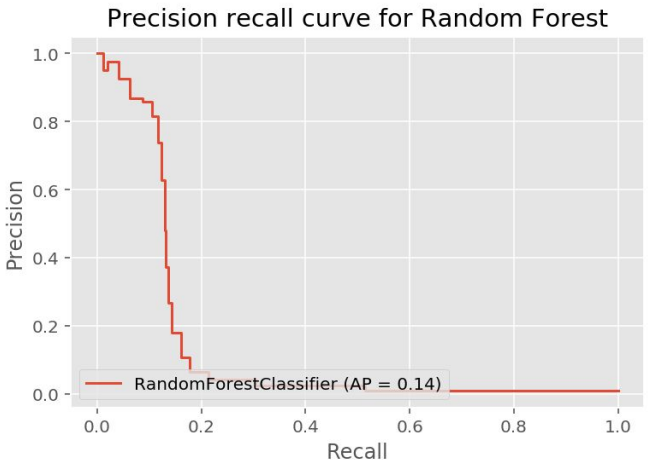
Precision-Recall:

When the dataset is not balanced, Confusion-Matrix can be misleading. In such a case, Precision-Recall curve gives us a better idea. The classifier usually predicts probabilities for each sample in a test dataset. Then, a threshold is decided in the case of binary classifier. For example, if probability > 0.5, classify the sample as 1

otherwise 0. We can adjust this threshold and achieve all the possible precision and recalls from Precision =1, Recall =0, to Precision = 0, Recall =1. There is a trade-off between Precision and Recall. Depending on the application, one metric may be preferred over another. Precision-Recall curves are the summary of this process of moving the threshold from 0 to 1 and therefore, of all the confusion matrices obtained along the way. The area under the Precision-Recall curve is another metric that is considered useful while comparing the classifiers.

We drew the Precision-Recall curves for Logistic Regression, Random Forests, and XGBoost. In this application, it is really hard to optimise both Precision and Recall simultaneously. Random Forests obtained the best Precision-Recall curve in this case as shown below.





All in all, For this case study, Random Forest and XGBoost is an appropriate machine-learning algorithm to predict fatal accidents. Logistic Regression is not an appropriate one because it needs strict data distribution. When solving the problems of oversampling with the help of Smote algorithm, the suitable data distribution has been affected. XGBoost is a boosted model, which is not strict to the data distribution.

4. Causes of Fatal Accidents & Traffic Condition Patterns

4.1. Feature Importance Based on Random Forest

In the previous section, predict models have been created with the help of SMOTE algorithm and random forests. Random Forests are also often used for feature selection in a data science workflow. The reason is because the tree-based strategies used by random forests naturally ranks by how well they improve the purity of the node. This mean decrease in impurity over all trees (called gini impurity). Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.[6] We have used this technique to view the important features as described by Random Forests. Therefore, the causes of fatal accidents with independent features can be viewed as in figure 4.1. The main causes of fatal accidents can be primarily concluded as Number of Vehicles, Light condition, Speed limit and Urban and Rural Areas.

