Data Science project: UK Road Accident Data, 2015

Published on September 12, 2020

Introduction: background and objectives

Reducing accidents on the road represents a major objective of all governments around the world. During 2018 on the UK's roads alone, the number of killed or seriously injured (KSI) amounted to 27,295. Total prevention costs from reported road accidents amounted to 16.5 billion British pounds in 2018. Between 2010 and 2018, figures had grown by approximately 9 per cent, while the lowest figure had been recorded in 2013. On average a fatal accident resulted in nearly 2.2 million British pounds in prevention costs in 2018, while slight accidents still resulted in costs of £26,100.

Gaining insight into how the statistics of KSI can be reduced is therefore of critical importance to the government's aims, in order to improve the safety for UK citizens whilst travelling, as well as for important economic reasons.

This paper seeks to investigate and come to some conclusions regarding data on road accidents, in particular what features of accidents we can link to their severity, and how we can use these features to accurately predict this characteristic. If we can understand some causes or correlations of KSI on the road, we can hopefully reduce the negative statistics. This would allow them to take steps to mitigate and reduce the severity of accidents on the country's roads, thereby improving public health and wellbeing.

Data outline

The data I will analyse was compiled and published by the UK Government's Department for Transport, is sourced at https://www.kaggle.com/akshay4/road-accidents-incidence, and is comprised of data on accidents and a range of characteristics of vehicles concerned. I intend to analyse the data-set's features, and see which may be correlated to our target variable (accident severity). This feature can have discrete integer values 1 ('slight'), 2 ('Severe') or 3 ('Fatal).

This data-set (.csv) reflects Road Accidents in UK during 2015 and has 70 features/columns and about 250K rows. *Italic* used for feature names and inverted commas for feature values.

The data set consists of the following features, which are a mixture of string, integer, and continuous (float) objects pertaining to the vehicle as well as the particular environmental conditions at the time of the accident, including a unique *accident index, vehicle type, engine capacity, weather conditions*, whether a police officer attended the scene of the accident etc.

	accident_index	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_location-restricted_lane	junction_locatic
0	201506E098757	2	9	0	18	0	
1	201506E098766	1	9	0	9	0	
2	201506E098766	2	9	0	18	0	
3	201506E098777	1	20	0	4	0	
4	201506E098780	1	9	0	15	0	
5	201506E098780	2	1	0	9	0	
6	201506E098792	1	3	0	4	0	
7	201506E098792	2	9	0	4	0	
8	201506E098804	1	9	0	14	0	
9	201506E098826	1	9	0	4	0	

10 rows × 70 columns

Methodology

To analyse this dataset I used standard Python libraries Pandas, NumPy and Matplotlib to assess correlations between features and the target variable Accident Severity. Prior to training and testing machine learning models, I looked to clean, change object type, balance, possibly normalise or standardise the data, perform feature-engineering, fill in or delete rows with missing values and so on. I may need to return to the data preparation stage at later points in my analysis. As there are a lot of features, I sought to select the ones which seem most closely correlated with the target variable.

As part of my analysis, I also wanted to understand what trends or skewed information there are in the data. Once I found some useful correlations, I proceeded to train and test different supervised learning models to see if any can accurately predict the target variable values. I will evaluate the models' performance, and fine tune their parameters if necessary.

As the target variable is a categorical and not continuous variable (1, 2 or 3), I tested out a range of classification models (K-Nearest-Neighbours, Logistical Regression etc.), to see which performed best.

Target Variable Balance

Checking that there is a balance of 'severity' data-points. It seems there is a reasonably balanced distribution, but this may be something I need to come back to if the models learn just to predict '3' incorrectly.

```
In [218]: 1 df['accident_severity'].value_counts()

Out[218]: 3    242477
          2     39206
          1      3648
          Name: accident_severity, dtype: int64
```

Missing values

The data shows some points relating to the features were marked as '-1' or 'Unclassified', or '8'/'9' ('other'; 'unknown', in the case of *weather conditions*). I will change these to NaN

(i.e. 'null'), as I want to know how many actual data-points I have before I decide what to do with these missing values in order to be able to plot the data and see which features may have predictive value.

