# Collision severity prediction in Seattle City

Xinyuan Liu

## Business understanding

The main objective of the current data science project is to reduce the collision rate in a community. Specifically, this report will be targeted to drivers driving on bad road conditions along highways in Seattle, USA.

To achieve this goal, we will apply machine learning algorithms to predict the severity level of car accidents by training historical traffic collision data. In practice, the model receives real-time road condition data and alerts drivers with a predicted collision severity level so that the drivers can take precautions accordingly.

## Data understanding

The data consists of collision events taking place in Seattle from 2004/01/01 to 2020/05/20. Each row represents a collision event. In total there are 194673 events are recorded. Based on the definition of our problem, information that is needed is:

- road conditions when the collision happened
- data measuring the severity level of collision

As a result, the following attributes will be considered:

- The target field is 'SEVERITYCODE', as it is used to measure the severity of an accident. It has two possible values, as follows 1—prop damage, 2—injury.
- Attributes will be used to predict the severity level are 'WEATHER', 'ROADCOND', and 'LIGHTCOND', which are weather conditions, road conditions, and light conditions, respectively. These attributes are categorical, so we will convert these features to numerical values so that the machine learning models can handle it.

## Data source

Example Dataset is downloaded via: **Clicking here**

## Data cleaning

To work effectively with the data, it will be prepared in a way that addresses missing or invalid values and removes duplicates, toward ensuring that everything is properly formatted. We will mainly focus on 4 variables used for prediction, which are 'WEATHER', 'ROADCOND', 'LIGHTCOND' and 'SEVERITYCODE'.

For road conditions, there are 10 distinct values, out of which 'nan' and 'Unknown' should be dropped out as they are missing values. Thus in total, there are 8 road conditions: 'Wet' 'Dry' 'Snow/Slush' 'Ice' 'Other' 'Sand/Mud/Dirt' 'Standing Water' 'Oil'

For weather conditions, there are 12 distinct values, out of which 'nan' and 'Unknown' should be dropped for the same reason. Thus in total, there are 10 weather conditions: 'Overcast' 'Raining' 'Clear' 'Other' 'Snowing' 'Fog/Smog/Smoke' 'Sleet/Hail/Freezing Rain' 'Blowing Sand/Dirt' 'Severe Crosswind' 'Partly Cloudy'

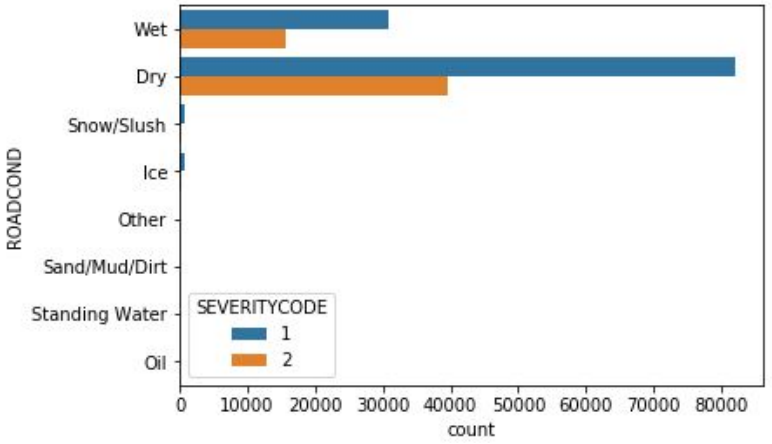
For light condition, there are 10 distinct values, out of which 'nan' and 'Unknown' should be dropped are for the same reason. 'Dark - No Street Lights' and 'Dark - Street Lights Off' should be merged as they represent the same condition. Thus in total, there are 7 light conditions: 'Daylight' 'Dark - Street Lights On' 'Dusk' 'Dawn' 'Dark - Street Lights Off' 'Other' 'Dark - Unknown Lighting'.

