Coursera_Capstone : Car Accident Severity(Seattle)

Hossain Mohammad Amran

Introduction:

Road accidents are an unstoppable problem in our life and societies. Death on the road is a big puzzle in the current world. According to the World Health Organization(WHO), an estimated 1.35 million deaths worldwide were related to road traffic injuries in 2016 [1] .The death rate has remained constant, compared to the world's population. Every 25 seconds, one person died in road accidents. In 2016, 34,436 motor vehicle crashes were recorded in the USA and 37,461 people died with an average of 102 per day [2].
In this work, I will try to find out the way to reduce the number of road accidents, by developing a machine learning model to predict the severity of accidents. When the road visibility and weather conditions are changing this model will alert the car user.

Business Understanding:

The local government of Seattle is trying to implement some method to alert the car user, police , traffic system and health system about critical situations to reduce the death and injuries on the road.
The final capstone project in the IBM certificate course, we want to analyze the accident “severity” in terms of human fatality, traffic delay, property damage, or any other type of accident bad impact. The data was collected by Seattle SPOT Traffic Management Division and provided by Coursera via a link. This dataset is updated weekly and is from 2004 to present. It contains information such as severity code, address type, location, collision type, weather, road condition, speeding, among others.
The target audience of the project is local Seattle government, police, rescue groups, and last but not least, car insurance institutes. The model and its results are going to provide some advice for the target audience to make insightful decisions for reducing the number of accidents and injuries for the city.

Data Understanding:

The dataset we select has 194,673 rows and 37 different independent variables.We will use SEVERITY CODE as our dependent variable Y, with different independent variables X to identify the cause of road accidents and level of severity.The dataset are quite large, we need to filter out the missing value and delete the unrelated columns. Then we select the independent variables such as address type, weather, road condition, and light condition to compare with Y which may have more impact on the accidents.The dependent variable, “SEVERITYCODE”, contains numbers that correspond to different levels of severity caused by an accident .
The code that corresponds to the severity of the collision:

- 3—fatality
- 2b—serious injury
- 2—injury
- 1—prop damage
- 0—unknown

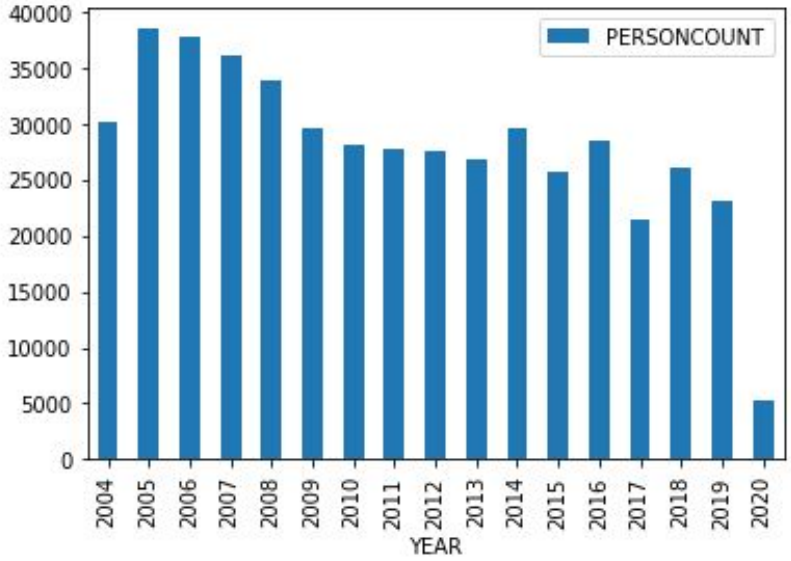
Other important variables include:

- ADDRTYPE: Collision address type: Alley, Block, Intersection
- LOCATION: Description of the general location of the collision
- PERSONCOUNT: The total number of people involved in the collision helps identify severity involved
- PEDCOUNT: The number of pedestrians involved in the collision helps identify severity involved
- PEDCYLCOUNT: The number of bicycles involved in the collision helps identify severity involved
- VEHCOUNT: The number of vehicles involved in the collision identify severity involved
- INCDTTM : The date and time of the incident.
- JUNCTIONTYPE: Category of junction at which collision took place helps identify where most collisions occur
- WEATHER: A description of the weather conditions during the time of the collision
- ROADCOND: The condition of the road during the collision
- LIGHTCOND: The light conditions during the collision
- SPEEDING: Whether or not speeding was a factor in the collision (Y/N)
- SEGLANEKEY: A key for the lane segment in which the collision occurred
- CROSSWALKKEY: A key for the crosswalk at which the collision occurred
- HITPARKEDCAR: Whether or not the collision involved hitting a parked car