The data indicates all the *imd (***Index of Multiple Deprivation)** features are now entirely composed of 'null' values. This is because the values were all 'unclassified' in the original dataset. Likewise, the *sex of driver* column had possible values of 1,2 or 3 (3 being 'unclassified').

```
In [219]:
df=df.replace([-1, 'Unclassified'], np.nan)
df['weather_conditions']=df['weather_conditions'].replace([8, 9], np.nan)
df['sex_of_driver']=df['sex_of_driver'].replace([3], np.nan)
#Sanity check to see whether this has worked
print(df['weather_conditions'].value_counts())
df['sex_of_driver'].value_counts()

1.0    233216
2.0     31375
4.0      4716
5.0      4627
7.0      1404
3.0      1050
6.0       357
Name: weather_conditions, dtype: int64
```

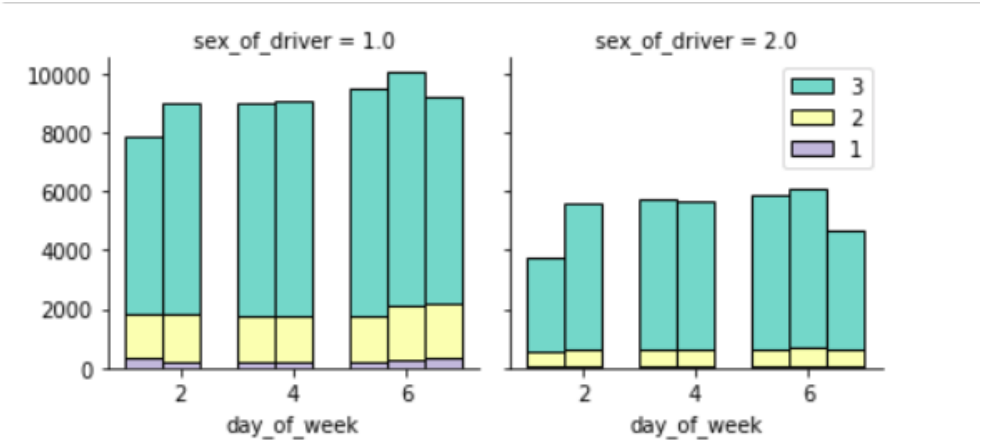
Data Cleaning

I dropped the rows with missing values from the following features, which I wish to plot, as there are quite a lot of missing values.

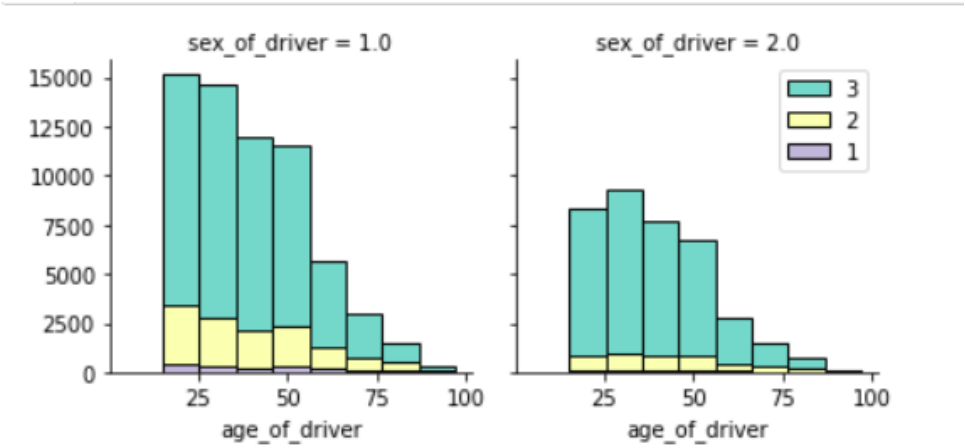
```
In [223]:
df.dropna(subset = ['road_surface_conditions', 'urban_or_rural_area', 'car_passenger', '
df.shape

Out[223]: (120808, 70)
```