## Exploratory Data Analysis

After cleaning the data, the frequency of each condition is examined to understand their distribution. As can be seen that dry, wet, ice, and snow are the most frequent factors. However, this doesn't mean these factors lead to the collision. As for the weather conditions, clear, ranging and overcast are the top three most frequent conditions. And the light conditions are distributed more evenly. It seems that the collision is mostly like happen in good conditions when the road is dry, weather clear, and during day time. This does make sense as the collisions tend to happen when the roads are busy. And no conclusion can be reached so far.

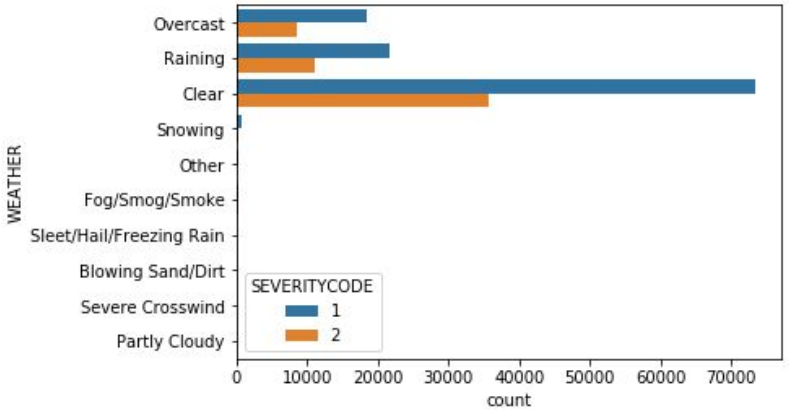
```
df_1['ROADCOND'].value_counts()
```

```
Dry                121754
Wet                 46469
Ice                  1096
Snow/Slush            845
Other                 111
Standing Water        108
Sand/Mud/Dirt          66
Oil                    61
```



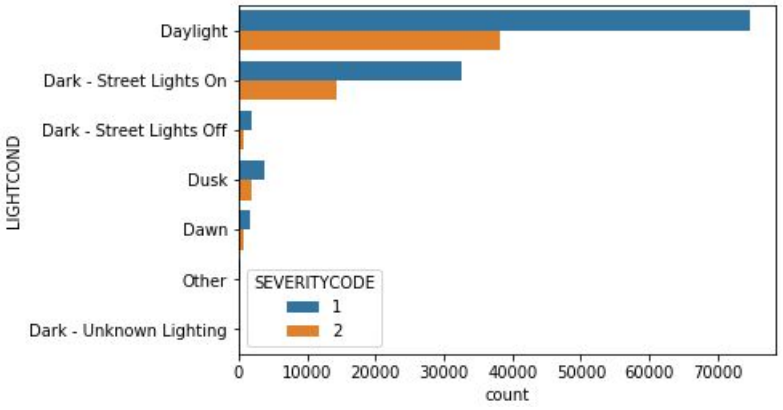

```
df_1['WEATHER'].value_counts()
```

```
Clear                       108960
Raining                      32735
Overcast                     26977
Snowing                        831
Fog/Smog/Smoke                 555
Other                          264
Sleet/Hail/Freezing Rain       111
Blowing Sand/Dirt               47
Severe Crosswind                25
Partly Cloudy                    5
```




```
df_1['LIGHTCOND'].value_counts()
```

```
Daylight                    112836
Dark - Street Lights On      46845
Dusk                          5662
Dawn                          2424
Dark - No Street Lights       1415
Dark - Street Lights Off      1118
Other                          201
Dark - Unknown Lighting          9
```



## Data Balance

By comparing the frequency of severity levels, it's found that the quantity of severity 1 events is nearly double that of severity 2. To make an accurate prediction, the dataset needs to be balanced. In practice, the severity 1 events are sampled with the amount that equals to severity 2, which is 55851.

