It Was An Accident: A Data Science Study

By JP Carigma

An IBM Capstone Project via Coursera

INTRODUCTION/BUSINESS PROBLEM

According to the National Safety Council, there have been approximately 38,800 people who died as a result of automobile accidents in 2019. Furthermore, approximately 4.4 million people suffered severe injuries which merited them to be admitted for medical attention in the same year (*source*). In an ongoing effort to determine the severity of an automobile accident based on various conditional factors, a predictive model is required to be developed. Such a model will help as a preventive measure to reduce the amount of automobile collisions in a defined area. For purposes of this study, the focus will be on the city of Seattle, Washington.

DATA UNDERSTANDING

A data set centered around Seattle was collected by the Seattle Department of Transportation Traffic Management Division dating from 2004 to the present. This set contains records of reported accidents in this community and contains 37 independent variables with a total of 194,673 rows of data. In this data set, 'SEVERITYCODE' will serve as the target variable as this measures the severity of an accident from a scale of 0 to 4.

The severity codes are defined as follows:

0: Little to no Probability – Unknown

1: Very Low Probability – Property damage

2: Low Probability – Injury

3: High Probability – Fatality

4: Highest Probability – Severe

In addition to the defined target variable, the following attributes are factors on the severity of an accident: 'WEATHER', 'ROADCOND', and 'LIGHTCOND'. Lastly, there are null values seen in several records; therefore, the data will need to be pre-processed.

METHODOLOGY

From the outset, the provided data set will still need to go under transformation prior to thorough conclusive analysis. In order to proceed and as aforementioned, the focus of this study will be on the following columns:

1. SEVERITYCODE
2. WEATHER
3. ROADCOND
4. LIGHTCOND

These columns focus on the severity rating, weather conditions, road conditions, and light conditions respectively, which all impact safety trends while driving. All other columns are irrelevant for purposes of this research; thus, the most relevant columns are called:

```
df1 = df[['SEVERITYCODE','WEATHER','ROADCOND','LIGHTCOND']]
df1
```

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

It is worth noting that the latter three columns are classified as object data types and will thus need to be converted into numerical data types. This is achieved by using the label encoder function from the *Scikit-learn* library. As a result, the columns 'WEATHER', 'ROADCOND', and 'LIGHTCOND' are now converted to integer references and introduced as new columns appended with "_CATEGORY":

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

# Encode labels in columns WEATHER, ROADCOND, and LIGHTCOND.
df1['WEATHER_CATEGORY']= label_encoder.fit_transform(df1['WEATHER'].astype(str))
df1['ROADCOND_CATEGORY']= label_encoder.fit_transform(df1['ROADCOND'].astype(str))
df1['LIGHTCOND_CATEGORY']= label_encoder.fit_transform(df1['LIGHTCOND'].astype(str))
print(df1.head())
```

Furthermore, the data set is imbalanced after performing the *value_count()* function on the SEVERITYCODE column:

```
#Required for pre-processing to determine imbalance in data

df1["SEVERITYCODE"].value_counts()
```

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

This screenshot illustrates that items with severity code labeled as 1 is much larger than those of severity code 2. In order to resolve this issue, class 1 is reduced by using the *resample* function:

```
from sklearn.utils import resample

#Required to resolve imbalance of data

df_high = df1[df.SEVERITYCODE==1]
df_low = df1[df.SEVERITYCODE==2]

df_high_dsample = resample(df_high,
                           replace=False,
                           n_samples=58188,
                           random_state=123)

balanced_df = pd.concat([df_high_dsample, df_low])

balanced_df.SEVERITYCODE.value_counts()
```

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

As a result, the data set is now balanced and ready for further data processing and analysis. This data set is now eligible to be integrated into machine learning models. For purposes of this study, the following models will be deployed:

1. K-Nearest Neighbor (KNN)
2. Decision Tree
3. Logistic Regression

These models and their reasons for use are explained in the proceeding pages.

K-Nearest Neighbor (KNN)

This supervised learning-specific algorithm, which has pre-determined classified data points, helps predict a point when it considers the 'K' nearest points to it. In this study, this will be useful in evaluating predictive measures pertaining to the accident. We first identify our variables and prepare the training sets. In our study, a split of 30%/70% is used for testing purposes:

```
# Identify X

X = df2
X[0:5]
```

	SEVERITYCODE	WEATHER_CATEGORY	ROADCOND_CATEGORY	LIGHTCOND_CATEGORY
0	2	4	8	5
1	1	6	8	2
2	1	4	0	5
3	1	1	0	5
4	2	6	8	5

