Car Accident Severity Prediction Model For City of Seattle

Intro and Business Understanding:

Car accidents are a leading cause of injuries. Over 5.6 million car accidents were reported in the United States in 2012. Of these, over 30,000 were fatal, and another 1.6 million involved other injuries. The damage they leave behind can be immense. The economic cost of car accidents is estimated to be $277 billion each year, or around $897 for every person living in the United States. While car accident injuries can vary from person to person and from crash to crash. However, if we can build a model to predict the severity of a car accident, that would avoid a lot of unnecessary accidents and injuries, even deaths. Therefore, the Seattle government is planning to use this model that alert drivers, health systems, and police to remind them to pay more attention to critical situations.

Imagine a scenario that if there is a rainy day and you are planning to drive to another city. The model can predict the severity of the accident severity based on the condition of traffic, report the accident location, weather conditions, and other factors that provide you more options for traveling, reschedule, or drive more carefully. However, distraction and not paying enough attention while driving are important reasons that cause car accidents. This model will alert drivers driving more carefully and can be prevented by enacting harsher regulations.

The target audience of this project is the Seattle government, police, rescue groups, and drivers. This model and its prediction will provide advice for decision making and prevent unnecessary accidents and injuries for the city of Seattle.

Data understanding:

The dataset used for this project is based on car accidents that have occurred in the city of Seattle, Washington from 2004 to 2020. The dataset contains 37 independent variables and 194,673 rows. The independent variables include “INATTENTIONIND”, “WEATHER”, “ROADCOND” and other factors that would cause the accident. The dependent variable is “SEVERITYCODE” which contains numbers of “1” and “2”, they correspond to different levels of severity of car accidents. “1” represents for “Property Damage Only” and “2” means “Physical Injury”.

In my consideration, I will drop some non-critical and indecisive attributes. The following features which I choose to remain for building model and prediction.

1. UNDERINFL which means whether or not the driver was under the influence
2. WEATHER which represents the weather condition while the collision occurs
3. ROADCOND which represents the road conditions while the collision occurs
4. LIGHTCOND which represents the light conditions while the collision occurs.

However, the existent data contains null values in some records, the data has to be preprocessed before further processing and analyzing.

Data Preparation:

The data should be cleaning and preprocessing before the next steps. At first, I selected several features for modeling which are “SEVERITYCODE”, “INATTENTIONIND”, “UNDERINFL”, “WEATHER”, “ROADCOND”, “LIGHTCOND”, and “SPEEDING” respectively. However, there are some missing data in the dataset. After printing the number of missing data for each feature, it can be seen that “INATTENTIONIND” and “SPEEDING” features have more than 16,000 missing data, the rest of the others only have around 5000 missing data which account for only around 4% of the total data. Therefore, I decided to drop the column “INATTENTIONIND” and “SPEEDING” and drop missing data rows for the rest of the others.

```
#missing data
df.replace('?',np.nan, inplace=True)
missing = df.isnull()
for column in missing.columns.values.tolist():
    print(column)
    print(missing[column].value_counts())
    print("")
```

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

INATTENTIONIND
True     164868
False     29805
Name: INATTENTIONIND, dtype: int64

UNDERINFL
False    189789
True       4884
Name: UNDERINFL, dtype: int64

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
True     185340
False      9333
Name: SPEEDING, dtype: int64
```

However, the data is not balanced and not standardized yet currently. The next step is to balance the data. Our dependent variable of the model is “SEVERITYCODE”, which has 132,285 data points for CODE “1” and 57,052 for CODE “2”. It is highly imbalanced and will cause bias for the model. Therefore, in order to make it more balanced and reduce the bias, resampling is used in this step (see figure below)

```
#Balancing the dataset
from sklearn.utils import resample

df_maj = df[df.SEVERITYCODE==1]
df_min = df[df.SEVERITYCODE==2]
resample_df=resample(df_maj,replace=False,n_samples=57052,random_state=123)

new_df=pd.concat([resample_df,df_min])
new_df.SEVERITYCODE.value_counts()
```

```
2    57052
1    57052
Name: SEVERITYCODE, dtype: int64
```

Each of the selected features has different catalogs, and machine learning models require numerical data. Then I converted each catalog value into different numerical data. The process is shown below:

```
#preprocessing the data
new_df['UNDERINFL'].replace(to_replace=['N','0','Y','1'],value=[0,0,1,1],inplace=True)
```

