IBM Data Science Capstone Project: Predict CAR Accident Severity

1. Introduction

Background

Imagine, you got a situation and you need to travel. However, you see the weather is wild. Now, is it better to drive?

You are working on a council office and you want to alert the people about the bad weather/bad road condition/bad lighting and provide any advice on Travel.

So, if there is any way we predict about the possible accidents and its severity based on the previous history of accidents and factors related to it. Yes, there is. That is what we are going to try in this project.

Using the accident history data, which has the details about weather, road, light conditions, severity, we are going to try is there any way we can predict accident severity and possibility.

Scope

This project's objective is to predict Severity of accident based on data that help severity of accident include Address type, Weather condition, Road condition, Junction type and Light condition.

Target Audience

The project potentially will help people, local government, Police and emergency departments, Engineers who design the roads. The project and result will provide some insights to audience and help to make decisions in reducing the number of accidents in their area.

2. Data

For this project, the data provided by the Seattle Department of Transportation (SDOT) is used and the same can be downloaded from here from here. The data has the key information that include Severity of the accident, Incident Date/Time, number of vehicles and persons involved in accidents, Road type and conditions, Weather and light conditions.

This dataset has details about 194k accident details. Each accident is defined by 38 different attributes (features). The metadata of the features are given here.

As part of the project we are going to predict the **Severity of the accident** (feature name: **SEVERITYCODE**).

So, the predictor is going to be **SEVERITYCODE.** The values of the predictor will range between 0 to 3. The definition is given below

· 0: Unknown/no data

· 1: Property damage only

· 2: Minor injury collision

· 2b: Major injury collision

· 3: Fatality collision

Based on the project objective, we would predominantly use the below features to predict the severity (initial analysis). The selected features can help to define the predictor than the dropped ones. We will furtherly do the Exploratory analysis and we may change the feature selection.

The columns in Deleted features category either do not have a clear data for all the accidents or they don't play any significance in predicting Severity.

Selected Features

'Severityvode','Addrtype', Junctiontype','Weather', 'Roadcond','Lightcond' ,

Deleted Features

'X', 'Y', 'Coldetkey', 'Reportno', 'Intkey', 'Location', 'Exceptrsncode', 'Exceptrsndesc', 'Severitycode.1', 'Severitydesc', 'Incdate','Incdttm', ', 'Sdot_Colcode', 'Sdot_Coldesc', 'Inattentionind', 'Underinfl', 'Pedrownotgrnt', Sdotcolnum', 'Speeding', 'St_Colcode', 'St_Coldesc','Seglanekey', 'Crosswalkkey', 'Hitparkedcar', 'Personcount', 'Pedcount', 'Pedcylcount', 'Vehcount', 'Objectid', 'Collisiontype','Status'

3. Data Cleaning:

Rows which are missing information about some of the features which we expect will be key to building the model: as we there are , a number of accidents have "Unknown" values for attributes like *WEATHER* , *ROADCOND* and *LIGHTCOND* , or have these fields blank/NaN, which in practice means the same thing. As these are expected to be among the features which influence the likelihood and severity of accidents, we must consider discarding these rows before training the model.

```
##############  CHECK FOR NULL VALUES IN KEY COLUMNS######
df_sev.isnull().sum()
```

```
SEVERITYCODE       0
ADDRTYPE        1926
JUNCTIONTYPE    6329
WEATHER         5081
ROADCOND        5012
LIGHTCOND       5170
dtype: int64
```

Check for Null values in each feature

```
##############  Replace NULL Values with "Others"  ######
df_sev['ADDRTYPE'] = df_sev['ADDRTYPE'].fillna(0)
df_sev['ADDRTYPE'] = df_sev['ADDRTYPE'].replace(0,'Other')

df_sev['JUNCTIONTYPE'] = df_sev['JUNCTIONTYPE'].fillna(0)
df_sev['JUNCTIONTYPE'] = df_sev['JUNCTIONTYPE'].replace(0,'Other')

df_sev['WEATHER'] = df_sev['WEATHER'].fillna(0)
df_sev['WEATHER'] = df_sev['WEATHER'].replace(0,'Unspecified')

df_sev['ROADCOND'] = df_sev['ROADCOND'].fillna(0)
df_sev['ROADCOND'] = df_sev['ROADCOND'].replace(0,'Unspecified')

df_sev['LIGHTCOND'] = df_sev['LIGHTCOND'].fillna(0)
df_sev['LIGHTCOND'] = df_sev['LIGHTCOND'].replace(0,'Unspecified')


df_sev.head(5)
```

