2020

Seattle Traffic Accident Severity Modeling

IBM COURSERA CAPSTONE PROJECT

VIDUR CHANNA

Contents

Introduction

Background Description

Traffic accidents result in severe fatalities across the world and are the leading cause of death for people aged 15-29. Weather patterns can have a dramatic impact on the safety of driving in specific regions, due to limited visibility and reduced grip from tyres.

When people start planning their trips between places, they tend to check their route first and see estimated time to arrival, but often do not see any warnings for weather or increased accident risks on certain roads. Due to this, drivers might end up taking a dangerous route and encounter treacherous road surfaces and end up in an accident.

The purpose of this project is to model these dangerous driving conditions and predict regions where accidents are most likely to be severe. This will be done to improve road safety by warning people and allowing them to postpone travel or choose safer routes to get to their destinations. The data obtained from this model will be delivered to the Seattle government and police as well, so they can remain on high alert in regions predicted to be high risk for severe crashes.

Data Description

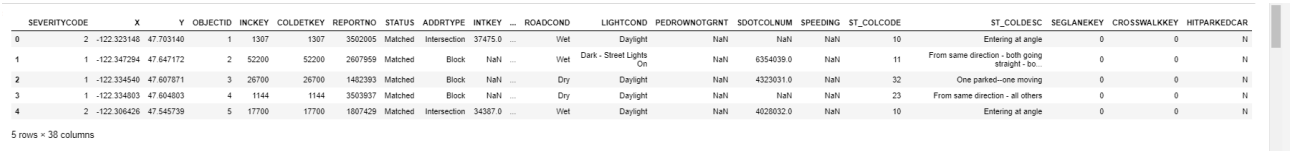
The Seattle SPOT Traffic Management Division has a dataset containing information regarding collisions from 2004 to 2020 with several key features surrounding severity and weather. The 'SEVERITYCODE' column is going to be the target variable which the model will predict to warn drivers of dangerous areas. The variable is effectively a binary class which classifies accidents into either 'property damage' or 'injury collision'. There are classifiers for 'severe injury' and 'fatality', but no data for those metrics is present within the data set hence they cannot be modeled.

The variables used to train the model will be 'WEATHER', 'LIGHTCOND' and 'ROADCOND' as they pertain directly to the conditions the drivers were in at the time of their accidents. The three variables are all TEXT data types and can hence be classified using frequency counts by processing the data.

Methodology

Initial Data Wrangling

To begin to understand the dataset, I first had to view the attributes of the csv file and extract the relevant ones for my modeling. When going through the initial file, there were 38 columns of data which were unnecessary and cluttered the flow of data, so I created a DataFrame using only the identified target variable of 'SEVERITYCODE' and the modelling variables of 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.



	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	NaN	NaN	10	Entering at angle	0	0	N
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN	6354039.0	NaN	11	From same direction - both going straight - bo...	0	0	N
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031.0	NaN	32	One parked--one moving	0	0	N
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	NaN	NaN	23	From same direction - all others	0	0	N
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032.0	NaN	10	Entering at angle	0	0	N

5 rows × 38 columns

As is visible above, the initial file contained many redundant variables and irrelevant metrics which do not aid in the analysis of the impacts of driving conditions on the severity of crashes in Seattle.

```
In [169]:
tup_crash_data = df_data_1[['SEVERITYCODE','ROADCOND','LIGHTCOND','WEATHER']]
df_crash_data = pd.DataFrame(tup_crash_data, columns = ['SEVERITYCODE','ROADCOND','LIGHTCOND','WEATHER'])
df_crash_data.head()
```

Out[169]:

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

Following the extraction, I now had a clearer dataset focused solely on the key variables for modeling. However, the variables within the DataFrame were mixed and contained categorical and binary variables as well which muddled the data clarity.

Data Cleaning

The next step was to scrub the data of any null values which would interrupt the modelling process There are several ways to deal with missing data in a given data set, which include replacing it with the average of the column in which the data is missing from or simply dropping the row from the DataFrame altogether.

Although it may seem prudent to replace the missing values with averages, one key factor to bear n mind is that the values in this dataset are discrete and hence any averaging would lead to significant overestimation or underestimation based on the boundaries defined for snapping to categories within the variables.