```
new_df['WEATHER'].replace(to_replace=['Clear',
                                      'Raining',
                                      'Overcast',
                                      'Partly Cloudy',
                                      'Snowing',
                                      'Fog/Smog/Smoke',
                                      'Sleet/Hail/Freezing Rain',
                                      'Blowing Sand/Dirt',
                                      'Severe Crosswind',
                                      'Other','Unknown'],value=[1,2,3,4,5,6,7,8,9,10,11],inplace=True)
```

```
new_df['ROADCOND'].replace(to_replace=['Dry',
                                       'Sand/Mud/Dirt',
                                       'Wet','Standing Water',
                                       'Ice','Snow/Slush',
                                       'Other','Oil','Unknown'],value=[1,2,3,4,5,6,7,8,9],inplace=True)
```

```
new_df['LIGHTCOND'].replace(to_replace=['Daylight',
                                        'Dark - Street Lights On',
                                        'Dark - No Street Lights',
                                        'Dark - Street Lights Off',
                                        'Dusk','Dawn',
                                        'Other','Unknown',
                                        'Dark - Unknown Lighting'],value=[1,2,3,4,5,6,7,8,9],inplace=True)
```

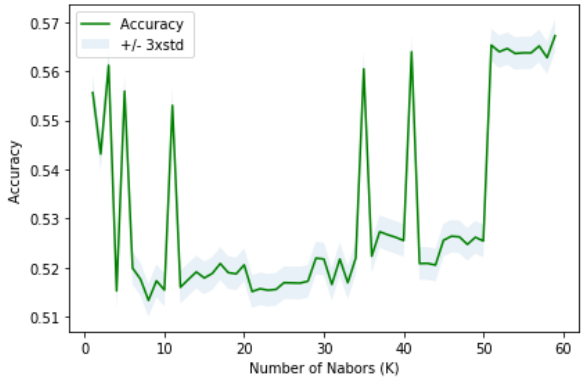
Modeling:

As for building the model for accident severity prediction in this project, the cleaning and re-sampling data were split into testing and training sets, 20% of the total data samples for testing, and the rest 80% data we used for training the model. Four machine learning models have been built and evaluated.

- K-Nearest Neighbor (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. It assumes that similar things exist in close proximity. In other words, similar things are near to each other. The value K is the most essential factor for modeling since it is dominant the

accuracy of the whole model. KNN model is constructed for k equals 1 to 60 in this case for modeling, and the best accuracy model was 0.57 with k equals 59.

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc-1*std_acc,mean_acc + 1 * std_acc,alpha=0.1)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
print("The best accuracy was with", mean_acc.max(),"with k=", mean_acc.argmax()+1)
```




```
The best accuracy was with 0.5671968800666053 with k= 59
```

- The decision tree algorithm can be used for solving classification and regression problems as well. The data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or outcomes. And the decision nodes are where the data is split.

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier(criterion='entropy',max_depth=None)
DT.fit(x_train,y_train)
yhat_DT= DT.predict(x_test)
yhat_DT
```

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

- Support-vector machines (SVM) algorithms analyze data used for classification and regression analysis. An SVM model is a representation of different classes in a hyperplane in

multidimensional space. The hyperplane will be generated iteratively by SVM so that the error can be minimized.

```
#SVM
from sklearn.svm import SVC
clf = SVC(kernel='rbf')
clf.fit(x_train,y_train)

yhat_svm=clf.predict(x_test)
```

- The logistic regression algorithm is used for classification problems, it is used to assign observations to a discrete set of classes, and it is a predictive analysis algorithm based on the concept of probability. It transforms its output using the logistic sigmoid function to return a probability value.

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
LR=LogisticRegression(C=0.01, solver = 'liblinear')
LR.fit(x_train,y_train)
yhat_LR=LR.predict(x_test)

yhat_prob = LR.predict_proba(x_test)
```

Evaluation:

The models are evaluated based on several factors:

- Precision: the ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall (Sensitivity): the ratio of correctly predicted positive observations to all observations in the actual class
- F1 Score: the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as

accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have a similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

- Jaccard Score: it compares members for two sets to see which members are shared and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it's easy to interpret, it is extremely sensitive to small sample sizes and may give erroneous results, especially with very small samples or data sets with missing observations.
- Log-Loss: an appropriate performance measure when you're model output is the probability of a binary outcome. The log-loss measure considers confidence of the prediction when assessing how to penalize incorrect classification.