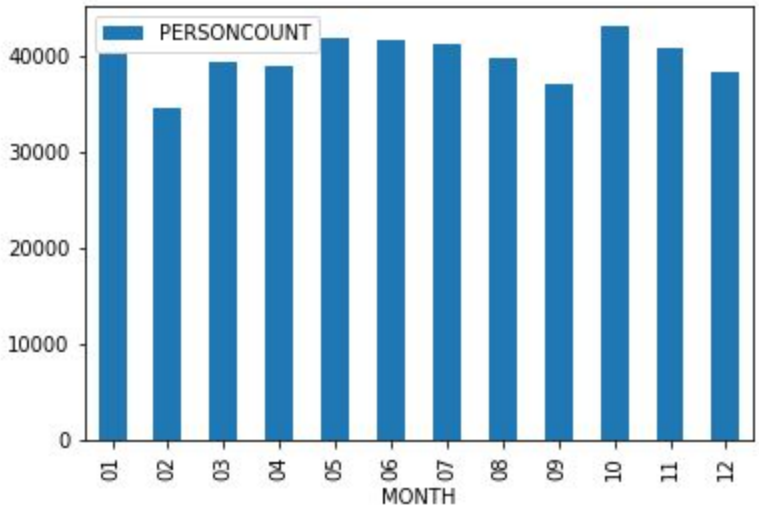
Furthermore, because of the existence of null values in some records, the data needs to be preprocessed before any further processing.

Data Preparation:

The original data are not ready for our desired model. We have to prepare the data set to build a perfect model. We have to drop some non-relevant columns, which will not be useful to model. We will drop some null values as missing data info. We update some missing/null value with the expected value. We replace the value with meaningful value. We will update date time as week of day. Time as a busy rush . Here we check the number of people affected by car accidents per year. We can told number of accidents are decreasing but we can't conclude it.

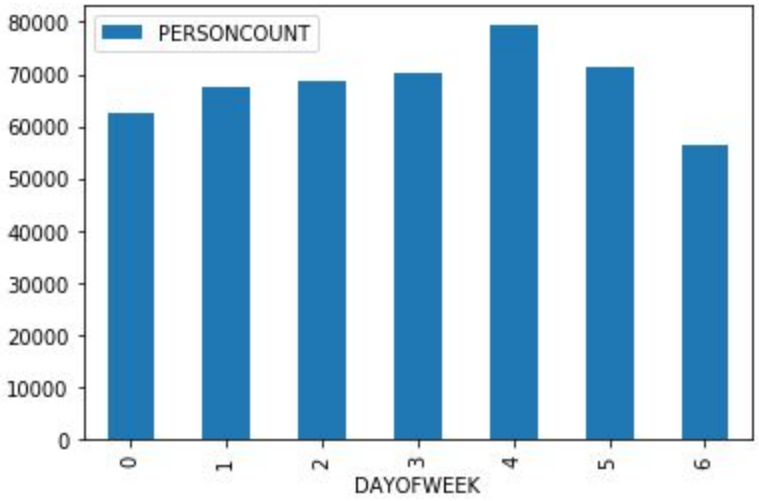


We also check the month to find out which have more incidents. January and October have more number of incidents compared to other months.



We also check the day of week. As we can see on graph drive on Friday there is more risk. Drivers should be more careful on this day. Here days are

- 0 #Monday
- 1 #Tuesday
- 2 #Wednesday
- 3 #Thursday
- 4 #Friday
- 5 #Saturday
- 6 #Sunday



In addition, most of the features are of object data types that need to be converted into numerical data types.
After analyzing the data set, I have decided to focus on only eight features, severity, weather conditions, road conditions, light conditions, Address Type, collision Type, the number of pedestrians and bicycles involved in the collision. among others.
To get a good understanding of the dataset, I have checked different values in the features. The results show the target feature is imbalance, so we use a simple statistical technique to balance it.

```
In [16]: #we use a simple statistical technique to balance the target feature "SEVERITYCODE".
         collision_df['SEVERITYCODE'].value_counts()

Out[16]: 1    125171
         2     56389
         Name: SEVERITYCODE, dtype: int64
```