After data is balanced, the distribution of each variable is also balanced in terms of severity levels. We are now ready for data modeling.

```
df_1['SEVERITYCODE'].value_counts()
```

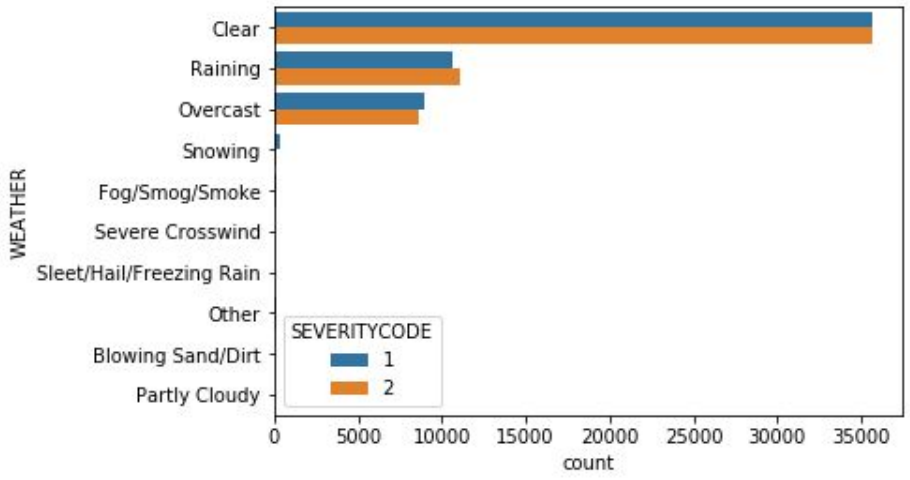
```
1    114659
2     55851
```

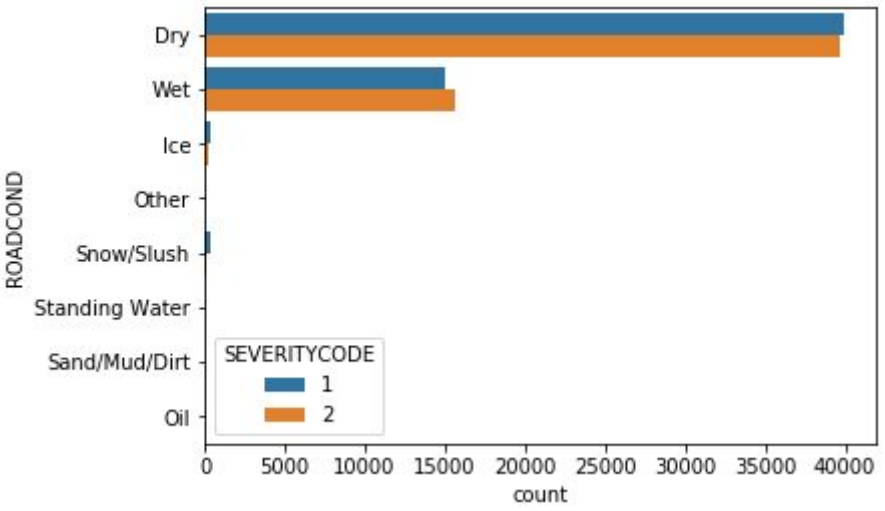
```
df_2 = df_1[df_1['SEVERITYCODE']==1].sample(n=55851)
df_2.shape
```

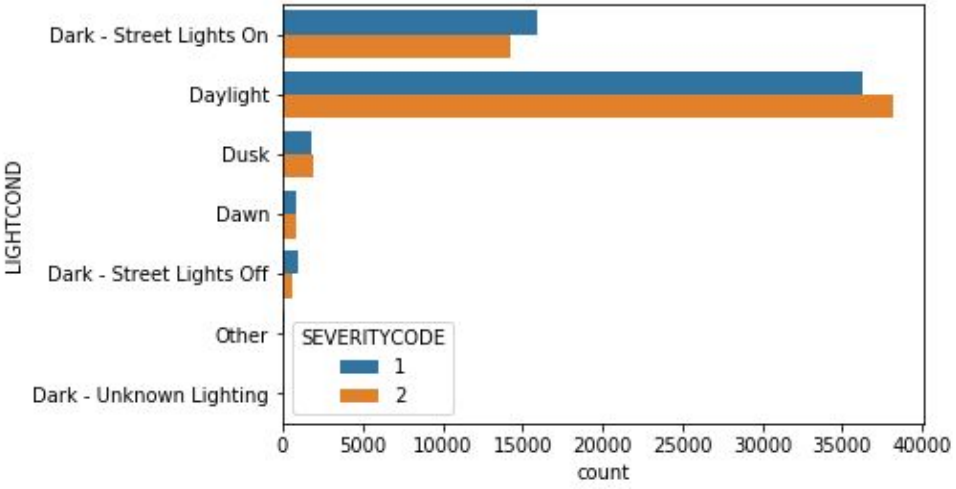
```
(55851, 4)
```

```
df_3 = df_2.append(df_1[df_1['SEVERITYCODE']==2])
df_3['SEVERITYCODE'].value_counts()
```

```
2    55851
1    55851
Name: SEVERITYCODE, dtype: int64
```