Therefore, I checked the raw value of missing data and upon finding that it comprised a small percentage of the overall dataset, I safely dropped the null values from the table.

```
In [172]: # Dropping rows with null values to clean up data
          df_crash_data = df_crash_data.dropna()
          df_crash_data.shape

Out[172]: (189337, 4)

In [173]: df_crash_data = df_crash_data.reset_index(drop = True)
          df_crash_data.isnull().sum()

Out[173]: SEVERITYCODE    0
          ROADCOND        0
          LIGHTCOND       0
          WEATHER         0
          dtype: int64
```

Exploratory Data Analysis

The next step in understanding the dataset better is to conduct some basic analysis of the categorical variables and see which metrics are the most dominant.

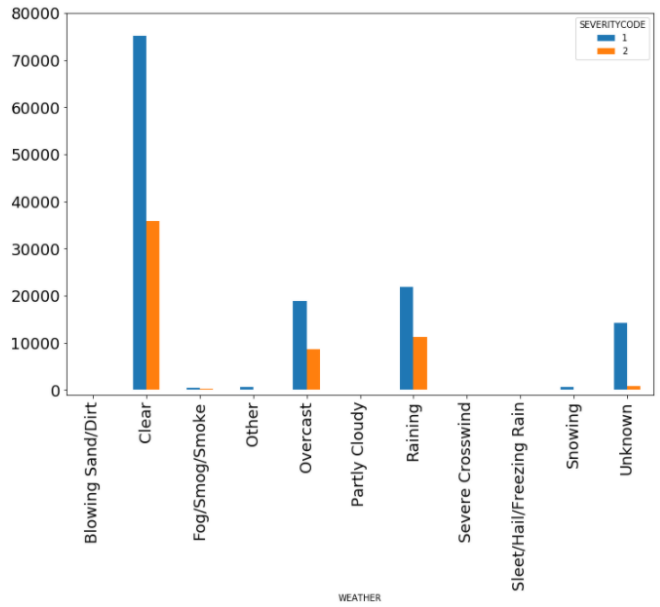
To do this, I first took the value counts of each variable and looked at the breakdown of the most common labels in each field.

```
In [174]: df_crash_data['SEVERITYCODE'].value_counts()

Out[174]: 1    132285
          2     57052
          Name: SEVERITYCODE, dtype: int64

In [158]: df_crash_data['ROADCOND'].value_counts()

Out[158]: Dry               124300
          Wet                47417
          Unknown            15031
          Ice                 1206
          Snow/Slush           999
          Other                131
          Standing Water       115
          Sand/Mud/Dirt         74
          Oil                   64
          Name: ROADCOND, dtype: int64

In [159]: df_crash_data['LIGHTCOND'].value_counts()

Out[159]: Daylight                   116077
          Dark - Street Lights On     48440
          Unknown                    13456
          Dusk                        5889
          Dawn                        2502
          Dark - No Street Lights     1535
          Dark - Street Lights Off    1192
          Other                        235
          Dark - Unknown Lighting       11
          Name: LIGHTCOND, dtype: int64

In [160]: df_crash_data['WEATHER'].value_counts()

Out[160]: Clear                       111008
          Raining                      33117
          Overcast                     27681
          Unknown                      15039
          Snowing                        901
          Other                          824
          Fog/Smog/Smoke                 569
          Sleet/Hail/Freezing Rain       113
          Blowing Sand/Dirt               55
          Severe Crosswind                25
          Partly Cloudy                    5
          Name: WEATHER, dtype: int64
```

As is visible, the leading variables for each field are ‘1’, ‘Dry’, ‘Daylight’ and ‘Clear’ respectively, which can further be visualised using a series of matplotlib bar charts.

