Predicting the injury level of car incidents

Elliot Ho

September 22,2020

1. Introduction

1.1 Background

Seattle Metropolitan Traffic Police have a huge dataset consisting of motor vehicle accident information. In the last 5 years they have handled several accident cases that could have been prevented. Hence, we are going to develop a model that will help prevent such incidents with a reasonable accuracy.

Now, wouldn't it be great if there is something in place that could warn you, given the weather and the road conditions about the possibility of you getting into a car accident and how severe it would be, so that you would drive more carefully or even change your travel if you are able to.

1.2 Problem

The problem statement here is to predict the severity of an accident given the data about the current weather conditions, road conditions and the location. When there is extensive data for these incidents, we might be able to come across a pattern that suggests a high probability for accidents.

1.3 Interest

The project stake holders are the Seattle City corporation for implementation of safety strategies and reduction of fatalities. Stakeholder groups includes state, federal and local government agencies, nongovernmental organizations and regional authorities. Car and life Insurance companies are benefited from the result to have a knowledge of the most recurring accident type and places. Urgent and Emergency care get data-driven information from this project to plan schedules of emergency room and professional to save the life of injured person.

2. Data acquisition and cleaning

2.1 Data sources

The data file that we need mainly are:

[Data-collisions.csv]

the dataset consists of list of incidents. Each incident has an incident number, X co-ordinate, y co-ordinate, severity code, location address, severity description, road conditions, weather, light conditions and many more details. For further information and details, click [here] for accessing the meta data for the dataset.

2.2 Data Cleaning

Drop the irrelevant attributes

In the data cleaning stage, we have to drop the irrelevant features columns. For example, ADDRTYPE, SEVERITYDESC, COLLISIONTYPE, etc. Also, we need to drop out the irrelevant unique IDs columns, such as OBJECTID, INCKEY, COLDETKEY and so on.

Drop the Unknown index

Besides, some of the independent variables have the 'Unknown' option in category but the 'Unknown' cannot help us to predict the severity. Therefore, we also need to drop the unknown items when in the data cleaning stage.

Dealing with the null values

Some of the independent variables have the missing values in rows, we need to replace them or drop them out from the dataset so that prevent any error when in the modeling stage.

Feature selection

After cleaning there were 167981 rows and 12 columns. Upon examining the data, it was clear that there were lots of redundancies. There were several columns to indicate the address, like 'addrtype' and 'location' but what we are really interested in is the latitude and longitude indication which is why they are definitely going inside our feature set. Between 'Severity code' and ‘severity desc’ it’s the ‘severity code’ that is easier to include in our feature set as number datatypes help to easily train the model. ‘person count’, ‘pedestrian count’ and ‘pedestrian cycle count’ are 3 redundant columns that we are going to omit, since they don’t really relate to accident severity.

The environmental conditions that contribute to accidents are ‘road conditions’ and ‘weather conditions’ that are categorical variables and would be a part of our feature set. Out of the redundant attributes ‘junction type’ and ‘address type’ we will be having ‘junction type’ as part of our feature set.

3. Exploratory Data Analysis

3.1 Calculation of target variable

Although the metaset pointed out that the SEVERITYCODE(target variable) has 0 to 3 values, it only consists 1 and 2.

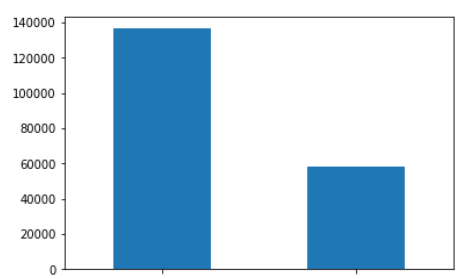
In [194]:
```
df['SEVERITYCODE'].describe()
```

Out[194]:
```
count    194673.000000
mean          1.298901
std           0.457778
min           1.000000
25%           1.000000
50%           1.000000
75%           2.000000
max           2.000000
Name: SEVERITYCODE, dtype: float64
```

We can see that the maximum and the minimum of the target variable only is 2 and 1 instead of 0 and 3. And the below is the distribution of the target variable.

