Predicting accident severity code using Collision data and ML algorithms

TARUN K. ABICHANDANI

APPLIED DATA SCIENCE CAPSTONE: COURSERA11

Background

- People use navigation platforms to plan trips but no accident warning is issued if the weather or road conditions are bad.
- No warning - driver can be careless and such circumstances can cause accidents

Proposal

- The proposal for the Navigation platforms is to use the past accident data, train a Machine Learning Model that will help in predicting how severe an accident can occur based on weather, road, light conditions.
- Speeding can also be included as an important factor for the analysis
- Once a model is trained, a circular blip can be shown right next to the estimated time showing the severity code and a warning can be issued if the predicted accident severity code is 2.
- This will help in reducing the number of accidents as drivers will be more cautious of the surroundings or might even delay the trip

Current Scenario presented

- Seattle past collision data shared.
- The data is opened and read using jupyter notebook

In [1]:
```
# Importing The Basics
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Path to the local copy of the shared collisions data
csv_path = '~/Desktop/00_GitProjects/Coursera_Capstone/Data-Collisions.csv'
```

In [3]:
```
# Allocating df1 to the dataframe that reads the file
df1 = pd.read_csv(csv_path,low_memory=False)
```

In [4]:
```
# Getting a preview of the data
df1.head(10)
```

Out[4]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On

Data exploration

```
In [5]: #Understanding the shape
        df1.shape

Out[5]: (194673, 38)

In [6]: #Checking the data types
        df1.dtypes

Out[6]: SEVERITYCODE        int64
        X                 float64
        Y                 float64
        OBJECTID            int64
        INCKEY              int64
        COLDETKEY           int64
        REPORTNO           object
        STATUS             object
        ADDRTYPE           object
        INTKEY            float64
        LOCATION           object
        EXCEPTRSNCODE      object
        EXCEPTRSNDESC      object
        SEVERITYCODE.1      int64
        SEVERITYDESC       object
        COLLISIONTYPE      object
        PERSONCOUNT         int64
```

```
In [7]: #01 Exploring various columns that will b
        a = df1['ROADCOND'].isnull().sum(axis=0)
        print('Empty cells:',a)
        b = df1['ROADCOND'].value_counts()
        print(b)
        c = len(df1['ROADCOND'])
        print('total:',c)

Empty cells: 5012
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
total: 194673
```

```
In [8]: #02 Exploring various columns that will b
        d = df1['SPEEDING'].isnull().sum(axis=0)
        e = df1['SPEEDING'].value_counts()
        f = len(df1['SPEEDING'])
        print('total counts:',f)
        print(f'Yes counts:',e)
        print(f'Blank counts:',d)

total counts: 194673
Yes counts: Y    9333
Name: SPEEDING, dtype: int64
Blank counts: 185340
```

```
In [9]: #03 Exploring various columns that will b
        df1['SEVERITYCODE'].value_counts()

Out[9]: 1    136485
        2     58188
        Name: SEVERITYCODE, dtype: int64
```

Data wrangling

- Finalizing the data factors that will be used for the ML model

In [21]:
```
# Finalizng the factors that will be used for the ML Model - feature as well as the results
## Not the final feature dataframe but being used to focus on the data that will be used
df2 = df1[['WEATHER','ROADCOND','LIGHTCOND','SPEEDING','SEVERITYCODE']]
df2.head(15)
```

Out[21]:

	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	SEVERITYCODE
0	Overcast	Wet	Daylight	NaN	2
1	Raining	Wet	Dark - Street Lights On	NaN	1
2	Overcast	Dry	Daylight	NaN	1
3	Clear	Dry	Daylight	NaN	1
4	Raining	Wet	Daylight	NaN	2
5	Clear	Dry	Daylight	NaN	1

Data wrangling – cont'd

- Replacing NaNs, deleting missing values

In [23]:
```
# Replacing NaN from Speeding to 'N'
df2['SPEEDING'].replace(np.nan,'N',inplace=True)
df2.head()
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pa
rsus-a-copy
  self._update_inplace(new_data)
```

Out[23]:

	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	SEVERITYCODE
0	Overcast	Wet	Daylight	N	2
1	Raining	Wet	Dark - Street Lights On	N	1
2	Overcast	Dry	Daylight	N	1
3	Clear	Dry	Daylight	N	1
4	Raining	Wet	Daylight	N	2

