Accidents casualty prediction – Capstone project

Jurijs

Introduction/Business Problem

Motor vehicle accidents are the number one safety problem in US transportation. Similar situation can be seen all around the world. In 1996 in the US, 41,907 people were killed and 3,511,000 people were injured in police reported crashes. The lifetime economic cost of these crashes is over $150 billion annually. The global average of road fatalities is 18.2 deaths per 100,000 people, with lower income countries suffering a higher prevalence and higher income countries seeing lower rates of fatalities.

While driver errors such as speeding, distracted driving and drunk driving are among the leading causes of automobile accident, dangerous road conditions are also a significant contributor.

There are many road conditions that could cause you to lose control of your vehicle and crash, including: Ice and snow, Black ice, Confusing signs, Lack of signs, Potholes.

All this can conclude that the importance of this topic is very high, therefore the analysis of the environmental conditions, car accidents details may possibly predict risk of future accidents and potentially save human lives and reduce cost of the crash's consequences.

Our research takes into account datasets with actual weather and road conditions, identified obstacles on the road, type of vehicles, casualty (severity class), age of casualty. The dataset was made out of several EU statistical reports about car accidents took place in Leeds, UK from 2017-2019. This new dataset is a compilation of 3 reports published on the EU statistical portal https://www.europeandataportal.eu

With this we will try to predict possible severity of the car accident given the independent variables in our dataset.

Table 1. Definitions of variables:

1st Road Class	1st Road Class Desc
1	Motorway
2	A(M)
3	A
4	B
5	C
6	Unclassified

Road Surface	Road Surface Desc
1	Dry
2	Wet / Damp
3	Snow
4	Frost / Ice
5	Flood (surface water over 3cm deep)

Lighting Conditions	Lighting Conditions Desc
1	Daylight: street lights present
2	Daylight: no street lighting
3	Daylight: street lighting unknown
4	Darkness: street lights present and lit
5	Darkness: street lights present but unlit
6	Darkness: no street lighting
7	Darkness: street lighting unknown

Weather Conditions	Weather Conditions Desc
1	Fine without high winds
2	Raining without high winds
3	Snowing without high winds
4	Fine with high winds
5	Raining with high winds
6	Snowing with high winds
7	Fog or mist – if hazard
8	Other
9	Unknown

Casualty Class	Casualty Class Desc
1	Driver or rider
2	Vehicle or pillion passenger
3	Pedestrian

Casualty Severity	Casualty Severity Desc
1	Fatal
2	Serious
3	Slight

Sex of Casualty	Sex of Casualty Desc
1	Male
2	Female

Age of Casualty	
[Age given in years]	

Type of Vehicle	Type of Vehicle Desc
1	Pedal cycle
2	M/cycle 50cc and under
3	Motorcycle over 50cc and up to 125cc
4	Motorcycle over 125cc and up to 500cc
5	Motorcycle over 500cc
6	[Not used]
7	[Not used]
8	Taxi/Private hire car
9	Car
10	Minibus (8 – 16 passenger seats)
11	Bus or coach (17 or more passenger seats)
12	[Not used]
13	[Not used]
14	Other motor vehicle
15	Other non-motor vehicle
16	Ridden horse
17	Agricultural vehicle (includes diggers etc.)
18	Tram / Light rail
19	Goods vehicle 3.5 tonnes mgw and under
20	Goods vehicle over 3.5 tonnes and under 7.5 tonnes mgw
21	Goods vehicle 7.5 tonnes mgw and over
22	Mobility Scooter
90	Other Vehicle
97	Motorcycle - Unknown CC

Data Wrangling

The dataset in the original form was not ready for data analysis. Original source had several separate file, and required to be adjusted in excel format first. In order to prepare the data, first, we had to drop the non-relevant columns. In addition, most of the features are of object data types that need to be converted into numerical data types. As we go forward with the analysis, it was performed at later steps.

