Data_Science_Capstone

October 13, 2020

1 Data Science Capstone Project

1.1 Vehicle Collision Prediction

1.2 Table of Contents

1.3 Introduction/Business Problem

Vehicular accidents are common on roads across the world. The type of accident varies in severity. They can simply range from property damage, such as minor fender-benders, to loss of life with one or more parties. It is an unfortunate commonplace. What if it was possible to predict the severity of an accident occurring given current conditions? Drivers across the world would benefit from this information. Decisions could be made about whether it was worth the risk of getting on the road or postponing the trip for a later time when conditions improve.

1.4 Data

A dataset from the Seattle Department of Transportation (SDOT) will be used to create and train multiple models, which will be evaluated for a comparison of each model's accuracy. The SDOT data set includes entries for nearly 195,000 accidents from 2004 to the present. The severity of each accident is categorized with multiple features to choose from for modeling. A few examples of the features are as follows:

- Collision
- Address Type (Alley, Block, Intersection)
- Location
- Collision Type
- Number of people involved in the collision
- Number of pedestrians involved in the collision

- Number of cyclists involved in the collision
- Number of vehicles involved in the collision
- Number of fatalities
- Weather conditions
- Road conditions
- Lighting Conditions
- And more

The primary focus of this investigation are the environmental driving conditions at the time of the collision. Therefore, the following features will be investigated:

- Weather conditions (WEATHER)
- Roadconditions (ROADCOND)
- Light conditions (LIGHTCOND)

1.5 Methodology

Load the required libraries.

[1]:
```
# Load all required libraries
import pandas as pd
import numpy as np
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 import svm
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
```

1.5.1 Load the Dataset and Create a Clean Dataframe

[2]:
```
# Load the desired columns from a csv into a dataframe
import pandas as pd
df = pd.read_csv('Data-Collisions.csv', usecols = ['WEATHER', 'ROADCOND',␣
↪'LIGHTCOND', 'SEVERITYCODE'])
df.head()
```

[2]:
```
   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
```

The initial dataframe is now created. As a matter of personal preference, the label, or SEVERITYCODE column, will be moved to the right end of the dataframe.

[3]:
```
# Reivse the column order of the dataframe
df = df[['WEATHER', 'ROADCOND', 'LIGHTCOND', 'SEVERITYCODE']]
df.head()
```

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

Now that the dataframe is in the desired layout, it is a good idea to check the column counts to see if there are any missing values.

[4]:
```
# Check the column counts to check for NAN or missing values
df.count()
```

[4]:
```
WEATHER         189592
ROADCOND        189661
LIGHTCOND       189503
SEVERITYCODE    194673
dtype: int64
```

It looks like the there are quite a few more SEVERITYCODE values the features. To fix this, all rows with missing values should be removed.

[5]:
```
# Drop rows with NAN or missing values
clean_df = df.dropna(axis = 0)
clean_df.count()
```

[5]:
```
WEATHER         189337
ROADCOND        189337
LIGHTCOND       189337
SEVERITYCODE    189337
dtype: int64
```

Great, the dataframe now has the same number of values for each column. The next item is to review the value counts for each column to evaluate any changes that may be needed to the data.

[6]:
```
clean_df['WEATHER'].value_counts(sort=True)
```

[6]:
```
Clear                       111008
Raining                      33117
Overcast                     27681
Unknown                      15039
```

```
Snowing                        901
Other                          824
Fog/Smog/Smoke                 569
Sleet/Hail/Freezing Rain       113
Blowing Sand/Dirt               55
Severe Crosswind                25
Partly Cloudy                    5
Name: WEATHER, dtype: int64
```

[7]:
```
clean_df['ROADCOND'].value_counts(sort=True)
```

[7]:
```
Dry                 124300
Wet                  47417
Unknown              15031
Ice                   1206
Snow/Slush             999
Other                  131
Standing Water         115
Sand/Mud/Dirt           74
Oil                     64
Name: ROADCOND, dtype: int64
```

[8]:
```
clean_df['LIGHTCOND'].value_counts(sort=True)
```

[8]:
```
Daylight                    116077
Dark - Street Lights On      48440
Unknown                      13456
Dusk                          5889
Dawn                          2502
Dark - No Street Lights       1535
Dark - Street Lights Off      1192
Other                          235
Dark - Unknown Lighting         11
Name: LIGHTCOND, dtype: int64
```

[9]:
```
clean_df['SEVERITYCODE'].value_counts(sort=True)
```

[9]:
```
1    132285
2     57052
Name: SEVERITYCODE, dtype: int64
```

Each of the 3 features has a quantity of Unknown and Other values. These will not help the analysis and need to be removed. No changes are required for the column containing the label.

