

# Capstone Project-Car accident severity

## Introduction/Business Understanding

In this project we are going to predict the severity of an accident given the current weather, road and visibility conditions. Our predictor or target variable will be 'SEVERITYCODE' because it is used to measure the severity of an accident from 0 to 5 within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.

## DATA

The predictor variable will be 'SEVERITYCODE' because it is used to measure the severity of an accident within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND'. The key features used for analysis are 'SEVERITYCODE', 'ADDRTYPE', 'JUNCTIONTYPE', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT'.

## Extract Dataset & Convert

In its original form, this data is not fit for analysis. For one, there are many columns that we will not use for this model. Also, most of the features are of type object, when they should be numerical type.

We must use label encoding to convert the features to our desired data type.

```
# Label Encoding
```

```
# Convert column to category
```

```
df_new["WEATHER"] = df_new["WEATHER"].astype('category')
```

```
df_new["ROADCOND"] = df_new["ROADCOND"].astype('category')
```

```
df_new["LIGHTCOND"] = df_new["LIGHTCOND"].astype('category')
```

```
# Assign variable to new column for analysis
```

```
df_new["WEATHER_CAT"] = df_new["WEATHER"].cat.codes
```

```
df_new["ROADCOND_CAT"] = df_new["ROADCOND"].cat.codes
```

```
df_new["LIGHTCOND_CAT"] = df_new["LIGHTCOND"].cat.codes
```

```
df_new.head(5)
```

The screenshot shows a Jupyter Notebook environment. The top bar indicates the notebook is named 'Business Understanding' and is autosaved. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The main area displays a pandas DataFrame with the following columns: SEVERITYCODE, ADORTYPE, JUNCTIONTYPE, WEATHER, ROADCOND, LIGHTCOND, WEATHER\_CAT, ROADCOND\_CAT, and LIGHTCOND\_CAT. The DataFrame contains 5 rows of data. Below the DataFrame, the output of a pandas command is shown: `df_new["SEVERITYCODE"].value_counts()`. The output is a Series with two categories: 1 (126276) and 2 (56638), with a total count of 182914. The dtype is int64. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating 12:03 AM on 10/6/2020.

```
Out[65]:
```

SEVERITYCODE	ADORTYPE	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND	WEATHER_CAT	ROADCOND_CAT	LIGHTCOND_CAT	
0	2	Intersection	At Intersection (intersection related)	Overcast	Wet	Daylight	4	8	5
1	1	Block	Mid-Block (not related to intersection)	Raining	Wet	Dark - Street Lights On	6	8	2
2	1	Block	Mid-Block (not related to intersection)	Overcast	Dry	Daylight	4	0	5
3	1	Block	Mid-Block (not related to intersection)	Clear	Dry	Daylight	1	0	5
4	2	Intersection	At Intersection (intersection related)	Raining	Wet	Daylight	6	8	5

```
In [67]: df_new["SEVERITYCODE"].value_counts()
Out[67]: 1    126276
         2     56638
         Name: SEVERITYCODE, dtype: int64

In [68]: df_new["WEATHER"].value_counts()
```

## Balancing the Dataset

Our target variable SEVERITYCODE is only 42% balanced. In fact, severitycode in class 1 is nearly three times the size of class 2.

We can fix this by downsampling the majority class.

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

Perfectly balanced.

## Methodology

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.

## Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. In context, the decision tree observes all possible outcomes of different weather conditions.