In [195]:
```
cnt_severitycode = df['SEVERITYCODE'].value_counts()
cnt_severitycode.plot(kind='bar')
```

Out[195]: <matplotlib.axes._subplots.AxesSubplot at 0x164c221b5c8>

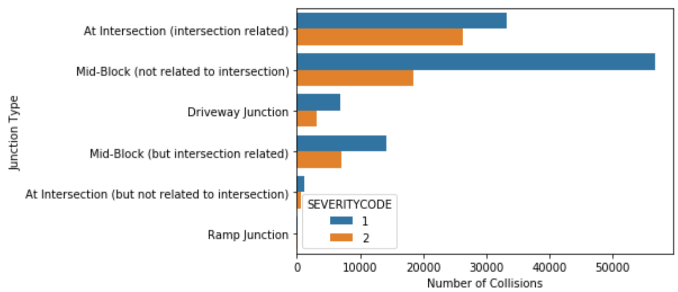


3.2 Relationship

Relationship between junction yype and severity

In [211]:
```
ax = sns.countplot(y="JUNCTIONTYPE", hue="SEVERITYCODE", data=df)
ax.set(xlabel='Number of Collisions', ylabel="Junction Type")
df.groupby(['JUNCTIONTYPE'])['SEVERITYCODE'].value_counts()
```

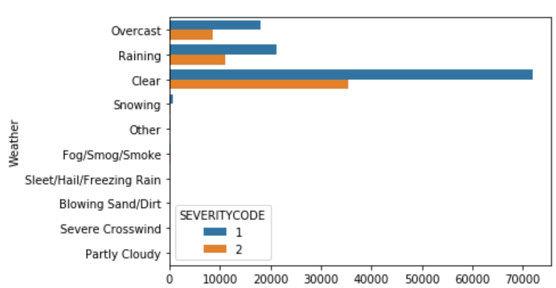
Out[211]:
```
JUNCTIONTYPE                                     SEVERITYCODE
At Intersection (but not related to intersection)  1               1197
                                                   2                601
At Intersection (intersection related)             1              33276
                                                   2              26233
Driveway Junction                                  1               6848
                                                   2               3133
Mid-Block (but intersection related)               1              14229
                                                   2               6986
Mid-Block (not related to intersection)            1              56823
                                                   2              18498
Ramp Junction                                      1                105
                                                   2                 52
Name: SEVERITYCODE, dtype: int64
```



Relationship between weather and severity

In [212]:
```
ax = sns.countplot(y="WEATHER", hue="SEVERITYCODE", data=df)
ax.set(xlabel='Number of Collisions', ylabel="Weather")
df.groupby(['WEATHER'])['SEVERITYCODE'].value_counts()
```

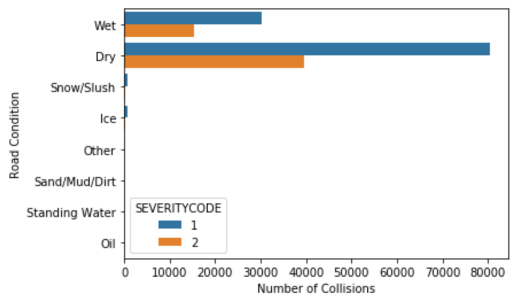
Out[212]:
```
WEATHER                   SEVERITYCODE
Blowing Sand/Dirt         1                  30
                          2                  13
Clear                     1               71897
                          2               35414
Fog/Smog/Smoke            1                 361
                          2                 183
Other                     1                 172
                          2                  78
Overcast                  1               17981
                          2                8585
Partly Cloudy             2                   3
                          1                   2
Raining                   1               21280
                          2               11029
Severe Crosswind          1                  18
                          2                   7
Sleet/Hail/Freezing Rain  1                  83
                          2                  27
Snowing                   1                 654
                          2                 164
Name: SEVERITYCODE, dtype: int64
```



Relationship between road conditions and serverity

In [213]:
```
ax = sns.countplot(y="ROADCOND", hue="SEVERITYCODE", data=df)
ax.set(xlabel='Number of Collisions', ylabel="Road Condition")
df.groupby(['ROADCOND'])['SEVERITYCODE'].value_counts()
```