In [25]:
```
# Finding missing values in the missing dataset to und
# TRUE - Value missing, FALSE - value present
for column in missing_data.columns.values.tolist():
    print(column)
    print(missing_data[column].value_counts())
    print("")
```

```
WEATHER
False    189592
True       5081
Name: WEATHER, dtype: int64

ROADCOND
False    189661
True       5012
Name: ROADCOND, dtype: int64

LIGHTCOND
False    189503
True       5170
Name: LIGHTCOND, dtype: int64

SPEEDING
False    194673
Name: SPEEDING, dtype: int64

SEVERITYCODE
False    194673
Name: SEVERITYCODE, dtype: int64
```

In [29]:
```
#rechecking to see if any missing values
for column in missing_data_2.columns.values.tolist():
    print(column)
    print(missing_data_2[column].value_counts())
    print("")
```

```
WEATHER
False    189337
Name: WEATHER, dtype: int64

ROADCOND
False    189337
Name: ROADCOND, dtype: int64

LIGHTCOND
False    189337
Name: LIGHTCOND, dtype: int64

SPEEDING
False    189337
Name: SPEEDING, dtype: int64

SEVERITYCODE
False    189337
Name: SEVERITYCODE, dtype: int64
```

Data wrangling – cont'd

- Dummy variables and dropping unnecessary columns

```
In [32]: #DummyVariables: FOR ROADCOND, dropped
         dv1 = pd.get_dummies(df2["ROADCOND"])
         dv1.drop(['Other','Unknown','Standing W
         dv1.head()
```

Out[32]:

	Dry	Ice	Snow/Slush	Wet
0	0	0	0	1
1	0	0	0	1
2	1	0	0	0
3	1	0	0	0
4	0	0	0	1

```
In [33]: #DummyVariables: FOR WEATHER, d
         dv2 = pd.get_dummies(df2["WEATH
         dv2.drop(['Other','Unknown','Bl
         n'],axis=1,inplace=True)
         dv2.head()
```

Out[33]:

	Clear	Overcast	Raining	Snowing
0	0	1	0	0
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0
4	0	0	1	0

```
In [34]: #Replaced Speeding Yes/No to 1/0
         df2['SPEEDING'].replace(to_replace=['Y','N'],value=[1,0],
         df2.head()
```

Out[34]:

	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	SEVERIT
0	Overcast	Wet	Daylight	0	2
1	Raining	Wet	Dark - Street Lights On	0	1
2	Overcast	Dry	Daylight	0	1
3	Clear	Dry	Daylight	0	1
4	Raining	Wet	Daylight	0	2

Data wrangling – cont'd

- Concatenating and selecting random 50000 datasets for the final model

In [50]:
```
#Taking a random sample of 50000 rows instead of all
df4 = df3.sample(n=50000)
df4.shape
```

Out[50]:
```
(50000, 14)
```

In [52]:
```
df4.reset_index(drop=True,inplace=True)
df4.head()
```

Out[52]:

	SPEEDING	SEVERITYCODE	Dry	Ice	Snow/Slush	Wet	Clear	Overcast	Raining	Snowing	Dark - Street Lights On	Dawn	Daylight	Dusk
0	0	1	1	0	0	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0	0	0	1	0	0
2	0	2	1	0	0	0	1	0	0	0	0	0	1	0
3	0	1	1	0	0	0	1	0	0	0	1	0	0	0

Designating X Feature and Y and standardizing the dataset

```
In [54]: X = Feature
         X[0:11]
```

Out[54]:

	SPEEDING	Dry	Ice	Snow/Slush	Wet	Clear	Overcast
0	0	1	0	0	0	1	0
1	0	1	0	0	0	0	1
2	0	1	0	0	0	1	0
3	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0
5	1	1	0	0	0	1	0
6	0	1	0	0	0	1	0
7	0	1	0	0	0	1	0
8	0	1	0	0	0	1	0
9	1	0	1	0	0	1	0
10	0	0	0	0	1	0	1

```
In [55]: y = df4['SEVERITYCODE'].values
         print(y.shape)
         y[0:11]
```

```
(50000,)
```

```
Out[55]: array([1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1])
```

```
In [57]: X = preprocessing.StandardScaler().fit(X).transform(X)
         X[0:11]
```

```
Out[57]: array([[-0.22679564,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
                  0.83806889, -0.41572578, -0.45948118, -0.06617471, -0.58689644,
                 -0.11177736,  0.78967555, -0.17969427],
                [-0.22679564,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
                 -1.19321933,  2.40543178, -0.45948118, -0.06617471, -0.58689644,
                  8.94635552, -1.26634286, -0.17969427],
                [-0.22679564,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
                  0.83806889, -0.41572578, -0.45948118, -0.06617471, -0.58689644,
                 -0.11177736,  0.78967555, -0.17969427],
                [-0.22679564,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
                  0.83806889, -0.41572578, -0.45948118, -0.06617471,  1.70387811,
                 -0.11177736, -1.26634286, -0.17969427],
                [-0.22679564,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
                  0.83806889, -0.41572578, -0.45948118, -0.06617471, -0.58689644,
                 -0.11177736,  0.78967555, -0.17969427],
                [ 4.40925586,  0.720301  , -0.08175808, -0.07017213, -0.57738106,
```

