

SEATTLE ACCIDENT COLLISSION ANALYSIS Report

Coursera Capstone Project – Sep 2020

Author

Adeel Naim Khan

Introduction/Business Understanding

- Predicting an accident collision based on the data collected is always challenging aspect of the data science topology. In our daily routine commuting, it is always advisable to take the route which has less blockage or rush due to accidents. Our main purpose here is to build a model which can predict a model to assist the daily commuters avoiding these issues.
- There are many factors which causes the accidents and to predict the pattern using these factors can be very helpful to avoid such incidents. Imagine an accident occurring at a particular place or time many times and by analysis we find the relation of the occurrence of accident with factors which can be avoided or warned against.
- This will be highly beneficial for authorities to take necessary steps and implement a plan for drivers to avoid it from happening in future.

Data Understanding

Here for my Coursera Capstone Project, I have chosen Seattle Collision data for from 2004 to present, which is downloaded from arcgis.com by filtering Seattle.

Using the provided data, I will try to analyze the data based on the **SEVERITY CODE** which ranges from 0 to 3, that includes **Unknown, prop damage, injury, serious injury and fatality** options.

I will analyze the data and propose a model by linking the **SEVERITY CODE** with the **Weather, Road, and Light** conditions at the time of accident and will try to predict the occurrence of future accident for other drivers so it can be avoided.

- I will also try to correlate the accidents severity with other provided factors like **Address type, Road Junctions, alcohol influence, Speeding, Vehicle direction** etc to check if they have any direct or indirect relation to the severity of the accident and whether these attributes makes any effect on the occurrence of accidents

Data Source: http://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0.csv?outSR={%22latestWkid%22:2926,%22wkid%22:2926}

Data Preperation

Loading the data set

First step after loading necessary libraries was to upload the dataset. The dataset I found was in csv format therefore I used dataframe `pd.read_csv` function to load the dataset in `DataSetCol` attribute.

Downloading the required CSV file from [seattlecitygis](http://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0.csv)

```
In [3]: #Loading dataset csv file
DataSetCol = pd.read_csv("http://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0.csv ")
DataSetCol.head()

/opt/conda/envs/Python36/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3020: DtypeWarning: Columns (35) have mixed types.
Specify dtype option on import or set low_memory=False.
interactivity=interactivity, compiler=compiler, result=result)
```

```
Out[3]:
```

	X	Y	OBJECTID	INCKEY	COLDKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	LOCATION	...	ROADCOND	LIGHTCOND	PEDF
0	-122.344896	47.717173	1	1003	1003	3503158	Matched	Block	NaN	AURORA AVE N BETWEEN N 117TH PL AND N 125TH ST	...	Dry	Daylight	Y
1	-122.376467	47.543774	2	56200	56200	1795087	Matched	Block	NaN	35TH AVE SW BETWEEN SW MORGAN ST AND SW HOLLY ST	...	Dry	Dark - Street Lights On	NaN
2	-122.360735	47.701487	3	327037	328537	E979380	Matched	Intersection	37122.0	3RD AVE NW AND NW 100TH ST	...	Wet	Daylight	NaN
3	-122.297415	47.599233	4	327278	328778	E996362	Unmatched	Intersection	30602.0	M L KING JR WAY S AND S JACKSON ST	...	NaN	NaN	NaN

Dataset Description Analysis

As per detailed analysis below, it can be seen the unique, top and most frequent values in the given attributes of the dataset. These details will be very handy when I need to decide about the possible values of empty / other categories

In [20]: DataSetCol.describe(include='all')

Out[20]:

	X	Y	OBJECTID	INCKEY	COLDEKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	LOCATION	...	ROAD
count	213918.000000	213918.000000	221389.000000	221389.000000	221389.000000	221389	221389	217677	71884.000000	216801	...	1950
unique	NaN	NaN	NaN	NaN	NaN	221386	2	3	NaN	25198	...	9
top	NaN	NaN	NaN	NaN	NaN	1780512	Matched	Block	NaN	BATTERY ST TUNNEL NB BETWEEN ALASKAN WY VI NB	Dry
freq	NaN	NaN	NaN	NaN	NaN	2	195232	144917	NaN	298	...	1283
mean	-122.330756	47.620199	110695.000000	144708.701914	144936.934541	NaN	NaN	NaN	37612.330964	NaN	...	NaN
std	0.030055	0.056043	63909.64371	89126.729589	89501.312920	NaN	NaN	NaN	51886.084219	NaN	...	NaN
min	-122.419091	47.495573	1.000000	1001.000000	1001.000000	NaN	NaN	NaN	23807.000000	NaN	...	NaN
25%	-122.349280	47.577151	55348.000000	71634.000000	71634.000000	NaN	NaN	NaN	28652.750000	NaN	...	NaN
50%	-122.330363	47.616053	110695.000000	127184.000000	127184.000000	NaN	NaN	NaN	29973.000000	NaN	...	NaN
75%	-122.311998	47.664290	166042.000000	209783.000000	210003.000000	NaN	NaN	NaN	33984.000000	NaN	...	NaN
max	-122.238949	47.734142	221389.000000	333843.000000	335343.000000	NaN	NaN	NaN	757580.000000	NaN	...	NaN

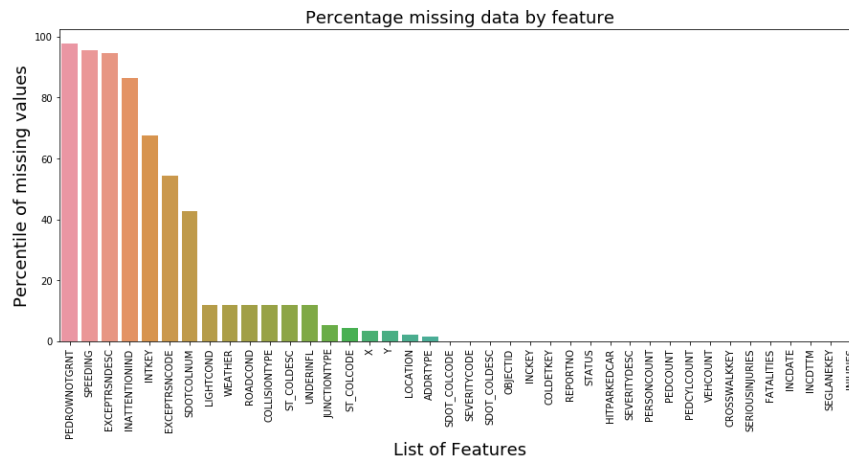
11 rows x 40 columns

Finding Empty / Missing values in Dataset

First thing we will do is to find the number of EMPTY / MISSING values in the dataset. We will then decide based on their percentage whether to keep those columns or drop them. Having high number of missing data in the attributes will make the data unreliable and model accuracy will be highly affected.

Out[4]:

	Total Nulls	Percent Nulls
PEDROWNOTGRNT	216330	97.654892
SPEEDING	211596	95.517887
EXCEPTRSNDESC	209746	94.682767
INATTENTIONIND	191337	86.372644
INTKEY	149589	67.526916
EXCEPTRSNCODE	120403	54.351879
SDOTCOLNUM	94320	42.577587
LIGHTCOND	26592	12.004063
WEATHER	26503	11.963887
ROADCOND	26422	11.927322



Deletion of unwanted attributes

From above percentile value, it is visible that top 7 categories can be dropped completely from the database because the NULL values are more than 50% and we just can't replace them with average value as it will not be beneficial for the Quality of the data.

```
In [5]: #removing the attributes having NULL values more than 40%
DataSetCol.drop(['PEDROWNOTGRNT', 'SPEEDING', 'EXCEPTSNDISC', 'INATTENTIONIND', 'INTKEY', 'EXCEPTSNCODE', 'SDOTCOLNUM'], axis=1, inplace=True)
```

We will drop the columns like Location, ObjectID, Report no, Status and different keys because they have NO IMPACT on our target variable SEVERITYCODE.

```
In [6]: DataSetCol.drop(['LOCATION', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPORTNO', 'STATUS', 'ST_COLCODE', 'ST_COLDESC', 'SEGLANEKEY', 'CROSSWALKKEY', 'SDOT_COLCODE', 'SDOT_COLDESC'], axis=1, inplace=True)
```