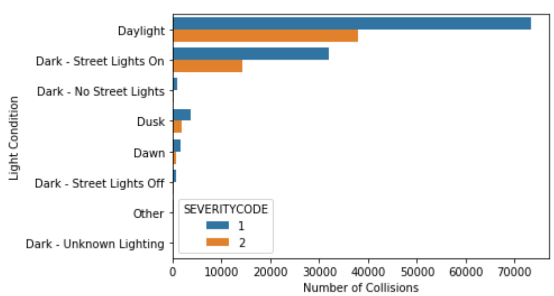
Out[213]:
```
ROADCOND        SEVERITYCODE
Dry             1               80410
                2               39483
Ice             1                 818
                2                 262
Oil             1                  34
                2                  24
Other           1                  63
                2                  42
Sand/Mud/Dirt   1                  40
                2                  21
Snow/Slush      1                 680
                2                 156
Standing Water  1                  75
                2                  30
Wet             1               30358
                2               15485
Name: SEVERITYCODE, dtype: int64
```



Relationship between light condition and severity

In [257]:
```
ax = sns.countplot(y="LIGHTCOND", hue="SEVERITYCODE", data=df)
ax.set(xlabel='Number of Collisions', ylabel="Light Condition")
print(df.groupby(['LIGHTCOND'])['SEVERITYCODE'].value_counts())
```

```
Other                        1                  136
                             2                   49
Name: SEVERITYCODE, dtype: int64
```



4. Predictive Modeling

The two commonly used strategies to balance classes in a dataset are random over-sampling (ROS) and random under-sampling (RUS). ROS is the process of supplementing the dataset with multiple, randomly chosen copies of cases from the minority class, until the number of samples match the majority class. RUS randomly deletes samples from the majority class until the number of samples matches the minority class.Both methods come with advantages and disadvantages. While ROS may inate or exaggerate underlying patterns in the minority class, RUS may potentially discard important samples of majority class and distort its underlying patterns. A rule of thumb is to use ROS when the given dataset is small and RUS when the given dataset is large. As the collision dataset is very large; this report will employ the random under-sampling method to balance the dataset and thereby reduce class 1 collisions to 55503 samples.

In [221]:

```python
from imblearn.under_sampling import RandomUnderSampler
from sklearn import preprocessing
import imblearn

x = df[['VEHCOUNT', 'SDOT_COLCODE', 'INATTENTIONIND',
        'UNDERINFL', 'SPEEDING', 'HITPARKEDCAR', 0, 1, 2, 3, 4, 10, 11, 12,
        13, 14, 15, 16, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
        32, 40, 41, 42, 43, 45, 48, 50, 51, 52, 53, 54, 56, 57, 64, 65, 66,
        67, 71, 72, 73, 74, 81, 82, 83, 84, 87, ' ', '0', '1', '10', '11',
        '12', '13', '14', '15', '16', '17', '18', '19', '2', '20', '21',
        '22', '23', '24', '25', '26', '27', '28', '29', '3', '30', '31',
        '32', '4', '40', '41', '42', '43', '45', '48', '49', '5', '50',
        '51', '52', '53', '54', '56', '57', '6', '60', '64', '65', '66',
        '67', '7', '71', '72', '73', '74', '8', '81', '82', '83', '84',
        '85', '87', '88',
        'At Intersection (but not related to intersection)',
        'At Intersection (intersection related)', 'Driveway Junction',
        'Mid-Block (but intersection related)',
        'Mid-Block (not related to intersection)', 'Ramp Junction',
        'Blowing Sand/Dirt', 'Clear', 'Fog/Smog/Smoke', 'Overcast',
        'Raining', 'Severe Crosswind', 'Sleet/Hail/Freezing Rain',
        'Snowing', 'Unknown Weather', 'Dry', 'Ice', 'Oil', 'Sand/Mud/Dirt',
        'Snow/Slush', 'Standing Water', 'Unknown Roadcond', 'Wet',
        'Dark - No Street Lights', 'Dark - Street Lights Off',
        'Dark - Street Lights On', 'Dark - Unknown Lighting', 'Dawn',
        'Daylight', 'Dusk', 'Other']]

y = df['SEVERITYCODE']


#Use random under sampling (RUS) method such that the resulting numbers of each severity class are equal.
#RUS randomly removes samples from the majority class (in this case SEVERITYCODE = 1) such that the number present
#equal that of the minority class SEVERITYCODE = 2 such that they are now equal.
RUS = RandomUnderSampler(random_state=12)
x_resampled, y_resampled = RUS.fit_resample(x, y)
```

In [222]:

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

x_prp = preprocessing.StandardScaler().fit(x_resampled).transform(x_resampled)
```

4.1 Logistic Regression

Logistic Regression

In [224]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.05).fit(X_train,y_train)
yhat = LR.predict(X_test)
yhat
```

