Chicago Car Accidents Analysis

Justin Lee

This notebook is prepared for the Vehicle Safety Board of Chicago. This board aims to understand how often driver accidents are caused by a failure to yield. The purpose of this analysis is to build a model that accurately predicts how often failure to yield accidents are actually due to a failure to yield.

Data Understanding

This dataset is from the Chicago Data Portal. This data contains information about people involved in a crash and if any injuries were sustained. Each record corresponds to an occupant in a vehicle listed in the Crash dataset. Some people involved in a crash may not have been an occupant in a motor vehicle, but may have been a pedestrian, bicyclist, or using another non-motor vehicle mode of transportation. Person data can be linked with the Crash and Vehicle dataset using the “CRASH_RECORD_ID” field.

In [1]:

```
# Import any relevant library
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
from sklearn.impute import SimpleImputer
from imblearn.under_sampling import RandomUnderSampler
```

In [2]:

```
# Load in our dataframe
df = pd.read_csv('traffic_crashes.csv')

df.head()
```

```
/Users/justinlee/anaconda3/envs/learn-env/lib/python3.8/site-packages/IPython/core/intera
ctiveshell.py:3145: DtypeWarning: Columns (19,28) have mixed types.Specify dtype option o
n import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[2]:

	PERSON_ID	PERSON_TYPE	CRASH_RECORD_ID	VEHICLE_ID	CRASH_DATE	SEAT_NO
0	O749947	DRIVER	81dc0de2ed92aa62baccab641fa377be7feb1cc47e6554...	834816.0	09/28/2019 03:30:00 AM	NaN
1	O871921	DRIVER	af84fb5c8d996fcd3aefd36593c3a02e6e7509eeb27568...	827212.0	04/13/2020 10:50:00 PM	NaN
2	O10018	DRIVER	71162af7bf22799b776547132ebf134b5b438dcf3dac6b...	9579.0	11/01/2015 05:00:00 AM	NaN
3	O10038	DRIVER	c21c476e2ccc41af550b5d858d22aaac4ffc88745a1700...	9598.0	11/01/2015 08:00:00 AM	NaN
4	O10039	DRIVER	eb390a4c8e114c69488f5fb8a097fe629f5a92fd528cf4...	9600.0	11/01/2015 10:15:00 AM	NaN

5 rows × 29 columns

In [3]:
```
df.describe()
```

Out[3]:

	VEHICLE_ID	SEAT_NO	AGE	BAC_RESULT VALUE
count	1.964730e+06	405468.00000	1.422362e+06	2216.000000
mean	9.441771e+05	4.16478	3.792867e+01	0.171340
std	5.491011e+05	2.21842	1.708682e+01	0.103318
min	2.000000e+00	1.00000	-1.770000e+02	0.000000
25%	4.680342e+05	3.00000	2.500000e+01	0.127500
50%	9.363780e+05	3.00000	3.500000e+01	0.170000
75%	1.422413e+06	6.00000	5.000000e+01	0.220000
max	1.900249e+06	12.00000	1.100000e+02	1.000000

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2005877 entries, 0 to 2005876
Data columns (total 29 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   PERSON_ID              object 
 1   PERSON_TYPE            object 
 2   CRASH_RECORD_ID        object 
 3   VEHICLE_ID             float64
 4   CRASH_DATE             object 
 5   SEAT_NO                float64
 6   CITY                   object 
 7   STATE                  object 
 8   ZIPCODE                object 
 9   SEX                    object 
 10  AGE                    float64
 11  DRIVERS_LICENSE_STATE  object 
 12  DRIVERS_LICENSE_CLASS  object 
 13  SAFETY_EQUIPMENT       object 
 14  AIRBAG_DEPLOYED        object 
 15  EJECTION               object 
 16  INJURY_CLASSIFICATION  object 
 17  HOSPITAL               object 
 18  EMS_AGENCY             object 
 19  EMS_RUN_NO             object 
 20  DRIVER_ACTION          object 
 21  DRIVER_VISION          object 
 22  PHYSICAL_CONDITION     object 
 23  PEDPEDAL_ACTION        object 
 24  PEDPEDAL_VISIBILITY    object 
 25  PEDPEDAL_LOCATION      object 
 26  BAC_RESULT             object 
 27  BAC_RESULT VALUE       float64
 28  CELL_PHONE_USE         object 
dtypes: float64(4), object(25)
memory usage: 443.8+ MB
```

