CAR ACCIDENT ANALYSIS IN THE UK

Data mining project

Team members:

Andrea Armani
Ricardo Holthausen Bermejo
Dimitris Tsesmelis

Prof:

Mahmoud Sakr

Introduction

In this report, the details regarding the implementation and the results obtained for the Data Mining course project "Car accident analysis in the UK" are provided. Given a dataset containing near half a million records of accidents that occurred between 2012 and 2014 in the UK, the goal was to obtain insights regarding three areas:

1) Identifying the areas in which large number of accidents occur
2) Discover the causes related to fatal accidents
3) Obtain a model capable of predict whether a set of conditions would lead to an accident

Thus, in the same order as those areas are mentioned, they are presented in this document, detailing the process, what decisions were made and why, what were the outcomes as well ass their meaning.

Regarding the code developed for the present project, it is available in three different Google Colab notebooks in this link

Identifying hotspots

In order to obtain the hotspots for car accidents (i.e.: those places where a considerable amount of accidents occur), our approach consisted in using a density-based clustering algorithm on the longitude and latitude of the accidents. The reason for using a density-based algorithm is that we actually want to take into account only small places in which a large amount of accidents happened; using for example DBSCAN with a small epsilon would provide us with the former, and a large tau (min points) would provide us with the latter.

When trying to use DBSCAN on a large number of points (in this case, almost 500000), we can face problems of performance or an excessive use of memory (resulting in Memory Errors in Python). These problems were avoided by changing the datatype of the latitude and longitude coordinates in our data frames from 64 bits floats to 32 bits floats.

The results obtained from clustering with the whole datasets are shown in figures N1 and N2. As can be seen, the hotspots are mainly located in cities, as any kind of accident is more prone to occur in places where more people are (and therefore use vehicles).