Replacing Null Values

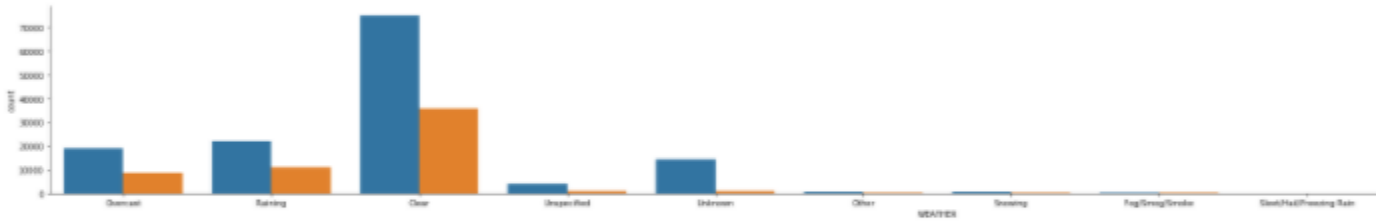
4. Exploratory Data Analysis

To Understand the relationship between the selected features have done the exploratory data analysis

Relationship between WEATHER & SEVERITY of accidents

```
import seaborn as sns
sns.catplot(x='WEATHER', kind='count',hue='SEVERITYCODE',data=df_sev,height=4,aspect=8)
```

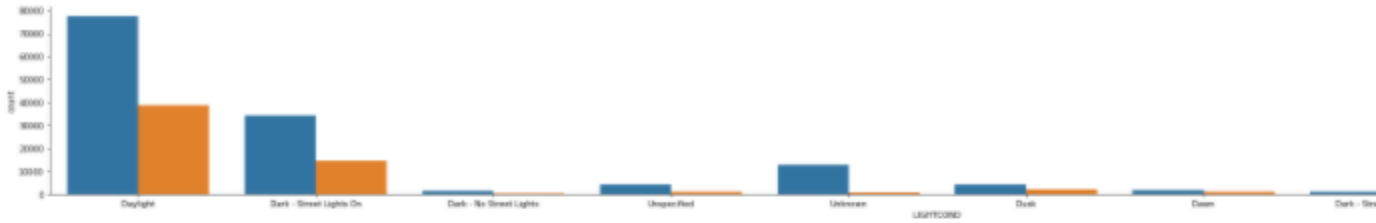
```
8]: <seaborn.axisgrid.FacetGrid at 0x200c9a521c8>
```



Relationship between LIGHT Condition & SEVERITY of accidents

```
import seaborn as sns
sns.catplot(x='LIGHTCOND', kind='count',hue='SEVERITYCODE',data=df_sev,height=4,aspect=8)
```

```
9]: <seaborn.axisgrid.FacetGrid at 0x200c93d0f48>
```



```
df_sev['WEATHER'].value_counts()
```

```
3]: 1     111135
    6      33145
    4      27714
    10     15091
    11      5081
    9        907
    3        832
    2        569
    8        113
    0         56
    7         25
    5          5
    Name: WEATHER, dtype: int64
```

Exploratory Data Analysis to Understand the relationship between the features

4.1 Encoding Feature Values

Encoding Feature values

```
## Encoding the features into Numarical values###

from sklearn import preprocessing


severity = preprocessing.LabelEncoder()
severity.fit(df_sev['SEVERITYCODE'])
df_sev['SEVERITYCODE'] = severity.transform(df_sev['SEVERITYCODE'])


addrtype = preprocessing.LabelEncoder()
addrtype.fit(df_sev['ADDRTYPE'])
df_sev['ADDRTYPE'] = addrtype.transform(df_sev['ADDRTYPE'])

junctype = preprocessing.LabelEncoder()
junctype.fit(df_sev['JUNCTIONTYPE'])
df_sev['JUNCTIONTYPE'] = junctype.transform(df_sev['JUNCTIONTYPE'])

weather = preprocessing.LabelEncoder()
weather.fit(df_sev['WEATHER'])
df_sev['WEATHER'] = weather.transform(df_sev['WEATHER'])

roadcond = preprocessing.LabelEncoder()
roadcond.fit(df_sev['ROADCOND'])
df_sev['ROADCOND'] = roadcond.transform(df_sev['ROADCOND'])

lightcond = preprocessing.LabelEncoder()
lightcond.fit(df_sev['LIGHTCOND'])
df_sev['LIGHTCOND'] = lightcond.transform(df_sev['LIGHTCOND'])


df_sev.head(5)
```