In [5]:
```
# Explore number of null values
df.isnull().sum()
```

Out[5]:
```
PERSON_ID                     0
PERSON_TYPE                   0
CRASH_RECORD_ID               0
VEHICLE_ID                41147
CRASH_DATE                    0
SEAT_NO                 1600409
CITY                     545926
STATE                    523661
ZIPCODE                  662176
SEX                       33887
AGE                      583515
DRIVERS_LICENSE_STATE    831360
DRIVERS_LICENSE_CLASS   1030131
SAFETY_EQUIPMENT           5603
AIRBAG_DEPLOYED           39600
EJECTION                  25264
INJURY_CLASSIFICATION       757
HOSPITAL                1682144
EMS_AGENCY              1806123
EMS_RUN_NO              1972477
DRIVER_ACTION            409055
DRIVER_VISION            409691
PHYSICAL_CONDITION       407959
PEDPEDAL_ACTION         1966543
PEDPEDAL_VISIBILITY     1966613
PEDPEDAL_LOCATION       1966542
BAC_RESULT               408136
BAC_RESULT VALUE        2003661
CELL_PHONE_USE          2004717
dtype: int64
```

In [6]:
```
# Explore our target variable counts
df['DRIVER_ACTION'].value_counts()
```

Out[6]:
```
NONE                                  568041
UNKNOWN                               406729
FAILED TO YIELD                       145118
OTHER                                 143764
FOLLOWED TOO CLOSELY                   93147
IMPROPER BACKING                       46726
IMPROPER TURN                          42036
IMPROPER LANE CHANGE                   41055
IMPROPER PASSING                       35904
DISREGARDED CONTROL DEVICES            28288
TOO FAST FOR CONDITIONS                23312
WRONG WAY/SIDE                          6449
IMPROPER PARKING                        5856
OVERCORRECTED                           3230
EVADING POLICE VEHICLE                  2473
CELL PHONE USE OTHER THAN TEXTING       2313
EMERGENCY VEHICLE ON CALL               1493
TEXTING                                  626
STOPPED SCHOOL BUS                       193
LICENSE RESTRICTIONS                      69
Name: DRIVER_ACTION, dtype: int64
```

Data Preparation

In order to prepare our analysis, we must prepare our data into binary classification. First we'll drop any unnecessary columns for our analylsis. These below columns will get dropped because they will not actually help us with our analysis. We'll handle the NONE, OTHER and UNKNOWN values to be set to null. Then we will create a new binary column where 1 represents FAILED TO YIELD and 0 represents all other driver actions. Finally, we will one-hot encode our categorical

In [7]:
```
# Columns to drop
columns_to_drop = ['HOSPITAL', 'EMS_AGENCY', 'EMS_RUN_NO', 'PERSON_ID', 'CRASH_RECORD_
                   'PERSON_TYPE', 'CRASH_DATE', 'CITY', 'STATE', 'ZIPCODE','SEAT_NO', '
                    'DRIVERS_LICENSE_CLASS', 'SAFETY_EQUIPMENT', 'AIRBAG_DEPLOYED', 'EJ
df.drop(columns=columns_to_drop, inplace=True)
```

In [8]:
```
# Convert NONE, OTHER and UNKOWN values to null
df['DRIVER_ACTION'] = df['DRIVER_ACTION'].replace(['UNKNOWN', 'NONE', 'OTHER'], np.nan
```

In [9]:
```
# Verify how many values are now missing
df['DRIVER_ACTION'].isnull().sum()
```

Out[9]:
```
1527589
```

In [10]:
```
# Creating a binary classification target, 1 = FAILED TO YIELD and 0 = all other value
df['target'] = (df['DRIVER_ACTION'] == 'FAILED TO YIELD').astype(int)
```

In [11]:
```
# Because there are lot of missing values, we will group these values into not failed
# This will help us not lose any data
df['DRIVER_ACTION'] = df['DRIVER_ACTION'].fillna('NOT_FAILED_TO_YIELD')
df['target'] = df['target'].fillna(0)  # Ensure all missing values are assigned to 0
```

In [12]:
```
# Because our target variable is cleaned, we can drop the DRIVER_ACTION column and sel
X = df.drop(columns=['DRIVER_ACTION', 'target'])
y = df['target']
```

In [13]:
```
# This shows us the class imbalance of our inital dataset
y.value_counts()
```

