# Car Accident Severity Prediction for Driver Warning and Assistance Systems

## 1. Introduction/Business Problem

1. Background
   Car accidents could occur due to several factors. Some of the most important ones are road conditions, weather and visibility conditions. These conditions also influence the severity of the accidents. In this study, we aim to predict the severity of car accidents through reliable and robust machine learning models.

2. Business Problem
   There could be several factors that influence the occurrence and severity of car accidents. This study aims to investigate three main factors: Weather, Road conditions and Visibility conditions on the severity of accidents. A robust algorithm must be developed to predict the severity of accidents, so that the drivers could be warned about it.

3. Interest
   The Seattle transportation department would be very interested in predicting the severity of accidents and so, make the target audience for this case study. Since the study deals with predicting the severity of car accidents, the results would be particularly helpful for them to plan and improve the road conditions in the city and placement of street lights. They would also be curious to know the influence of other factors like weather and visibility conditions on the occurrence of accidents and the level of severity. The results of this study could also be useful to the drivers to show them the consequences of driving under these conditions. Another interesting application of this study is to assist the Active safety or Driver Assistance Systems of cars about the current driving conditions, in order to enhance the development and performance of such systems.

## 2. Data

The data being used in this study is the collision reports from Seattle, collected over several years (2004-present) by the Seattle Police Department and Traffic Records department. The data has 37 independent variables and 194,673 records. Most columns are of type 'object' and even if there are numbers, they might correspond to different categories. Some columns and rows have null values, which will be dealt with during the data pre-processing phase. There are several attributes in this data, but the ones that are of interest in this study are: "WEATHER", "ROADCOND" and "LIGHTCOND". Different conditions of weather, road and light conditions are mentioned in the data.

The target variable, also known as the dependent variable, is "SEVERITYCODE". It is made up of numbers, which correspond to different levels of severity:

0 No/negligible probability: no/negligible chance

1 Very low probability: Chance of car damage

2 Low probability: Chance of injury and/or car damage

3 Mild probability: Chance of serious injury and car damage

4 High probability: Fatal and car damage

The first five rows of the raw data is shown in the below image.

```
df.head()
```

| | SEVERITYCODE | X | Y | OBJECTID | INCKEY | COLDETKEY | REPORTNO | STATUS | ADDRTYPE | INTKEY | ... | ROADCOND | LIGHTCOND | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | -122.323148 | 47.703140 | 1 | 1307 | 1307 | 3502005 | Matched | Intersection | 37475.0 | ... | Wet | Daylight | N |
| **1** | 1 | -122.347294 | 47.647172 | 2 | 52200 | 52200 | 2607959 | Matched | Block | NaN | ... | Wet | Dark - Street Lights On | N |
| **2** | 1 | -122.334540 | 47.607871 | 3 | 26700 | 26700 | 1482393 | Matched | Block | NaN | ... | Dry | Daylight | N |
| **3** | 1 | -122.334803 | 47.604803 | 4 | 1144 | 1144 | 3503937 | Matched | Block | NaN | ... | Dry | Daylight | N |
| **4** | 2 | -122.306426 | 47.545739 | 5 | 17700 | 17700 | 1807429 | Matched | Intersection | 34387.0 | ... | Wet | Daylight | N |

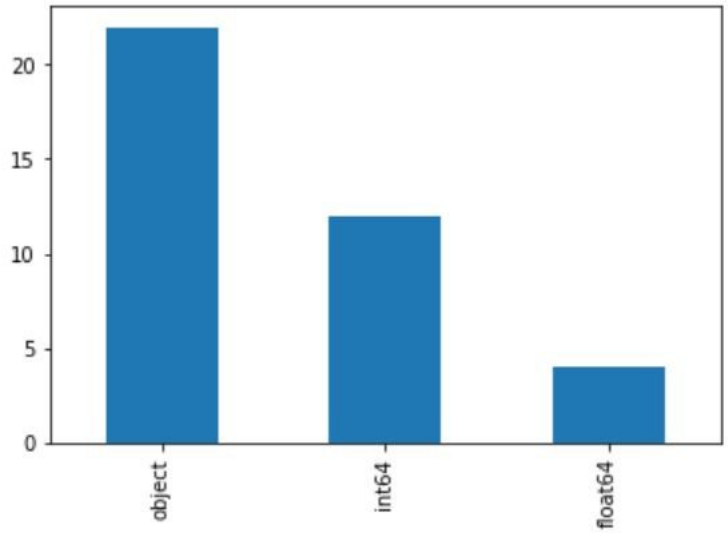
5 rows × 38 columns

### 2.1 Data Pre-processing

Most columns are of type ‘object’ and even if there are numbers, they might correspond to different categories. Some columns and rows have null values, which will be dealt with during this phase. We must use label encoding on these columns to convert the features to numbers. But first, we should drop the columns which are not required for this study.

```
df.dtypes.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb02865198>
```




| | SEVERITYCODE | WEATHER | ROADCOND | LIGHTCOND | WEATHER_CAT | ROADCOND_CAT | LIGHTCOND_CAT |
|---|---|---|---|---|---|---|---|
| **0** | 2 | Overcast | Wet | Daylight | 4 | 8 | 5 |
| **1** | 1 | Raining | Wet | Dark - Street Lights On | 6 | 8 | 2 |
| **2** | 1 | Overcast | Dry | Daylight | 4 | 0 | 5 |
| **3** | 1 | Clear | Dry | Daylight | 1 | 0 | 5 |
| **4** | 2 | Raining | Wet | Daylight | 6 | 8 | 5 |