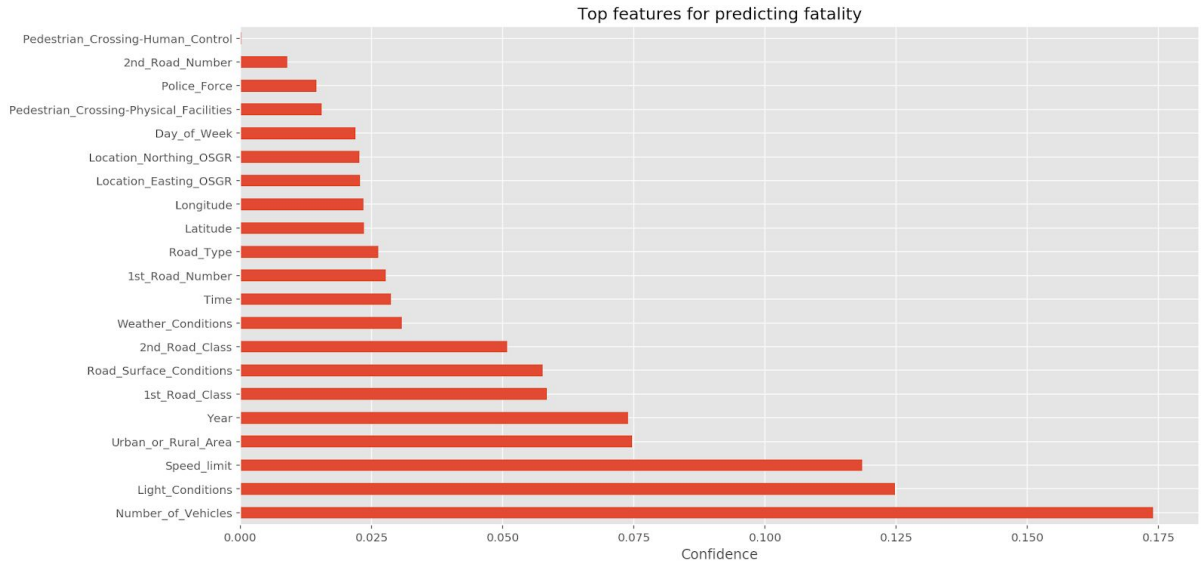


Figure 4.1 Results of Feature Importance for Fatal Accidents

4.2. Association Rule Mining

Given the results of the important features, we further examine the potential association patterns underlying these attributes and thus define the association rules leading to fatal accidents.

It is worth mentioning the additional data processing steps. For attributes with too much distinct values (i.e. *Longitude, Latitude*), we divide each column into 10 bins with the same range. Then, under the OneHot encoding, the dataset is expanded into 129 attributes with binominal values. The establishment of association rules is performed in RapidMiner platform, and the designed process is shown in Figure 4.2. FP-Growth parameter are set in default with minimum support requirement of 0.8, and association rules are generated in default setting with minimum confidence criterion of 0.8.

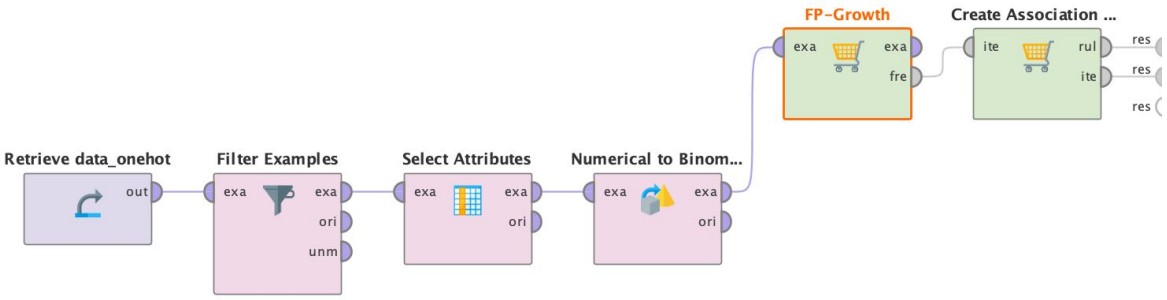


Figure 4.2 Designed Process of Association Rules in RapidMiner

After processing, the preliminary results of association rules are demonstrated as Table 4.3. This result is based on the total accident data and must be biased due to the unbalanced distribution of 'Slight' accident data. However, this primary result is still essential to identify the fatal predictive causes in the later stages. The highly frequent sets in this case will not be considered later since they merely shows a general statement of daily life situations. These items demonstrate a high possibility of appearance regardless of the accident (i.e. *'Weather Condition 1*' (*'Fine without high winds*'), '*Road surface condition 0*' (*'Dry'*), and '*Light condition 4*' (*'Daylight: Street light present*')).

No.	Premises	Conclusion	Support	Confidence	Lift
49	Weather_Conditions_1	Slight	0.69	0.84	1.00
13	Weather_Conditions_1	Road_Surface_Conditions_0	0.68	0.83	1.20
178	Road_Surface_Conditions_0	Weather_Conditions_1	0.68	0.98	1.20
32	Road_Type_3	Slight	0.63	0.83	0.99
74	Light_Conditions_4	Slight	0.63	0.85	1.01
73	Light_Conditions_4	Weather_Conditions_1	0.63	0.85	1.04
10	Road_Type_3	Weather_Conditions_1	0.62	0.82	1.01
53	Road_Surface_Conditions_0	Slight	0.58	0.84	1.00
121	Urban_or_Rural_Area_1	Speed_limit_30	0.57	0.87	1.33

Table 4.3. Preliminary Top 10 Association Rules with Full Dataset

Therefore, we further conduct the association rule calculation with only 'Fatal' accidents. Table 4.4 below shows the results.