Fig.1. Checking for missing data:

```
#Checkinh how much missing values we have
missing_data = df.isnull()

for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
1st Road Class & No
False    3902
True     2203
Name: 1st Road Class & No, dtype: int64

Local Authority
False    3902
True     2203
Name: Local Authority, dtype: int64

Vehicle Number
False    3902
True     2203
Name: Vehicle Number, dtype: int64
```

Some columns with missing values were dropped, as they were Irrelevant for our analysis. 'Vehicle Number' column has been Kept and missing data was filled by random value, and data type Changed to integer:

```
def fill_missing(column_val):
    if np.isnan(column_val) == True:
        column_val = np.random.randint(1,3)
    else:
         column_val = column_val
    return column_val
```

The number of columns were dropped as they were not required for the analysis. While checking out target variable **'Casualty Severity'** , it appears that the number of rows in class 3 is almost 5 times bigger than the number of rows in class 2. It is possible to solve the issue by down sampling the class 3.

Fig.2.Modifying data

```
df1['Casualty Severity'].value_counts()
```

```
3    5088
2     954
1      63
Name: Casualty Severity, dtype: int64
```

Downsampling

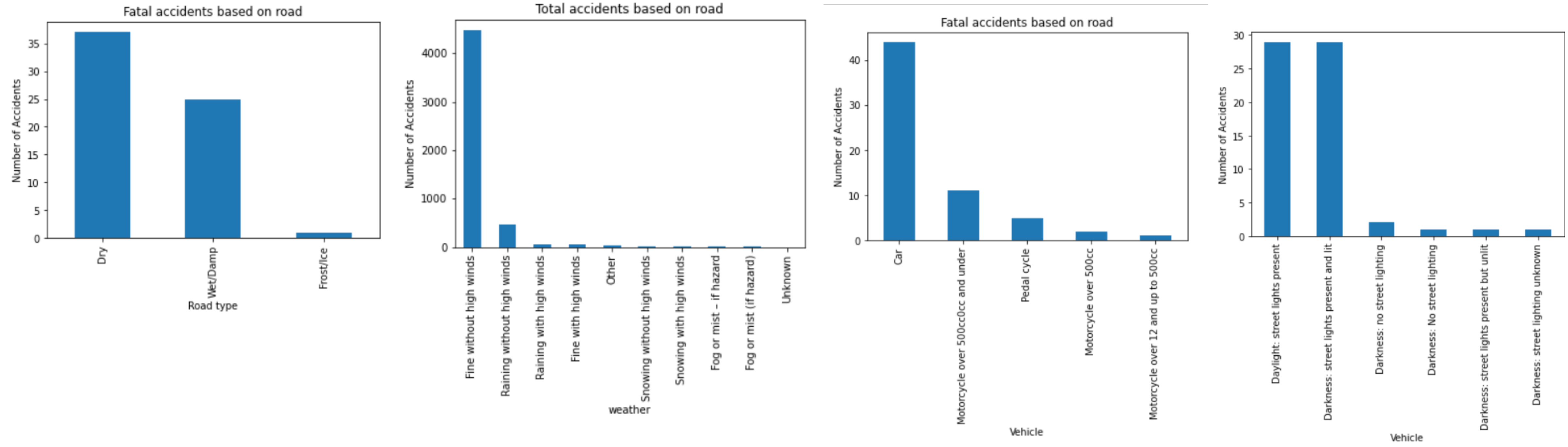
```
In [260]: #Downsampling the data

          from sklearn.utils import resample

          df_maj = df1[df1.Casualty_Severity==3]
          df_min = df1[df1.Casualty_Severity!=3]
          df_dsample=resample(df_maj,replace=False,n_samples=1000,random_state=123)
          balanced_df = pd.concat([df_dsample,df_min])
          balanced_df['Casualty_Severity'].value_counts()
```

```
Out[260]: 3    1000
          2     954
          1      63
          Name: Casualty_Severity, dtype: int64
```