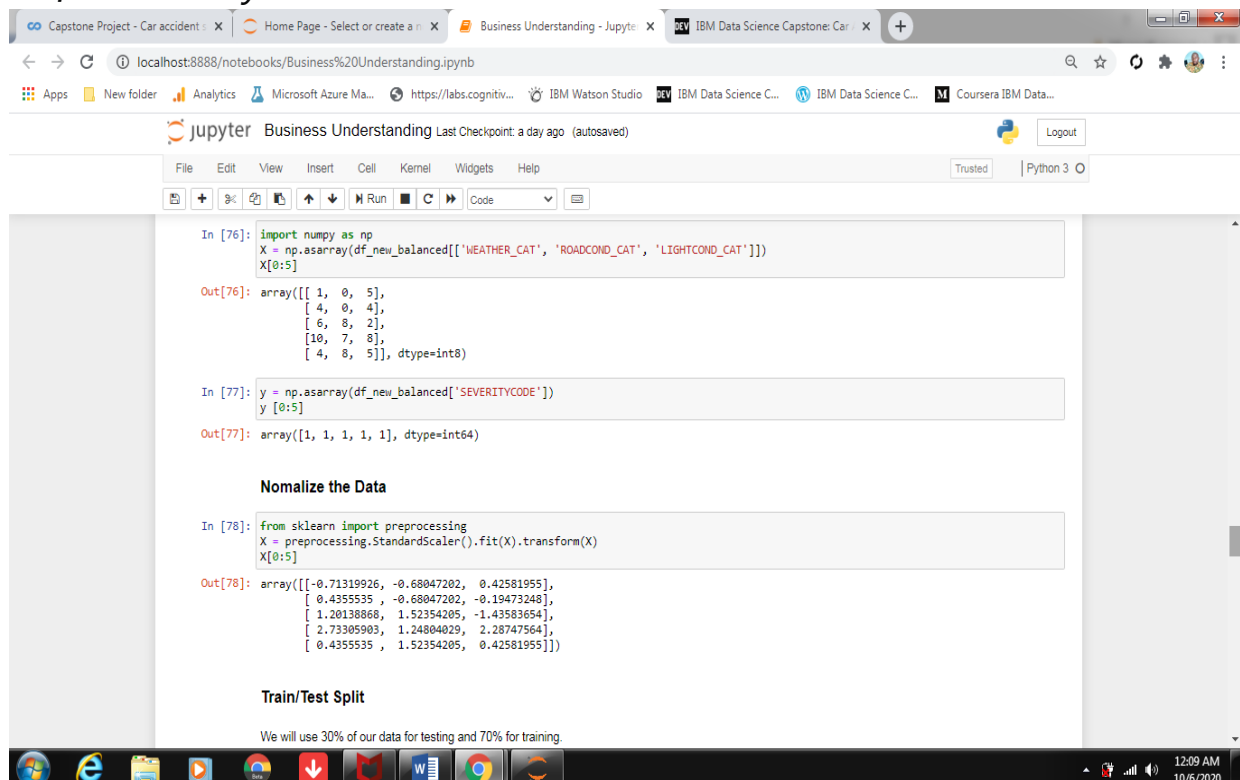
## Logistic Regression

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.

Let's get started!

## Initialization

### *Define X and y*



The screenshot shows a Jupyter Notebook titled "Business Understanding" with the following code and output:

```
In [76]: import numpy as np
X = np.asarray(df_new_balanced[['WEATHER_CAT', 'ROADCOND_CAT', 'LIGHTCOND_CAT']])
X[0:5]
```

```
Out[76]: array([[ 1,  0,  5],
 [ 4,  0,  4],
 [ 6,  8,  2],
 [10,  7,  8],
 [ 4,  8,  5]], dtype=int8)
```

```
In [77]: y = np.asarray(df_new_balanced['SEVERITYCODE'])
y[0:5]
```

```
Out[77]: array([1, 1, 1, 1, 1], dtype=int64)
```

**Normalize the Data**

```
In [78]: from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
Out[78]: array([[ -0.71319926, -0.68047202,  0.42581955],
 [ 0.4355535 , -0.68047202, -0.19473248],
 [ 1.20138868,  1.52354205, -1.43583654],
 [ 2.73305903,  1.24804029,  2.28747564],
 [ 0.4355535 ,  1.52354205,  0.42581955]])
```

**Train/Test Split**

We will use 30% of our data for testing and 70% for training.