Dry
Wet
Ice
Other
Snow/Slush
Standing Water
Sand/Mud/Dirt
Oil
ROADCOND
SEVERITYCODE
1
2
0
5000
10000
15000
20000
25000
30000
35000
40000
count


Dark - Street Lights On
Daylight
Dusk
Dawn
Dark - Street Lights Off
Other
Dark - Unknown Lighting
LIGHTCOND
SEVERITYCODE
1
2
0
5000
10000
15000
20000
25000
30000
35000
40000
count

## Methodology

The problem at hand is a classification problem and I will apply 3 algorithms to solve it, which are K nearest neighbor (KNN), Decision Tree, and Logistic Regression. KNN predicts the severity level by finding the most similar data points by measuring a distance. A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. As there are only two severity levels within the provided dataset, our model will only predict one of those two classes. This binary feature makes it perfect to apply logistic regression algorithm.

## Data preprocessing

To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array.  First Declare the following variables:

- X as the Feature Matrix ('WEATHER', 'ROADCOND', 'LIGHTCOND')

```
X = df_3[['WEATHER', 'ROADCOND', 'LIGHTCOND']].values
X[0:5]
```

```
array([['Clear', 'Dry', 'Daylight'],
       ['Clear', 'Dry', 'Dark - Street Lights On'],
       ['Clear', 'Dry', 'Daylight'],
       ['Clear', 'Dry', 'Daylight'],
       ['Clear', 'Dry', 'Dusk']], dtype=object)
```

- y as the response vector ('SEVERITYCODE')

```
y = df_3["SEVERITYCODE"].values
y[0:5]
```

```
array([2, 2, 2, 1, 2])
```

Then convert categorical variables to numerical values by using the preprocessing function in the scikit-learn library.

```
from sklearn import preprocessing
le_weather = preprocessing.LabelEncoder()
le_weather.fit(['Overcast', 'Raining', 'Clear', 'Snowing', 'Other',
       'Fog/Smog/Smoke', 'Sleet/Hail/Freezing Rain', 'Blowing Sand/Dirt',
       'Severe Crosswind', 'Partly Cloudy'])
X[:,0] = le_weather.transform(X[:,0])


le_roadcond = preprocessing.LabelEncoder()
le_roadcond.fit(['Wet', 'Dry', 'Snow/Slush', 'Ice', 'Other', 'Sand/Mud/Dirt',
       'Standing Water', 'Oil'])
X[:,1] = le_roadcond.transform(X[:,1])


le_lightcond = preprocessing.LabelEncoder()
le_lightcond.fit(['Daylight', 'Dark - Street Lights On', 'Dark - Street Lights Off',
       'Dusk', 'Dawn', 'Other', 'Dark - Unknown Lighting'])
X[:,2] = le_lightcond.transform(X[:,2])

X[0:5]
```

```
array([[1, 0, 4],
       [1, 0, 1],
       [1, 0, 4],
       [1, 0, 4],
       [1, 0, 5]], dtype=object)
```

## Data normalization

Data Standardization gives data to zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on the distance of cases:

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

```
array([[-0.71008253, -0.6236523 ,  0.59837554],
       [-0.71008253, -0.6236523 , -1.52606146],
       [-0.71008253, -0.6236523 ,  0.59837554],
       [-0.71008253, -0.6236523 ,  0.59837554],
       [-0.71008253, -0.6236523 ,  1.30652121]])
```

