IBM Data Science Capstone – Vehicle Accident Severity

Background/Problem

Every time we get behind the wheel, we are inherently incur some risk while going from point A to point B. Sometimes conditions exist that dramatically increase that risk. This project aims to discover whether or not certain conditions significantly increase the severity of an accident so we can take precautions and avoid serious injury.

Data

Using the Seattle PD Collisions dataset, I will use various machine learning models to identify correlations between certain road conditions and the severity of a collision. We will target the SEVERITYCODE and use ROADCOND, LIGHTCOND, and WEATHER as independent variables.

Loading the Data

In [120]:
```
# Load Data
df = pd.read_csv("~/Desktop/ibm_datascience/9_capstone/data/Data-Collisions.csv")
df.head()
```

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN

5 rows × 38 columns

Here you can see the column titled SEVERITYCODE with integer values 1 and 2. These values indicate the seriousness of the accident, with 1 indicating property damage and 2 indicating bodily injury. This will be our target variable. There are several other columns that we will drop from the dataframe as they are not relevant to us for this project. The dependent variables we will consider are WEATHER, ROADCOND, and LIGHTCOND.

Preprocessing

In order to make the Data usable to us, we will need to clean and balance it. To do this we will remove all unneeded columns, convert datatypes, and balance the dataset to reflect an even split of SEVERITYCODE classes 1 and 2.

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	WEATHER_CAT	ROADCOND_CAT	LIGHTCOND_CAT
0	2	Overcast	Wet	Daylight	4	8	5
1	1	Raining	Wet	Dark - Street Lights On	6	8	2
2	1	Overcast	Dry	Daylight	4	0	5
3	1	Clear	Dry	Daylight	1	0	5
4	2	Raining	Wet	Daylight	6	8	5

Initially, the size of class 1 vs class 2 for SEVERITYCODE was nearly 3 to 1. In order balance the set, we downsampled the majority class to the size of the minority class resulting in an even split between class 1 and class 2.

```
# Balance target value SEVERITYCODE
df.SEVERITYCODE.value_counts()
```

```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

```
# Downsampling using resample
from sklearn.utils import resample

# Separate majority and minority classes
sc1 = df[df.SEVERITYCODE == 1]
sc2 = df[df.SEVERITYCODE == 2]

# Downsample majority labels equal to number of samples in minority
sc1 = sc1.sample(len(sc2), random_state = 0)

# Concat minority and majority dataframes
df_bal = pd.concat([sc1, sc2])

# Shuffle dataset to prevent bias
df_bal = df_bal.sample(frac = 1, random_state = 0)

df_bal.SEVERITYCODE.value_counts()
```

```
2    58188
1    58188
Name: SEVERITYCODE, dtype: int64
```

After cleaning, we are left with three independent variables and one target variable where all index values are random and shuffled to prevent bias.

	SEVERITYCODE	WEATHER_CAT	ROADCOND_CAT	LIGHTCOND_CAT
29959	2	1	0	5
110326	1	1	0	5
5970	2	1	0	2
130027	1	10	7	8
76480	2	6	5	2

Processing

Next we will define variables and normalize the dataset to prepare it for use in our models.

```
# Define X and y
X = np.asarray(df_bal[['WEATHER_CAT','ROADCOND_CAT','LIGHTCOND_CAT']])
X
```

```
array([[1, 0, 5],
       [1, 0, 5],
       [1, 0, 2],
       ...,
       [4, 8, 2],
       [4, 8, 2],
       [1, 0, 5]], dtype=int8)
```

```
y = np.asarray(df_bal['SEVERITYCODE'])
y
```

```
array([2, 1, 2, ..., 1, 1, 2])
```

```
# Normalize the dataset
from sklearn import preprocessing

X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X
```

```
array([[-0.67567888, -0.66964064,  0.42656848],
       [-0.67567888, -0.66964064,  0.42656848],
       [-0.67567888, -0.66964064, -1.21967041],
       ...,
       [ 0.41777987,  1.531829  , -1.21967041],
       [ 0.41777987,  1.531829  , -1.21967041],
       [-0.67567888, -0.66964064,  0.42656848]])
```

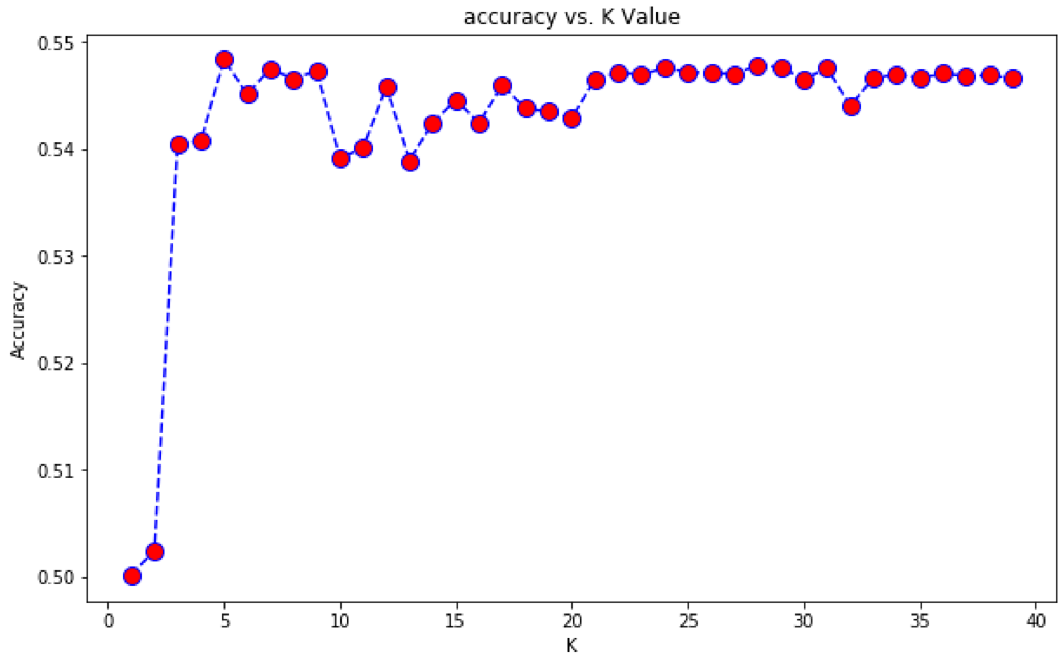
```
# Train/Test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 0)
print('Train set:', X_train.shape, y_train.shape)
print('Test set:', X_test.shape, y_test.shape)
```