No.	Premises	Conclusion	Support	Confidence	Lift
43	Fatal	Weather_Conditions_1	0.8451	0.8451	43
80	Weather_Conditions_1	Fatal	0.8451	1.0000	80
81	Road_Type_3	Fatal	0.7827	1.0000	81
82	Road_Surface_Conditions_0	Fatal	0.6700	1.0000	82
44	Road_Type_3	Weather_Conditions_1	0.6656	0.8505	44
45	Road_Type_3	Fatal, Weather_Conditions_1	0.6656	0.8505	45
46	Fatal, Road_Type_3	Weather_Conditions_1	0.6656	0.8505	46
93	Weather_Conditions_1, Road_Type_3	Fatal	0.6656	1.0000	93
74	Road_Surface_Conditions_0	Weather_Conditions_1	0.6580	0.9820	74

Table 4.4.Top 10 Association Rules with 'Fatal' Accidents

By comparing with the previous results, we can further come up with more specific pattern related to 'fatal' accidents. Details are shown in Table 4.5.

No.	Premises	Conclusion	Support	Confidence	Lift
52	Road_Type_3	Fatal	0.7827	1.0000	1.0000
53	Urban_or_Rural_Area_2	Fatal	0.6244	1.0000	1.0000
54	1st_Road_Class_3	Fatal	0.5578	1.0000	1.0000
71	Road_Type_3, Urban_or_Rural_Area_2	Fatal	0.4844	1.0000	1.0000
55	Number_of_Vehicles_1	Fatal	0.4573	1.0000	1.0000
72	Road_Type_3, 1st_Road_Class_3	Fatal	0.4130	1.0000	1.0000
56	Number_of_Vehicles_2	Fatal	0.4039	1.0000	1.0000
57	Year_2012	Fatal	0.3848	1.0000	1.0000
58	Urban_or_Rural_Area_1	Fatal	0.3756	1.0000	1.0000

84	Urban_or_Rural_Area_2, 1st_Road_Class_3	Fatal	0.3748	1.0000	1.0000
59	Speed_limit_30	Fatal	0.3596	1.0000	1.0000
60	Speed_limit_60	Fatal	0.3565	1.0000	1.0000
73	Road_Type_3, Number_of_Vehicles_1	Fatal	0.3533	1.0000	1.0000
41	Speed_limit_60	Fatal, Urban_or_Rural_Area_2	0.3494	0.9802	1.5698
88	Urban_or_Rural_Area_2, Speed_limit_60	Fatal	0.3494	1.0000	1.0000
35	Speed_limit_60	Fatal, Road_Type_3	0.3420	0.9593	1.2256
78	Road_Type_3, Speed_limit_60	Fatal	0.3420	1.0000	1.0000
29	Speed_limit_60	Fatal, Road_Type_3, Urban_or_Rural_Area_2	0.3370	0.9454	1.9515
38	Urban_or_Rural_Area_2, Speed_limit_60	Fatal, Road_Type_3	0.3370	0.9645	1.2322
47	Road_Type_3, Speed_limit_60	Fatal, Urban_or_Rural_Area_2	0.3370	0.9855	1.5783
98	Road_Type_3, Urban_or_Rural_Area_2, Speed_limit_60	Fatal	0.3370	1.0000	1.0000
61	Latitude_range2 [50.980 - 52.047]	Fatal	0.3317	1.0000	1.0000
20	Speed_limit_30	Fatal, Road_Type_3	0.3267	0.9087	1.1609
77	Road_Type_3, Speed_limit_30	Fatal	0.3267	1.0000	1.0000
74	Road_Type_3, Number_of_Vehicles_2	Fatal	0.3206	1.0000	1.0000
62	Year_2014	Fatal	0.3124	1.0000	1.0000
63	Road_Surface_Conditions_4	Fatal	0.3086	1.0000	1.0000
64	Year_2013	Fatal	0.3028	1.0000	1.0000
75	Road_Type_3, Year_2012	Fatal	0.3023	1.0000	1.0000

Table 4.5. Specific Association Rules Towards ‘Fatal’ Accidents

According to the result, some association rules can be intuitively understood. For instance, single items supporting ‘fatal’ may indicate a higher fatality rate if accidents happened under scenarios such as dual carriageway (*No. 52*), rural area (*No. 53*), radial roads coming out of regional centre (road class: A) (*No. 54*), speed limit over 60 (*No. 60*) and frost or ice road surface (*No. 63*). Rule *No.71* implies a correlation

between a dual carriageway and rural area in the fatal accidents. Rule *No. 72* indicates a potential higher fatality rate in the dual carriageway of a radial routes.