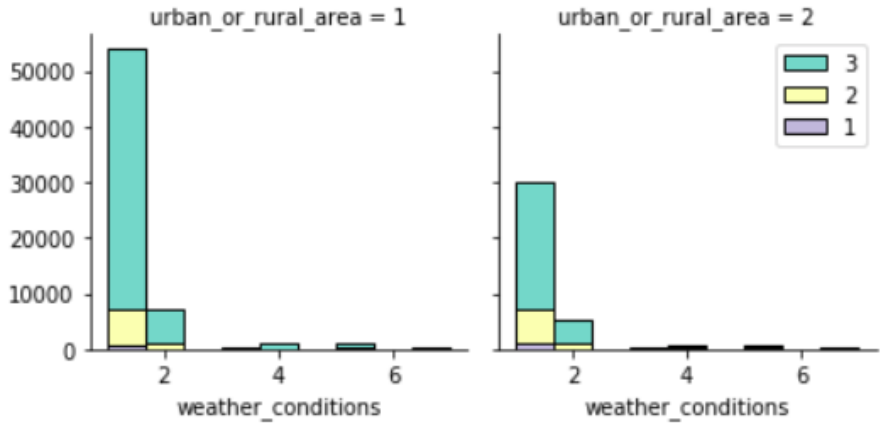
Plotting data to see correlations



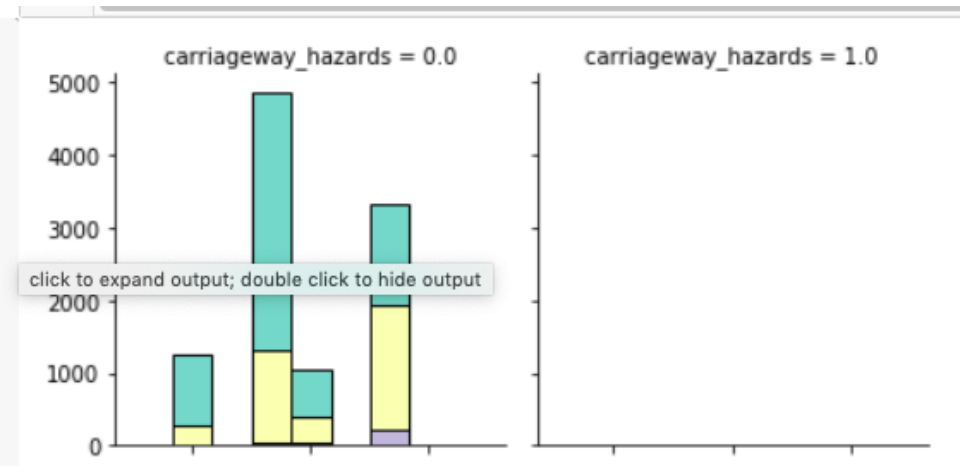
Here we can see the greater overall proportion of drivers involved in accidents is male, as well as in serious and fatal ones. The spread over days of the week is quite even. Male = 1.0; Female = 2.0; day 6=Saturday.



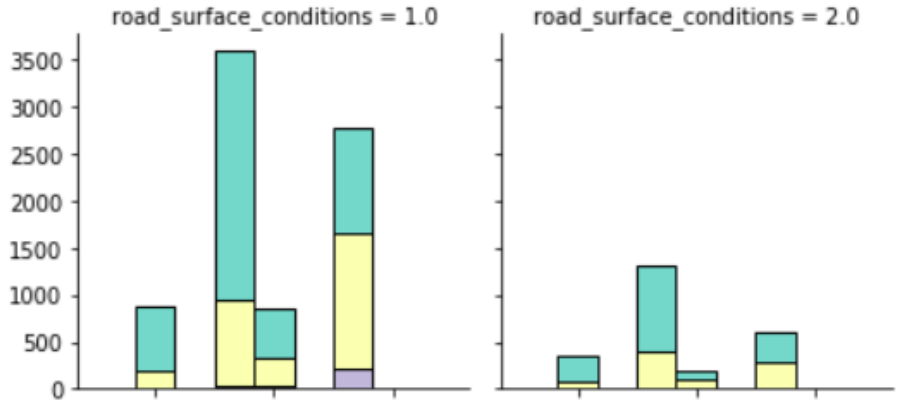
Drivers around 25 represent the peak age for likelihood of being involved in all categories of road accidents, with the data gradually declining from that point. The proportion of severe accidents also seems to follow this trend.



