Predicting Car Accident Severity

By: Prabhleen Singh

Table of Contents

Introduction & Business Understanding

Description of Problem

To minimize the number of car crashes in a city, an algorithm has to be developed to predict the severity of an accident given the current conditions of weather, road and visibility.

The main goal of this report is to shed light on why accidents happen, how they can be predicted based on environmental factors and how a model can be created that can alert drivers to be careful or postpone their road trip. This could be a business proposition for automobile sector as security/caretaking systems like these could be a game-changer.

Another target audience could be the local health institutes, police, insurance companies etc. The use of this model will allow them to be aware of all circumstances and provide services to aid the victims.

Background

In most cases, carelessness while driving, drug use, drunk driving, speeding, and many other crimes are the primary cause of injuries that can be prevented by stringent laws enforced.

In addition to the above causes, weather, visibility or road conditions are major uncontrollable variables that can be prevented by uncovering data-hidden trends and advising local government, police and drivers on highways, targeted highways and alarm systems.

Data Understanding

Description & Data Source

The data is obtained from the example data set in the course. It is Type of Collisions data from Seattle, WA, provided by the Seattle Police Department and recorded by Traffic Records in a timeframe of 2004 to present.

The data consists of 37 independent variables and 194, 673 rows. The dependent variable and also the predictor that we will use for our model is “SEVERITYCODE” that contains numbers encoding different levels of severity caused by accident from 0 to 4.

Severity codes are as follows:

0: Little to no Probability (Clear Condition)

1: Very Low Probability – Chance of Property Damage

2: Low Probability – Chance of Injury

3: Mild Probability – Chance of Serious Injury

4: High Probability – Chance of Fatality

Data Set Summary

Data Set Basics	
Title	Collisions—All Years
Abstract	All collisions provided by SPD and recorded by Traffic Records.
Description	This includes all types of collisions. Collisions will display at the intersection or mid-block of a segment. Timeframe: 2004 to Present.
Supplemental Information	
Update Frequency	Weekly
Keyword(s)	SDOT, Seattle, Transportation, Accidents, Bicycle, Car, Collisions, Pedestrian, Traffic, Vehicle

The data is unbalanced in some attributes and existence of null values is there in many records. The Data needs to be preprocessed, cleaned & balanced before analytics.

Use of Data

In our use case, we can see that Severity is impacted by several attributes, the most important ones being “WEATHER”, “ROADCOND” and “LIGHTCOND”.

These features have the highest influence over the accuracy of the predictions.

Preprocessing

Before we begin lets get some info on the data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194673 entries, 0 to 194672
Data columns (total 38 columns):
SEVERITYCODE      194673 non-null int64
X                 189339 non-null float64
Y                 189339 non-null float64
OBJECTID          194673 non-null int64
INCKEY            194673 non-null int64
COLDETKEY         194673 non-null int64
REPORTNO          194673 non-null object
STATUS            194673 non-null object
ADDRTYPE          192747 non-null object
INTKEY            65070 non-null float64
LOCATION          191996 non-null object
EXCEPTRSNCODE     84811 non-null object
EXCEPTRSNDESC     5638 non-null object
SEVERITYCODE.1    194673 non-null int64
SEVERITYDESC      194673 non-null object
COLLISIONTYPE     189769 non-null object
PERSONCOUNT       194673 non-null int64
PEDCOUNT          194673 non-null int64
PEDCYLCOUNT       194673 non-null int64
VEHCOUNT          194673 non-null int64
INCDATE           194673 non-null object
INCDTTM           194673 non-null object
JUNCTIONTYPE      188344 non-null object
SDOT_COLCODE      194673 non-null int64
SDOT_COLDESC      194673 non-null object
INATTENTIONIND    29805 non-null object
UNDERINFL         189789 non-null object
WEATHER           189592 non-null object
ROADCOND          189661 non-null object
LIGHTCOND         189503 non-null object
PEDROWNOTGRNT     4667 non-null object
SDOTCOLNUM        114936 non-null float64
SPEEDING          9333 non-null object
ST_COLCODE        194655 non-null object
ST_COLDESC        189769 non-null object
SEGLANEKEY        194673 non-null int64
CROSSWALKKEY      194673 non-null int64
HITPARKEDCAR      194673 non-null object
dtypes: float64(4), int64(12), object(22)
memory usage: 56.4+ MB
```

These features “WEATHER”, “ROADCOND” AND “LIGHTCOND” are of type objects. We have to convert these to categorical variables for classification. Before that let us check the value counts for these.