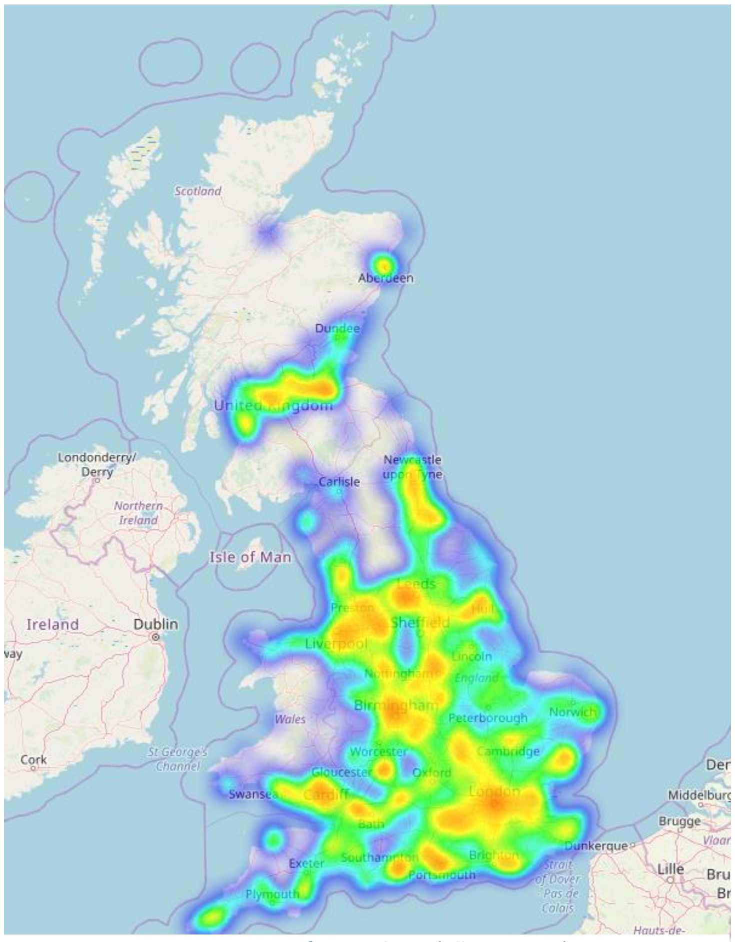

Figure N1 Heatmap for accidents (all severities) in UK

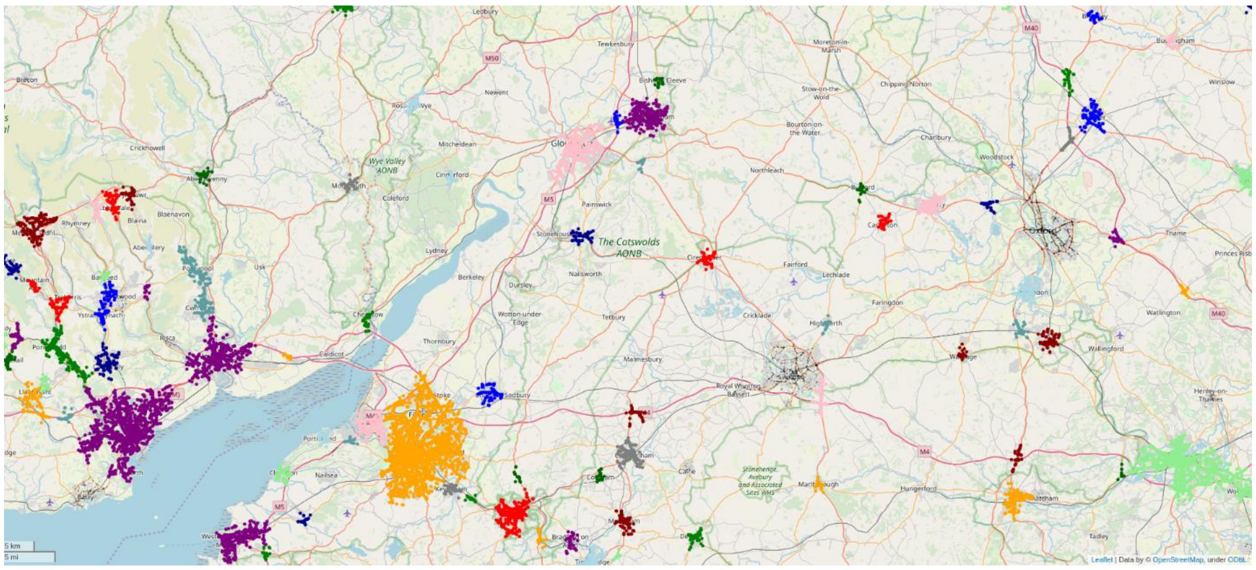
Figure N2 Clusters obtained using the whole dataset

Fatal accidents hotspots can be more interesting to be studied, as it can help to improve the road conditions on those spots or to conduct more specific studies in order to reduce the number of casualties in the most dangerous places. For this reason, we conducted another clustering in which only fatal accidents were taken into account. Besides, in order to give more importance to accidents in which more casualties occur, the points were multiplied by the number of casualties in the accidents, so that a small amount of accidents whose total number of casualties were larger than the min_pts parameter would be considered as a hotspot. Two different sets of parameters were used:

- Epsilon = 0.01 and min_pts = 50
- Epsilon = 0.005 and min_pts = 25

In the first case, clusters in most of the cities in the UK were found (Figures N3 and N4). By the results obtained it can be seen that there are cities which are safer than others (v.g.: Cardiff does not have a cluster of fatal accidents).