Out[13]:
```
0    1860759
1     145118
Name: target, dtype: int64
```

In [14]:
```
# One-hot encode our categorcial before our modeling
# Identify categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns
print(categorical_cols)
```

```
Index(['DRIVER_VISION', 'PHYSICAL_CONDITION', 'PEDPEDAL_ACTION',
       'PEDPEDAL_VISIBILITY', 'PEDPEDAL_LOCATION', 'BAC_RESULT',
       'CELL_PHONE_USE'],
      dtype='object')
```

In [15]:
```
# Apply one-hot encoding, get_dummies to convert categorical columns into numerical on
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)
```

Baseline Model - Logistic Regression

Logisic regression is a good baseline model because it is simple, interpretable and efficient to provide a strong foundation for comparison. This will help us determine what improvements will be needed when iterating on our model. We'll first impute our null values to fill with mode values so that we can maintain the integrity/size of our data. Then we will scale our features because logistic regression is sensitive to feature magnitudes.

In [16]:
```
# Split the dataset to evaluate the model's generalization ability
# stratify = y helps to ensure the "FAILED TO YIELD" accidents is maintained in both t
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [17]:
```
# Handling any missing NaN values in X_test or X_train or else this will roadblock us
# This imputer fills NaNs with the most frequent value (mode) for categorical and medi
imputer = SimpleImputer(strategy="most_frequent")

# Apply imputation to both X_train and X_test
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)
```

In [18]:
```
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [19]:
```
# Build a baseline logistic regression model
model = LogisticRegression(random_state=42, max_iter=1000)
model.fit(X_train, y_train)
```

Out[19]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

Baseline Model - Logistic Regression Evaluation

In [20]:
```
# Evaluate the baseline model
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9276601790735238
Confusion Matrix:
 [[372149      3]
 [ 29018      6]]
Classification Report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96    372152
           1       0.67      0.00      0.00     29024

    accuracy                           0.93    401176
   macro avg       0.80      0.50      0.48    401176
weighted avg       0.91      0.93      0.89    401176
```

Our model achieved 93% accuracy which looks great but this class is also very imbalanced.

Our model had 372,149 true negatives which is the total correctly predicated "NOT FAILED TO YIELD". We had 3 false positives which are the incorrectly predicted "FAILED TO YIELD" cases but were actually "NOT FAILED TO YIELD". Our model missed 29,018 actual "FAILED TO YIELD" cases. Our model only correctly predicted 6 "FAILED TO YIELD" cases.

Precision for Class 1 ("FAILED TO YIELD") was 0.67 but this is misleading due to only having 6 true positives. We also had a recall score of 0 and f1-score of 0. A recall of 0 means that our model is not detecting actual "FAILED TO YIELD" cases at all. Out of 29,024 actual "FAILED TO YIELD" cases it only found 6. This means it failes to identify accidents caused by "FAILED TO YIELD". Since our recall is 0, it makes sense that our f1-score is also 0. Our baseline model is completely missing the "FAILED TO YIELD" category.

Our baseline model is heavily influenced by class imbalance. We are seeing 372,152 "NOT FAILED TO YIELD" cases versus only 29,024 "FAILED TO YIELD" cases. Since "NOT FAILED TO YIELD" dominates the data, the model learns to always predict the majority class (0) because it minimizes overall errors.

We will now use statsmodels to understand the statistical significance and interpretability of our model.

