Coursera Capstone Report

Car Accident Severity- Seattle, USA

By: George Saliba

Date: 09/09/2020

Introduction

According to online sources, the USA experiences an average of approximately 102 vehicle crashes per day. From approximately 6 million crashes a year, over 35,000 crashes are fatal. To improve the country's road safety, it is important to understand the causes of accidents as well as the conditions leading to the incident.

Moreover, understanding the factors prior to the incidents through graphical and statistical analysis, including the level of severity, likelihood of collisions, and probability of injury; can further improve the education sector within road safety as well as give governmental bodies the data needed to build safer roads and raise awareness of the main factors and conditions leading to road incidents.

The aim of this project is to use publicly available data- published by the Seattle Deport of Transportation (SDOT) in the form of a dataset, to identify the relevant information that allow for the comprehension on the number, injuries, and severity of collision occur in Seattle.

The data will be analyzed mathematically and graphically through data science methodologies such as data visualization tools and machine learning models in order to provide insight on traffic collisions in Seattle.

Data

The data used throughout this project is extracted from the SDOT's website entitled "Collisions- All Years". This dataset includes all types of collisions, whether be vehicle, bicycle or pedestrian, filed by the Seattle Police Department.

The dataset provided includes 65,000 collisions from 2004 to present day, updated on a weekly basis. The severity is also an important factor that is incorporated within the collision's dataset with 5 codes, explained:

- 0: unknown
- 1: property damage
- 2: injury
- 2b: serious injury
- 3: fatality

The actual dataset only includes codes 1 and 2 without the inclusion of any other alternatives; but gives 37 attributes related to the collisions reported such as day, time, month, weather etc. All of which will be used to build our model and explain trends related to traffic collisions.

The full description of the data can be found at: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Metadata.pdf

Methodology

The first important step taken was to import the needed libraries into my Jupyter Notebook. The screenshot below represents the above:

Importing Required Libraries & Tools

In [80]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
from sklearn import preprocessing
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss


from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib.ticker import NullFormatter
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
from sklearn.utils import resample
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.image as mpimg
from sklearn import tree
from sklearn.tree import export_graphviz
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
import matplotlib as mpl
```

Next, the dataset, as a .csv file was imported into the Notebook and the codes to generate the following were implemented:

1)

```
# List of Columns df_data_1.columns
```

2)

```
# Check Data that has missing valuesdf_data_1.info()
```

3)

```
# Check size of Data df_data_1.isnull().sum()
```

Then, all the columns with null values where dropped as well as the columns that will not be beneficial throughout the report.

This makes the remaining factors: Severity Code, Weather Conditions, Road Conditions, and Light Conditions.

```
In [85]: # The columns with mostly null values will be dropped
df_data_1.drop(columns = ['INTKEY','EXCEPTRSNCODE','EXCEPTRSNDESC','INATTENTIONIND','PEDROWNOTGRNT','SPEEDING','SDOTCOLNUM'],a
xis = 1,inplace = True)

In [86]: # Drop columns that will not be beneficial throughout the project
df_data_1.drop(columns = ['X','Y','OBJECTID','INCKEY','COLDETKEY','REPORTNO','STATUS','LOCATION','SEVERITYCODE.1','SEVERITYDES
C','INCDATE','INCDTTM','SDOT_COLCODE','SDOT_COLDESC','ST_COLCODE','ST_COLDESC','SEGLANEKEY','CROSSWALKKEY','ADDRTYPE','COLLISI
ONTYPE','JUNCTIONTYPE','UNDERINFL','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','HITPARKEDCAR'],axis = 1,inplace = True)

In [87]: # The Severity Code, Weather, Road Condition, and Light Condition are the main driving factors
df_data_1.head(10)
```

Out[87]:

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
0	2	Overcast	Wet	Daylight
1	1	Raining	Wet	Dark - Street Lights On
2	1	Overcast	Dry	Daylight
3	1	Clear	Dry	Daylight
4	2	Raining	Wet	Daylight
5	1	Clear	Dry	Daylight
6	1	Raining	Wet	Daylight
7	2	Clear	Dry	Daylight
8	1	Clear	Dry	Daylight
9	2	Clear	Dry	Daylight

The NaN values were also adjusted within the Severity Code, Weather Conditions, Road Conditions, and Light Conditions columns using the following code:

```
#We need to adjust each of the 4 factors by replacing NaN values with unknown values#Weatherdf_data_1['WEATHER'].replace(np.NaN, "Unknown", inplace=True)

#Road Condition df_data_1['ROADCOND'].replace(np.NaN, "Unknown", inplace=True)

#Light Conditiondf_data_1['LIGHTCOND'].replace(np.NaN, "Unknown", inplace=True)
```

After the data was checked and successfully implemented, the ***Exploratory Data Analysis*** phase began.

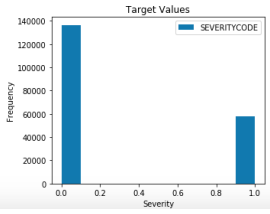
This section of the Notebook was dedicated towards understanding the relationship between the Severity accidents and the other factors (Weather, Road and Light Conditions).

The following screenshots represent the codes used to execute the needed graphs.

FREQUENCY vs. SEVERITY PLOT

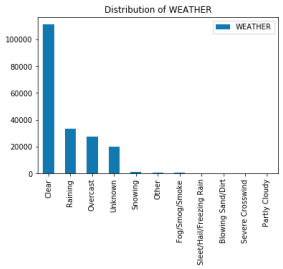
```
In [94]: severity_code = df_data_1['SEVERITYCODE'].values
labels = preprocessing.LabelEncoder()
labels.fit([1, 2])
severity_code = labels.transform (severity_code)
df_data_1["SEVERITYCODE"] = severity_code

In [95]: #Graphical Representation of Frequency versus Severity Plot and Pie Chart
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df_data_1.plot.hist(figsize=(5,4))
plt.title('Target Values')
plt.xlabel('Severity')
plt.ylabel('Frequency')
print('Accidents in each severity level:')
```

```
Accidents in each severity level:
```



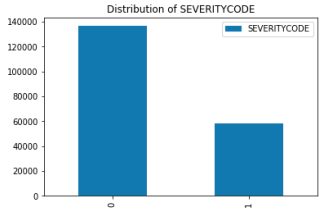
DISTRIBUTION OF WEATHER

```
In [98]: #Distribution of Weather
df_data_1['WEATHER'].value_counts().to_frame().plot(kind='bar', y='WEATHER')
plot.title('Distribution of WEATHER')
plot.show()
```



DISTRIBUTION OF SEVERITY CODE

```
In [99]: #Distribution of Severity Code
df_data_1['SEVERITYCODE'].value_counts().to_frame().plot(kind='bar', y='SEVERITYCODE')
plot.title('Distribution of SEVERITYCODE')
plot.show()
```



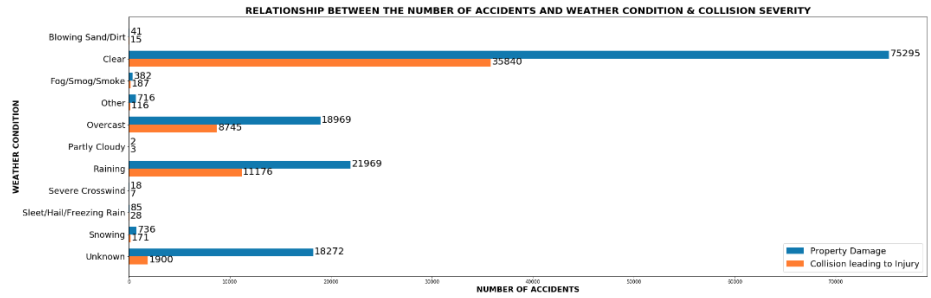
After generating these graphs, 2 main relationships have also been determined:

- The relationship between number of accidents and weather conditions & collision severity
- The relationship between number of accidents and road conditions & collision severity

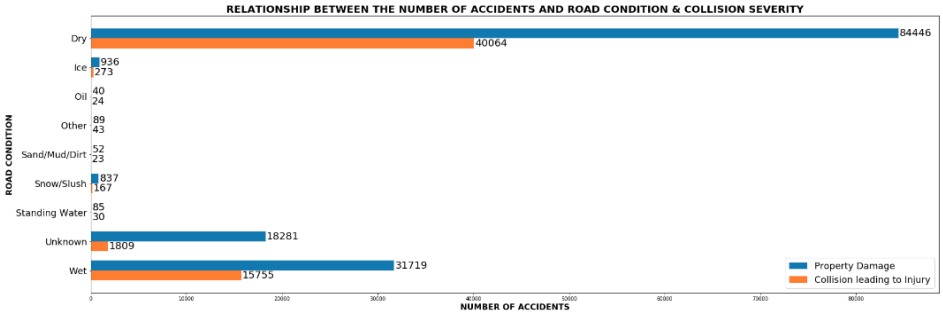
The same code was used for both, but the terms ‘WEATHER’ and ‘WEATHERCOND’ have been replaced with ‘ROAD’ and ‘ROADCOND’.