Matplotlib package was used further in order to check dependencies of the obj data with the target variable Casualty Severity. Some plots were taken and showed some statistical summary.
Best on 3 years collected data, the fatal accidents happened on dry road, with good weather, on the car, and in general happened similar times at daylight and nigh time.

```
#Checking dependencies of the fatal accidents using deifferent independent
df1[df1.Casualty_Severity==1].Road_Surface.value_counts().plot(kind='bar')
plt.title('Fatal accidents based on road')
plt.ylabel('Number of Accidents')
plt.xlabel('Road type')
plt.show()
```

Fig.3.Plots



Also we have run a value count on road (‘**Road_Surface**’) and weather condition (‘**Weather_Conditions**’) to get ideas of the different road and weather conditions, on light condition (’**Lighting_Condition**’), to see the breakdowns of accidents occurring during the different light conditions. Similarly we have run value count for other attributes as well. The results of a few can be seen below:

Fig.4.Value_Counts

```
In [269]: df1['Road_Surface'].value_counts()

Out[269]: Dry          4532
          Wet/Damp     1467
          Frost/Ice      74
          Snow           30
          Unknown         2
          Name: Road_Surface, dtype: int64

In [270]: df1['Lighting_Conditions'].value_counts()

Out[270]: Daylight: street lights present              4013
          Darkness: street lights present and lit      1429
          Darkness: street lighting unknown             502
          Darkness: no street lighting                   87
          Darkness: No street lighting                   46
          Darkness: street lights present but unlit      28
          Name: Lighting_Conditions, dtype: int64

In [271]: df1['Weather_Conditions'].value_counts()

Out[271]: Fine without high winds       5344
          Raining without high winds     564
          Raining with high winds         64
          Fine with high winds            53
          Other                           38
          Snowing without high winds      25
          Snowing with high winds          7
          Fog or mist - if hazard          5
          Fog or mist (if hazard)          3
          Unknown                          2
          Name: Weather_Conditions, dtype: int64

In [272]: df1['TypeofVehicle'].value_counts()

Out[272]: Car                                     4009
          Pedal cycle                             1350
          Motorcycle over 500cc0cc and under       344
          Taxi/Private hire car                    202
          Motorcycle over 500cc                    129
          Motorcycle over 12 and up to 500cc        71
          Name: TypeofVehicle, dtype: int64

In [273]: df1['SexofCasualty'].value_counts()

Out[273]: Male      3575
          Female    2530
          Name: SexofCasualty, dtype: int64
```

Data Pre-processing

As you may figure out, some features in this dataset are categorical such as **Road_Surface, Weather_Conditions,** etc. Unfortunately; Sklearn do not handle categorical variables., but still we can convert these features to numerical values

Fig.5.Changing to numerical values

Road_Surface	Lighting_Conditions	Weather_Conditions	Vehicle_Number	TypeofVehicle
Dry	Daylight: street lights present	Fine without high winds	2	Pedal cycle
Dry	Daylight: street lights present	Fine without high winds	2	Pedal cycle
Dry	Daylight: street lights present	Fine without high winds	2	Pedal cycle
Dry	Daylight: street lights present	Fine without high winds	2	Pedal cycle
Dry	Daylight: street lights present	Fine without high winds	2	Pedal cycle

```
df_lc=df1.Lighting_Conditions.value_counts().to_dict()
df1.Lighting_Conditions=df1.Lighting_Conditions.map(df_lc)
df1.head()
```

```
df_wc=df1.Weather_Conditions.value_counts().to_dict()
df1.Weather_Conditions=df1.Weather_Conditions.map(df_wc)
df_tv=df1.TypeofVehicle.value_counts().to_dict()
df1.TypeofVehicle=df1.TypeofVehicle.map(df_tv)
df_sex=df1.SexofCasualty.value_counts().to_dict()
df1.SexofCasualty=df1.SexofCasualty.map(df_sex)
df1.head()
```

Road_Surface	Lighting_Conditions	Weather_Conditions	Vehicle_Number	TypeofVehicle
4532	4013	5344	2	1350
4532	4013	5344	2	1350
4532	4013	5344	2	1350
4532	4013	5344	2	1350
4532	4013	5344	2	1350

Data Pre-processing

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases.