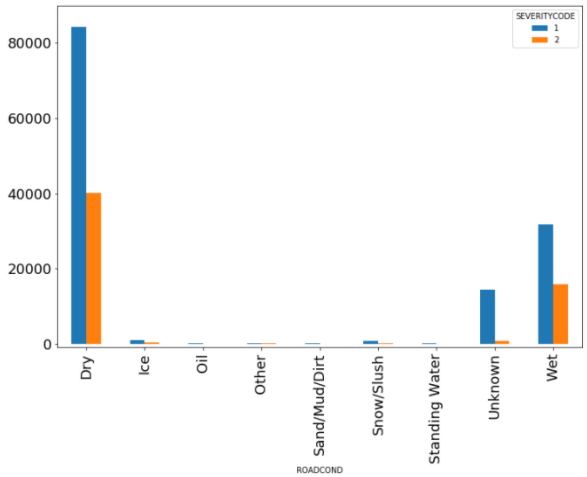
```
df_crash_data.groupby(['WEATHER', 'SEVERITYCODE']).agg('size').unstack().plot(kind = 'bar', legend=True, figsize=(12, 8), fontsize=18)
plt.ylim((-1000,80000))
```

9]: (-1000, 80000)



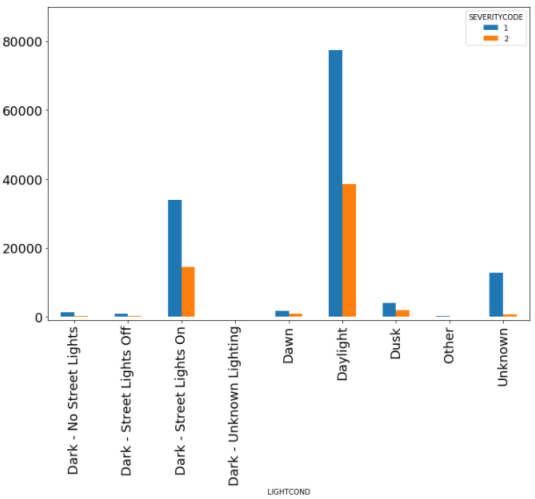
In [116]:
```
df_crash_data.groupby(['ROADCOND', 'SEVERITYCODE']).agg('size').unstack().plot(kind = 'bar', legend=True, figsize=(12, 8), fontsize=18)
plt.ylim((-1000,90000))
```

Out[116]: (-1000, 90000)



In [117]:
```
df_crash_data.groupby(['LIGHTCOND', 'SEVERITYCODE']).agg('size').unstack().plot(kind = 'bar', legend=True, figsize=(12, 8), fontsize=18)
plt.ylim((-1000,90000))
```

Out[117]: (-1000, 90000)



One Hot Encoding

This method of data wrangling is used to cluster categorical variables into a series of binary variables which can be analysed by a machine learning algorithm. This was done by using the Pandas dummies function, which allows you to automatically sort different classes into discrete columns of binary values for each row.

```
In [189]: encoded_data = df_crash_data.copy()
          encoded_data = pd.concat([encoded_data, pd.get_dummies(df_crash_data['ROADCOND'])], axis =1)
          encoded_data = pd.concat([encoded_data, pd.get_dummies(df_crash_data['LIGHTCOND'])], axis =1)
          encoded_data = pd.concat([encoded_data, pd.get_dummies(df_crash_data['WEATHER'])], axis =1)
          encoded_data.head()
```

Out[189]:

	SEVERITYCODE	ROADCOND	LIGHTCOND	WEATHER	Dry	Ice	Oil	Other	Sand/Mud/Dirt	Snow/Slush	...	Clear	Fog/Smog/Smoke
0	2	Wet	Daylight	Overcast	0	0	0	0	0	0	...	0	0
1	1	Wet	Dark - Street Lights On	Raining	0	0	0	0	0	0	...	0	0
2	1	Dry	Daylight	Overcast	1	0	0	0	0	0	...	0	0
3	1	Dry	Daylight	Clear	1	0	0	0	0	0	...	1	0
4	2	Wet	Daylight	Raining	0	0	0	0	0	0	...	0	0