```
In [100]: #Relationship between number of accidents and weather conditions & collision severity
labels = sorted(df_data_1['WEATHER'].unique())
property_damage_Only_Collision = list(df_data_1.SEVERITYCODE.eq(0).astype('int').groupby(df_data_1.WEATHER,sort=True).sum())
Injury_Collision = list(df_data_1.SEVERITYCODE.eq(1).astype('int').groupby(df_data_1.WEATHER,sort=True).sum())
y = np.arange(len(labels))  # the label locations
height = 0.36  # the width of the bars

fig, ax = plot.subplots()

rects1 = ax.barh(y - height/2, property_damage_Only_Collision, height, label='Property Damage')
rects2 = ax.barh(y + height/2, Injury_Collision, height, label='Collision leading to Injury')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('NUMBER OF ACCIDENTS',size=16,fontweight='bold')
ax.set_ylabel('WEATHER CONDITION',size=16,fontweight='bold')
ax.set_title('RELATIONSHIP BETWEEN THE NUMBER OF ACCIDENTS AND WEATHER CONDITION & COLLISION SEVERITY',size=20,fontweight='bo
ld')
ax.set_yticks(y)
ax.set_yticklabels(labels,fontsize=18)
ax.legend(loc=4,prop={'size': 18})

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_width()+100, i.get_y()+.27,\
            str(i.get_width()), fontsize=20, color='black',size=20,)

ax.invert_yaxis()
fig.set_size_inches(30,10)
plot.show()
```

The relationship between number of accidents and weather conditions & collision severity



The relationship between number of accidents and road conditions & collision severity



After generating these relational graphs, the ***Data Preparation*** phase was started and the following was achieved:

- Encoding and replacing each of the elements' subsections into values for grouping purposes
- The data frame conditions have been set (X)
- The SEVERITYCODE has been set on the axis (Y)
- Preprocessing and splitting was completed using the following codes
    - Preprocessing
        - `from sklearn import preprocessing`
        - `X= preprocessing.StandardScaler().fit(X).transform(X)`
        - `X[0:5]`
    - Splitting
        - `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)`

Results

The results section is catered towards the models formed through Machine Learning techniques; K-Nearest Neighbor, Logistical Regression, and Decision Tree models.

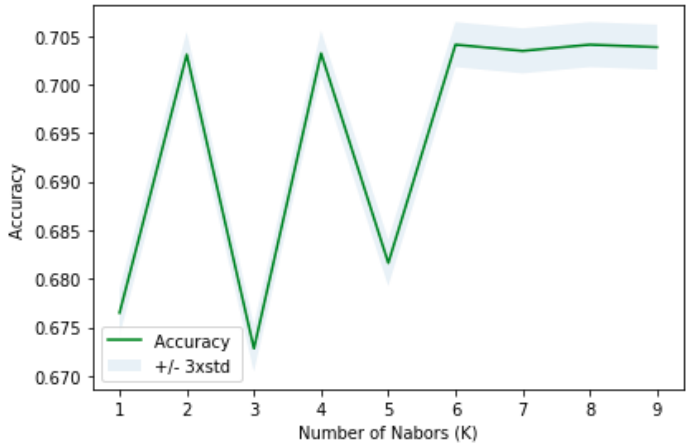
K-Nearest Neighbor

The following lines of code were used:

```
In [37]: import itertools
         import numpy as np
         import matplotlib.pyplot as plt
         from matplotlib.ticker import NullFormatter
         import pandas as pd
         import numpy as np
         import matplotlib.ticker as ticker
         from sklearn import preprocessing
         %matplotlib inline

In [38]: from sklearn.neighbors import KNeighborsClassifier

In [39]: k = 4
         #Train Model and Predict
         neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
         neigh

Out[39]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                    weights='uniform')

In [40]: yhat = neigh.predict(X_test)
         yhat[0:5]

Out[40]: array([1, 1, 1, 1, 1])