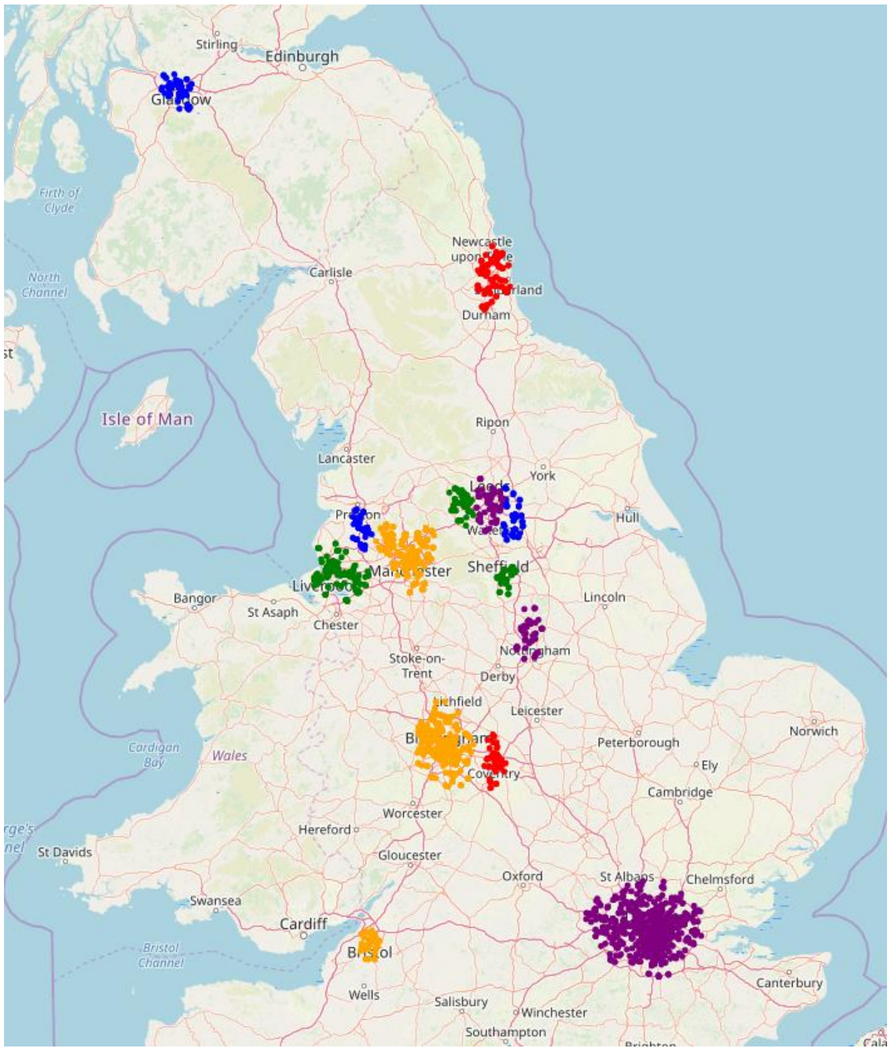
Figure N3 Hotspots found located in UK cities

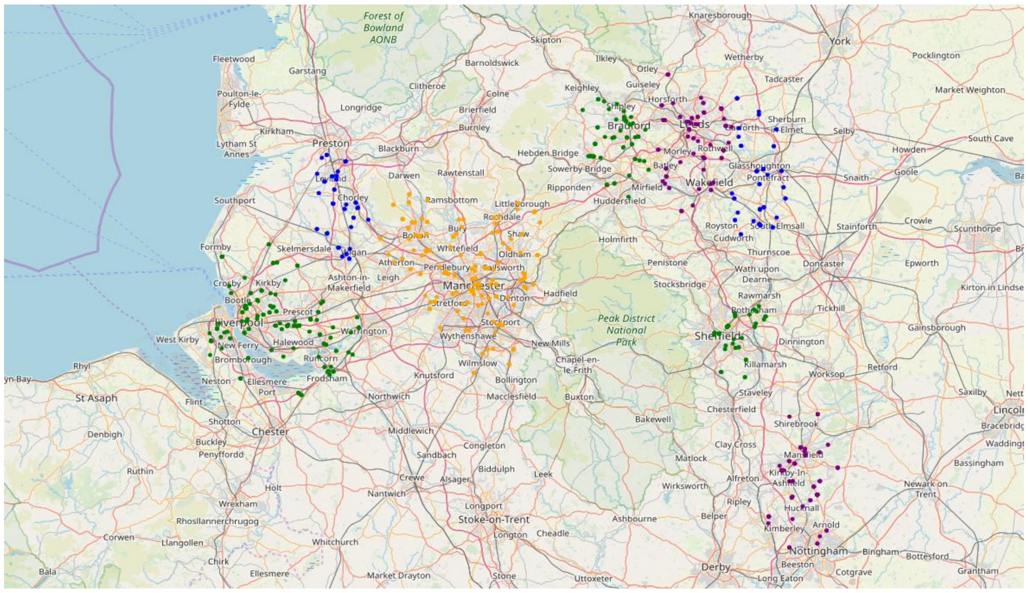
Figure N4 Hotspots found located in UK cities (detail)

In the second case, besides clusters in UK cities, we can also find certain roads which are outstandingly dangerous, having a handful of fatal accidents spread in a reduced number of kilometers (Figures N5, N6 and N7).