As per our understanding we don't need INCDTTM and INCDATE along with SEVERITYDESC as this will not impact our model so we will delete it. This is because we are already taking LIGHTCOND attribute which can clearly state whether it's day time or night time at the time of the accident. It will be more than enough.

```
In [10]: DataSetCol.drop(['INCDATE', 'INCDTTM', 'SEVERITYDESC'], axis=1, inplace=True)
```

```
In [37]: DataSetCol['COLLISIONTYPE'].value_counts()
```

```
Out[37]: Parked Car      46931
Angles      35341
Rear Ended  33553
Other       23126
Sideswipe   18312
Left Turn   14031
Pedestrian   7600
Cycles       5886
Right Turn   2969
Head On      2151
Name: COLLISIONTYPE, dtype: int64
```

```
In [38]: DataSetCol['UNDERINFL'].value_counts()
```

```
Out[38]: N      101205
0        79339
Y         5270
1         4106
Name: UNDERINFL, dtype: int64
```

From above COLLISIONTYPE attribute will not affect our target variable SEVERITYCODE because it is only telling us what type of Collision occurred after accident already happened, therefore we can drop this attribute. UNDERINFL attribute has more than 80%

of values which suggests that the driver was not DRUNK, therefore this attribute will also have no major effect on the SEVERITY of ACCIDENT. We will drop this attribute too.

Now as we need to predict the model based on the SEVERITY of the Accident therefore variables like PERSONCOUNT, PEDCOUNT, PEDCYLCOUNT, COLLISIONTYPE, VEHCOUNT, HITPARKEDCAR has no effect on the SEVERITY of the accident therefore we will drop these attributes.

```
In [17]: DataSetCol.drop(['PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'COLLISIONTYPE', 'VEHCOUNT', 'UNDERINFL', 'HITPARKEDCAR'], axis=1, inplace=True)
```

We will see below what exactly is relationship of SEVERITYCODE with INJURIES, SERIOUSINJURIES & FATALITIES is.

```
In [21]: DataSetCol.groupby(['SEVERITYCODE'])['SEVERITYDESC'].value_counts()
```

```
Out[21]: SEVERITYCODE  SEVERITYDESC
1      Property Damage Only Collision    159191
2      Injury Collision                  58747
2b     Serious Injury Collision          3102
3      Fatality Collision                 349
Name: SEVERITYDESC, dtype: int64
```

As we can see from above image of SEVERITY CODE description that it already contains INJURY, SERIOUS INJURY and FATALITY data, therefore using it PREDICTOR data will make the prediction reaching almost 100%. Therefore we need to drop these attributes so our TARGET can predict these values based on other attributes.

```
In [26]: DataSetCol.drop(['INJURIES', 'SERIOUSINJURIES', 'FATALITIES'], axis=1, inplace=True)
DataSetCol.reset_index(drop=True, inplace=True)
DataSetCol.head()
```

```
Out[26]:
```

	LONGITUDE	LATITUDE	ADDRTYPE	SEVERITYCODE	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND
0	-122.344896	47.717173	Block	2	Driveway Junction	Clear	Dry	Daylight
1	-122.376467	47.543774	Block	2	Mid-Block (not related to intersection)	Overcast	Dry	Dark - Street Lights On
2	-122.360735	47.701487	Intersection	1	At Intersection (intersection related)	Overcast	Wet	Daylight
3	-122.297415	47.599233	Intersection	1	At Intersection (intersection related)	Clear	Dry	Daylight
4	-122.368001	47.653585	Block	1	Mid-Block (not related to intersection)	Clear	Dry	Daylight

Target Variable Attributes Value Tuning

```
In [12]: DataSetCol['SEVERITYDESC'].value_counts()
```

```
Out[12]: Property Damage Only Collision    137596
         Injury Collision                  58747
         Unknown                          21595
         Serious Injury Collision          3102
         Fatality Collision                 349
         Name: SEVERITYDESC, dtype: int64
```

```
In [13]: DataSetCol['SEVERITYCODE'].value_counts()
```

```
Out[13]: 1    137596
         2     58747
         0     21595
         2b    3102
         3      349
         Name: SEVERITYCODE, dtype: int64
```