Data Balancing using Imblearn

```
Note: you may need to restart the kernel to use updated packages.
```

In [109]:
```
from sklearn.metrics import recall_score
from imblearn.over_sampling import SMOTE
print("DONE")
```

```
DONE
```

K Nearest Neighbor (KNN)

In [111]:
```
# We split the X into train and test to find the best k
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=12)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (45000, 13) (45000,)
Test set: (5000, 13) (5000,)
```

In [112]:
```
#Oversampling on the training dataset only to make sure none of the information in the val.
#is being used to create synthetic data
sm = SMOTE(random_state=12,ratio = 1.0)
X_train_res, y_train_res = sm.fit_sample(X_train,y_train)
print(X_train_res)
print('----')
print(y_train_res)
```

```
[[-0.22825423  0.7252393  -0.07703906 ... -0.10983533 -1.26201768
  -0.17915986]
 [-0.22825423 -1.37885522 -0.07703906 ... -0.10983533 -1.26201768
```

K Nearest Neighbor (KNN) model

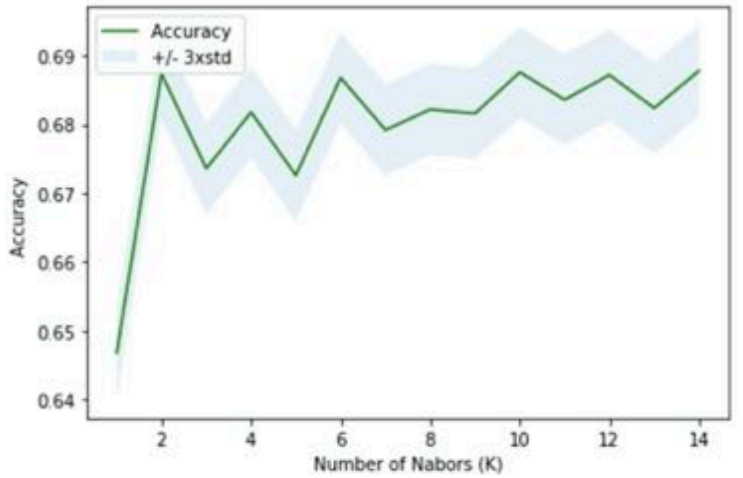
In [113]:
```
# Modeling
from sklearn.neighbors import KNeighborsClassifier
k = 3
#Train Model and Predict
kNN_model = KNeighborsClassifier(n_neighbors=k).fit(X_t
kNN_model
```

Out[113]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
                     metric_params=None, n_jobs=None, n_nei
                     weights='uniform')
```

In [114]:
```
# For Best k
Ks=15
mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))
ConfustionMx=[];
for n in range(1,Ks):

    #Train Model and Predict
    kNN_model = KNeighborsClassifier(n_neighbors=n).fit
    yhat = kNN_model.predict(X_test)


    mean_acc[n-1]=np.mean(yhat==y_test);

    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shap
mean_acc
```

Out[114]:
```
array([0.6468, 0.6874, 0.6736, 0.6818, 0.6726, 0.6868, 0.67
       0.6816, 0.6876, 0.6836, 0.6872, 0.6824, 0.6878])
```

In [115]:
```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mea
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



In [116]:
```
print( "The best accuracy was with", mean_acc.max(), "w
```

```
The best accuracy was with 0.6878 with k= 14
```

Support Vector Machine (SVM) Model

Support Vector Machine (SVM)