As We can see, the number of rows in class 1 is almost three times bigger than the number of rows in class 2. It is possible to solve the issue by downsampling the class 1.

```
In [17]: #the value of 1 is almost three times bigger than the value of 2. It is possible to solve the issue by downsampling the class 1
         from sklearn.utils import resample
         collision_df_max = collision_df[collision_df.SEVERITYCODE ==1]
         collision_df_min = collision_df[collision_df.SEVERITYCODE ==2]
         collision_df_max_ds = resample(collision_df_max,
                                        replace = False,
                                        n_samples= 56389,
                                        random_state = 123
                                       )

         collision_df_balance = pd.concat([collision_df_max_ds,collision_df_min])
         collision_df_balance.SEVERITYCODE.value_counts()

Out[17]: 2    56389
         1    56389
         Name: SEVERITYCODE, dtype: int64
```

Modeling:

I have used watson studio, Github as a repository and running Jupyter Notebook to preprocess data and build Machine Learning models. Regarding coding, I have used Python and some popular packages such as Pandas, NumPy and Sklearn.
Once I have load data into Pandas Dataframe, used ‘dtypes’ attribute to check the feature names and their data types. Then I have selected the most important features to predict the

severity of accidents in Seattle. Among all the features, the following features have the most influence in the accuracy of the predictions:
‘WEATHER’,
‘ROADCOND’,
‘LIGHTCOND’,
'ADDRTYPE',
'COLLISIONTYPE',
'JUNCTIONTYPE',
'PEDCOUNT',
'PEDCYLCOUNT'
As mentioned earlier, “SEVERITYCODE” is the target variable.
I have run a value count on road (‘ROADCOND’) and weather condition (‘WEATHER’) to get ideas of the different road and weather conditions. I also have run a value count on light conditions (’LIGHTCOND’), to see the breakdowns of accidents occurring during the different light conditions.

```
In [18]: collission_df['WEATHER'].value_counts()

Out[18]: 1.0    108766
         2.0     60568
         0.0     12226
         Name: WEATHER, dtype: int64

In [19]: collission_df['ROADCOND'].value_counts()

Out[19]: 1.0    122002
         2.0     46671
         3.0     11479
         4.0      1176
         6.0       108
         7.0        64
         8.0        60
         Name: ROADCOND, dtype: int64

In [20]: collission_df['LIGHTCOND'].value_counts()

Out[20]: 1.0    113306
         2.0     47028
         3.0     10277
         4.0      5744
         5.0      2420
         6.0      1432
         7.0      1142
         8.0       200
         9.0        11
         Name: LIGHTCOND, dtype: int64
```

Also run value counts on the features 'ADDRTYPE', 'COLLISIONTYPE', 'JUNCTIONTYPE', 'PEDCOUNT', 'PEDCYLCOUNT' for check the breakdown of those feature

```
In [27]: collission_df['ADDRTYPE'].value_counts()
Out[27]: 1.0    118484
         2.0     63076
         Name: ADDRTYPE, dtype: int64

In [28]: collission_df['COLLISIONTYPE'].value_counts()
Out[28]: 1.0     42534
         2.0     34339
         3.0     33522
         4.0     22619
         5.0     18239
         6.0     13607
         9.0      6470
         10.0     5352
         7.0      2919
         8.0      1959
         Name: COLLISIONTYPE, dtype: int64

In [29]: collission_df['JUNCTIONTYPE'].value_counts()
Out[29]: 2.0    85731
         1.0    60997
         3.0    22208
         4.0    10418
         5.0     2042
         6.0      159
         7.0        5
         Name: JUNCTIONTYPE, dtype: int64

In [30]: collission_df['PEDCOUNT'].value_counts()
Out[30]: 0    174762
         1      6545
         2       225
         3        22
         4         4
         6         1
         5         1
         Name: PEDCOUNT, dtype: int64

In [33]: collission_df['PEDCYLCOUNT'].value_counts()
Out[33]: 0    176155
         1      5364
         2        41
         Name: PEDCYLCOUNT, dtype: int64
```

After balancing SEVERITYCODE feature, and standardizing the input feature, the data has been ready for building machine learning models.
WE have employed four machine learning models:

- K Nearest Neighbour (KNN)
- Decision Tree
- Linear Regression

- ❖ Support Vector Machine (SVM)

We define our 70% of data as train and 30% as test data set :

Define Feature data set for X

```
In [30]: ft_df = collission_df_balance[['WEATHER','ROADCOND','LIGHTCOND','ADDRTYPE','COLLISIONTYPE','JUNCTIONTYPE','PEDCOUNT','PEDCYLCOUNT']]
         X = ft_df
         X[0:5]
```