We will replace the **Unknown** values of SEVERITYDESC and **0** of SEVERITYCODE to Empty values so later we can check the percentile value of EMPTY values in the our TARGET attribute which is SEVERITYCODE to see if it will be feasible to drop these values or convert them to top used value

```
In [7]: DataSetCol['SEVERITYCODE'].replace("0",np.nan, inplace = True)
        DataSetCol['SEVERITYDESC'].replace("Unknown",np.nan, inplace = True)

In [8]: DataSetCol[['SEVERITYCODE', 'SEVERITYDESC']].isnull().sum()
Out[8]: SEVERITYCODE    21595
        SEVERITYDESC    21595
        dtype: int64

In [8]: Severity_unknown=DataSetCol[['SEVERITYDESC']].isnull().sum().sort_values(ascending=False)
        percent_unknown = (DataSetCol[['SEVERITYDESC']].isnull().sum()/DataSetCol[['SEVERITYDESC']].isnull().count()*100).sort_values(ascending=False)
        unknown_data_perc = pd.concat([Severity_unknown, percent_unknown], axis=1, keys=['Severity Unknown', 'Percent Unknown'])
        unknown_data_perc.head()
Out[8]:
```

	Severity Unknown	Percent Unknown
SEVERITYDESC	21616	9.757815

Above calculation shows that Unknown categories in SEVERITY CODE list which was coded as 0 are around 10% of the data. Based on the average value, we can either change this data to CODE 1 or drop this value. Here we will convert them to Code 1 which is "Property Damage only Collision"

```
In [17]: DataSetCol['SEVERITYCODE'].describe(include='all')
```

```
Out[17]: count    199794
         unique      4
         top         1
         freq    137596
         Name: SEVERITYCODE, dtype: object
```


As we can see from above that the top used CODE is 1, therefore now we will convert all Unknown values to 1 so we can normalize the data.

```
In [9]: DataSetCol['SEVERITYCODE'].replace(np.nan,"1", inplace = True)
DataSetCol['SEVERITYDESC'].replace(np.nan,"Property Damage Only Collision", inplace = True)
```

```
In [19]: DataSetCol[['SEVERITYCODE','SEVERITYDESC']].isnull().sum()
```

```
Out[19]: SEVERITYCODE    0
SEVERITYDESC    0
dtype: int64
```

```
In [20]: DataSetCol['SEVERITYCODE'].value_counts()
```

```
Out[20]: 1    159191
2     58747
2b     3102
3        349
Name: SEVERITYCODE, dtype: int64
```

```
In [21]: DataSetCol.groupby(['SEVERITYCODE'])['SEVERITYDESC'].value_counts()
```

```
Out[21]: SEVERITYCODE  SEVERITYDESC
1      Property Damage Only Collision    159191
2      Injury Collision                  58747
2b     Serious Injury Collision          3102
3      Fatality Collision                 349
Name: SEVERITYDESC, dtype: int64
```

Removal / Replacement of Null/Other/Unknown values

Provided coordinates might help us in improving the prediction of the Model therefor we will need X and Y coordinates. To understand clearly, we will change the name as follows:
X = LONGITUDE, Y = LATITUDE

```
In [11]: DataSetCol.rename(columns = {"X":"LONGITUDE", "Y":"LATITUDE"}, inplace=True)
```

Now we will check the number of Null or empty values in rest of the Data Attributes. We will decide whether we need to drop those values or change the values to average value based on their percentile value which we have already calculated earlier.

```
In [15]: DataSetCol.isnull().sum()
```

```
Out[15]: LONGITUDE      7471
LATITUDE      7471
ADDRTYPE      3712
SEVERITYCODE    0
COLLISIONTYPE  26230
PERSONCOUNT   0
PEDCOUNT      0
PEDCYLCOUNT    0
VEHCOUNT       0
INJURIES       0
SERIOUSINJURIES 0
FATALITIES     0
JUNCTIONTYPE   11972
UNDERINFL      26210
WEATHER        26420
ROADCOND       26339
LIGHTCOND      26509
HITPARKEDCAR   0
dtype: int64
```

Based on above, Longitude and Latitude having some NULL values. It is better to drop these NULL values as they indicate specific place of accident. After dropping them we will reset the index of the dataframe.

```
In [12]: DataSetCol.dropna(subset=["LONGITUDE"],axis=0, inplace=True)
DataSetCol.dropna(subset=["LATITUDE"],axis=0, inplace=True)
DataSetCol.reset_index(drop=True, inplace=True) #resetting index
```