In [118]:
```
1 from sklearn import svm
2 SVM_model = svm.SVC()
3 SVM_model.fit(X_train_res, y_train_res)
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/ba
value of gamma will change from 'auto' to 'scale' in version 0.22 to account bet
explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[118]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [120]:
```
1 yhat = SVM_model.predict(X_test)
2 yhat[0:25]
```

Out[120]:
```
array([2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2,
       2, 2, 2])
```

Logistic Regression (LR) model

Logistic Regression (LR)

In [121]:
```
1 from sklearn.linear_model import LogisticRegression
2 LR_model = LogisticRegression(C=0.01).fit(X_train_res,y_train_res)
3 LR_model
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/lin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to sil
  FutureWarning)
```

Out[121]:
```
LogisticRegression(C=0.01, 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='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [122]:
```
1 yhat = LR_model.predict(X_test)
2 yhat[0:25]
```

Out[122]:
```
array([2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2,
       2, 2, 2])
```

Model evaluations

Model Evaluation

In [123]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

In [124]:
```
knn_yhat = kNN_model.predict(X_test)
print("KNN Jaccard index: %.4f" % jaccard_similarity_score(y_test, knn_yhat))
print("KNN F1-score: %.4f" % f1_score(y_test, knn_yhat, average='weighted'))
```

```
KNN Jaccard index: 0.6844
KNN F1-score: 0.5780
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cl
arning: jaccard_similarity_score has been deprecated and replaced with jaccard_score
0.23. This implementation has surprising behavior for binary and multiclass classifi
  'and multiclass classification tasks.', DeprecationWarning)
```

In [125]:
```
SVM_yhat = SVM_model.predict(X_test)
print("SVM Jaccard index: %.4f" % jaccard_similarity_score(y_test, SVM_yhat))
print("SVM F1-score: %.4f" % f1_score(y_test, SVM_yhat, average='weighted'))
```

```
SVM Jaccard index: 0.4710
SVM F1-score: 0.4629
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cl
arning: jaccard_similarity_score has been deprecated and replaced with jaccard_score
0.23. This implementation has surprising behavior for binary and multiclass classifi
  'and multiclass classification tasks.', DeprecationWarning)
```

In [126]:
```
LR_yhat = LR_model.predict(X_test)
LR_yhat_prob = LR_model.predict_proba(X_test)
print("LR Jaccard index: %.4f" % jaccard_similarity_score(y_test, LR_yhat))
print("LR F1-score: %.4f" % f1_score(y_test, LR_yhat, average='weighted') )
```

Results

In [127]:

```
results = {'ALGORITHM':['KNN','SVM','LOGLOSS'],\
          'Jaccard':\
          [jaccard_similarity_score(y_test, knn_yhat),\
           jaccard_similarity_score(y_test, SVM_yhat),\
           jaccard_similarity_score(y_test, LR_yhat)],\
         'F-1 Score':\
          [f1_score(y_test, knn_yhat, average='weighted'),\
           f1_score(y_test, SVM_yhat, average='weighted'),\
           f1_score(y_test, LR_yhat, average='weighted')],\
         'LogLoss':['NA','NA',log_loss(y_test, LR_yhat_prob)]}
df5 = pd.DataFrame(results)
df5.head()
```

```
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cl
arning: jaccard_similarity_score has been deprecated and replaced with jaccard_score
0.23. This implementation has surprising behavior for binary and multiclass classifi
  'and multiclass classification tasks.', DeprecationWarning)
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cl
arning: jaccard_similarity_score has been deprecated and replaced with jaccard_score
0.23. This implementation has surprising behavior for binary and multiclass classifi
  'and multiclass classification tasks.', DeprecationWarning)
/Users/tarunabichandani/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cl
arning: jaccard_similarity_score has been deprecated and replaced with jaccard_score
0.23. This implementation has surprising behavior for binary and multiclass classifi
  'and multiclass classification tasks.', DeprecationWarning)
```

Out[127]:

	ALGORITHM	Jaccard	F-1 Score	LogLoss
0	KNN	0.6844	0.578031	NA
1	SVM	0.4710	0.462905	NA
2	LOGLOSS	0.4676	0.456333	0.66559

Screenshot

Discussions

- The KNN model predicts ~70% of the times correctly about the Severity Code as per the Jaccard index and the room for improvement is as follows:
  - Maybe use all data instead of 50000 random samples
  - Use more columns and more predictors to train the model efficiently

Conclusion

- Though there is room for improvement, the current trained models can be used to predict the severity of the accident based on weather, road, and light conditions.
- This can be a good starting point for Navigation companies to start training ML models and make better predictions over time by adding more predictors
- The ML model can definitely help the drivers drive more carefully once the accident severity code is issued based on environmental conditions


JUST
THANK
YOU!