[10]:
```
# Drop Unknown & Other values from WEATHER, ROADCOND, LIGHTCOND
index_values_to_drop = clean_df[((clean_df['WEATHER'] == 'Unknown') |
 ↪(clean_df['WEATHER'] == 'Other'))
```

```
                                  | ((clean_df['ROADCOND'] == 'Unknown') |␣
↪(clean_df['ROADCOND'] == 'Other'))
                                  | ((clean_df['LIGHTCOND'] == 'Unknown') |␣
↪(clean_df['LIGHTCOND'] == 'Other'))].index
clean_df = clean_df.drop(index_values_to_drop) #used this notation to prevent␣
↪warnings from setting inplace = True
clean_df.count()
```

[10]:
```
WEATHER         169957
ROADCOND        169957
LIGHTCOND       169957
SEVERITYCODE    169957
dtype: int64
```

The column counts are equal and approximately 19,000 rows were removed. It is appropriate the review the values counts of each feature one more time. The label doesn't require any reanalysis for the reason noted above.

[11]:
```
clean_df['WEATHER'].value_counts(sort=True)
```

[11]:
```
Clear                       108825
Raining                      32648
Overcast                     26923
Snowing                        825
Fog/Smog/Smoke                 553
Sleet/Hail/Freezing Rain       107
Blowing Sand/Dirt               46
Severe Crosswind                25
Partly Cloudy                    5
Name: WEATHER, dtype: int64
```

[12]:
```
clean_df['ROADCOND'].value_counts(sort=True)
```

[12]:
```
Dry               121490
Wet                46324
Ice                 1080
Snow/Slush           833
Standing Water       105
Sand/Mud/Dirt         65
Oil                   60
Name: ROADCOND, dtype: int64
```

[13]:
```
clean_df['LIGHTCOND'].value_counts(sort=True)
```

[13]:
```
Daylight                    112618
Dark - Street Lights On      46748
Dusk                          5648
Dawn                          2413
```

```
Dark - No Street Lights       1408
Dark - Street Lights Off      1114
Dark - Unknown Lighting          8
Name: LIGHTCOND, dtype: int64
```

The WEATHER AND ROADCOND features have values that are different enough when considering envrionmental factors. The LIGHCOND feature has 4 types of "Dark." As this analysis is for environmental factors only, all 4 variants of "Dark" will be replaced with a "Dark" value.

[14]:
```
clean_df['LIGHTCOND'].replace(['Dark - Street Lights On','Dark - No Street␣
↪Lights', 'Dark - Street Lights Off', 'Dark - Unknown Lighting'],'Dark',␣
↪inplace = True)
clean_df['LIGHTCOND'].value_counts(sort=True)
```

[14]:
```
Daylight    112618
Dark         49278
Dusk          5648
Dawn          2413
Name: LIGHTCOND, dtype: int64
```

The dataframe is nearly ready for analysis.

[15]:
```
clean_df.head()
```

[15]:

	WEATHER	ROADCOND	LIGHTCOND	SEVERITYCODE
0	Overcast	Wet	Daylight	2
1	Raining	Wet	Dark	1
2	Overcast	Dry	Daylight	1
3	Clear	Dry	Daylight	1
4	Raining	Wet	Daylight	2

[16]:
```
clean_df.dtypes
```

[16]:
```
WEATHER         object
ROADCOND        object
LIGHTCOND       object
SEVERITYCODE     int64
dtype: object
```

The values for each features need to be converted to a numerical value for analysis.

