IBM Data Science Capstone: Car Accident Severity Report

Introduction | Business Understanding

In the final capstone project, I would like to develop an algorithm to predict the severity of an accident using data collected by Seattle SPOT Traffic Management Division. The data was updated weekly from 2004 till present and contains information such as severity code, address type, location, collision type, weather, road condition, speeding etc. The algorithm must be developed to reduce the frequency of car collisions, e.g when conditions are bad, this model will alert drivers to remind them to be more careful.

Data Understanding

There are 38 variables in this data set. Since we would like to identify the factors that cause the accident and the level of severity, we will use SEVERITYCODE as our dependent variable Y, and try different combinations of independent variables X to get the result. Since the observations are quite large, we may need to filter out the missing value and delete the unrelated columns first. Then we can select the factor which may have more impact on the accidents, such as 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.

Methodology

We used Jupyter Notebook to perform data analysis. To generate the table and graph for the dataset, we imported Python libraries (Pandas, Numpy, Matplotlib, Seaborn). First, we imported the data through pd.read_csv. We noticed that data not fit for analysis, for example here 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 had used cat.codes to convert category labels to numerical values.

	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

In addition, we have created three dictionaries for convenient matching of each category:

```
c1 = df.WEATHER.astype('category')
Weather_dictionary = dict(enumerate(c1.cat.categories))
print (Weather_dictionary)
```

```
{0: 'Blowing Sand/Dirt', 1: 'Clear', 2: 'Fog/Smog/Smoke', 3: 'Other', 4: 'Overcast', 5: 'Partly Cloudy', 6: 'Raining', 7: 'Severe Crosswind', 8: 'Sleet/Hail/F
reezing Rain', 9: 'Snowing', 10: 'Unknown'}
```

```
c2 = df.ROADCOND.astype('category')
Roadcondition_dictionary = dict(enumerate(c2.cat.categories))
print (Roadcondition_dictionary)
```

```
{0: 'Dry', 1: 'Ice', 2: 'Oil', 3: 'Other', 4: 'Sand/Mud/Dirt', 5: 'Snow/Slush', 6: 'Standing Water', 7: 'Unknown', 8: 'Wet'}
```

```
c3 = df.LIGHTCOND.astype('category')
Lightcondition_dictionary = dict(enumerate(c3.cat.categories))
print (Lightcondition_dictionary)
```

```
{0: 'Dark - No Street Lights', 1: 'Dark - Street Lights Off', 2: 'Dark - Street Lights On', 3: 'Dark - Unknown Lighting', 4: 'Dawn', 5: 'Daylight', 6: 'Dusk',
7: 'Other', 8: 'Unknown'}
```

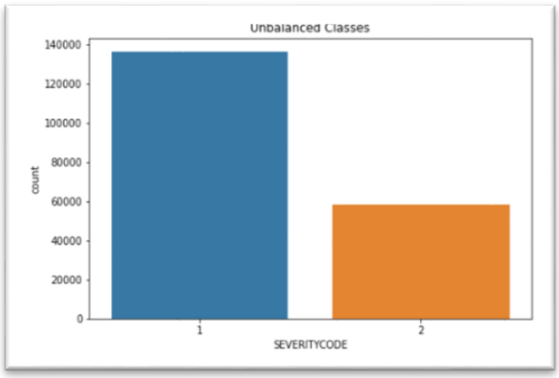
Data Balancing

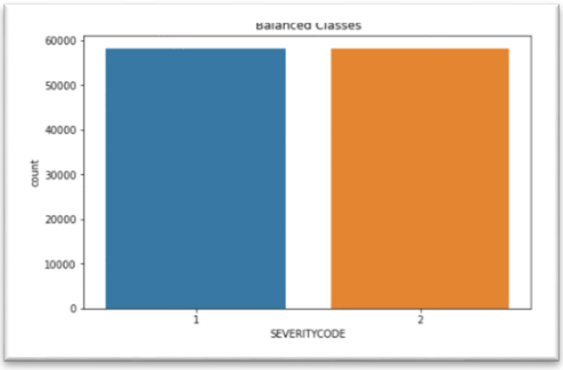
In fact, SEVERITYCODE in class 1 (prop damage) is nearly three times the size of class 2 (injury).

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

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

We can fix this by downsampling the majority class.





Methodology

Our data is now ready for 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.

Initialization

- Let's define feature sets, X and Y:

```
X = df[['WEATHER_CAT', 'ROADCOND_CAT','LIGHTCOND_CAT']] .values
X[0:5]
```

```
array([[6, 8, 5],
       [4, 0, 2],
       [1, 0, 5],
       [1, 0, 2],
       [1, 0, 5]], dtype=int8)
```

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

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

- Normalize dataset:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn
d to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
array([[ 1.15233872,  1.52995855,  0.42522086],
       [ 0.42149176, -0.67020526, -1.22653494],
       [-0.67477869, -0.67020526,  0.42522086],
       [-0.67477869, -0.67020526, -1.22653494],
       [-0.67477869, -0.67020526,  0.42522086]])
```

- 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: (81463, 3) (81463,)
Test set: (34913, 3) (34913,)
```

Modelling and Predictions

KNN (K-Nearest Neighbours)

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

```
k = 25
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=25, p=2,
           weights='uniform')
```

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

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

Decision Tree

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

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

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

```
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')
```

```
predTree = conTree.predict(X_test)
```

```
print (predTree [0:5])
print (y_test [0:5])
```

```
[2 1 2 2 2]
[1 1 2 2 2]
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=2, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=2, 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)
```

```
yhat2 = LR.predict(X_test)
yhat2
```

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

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

```
array([[0.46987822, 0.53012178],
       [0.46188403, 0.53811597],
       [0.47064745, 0.52935255],
       ...,
       [0.47064745, 0.52935255],
       [0.47064745, 0.52935255],
       [0.47064745, 0.52935255]])
```

Results & Evaluation

Now we should check accuracy of our models.

KNN:

```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

```
0.5610517572251024
```

```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average = 'macro')
```

```
0.5364827523846807
```

Decision Tree:

```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, predTree)
```

```
0.5605075473319394
```

```
from sklearn.metrics import f1_score
f1_score(y_test, predTree, average = 'macro')
```

```
0.49779972176286846
```

Logistic Regression:

```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat2)
```

```
0.526250966688626
```

```
from sklearn.metrics import f1_score
f1_score(y_test, yhat2, average = 'macro')
```

```
0.5118777863558591
```

```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

```
0.684623485318135
```

Discussion

When we had started analyze the data, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so we add columns with numerical interpretation of categorical data (WEATHER_CAT, LIGHTCOND_CAT, ROAD_CAT)

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 remove randomly data from majority class

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