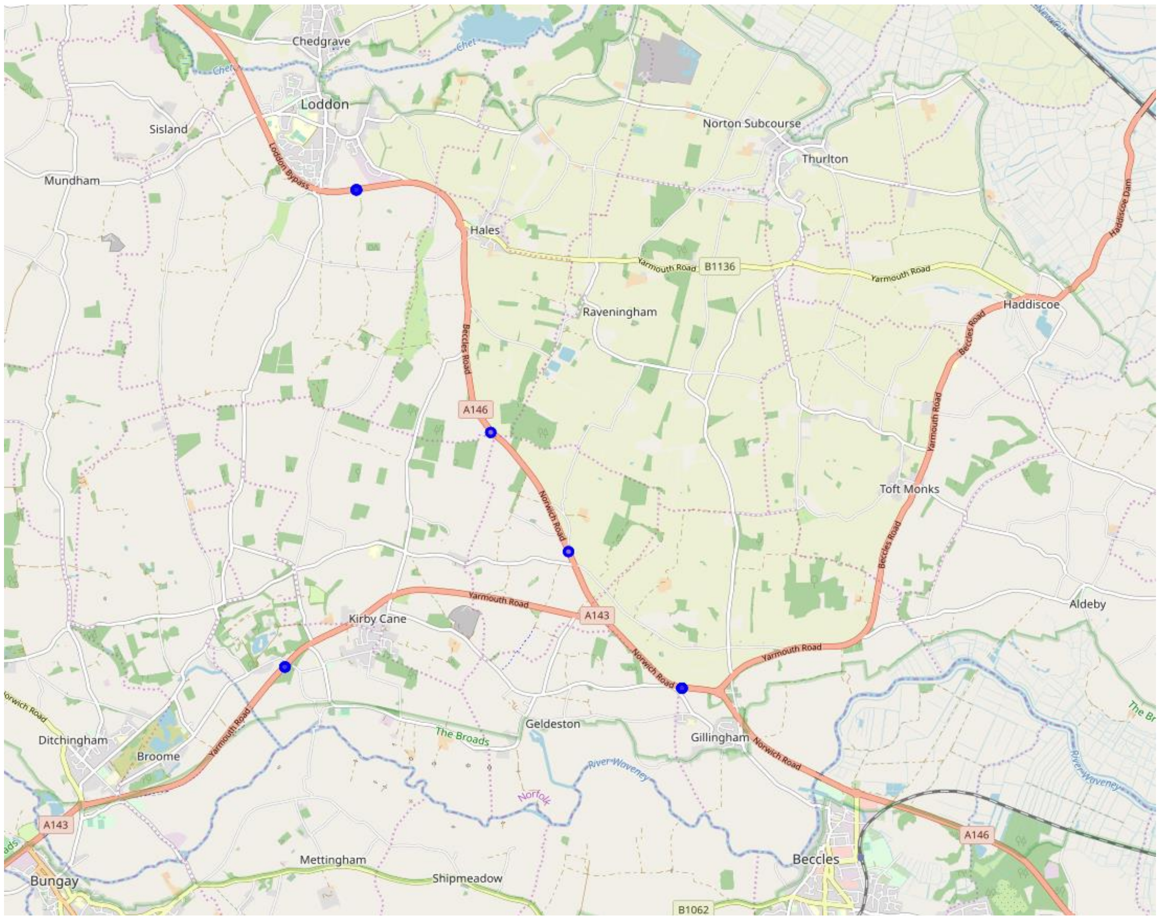

Figure N5: Hotspot found in a secondary road where 5 fatal accidents occurred

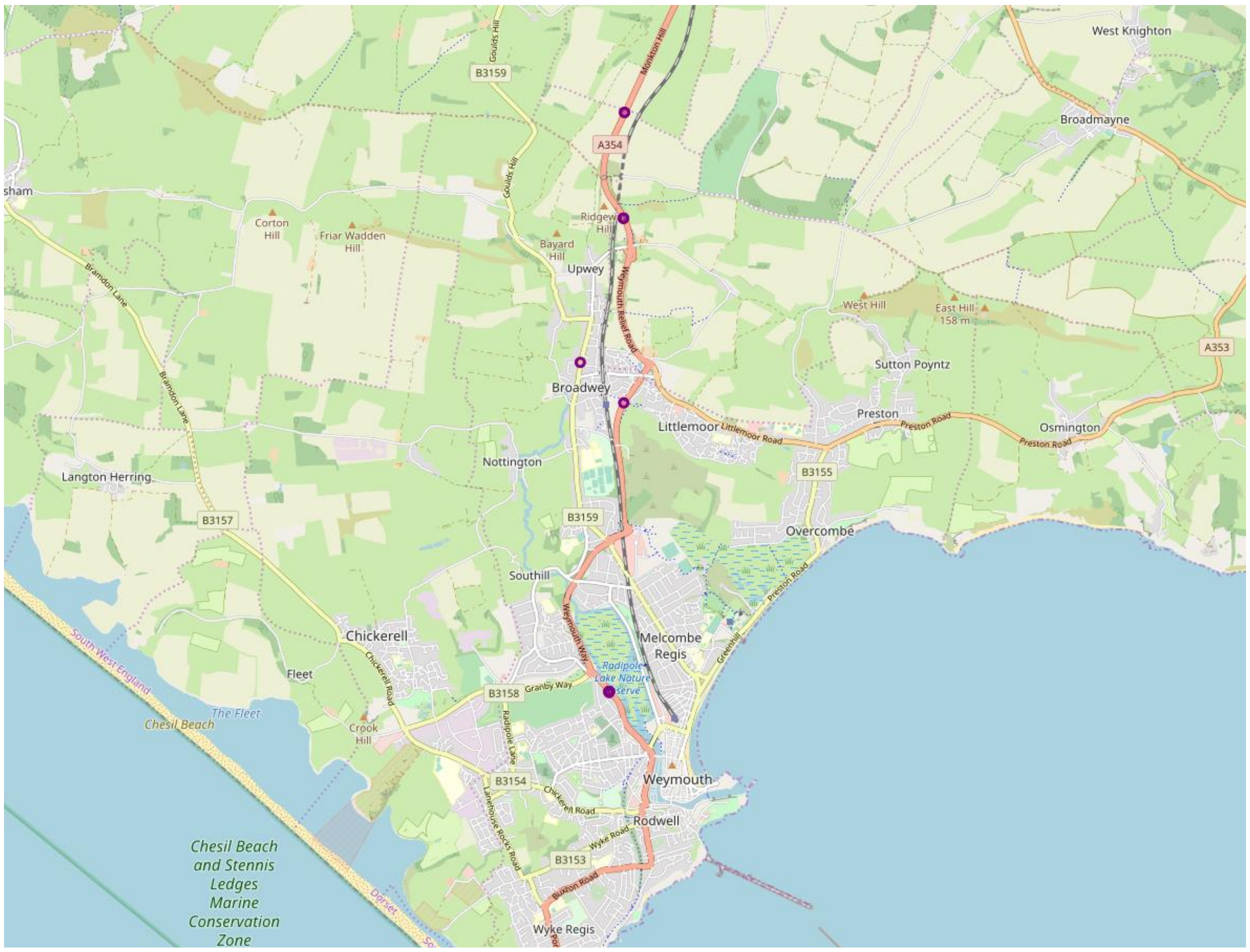

Figure N6: Hotspot found in a secondary road where 5 fatal accidents occurred

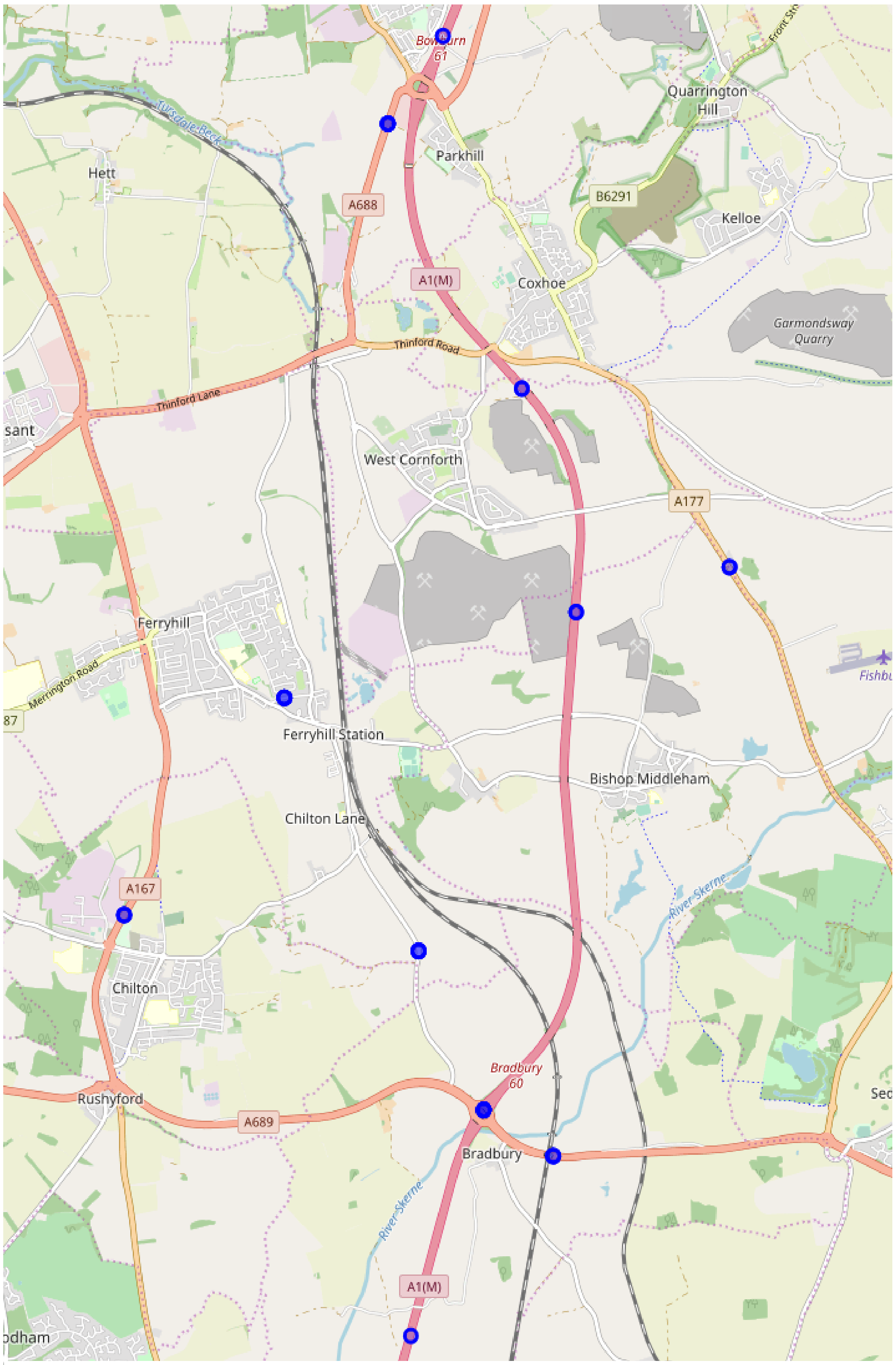

Figure N7: Hotspot found in road where 11 fatal accidents occurred

Data preprocessing

During the data preprocessing phase, we made several decisions to correctly input data to our model.

Dimensionality reduction

We initially approach the dataset removing JUNCTION_DETAIL because it has the same value replicated in all the rows.

Secondly, we produced the Correlation matrix to check if there were column highly correlated. What emerged was that:

- Location_Northing_OSGR has a correlation of 0.99 with Latitude
- Location_Easting_OSGR has a correlation of 0.99 with Longitude
- Police_Force has a correlation of 0.98 with Local_Authority_(District)

Since high correlated columns aren't valuable for the model, we decided to drop Location_Northing_OSGR, Location_Easting_OSGR and Police_Force.

We also drop the index of the accidents (Accident_Index) because it doesn't hold valuable information.