References

1. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321-357.
2. Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, No. 34, pp. 226-231).
3. Wright, R. E. (1995). Logistic regression.
4. Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R news*, *2*(3), 18-22.
5. Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.
6. Albon. Chris. (2017, December). Feature Selection Using Random Forest [Blog]. https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_forest/
7. Downes. Natasha. (2018, August). Half of London car crashes take place in 5% of the city's junctions. [Blog]. https://www.ucl.ac.uk/news/2018/aug/half-london-car-crashes-take-place-5-citys-junctions
8. Chawla N V , Bowyer K W , Hall L O , et al. SMOTE: Synthetic Minority Over-sampling Technique[J]. Journal of Artificial Intelligence Research, 2002, 16(1):321-357.
9. Dong-Sheng Cao, Yi-Zeng Liang, Qing-Song Xu, etc. Feature importance sampling-based adaptive random forest as a useful tool to screen underlying lead compounds[J]. Journal of Chemometrics, 2011, 25(4):201-207.
10. Chen T, Tong H, Benesty M. xgboost: Extreme Gradient Boosting[J]. 2016.

Appendices

Appendix 1: Script: Data partition

Note:

- Following Script is run to subdivide the given files into smaller blocks of files with Longitude and Latitude columns only
- Dependencies: pandas

```
import pandas as pd

dataread=pd.read_csv(r"C:/Users/ayadi/OneDrive/Desktop/2000-16-traffic-f
low-england-scotland-wales/accidents_2012_to_2014.csv",usecols =
['Longitude','Latitude'])
filename = 'block'

#for subdivision of file, if block file is large
#filename = 'block_6_1'
#dataread=pd.read_csv(r"C:/Users/ayadi/2000-16-traffic-flow-england-scot
land-wales/blocks/"+filename+".csv",usecols = ['Longitude','Latitude'])

#To get the maximum and minimum values of longitude and latitude
def getBoundaryPoints(data):
    return
data.Longitude.min(),data.Longitude.max(),data.Latitude.min(),
data.Latitude.max()

minLongitude,maxLongitude,minLatitude,maxLatitude =
getBoundaryPoints(dataread)

#number of data blocks subdivided from input file =
numberOfSquares*numberOfSquares
numberOfSquares = 2
longitude_range = maxLongitude-minLongitude
latitude_range = maxLatitude - minLatitude
blocksize_long = longitude_range/numberOfSquares
blocksize_lat = latitude_range/numberOfSquares

print('--------')
#this method divides the input files into smaller blocks of data based
on blocksize_long and blocksize_lat and writes them to different files
def dataPartioning(data):
    for i in range(numberOfSquares):
        startLongitude = minLongitude + i*blocksize_long
```

```
        endLongitude = minLongitude + (i+1)*blocksize_long
        #initialise latitude here
        for j in range(numberOfSquares):
            print('Counter: ', i, j)
            startLatitude = minLatitude + j*blocksize_lat
            endLatitude = minLatitude + (j+1)*blocksize_lat
            block = data.loc[(data['Longitude']>= startLongitude) &
(data['Longitude'] < endLongitude) & (data['Latitude'] >= startLatitude)
& (data['Latitude'] < endLatitude)]
            block.to_csv('block_'+str(i)+'_'+str(j)+'.csv',sep=',',
index=False)
            #subdivision
            #block.to_csv(filename+'_'+str(i)+'_'+str(j)+'.csv',sep=',',
index=False)

dataPartioning(dataread)
print('--Completed--')
```

Appendix 2: Finding Hotspots

Note:

- Following Script is run to pass accident locations to dbscan to find clusters and finally plot them in google map
- Dependencies: pandas, sklearn, os, gmaps, numpy
- Gmaps module will be executed from Jupyter notebook only.

```
import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
import os
import gmaps

gmaps.configure(api_key='AIzaSyAsnB1GH9bSBUq7w4KO5CDuaJBi8io8X7A')
filepath=
r"C:\Users\ayadi\2000-16-traffic-flow-england-scotland-wales\Finding
Hotspots\blocks"
col_names = ['Latitude','Longitude']
locations=pd.DataFrame(columns = col_names)
files = []

#we are passing the path for individual block files to files[]
def findFiles(filepath):
    for filename in os.listdir(filepath):
        if filename.endswith(".csv"):
            files.append(os.path.join(filepath, filename))
            continue
        else:
            continue

def dbscan():
    global locations
    print('--DBSCAN Method Started--')
    for file in files:
        dataread = pd.read_csv(file, usecols = ['Longitude','Latitude'],
error_bad_lines = False)
        print('Reading:', file, '- empty: ',dataread.empty)
        #dataread.empty = true means the file is empty
        if dataread.empty:
```

```
            continue
        #if file has data, it is passed to DBSCAN
        db = DBSCAN(eps=0.1, min_samples=80).fit(dataread)
        core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = db.labels_
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        n_noise_ = list(labels).count(-1)
        unique_labels = set(labels)
        for k in unique_labels:
            if k == -1:
                continue
            #if k == -1, continue referes to disregard of noise points
for plotting
            class_member_mask = (labels == k)
            #if datapoints are part of cluster they are pushed to
locations dataframe for plotting
            xy = np.array(dataread[class_member_mask &
core_samples_mask])
            temp=pd.DataFrame({'Latitude':xy[:,1], 'Longitude':xy[:,0]})
            locations=locations.append(temp)
            xy = np.array(dataread[class_member_mask &
~core_samples_mask])
            temp=pd.DataFrame({'Latitude':xy[:,1], 'Longitude':xy[:,0]})
            locations=locations.append(temp)

findFiles(filepath)
if len(files) == 0:
    print('No files in the given directory')
dbscan()
#google map is initialised
london_coordinates = (51.50, 0.12)
gmaps.figure(center=london_coordinates, zoom_level=10)
gmaps.figure(map_type='HYBRID')
figure = gmaps.figure()
print('---')
heatmap_layer = gmaps.heatmap_layer(locations)
figure.add_layer(heatmap_layer)
figure
```