We can see from above that most of our attributes have EMPTY values, therefore we will now look at them one by one in details to see if they have some Unknown or Other values available and whether we need to change them TOP value or drop them.

```
In [29]: DataSetCol['JUNCTIONTYPE'].value_counts()
```

```
Out[29]: Mid-Block (not related to intersection)    98823
At Intersection (intersection related)             68824
Mid-Block (but intersection related)                24044
Driveway Junction                                 11391
At Intersection (but not related to intersection)   2463
Ramp Junction                                       165
Unknown                                             18
Name: JUNCTIONTYPE, dtype: int64
```

```
In [13]: DataSetCol['JUNCTIONTYPE'].replace("Unknown","Mid-Block (not related to intersection)", inplace = True)
DataSetCol['JUNCTIONTYPE'].replace(np.nan,"Mid-Block (not related to intersection)", inplace = True)
DataSetCol['JUNCTIONTYPE'].value_counts()
```

```
Out[13]: Mid-Block (not related to intersection)    107108
At Intersection (intersection related)             68872
Mid-Block (but intersection related)                24049
Driveway Junction                                 11391
At Intersection (but not related to intersection)   2465
Ramp Junction                                       165
Name: JUNCTIONTYPE, dtype: int64
```

We can see the Unknown value exist in JUNCTIONTYPE attributes which consisted of very small number. Therefore we replaced it with the Top tabled value.

```
In [31]: DataSetCol['WEATHER'].value_counts()

Out[31]: Clear                112466
         Raining              32898
         Overcast             27953
         Unknown              13933
         Snowing               906
         Other                 801
         Fog/Smog/Smoke        561
         Sleet/Hail/Freezing Rain 115
         Blowing Sand/Dirt      50
         Severe Crosswind       25
         Partly Cloudy          10
         Blowing Snow           1
         Name: WEATHER, dtype: int64

In [14]: DataSetCol['WEATHER'].replace("Unknown","Clear", inplace = True)
         DataSetCol['WEATHER'].replace(np.nan,"Clear", inplace = True)
         DataSetCol['WEATHER'].replace("Other","Clear", inplace = True)
         DataSetCol['WEATHER'].value_counts()

Out[14]: Clear                151522
         Raining              32898
         Overcast             27962
         Snowing               906
         Fog/Smog/Smoke        561
         Sleet/Hail/Freezing Rain 115
         Blowing Sand/Dirt      50
         Severe Crosswind       25
         Partly Cloudy          10
         Blowing Snow           1
         Name: WEATHER, dtype: int64
```

Similarly in Weather attribute, we replaced Unknown, Empty and Other variables to Clear as Clear was the top most category.

```
In [33]: DataSetCol['ROADCOND'].value_counts()

Out[33]: Dry                126042
         Wet                47312
         Unknown            13900
         Ice                1199
         Snow/Slush          999
         Other               121
         Standing Water      106
         Sand/Mud/Dirt        66
         Oil                 53
         Name: ROADCOND, dtype: int64

In [15]: DataSetCol['ROADCOND'].replace("Unknown","Dry", inplace = True)
         DataSetCol['ROADCOND'].replace(np.nan,"Dry", inplace = True)
         DataSetCol['ROADCOND'].replace("Other","Dry", inplace = True)
         DataSetCol['ROADCOND'].value_counts()

Out[15]: Dry                164315
         Wet                47312
         Ice                1199
         Snow/Slush          999
         Standing Water      106
         Sand/Mud/Dirt        66
         Oil                 53
         Name: ROADCOND, dtype: int64
```

For ROADCOND, we replaced Unknown, empty and Other to Dry.

```
In [35]: DataSetCol['LIGHTCOND'].value_counts()
```

```
Out[35]: Daylight          116857
Dark - Street Lights On   48902
Unknown                  12491
Dusk                     5952
Dawn                     2526
Dark - No Street Lights   1493
Dark - Street Lights Off  1192
Other                    196
Dark - Unknown Lighting   23
Name: LIGHTCOND, dtype: int64
```

```
In [16]: DataSetCol['LIGHTCOND'].replace("Unknown","Daylight", inplace = True)
DataSetCol['LIGHTCOND'].replace(np.nan,"Daylight", inplace = True)
DataSetCol['LIGHTCOND'].replace("Other","Daylight", inplace = True)
DataSetCol['LIGHTCOND'].value_counts()
```

```
Out[16]: Daylight          153953
Dark - Street Lights On   48910
Dusk                     5952
Dawn                     2527
Dark - No Street Lights   1493
Dark - Street Lights Off  1192
Dark - Unknown Lighting   23
Name: LIGHTCOND, dtype: int64
```

In LIGHTCOND, we replaced Unknown, empty and Other to Daylight.

```
In [22]: DataSetCol.isnull().sum()
```

```
Out[22]: LONGITUDE          0
LATITUDE                    0
ADDRTYPE                    0
SEVERITYCODE                0
INJURIES                    0
SERIOUSINJURIES            0
FATALITIES                  0
JUNCTIONTYPE                0
WEATHER                     0
ROADCOND                    0
LIGHTCOND                   0
dtype: int64
```

As we have now transformed and cleaned our data and got our required attributes, therefore next phase is to start MODELLING the data.

