IBM Data Science Capstone: Car Accident Severity Report

Business Understanding

In an effort to reduce the frequency of car collisions in a community, an algorithm must be developed to predict the severity of an accident given the current weather, road, and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful. The model and its results should identify key causes of accidents and allow them to identify trends for when accidents can be prevented. This will reduce the number of accidents and injuries for the city.

Data Understanding

Using the data provided by Coursera on Collisions, I will investigate the connection between the severity of car accidents and weather conditions. This data provides collisions from 2004 to the present in Seattle.

The data has 37 independent variables and 194,673 records. The dependent variable, “SEVERITYCODE”, has numbers that correspond to different levels of severity caused by the accident. Many of the columns are object types. In addition, other columns that appear to be integer types are also actually objects, because the numbers correspond to different categories. Finally, some columns and rows have null values, which will be dealt with during the data pre-processing phase.

Data Preprocessing

The dataset in the original form is not ready for data analysis. In order to prepare the data, first, we need to drop the non-relevant columns. 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 four features, severity, weather conditions, road conditions, and light conditions, 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 [34]: #Showing the unique value counts of the SEVERITY CODE
         df['SEVERITYCODE'].value_counts()

Out[34]: 1    136485
         2     58188
         Name: SEVERITYCODE, dtype: int64
```

As you 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 down sampling the class 1.

```
In [38]: from sklearn.utils import resample

         # Splitting majority and minority classes
         df_majority = df[df.SEVERITYCODE==1]
         df_minority = df[df.SEVERITYCODE==2]

         #Turning majority class to Downsample majority class
         df_majority_downsampled = resample(df_majority,
                                            replace=False,
                                            n_samples=58188,
                                            random_state=123)

         # Combine minority class with Downsampled majority class
         df_balanced = pd.concat([df_majority_downsampled, df_minority])

         # Showing the unique value counts of the WEATHER
         df_balanced.SEVERITYCODE.value_counts()

Out[38]: 2    58188
         1    58188
         Name: SEVERITYCODE, dtype: int64
```

Methodology

For implementing the solution, I have used Watson Studio as a repository and running Jupyter Notebook to preprocess data and build Machine Learning models. Regarding coding, I have used Python and its 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"

Also, as I 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 condition ('LIGHTCOND'), to see the breakdowns of accidents occurring during the different light conditions. The results can be seen below:

```
In [35]: #Showing the unique value counts of the WEATHER
         df['WEATHER'].value_counts()

Out[35]: Clear                       111135
         Raining                      33145
         Overcast                     27714
         Unknown                      15091
         Snowing                        907
         Other                          832
         Fog/Smog/Smoke                 569
         Sleet/Hail/Freezing Rain       113
         Blowing Sand/Dirt               56
         Severe Crosswind                25
         Partly Cloudy                    5
         Name: WEATHER, dtype: int64
```

```
In [36]: #Showing the unique value counts of the ROAD CONDITION
         df['ROADCOND'].value_counts()

Out[36]: Dry                124510
         Wet                 47474
         Unknown             15078
         Ice                  1209
         Snow/Slush           1004
         Other                 132
         Standing Water        115
         Sand/Mud/Dirt          75
         Oil                    64
         Name: ROADCOND, dtype: int64
```

```
In [37]: #Showing the unique value counts of the LIGHT CONDITION
         df['LIGHTCOND'].value_counts()

Out[37]: Daylight                    116137
         Dark - Street Lights On      48507
         Unknown                      13473
         Dusk                          5902
         Dawn                          2502
         Dark - No Street Lights       1537
         Dark - Street Lights Off      1199
         Other                          235
         Dark - Unknown Lighting         11
         Name: LIGHTCOND, dtype: int64
```

After balancing SEVERITYCODE feature, and standardizing the input feature, the data has been ready for building machine learning models.
Our data is now ready to be fed into machine learning models.
We will use the following models:

K-Nearest Neighbor (KNN)

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

In [78]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

In [79]:
```
# Building the KNN Model
from sklearn.neighbors import KNeighborsClassifier
k = 23
knn = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)

knn_y_pred = knn.predict(X_test)
knn_y_pred[0:5]
```

Out[79]:
```
array([2, 2, 1, 1, 2])
```

In [80]:
```
jaccard_similarity_score(y_test, knn_y_pred)
```

Out[80]:
```
0.5640878755764328
```

In [81]:
```
f1_score(y_test, knn_y_pred, average='macro')
```

Out[81]:
```
0.5393282758446943
```

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 [125]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='entropy', max_depth = 7)

dt.fit(X_train,y_train)
```

Out[125]:
```
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 [126]:
```
dt_y_pred = dt.predict(X_test)
```

In [127]:
```
jaccard_similarity_score(y_test, knn_y_pred)
```

Out[127]:
```
0.5640878755764328
```

In [128]:
```
f1_score(y_test, dt_y_pred, average='macro')
```

Out[128]:
```
0.5450597937389444
```

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 [100]: from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import confusion_matrix

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

```
In [101]: LR_y_pred = LR.predict(X_test)
```

```
In [102]: LR_y_prob = LR.predict_proba(X_test)
          log_loss(y_test, LR_y_prob)

Out[102]: 0.6849535383198887
```

```
In [103]: jaccard_similarity_score(y_test, LR_y_pred)

Out[103]: 0.5260218256809784
```

```
In [104]: f1_score(y_test, LR_y_pred, average='macro')

Out[104]: 0.511602093963383
```

Results and Evaluations

The final results of the model evaluations are summarized in the following table:

Model	Jaccard	F1-score	LogLoss
KNN	0.56	0.53	NA
Decision Tree	0.56	0.54	NA
LogisticRegression	0.52	0.51	0.68

Although the first two are ideal for this project, logistic regression made most sense because of its binary nature.

Evaluation metrics used to test the accuracy of our models were jaccard index, f-1 score and logloss for logistic regression. Choosing different k, max depth and hyparameter C values helped to improve our accuracy to be the best possible.

Conclusion

Based on the dataset provided for this capstone from weather, road, and light conditions 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).