Applied Capstone Project

Model and Prediction of severity of road accidents in Seattle

Elliot Ho

26th September

Introduction and Data Exploration

- Data file used is Data_Collisions.csv available on the Capstone Project
- This project intends to analyse and process the traffic incidents data in Seattle. The aim of the project is to predict the severity of an accident with the data given like latitude, longitude, weather conditions, junction types and others.
- Pre-process the data
- Build the machine learning model
- Evaluate the model for accuracy

Business Understanding

- This data science study is to predict the severity (1 or 2) of a vehicular accident based on already existing data for the Seattle region
- Severity of 1 indicates that there was just property damage. Severity of 2 indicates serious injury or fatality
- The occurrence of each incident is highly dependent on the location of the accident and the environmental conditions

Business Understanding

Target VARIABLE : severity code (SEVERITYCODE)

Independent variables : VEHCOUNT, SDOT_COLCODE, INATTENTIONIND, UNDERINFL, SPEEDING, WEATHER, JUNCTIONTYPE, ROADCOND, LIGHTCOND.

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, non-governmental organisations Car and life Insurance companies, Urgent and Emergency care and other regional authorities

Data Understanding

The Data Set consists of a record of all accidents. Each row corresponds to a single incident. The main features or attributes that are going to form our training set are:

Road Condition

Weather Condition

Driver Inattention

Junction

Car Speeding

No. of people/vehicles involved

light conditions

Data Understanding

'speeding' feature has mostly 'na' values and is not really suitable for the training set

'Location' feature contains the literal address of the accident location and hence is not a suitable attribute for the feature set.

‘Road Conditions' is a categorical value and comprises:

Dry, Wet, Unknown, Ice, Snow/Slush, Other, Standing Water, Sand/Mud/Dirt, Oil

The categorical values for 'Weather' feature are:

Clear, Raining, Overcast, Snowing, Fog/Smog/Smoke, Sleet/Hail/Freezing Rain, Blowing Sand/Dirt, Severe Crosswind, Partly Cloudy

'JUNCTIONTYPE' categorical values:

Mid-Block (not related to intersection), At Intersection (intersection related), Mid-Block (but intersection related), Driveway Junction, At Intersection (but not related to intersection), Ramp Junction,

Methodology – Data Cleaning

Drop the irrelevant features and unique IDs

In [237]:

```
# Drop the irrelevant features
# Drop the irrelevant unique IDs

df.drop(['OBJECTID','EXCEPTRSNDESC','EXCEPTRSNCODE','INTKEY','X','Y',
         'SDOTCOLNUM','INCDATE','INCDTTM','SEGLANEKEY','CROSSWALKKEY',
        'LOCATION','ST_COLDESC','SDOT_COLDESC','INCKEY','COLDETKEY','REPORTNO',
        'STATUS','SEVERITYCODE.1', 'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT'],axis=1,inplace=True)
df.drop(['ADDRTYPE', 'SEVERITYDESC', 'COLLISIONTYPE'],axis=1,inplace=True)
df.drop(['PEDROWNOTGRNT'],axis = 1, inplace=True)

```

Drop the Unknown index

In [238]:

```
# df[df['COL_NAME'] == 'XXXXX'].index => get the index from the column by specific values
# df_actual = df_actual[df_actual.ROADCOND != 'Unknown']

df.drop(index=(df[df['LIGHTCOND'] == 'Unknown'].index), axis = 0, inplace = True)
df.drop(index=(df[df['JUNCTIONTYPE'] == 'Unknown'].index), axis = 0, inplace = True)
df.drop(index=(df[df['WEATHER'] == 'Unknown'].index), axis = 0,inplace = True)
df.drop(index=(df[df['ROADCOND'] == 'Unknown'].index), axis = 0, inplace = True)

```

Methodology – Data Cleaning

Dealing with the null values

In [239]:
```
Features = df.columns
num_of_null_by_cols = df.isnull().sum().sort_values(ascending=False)
percentage_of_null_by_cols = (df.isnull().sum() / len(df)).sort_values(ascending=False)
null_df = pd.DataFrame({'Number of Null': num_of_null_by_cols,
                        'Percentage of Null':percentage_of_null_by_cols})
null_df.index.name = 'Features'
null_df
```

Out[239]:

Features	Number of Null	Percentage of Null
SPEEDING	166608	0.947913
INATTENTIONIND	147081	0.836814
LIGHTCOND	5119	0.029124
WEATHER	5065	0.028817
ROADCOND	4993	0.028408
UNDERINFL	4882	0.027776
JUNCTIONTYPE	2676	0.015225
ST_COLCODE	16	0.000091
HITPARKEDCAR	0	0.000000
SDOT_COLCODE	0	0.000000
VEHCOUNT	0	0.000000
SEVERITYCODE	0	0.000000

Dealing with the missing values

In [240]:
```
df['SPEEDING'].fillna('N',inplace = True)
df['INATTENTIONIND'].fillna('N',inplace = True)
```