4.2 Understand the Correlation

Understand the Correlation

```
df_sev.corr()
```

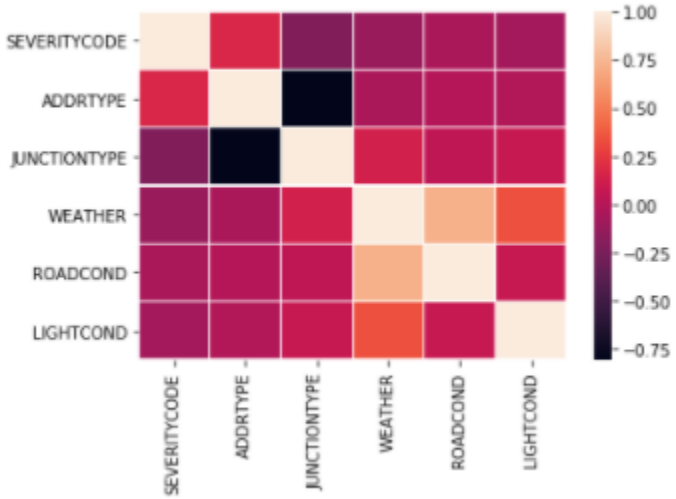
14]:

	SEVERITYCODE	ADDRTYPE	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND
SEVERITYCODE	1.000000	0.172032	-0.216118	-0.112507	-0.045574	-0.067817
ADDRTYPE	0.172032	1.000000	-0.804215	-0.038823	-0.002626	-0.013375
JUNCTIONTYPE	-0.216118	-0.804215	1.000000	0.134021	0.048077	0.077417
WEATHER	-0.112507	-0.038823	0.134021	1.000000	0.728901	0.339707
ROADCOND	-0.045574	-0.002626	0.048077	0.728901	1.000000	0.083132
LIGHTCOND	-0.067817	-0.013375	0.077417	0.339707	0.083132	1.000000

```
import seaborn as sns

sns.heatmap(df_sev.corr(), linewidth=.2, cbar_kws={"shrink": 1})
```

15]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c4b8b65508>`



Correlation Matrix

5. Modeling

As a beginning of the modelling exercise, lets split the data into Train-Test data splits.

Test Train Data Split ¶

```
### Split the source data into TRAIN - TEST data sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (155738, 49) (155738,)
Test set: (38935, 49) (38935,)
```

For this exercise, we are going to predict the Accident Severity category based on the available features. We have the labeled data to predict and we chose classification algorithms for the same. In this exercise I am going to build models using below algorithms and evaluate the efficiency of the models.

1. KNN—K—Nearest Neighbors

2. Decision Tree

3. Logistic regressions

5.1 KNN—K—Nearest Neighbors

To build KNN algorithm, I have taken it for set of 30 values the model will be trained on training set of data, and then predicting the values based on test data set.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics


Ks = 30
mean_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train_part_train, np.ravel(y_train_part_train))
    yhat = neigh.predict(X_train_part_test)
    mean_acc[n-1] = metrics.accuracy_score(y_train_part_test, yhat)


print("The mean accuracy array is:", mean_acc)
print("\nThe maximum mean accuracy value is:", mean_acc.max())

k = list(mean_acc).index(mean_acc.max()) + 1
print("\nThe best model is the model with k-value = ", k)
```

```
The mean accuracy array is: [0.65  0.675 0.665 0.695 0.685 0.685 0.675 0.68  0.65  0.66  0.665 0.66
 0.66  0.68  0.67  0.68  0.685 0.685 0.7   0.69  0.68  0.69  0.695 0.695
 0.69  0.69  0.695 0.695 0.69 ]

The maximum mean accuracy value is: 0.7

The best model is the model with k-value =  19
```

```
KNN = KNeighborsClassifier(n_neighbors = k).fit(X_train, y_train) ## This takes about 2.5 minutes on an Intel i5-930
KNN
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                     weights='uniform')
```

K Nearest Neighbors Modelling

As per the above model the most accuracy we have for when K=19 which is 0.7. So I trained my model with the K-Value =19.

5.2 Decision Tree

Modeling - Decision Tree