Fig.6.Normalisation

```
In [292]: from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import StandardScaler

In [295]: s=StandardScaler()
          s.fit(x)
          x_scale=s.transform(x)
          x_scale

Out[295]: array([[ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
                  -1.18871505, -1.61465333],
                 [ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
                   0.8412445 , -1.45409707],
                 [ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
                  -1.18871505,  0.15146552],
                 ...,
                 [ 0.0790905 , -1.61528478, -1.12388738, ...,  0.43387037,
                   0.8412445 , -0.86539079],
                 [ 0.0790905 , -1.61528478, -1.12388738, ..., -2.01870241,
                   0.8412445 , -0.11612824],
                 [ 0.0790905 , -1.61528478,  0.7068143 , ...,  0.43387037,
                   0.8412445 , -0.49075952]])
```

Train Test Split

Out of Sample Accuracy is the percentage of correct predictions that the model makes on data that that the model has NOT been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

It is important that our models have a high, out-of-sample accuracy, because the purpose of any model, of course, is to make correct predictions on unknown data

Fig.7.Train-test-split

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
s=StandardScaler()
s.fit(x)
x_scale=s.transform(x)
x_scale
```

```
array([[ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
        -1.18871505, -1.61465333],
       [ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
         0.8412445 , -1.45409707],
       [ 0.0790905 ,  0.5842581 ,  0.7068143 , ...,  0.43387037,
        -1.18871505,  0.15146552],
       ...,
       [ 0.0790905 , -1.61528478, -1.12388738, ...,  0.43387037,
         0.8412445 , -0.86539079],
       [ 0.0790905 , -1.61528478, -1.12388738, ..., -2.01870241,
         0.8412445 , -0.11612824],
       [ 0.0790905 , -1.61528478,  0.7068143 , ...,  0.43387037,
         0.8412445 , -0.49075952]])
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x_scale,y,test_size=0.2,random_state=
```

```
print('Train set:', xtrain.shape, ytrain.shape)
print('Test set:', xtest.shape, ytest.shape)
```

```
Train set: (4884, 10) (4884,)
Test set: (1221, 10) (1221,)
```

Modeling

After balancing SEVERITYCODE feature, and standardizing the input feature, the data has been ready for building machine learning models.

I have employed three machine learning models:

K Nearest Neighbor (KNN) Decision Tree Logistic Regression

After importing necessary packages and splitting preprocessed data into test and train sets, for each machine learning model, I have built and evaluated the model.

Fig.8.Train-test-split

```
from sklearn.neighbors import KNeighborsClassifier
k=4
neigh = KNeighborsClassifier(n_neighbors = k).fit(xtrain,ytrain)
neigh
```

```
KNeighborsClassifier(n_neighbors=4)
```

```
yhat = neigh.predict(xtest)
yhat[0:5]
```

```
array([3, 3, 3, 3, 2], dtype=int64)
```

```
from sklearn import metrics
print('Train set accuracy: ', metrics.accuracy_score(ytrain, neigh.predict(xtrain)))
print('Test set accuracy: ', metrics.accuracy_score(ytest,yhat))
```

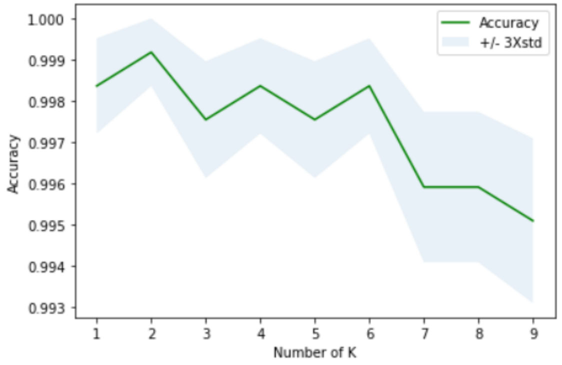
```
Train set accuracy:  0.9989762489762489
Test set accuracy:  0.9983619983619983
```

Calculate accuracy for different Ks