Modelling with Evaluation

First step is to reset the index as a precautionary purpose.

```
In [18]: DataSetCol.reset_index(drop=True, inplace=True)
```

Now we will define the dataframes X and y to proceed with modelling.

```
In [27]: X=DataSetCol[['LONGITUDE','LATITUDE','ADDRTYPE','JUNCTIONTYPE','WEATHER','ROADCOND','LIGHTCOND']].values
X[0:5]
```

```
Out[27]: array([-122.344896078772, 47.717173100926296, 'Block',
                'Driveway Junction', 'Clear', 'Dry', 'Daylight'],
                [-122.376466591478, 47.543773580049105, 'Block',
                'Mid-Block (not related to intersection)', 'Overcast', 'Dry',
                'Dark - Street Lights On'],
                [-122.36073528137301, 47.70148713459311, 'Intersection',
                'At Intersection (intersection related)', 'Overcast', 'Wet',
                'Daylight'],
                [-122.297414682629, 47.59923283954921, 'Intersection',
                'At Intersection (intersection related)', 'Clear', 'Dry',
                'Daylight'],
                [-122.368000760038, 47.6535853541642, 'Block',
                'Mid-Block (not related to intersection)', 'Clear', 'Dry',
                'Daylight']], dtype=object)
```

Now we need to transform our attributes types to integer so we can plot them and use appropriate Modelling. We will sue LABEENCODER from SKLEARN

```
In [28]: from sklearn import preprocessing
```

```
le_add = preprocessing.LabelEncoder()
le_add.fit(['Block','Intersection'])
X[:,2] = le_add.transform(X[:,2])

le_jt = preprocessing.LabelEncoder()
le_jt.fit(['Mid-Block (not related to intersection)', 'At Intersection (intersection related)', 'Mid-Block (but intersection relate
d)', 'Driveway Junction', 'At Intersection (but not related to intersection)', 'Ramp Junction'])
X[:,3] = le_jt.transform(X[:,3])

le_wea = preprocessing.LabelEncoder()
le_wea.fit(['Clear', 'Raining', 'Overcast', 'Snowing', 'Fog/Smog/Smoke', 'Sleet/Hail/Freezing Rain', 'Blowing Sand/Dirt', 'Severe Crosswin
d', 'Partly Cloudy', 'Blowing Snow'])
X[:,4] = le_wea.transform(X[:,4])

le_rd = preprocessing.LabelEncoder()
le_rd.fit(['Dry', 'Wet', 'Ice', 'Snow/Slush', 'Standing Water', 'Sand/Mud/Dirt', 'Oil'])
X[:,5] = le_rd.transform(X[:,5])

le_lc = preprocessing.LabelEncoder()
le_lc.fit(['Daylight', 'Dark - Street Lights On', 'Dusk', 'Dawn', 'Dark - No Street Lights', 'Dark - Street Lights Off', 'Dark - Unknown Ligh
ting'])
X[:,6] = le_lc.transform(X[:,6])
```

```
In [29]: y=DataSetCol['SEVERITYCODE'].values
y
```

```
Out[29]: array(['2', '2', '1', ..., '1', '1', '1'], dtype=object)
```

Next step is to NORMALIZE the data by using StandardScaler function from sklearn

```
In [30]: X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype object
was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype object
was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

Out[30]: array([[ -0.4704469 ,  1.73024804, -0.70884677, -0.55986245, -0.59051875,
        -0.54308006,  0.53706301],
       [-1.52083618, -1.36351084, -0.70884677,  0.89309596,  0.70481426,
        -0.54308006, -1.67958529],
       [-0.9974364 ,  1.45038223,  1.41074212, -1.28634165,  0.70481426,
        1.86283687,  0.53706301],
       [ 1.10931679, -0.37401805,  1.41074212, -1.28634165, -0.59051875,
        -0.54308006,  0.53706301],
       [-1.23916769,  0.59572843, -0.70884677,  0.89309596, -0.59051875,
```

Now we will do the TRAIN TEST SPLIT. We plan to split the test portion at 30% with random state of 3. I have tried different test portions and random state but in the end got satisfied with 20% test split with random state 3.