In [21]:

```
# Add an intercept to X_train
X_train_sm = sm.add_constant(X_train)

# Fit the logistic regression model
model_sm = sm.Logit(y_train, X_train_sm)
result = model_sm.fit()

# Print the summary
print(result.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.239461
         Iterations: 35

/Users/justinlee/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/base/mo
del.py:566: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:              1604701
Model:                          Logit   Df Residuals:                  1604639
Method:                           MLE   Df Model:                           61
Date:                Fri, 21 Feb 2025   Pseudo R-squ.:                 0.07781
Time:                        15:01:55   Log-Likelihood:            -3.8426e+05
converged:                      False   LL-Null:                   -4.1669e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.7097      2.164     -1.715      0.086      -7.950       0.531
x1            -0.0039      0.004     -0.878      0.380      -0.013       0.005
x2             0.2478      0.006     44.191      0.000       0.237       0.259
x3             0.0496      0.003     17.132      0.000       0.044       0.055
x4             0.1304      0.003     37.993      0.000       0.124       0.137
x5             0.0483      0.002     20.074      0.000       0.044       0.053
x6             0.0525      0.002     22.666      0.000       0.048       0.057
x7             0.5488      0.012     46.839      0.000       0.526       0.572
x8             3.1476      0.085     36.973      0.000       2.981       3.315
x9             0.6252      0.015     40.525      0.000       0.595       0.655
x10            0.4341      0.009     46.998      0.000       0.416       0.452
x11            0.0329      0.002     15.190      0.000       0.029       0.037
x12            0.1325      0.004     36.826      0.000       0.125       0.140
x13            3.2067      0.085     37.909      0.000       3.041       3.373
x14            0.3173      0.008     38.703      0.000       0.301       0.333
x15           -0.0094      0.004     -2.151      0.032      -0.018      -0.001
x16            0.0077      0.003      2.488      0.013       0.002       0.014
x17           -0.0236      0.005     -4.988      0.000      -0.033      -0.014
x18            0.0119      0.005      2.406      0.016       0.002       0.022
x19            0.0036      0.003      1.242      0.214      -0.002       0.009
x20            0.0086      0.003      3.019      0.003       0.003       0.014
x21           -0.0050      0.004     -1.315      0.189      -0.013       0.002
x22            0.1152      0.032      3.633      0.000       0.053       0.177
x23           -0.0012      0.004     -0.273      0.785      -0.010       0.007
x24           -0.0095      0.005     -1.970      0.049      -0.019   -4.87e-05
x25            0.0726      0.028      2.581      0.010       0.017       0.128
x26            0.0321      0.004      8.749      0.000       0.025       0.039
x27           -0.0127      0.005     -2.647      0.008      -0.022      -0.003
x28           -0.0107      0.005     -2.109      0.035      -0.021      -0.001
x29           -0.0138      0.005     -2.797      0.005      -0.023      -0.004
x30           -0.0791      0.009     -8.557      0.000      -0.097      -0.061
x31            0.0211      0.002      9.379      0.000       0.017       0.025
x32           -0.0036      0.004     -1.027      0.304      -0.010       0.003
x33           -0.0497      0.007     -7.287      0.000      -0.063      -0.036
x34           -0.0029      0.004     -0.760      0.447      -0.010       0.005
x35           -0.0308      0.006     -5.095      0.000      -0.043      -0.019
x36           -0.0216      0.006     -3.873      0.000      -0.033      -0.011
x37           -0.0009      0.003     -0.323      0.747      -0.007       0.005
x38           -0.0865    585.362     -0.000      1.000   -1147.375    1147.202
x39           -0.0009      0.003     -0.261      0.794      -0.008       0.006
x40           -0.0640      0.012     -5.206      0.000      -0.088      -0.040
x41           -0.0015      0.004     -0.420      0.675      -0.009       0.006
x42            0.0073      0.003      2.563      0.010       0.002       0.013
x43            0.0049      0.003      1.897      0.058      -0.000       0.010
x44           -0.0528      0.007     -7.318      0.000      -0.067      -0.039
x45           -0.0482    306.409     -0.000      1.000    -600.599     600.502
x46           -0.0451      0.006     -7.159      0.000      -0.057      -0.033
x47           -0.0240      0.008     -2.847      0.004      -0.040      -0.007
x48           -0.0119      0.010     -1.224      0.221      -0.031       0.007
x49           -0.0098      0.005     -1.947      0.052      -0.020    6.61e-05
x50           -0.0282      0.007     -3.857      0.000      -0.043      -0.014
x51           -0.0107      0.006     -1.897      0.058      -0.022       0.000
x52           -0.0011      0.004     -0.260      0.795      -0.009       0.007
x53           -0.0456      0.010     -4.469      0.000      -0.066      -0.026
x54            0.0212      0.010      2.182      0.029       0.002       0.040
x55           -0.0055      0.005     -1.023      0.306      -0.016       0.005
x56           -0.0099      0.006     -1.632      0.103      -0.022       0.002
```

```
x57           -0.0262      0.007     -3.747      0.000      -0.040      -0.013
x58            0.0051      0.003      1.874      0.061      -0.000       0.010
x59           -0.0111      0.003     -3.697      0.000      -0.017      -0.005
x60            0.0061      0.004      1.454      0.146      -0.002       0.014
x61            0.0031      0.003      1.188      0.235      -0.002       0.008
==============================================================================
```

Overall our model is not detecting meaningful relationships between features and "FAILED TO YIELD". Pseudo R-squared compares the log-likelihood of our model versus a null model (a model with no predictors) so a lower score suggests a model doesn't explain much variation. Our pseudo R-squared value of 0.07781 means our model doesn't explain much variation in the outcome. Logistic regression maximizes the log-likelihood to find the best-fit model, so a higher value suggests a better fit. Our log-likelihood means of -3.8426e+05 suggests a week model, which could be due to a large class imbalance making our model predict mostly 0s - inflating accuracy but weakening predictive power. All p-values are high, meaning features may not be relevant predictors. This model also did not converge which is likely due to our high class imbalance.

