Predicting Car Accident Severity

1.0. Introduction

Road traffic accidents are common safety problem around the world. This automotive accidents result in over result in over 30,000 fatalities in the United States annually. Road traffic accidents are estimated to cost the US economy approximately $810bn per. Identifying the factors which influence accident severity is therefore of paramount importance.

In an effort to reduce the frequency of car collisions in our community, this project will leverage existing accident data to predict the different accidents' severity given the current weather, road and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful.I will use different supervised machine learning algorithms and select the machine learning model that gives the highest prediction accuracy.

The study of building this kind of model will be of significance to a lot of stakeholders and beneficiaries: (1) town/city planners, who may be able to use the model to inform their road planning and traffic calming strategies; (2) emergency service responders, who may be able to use the model to predict the severity of an accident based on information that's provided at the time the accident is reported in order to optimally allocate resources across the city, and (3) traffic police officers.

2.0. Data

A comprehensive dataset of over 190,000 observations collected occurring between 2004–2019 in the Seattle city area was obtained from the Seattle Open Data Portal. The dataset has almost 40 columns describing the details of each accident including the weather conditions, collision type, date/time of accident and location. To accurately build a model to prevent future accidents and/or reduce their severity, we will use the following attributes — ADDRTYPE, WEATHER, ROADCOND, VEHCOUNT, PERSONCOUNT.

Each entry contains information regarding an accident incident, generally including information on:

- **Severity of the accident:** This includes severity class/code, severity description
- **Location of the accident:** This includes (X, y) coordinates, address, location type, junction type
- **Date-Time of the accident:** This includes the date and the time
- **Environment conditions:** This includes the weather, road surface conditions, lighting conditions
- **Parties involved:** This includes the number of pedestrians, vehicles, cyclists involved
- **Event information:** This includes information on the type of collision, the description of the collision and if the vehicle was speeding

3.0. Methodology

The following will be covered:

- Exploratory data analysis
- Inferential statistical testing
- Machine learnings modeling

4.0. Exploratory Data Analysis

Data Cleaning

Firstly, the selected data *INCDTTM* and *INCDATE* column values were converted to the appropriate datetime format, for better manipulation. This generated a number of missing values, in particular from the *INCDTTM* where there are a lot of missing time values. We decided to keep these entries during the EDA, but eventually we remove these missing value entries when we eventually decided that the time feature was of importance to our analysis, and that the SEVERITYCODE distribution within the entries with missing time values were similar to the overall dataset true distribution.

To resolve the other missing values coming from *ROADCOND, LIGHTCOND, WEATHER*, we used the filtered out those that had 'Unmatched' values in the *STATUS* column as these had a high proportion of missing *ROADCOND, LIGHTCOND, WEATHER* values. For the remaining entries with null values, they constitute a small proportion <1% of the total dataset and were also removed.

For the remaining missing values in *ADDRTYPE, JUNCTIONTYPE*, we found that a proportion of them had missing values in both *ADDRTYPE, JUNCTIONTYPE* and these were dropped as they will not provide much value. We would that a large proportion of the remaining missing values in the *JUNCTIONTYPE* column contain the *ADDRTYPE* 'Block' value. As it is unknown what the led to the missing values, we decided to relabel all remaining missing values in these two columns as 'Others.

Data Processing

As we've kept a couple of categorical features, we utilized both Label Encoding during the EDA to visualize data correlation, and One-Hot Encoding on the final feature set so that they can be input into the model. To enable ease of manipulation, the feature names were all also converted to lowercase.

We created a *day*, *month* and *hour* feature from the *incdate* and *incdttm* features for EDA and model buildings. Other features such as *severitycode*, *hitparkedcar* were all relabelled to appropriate values of 0 and 1 according.