5 rows × 33 columns

Normalisation and Train/Test Splitting

Following the conversion of the variables into a suitable format, the DataFrame now needed to be normalised in order to reduce bias while training the model. This will be done by utilising the StandardScaler function and fitting t he encoded data set within it.

```
In [199]: from sklearn import preprocessing
          X= preprocessing.StandardScaler().fit(X).transform(X)
          X[0:5]
```

```
Out[199]: array([[ 1.52271934, -1.38246872, -0.08006514, -0.01838847, -0.02631287,
                  -0.01977348, -0.07283058, -0.02465262, -0.29365525,  1.7300345 ,
                  -0.09040742, -0.0795961 , -0.58634226, -0.00762239, -0.11572163,
                   0.79443866, -0.17916958, -0.03525217, -0.27659768, -0.01704616,
                  -1.19046285, -0.05490248, -0.06611393,  2.41660142, -0.00513893,
                  -0.46042318, -0.01149161, -0.02443718, -0.06914813, -0.29374013],
                 [-0.65671984, -1.38246872, -0.08006514, -0.01838847, -0.02631287,
                  -0.01977348, -0.07283058, -0.02465262, -0.29365525,  1.7300345 ,
                  -0.09040742, -0.0795961 ,  1.70548854, -0.00762239, -0.11572163,
                  -1.25875043, -0.17916958, -0.03525217, -0.27659768, -0.01704616,
                  -1.19046285, -0.05490248, -0.06611393, -0.41380428, -0.00513893,
                   2.17191499, -0.01149161, -0.02443718, -0.06914813, -0.29374013],
                 [-0.65671984,  0.72334367, -0.08006514, -0.01838847, -0.02631287,
                  -0.01977348, -0.07283058, -0.02465262, -0.29365525, -0.57802315,
                  -0.09040742, -0.0795961 , -0.58634226, -0.00762239, -0.11572163,
                   0.79443866, -0.17916958, -0.03525217, -0.27659768, -0.01704616,
                  -1.19046285, -0.05490248, -0.06611393,  2.41660142, -0.00513893,
                  -0.46042318, -0.01149161, -0.02443718, -0.06914813, -0.29374013],
                 [-0.65671984,  0.72334367, -0.08006514, -0.01838847, -0.02631287,
                  -0.01977348, -0.07283058, -0.02465262, -0.29365525, -0.57802315,
                  -0.09040742, -0.0795961 , -0.58634226, -0.00762239, -0.11572163,
                   0.79443866, -0.17916958, -0.03525217, -0.27659768, -0.01704616,
                   0.84000941, -0.05490248, -0.06611393, -0.41380428, -0.00513893,
                  -0.46042318, -0.01149161, -0.02443718, -0.06914813, -0.29374013],
                 [ 1.52271934, -1.38246872, -0.08006514, -0.01838847, -0.02631287,
                  -0.01977348, -0.07283058, -0.02465262, -0.29365525,  1.7300345 ,
                  -0.09040742, -0.0795961 , -0.58634226, -0.00762239, -0.11572163,
                   0.79443866, -0.17916958, -0.03525217, -0.27659768, -0.01704616,
                  -1.19046285, -0.05490248, -0.06611393, -0.41380428, -0.00513893,
                   2.17191499, -0.01149161, -0.02443718, -0.06914813, -0.29374013]])
```

Finally, the last step before the model training can begin is to create the training and testing sets. This is done by using the standard sklearn train_test_split method.

```
In [201]: 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)
```

Modelling

K-Nearest Neighbors

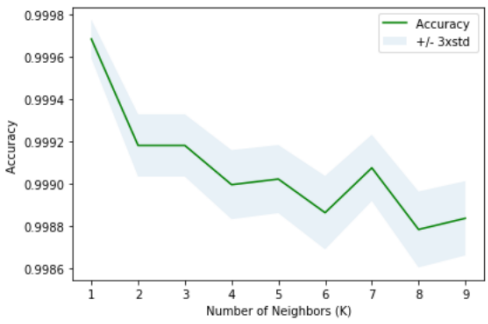
The first model I used to predict the severity values of crashes was KNN Classification, as it encapsulates a variety of different metrics and groups them by similarity to predict what unknown labels would be. I began by testing the model with a k value of 9, which led to an extremely accurate model with a high accuracy rating of 0.9988.

In [202]:
```
from sklearn.neighbors import KNeighborsClassifier
k = 9

k9neighbors = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat = k9neighbors.predict(X_test)
from sklearn import metrics

print("9 Nearest Neighbors Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
9 Nearest Neighbors Accuracy:  0.9988380690820746
```

However, to get the most out of the KNN algorithm, I had to iterate through several instances of K to find the best value for the model. I then plotted a graph of the accuracy of the KNN against the values of K that were being tested. I placed an upper limit of 10 due to the length of time the algorithm took to run, as training with a large dataset took around an hour to come up with a final model.