Resampled Model

We need to balance the dataset. As we saw earlier our target column had one class's value almost be 10x the amount of values of the other class. For this reason we will balance the dataset using an under sampling technique. Under sampling removes excess majority class samples and ensures the model learns from real data only, it helps the model focus to learn from minority class cases instead of predicting "NOT FAILED TO YIELD" cases, and it would help us increase our F1-score as we strive for a healthy balance between recall and precision.

In [22]:
```
# Apply random under sampling
undersampler = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = undersampler.fit_resample(X_train, y_train)

# Check the new class distribution
print("New class distribution after undersampling:\n", y_train_resampled.value_counts(
```

```
New class distribution after undersampling:
 1    116094
0    116094
Name: target, dtype: int64
```

In [23]:
```
# Train our new model on under sampled data
model = LogisticRegression(random_state=42, max_iter=1000)
model.fit(X_train_resampled, y_train_resampled)
```

Out[23]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

In [24]:
```
# Make predictions on the original dataset
y_pred_undersampled = model.predict(X_test)
```

Resampled Model - Evaluation

Now, we will test our model on the original (imbalanced) test set to see if recall, precision and f1-score have improved.

In [25]:
```
# Print evaluation results
print("Accuracy:", accuracy_score(y_test, y_pred_undersampled))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_undersampled))
print("Classification Report:\n", classification_report(y_test, y_pred_undersampled))
```

```
Accuracy: 0.29737322272518796
Confusion Matrix:
 [[ 90673 281479]
 [   398  28626]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.24      0.39    372152
           1       0.09      0.99      0.17     29024

    accuracy                           0.30    401176
   macro avg       0.54      0.61      0.28    401176
weighted avg       0.93      0.30      0.38    401176
```

Our accuracy dropped from 92.76% to 29.73%. This drop is expected due to balancing the dataset.

Our precision from our baseline model to our resampled balanced model went from 0.67 to 0.09, meaning we have more false positives now. Our recall jumped from 0 to 0.99 and our F1-score also increased from 0 to 0.17.

We have 90,673 true negatives, meaning these are the cases our model correctly predicted "NOT FAILED TO YIELD". We have 281,479 false positives, meaning our model incorrectly predicted "FAILED TO YIELD" when it was actually "NOT FAILED TO YIELD". We have 398 false negatives, meaning our model incorrectly predicted "NOT FAILED TO YIELD" when it was actually "FAILED TO YIELD". And we have 28,626 true positives, meaning these are all the cases that were correctly predicted as "FAILED TO YIELD".

Conclusion

The Vehicle Safety Board of Chicago is likely focused on identifying as many "FAILED TO YIELD" caes as possible (high recall) and minimizing incorrect classifications of "FAILED TO YIELD" (high precision), therefore having a nice balance between the two metrics. This would mean our F1-score is the strongest metric our board cares about.

For this reason, our resampled model is the better choice for the Vehicle Safety Board of Chicago. Our Board needs a balance between recall and precision, so going from an F1-score of 0 (from our baseline) to 0.17 is a significant increase. Also, our baseline model is useless for a safety analysis as the recall score was 0. This means that it completely ignores the "FAILED TO YIELD" cases. The board cannot make policy recommendations if the model fails to detect real cases.

Next Steps

The Vehicle Safety Board of Chicago should take these next steps as a result of utilizing our iterated resampled model.

1) Launch targeted public awareness campaigns. The model identified a high number of "FAILED TO YIELD" accidents so it would be important to educate drivers on right-of-way laws at intersections, crossings, and yield signs.

2) Implement more effective traffic control. It would be good to know where "FAILED TO YIELD" accidents happened most frequently. If we understood this, we could improve intersection designs and improve upon traffic signaling.

3) Increase penalties for failure to yield. Our model obviously exposes negative trends when drivers failed to yield so maybe weak enforcement may be causing this. We could think to increase policing in high failure to yield accident centers and increase penalties for failure to yield violations.