```
df['LIGHTCOND'].value_counts()
```

```
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
```

```
df['ROADCOND'].value_counts()
```

```
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
```

```
df['WEATHER'].value_counts()
```

```
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
```

Now let us check the balance of the data.

```
df['SEVERITYCODE'].value_counts()
```

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

We can see that the data is not balanced. For 'SEVERITYCODE' 1 there are 136485 and for 'SEVERITYCODE' 2 there are 58188 entries only.

We need to now downsample the first 136485 readings to 58188 readings to create balance in data.

```
from sklearn.utils import resample

df_maj = df[df['SEVERITYCODE'] == 1]
df_min = df[df['SEVERITYCODE'] == 2]

df_sample = resample(df_maj, replace= False, n_samples=58188, random_state = 1)
df = pd.concat([df_sample,df_min])
```

```
df['SEVERITYCODE'].value_counts()
```

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

Now our data is ready, but we still need to convert 'WEATHER', 'ROADCOND', 'LIGHTCOND' to categorical variables for model building.

```
WEATHER          object
ROADCOND         object
LIGHTCOND        object
```

To

```
# CONVERT FROM OBJECTS TO CATEGORICAL VARIABLE

df = df.astype({"WEATHER":'category', "ROADCOND":'category', "LIGHTCOND":'category'})
df.dtypes
```

```
WEATHER          category
ROADCOND         category
LIGHTCOND        category
```

With this new data, we can create a categorical coded dataframe. This dataframe will be used for model building purposes.

```
df["WEATHER_C"] = df["WEATHER"].cat.codes
df["ROADCOND_C"] = df["ROADCOND"].cat.codes
df["LIGHTCOND_C"] = df["LIGHTCOND"].cat.codes
Feature = df[['WEATHER','ROADCOND','LIGHTCOND','WEATHER_C','ROADCOND_C','LIGHTCOND_C
X = np.asarray(Feature[['WEATHER_C','ROADCOND_C','LIGHTCOND_C']])
print(X[0:])

y = df['SEVERITYCODE'].values
print(y[0:])
Feature.head()
```

```
[[1 0 5]
 [1 0 2]
 [1 0 5]
 ...
 [1 0 5]
 [1 0 5]
 [1 0 6]]
[1 1 1 ... 2 2 2]
```

	WEATHER	ROADCOND	LIGHTCOND	WEATHER_C	ROADCOND_C	LIGHTCOND_C
88984	Clear	Dry	Daylight	1	0	5
166664	Clear	Dry	Dark - Street Lights On	1	0	2
53641	Clear	Dry	Daylight	1	0	5
123636	Overcast	Dry	Dark - Street Lights On	4	0	2
171163	Raining	Wet	Daylight	6	8	5

Methodology

Technology & Tools used

The facilitate the solution, I have used IBM Watson Studio on cloud to build the notebook, Github Repository to host it and Jupyter Notebook as environment.

Preliminary Analysis

We have already conducted a preliminary analysis in the process of pre-processing.

Model Building

The model is built in Jupyter Notebook using Python and its libraries Pandas, Numpy, Scikitlearn & Matplotlib.

We have our balanced dataset ready after preprocessing to build models.

Before we build our model, the first step is to Normalize the data.

Normalizing the Data

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
```

The next step is to create the Train-Test splits. An 80-20 split is standard for any project.

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=0.3)
print('Test set shape: ', X_test.shape, y_test.shape)
print('Training set shape: ', X_train.shape, y_train.shape)
```

```
Test set shape:  (34913, 3) (34913,)
Training set shape:  (81463, 3) (81463,)
```

Now we can start building our models.

K- Nearest Neighbours

KNN

```
from sklearn.neighbors import KNeighborsClassifier as knn

accuracy = []
#Train Model and Predict
for i in range(15,25):
    model = knn(i)
    model.fit(X_train, y_train)
    accuracy.append(model.predict(X_test))

for i in range(len(accuracy)):
    print(f"For k = {i+1} Score = {accuracy[i]}")
```