The above table shows first five rows of the pre-processed dataframe. The columns WEATHER, ROADCOND and LIGHTCOND have been encoded such that the 'object' types are converted into 'int8' for easy handling of data. The attributes that are not of interest in this study are dropped.

### 2.2 Balancing the Dataset

```
pro_data["SEVERITYCODE"].value_counts()
```

```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

The dataset is checked if it is balanced. It has been observed that the data pertaining to SEVERITYCODE = 1 is almost three times the other code. Since this can influence our model, it has to be balanced such that the data pertaining to both codes are equal. To balance the dataset, the number of accidents with SEVERITYCODE=1 is downsampled.

```
from sklearn.utils import resample

pro_data_majority = pro_data[pro_data.SEVERITYCODE==1]
pro_data_minority = pro_data[pro_data.SEVERITYCODE==2]

#Downsample majority class
pro_data_majority_downsampled = resample(pro_data_majority,
                                         replace=False,
                                         n_samples=58188,
                                         random_state=123)

# Combine minority class with downsampled majority class
pro_data_balanced = pd.concat([pro_data_majority_downsampled, pro_data_minority])

# Display new class counts
pro_data_balanced.SEVERITYCODE.value_counts()
```

```
2    58188
1    58188
Name: SEVERITYCODE, dtype: int64
```

## 3. Methodology

Once the data is pre-processed and balanced, it is ready to be trained. A supervised learning models will be built using this data to come up with a formula that can predict the severity of an accident based on the inputs. Now that the data is ready, we can train the machine learning models with this data. Since the dataset has pre-defined severity codes and we have to predict the code class every case belongs to, it is clearly a classification problem.

The different models investigated here are:

1. K Nearest Neighbour:
2. Decision Tree
3. Logistic Regression

### 3.1 Feature vector and target variable

The feature vector consists of values describing the corresponding weather, road and light conditions for every datapoint. The target values are the Severity codes.

```
import numpy as np
X = np.asarray(pro_data_balanced[['WEATHER_CAT', 'ROADCOND_CAT', 'LIGHTCOND_CAT']])
X[0:5]
```

```
array([[ 6,  8,  2],
       [ 1,  0,  5],
       [10,  7,  8],
       [ 1,  0,  5],
       [ 1,  0,  5]], dtype=int8)
```

```
y = np.asarray(pro_data_balanced['SEVERITYCODE'])
y [0:5]
```

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

### 3.2 Normalize the feature vector

Data normalization is the process of rescaling one or more attributes to the range of 0 to 1. It is important to normalize the feature vectors, so that there is no bias in the model when learning happens.

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int8 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int8 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
array([[ 1.15236718,  1.52797946, -1.21648407],
       [-0.67488   , -0.67084969,  0.42978835],
       [ 2.61416492,  1.25312582,  2.07606076],
       [-0.67488   , -0.67084969,  0.42978835],
       [-0.67488   , -0.67084969,  0.42978835]])
```

### 3.3 Train-test split

This is a common procedure in machine learning. The whole dataset is split into training and test sets. This helps in evaluating the models effectively using unseen data, which in turn prevents overfitting. In this study, a split of 3:1 has been chosen, meaning, the models would be trained using 75% of the dataset. The rest 25% of the dataset is the test set, on which the model evaluation is performed. Here, an inbuilt function from scikit-learn has been used to make the split.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (87282, 3) (87282,)
Test set: (29094, 3) (29094,)
```

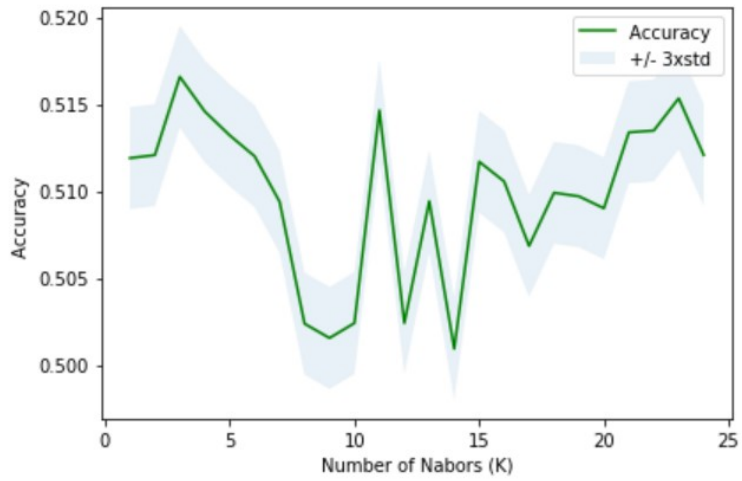
### 3.4 K Nearest Neighbours Model

Along with creating a KNN model, it is also essential that the correct value of K is identified. The value of K which provides the most accurate results on the test dataset would be chosen as the ideal K. In order to find this, a range of values of K starting from 1 until 10 are chosen. The model is built for each of the value of K. The accuracy on the test dataset is calculated and the K pertaining to the maximum accuracy is chosen.

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

Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

