Applied Capstone project - Car Accident Severity

September 19, 2020

1 Introduction/Business Problem

In an effort to reduce the frequency of car collisions in a community, an algorithim 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.

2 Data Description

Our predictor or target variable will be 'SEVERITYCODE' because it is used 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'.
Severity codes are as follows:
1. Little to no Probability (Clear Conditions)
2. Very Low Probability - Chance or Property Damage 3. Low Probability - Chance of Injury 4. Mild Probability - Chance of Serious Injury 5. High Probability - Chance of Fatality

Now we will import the necessary libraries & extract the dataset

[58]:
```
import os
import numpy as np
import pandas as pd
from sklearn.utils import resample
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

[4]:
```
os.chdir(r'C:\Users\avira\OneDrive\Desktop\Extra Notes')
```

[5]:
```
data = pd.read_csv('Data-Collisions.csv')
```

```
C:\Users\avira\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (33) have
```

```
mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[6]:
```
data
```

[6]:
```
        SEVERITYCODE           X          Y  OBJECTID  INCKEY  COLDETKEY  \
0                  2 -122.323148  47.703140         1    1307       1307
1                  1 -122.347294  47.647172         2   52200      52200
2                  1 -122.334540  47.607871         3   26700      26700
3                  1 -122.334803  47.604803         4    1144       1144
4                  2 -122.306426  47.545739         5   17700      17700
…                …           …          …         …       …          …
194668             2 -122.290826  47.565408    219543  309534     310814
194669             1 -122.344526  47.690924    219544  309085     310365
194670             2 -122.306689  47.683047    219545  311280     312640
194671             2 -122.355317  47.678734    219546  309514     310794
194672             1 -122.289360  47.611017    219547  308220     309500

       REPORTNO  STATUS      ADDRTYPE   INTKEY  … ROADCOND  \
0       3502005  Matched  Intersection  37475.0  …      Wet
1       2607959  Matched         Block      NaN  …      Wet
2       1482393  Matched         Block      NaN  …      Dry
3       3503937  Matched         Block      NaN  …      Dry
4       1807429  Matched  Intersection  34387.0  …      Wet
…           …       …           …        …  …      …
194668  E871089  Matched         Block      NaN  …      Dry
194669  E876731  Matched         Block      NaN  …      Wet
194670  3809984  Matched  Intersection  24760.0  …      Dry
194671  3810083  Matched  Intersection  24349.0  …      Dry
194672  E868008  Matched         Block      NaN  …      Wet

                      LIGHTCOND PEDROWNOTGRNT  SDOTCOLNUM SPEEDING ST_COLCODE  \
0                      Daylight           NaN         NaN      NaN         10
1       Dark - Street Lights On           NaN   6354039.0      NaN         11
2                      Daylight           NaN   4323031.0      NaN         32
3                      Daylight           NaN         NaN      NaN         23
4                      Daylight           NaN   4028032.0      NaN         10
…                           …           …         …       …        …
194668                 Daylight           NaN         NaN      NaN         24
194669                 Daylight           NaN         NaN      NaN         13
194670                 Daylight           NaN         NaN      NaN         28
194671                     Dusk           NaN         NaN      NaN          5
194672                 Daylight           NaN         NaN      NaN         14

                                               ST_COLDESC  SEGLANEKEY  \
0                                        Entering at angle           0
1       From same direction - both going straight - bo…           0
```

```
2                                  One parked--one moving            0
3                            From same direction - all others            0
4                                             Entering at angle            0
…                                                          …         …
194668    From opposite direction - both moving - head-on            0
194669   From same direction - both going straight - bo…          0
194670   From opposite direction - one left turn - one …          0
194671                         Vehicle Strikes Pedalcyclist        4308
194672   From same direction - both going straight - on…          0

        CROSSWALKKEY  HITPARKEDCAR
0                  0             N
1                  0             N
2                  0             N
3                  0             N
4                  0             N
…               …            …
194668             0             N
194669             0             N
194670             0             N
194671             0             N
194672             0             N

[194673 rows x 38 columns]
```