## Train/Test Spit

Then we will apply Train/Test Split to our datasets, it involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. This will provide a more accurate evaluation of out-of-sample accuracy because the testing dataset is not part of the dataset that has been used to train the data. In this case, we will use 70% of data for training and the rest 30% for testing.

```
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=4)
```

```
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (78191, 3) (78191,)
Test set: (33511, 3) (33511,)
```

## Prediction

## K nearest neighbor (KNN)

K-Nearest Neighbors is an algorithm for supervised learning. Where the data is 'trained' with data points corresponding to their classification. Once a point is to be predicted, it takes into account the 'K' nearest points to it to determine it's classification.

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

Lets start the algorithm with k = 8:

```
k = 8
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
```

We can use the model to predict the test set:

```
y_test_predKNN = neigh.predict(X_test)
```

## Decision Tree

We will first create an instance of the DecisionTreeClassifier called decisionTree.

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

Inside of the classifier, specify criterion="entropy" so we can see the information gain of each node. Next, we will fit the data with the training feature matrix X_train and training response vector y_train

```
decisionTree = DecisionTreeClassifier(criterion="entropy", max_depth = 9).fit(X_train,y_train)
```

Let's make some predictions on the testing dataset and store it into a variable called y_test_predTree.

```
y_test_predTree = decisionTree.predict(X_test)
```

## 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.

Lets build our model using LogisticRegression from Scikit-learn package. This function implements logistic regression and can use different numerical optimizers to find parameters, we choose ‘liblinear’ for current study.

Regularization is a technique used to solve the overfitting problem in machine learning models. C parameter indicates inverse of regularization strength which must be a positive float. Let's first try with 0.01. Smaller values specify stronger regularization. Now lets fit our model with train set:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
```

Now we can predict using our test set:

```
y_test_predLR = LR.predict(X_test)
```

## Result evaluation

We can use classification_report to get a more detailed overview of the prediction, where

- Precision is a measure of the accuracy provided that a class label has been predicted. It is defined by: precision = TP / (TP + FP)
- Recall is a true positive rate. It is defined as: Recall = TP / (TP + FN)

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifier has a good value for both recall and precision.

And finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels. And it can be concluded that KNN provides the highest accuracy.

```
from sklearn.metrics import classification_report
print (classification_report(y_test, y_test_predKNN))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.50 | 0.54 | 0.52 | 16602 |
| 2 | 0.51 | 0.48 | 0.50 | 16909 |
| avg / total | 0.51 | 0.51 | 0.51 | 33511 |

```
print (classification_report(y_test, y_test_predTree))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.53 | 0.26 | 0.34 | 16602 |
| 2 | 0.52 | 0.78 | 0.62 | 16909 |
| avg / total | 0.52 | 0.52 | 0.48 | 33511 |

```
print (classification_report(y_test, y_test_predLR))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.52 | 0.31 | 0.38 | 16602 |
| 2 | 0.51 | 0.72 | 0.60 | 16909 |
| avg / total | 0.52 | 0.52 | 0.49 | 33511 |

## Discussion

In our study, we apply K-Nearest Neighbor, Decision Tree, and Logistic Regression to solve the classification problem. Choosing different k, max depth and parameter C values helped to improve their accuracy to be the best possible.
The K-Nearest Neighbor model performs pretty well in predicting both the severity level 1 and 2 events, as the recall is 54% and 48%.On the contrary, the Decision Tree, and Logistic Regression gave bad recall for severity level 1 event, meaning only around 30% of the severity 1 event were successfully detected. However, the

accuracy of the prediction is around 50% for all the models, meaning we can only use our prediction result as a reference.

## Conclusion

By using the historical collision data consists of weather, road, and light conditions and their associated severity level, we created machine learning models based on the classification algorithms to predict collision severity.

The application of a specific model is mainly based on purposes. If the stakeholder cares more about avoiding severity level 2 events, then the decision Tree or logistic regression model is a good choice. If both severity level 1 & 2 events are to be alerted, then the KNN model is the best choice.