Out[30]:

	WEATHER	ROADCOND	LIGHTCOND	ADDRTYPE	COLLISIONTYPE	JUNCTIONTYPE	PEDCOUNT	PEDCYLCOUNT
23477	2.0	1.0	2.0	2.0	6.0	1.0	0	0
130631	1.0	1.0	1.0	1.0	3.0	3.0	0	0
96633	2.0	1.0	1.0	2.0	2.0	1.0	0	0
52708	2.0	2.0	1.0	1.0	3.0	3.0	0	0
77256	1.0	1.0	1.0	1.0	4.0	2.0	0	0

```
In [31]: y = collission_df_balance['SEVERITYCODE'].values
         y[0:5]
```

```
Out[31]: array([1, 1, 1, 1, 1])
```

Train-Test Split

```
In [32]: #import train_test_split
         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=4)
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (78944, 8) (78944,)
Test set: (33834, 8) (33834,)
```

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 and shown the results as follow:

K-Nearest Neighbor (KNN)

KNN will help us predict the severity code of an outcome by finding the most similar to the data point within k distance.

```
In [34]: from sklearn.neighbors import KNeighborsClassifier
         k = 4
         neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
         yhat = neigh.predict(X_test)
         yhat[0:5]
```

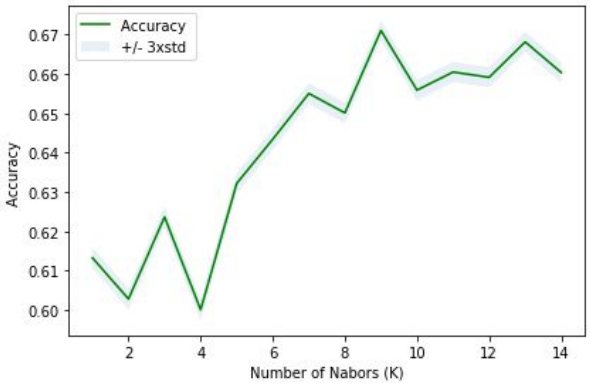
```
Out[34]: array([1, 2, 1, 2, 2])
```

```
In [35]: from sklearn import metrics
         Ks = 15
         mean_acc = np.zeros((Ks-1))
         std_acc = np.zeros((Ks-1))
         ConfustionMx = [];
         for n in range(1,Ks):

             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[35]: array([0.61328841, 0.60285512, 0.62366259, 0.6001064 , 0.63217474,
                0.64340604, 0.65502158, 0.65008571, 0.67101141, 0.6558787 ,
                0.66045989, 0.65912987, 0.66811491, 0.66034167])
```

The Best accuracy of KNN model with K = 9

```
In [36]: 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()

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




```
The best accuracy was with 0.6710114086421942 with k= 9
```

model with K = 9

```
In [37]: k = 9
         neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
         yhat = neigh.predict(X_test)
         yhat[0:5]

Out[37]: array([1, 2, 2, 2, 2])
```

KNN Model Evaluation

Knn Jaccard and F1 score are below:

```
In [34]: from sklearn.metrics import jaccard_similarity_score
         from sklearn.metrics import f1_score
         from sklearn.metrics import log_loss
         from sklearn.metrics import accuracy_score
```

```
In [35]: #Predict y
         knn_yhat = neigh.predict(X_test)

         # jaccard
         knn_jaccard = jaccard_similarity_score(y_test, knn_yhat)
         print("KNN Jaccard index: ", knn_jaccard)


         # f1_score
         knn_f1_score = f1_score(y_test, knn_yhat, average='weighted')
         print("KNN F1-score: ", knn_f1_score)
```

```
KNN Jaccard index:  0.6710114086421942
KNN F1-score:  0.670738826660399
```

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 [36]:
```
# import library DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

# create DecisionTreeClassifier as named d_tree
d_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 7)
#train
d_tree.fit(X_train,y_train)
```

Out[36]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=7,
                       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 [37]:
```
dt_yhat = d_tree.predict(X_test)
dt_yhat
```

Out[37]:
```
array([1, 2, 2, ..., 1, 2, 1])
```

Decision tree evaluation

Decision tree jaccard and f1 score are :

In [38]:
```
# jaccard
dt_jaccard = jaccard_similarity_score(y_test, dt_yhat)
print("DT Jaccard index: ", dt_jaccard)