```
X_train_train, X_train_test, y_train_train, y_train_test = \
                    train_test_split( X_train, y_train, test_size=0.2, random_state=0)

print ('Part Train set:', X_train_train.shape,  y_train_train.shape)
print ('Part Test set:', X_train_test.shape,  y_train_test.shape)
```

```
Part Train set: (124590, 49) (124590,)
Part Test set: (31148, 49) (31148,)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

depth = 21
mean_acc1 = np.zeros((depth - 4))

for n in range(4, depth):

    tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = depth).fit(X_train_train, y_train_train)
    yhat = tree.predict(X_train_test)
    mean_acc1[n-4] = metrics.accuracy_score(y_train_test, yhat)


print("The mean accuracy array is:", mean_acc1)
print("\nThe maximum mean accuracy value is:", mean_acc1.max())

max_depth = list(mean_acc1).index(mean_acc1.max()) + 4
print("\nThe best model is the model with max_depth = ", max_depth)
```

```
The mean accuracy array is: [0.6984076  0.69856813 0.69850392 0.69850392 0.69847181 0.69847181
 0.69850392 0.69847181 0.69843971 0.6984076  0.69843971 0.69850392
 0.69850392 0.69847181 0.69856813 0.69843971 0.69850392]

The maximum mean accuracy value is: 0.6985681263644535

The best model is the model with max_depth =  5
```

```
severityTree = DecisionTreeClassifier(criterion = 'entropy', max_depth = max_depth).fit(X_train, y_train)
severityTree
```

```
1]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                           max_depth=5, 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')
```

5.3 Logistic Regression

Modeling - Logistic Regression

```
X_train_part = pd.DataFrame(X_train).sample(n = 1000, random_state = 0)
y_train_part = pd.DataFrame(y_train).sample(n = 1000, random_state = 0) # keeping it consistent with X_train_part
                                                                        # same seed for ramdom_state as in X_train

X_train_part_train, X_train_part_test, y_train_part_train, y_train_part_test = \
                    train_test_split( X_train_part, y_train_part, test_size=0.2, random_state=42)

print ('Part Train set:', X_train_part_train.shape,  y_train_part_train.shape)
print ('Part Test set:', X_train_part_test.shape,  y_train_part_test.shape)
```

```
Part Train set: (800, 49) (800, 1)
Part Test set: (200, 49) (200, 1)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics


solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
mean_acc2 = np.zeros((len(solvers)))

for i in range(len(solvers)):
    LR = LogisticRegression(C=0.01, solver=solvers[i]).fit(X_train_part_train,np.ravel(y_train_part_train))
    yhat = LR.predict(X_train_part_test)
    mean_acc2[i] = metrics.accuracy_score(y_train_part_test, yhat)

print("mean accuracy array:", mean_acc2)
print("\nmaximum mean accuracy value:", mean_acc2.max())
```

```
mean accuracy array: [0.695 0.695 0.695 0.695 0.695]

maximum mean accuracy value: 0.695
```

```
LR_final = LogisticRegression(C=0.01, solver= best_solver).fit(X_train, y_train)
LR_final
```

```
6]: LogisticRegression(C=0.01, 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='newton-cg', tol=0.0001, verbose=0,
                       warm_start=False)
```

6. Report

Report

Summary of the ccuracy scores of the different models

```
scores_dict = {'': ['K Nearest Neighbors','Decision Tree','Logistic Regression'], \
               'Jaccard Score': [KNN_jacc, DT_jacc, LR_jacc], \
               'F1-score': [KNN_f1, DT_f1, LR_f1], \
               'Subset Accuracy Score': [KNN_acc, DT_acc, LR_acc], \
               'Log Loss': ['NA','NA', LR_logloss] }

scores_Report = pd.DataFrame.from_dict(scores_dict)

scores_Report.set_index('', drop = True, inplace = True)

scores_Report
```

54]:

	Jaccard Score	F1-score	Subset Accuracy Score	Log Loss
K Nearest Neighbors	0.502418	0.607582	0.694311	NA
Decision Tree	0.496116	0.582208	0.704328	NA
Logistic Regression	0.496086	0.582288	0.704199	0.56721

6. Conclusion

Based on the above values I would conclude the model gives the better results with the KNN algorithm almost with the Accuracy score with 0.7.