Making Train Test Split again with updated dataset

```
In [31]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=3)
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)
```

```
Train set: (171240, 7) (171240,)
Test set: (42810, 7) (42810,)
```

Modelling Techniques

I have utilized 3 Modelling techniques in the analysis.

-  KNN
-  Decision Tree
-  Logistic Regression

I did the Modelling alongwith Evaluation which includes calculating the set Accuracy, F1 Score and Jaccard set accuracy. For Logistic Regression, Logloss was also calculated.

I will discuss each Modelling technique along with its result individually.

1 - KNN ---- K-Nearest Neighbor

- The K-Nearest Neighbours algorithm is a classification algorithm that takes a bunch of labelled points and uses them to learn how to label other points. **A method of classifying cases, based on the similarity of other cases.** Cases that are near each other are said to be neighbours.

****** Based on similar cases with same class labels are near each other.

At start I calculated the Train and Test set accuracy by declaring value for K = 4.

```
In [63]: #Using same value as we took before which is 4 to fit the model now.
k = 4
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
Out[63]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                             weights='uniform')
```

```
In [64]: #Predicting the Model
yhat = neigh.predict(X_test)
yhat[0:5]
```

```
Out[64]: array(['1', '1', '1', '1', '1'], dtype=object)
```

```
In [65]: #Now checking the updated Accuracy after removing the SIMILAR attributes
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:  0.7553788259492561
Test set Accuracy:  0.6901645474943904
```

K in KNN, is the number of nearest neighbors to examine. It is supposed to be specified by the User. So, how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then chose k =1, use the training part for modeling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k, and see which k is the best for your model.

```
In [66]: Ks = 20 #It will check for value of k until 20 and provide the best k value. We can use it higher but as the number of data values are very high so it will take lot of time
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfusionMx = [];
for n in range(1,Ks):

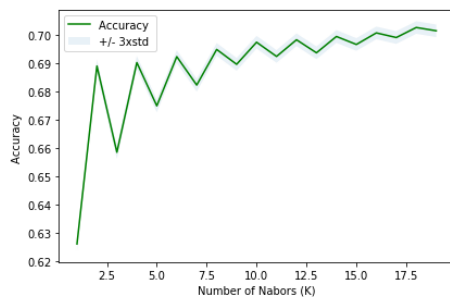
    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)

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

mean_acc

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 0.7025757292445775 with k= 18

Now as per above the best value for k within 20 counts is **18** to get the best accuracy. Therefore we recalculated the KNN by entering the value of K = 20. This type we also calculated f1-score and Jaccard set accuracy.


```

In [32]: from sklearn.neighbors import KNeighborsClassifier
         k = 18
         neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
         neigh

Out[32]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=18, p=2,
                             weights='uniform')

In [33]: #Predicting the Model with k = 18
         yhat = neigh.predict(X_test)
         yhat[0:5]

Out[33]: array(['1', '1', '1', '1', '1'], dtype=object)

In [34]: #Now checking the updated Accuracy after changing value of k to 18
         from sklearn import metrics
         from sklearn.metrics import jaccard_similarity_score
         from sklearn.metrics import f1_score
         from sklearn.metrics import log_loss
         print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
         print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
         print("F1 Score: ", f1_score(y_test, yhat, average='weighted'))
         print("Jaccard set Accuracy: ", jaccard_similarity_score(y_test, yhat))

Train set Accuracy:  0.7277738846064004
Test set Accuracy:  0.7046484466246205
F1 Score:  0.6376534810260911
Jaccard set Accuracy:  0.7046484466246205

```

Now with k=18 our Train set accuracy was reduced but Test set accuracy improved and the difference between Train and Test set is reduced, making it more appropriate.

2 --- Decision Tree

- The basic intuition behind a decision tree is to map out all possible decision paths in the form of a tree.
- Decision trees are about testing an attribute and branching the cases based on the result of the test.

I took the criterion value as “entropy with maximum depth of 8 branches of tree. These values can be changed as per understanding.