Out[224]:

```
array([2, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [225]:

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[225]:

```
array([[0.21455623, 0.78544377],
       [0.55680775, 0.44319225],
       [0.5452639 , 0.4547361 ],
       ...,
       [0.63097897, 0.36902103],
       [0.52227136, 0.47772864],
       [0.86935019, 0.13064981]])
```

In [226]:

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

jaccard_score_LR = jaccard_score(y_test, yhat)
f1_score_LR = f1_score(y_test, yhat, average='weighted')

print('The jaccard score is: ',jaccard_score_LR)
print('The F-1 score is: ',f1_score_LR)
```

```
The jaccard score is:  0.7034376435084957
The F-1 score is:  0.6878933101422043
```

4.2 Decision Tree

Decision Tree

In [227]:

```
X_train_DT, X_test_DT, y_train_DT, y_test_DT = train_test_split( x, y, test_size=0.2, random_state=4)
print ('Train set:', X_train_DT.shape,  y_train_DT.shape)
print ('Test set:', X_test_DT.shape,  y_test_DT.shape)
```

```
Train set: (134384, 152) (134384,)
Test set: (33597, 152) (33597,)
```

In [228]:

```
from sklearn.tree import DecisionTreeClassifier

drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 8)
drugTree # it shows the default parameters
```

Out[228]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=8)
```

In [229]:

```
drugTree.fit(X_train_DT,y_train_DT)

predTree_DT = drugTree.predict(X_test_DT) #prediction

```

In [230]:

```
from sklearn import metrics

print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test_DT, predTree_DT))
jaccard_score_DT = jaccard_score(y_test_DT,predTree_DT)
f1_score_DT = f1_score(y_test_DT, predTree_DT, average = 'weighted')
print('The jaccard score is: ',jaccard_score_DT)
print('The F-1 score is: ',f1_score_DT)
```

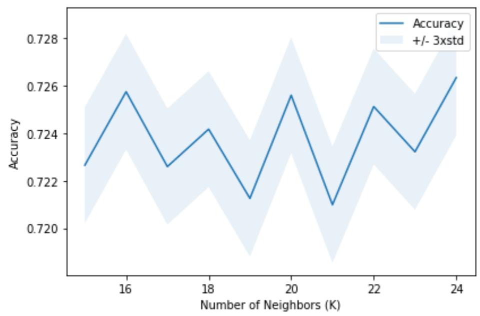
```
DecisionTrees's Accuracy:  0.7314045896955085
The jaccard score is:  0.7024531785808494
The F-1 score is:  0.691554267748142
```

4.3 K-Nearest Neighbour (KNN)

In [46]:
```python
# Visualize k-value over accuracy
plt.plot(range(15,Ks),mean_acc[14:25])
plt.fill_between(range(15,Ks),mean_acc[14:25] - 1 * std_acc[14:25],mean_acc[14:25] + 1 * std_acc[14:25], alpha=0.10)

#Plotting line graph displaying the accuracy of the classifier with each value K = n neighbours.
plt.legend(('Accuracy', '+/- 3xstd'))
plt.ylabel('Accuracy')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()

#Print classifier with the best accuracy.
print( "The best general accuracy was at", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```




```
The best general accuracy was at 0.7263465204957102 with k= 24
```

In [48]:
```python
# build final KNN model
k = 24
KNN = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
y_hat = KNN.predict(X_test)

# full evalution
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score

#Printing off evaluation metrics for K-NN classifier
acc1 = metrics.accuracy_score(y_test, y_hat)
jc1 = jaccard_score(y_test, y_hat)
fs1 = f1_score(y_test, y_hat, average='weighted')
print("Accuracy Score: ", acc1)
print("Jaccard Score: ", jc1)
print("F1 Score: ", fs1)
```

```
Accuracy Score:  0.7232185414680649
Jaccard Score:  0.6848799348799349
F1 Score:  0.6992675464903947
```

4.4 Comparison table for different predictive model

In [265]:
```python
df_result
```

Out[265]:

	Model	Jaccard	f1_Score
0	KNN	0.684880	0.699268
1	DT	0.702453	0.691554
2	LR	0.703438	0.687893

5. Conclusions

In this study I was able to analyze the patterns of road accidents and their severity. With the right prediction system in place many of these accidents can be avoided with the right information in place. This system is highly beneficial for the Traffic police to avert accidents on a day-to-day basis.