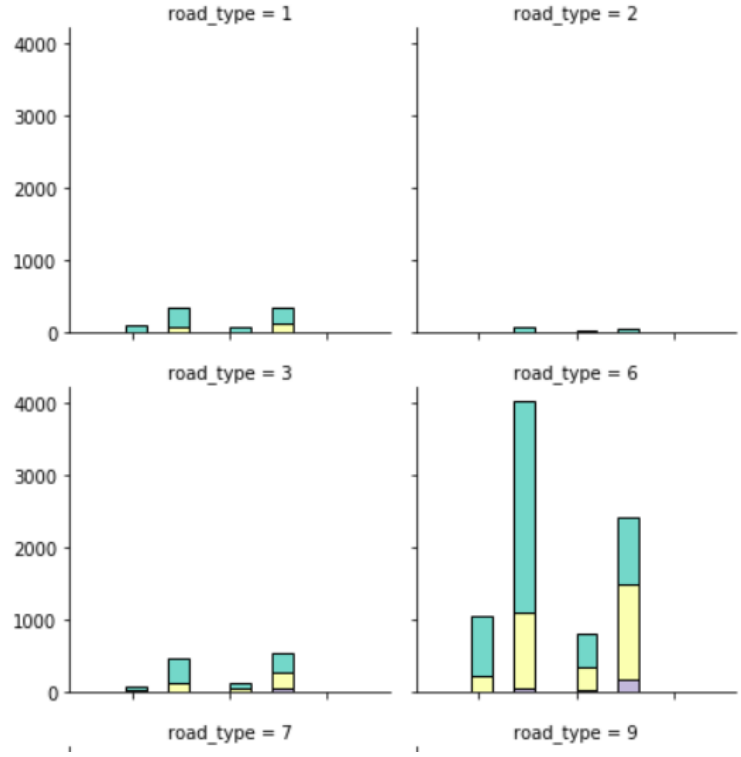
This chart seems to show that the data is heavily unbalanced for *weather conditions* data-points (almost all of the points relate to 'fine weather'). There is also not much relationship between weather type and accident severity.



We can see the data is fairly unbalanced in favour of there being no *carriageway hazards*.



This is interesting: a higher proportion of accidents occurring on dry road conditions, including more severe ones. There are very few results for road conditions including 'Snow, Frost or ice', 'Flood over 3cm. deep', 'Oil or diesel', 'Mud'. I will drop these columns.



As we can see here, there is a clear correlation between accidents occurring on a 'single-carriageway' (road-type '6') as opposed to 'dual-carriageway' etc.

Creating X feature and y target variable arrays

```
In [237]:
X = np.asarray(df_1)
X[0:5]

Out[237]: array([[ 1., 20.,  1.,  6.],
                 [ 2.,  3.,  1.,  3.],
                 [ 1.,  9.,  1.,  3.],
                 [ 1.,  9.,  1.,  3.],
                 [ 2.,  9.,  1.,  6.]])

In [238]:
y = np.asarray(df['accident_severity'])

In [239]:
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]

Out[239]: array([[-0.73,  1.9 , -0.61,  0.54],
                 [ 1.37, -1.07, -0.61, -1.31],
                 [-0.73, -0.02, -0.61, -1.31],
                 [-0.73, -0.02, -0.61, -1.31],
                 [ 1.37, -0.02, -0.61,  0.54]])
```

I'm created NumPy arrays for our chosen features, before normalising their values using Sklearn's standardscaler. I did this as the scales are different across the features, so they need to be converted into comparable values with equal weights. I then created our chosen target variable array, to see how it performs on a Logistic Regression model.

Splitting the data for training the model

```
In [240]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)

Train set: (95366, 4) (95366,)
Test set: (23842, 4) (23842,)
```

Here, I've split the data into training and test data.

```
In [242]:
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.1, solver='liblinear').fit(X_train,y_train)
LR

Out[242]: LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='auto', n_jobs=None, penalty='l2',
                             random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                             warm_start=False)

In [243]:
yhat = LR.predict(X_test)
print(yhat[0:5])
print(y[0:5])

[3 3 3 3 3]
[3 3 3 3 3]

In [244]:
yhat_prob = LR.predict_proba(X_test)
print(yhat_prob[0:5])

[[0.03 0.22 0.75]
 [0.02 0.18 0.8 ]
 [0.02 0.19 0.78]
 [0.01 0.11 0.88]
 [0.02 0.18 0.8 ]]
```

Logistic Regression- The logistic regression algorithm can predict categorical target variable values, as well as provide an estimated probability of each feature variable value

belonging to which category. Let's see how it performs. I added a regularisation parameter to check slightly overfitting on the data.

