Capstone Project - Car accident severity

1 Introduction/Business Understanding:

In an effort to reduce the frequency of car collisions in a community, an algorithm must be developed to predict the severity of an accident given the current weather, road and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful.

In most cases, not paying enough attention during driving, abusing drugs and alcohol are the main causes of occurring accidents that can be prevented by enacting harsher regulations. Besides the mentioned reasons weather, visibility, or road conditions are the major 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 locations.

2 Data

Import Primary Modules. The first thing we'll do is import two key data analysis modules: pandas and Numpy.

In [3]:

2.1 Data Understanding

In this assignment we will be using shared data file : Data-Collision.csv

Download the dataset and read it into a pandas dataframe.

```
In [4]: df_collision = pd.read_csv("https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv")
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3020: DtypeWarning: Columns (33) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Let's take a look at the overview of our dataset.

:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	INTKEY	SEVERITYCODE.1	PE
count	194673.000000	189339.000000	189339.000000	194673.000000	194673.000000	194673.000000	65070.000000	194673.000000	19
mean	1.298901	-122.330518	47.619543	108479.364930	141091.456350	141298.811381	37558.450576	1.298901	2.4
std	0.457778	0.029976	0.056157	62649.722558	86634.402737	86986.542110	51745.990273	0.457778	1.3
min	1.000000	-122.419091	47.495573	1.000000	1001.000000	1001.000000	23807.000000	1.000000	0.0
25%	1.000000	-122.348673	47.575956	54267.000000	70383.000000	70383.000000	28667.000000	1.000000	2.0
50%	1.000000	-122.330224	47.615369	106912.000000	123363.000000	123363.000000	29973.000000	1.000000	2.0
75%	2.000000	-122.311937	47.663664	162272.000000	203319.000000	203459.000000	33973.000000	2.000000	3.0
max	2.000000	-122.238949	47.734142	219547.000000	331454.000000	332954.000000	757580.000000	2.000000	81.

Let's take a look at the first five items in our dataset.

Our predictor or target variable will be 'SEVERITYCODE' because it is used to measure the severity of an accident from 0 to 5 within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.

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)

Furthermore, because of the existence of null values in some records, the data needs to be pre-processed before any further processing.

2.2 Data Pre-processing

The dataset in the original form is not ready for data analysis. In order to prepare the data, first, we need to drop the non-relevant columns. Also, most of the features are of type object, when they should be numerical type.

After analyzing the data set, I have decided to focus on only four features, severity, weather conditions, road conditions, and light conditions, among others.

To get a good understanding of the dataset, Now let's check different values in the features.

```
In [7]: df_collision["SEVERITYCODE"].value_counts()
```

```
Out[7]: 1    136485
        2     58188
        Name: SEVERITYCODE, dtype: int64
```

The results show, the target feature is imbalance, so we use a simple statistical technique to balance it by down sampling the majority class.

```
In [8]: df_col_max = df_collision[df_collision.SEVERITYCODE == 1]
        df_col_min = df_collision[df_collision.SEVERITYCODE == 2]

        df_coll_downsampl = resample(df_col_max,
                                     replace = False,
                                     n_samples = 58188,
                                     random_state = 100
                                    )
        df_balanced = pd.concat([df_coll_downsampl, df_col_min])
        df_balanced.SEVERITYCODE.value_counts()
```

```
Out[8]: 2    58188
        1    58188
        Name: SEVERITYCODE, dtype: int64
```

We will be concerntrating on features 'WEATHER', 'ROADCOND' and 'LIGHTCOND' so convert them into int format for analysis.

3 Methodology

For implementing the solution, we will be using Github as a repository and Jupyter Notebook to pre-process data and build Machine Learning models.

We will use the following models:

- **K-Nearest Neighbor (KNN)**
- **Decision Tree**
- **Logistic Regression**

After importing necessary packages and splitting preprocessed data into test and train sets, for each machine learning model, I have built and evaluated the model and shown the results as follow:

3.1 K-Nearest Neighbour (KNN)

KNN will help us predict the severity code of an outcome by finding the most similar to data point within k distance.

In [77]:
```
#Check best value of k
best_knn = 0
prev_score = 0
for k in range(1, 10):
    knn_model  = KNeighborsClassifier(n_neighbors = k).fit(x_train, y_train)
    knn_yhat = knn_model.predict(x_test)
    print("For K = {} accuracy = {}".format(k,accuracy_score(y_test,knn_yhat)))
    if ( accuracy_score(y_test,knn_yhat) >= prev_score ):
        best_knn = k
        prev_score = accuracy_score(y_test,knn_yhat)
```

```
For K = 1 accuracy = 0.5081512160733169
For K = 2 accuracy = 0.5087240042298202
For K = 3 accuracy = 0.514760310186817
For K = 4 accuracy = 0.511059217483257
For K = 5 accuracy = 0.510927035600987
For K = 6 accuracy = 0.511587945012337
For K = 7 accuracy = 0.5165667959111738
For K = 8 accuracy = 0.5159058864998237
For K = 9 accuracy = 0.5545470567500881
```

In [78]:
```
print ("KNN model is best for k = ", best_knn)
```

```
KNN model is best for k =  9
```

In [79]:
```
#Building the model with best value of K
best_knn_model = KNeighborsClassifier(n_neighbors = best_knn).fit(x_train, y_train)
best_knn_model
```

Out[79]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=9, p=2,
           weights='uniform')
```

3.2 Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyse the consequences of a decision. It context, the decision tree observes all possible outcomes of different weather conditions.