# f1_score
dt_f1_score = f1_score(y_test, dt_yhat, average='weighted')
print("DT F1-score: ", dt_f1_score)
```

```
DT Jaccard index:  0.6980847668026245
DT F1-score:  0.6970328787447123
```

Support Vector Machine (SVM)

SVM will provide us with the Hyperline of our dataset.

In [39]:
```
# import SVM library
from sklearn import svm

clf = svm.SVC()
clf.fit(X_train, y_train)
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/bas
caled features. Set gamma explicitly to 'auto' or 'scale' to avoid t
  "avoid this warning.", FutureWarning)
```

Out[39]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
In [40]: svm_yhat = clf.predict(X_test)
         svm_yhat

Out[40]: array([1, 2, 2, ..., 1, 2, 1])
```

SVM evaluation

SVM jaccard and f1 score are:

```
In [41]: # jaccard
         svm_jaccard = jaccard_similarity_score(y_test, svm_yhat)
         print("SVM Jaccard index: ", svm_jaccard)

         # f1_score
         svm_f1_score = f1_score(y_test, svm_yhat, average='weighted')
         print("SVM F1-score: ", svm_f1_score)

         SVM Jaccard index:  0.6956611692380446
         SVM F1-score:  0.6930330013606999
```

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 [42]: #import Library LogisticRegression
         from sklearn.linear_model import LogisticRegression

         #Train
         lr = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
         lr

Out[42]: LogisticRegression(C=6, 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 [43]: lr_yhat = lr.predict(X_test)
         lr_yhat

Out[43]: array([2, 1, 2, ..., 1, 1, 1])
```

Logistic regression evaluation

Logistic regression jaccard , f1, and log loss score:

In [44]:
```
lr_yhat_prob = lr.predict_proba(X_test)

# jaccard
lr_jaccard = jaccard_similarity_score(y_test, lr_yhat)
print("LR Jaccard: ", lr_jaccard)

# f1_score
lr_f1_score = f1_score(y_test, lr_yhat, average='weighted')
print("LR F1-score: ", lr_f1_score)

# logloss
lr_logloss = log_loss(y_test, lr_yhat_prob)
print("LR log loss: ", lr_logloss)
```

```
LR Jaccard:  0.6304013714015487
LR F1-score:  0.6285955849476118
LR log loss:  0.6142731759680288
```

Result

The algorithms implemented above did not give an accuracy score equal to or greater than 0.7, they all ranged from 0.6 to 0.7. Meaning, these models can predict the severity code of an accident with an accuracy equalling 60–70%. A data frame is created below with the bars representing the accuracy of each model.

In [45]:
```
# create a jaccard list
jaccard_score = [knn_jaccard,dt_jaccard,svm_jaccard,lr_jaccard]

# create a f1-score list
f1_score = [knn_f1_score,dt_f1_score,svm_f1_score, lr_f1_score]

# create a Log_Loss list
log_los = ['NA','NA','NA',lr_logloss]
# fomulate the report format
df_report = pd.DataFrame(jaccard_score, index=['KNN','Decision Tree','SVM','Logistic Regression'])
df_report.columns = ['Jaccard']
df_report.insert(loc=1, column='F1-score', value=f1_score)
df_report.insert(loc=2, column='LogLoss', value=log_los)
df_report.columns.name = 'Algorithm'
df_report
```

Out[45]:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.671011	0.670739	NA
Decision Tree	0.698085	0.697033	NA
SVM	0.695661	0.693033	NA
Logistic Regression	0.630401	0.628596	0.614273

Conclusion

Based on the dataset provided for this capstone from weather, road, light conditions, type of address, collision junction and pedestrians or bicycles can be pointing to certain classes, we can conclude that particular conditions have a somewhat impact on whether or not travel could result in property damage (class 1) or injury (class 2).
The accuracy of the classifiers is not good, highest being 69%. This usually means that the model is under fitted i.e. It needs to be trained on more data. Though the dataset has a lot of variety in terms of scenarios, more volume of the data for such scenarios has to be collected. Certain features with missing values were removed, this reduced the dimensionality of the dataset, these features could have been correlated to other important features but they had to be removed. A better effort has to be made to collect data to reduce the number of missing values.

References:

1. GLOBAL STATUS REPORT ON ROAD SAFETY 2018. `World Health Organization.` https://www.who.int/publications/i/item/global-status-report-on-road-safety-2018
2. NHTSA's Quick Facts 2016 https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812451
   ADDRTYPE: Collision address type: Alley, Block, Intersection