Model evaluation

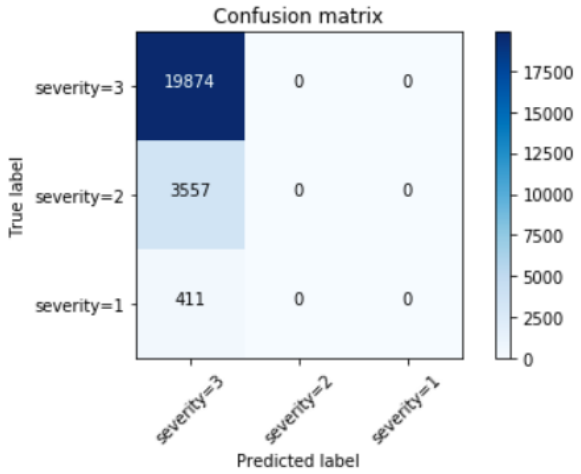
In [246]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

print("LR Jaccard index: %.2f" % jaccard_similarity_score(y_test, yhat))
print("LR F1-score: %.2f" % f1_score(y_test, yhat, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(y_test, yhat_prob))
```

```
LR Jaccard index: 0.83
LR F1-score: 0.76
LR LogLoss: 0.50
```

Though it seems our model performed well, I am worried that it has learnt to predict the value '3' just because that is what is predominantly represented within the data. Let's have a look at a confusion matrix to obtain more granularity on the predicted results.

```
Confusion matrix, without normalization
[[19874     0     0]
 [ 3557     0     0]
 [  411     0     0]]
```



As we can see in this confusion matrix, the model has not labelled any points as anything but 3.

In [250]:
```
drugTree.fit(X_train,y_train)
predTree = drugTree.predict(X_test)
print (predTree [0:5])
print (y_test [0:5])
```

```
[3 3 3 3 3]
[3 3 3 3 3]
```

In [251]:
```
print("DT Jaccard index: %.2f" % jaccard_similarity_score(y_test, predTree))
print("DT F1-score: %.2f" % f1_score(y_test, predTree, average='weighted') )
```