In [81]:
```
# importing libraries
from sklearn.tree import DecisionTreeClassifier
```

In [82]:
```
#Find best value for depth
best_depth = 0
prev_dscore = 0

for d in range(1,10):
    dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = d).fit(x_train, y_train)
    dt_yhat = dt.predict(x_test)
    print("For depth = {}  the accuracy score is {} ".format(d, accuracy_score(y_test, dt_yhat)))
    if (accuracy_score(y_test, dt_yhat) >= prev_dscore ):
        best_depth = d
        prev_dscore = accuracy_score(y_test, dt_yhat)
```

```
For depth = 1  the accuracy score is 0.5493479027141347
For depth = 2  the accuracy score is 0.5493479027141347
For depth = 3  the accuracy score is 0.5510662671836447
For depth = 4  the accuracy score is 0.5582481494536482
For depth = 5  the accuracy score is 0.5581600281988015
For depth = 6  the accuracy score is 0.5574991187874515
For depth = 7  the accuracy score is 0.5584684525907649
For depth = 8  the accuracy score is 0.5587768769827283
For depth = 9  the accuracy score is 0.559525907648925
```

In [83]:
```
print ("Decision Tree model is best for depth d = ", best_depth)
```

```
Decision Tree model is best for depth d =  9
```

In [85]:
```
# Creating the best model for decision tree with best value of depth

best_dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = best_depth).fit(x_train, y_train)
best_dt_model
```

Out[85]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=9,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
```

3.3 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.

Logistic Regression

In [87]:
```
# importing libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
```

In [88]:
```
best_solver = ''
prev_rscore = 0

for k in ('lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag'):
    lr_model = LogisticRegression(C = 0.01, solver = k).fit(x_train, y_train)
    lr_yhat = lr_model.predict(x_test)
    y_prob = lr_model.predict_proba(x_test)
    print('When Solver is {}, logloss is : {}'.format(k, log_loss(y_test, y_prob)))
    if (log_loss(y_test, y_prob) >= prev_rscore ):
        best_solver = k
        prev_rscore = log_loss(y_test, y_prob)
```

```
When Solver is lbfgs, logloss is : 0.6830913862931347
When Solver is saga, logloss is : 0.6830914098436894
When Solver is liblinear, logloss is : 0.6830913271342262
When Solver is newton-cg, logloss is : 0.6830914111953412
When Solver is sag, logloss is : 0.683091376175836
```

In [89]:
```
print("Solver : '{}' has the best score {}".format(best_solver,prev_rscore))
```

```
Solver : 'newton-cg' has the best score 0.6830914111953412
```

In [90]:
```
# Best logistic regression model with best solver

best_lr_model = LogisticRegression(C = 0.01, solver = best_solver).fit(x_train, y_train)
best_lr_model
```

Out[90]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='newton-cg',
          tol=0.0001, verbose=0, warm_start=False)
```

In [91]:
```
# Evaluation Metrics
```

4 Model Evaluation

Model Evaluation

In [92]:
```
# Jaccard

# KNN
knn_yhat = best_knn_model.predict(x_test)
jacc1 = round(jaccard_similarity_score(y_test, knn_yhat), 2)

# Decision Tree
dt_yhat = best_dt_model.predict(x_test)
jacc2 = round(jaccard_similarity_score(y_test, dt_yhat), 2)

# Logistic Regression
lr_yhat = best_lr_model.predict(x_test)
jacc3 = round(jaccard_similarity_score(y_test, lr_yhat), 2)

jss = [jacc1, jacc2, jacc3]
jss
```

Out[92]: [0.55, 0.56, 0.53]

In [93]:
```
# F1_score

# KNN
knn_yhat = best_knn_model.predict(x_test)
f1 = round(f1_score(y_test, knn_yhat, average = 'weighted'), 2)

# Decision Tree
dt_yhat = best_dt_model.predict(x_test)
f2 = round(f1_score(y_test, dt_yhat, average = 'weighted'), 2)

# Logistic Regression
lr_yhat = best_lr_model.predict(x_test)
f3 = round(f1_score(y_test, lr_yhat, average = 'weighted'), 2)

f1_list = [f1, f2, f3]
f1_list
```

Out[93]: [0.49, 0.53, 0.53]

In [96]:
```
# Log Loss

# Logistic Regression
lr_prob = best_lr_model.predict_proba(x_test)
ll_list = ['NA','NA', round(log_loss(y_test, lr_prob), 2)]
ll_list
```

Out[96]: ['NA', 'NA', 0.68]

In [97]:
```
columns = ['KNN', 'Decision Tree', 'Logistic Regression']
index = ['Jaccard', 'F1-score', 'Logloss']

accuracy_df = pd.DataFrame([jss, f1_list, ll_list], index = index, columns = columns)
accuracy_df1 = accuracy_df.transpose()
accuracy_df1.columns.name = 'Algorithm'
accuracy_df1
```

Out[97]:

Algorithm	Jaccard	F1-score	Logloss
KNN	0.55	0.49	NA
Decision Tree	0.56	0.53	NA
Logistic Regression	0.53	0.53	0.68

5 Discussion

In the beginning of this notebook, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so label encoding was used to created new classes that were of type int; a numerical data type.

After solving that issue we were presented with another - imbalanced data. As mentioned earlier, class 1 was nearly three times larger than class 2. The solution to this was down sampling the majority class with sklearn's resample tool. We down sampled to match the minority class exactly with 58188 values each.

Once we analysed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbour, Decision Tree and Logistic Regression. Although the first two are ideal for this project, logistic regression made most sense because of its binary nature.

Evaluation metrics used to test the accuracy of our models were Jaccard index, f-1 score and log loss for logistic regression. Choosing different k, max depth and hypermeter C values helped to improve our accuracy to be the best possible.

6 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).