[17]:
```
# create numerical and string lists for each feature
weather_numeric = [*range(0, 9, 1)]
roadcond_numeric = [*range(0, 7, 1)]
lightcond_numeric = [*range(0, 4, 1)]

weather_list = ['Clear', 'Raining', 'Overcast', 'Snowing', 'Fog/Smog/Smoke',␣
↪'Sleet/Hail/Freezing Rain', 'Blowing Sand/Dirt', 'Severe Crosswind', 'Partly␣
↪Cloudy']
```

```
roadcond_list = ['Dry', 'Wet', 'Ice', 'Snow/Slush', 'Standing Water', 'Sand/Mud/
 ↪Dirt', 'Oil']
lightcond_list = ['Daylight', 'Dark', 'Dusk', 'Dawn']

# create final dataframe and change the string values to integers
final_df = clean_df.copy()
final_df['WEATHER'].replace(to_replace = weather_list, value = weather_numeric,␣
 ↪inplace = True)
final_df['ROADCOND'].replace(to_replace = roadcond_list, value =␣
 ↪roadcond_numeric, inplace = True)
final_df['LIGHTCOND'].replace(to_replace = lightcond_list, value =␣
 ↪lightcond_numeric, inplace = True)

final_df.head()
```

[17]:

	WEATHER	ROADCOND	LIGHTCOND	SEVERITYCODE
0	2	1	0	2
1	1	1	1	1
2	2	0	0	1
3	0	0	0	1
4	1	1	0	2

[18]:

```
final_df.dtypes
```

[18]:

```
WEATHER         int64
ROADCOND        int64
LIGHTCOND       int64
SEVERITYCODE    int64
dtype: object
```

All columns in the dataframe (both features and label) are now integers. The dataframe is ready for modeling.

1.6 Modeling

As the data set has labels, supervised machine models are the choice with classifiers being the main choice. K-Nearest Neighbors (KNN), Decision Tree, Logistic Regression, and Support Vector Machine (SVM) models will all be evaluated and compared to find the most accurate model.

Before starting the modeling, the feature and label sets need to be created from the dataframe. To start, the feature set X will be defined.

[19]:

```
X = final_df[['WEATHER', 'ROADCOND', 'LIGHTCOND']]
X[0:5]
```

[19]:

	WEATHER	ROADCOND	LIGHTCOND
0	2	1	0
1	1	1	1

```
2         2         0         0
3         0         0         0
4         1         1         0
```

Now, the label set, y, will be created.

[20]:
```
y = final_df['SEVERITYCODE'].values
y[0:5]
```

[20]:
```
array([2, 1, 1, 1, 2], dtype=int64)
```

Split the data into training and testing data sets using 20% for the test data.

[21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 4)
```

Normalize the feature train and test data.

[22]:
```
X_train = preprocessing.StandardScaler().fit(X_train).transform(X_train)
X_train[0:5]
```

[22]:
```
array([[-0.66539872, -0.57845596, -0.63726064],
       [-0.66539872, -0.57845596, -0.63726064],
       [-0.66539872, -0.57845596, -0.63726064],
       [-0.66539872, -0.57845596,  2.55426532],
       [-0.66539872, -0.57845596,  0.95850234]])
```

[23]:
```
X_test = preprocessing.StandardScaler().fit(X_test).transform(X_test)
X_test[0:5]
```

[23]:
```
array([[-0.67312293, -0.58065715,  0.96901319],
       [-0.67312293, -0.58065715, -0.63963578],
       [ 1.77506825, -0.58065715,  0.96901319],
       [-0.67312293,  1.26882728, -0.63963578],
       [-0.67312293, -0.58065715, -0.63963578]])
```

1.6.1 KNN Model

Determine the value of k that provides the highest level of accuracy for a KNN model.

[24]:
```
# find the best K
K_max = 10

mean_acc = np.zeros((K_max - 1))

for i in range(1, K_max):
    knn = KNeighborsClassifier(n_neighbors = i).fit(X_train, y_train)
    yhat = knn.predict(X_test)
    mean_acc[i - 1] = metrics.accuracy_score(y_test, yhat)
```

```
print('k = ', mean_acc.argmax() + 1, 'provided the highest accuarcy of ',␣
↪mean_acc.max())
```

```
k =  4 provided the highest accuarcy of  0.670834313956225
```

Create the final KNN model with the best value of k and fit the model using the training dataset.