Methodology – Data Cleaning

Dealing with the null values

In [241]:
```
# drop the particular missing item


df.drop(df[df['JUNCTIONTYPE'].isnull()].index,inplace = True)
df.drop(df[df['WEATHER'].isnull()].index,inplace = True)
df.drop(df[df['ROADCOND'].isnull()].index,inplace = True)
df.drop(df[df['UNDERINFL'].isnull()].index,inplace = True)
df.drop(df[df['ST_COLCODE'].isnull()].index,inplace = True)
df.drop(df[df['LIGHTCOND'].isnull()].index,inplace = True)

```

In [242]:
```
df.isnull().sum().sort_values(ascending=False)[0:5]
```

Out[242]:
```
HITPARKEDCAR    0
ST_COLCODE      0
SPEEDING        0
LIGHTCOND       0
ROADCOND        0
dtype: int64
```

All the missing and null values are settled down.

Train and Test set

Before normalizing the data set, we need to balance the data set by using the random under sampling (RUS) method.

In [223]:

```
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)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

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

Logistic Regression Algorithm

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
```

Decision Tree Algorithm

Decision Tree

In [227]:
```python
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]:
```python
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]:
```python
drugTree.fit(X_train_DT,y_train_DT)

predTree_DT = drugTree.predict(X_test_DT) #prediction

```

In [230]:
```python
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
```

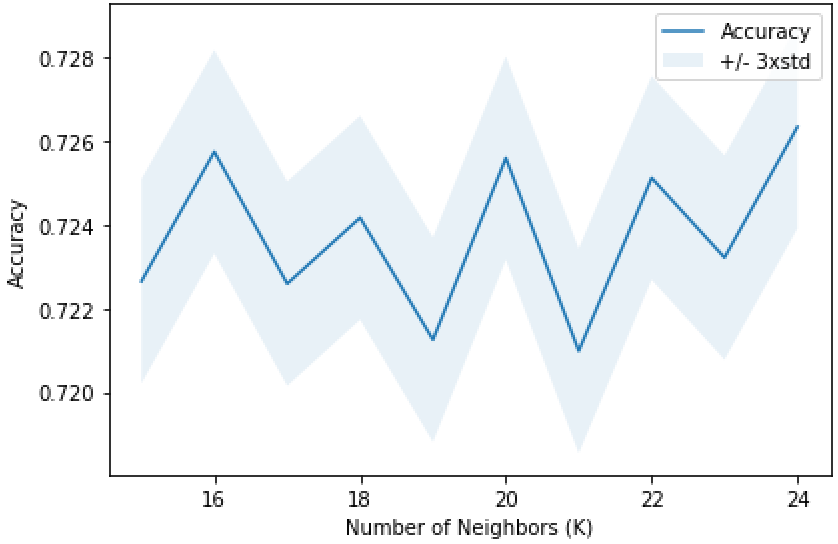
K-Nearest Neighbour Algorithm

```
In [45]:  # finding a good K for the KNN model
```

```
In [46]:
# 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
```

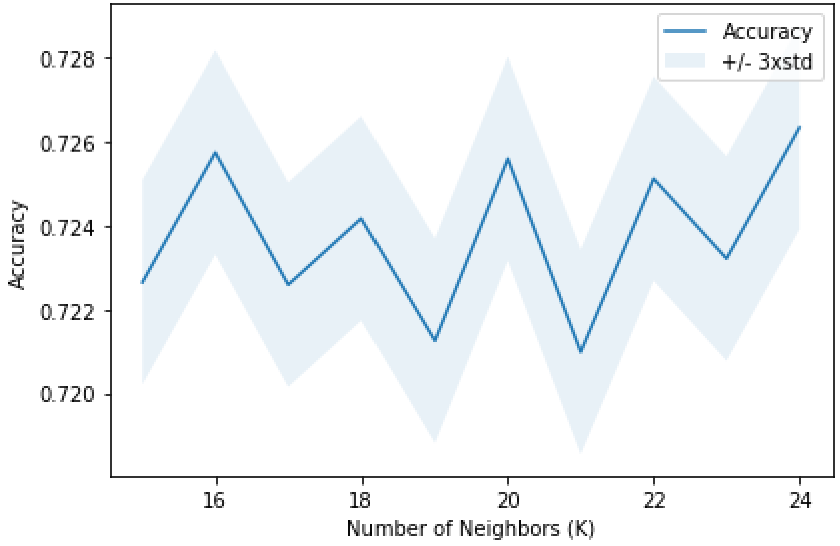
K-Nearest Neighbour Algorithm

In [46]:

```
# 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
```

K-Nearest Neighbour Algorithm

In [48]:

```
# 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
```

Results

In [265]:
```
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

From the above table, we can see that the logistic regression is the best model for this project. And the worst model is KNN.

Conclusion

Data science equips us with a system to predict an outcome before it happens

This is especially very useful in the case of road accidents

Using the data of environment conditions and locations of previous accidents we can now say with a 70% certainty that dry road conditions and mid-block junctions related to intersections cause a higher probability of accidents than any other conditions