```
Train set: (77971, 3) (77971,)
Test set: (38405, 3) (38405,)
```

I used 1/3 of the dataset for testing and 2/3 of the dataset for training, which seems to give the best results for accuracy and error rate as will be shown later. The models we will build are K-Nearest Neighbors, Logistic Regression, and Decision Tree.

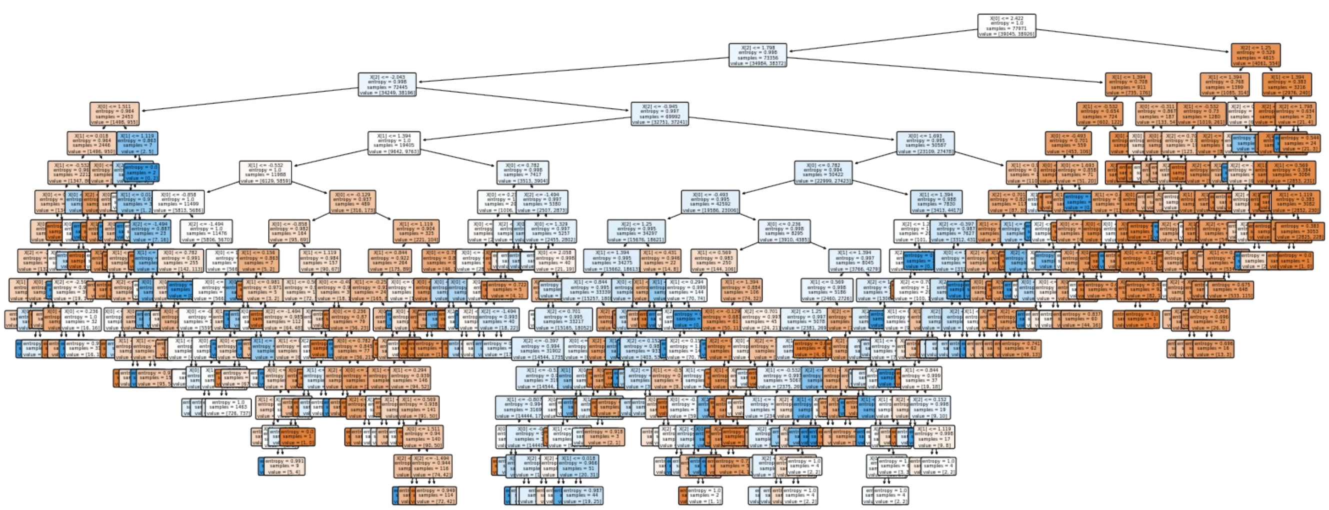
K-Nearest Neighbor

For our KNN model, we ran optimization to find the K-value with maximum accuracy. As we can see below, out optimal K based on accuracy was K=5.



Decision Tree

Our decision tree model was built using entropy criterion and a max depth of 16.



Logistic Regression

```python
# Building LR Model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```python
# Train Model & Predict
LRyhat = LR.predict(X_test)
LRyhat
```

```
array([2, 2, 2, ..., 1, 1, 2])
```

```python
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
array([[0.47172552, 0.52827448],
       [0.36388099, 0.63611901],
       [0.47172552, 0.52827448],
       ...,
       [0.53582723, 0.46417277],
       [0.68581436, 0.31418564],
       [0.47172552, 0.52827448]])
```

Accuracy

To check accuracy of our models we will use Jaccard Similarity and F-1 scores.

K-Nearest Neighbor

```
# Jaccard Similarity Score
jaccard_score(y_test,Kyhat)
```

```
0.348655421326252
```

```
# F1-score
f1_score(y_test,Kyhat, average='macro')
```

```
0.5448450196284904
```

Decision Tree

```
# Jaccard Similarity Score
jaccard_score(y_test,DTyhat)
```

```
0.26753132556605846
```

```
# F1-score
f1_score(y_test,DTyhat, average='macro')
```

```
0.5374508571541762
```

Logistic Regression

```
# Jaccard Similarity Score
jaccard_score(y_test,LRyhat)
```

```
0.27907348881865823
```

```
# F1-score
f1_score(y_test,LRyhat, average='macro')
```

```
0.5172228331543939
```

Results

Based on our models, there doesn't seem to be a significant correlation between the severity of an accident and the conditions we chose. More analysis is needed to discover which conditions significantly increase the severity of an accident. Perhaps scaling up the data to train on more data would have increased the accuracy of our models.