Now we will drop all the columns that are not required & cleanup the data

[9]:
```
data1 = data.drop(columns = ['OBJECTID', 'SEVERITYCODE.1', 'REPORTNO',␣
 ↪'INCKEY', 'COLDETKEY',
              'X', 'Y', 'STATUS','ADDRTYPE',
              'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
              'EXCEPTRSNDESC', 'SEVERITYDESC', 'INCDATE',
              'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE',
              'SDOT_COLDESC', 'PEDROWNOTGRNT', 'SDOTCOLNUM',
              'ST_COLCODE', 'ST_COLDESC', 'SEGLANEKEY',
              'CROSSWALKKEY', 'HITPARKEDCAR', 'PEDCOUNT', 'PEDCYLCOUNT',
              'PERSONCOUNT', 'VEHCOUNT', 'COLLISIONTYPE',
              'SPEEDING', 'UNDERINFL', 'INATTENTIONIND'])
```

[14]:
```
data1
```

[14]:
```
        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
```

```
...                    ...       ...      ...                     ...
194668                   2     Clear     Dry                   Daylight
194669                   1   Raining     Wet                   Daylight
194670                   2     Clear     Dry                   Daylight
194671                   2     Clear     Dry                       Dusk
194672                   1     Clear     Wet                   Daylight

[194673 rows x 4 columns]
```

In it's 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 covert the features to our desired data type:

[17]:
```
data1["WEATHER"] = data1["WEATHER"].astype('category')
data1["ROADCOND"] = data1["ROADCOND"].astype('category')
data1["LIGHTCOND"] = data1["LIGHTCOND"].astype('category')

data1["WEATHER_CAT"] = data1["WEATHER"].cat.codes
data1["ROADCOND_CAT"] = data1["ROADCOND"].cat.codes
data1["LIGHTCOND_CAT"] = data1["LIGHTCOND"].cat.codes
```

[20]:
```
data1.head()
```

[20]:
```
   SEVERITYCODE   WEATHER ROADCOND                LIGHTCOND  WEATHER_CAT  \
0             2  Overcast      Wet                 Daylight            4
1             1   Raining      Wet  Dark - Street Lights On            6
2             1  Overcast      Dry                 Daylight            4
3             1     Clear      Dry                 Daylight            1
4             2   Raining      Wet                 Daylight            6

   ROADCOND_CAT  LIGHTCOND_CAT
0             8              5
1             8              2
2             0              5
3             0              5
4             8              5
```

[22]:
```
data1.dtypes
```

[22]:
```
SEVERITYCODE        int64
WEATHER          category
ROADCOND         category
LIGHTCOND        category
WEATHER_CAT          int8
ROADCOND_CAT         int8
LIGHTCOND_CAT        int8
```

```
dtype: object
```

[24]:
```
data1.columns
```

[24]:
```
Index(['SEVERITYCODE', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'WEATHER_CAT',
       'ROADCOND_CAT', 'LIGHTCOND_CAT'],
      dtype='object')
```

2.1 Analyzing Value Counts

[27]:
```
data1["SEVERITYCODE"].value_counts()
```

[27]:
```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

[28]:
```
data1["WEATHER"].value_counts()
```

[28]:
```
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
```

[29]:
```
data1["ROADCOND"].value_counts()
```

[29]:
```
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
```

[30]:
```
data1["LIGHTCOND"].value_counts()
```

```
[30]: 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
```

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:

```
[33]: data1_majority = data1[data1.SEVERITYCODE==1]
data1_minority = data1[data1.SEVERITYCODE==2]