Furthermore, the attributes Special_Conditions_at_Site, Carriageway_Hazards and Pedestrian_Crossing-Human_Control have more than 90% of the data with the same value, so we decided to inspect them. We based our choice on informations found at:

- https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales#accidents_2005_to_2007.csv
- http://data.dft.gov.uk/gb-traffic-matrix/all-traffic-data-metadata.pdf

After the analysis of these resources we finally decided to drop these columns.

Lastly, we noticed that almost 40% of the rows in the column Junction_Control are null values. So we thought three different ways to deal with this attribute; Since the most observed category was 'UNCONTROLLED ' we could have used this value to impute the missing rows because we thought that maybe a Nan value could mean that there weren't controlled area. The problem is that we cannot be sure that this is the case and if we were wrong, we would have introduced a lot of bias inside the model. The second approach that we thought to use is to impute the missing values using KNN algorithm, but also this solution seemed difficult to realize since we have no information to properly tune the number of neighbours for every cluster. The third and last approach is the most trivial: we simply decided to drop the column.

Multiple features in the initial dataset refer to spatial data. Also, Latitude and Longitude have numeric values but actually are categorical attributes. The problem is that considering them as category would lead us to deal with more than 380k categories for each dimension. So, instead of adding redundant information, we used Latitude and Longitude to create a geohash index.

Not only this technique allowed us to significantly reduce the number of variables, it was useful also to merge the localization of accident that occurred in the same spot but have slightly different coordinates. Geohash requires a numeric parameter that specify the precision of the function: increasing the number lead to merge together point that are closer. Due to hardware limitations, our analysis used 50 different geohash values.

In our model we are not taking in consideration the date in which an accident happened, we just store the day of the week.