Results

Model	f1-score	Jaccard Score	LogLoss	Severity Code	Precision	Recall
KNN	0.54	0.57	NA	1	0.64	0.31
				2	0.54	0.82
Decision Tree	0.54	0.57	NA	1	0.64	0.31
				2	0.54	0.82
SVM	0.53	0.57	NA	1	0.65	0.28
				2	0.54	0.85
Logistic Regression	0.53	0.56	0.67	1	0.61	0.33
				2	0.54	0.78

Discussion and Conclusion

From the analysis of the result of 4 machine learning algorithm models on the city of Seattle collision dataset, it can be seen that four models are performed very similarly, but KNN and Decision Tree methods are both 1% higher than the rest of the others in F1 Score and Logistic Regression has 1% lower than the rest of 3 others in Jaccard Score. In Logistic Regression, the Log-loss is 0.67 which is a relatively high value indicates a high uncertainty/entropy of my model.

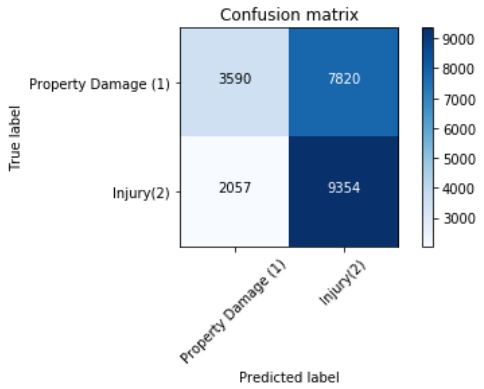
Therefore, the models can be further improved or optimized for higher accuracy. The models may perform better if different features can be selected for building the models, or different criterion and a maximum depth of the decision tree model can be selected, or different kernel of the SVM model can be selected.

Based on the dataset from knowing the driver is under influenced by alcohol/drug or not, weather, road, and light conditions, we build different models and it can be concluded that particular conditions have a somewhat impact on the severity code which results in property damage (class 1) or injury (class 2).

Appendix’

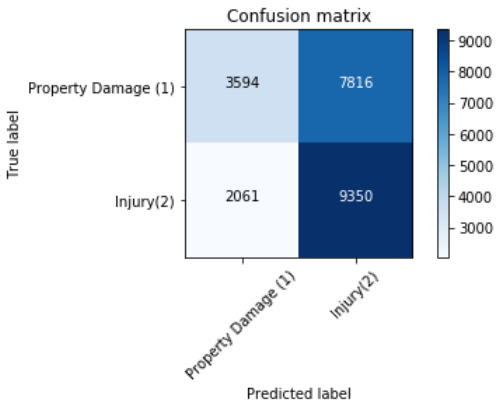
- KNN Confusion Matrix

```
              precision    recall  f1-score   support

           1       0.64      0.31      0.42     11410
           2       0.54      0.82      0.65     11411

    accuracy                           0.57     22821
   macro avg       0.59      0.57      0.54     22821
weighted avg       0.59      0.57      0.54     22821

Confusion matrix, without normalization
[[3590 7820]
 [2057 9354]]
```



- Decision Tree Confusion Matrix

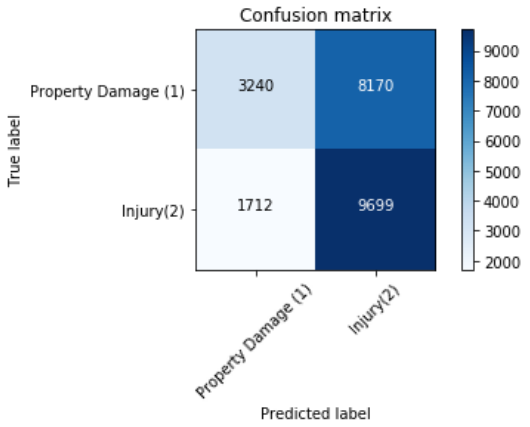
```
              precision    recall  f1-score   support

           1       0.64      0.31      0.42     11410
           2       0.54      0.82      0.65     11411

    accuracy                           0.57     22821
   macro avg       0.59      0.57      0.54     22821
weighted avg       0.59      0.57      0.54     22821

Confusion matrix, without normalization
[[3594 7816]
 [2061 9350]]
```



- SVM Confusion Matrix

	precision	recall	f1-score	support
1	0.65	0.28	0.40	11410
2	0.54	0.85	0.66	11411
accuracy			0.57	22821
macro avg	0.60	0.57	0.53	22821
weighted avg	0.60	0.57	0.53	22821

```
Confusion matrix, without normalization
[[3240 8170]
 [1712 9699]]
```



- Logistic Regression Confusion Matrix

	precision	recall	f1-score	support
1	0.61	0.33	0.43	11410
2	0.54	0.78	0.64	11411
accuracy			0.56	22821
macro avg	0.57	0.56	0.53	22821
weighted avg	0.57	0.56	0.53	22821

```
Confusion matrix, without normalization
[[3792 7618]
 [2475 8936]]
```