Capstone Project Report

Seattle Car Accident Severity

Submited by,
Akshay Kumar
Oct/2020

Introduction | Business Understanding

The Seattle government is going to prevent avoidable car accidents by employing methods that alert drivers, health system, and police to remind them to be more careful in critical situations.

In most cases, not paying enough attention during driving, abusing drugs and alcohol or driving at very high speed are the main causes of occurring accidents that can be prevented by enacting harsher regulations. Besides the aforementioned reasons, weather, visibility, or road conditions are the major uncontrollable factors that can be prevented by revealing hidden patterns in the data and announcing warning to the local government, police and drivers on the targeted roads.

The target audience of the project is local Seattle government, police, rescue groups, and last but not least, car insurance institutes. The model and its results are going to provide some advice for the target audience to make insightful decisions for reducing the number of accidents and injuries for the city.

Data Understanding

The data was collected by the Seattle Police Department and Accident Traffic Records Department from 2004 to present.

The data consists of 37 independent variables and 194,673 rows. The dependent variable, “SEVERITYCODE”, contains numbers that correspond to different levels of severity caused by an accident from 0 to 4.
Severity codes are as follows:
0: Little to no Probability (Clear Conditions)
1: Very Low Probability — Chance or Property Damage
2: Low Probability — Chance of Injury
3: Mild Probability — Chance of Serious Injury
4: High Probability — Chance of Fatality

Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.

Data Preprocessing

In its original form, this data is not fit for analysis. For one, there are many columns that we will not use for this mode so we need to drop the non-relevant columns. Also, most of the features are of type object, when they should be numerical type, so I used label encoding to covert the features to our desired data type.
After that our Data will look like this.

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	WEATHER_ENC	ROADCOND_ENC	LIGHTCOND_ENC
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

With the new columns, we can now use this data in our analysis and ML models!

To get a good understanding of the dataset, I have checked different values in the features. The results show, the target feature is imbalance, so we use a simple statistical technique to balance it.

```
df2['SEVERITYCODE'].value_counts()
```

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

As you can see, the number of rows in class 1 is almost three times bigger than the number of rows in class 2. It is possible to solve the issue by down sampling the class 1.

```
from sklearn.utils import resample
```

```
df2_major=df2[df2.SEVERITYCODE==1]
df2_minor=df2[df2.SEVERITYCODE==2]
df2_major_dsample=resample(df2_major,
                           replace= False,
                           n_samples=58188,
                           random_state=123)
balanced_df2=pd.concat([df2_major_dsample,df2_minor])
balanced_df2['SEVERITYCODE'].value_counts()
```

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

Now our data is 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. 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.

Initialization

Define X and y

```
#Defining array X
X=np.asarray(df2[['WEATHER_ENC','ROADCOND_ENC','LIGHTCOND_ENC']])
X[0:5]
```

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

```
# Defining array y
y=np.asarray(df2['SEVERITYCODE'])
y[0:5]
```

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

Normalize the dataset:

Now we will normalize the data for using to train and test our models.

```
# Normalizing the data
from sklearn import preprocessing
X=preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[ 0.22981187,  1.39847224,  0.25900713],
       [ 0.87758556,  1.39847224, -1.36653782],
       [ 0.22981187, -0.73846749,  0.25900713],
       [-0.74184867, -0.73846749,  0.25900713],
       [ 0.87758556,  1.39847224,  0.25900713]])
```

Train/Test Split:

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

```
# Train/Test split of data
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=42)
print('Train set:',X_train.shape,y_train.shape)
print('Test set:',X_test.shape,y_test.shape)
```

```
Train set: (136271, 3) (136271,)
Test set: (58402, 3) (58402,)
```

Now we will build and test our models.

K-Nearest Neighbors:

We will not build, train and test our model as shown below.

```
# Building the KNN Model
from sklearn.neighbors import KNeighborsClassifier
k=25
```

```
# Train Model and predict
neigh=KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
neigh
Kyhat=neigh.predict(X_test)
Kyhat[0:5]
```

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

```
# Jaccard similarity score
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, Kyhat)
```

```
C:\Users\Akshay.DESKTOP-E4NEDBT\anaconda3\lib\site-pack
imilarity_score has been deprecated and replaced with j
has surprising behavior for binary and multiclass class
  FutureWarning)
```

```
7]: 0.7048217526797027
```

```
# F1- score
from sklearn.metrics import f1_score
f1_score(y_test, Kyhat, average='weighted')
```

```
8]: 0.582846780130164
```

Model is most accurate when K=25

Decision Tree:

We will not build, train and test our model as shown below.

```
# Building the Decision tree
from sklearn.tree import DecisionTreeClassifier
colDataTree = DecisionTreeClassifier(criterion="entropy",max_depth = 7)
colDataTree
colDataTree.fit(X_train,y_train)
```

```
: DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                         random_state=None, splitter='best')
```

```
# Train model and predict
DTyhat = colDataTree.predict(X_test)
print(DTyhat [0:5])
print(y_test [0:5])
```

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

```
# Jaccard similarity score
jaccard_similarity_score(y_test,DTyhat)
```

```
C:\Users\Akshay.DESKTOP-E4NEDBT\anaconda3\
imilarity_score has been deprecated and re
has surprising behavior for binary and mul
  FutureWarning)
```

22]: 0.7047532618745933

```
# F1- score
f1_score(y_test,DTyhat,average='macro')
```

23]: 0.41357355147211866

Model is most accurate with max depth of 7

Logistic Regression:

We will not build, train and test our model as shown below.

```
# Building logistic Regression 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
```

```
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, verbose=0,
                   warm_start=False)
```

```
# Predict result
LRyhat = LR.predict(X_test)
LRyhat
```

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

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

```
array([[0.74712365, 0.25287635],
       [0.83670319, 0.16329681],
       [0.67181946, 0.32818054],
       ...,
       [0.67181946, 0.32818054],
       [0.70037211, 0.29962789],
       [0.75391483, 0.24608517]])
```

```
#Jaccard similarity score
jaccard_similarity_score(y_test,LRyhat)
```

```
C:\Users\Akshay.DESKTOP-E4NEDBT\anaconda3\
imilarity_score has been deprecated and re
has surprising behavior for binary and mul
  FutureWarning)
```

```
]: 0.7048559980822574
```

```
#F1-score
f1_score(y_test,LRyhat,average='macro')
```

```
]: 0.41344019604889165
```

```
# LOGLOSS
yhat_prob = LR.predict_proba(X_test)
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

```
]: 0.5989587236357746
```

Results and Evaluations

The final results of the model evaluations are summarized in the following table:

ML Model	Jaccard Score	F1 Score
KNN	0.7047	0.4135
Decision Tree	0.69	0.41
Logistic Regression	0.69	0.41

Based on the above table, KNN is the best model to predict car accident severity.

Conclusion

Based on the dataset provided for this capstone from weather, road, and light conditions pointing to certain classes, we can conclude that particular conditions have a somewhat impact on whether or not travel could result in property damage (class 1) or injury (class 2).