Inferential Statistical Testing - Understand the Correlation

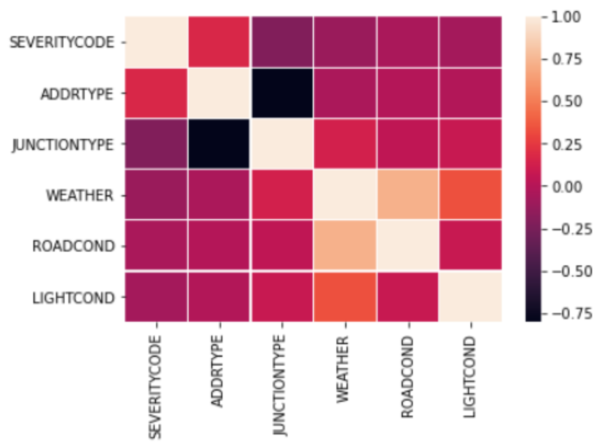
```
In [21]: df_sev.corr()
```

```
Out[21]:
```

	SEVERITYCODE	ADDRTYPE	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND
SEVERITYCODE	1.000000	0.172032	-0.216118	-0.112507	-0.045574	-0.067817
ADDRTYPE	0.172032	1.000000	-0.804215	-0.038823	-0.002626	-0.013375
JUNCTIONTYPE	-0.216118	-0.804215	1.000000	0.134021	0.048077	0.077417
WEATHER	-0.112507	-0.038823	0.134021	1.000000	0.728901	0.339707
ROADCOND	-0.045574	-0.002626	0.048077	0.728901	1.000000	0.083132
LIGHTCOND	-0.067817	-0.013375	0.077417	0.339707	0.083132	1.000000

```
In [22]: sns.heatmap(df_sev.corr(), linewidth=.2, cbar_kws={"shrink": 1})
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x15e0d9948b0>
```



5.0. Modeling

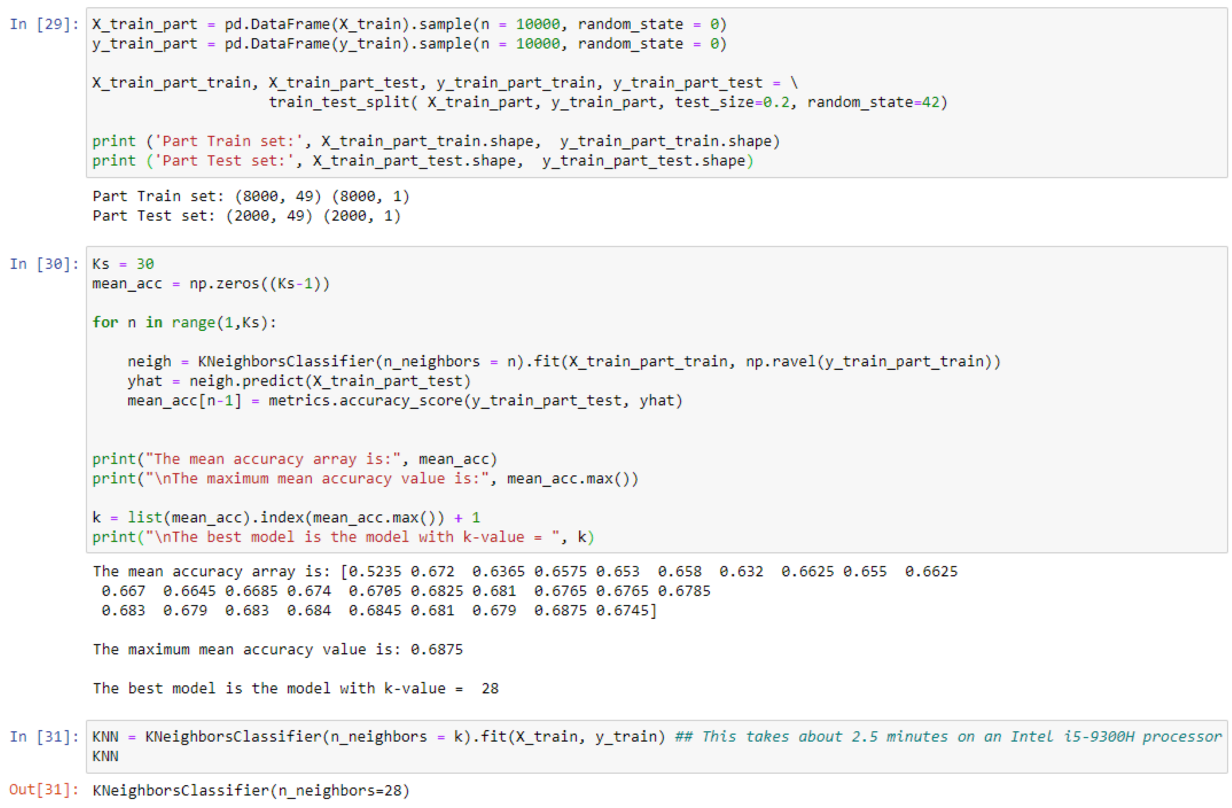
Our data is now ready to be fed into machine learning models.