```
array([0.51189249, 0.51206434, 0.51656699, 0.51457345, 0.5131986 ,
       0.5119956 , 0.50938338, 0.50237162, 0.50154671, 0.50240599,
       0.51464219, 0.50240599, 0.50941775, 0.50092803, 0.51168626,
       0.510552  , 0.5068399 , 0.50989895, 0.50969272, 0.50900529,
       0.51337045, 0.51347357, 0.51532962, 0.51206434])
```

From the results above, K=3 has the maximum accuracy. This can also be visualized through a chart.



### 3.5 Decision Tree

A decision tree is also built using the training set. Different values of the depth is experimented with and the model corresponding to the highest accuracy is chosen.

```
from sklearn import tree

clf_tree = tree.DecisionTreeClassifier(criterion="entropy", max_depth = 10)
clf_tree = clf_tree.fit(X_train, y_train)
yhat_dt = clf_tree.predict(X_test)
yhat_dt
```

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

```
print (yhat_dt [0:5])
print (y_test [0:5])
```

```
[2 2 1 1 2]
[2 2 1 1 1]
```

### 3.6 Logistic Regression

Due to the fact that the Severity code in the dataset is made up of just two values, the problem can be treated as a binary classification problem. Additionally, a probability score for the occurrence and severity level of accidents would also be useful. To address this, a logistic regression approach has also been tried. Different values of parameter C is experimented with and the model corresponding to the highest accuracy is chosen.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=6, 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='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

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

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

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
array([[0.57263745, 0.42736255],
       [0.47083235, 0.52916765],
       [0.67438763, 0.32561237],
       ...,
       [0.47083235, 0.52916765],
       [0.68326727, 0.31673273],
       [0.46947758, 0.53052242]])
```

## 4. Results and Evaluation

It is important to check how the different models perform using some evaluation metrics. This will help us identify the most reliable and robust model for this data science problem. For evaluating the models, we use the test set, which is 25% of the total dataset. Model evaluation also helps in tuning hyperparameters and model parameters like the value of K in KNN, the depth of decision tree and the value of C in logistic regression.

There are different metrics that are compared:

1. Jaccard similarity score
2. F1 score
3. Log loss for logistic regression model

All the three metrics are readily available from scikit-learn package. The results of evaluation for the different approaches are shown in this section.

### K Nearest Neighbour

```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, yhat_knn)
```

```
0.5165669897573383
```

```
# F1-SCORE
f1_score(y_test, yhat_knn, average='macro')
```

```
0.46423304964062817
```

K=3 gives the maximum accuracy

### Decision Tree

```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, yhat_dt)
```

```
0.56688664329415
```

```
# F1-SCORE
f1_score(y_test, yhat_dt, average='macro')
```

```
0.5431403441878632
```

For logistic regression, an additional log loss is also calculated.

### Logistic Regression

```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, LRyhat)
```

```
0.5274283357393277
```

```
# F1-SCORE
f1_score(y_test, LRyhat, average='macro')
```

```
0.5127901543870825
```

```
# LOGLOSS
yhat_prob = LR.predict_proba(X_test)
log_loss(y_test, yhat_prob)
```

```
0.6849604847680921
```

## 5. Discussion

To start with, the data was loaded and studied. Columns with type 'object' were converted into categorial classes using label encoding.

An important observation at this stage was that the dataset was imbalanced. The solution to this was down-sampling the majority class with scikit-learn's resample tool. The majority class was down-sampled appropriately in order to make the dataset balanced.

After the pre-processing stage, the dataset was split into training and test sets in the ratio of 3:1. This is important in order to build accurate models and to prevent overfitting. The training data was then fed through three ML models: K-Nearest Neighbour, Decision Tree and Logistic Regression.

Evaluation metrics used to test the accuracy of our models were jaccard index, f-1 score and additionally, log loss for logistic regression. Choosing different k values helped determine the most accurate KNN model. Similarly, the max depth of the Decision tree and C for Logistic regression is also varied and the most accurate values are found. The results of evaluation were compared to determine the best suited model for this data science problem. Based on the evaluation results, it can be observed that the decision tree performs the best, though the differences amongst other models were small.

## 6. Conclusion

In this study, useful machine learning models to predict the severity of car accidents have been built. Although there is room for improvement of accuracy, use of evaluation metrics have helped in determining the best model.