In [41]: #Accuracy Evaluation
         from sklearn import metrics
         print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
         print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))

         Train set Accuracy:  0.6993925695719735
         Test set Accuracy:  0.7031976370874534
```

```
In [42]: #Calculated accuracy of KNN for different Ks
         Ks = 10
         mean_acc = np.zeros((Ks-1))
         std_acc = np.zeros((Ks-1))
         ConfustionMx = [];
         for n in range(1,Ks):

             #Train Model and Predict
             neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
             yhat=neigh.predict(X_test)
             mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


             std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

         mean_acc

Out[42]: array([0.67648645, 0.70306922, 0.67281366, 0.70319764, 0.68162322,
                0.70409657, 0.70345448, 0.70409657, 0.70383973])

In [43]: print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)

         The best accuracy was with 0.7040965712084243 with k= 6

In [44]: #Plot
         plt.plot(range(1,Ks),mean_acc,'g')
         plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
         plt.legend(('Accuracy ', '+/- 3xstd'))
         plt.ylabel('Accuracy ')
         plt.xlabel('Number of Nabors (K)')
         plt.tight_layout()
         plt.show()
```

The Model



Average F-1 Score and Jaccard Score

In [69]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

KnnJaccard = jaccard_similarity_score(y_test, yhat)
KnnF1 = f1_score(y_test, yhat, average='weighted')
print("K-Nearest Neighbors Average F1-score: %.7f" % KnnF1 )
print("K-Nearest Neighbors Jaccard Score: %.7f" % KnnJaccard )
```

```
K-Nearest Neighbors Average F1-score: 0.5822889
K-Nearest Neighbors Jaccard Score: 0.7043534
```

Logistical Regression

The following lines of code were used:

In [45]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

In [48]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[48]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [49]:
```
yhat = LR.predict(X_test)
yhat
```

Out[49]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [50]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[50]:
```
array([[0.68356804, 0.31643196],
       [0.67078904, 0.32921096],
       [0.67593184, 0.32406816],
       ...,
       [0.70746524, 0.29253476],
       [0.68269417, 0.31730583],
       [0.68356804, 0.31643196]])
```

In [51]:
```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

Out[51]:
```
0.7043790933607295
```

In [52]:
```
#Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

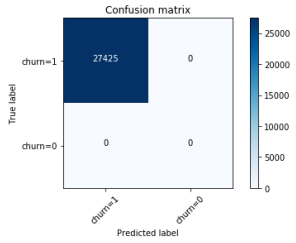
```
[[27425     0]
 [    0     0]]
```

In [53]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[27425     0]
 [    0     0]]
```

The Model & Log-Loss

```
Confusion matrix, without normalization
[[27425     0]
 [    0     0]]
```




```
In [54]: from sklearn.metrics import log_loss
         log_loss(y_test, yhat_prob)
Out[54]: 0.6068093485491171
```

Decision Tree Model & Accuracy

```
In [70]: ###Decision Tree

In [72]: from sklearn.tree import DecisionTreeClassifier
         dtc = DecisionTreeClassifier(criterion="entropy", max_depth = 9)
         dtc.fit(X_train, y_train)
Out[72]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=9,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')

In [73]: yhat_dtc = dtc.predict(X_test)

In [75]: # Accuracy:
         len(yhat_dtc[yhat_dtc==y_test])/len(y_test)
Out[75]: 0.7043534095287017
```

Discussion & Conclusion

Based on all the results generated, the Decision tree model displayed an accuracy of 0.70435, while the KNN gave off an Average F-1 score of 0.58 and a Jaccard score of 0.70435 and logistical regression also gave the same Jaccard score, similar to the Decision tree's accuracy and that of the KNN.

The graphs showed an important relationship between the number of accidents and road conditions and severity of collision. On 'DRY' road conditions, there was an estimate of 84,446 collisions that lead to property damage and 40,064 that lead to injuries; the 'DRY' road served the highest number of accidents. Taking a look at the weather conditions, 'CLEAR' weather resulted in the highest number of accidents with 75,295 leading to property damage and 35,840 leading to injuries.

Based on all the collisions that took place, 70.1% of them lead to property damage while 29.9% lead to injuries.

To conclude, the results give beneficial insights on the severity of vehicle collisions with focus on the different factors such as weather, road and light conditions. In terms of modeled results and values, they can be improved with more accurate and precise input data.