In [203]:
```
nKs = 10
ConfusionMatrix = [];
mean_acc = np.zeros((nKs-1))
std_acc = np.zeros((nKs-1))
for n in range(1,nKs):
    nkneighbors = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat= nkneighbors.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])

plt.plot(range(1,nKs),mean_acc,'g')
plt.fill_between(range(1,nKs),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [205]:
```
print("Best K Nearest Neighbors accuracy is", mean_acc.max(), "where k is", mean_acc.argmax()+1)
```

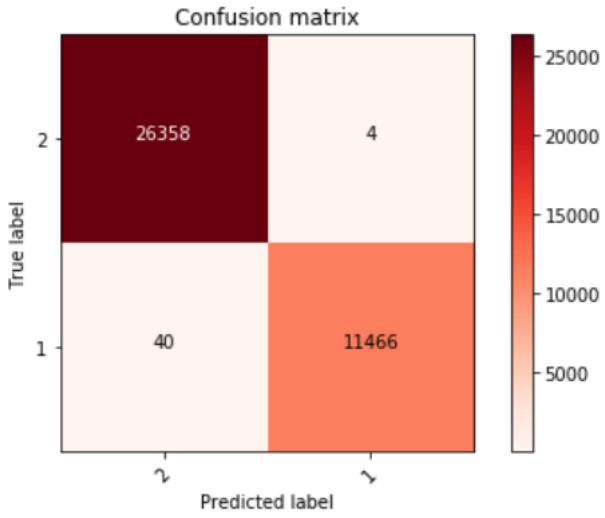
```
Best K Nearest Neighbors accuracy is 0.9996831097496567 where k is 1
```

We see here that the accuracy of the model varies a lot with K and is the most accurate with a K value of 1, which can be assumed as a result of overfitting from higher values of K. Therefore, we conclude that a KNN model is extremely accurate, but may be susceptible to some overfitting to the dataset.

Support Vector Machine

To test whether the KNN's accuracy was the best possible attainable by a predictor model I used an SVM algorithm with a radial basis kernel. The SVM splits the dataset into higher dimensional planes and solves for a hyperplane within the given dimension to classify the dataset. To check the accuracy of the model, we can plot a confusion matrix and check for 4 different scenarios: True Positives, True Negatives, False Positives, False Negatives.

```
Confusion matrix, without normalization
Accuracy score is 0.9994454420618992
```



Through this, we see that although the SVM was extremely accurate as well, with a rating of 0.9994, the KNN with a K of 1 is more accurate with a rating of 0.9997.

Results

Both the models were accurate in predicting the labels of the testing data set and acting as a gauge for whether certain driving conditions would lead to an injury accident or just a property damage one. To further evaluate the models, I used the Jaccard Index and F1 score in combination with standard accuracy measures (R^2) and tabled the SVM against the KNN. This was done by importing the standard functions from the sklearn metrics toolkit.

Algorithm	Jaccard	F1-score	Accuracy
KNN	0.9988381	0.9988376	0.9996
SVM	0.9994454	0.9994453	0.9994

Discussion

As is visible, the margin of error for each of these models is extremely low due to the large quantity of training data. Due to the extremely strong performance of the models, both would suffice as strong predictors for the Seattle Government to use to gauge the severity of accidents with given weather, road, and lighting conditions.

However, performance is also a key metric which arises when deciding between models for usage in a larger scale setting. As the KNN algorithm took almost an hour to iterate through several variations of K and 15 minutes for a specific K, the time complexity of this algorithm is quite slow. On the other hand, the SVM only took 5 minutes to run the same dataset and came up with predictions that were in an error range of 6 significant figures of the KNN. The Jaccard and F1 score were extremely tight as well, with both algorithms sporting strong readings in both tests.

Conclusion

Overall, based on the time performance and similar accuracy of the model, the Support Vector Machine is the better algorithm to model the impact of different driving conditions on the severity of an accident. The Seattle government can use this to deploy safety services and put up warning signs in regions where dangerous conditions are being exhibited and potentially lower the severity and number of accidents in the areas.