We will use the following models:

1. KNN (K — Nearest Neighbors)
2. Decision Tree
3. Logistic regressions

K-Nearest Neighbor (KNN)

KNN will help us predict the severity code of an outcome by finding the most similar to data point within k distance. To build KNN algorithm, we have taken it for set of 30 values the model will be trained on training set of data, and then predicting the values based on test data set.



```
In [29]: X_train_part = pd.DataFrame(X_train).sample(n = 10000, random_state = 0)
         y_train_part = pd.DataFrame(y_train).sample(n = 10000, random_state = 0)

         X_train_part_train, X_train_part_test, y_train_part_train, y_train_part_test = \
                             train_test_split( X_train_part, y_train_part, test_size=0.2, random_state=42)

         print ('Part Train set:', X_train_part_train.shape,  y_train_part_train.shape)
         print ('Part Test set:', X_train_part_test.shape,  y_train_part_test.shape)

         Part Train set: (8000, 49) (8000, 1)
         Part Test set: (2000, 49) (2000, 1)

In [30]: Ks = 30
         mean_acc = np.zeros((Ks-1))

         for n in range(1,Ks):

             neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train_part_train, np.ravel(y_train_part_train))
             yhat = neigh.predict(X_train_part_test)
             mean_acc[n-1] = metrics.accuracy_score(y_train_part_test, yhat)


         print("The mean accuracy array is:", mean_acc)
         print("\nThe maximum mean accuracy value is:", mean_acc.max())

         k = list(mean_acc).index(mean_acc.max()) + 1
         print("\nThe best model is the model with k-value = ", k)

         The mean accuracy array is: [0.5235 0.672  0.6365 0.6575 0.653  0.658  0.632  0.6625 0.655  0.6625
          0.667  0.6645 0.6685 0.674  0.6705 0.6825 0.681  0.6765 0.6765 0.6785
          0.683  0.679  0.683  0.684  0.6845 0.681  0.679  0.6875 0.6745]

         The maximum mean accuracy value is: 0.6875

         The best model is the model with k-value =  28

In [31]: KNN = KNeighborsClassifier(n_neighbors = k).fit(X_train, y_train) ## This takes about 2.5 minutes on an Intel i5-9300H processor
         KNN

Out[31]: KNeighborsClassifier(n_neighbors=28)
```

s per the above model the most accuracy we have for when K=19 which is 0.7. So I trained my model with the K-Value =19.

Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. It context, the decision tree observes all possible outcomes of different weather conditions.