The very last part of the removing was to delete 13 rows that hold null values in an attribute. We could also have imputed these values, but to delete 13 rows out of half a million it's an operation that preserve the value of the dataset.

Feature extraction

The dataset presents a lot of categorical features; we decided to deal with such categories encoding them using the One-Hot-Encoding algorithm. We faced a problem related to the dimension of our dataset: preserving all the different value for each column would have led us to obtain a dataset with a very high number of columns and such a high dimensionality wouldn't be suitable for our analysis. So, we invested time in transforming the feature's values in order to achieve better matrix dimensions. The columns affected by this step were:

FEATURES	ORIGINAL NUMBER OF VALUES	TRANSFORMED NUMBER OF VALUES
Number_of_Vehicles	18	3
Road_Type	6	3
Pedestrian_Crossing-Physical_Facilities	6	2
Weather_Conditions	9	3
Road_Surface_Conditions	5	3
Number_of_Casualties	37	3
Light_Conditions	5	3
Speed_limit	7	6
Time	1439	4

Feature Conversion

The dataset presents some columns that should be categories but are represented by numeric values. If we don’t handle these exceptions, the column would preserve an order between the categories and since there is not a meaningful order, we needed to modify them and an efficient way to achieve this is to cast the column to string.

Feature Transformation – One-Hot-Encoding

Now that we have finally decide the features that we will use for train the model, we have to represent them in a correct way and we have two different choices, mapping each category to a number or add dimension to the table to equally distribute the features into the n-dimensional space. We used the second approach because the first one presuppose a natural order between the features, and for the most of them this is not the case. All the assumptions made in the Feature Extraction step are crucial for modelling a bigger dimensional data frame.

Detecting patterns that lead to Fatal accidents using Association rules

To detect pattern and correlation to the features that often lead to fatal accident, we used association rules. The goal of this experiment was to obtain features or combination of features that imply fatal accidents. Because of the size of the given dataset, we used the FP Growth algorithm which is faster than the simple Apriori algorithm.

Data preprocessing:
Before applying the association rules algorithm, we had to preprocess the dataset. Namely, we applied the transformations that are mentioned in the chapter of Data preprocessing as well as the following:

1. We did not apply the one hot encoding as we do not want our rows to have values in the range of (0, 1).
2. We transformed the features that contained value like 1, 2, 3… to [name_of_the_feature]_value (ex: for speed_limit, value 20 was transformed to speed_limit_20). This changes was vital, as value of different attributes were the same for the association rule algorithm (ex: value 1 of Day_of_Week and value 1 of Urban_or_Rural_Area).

Our approach:
The severity of the accident in our dataset is split in 3 different categories, namely Fatal, Serious and Slight. The number of rows of Fatal accident are approximately 5000, while Serious are 67000 and Slight are 393000.
Our first approach was to try to find pattern in the whole dataset. However, this method did not produce any rules that implied Fatal accident, as the number of rows that refer to this type of accident was only 1.5%.
Our second thought was to use every row of Fatal accidents and a subset of the rest of the rows in order to increase the percentage of the first. By picking randomly 2500 rows of Serious and 2500 rows of Slight accidents (10000 rows in total), we achieved to get rules that implied Fatal one. However, to ensure that we did not get some rules that were valid only to this sub-dataset, we run this experiment 50 times in total with different sub-datasets and we finally chose only the rules that occurred to more than 45 runs.

The rules and the confidence we mined are the following:

1. ('Fine without high winds', 'local_auth_5') **0.98**
2. ('Single carriageway', 'local_auth_9') **0.99**
3. ('1st_road_class_3', 'local_auth_9') **0.99**
4. ('No physical crossing within 50 meters', 'local_auth_9') **0.99**
5. ('Fine without high winds', 'No physical crossing within 50 meters', 'local_auth_9') **0.99**

6. ('Fine without high winds', 'No physical crossing within 50 meters', 'Rural', 'Single carriageway', 'speed_60') **0.71**
7. ('Daylight: Street light present', 'Dry', 'Fine without high winds', 'No physical crossing within 50 meters', 'Rural', 'Single carriageway', 'speed_60') **0.71**
8. ('1st_road_class_3', 'Fine without high winds', 'No physical crossing within 50 meters', 'Rural', 'Single carriageway', 'speed_60') **0.72**
9. ('Darkness: No street lighting', 'No physical crossing within 50 meters', 'Rural') **0.76**

Notes:

- 'local_auth_5' is the value for the local authority of Tower Hamlets.
- 'local_auth_9' is the value for the local authority of Lambeth.
- '1st_road_class_3' is the value for road class of type A (the available values are Motorway, A(M), A, B, C and Unclassified).