## Train/Test Split

We will use 30% of our data for testing and 70% for training.

```
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=4)
print('Train set:', X_train.shape, y_train.shape)
print('Test set:', X_test.shape, y_test.shape)
Train set: (80378, 3) (80378,)
Test set: (34448, 3) (34448,)
```

## K-Nearest Neighbors (KNN)

*# Building the KNN Model*

```
from sklearn.neighbors import KNeighborsClassifier
```

```
k = 25
```

*#Train Model & Predict*

```
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
Kyhat = neigh.predict(X_test)
Kyhat[0:5]
array([2, 1, 2, 2, 2], dtype=int64)
```

## Decision Tree

*#Building the Decision Tree*

```
from sklearn.tree import DecisionTreeClassifier
```

```
df_newTree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
```

```
df_newTree
```

```
df_newTree.fit(X_train,y_train)
```

*# Train Model & Predict*

```
DTyhat = df_newTree.predict(X_test)
```

## Logistic Regration

*# Building the 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
```

Out[91]:

```
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, verbo
se=0,
```

```

warm_start=False)

# Train Model & Predict
LRyhat = LR.predict(X_test)
LRyhat
array([1, 1, 2, ..., 1, 2, 2], dtype=int64)
yhat_prob = LR.predict_proba(X_test)
yhat_prob
array([[0.54940819, 0.45059181],
       [0.52458009, 0.47541991],
       [0.45649265, 0.54350735],
       ...,
       [0.50632209, 0.49367791],
       [0.48136103, 0.51863897],
       [0.45649265, 0.54350735]])

```

## Results & Evaluation

Now we will check the accuracy of our models.

```

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

```

## K-Nearest Neighbor

In [95]:

```

# Jaccard Similarity Score
jaccard_similarity_score(y_test, Kyhat)

```

```

C:\Users\SIDNEY\anaconda3\lib\site-packages\sklearn\metrics\_classification
.py:664: FutureWarning: jaccard_similarity_score has been deprecated and re
placed with jaccard_score. It will be removed in version 0.23. This impleme
ntation has surprising behavior for binary and multiclass classification ta
sks.

```

```

FutureWarning)
0.5328611240130051
# F1-SCORE
f1_score(y_test, Kyhat, average='macro')

```

```

0.4921075848964933
Model is most accurate when k is 25.

```

## Decision Tree

```

# Jaccard Similarity Score
jaccard_similarity_score(y_test, DTyhat)
0.5465048769159313
# F1-SCORE
f1_score(y_test, DTyhat, average='macro')

```

```

0.5203507728191309
Model is most accurate with a max depth of 7.

```

# Logistic Regression

*# Jaccard Similarity Score*

```
jaccard_similarity_score(y_test, LRyhat)  
0.5215977705527172
```

*# F1-SCORE*

```
f1_score(y_test, LRyhat, average='macro')
```

```
0.5097825024463251
```

*# LOGLOSS*

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

```
log_loss(y_test, yhat_prob)
```

```
0.6868866924905638
```

Model is most accurate when hyperparameter C is 6.

## Discussion

In the beginning of this notebook, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so label encoding was used to create new classes that were of type int8; a numerical data type.

After solving that issue we were presented with another - imbalanced data. As mentioned earlier, class 1 was nearly three times larger than class 2. The solution to this was downsampling the majority class with sklearn's resample tool. We downsampled to match the minority class exactly with 58188 values each.

Once we analyzed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbor, Decision Tree and Logistic Regression. Although the first two are ideal for this project, logistic regression made the 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 hyperparameter C values helped to improve our accuracy to be the best possible.

## Conclusion

Based on historical data from weather conditions pointing to certain classes, we can conclude that particular weather conditions have a somewhat impact on whether or not travel could result in property damage (class 1) or injury (class 2).

Type *Markdown* and LaTeX:  $\alpha^2$