In [32]:
```
X_train_train, X_train_test, y_train_train, y_train_test = \
                    train_test_split( X_train, y_train, test_size=0.2, random_state=0)

print ('Part Train set:', X_train_train.shape,  y_train_train.shape)
print ('Part Test set:', X_train_test.shape,  y_train_test.shape)
```

```
Part Train set: (124590, 49) (124590,)
Part Test set: (31148, 49) (31148,)
```

In [33]:
```
depth = 21
mean_acc1 = np.zeros((depth - 4))

for n in range(4, depth):

    tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = depth).fit(X_train_train, y_train_train)
    yhat = tree.predict(X_train_test)
    mean_acc1[n-4] = metrics.accuracy_score(y_train_test, yhat)


print("The mean accuracy array is:", mean_acc1)
print("\nThe maximum mean accuracy value is:", mean_acc1.max())

max_depth = list(mean_acc1).index(mean_acc1.max()) + 4
print("\nThe best model is the model with max_depth = ", max_depth)
```

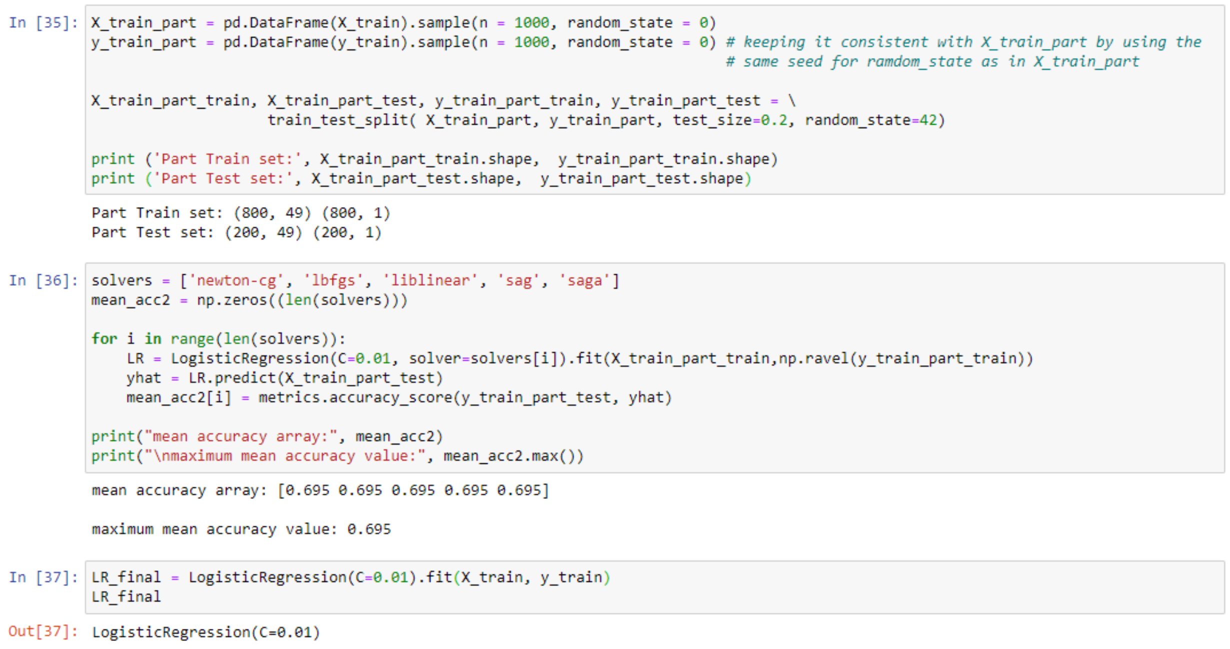
```
The mean accuracy array is: [0.69853602 0.69847181 0.69850392 0.69850392 0.69853602 0.69853602
 0.69853602 0.69847181 0.69853602 0.69860023 0.69853602 0.69850392
 0.69843971 0.69847181 0.69843971 0.69843971 0.69853602]

The maximum mean accuracy value is: 0.6986002311544882

The best model is the model with max_depth =  13
```

In [34]:
```
severityTree = DecisionTreeClassifier(criterion = 'entropy', max_depth = max_depth).fit(X_train, y_train)
severityTree
```

Out[34]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=13)
```

Logistic Regression

Because our dataset only provides us with two severity code outcomes, our model will only predict one of those two classes. This makes our data binary, which is perfect to use with logistic regression.