#Downsample majority class
data1_majority_downsampled = resample(data1_majority,
                                       replace=False,
                                       n_samples=58188,
                                       random_state=123)

data1_balanced = pd.concat([data1_majority_downsampled, data1_minority])

data1_balanced.SEVERITYCODE.value_counts()
```

```
[33]: 2    58188
      1    58188
      Name: SEVERITYCODE, dtype: int64
```

Now the data is ready to be analyzed.

3 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. It 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.

4 Initialization

Define x and y:

[35]:
```
x = np.asarray(data1_balanced[['WEATHER_CAT', 'ROADCOND_CAT', 'LIGHTCOND_CAT']])
x[0:5]
```

[35]:
```
array([[ 6,  8,  2],
       [ 1,  0,  5],
       [10,  7,  8],
       [ 1,  0,  5],
       [ 1,  0,  5]], dtype=int8)
```

[36]:
```
y = np.asarray(data1_balanced['SEVERITYCODE'])
y [0:5]
```

[36]:
```
array([1, 1, 1, 1, 1], dtype=int64)
```

Normalize the dataset

[39]:
```
x = preprocessing.StandardScaler().fit(x).transform(x)
x[0:5]
```

[39]:
```
array([[ 1.15236718,  1.52797946, -1.21648407],
       [-0.67488   , -0.67084969,  0.42978835],
       [ 2.61416492,  1.25312582,  2.07606076],
       [-0.67488   , -0.67084969,  0.42978835],
       [-0.67488   , -0.67084969,  0.42978835]])
```

Train/Test Split We will use 30% of our data for testing and 70% for training:

[41]:
```
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,)
```

K-Nearest Neighbors (KNN)

[43]:
```
k = 25
```

[45]:
```
neigh = KNeighborsClassifier(n_neighbors = k).fit(x_train,y_train)
neigh

Kyhat = neigh.predict(x_test)
Kyhat[0:5]
```

[45]:
```
array([2, 2, 1, 1, 2], dtype=int64)
```

Decision Tree

[47]:
```
# Building the Decision Tree
data1_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
data1_Tree
data1_Tree.fit(x_train,y_train)
```

[47]:
```
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')
```

[51]:
```
# Train Model & Predict
predTree = data1_Tree.predict(x_test)
print (predTree [0:5])
print (y_test [0:5])
```

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

Logistic Regression

[54]:
```
# Building the LR Model
LR = LogisticRegression(C=6, solver='liblinear').fit(x_train,y_train)
LR
```

[54]:
```
LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

[55]:
```
# Train Model & Predicr
LRyhat = LR.predict(x_test)
LRyhat
```

[55]:
```
array([1, 2, 1, …, 2, 2, 2], dtype=int64)
```

[56]:
```
yhat_prob = LR.predict_proba(x_test)
yhat_prob
```

[56]:
```
array([[0.57295252, 0.42704748],
       [0.47065071, 0.52934929],
       [0.67630201, 0.32369799],
       …,
       [0.46929132, 0.53070868],
       [0.47065071, 0.52934929],
       [0.46929132, 0.53070868]])
```

5 Results & Evaluation

Now we will check the accuracy of our models:

K-Nearest Neighbor

[59]:
```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, Kyhat)
```

```
C:\Users\avira\Anaconda3\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

[59]:
```
0.564001947698565
```

[60]:
```
# F1-SCORE
f1_score(y_test, Kyhat, average='macro')
```

[60]:
```
0.5401775308974308
```

Model is most accurate when k is 25.

Decision Tree

[61]:
```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, DTyhat)
```

```
C:\Users\avira\Anaconda3\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

[61]: 0.5664365709048206

[64]:
```
# F1-SCORE
f1_score(y_test, DTyhat, average='macro')
```

[64]: 0.5450597937389444

Model is most accurate with a max depth of 7.

Logistic Regression

[66]:
```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, LRyhat)
```

```
C:\Users\avira\Anaconda3\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

[66]: 0.5260218256809784

[67]:
```
# F1-SCORE
f1_score(y_test, LRyhat, average='macro')
```

[67]: 0.511602093963383

[68]:
```
#### LOGLOSS
yhat_prob = LR.predict_proba(x_test)
log_loss(y_test, yhat_prob)
```

[68]: 0.6849535383198887

Model is most accurate when hyperparameter C is 6.

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 created 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 hyperamater C values helped to improve our accuracy to be the best possible.

7 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).