```
For k = 1 Score = [1 1 1 ... 1 1 1]
For k = 2 Score = [1 1 1 ... 1 1 1]
For k = 3 Score = [1 1 1 ... 1 1 1]
For k = 4 Score = [1 1 1 ... 1 1 1]
For k = 5 Score = [2 2 1 ... 1 2 1]
For k = 6 Score = [2 2 1 ... 1 2 1]
For k = 7 Score = [2 2 1 ... 1 2 1]
For k = 8 Score = [2 2 1 ... 1 2 1]
For k = 9 Score = [2 2 1 ... 1 2 1]
For k = 10 Score = [2 2 1 ... 1 2 1]
```

Decision Tree

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
dt.fit(X_train, y_train)
pt = dt.predict(X_test)
```

Logistic Regression

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train, y_train)
LRpred = LR.predict(X_test)
LRprob = LR.predict_proba(X_test)
```

Results & Evaluation

Results

The results of three models is shown below. With the exception of Logistic Regression on which the Log Loss score was calculated, rest were all calculated on Jaccard Score & F1 Score.

KNN

```
#FOR K = 24
model = knn(24)
model.fit(X_train, y_train)
accuracy = model.predict(X_test)
from sklearn.metrics import f1_score, jaccard_similarity_score, log_loss
print("F1-Score of KNN is : ", f1_score(y_test, accuracy, average='macro'))
print("Jaccard Score of KNN is : ", jaccard_similarity_score(y_test, accuracy))
```

```
F1-Score of KNN is :  0.5427680932865123
Jaccard Score of KNN is :  0.5434078996362387
```

DECISION TREE

```
print("F1-Score of Decision Tree is : ", f1_score(y_test, pt, average='macro'))
print("Jaccard Score of Decision Tree is : ", jaccard_similarity_score(y_test, pt))
```

```
F1-Score of Decision Tree is :  0.5366857298388111
Jaccard Score of Decision Tree is :  0.5590467734081861
```

LOGISTIC REGRESSION

```
print("F1-Score of Logistic Regression is : ", f1_score(y_test, LRpred, average='macro'))
print("Jaccard Score of Logistic Regression is : ", jaccard_similarity_score(y_test, LRpred))
print("LogLoss of Logistic Regression is : ", log_loss(y_test, LRprob))
```

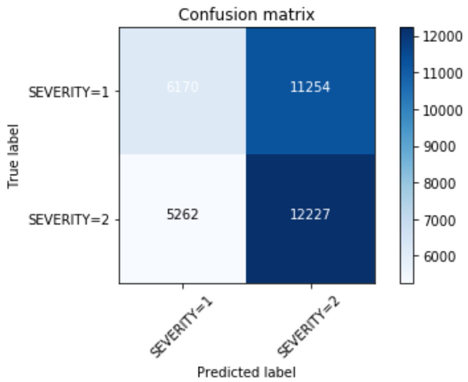
```
F1-Score of Logistic Regression is :  0.5122582306973823
Jaccard Score of Logistic Regression is :  0.5269383897115688
LogLoss of Logistic Regression is :  0.6846604312217874
```

Confusion Matrix

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, LRpred, labels=[1,2])
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['SEVERITY=1','SEVERITY=2'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 6170 11254]
 [ 5262 12227]]
```



Evaluation

	KNN	DECISION TREE	LOGISTIC REGRESSION
F1	0.54	0.53	0.51
JACCARD	0.54	0.55	0.52
LOGLOSS	NA	NA	0.68

We can see that from the evaluation report, the Decision Tree classification will be the best predictor for car accident.

Discussion

During the start of the solution, we had a large dataset consisting of many attributes that could possibly be useful for our case. But as we did some preliminary analysis, we saw that 'SEVERITYCODE' was the attribute we wanted to predict. We decided on the best parameters 'WEATHER', 'ROADCOND' and 'LIGHTCOND' to train the model.

We saw that the dataset was heavily unbalanced. So we balanced it by downsampling it. We also had to convert the datatype of these parameters from object to category for model building purposes.

The 'SEVERITYCODE' initially was a binary classification problem, we only had to predict class 1 or class 2. This gave an impression that logistic regression would be the best fit in the scenario. Contrary to our assumption, the Decision Tree model performed better on all the tests. We can still improve the model by tuning the hyperparameters like k in KNN, depth in Decision Tree & C parameter in Logistic regression.

Conclusion

Based on the data to start, we can now predict that a car accident can fall into two categories of severity, class 1 or class 2. We did this by examining the impact of Weather, Road Condition & Lighting conditions on SeverityCode attribute in the past records. Training a classifier model and finally finding a solution.