```
Ks=10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfusionMx = [];
for n in range(1,Ks):
    neigh = KNeighborsClassifier(n_neighbors = n).fit(xtrain, ytrain)
    yhat = neigh.predict(xtest)
    mean_acc[n-1]=metrics.accuracy_score(ytest,yhat)
    std_acc[n-1]=np.std(yhat==ytest)/np.sqrt(yhat.shape[0])
mean_acc
```

```
array([0.998362, 0.999181, 0.997543, 0.998362, 0.997543, 0.998362,
       0.995905, 0.995905, 0.995086])
```

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1*std_acc, mean_acc + 1 * std_acc, al
plt.legend(('Accuracy', '+/- 3Xstd'))
plt.ylabel('Accuracy')
plt.xlabel('Number of K')
plt.tight_layout()
plt.show()
```




```
print("The best accuracy: ", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy:  0.9991809991809992 with k= 2
```

The we rebuild model using best K = 2

Fig.9

```python
#rebuild model with best walue for K=2
k=2
neigh = KNeighborsClassifier(n_neighbors = k).fit(xtrain, ytrain)
neigh
```

```
KNeighborsClassifier(n_neighbors=2)
```

```python
yhat = neigh.predict(xtest)
yhat[0:5]
```

```
array([3, 3, 3, 3, 2], dtype=int64)
```

```python
from sklearn import metrics
print('Train set accuracy: %.5f' % metrics.accuracy_score(ytrain, neigh.predict(xtrain)))
print('Test set accuracy: ', metrics.accuracy_score(ytest,yhat))
```

```
Train set accuracy: 1.00000
Test set accuracy:  0.9991809991809992
```

```python
from sklearn.metrics import f1_score
print('F1 score: %.5f' % f1_score(ytest,yhat, average = 'weighted'))
```

```
F1 score: 0.99916
```

Decision Tree model and Logistic Regression

```
from sklearn.tree import DecisionTreeClassifier
DT_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)
DT_model.fit(xtrain,ytrain)
DT_model
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
yhat = DT_model.predict(xtest)
yhat
```

```
array([3, 3, 3, ..., 3, 3, 3], dtype=int64)
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
print('DecisionTree accuracy: %.5f' % metrics.accuracy_score(ytest,yhat))
```

```
DecisionTree accuracy: 1.00000
```

```
from sklearn.metrics import f1_score
print('F1 score: %.5f' % f1_score(ytest,yhat, average = 'weighted'))
```

```
F1 score: 1.00000
```

Fig.10

```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(C=0.01).fit(xtrain, ytrain)
LR_model
```

```
LogisticRegression(C=0.01)
```

```
yhat= LR_model.predict(xtest)
yhat
```

```
array([3, 3, 3, ..., 3, 3, 3], dtype=int64)
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
LR_yhat = LR_model.predict(xtest)
LR_yhat_prob = LR_model.predict_proba(xtest)
```

```
print('LR f1 score:',f1_score(ytest, LR_yhat, average = 'weighted'))
print('LR LogLoss:', log_loss(ytest,LR_yhat_prob))
```

```
LR f1 score: 0.9890324153058737
LR LogLoss: 0.05359894464780914
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
print('Decision Tree Accuracy: ', metrics.accuracy_score(ytest, yhat))
```

```
Decision Tree Accuracy:  0.9926289926289926
```

```
print('f1 score:',metrics.f1_score(ytest,yhat, average = 'weighted'))
print('Accuracy score:',metrics.accuracy_score(ytest,yhat))
```

```
f1 score: 1.0
Accuracy score: 1.0
```

Conclusion

Based on the dataset provided for this capstone from weather, road, light conditions, etc. pointing to certain classes, we can conclude that particular conditions have some impact on the severity of casualty and we can with predict it with certain values.