```
# Identify y

y = df2['SEVERITYCODE'].values
y[0:5]
```

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

```
# Setting up for testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=7 )

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

```
Train set: (136271, 4) (136271,)
Test set: (58402, 4) (58402,)
```

Once the variables and test split are defined, we then proceed to identify the array as well as the Jaccard index and F1 scores, both help benchmark relative proximity of actual and predictive labels in a test set. Note that we are using 'k' value of 16:

```
# KNN Model

from sklearn.neighbors import KNeighborsClassifier

k = 16

knn = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)

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

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

```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

# Jaccard score
jaccard_knn = jaccard_similarity_score(y_test, knn_target)
print("KNN Jaccard index: ", jaccard_knn)

# F1 score
f1_score_knn = f1_score(y_test, knn_target, average='weighted')
print("KNN F1-score: ", f1_score_knn)
```

Decision Tree

A decision tree model will help analyze and map out all potential results from a given selection. For purposes of this study, this model will help determine all possible outcomes as a result of various conditions. Like the previous model, we also use decision tree analysis to determine the Jaccard index and F1 score:

```
#Decision Tree

from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
model_tree.fit(X_train,y_train)

model_tree
```

```
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')
```

```
yhat_dt = knn.predict(X_test)

# Jaccard score
jaccard_dt = jaccard_similarity_score(y_test, yhat_dt)
print("DT Jacard index: ", jaccard_dt)

# f1_score
f1_score_dt = f1_score(y_test, yhat_dt, average='weighted')
print("DT F1-score: ", f1_score_dt)
```

Logistic Regression

The logistic regression model is an appropriate model when laying out the probability of a binary dependent variable. In this study, SEVERITYCODE contains two values; therefore, this model may be appropriately tested. We follow a similar research pattern as the previous two models in which we also determined the Jaccard index and F1 score; however, in logistic regression, we now include log loss, which helps measure the accuracy of a classifier.

```
# Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
model_lr = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
model_lr
```

```
LogisticRegression(C=6, 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)
```

```
yhat_lg = knn.predict(X_test)
yhat_lg_prob = knn.predict_proba(X_test)

# Jaccard score
jaccard_lg = jaccard_similarity_score(y_test, yhat_lg)
print("LR Jaccard Index: ", jaccard_lg)

# f1_score
f1_score_lg = f1_score(y_test, yhat_lg, average = 'weighted')
print("LR F1-score: ", f1_score_lg)

# Log Loss
logloss_lg = log_loss(y_test, yhat_lg_prob)
print("LR log loss: ", logloss_lg)
```

RESULTS

The results of our models can now be consolidated and compared adjacent to each other in the following chart format:

```
report = pd.DataFrame(data=np.array([["KNN", jaccard_knn, f1_score_knn, 'NA'],
                                     ["Decision Tree", jaccard_dt, f1_score_dt, 'NA'],
                                    ["LogisticRegression", jaccard_lg, f1_score_lg, logloss_lg]]), columns=["Algorithm", "Jaccard", "
report = report.set_index(["Algorithm", "Jaccard", "F1-score", "LogLoss"])

report
```

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.564001947698565	0.5401775308974308	NA
Decision Tree	0.5664365709048206	0.5450597937389444	NA
LogisticRegression	0.5260218256809784	0.511602093963383	0684953583198887

DISCUSSION

We must recall that we were not able to rely on processing the data immediately once received. Throughout the course of this study, we see that the data was imbalanced and contained formats such as object defined columns. Such factors made it impossible to begin the transformation of the data for our analytic purposes. In order to combat this, we used libraries from Pandas, NumPy and Scikit-learn to bring the data to a more understanding position. Furthermore, we also analyzed the accuracy of our models by using metrics such as Jaccard index, F1 score and, for logistic regression, log loss; furthermore, by adjusting our variable "K" we can see the impact of our models' accuracy.

In order to predict the most appropriate model, we must understand what is presented to us in our data set. As we can see, the SEVERITYCODE column provides us a binary form of classification. Such a classification, therefore, is best processed in the logistic model.

CONCLUSION

In conclusion and based on this study, we see that this cumulative and historical data set shows that the various conditions from the weather, road and lighting have an impact on the outcome of driving in such conditions to a certain extent. Such outcomes may potentially result in either property or injury damage, i.e. severity codes 1 and 2 respectively.