```
[25]: knn_final = KNeighborsClassifier(n_neighbors = (mean_acc.argmax() + 1))
      knn_final.fit(X_train, y_train)
      knn_final
```

```
[25]: KNeighborsClassifier(n_neighbors=4)
```

1.6.2 Decision Tree Model

Create the a Decision Tree model with the entropy criteria for information gain and max_depth value equal to 4. Fit the model using the training dataset.

```
[26]: dt_final = DecisionTreeClassifier(criterion = 'entropy', max_depth = 4)
      dt_final.fit(X_train, y_train)
      dt_final
```

```
[26]: DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

1.6.3 Logistic Regression

Create the a Logistic Regression model with the default lbfgs solver and fit the model using the training data.

```
[27]: lr_final = LogisticRegression(C = 0.01)
      lr_final.fit(X_train, y_train)
      lr_final
```

```
[27]: LogisticRegression(C=0.01)
```

1.6.4 Support Vector Machine

Create the a Support Vector Machine model with the default rbf kernel using the training data.

```
[28]: svm_final = svm.SVC()
      svm_final.fit(X_train, y_train)
      svm_final
```

```
[28]: SVC()
```

Now that the models have been created and fitted to the training data, it's time to determine the accuarcy and F1 scores of each model.

1.7 Results

Determine the accuracy and F1 scores for each model usint the testing data.

[29]:
```
# Calculate the Jaccard(accuarcy) and F1 scores for each model
yhat_knn = knn_final.predict(X_test)
yhat_dt = dt_final.predict(X_test)
yhat_lr = lr_final.predict(X_test)
yhat_svm = svm_final.predict(X_test)

jaccard_values = [jaccard_score(y_test, yhat_knn),
                  jaccard_score(y_test, yhat_dt),
                  jaccard_score(y_test, yhat_lr),
                  jaccard_score(y_test, yhat_svm)]

f1_values = [f1_score(y_test, yhat_knn),
             f1_score(y_test, yhat_dt),
             f1_score(y_test, yhat_lr),
             f1_score(y_test, yhat_svm)]

# Add jaccard and F1 scores to a dataframe
data = {'Accuracy' : pd.Series(jaccard_values, index = ['KNN', 'Decision Tree',␣
 ↪'Logistic Regression', 'SVM']),
        'F1-Score' : pd.Series(f1_values, index = ['KNN', 'Decision Tree',␣
 ↪'Logistic Regression', 'SVM'])}

final_values = pd.DataFrame(data)

final_values
```

[29]:

	Accuracy	F1-Score
KNN	0.670243	0.802569
Decision Tree	0.673305	0.804761
Logistic Regression	0.673305	0.804761
SVM	0.673247	0.804719

The Decsion Tree and Logistic Regression models are tied in both the highest accuarcy and F1 scores. Therefore, either model may be selected as the "best" of the created models.

Unfortnately, none of the four models created is that accurate in predicting the severity of a collosion. This brings into question quality of the models as to be discussed in the next section.

1.8 Discussion

As noted above, none of the models are accurate. The root cause is believed to be the lack of features for each model to consider from the selected data set. More factors, such as time of day, investigating if certain causes (inattention or DUI) cluster in certain areas, etc., may create a more accurate model that would benefit a driver deciding whether to leave the house. The goal of creating a model solely based on environmental factors of the weather, road conditions, and lighting conditions does not yield good results. Until completion of further investigation in future models, it is recommended that these models generated in this report are not used for any reason until the accuracy greatly improves.

1.9 Conclusion

Overall, the results of this investigation are disappointing. The author hoped it was possible to create a model for determining the severity of an accident based on only environment factors (weather, road conditions, and lighting conditions). The models created are very inaccurate and the primary culprit is thought to be the lack of variance in the selected features from the data set. The model is too simple and requires more information to create accurate predictions. It is the recommendation of the author to not use the created models and learn from the results. Future models with more features may yield better results.