Creating a model that predicts Fatal accidents

We tried to apply many algorithms and techniques in order to predict whether a fatal accident is going to happen by using the given traffic conditions. The most meaningful and useful one was the classification. However, in this chapter we will discuss also the other attempts and some reasons for which they were not efficient.

Classification with down sampling

We experimented with many classifiers as well as many different techniques to tackle the problem of imbalance of the classes, as we discussed in the chapter of association rules. We concluded that we took the best results by down sampling the dataset and by using Random forest and SVMs classifiers. Regarding the SVM, we used the SGDClassifier from sklearn library which is a classic linear classifier with stochastic gradient descent (SGD) training.
In the next two tables we are comparing the results of both classifiers in two datasets. The first one (**dataset 1**) is generated by using 3 different types of accidents (Slight, Serious and Fatal) while the second one (**dataset 2**) is generated by using 2 types (Slight and Serious/Fatal). In the second case, we merged the Serious and the Fatal accidents because we assumed that both categories are referring to accidents that a government would like to reduce. This means that if an accident is Serious is “closer” to a Fatal, as people got seriously injured (otherwise it would be a Slight one) and that is why we are considering them as being the same class.

The dataset 1 contains approximately 10.000 points and the dataset 2 contains 150.000.

RESULTS FOR DATASET 1

METRIC	Stochastic SVM	Randomforest
PRECISION	47.49%	57.48%
RECALL	80.26%	67.59%
F-MEASURE	59.63%	62.13%
ACCURACY	48.98%	53.17%

RESULTS FOR DATASET 2

METRIC	Stochastic SVM	Randomforest
PRECISION	60.57%	56.46%
RECALL	66.53%	65.52%
F-MEASURE	63.07%	60.65%
ACCURACY	61.36%	52.82%

From the above results, we can observe that we generally get a low Accuracy, a high Recall and medium Precision and F-Measure. These results are reasonable, as the classes that our algorithms are predicting are not of the same importance. This means that if our predictions are

not correct regarding a Slight accident it is not the same as being wrong for a Fatal one. That is why we do not take into consideration the Accuracy.
On the other hand, Recall is a very important measure for us. When we have high Recall, taking into consideration that the positive class in the above tables is the Fatal accidents, it means that we do not have a lot of False-Positives (FP). In our case, False-Positives are the accidents that are Fatal, and we predict them as Slight or Serious. We strongly want to avoid such mistakes, as we want to prevent Fatal accidents (high Recall), but we can tolerate to have more True-Negative, which means that we predict some Slight or Serious Accident as Fatal (medium Precision).

Finding the best hyper parameters
To find the hyper parameters that give the best performance to our models, we used a Grid Search method. This algorithm is basically exhaustively search over specified parameter values to find the best performance of an estimator. Finally, the best hyper parameter found by the Grid Search were used to our experiments.

Classification with over sampling
Another thing that we tried to overcome the problem of the imbalance of the classes was the oversampling of the dataset. More precisely, we added more Fatal accidents by using 2 techniques.

1. Firstly, we used **RandomOverSampler** from imblearn.over_sampling library that generates more samples of the minority class by randomly sampling with replacement the current available samples. This naive strategy did not give good results as it is just adding duplicate points to the dataset.
2. Secondly, we used **Synthetic Minority Over-sampling Technique (SMOTE)** from the same library. It creates synthetic (not duplicate) samples of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighbouring records.

Class_weighted parameter
Both RandomForest and SGDClassifier have the parameter Class_weighted that is a useful setting where the classes are automatically weighted inversely proportional to how frequently they appear in the data. This technique is like down sampling and over sampling the dataset with the difference that the number of points of the classes is not changed. They are penalizing mistakes on the minority class by an amount proportional to how under-represented it is.

Outlier mining
An approach that we consider modelling the problem is the outlier mining. Our data are skewed in a way that the fatal accidents are just the 1% of the total, so we aimed to detect fatal accident as outliers in the imported dataset. The problem that we faced is that the outliers don’t lay in a sufficiently sparse region of our n-dimensional space, therefore the algorithms isn’t able to

correctly mark the accidents as fatal. Adding a higher level of contamination lead the algorithm to perform better but in such a way we are not correctly representing the initial situation. That’s why, ultimately, we decide to abandon this approach.