Appendix 3: Classification of Accidents as Fatal/Non-Fatal and investigating the factors

```
import numpy as np
import pandas as pd

#Visualisation Libraries
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import warnings
warnings.filterwarnings('ignore')
from pandas.plotting import scatter_matrix
plt.style.use('ggplot')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'retina'")

#Training and Preprocessing Libraries
from xgboost import XGBClassifier
from imblearn.ensemble import EasyEnsembleClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score
import datetime as dt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.feature_selection import SelectFromModel
from numpy import loadtxt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve

class_names = ['Fatal', 'Severe', 'Slight']

data =
pd.read_csv("./2000-16-traffic-flow-england-scotland-wales/accidents_2012_to_201
4.csv")

data['Did_Police_Officer_Attend_Scene_of_Accident'].head()

def preprocessing(data):
    #Drop useless columns and nan values
```

```
    data.drop(['Junction_Detail',
               'Junction_Control',
               'Accident_Index',
               'Date',
               'Special_Conditions_at_Site',
               'Carriageway_Hazards',
               'Did_Police_Officer_Attend_Scene_of_Accident',
               'LSOA_of_Accident_Location',
               'Local_Authority_(District)',
               'Local_Authority_(Highway)','Number_of_Casualties'], axis=1,
inplace=True)
    data.dropna(inplace=True)

    #Drop rows with 'Unknown' values
    data = data[data.Weather_Conditions!='Unknown']
    data = data[data.Road_Type!='Unknown']

    #Encode "String" Labels into "Int" Labels for easy training
    le = LabelEncoder()
    data["Pedestrian_Crossing-Physical_Facilities"]=
le.fit_transform(data["Pedestrian_Crossing-Physical_Facilities"])
    data["Light_Conditions"]= le.fit_transform(data["Light_Conditions"])
    data["Weather_Conditions"] = le.fit_transform(data["Weather_Conditions"])
    data["Road_Surface_Conditions"] =
le.fit_transform(data["Road_Surface_Conditions"])
    data["Pedestrian_Crossing-Human_Control"] =
le.fit_transform(data["Pedestrian_Crossing-Human_Control"])
    data["Road_Type"] = le.fit_transform(data["Road_Type"])

    #Converting Time into Int for easy training
    data["Time"]= data["Time"].astype(str)
    data['Time']=data['Time'].str.slice(0,2, 1)
    data["Time"]= data["Time"].astype(int)

    #Creating 3 additional columns, one each for each class we need to classify
into
    onehot = pd.get_dummies(data.Accident_Severity,prefix=['Severity'])
    data["Fatal"] = onehot["['Severity']_1"]
    data["Severe"] = onehot["['Severity']_2"]
    data["Slight"] = onehot["['Severity']_3"]

    return (data)

data = preprocessing(data)

data.head()

X = [X for X in data if X not in
['Accident_Severity','Severe','Slight','Fatal']]
```

```
X_train, X_test, y_train, y_test = train_test_split(data[X], data['Fatal'],
test_size=0.2, random_state = 400)

# logistic Regression model and accuracy
def logistic_regression(X_train, y_train, X_test, y_test):
    lr = LogisticRegression()
    lr.fit(np.array(X_train), np.array(y_train))
    y_pred = lr.predict(np.array(X_test))
    sk_report = metrics.classification_report(
        digits=6,
        y_true=y_test,
        y_pred=y_pred)
    print("Accuracy", round(metrics.accuracy_score(y_pred, y_test)*100,2))
    print(sk_report)
    print(pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'], margins=True))
    #Plot Precision-recall curve
    disp = plot_precision_recall_curve(lr, X_test, y_test)
    disp.ax_.set_title("Precision recall curve for Logistic Regression")

# random_forest model and accuracy
def random_forest_classifier(X_train, y_train, X_test, y_test):
    random_forest = RandomForestClassifier(n_estimators=20)
    random_forest.fit(np.array(X_train),np.array(y_train))
    Y_pred = random_forest.predict(np.array(X_test))
    sk_report = metrics.classification_report(
        digits=4,
        y_true=y_test,
        y_pred=Y_pred)
    print(sk_report)
    print(pd.crosstab(y_test, Y_pred, rownames=['Actual'],
colnames=['Predicted'], margins=True))
    #Plot Precision-recall curve
    disp = plot_precision_recall_curve(random_forest, X_test, y_test)
    disp.ax_.set_title("Precision recall curve for Random Forest")
    #Plot feature importances
    plt.figure(figsize=(15,8))
    plt.title("Top features for predicting fatality")
    plt.xlabel("Confidence")
    feat_importances = pd.Series(random_forest.feature_importances_,
index=X_train.columns)
    feat_importances.nlargest(40).plot(kind='barh')

# XGBoost model
def xgboost(X_train, y_train, X_test, y_test):
    xgb = XGBClassifier(
                            learning_rate =0.05,
                             n_estimators=100,
                             max_depth=3,
                             min_child_weight=1,
```

```
                            gamma=0.3,
                            subsample=0.8,
                            colsample_bytree=0.8,
                            objective= 'multi:softprob',
                            nthread=4,
                            scale_pos_weight=1,
                            num_class=2,
                            seed=27
                        ).fit(np.array(X_resampled_train),
np.array(y_resampled_train))
    Y_pred = xgb.predict(np.array(X_test))
    sk_report = metrics.classification_report(
        digits=4,
        y_true=y_test,
        y_pred=Y_pred)
    print(sk_report)
    print(pd.crosstab(y_test, Y_pred, rownames=['Actual'],
colnames=['Predicted'], margins=True))
    #Plot Precision-recall curve
    disp = plot_precision_recall_curve(xgb, np.array(X_test), np.array(y_test))
    disp.ax_.set_title("Precision recall curve for XGBOOST")

#SMOTE
from imblearn.over_sampling import SMOTE
oversampler=SMOTE(random_state=0)
X_resampled_train, y_resampled_train = oversampler.fit_sample(X_train, y_train)

#Count before SMOTE
y_train.value_counts()

#Count after SMOTE
pd.Series(y_resampled_train).value_counts()

# random forest after SMOTE
random_forest_classifier(pd.DataFrame(X_resampled_train,
columns=X_train.columns), y_resampled_train, X_test, y_test)

#xgboost after SMOTE
xgboost(X_resampled_train, y_resampled_train, X_test, y_test)

#Logistic regression after SMOTE
logistic_regression(X_resampled_train, y_resampled_train, X_test, y_test)


# k-fold cross validation evaluation of logistic Regression model
# CV model
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(lr, X_test, y_test, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
# k-fold cross validation evaluation of random_forest model
# CV model
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(random_forest, X_test, y_test, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))

# k-fold cross validation evaluation of xgboost model
# CV model
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(xgb, X_test, y_test, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```