```
DT Jaccard index: 0.83
DT F1-score: 0.76
```

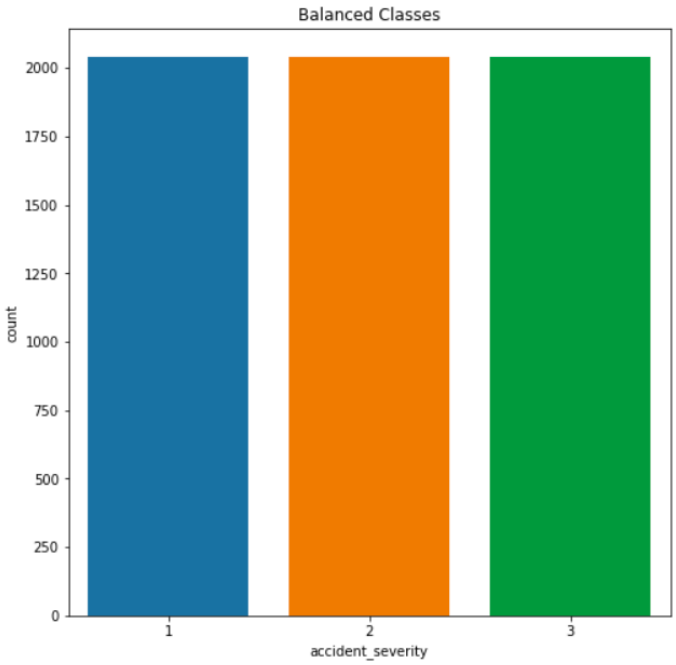
Decision Tree models tend to do better with unbalanced data-sets, but we can see similar results to our logistic regression model here.

Unbalanced data?

The models score highly, but I am worried it is assuming every point is a '3', and is not learning to generalise effectively (overfitting on the data).

If this model was trying to predict presence of a rare disease, predicting 'no' all the time might seem like a good strategy. But, crucially, it will get 0% right when the patient does have the disease!

Balancing the data-set



To attempt to deal with the unbalanced nature of this data-set in terms of its target-variable distribution, I shuffled the data, to ensure the target values ('slight', 'severe', and 'fatal') are equally represented. Therefore, I am going to reduce the number of data-points labelled as '2' and '3' to the same number as '1' (2042).

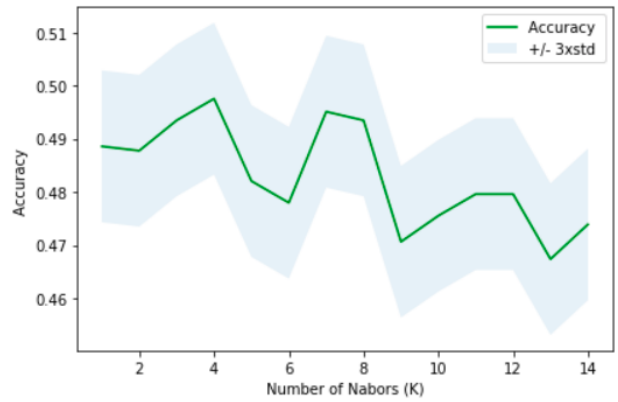
Re-run the models to see if we reduce over-fitting

I then recreated the feature and target arrays from the reduced data-set, and split it into test and training data again.

In [260]:
```
print("LR Jaccard index: %.2f" % jaccard_similarity_score(y_test, y_hat_1))
print("LR F1-score: %.2f" % f1_score(y_test, y_hat_1, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(y_test, yhat_1_prob))
```

```
LR Jaccard index: 0.46
LR F1-score: 0.43
LR LogLoss: 1.02
```

As we can see, the model has not really learnt the trends underlying the data. It is not really sure what to predict. Let's see if the Decision Tree, 'K' Nearest Neighbour, or Support Vector Machine models perform better.




```
print( "The best accuracy was with", mean_acc.max(), "wi
```

```
The best accuracy was with 0.49755301794453505 with k= 4
```

```
DT Jaccard index: 0.46
DT F1-score: 0.42
```

```
KNN Jaccard index: 0.50
KNN F1-score: 0.49
```

```
SVM Jaccard index: 0.50
SVM F1-score: 0.47
```

As we can see, the models perform fairly similarly. the overall best one being the 'K' Nearest Neighbours.

Results and Discussion

Although I think I selected features with decent predictive ability, in the first instance my classification models overfitted a data-set with unbalanced target data and the models went for the majority target variable value. My method for dealing with this unbalanced data was to re-balance it, but this caused the classification models to underperform perhaps because the re-balanced data did not reflect the actual distribution in real-life of the accident severity categories and therefore the predicted results were therefore roughly evenly split. The models had not learnt the data's underlyind trends effectively.

A couple of methods I could use to mitigate against this low performance are using a metric like AOC-weighted, which performs better with imbalance, as well as making increased use of hyper-parameters (regularisation) to penalise incorrect classifications. Regularisation is traditionally used to penalise against overfitting when there is not much data but a lot of features.

Conclusion

Having tested out four classification models on selected features from a comprehensive road accident data-set, I tried to test how well these features could predict the severity of an accident on new data. Although this was not entirely successful, in that I have not been able to create a model which adequately predicts the target variable, I was able to identify an imbalance in the data and modify my approach to deal with this. If I was to continue with this

project, I would try out further methods to deal with the classification models' tendency to predict the majority category class. For me, additionally, it has been a very useful learning process to practise some of the knowledge I have obtained whilst learning about data science, in particular exploring methods to deal with an unbalanced data-set.

Report this
Published by

Error! Filename not specified.

Andrew Murgatroyd
Lingüista-Traductor-Docente (MCIL CL, Chartered Linguist)
Published • 5s
1 article
hashtag#Capstone Project for IBM Data Science Professional Certificate
Like
Comment
Share