```

In [35]: from sklearn.tree import DecisionTreeClassifier
AccidentTree = DecisionTreeClassifier(criterion="entropy", max_depth=8) # We are using maximum depth of 8.
AccidentTree

Out[35]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=8,
                                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 [36]: #Fitting the Training Model
AccidentTree.fit(X_train,y_train)

Out[36]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=8,
                                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]: PredAccTree = AccidentTree.predict(X_test)
print (PredAccTree [0:5])
print (y_test [0:5])

['1' '1' '1' '1' '1']
['1' '1' '1' '1' '1']

In [38]: print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, PredAccTree))
print("F1 Score: ", f1_score(y_test, PredAccTree, average='weighted'))
print("Jaccard set Accuracy: ", jaccard_similarity_score(y_test, PredAccTree))

DecisionTrees's Accuracy:  0.712660593319318
F1 Score:  0.5993730903349023
Jaccard set Accuracy:  0.712660593319318

```

Here it is visible that Decision Tree and Jaccard set Accuracy is almost similar to the value we received in KNN, but f1-score value is reduced by 4%. This is still acceptable value as we have very high number of data counts.

3 --- Logistic Regression

- It is a classification algorithm for categorical variables
- It is analogous to linear regression but tries to predict a categorical or discrete target field instead of numeric one.

We selected the value for $C = 0.01$ and solver as liblinear to fit the Logistic Regression.

```
In [39]: 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
```

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
"this warning.", FutureWarning)

```
Out[39]: 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 [40]: yhat_LR = LR.predict(X_test)
yhat_LR
```

```
Out[40]: array(['1', '1', '1', ..., '1', '1', '1'], dtype=object)
```

Predict_proba returns estimates for all classes, ordered by the label of classes. So, the first column is the probability of class 1, $P(Y=1|X)$, and second column is probability of class 0, $P(Y=0|X)$:

```
In [41]: yhat_LR_prob = LR.predict_proba(X_test)
yhat_LR_prob
```

```
Out[41]: array([[0.80581288, 0.17913094, 0.01100675, 0.00404943],
[0.80999427, 0.17390656, 0.01185915, 0.00424002],
[0.61351821, 0.35895755, 0.0224345 , 0.00508974],
...,
[0.74720496, 0.23861005, 0.01021429, 0.0039707 ],
[0.51545934, 0.44468319, 0.03414558, 0.00571189],
[0.69286978, 0.29157159, 0.01134915, 0.00420949]])
```

```
In [42]: print("Logistic Regression Accuracy: ", metrics.accuracy_score(y_test, yhat_LR))
print("F1 Score: ", f1_score(y_test, yhat_LR, average='weighted'))
print("JAccard set Accuracy: ", jaccard_similarity_score(y_test, yhat_LR))
print ("LogLoss: : %.4f" % log_loss(y_test, yhat_LR_prob))
```

Logistic Regression Accuracy: 0.7133380051389863

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
'precision', 'predicted', average, warn_for)

F1 Score: 0.5962111070758105
JAccard set Accuracy: 0.7133380051389863
LogLoss: : 0.6394

LR accuracy and Jaccard Score is still same as of KNN and Decision Tree. F1-score for LR is similar to Decision Tree.

Additional entity in Logistic Regression is to calculate the Log loss which is around 64%.

Result

We have modelled and evaluated our data with following models individually

- KNN
- Decision Tree
- Logistic Regression

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.7046	0.637	NA
Decision Tree	0.7126	0.5993	NA
Logistic Regression	0.7133	0.6394	0.5566

Discussion

From the results of their accuracy models, we can see that all of them experienced accuracy of more than 70% which is a good indication based on the huge data we received. From Evaluation point of the view, we can see the Jaccard set Accuracy for all of them is also greater than 70%, which is also a good sign and proves the confidence on our working model. F1 Score for KNN is around 63% whereas for Decision Tree and Logistic Regression it is around 59%. We can consider this as acceptable.

This model can help predict the severity of the accidents based on factors considered as Road condition, Light conditions, Weather conditions, Junction types and physical location of the accident area.

Conclusion

As a conclusion, I can say that the models discussed in this report can benefit the traffic department system of Seattle to help them predict the different possibilities of Severity faced during the accidents.