```
In [35]: X_train_part = pd.DataFrame(X_train).sample(n = 1000, random_state = 0)
         y_train_part = pd.DataFrame(y_train).sample(n = 1000, random_state = 0) # keeping it consistent with X_train_part by using the
                                                                                  # same seed for ramdom_state as in X_train_part

         X_train_part_train, X_train_part_test, y_train_part_train, y_train_part_test = \
                             train_test_split( X_train_part, y_train_part, test_size=0.2, random_state=42)

         print ('Part Train set:', X_train_part_train.shape,  y_train_part_train.shape)
         print ('Part Test set:', X_train_part_test.shape,  y_train_part_test.shape)

         Part Train set: (800, 49) (800, 1)
         Part Test set: (200, 49) (200, 1)

In [36]: solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
         mean_acc2 = np.zeros((len(solvers)))

         for i in range(len(solvers)):
             LR = LogisticRegression(C=0.01, solver=solvers[i]).fit(X_train_part_train,np.ravel(y_train_part_train))
             yhat = LR.predict(X_train_part_test)
             mean_acc2[i] = metrics.accuracy_score(y_train_part_test, yhat)

         print("mean accuracy array:", mean_acc2)
         print("\nmaximum mean accuracy value:", mean_acc2.max())

         mean accuracy array: [0.695 0.695 0.695 0.695 0.695]

         maximum mean accuracy value: 0.695

In [37]: LR_final = LogisticRegression(C=0.01).fit(X_train, y_train)
         LR_final

Out[37]: LogisticRegression(C=0.01)
```

The main targeted feature would be the *SEVERITYCODE*, which represents the two possible state/class of the accidents - (0) Property damage only, (1) Injury. Given that this is a binary classification problem, we will explore the use of Logistic Regression (LR) and the ensemble Random Forest Classification (RF) models. Gradient Boosting Classifiers (GBC) which were found to normally provide superior results for binary and multi-class classification tasks will be implemented.

6.0. Conclusion

Most crashes happened in clear, dry, and bright conditions. Most days are clear, dry, and bright, so it’s no surprise that most car crashes occur under these conditions. I also found out that crashes with a distracted driver or an impaired driver are statistically more likely to result in injury, which is also not a surprise.

Based on the above values I would conclude the model gives the better results with the KNN algorithm almost with the Accuracy score with 0.7.

Summary of the Accuracy scores of the different models

In [41]:
```
scores_dict = {'': ['K Nearest Neighbors','Decision Tree','Logistic Regression'], \
               'Jaccard Score': [KNN_jacc, DT_jacc, LR_jacc], \
               'F1-score': [KNN_f1, DT_f1, LR_f1], \
               'Subset Accuracy Score': [KNN_acc, DT_acc, LR_acc], \
               'Log Loss': ['NA','NA', LR_logloss] }

scores_Report = pd.DataFrame.from_dict(scores_dict)

scores_Report.set_index('', drop = True, inplace = True)

scores_Report
```

Out[41]:

	Jaccard Score	F1-score	Subset Accuracy Score	Log Loss
K Nearest Neighbors	0.500935	0.599115	0.699114	NA
Decision Tree	0.496902	0.584641	0.703814	NA
Logistic Regression	0.496086	0.582288	0.704199	0.56721

We see an improvement in the Jaccard and Recall scores of all models after rebalancing the training dataset. With a balanced dataset, the GBC model has improved and performed better in all aspects than the RF model, with the LR model close in predictive performance. A closer look into the classification reports above show that most model suffer from poor precision for the severe case, i.e. when *severitycode* is 1, i.e. there is a tendency for the model to make False Positives.

The results of the data indicate to city officials that they should ask drivers to be more alert in ideal conditions