Coursera Capstone Project

By: Shady Sakhel

Introduction

Did you know that on average 3,700 people lose their lives everyday due to road accidents? That is 1.35 million people a year! Not only that, another 50 million people suffer long term disabilities!

Countries all over the world work hard to reduce the number of fatalities and injuries that are caused by road accidents. From bicycle dedicated lanes, speed control, black-point systems to awareness campaigns, all these did help in reducing the number of accidents. Yet, "Road crashes are the leading cause of death in the U.S. for people aged 1-54" as reported by The Association for Safe International Road Travel. In this project, I will attempt to create a model that can predict the severity of an accident given the current weather and road conditions. This model can help warn drivers of the risk they are going to be taking when they decide to drive. In turn, they might change their route or even postpone the commute if its not urgent.

Business Understanding

Less accidents means less damages on health & property. Tools that can reduce the number of accidents can be of a great value to governments who would want a smooth traffic flow across their cities and protect the wellbeing of their citizens. Also, insurance companies who would be interested in minimizing their losses of repairing properties & covering medical bills. In this project, we will use the data collected by Seattle SPOT Traffic Management Division which contains records of all accidents from 2004 to present to help us predict the severity of accidents that could happen as a result of the current weather and road conditions. The record contains 37 attributes which we will analyze and use to be able to predict the severity of potential accidents with the highest level of accuracy.

Data Understanding

The dataset we selected has 194,673 rows and 37 different independent attributes. The target variable Y also known as Dependent Variable will be SEVERITY CODE. We will use the relevant attributes from the 37 attributes as our independent variable X. The data set is quite large and we need to get rid of irrelevant columns, rows with missing data which we can’t make an assumption for and also we need to make sure our data set is balanced.

The dependent variable, “SEVERITYCODE”, contains numbers that correspond to different levels of severity caused by an accident. The code that corresponds to the severity of the collision:

- 3—fatality

- 2b—serious injury
- 2—injury
- 1—prop damage
- 0—unknown

At this point we would like to note that the dataset provided by Seattle SPOT Traffic Management Division has data for accidents with “SEVERITYCODE” 1 and 2 only.

The independent variables are many and we list below the most important ones:

- PERSONCOUNT: The total number of people involved in the collision helps identify severity involved
- PEDCOUNT: The number of pedestrians involved in the collision helps identify severity involved
- PEDCYLCOUNT: The number of bicycles involved in the collision helps identify severity involved
- VEHCOUNT: The number of vehicles involved in the collision identify severity involved
- INCDATE : The date of the incident.
- ADDRTYPE: Collision address type: Alley, Block, Intersection
- COLLISIONTYPE: Collision Type
- JUNCTIONTYPE: Category of junction at which collision took place helps identify where most collisions occur
- INATTENTIONIND: Whether or not collision was due to inattention.
- UNDERINFL: Whether or not a driver involved was under the influence of drugs or alcohol.
- LOCATION: Description of the general location of the collision
- WEATHER: A description of the weather conditions during the time of the collision
- ROADCOND: The condition of the road during the collision
- LIGHTCOND: The light conditions during the collision
- SPEEDING: Whether speeding was a factor in the collision (Y/N)

Attributes like "INATTENTIONIND", "UNDERINFL" & "SPEEDING" cause noise in our data as we are interested in the effect of the weather and road condition on the severity of the accident. We know that these attributes do contribute to the severity of the accident but for the purpose of analyzing the weather and road condition only we will drop incidents that had been affected by "INATTENTIONIND", "UNDERINFL" & "SPEEDING".

Loading the Data

I will be using JupyterLab on Skills Network Lab for this project. The language of the code will be Python.

Loading the Data

[6]:
```
df = pd.read_csv("Data-Collisions-Edited.csv")
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3072: DtypeWarning: Columns (34) have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[7]:
```
#Filter out attributes that will be used
df=df[['SEVERITYCODE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','INCDATES','ADDRTYPE','COLLISIONTYPE','JUNCTIONTYPE','INATTENTIONIND','UNDERINFL','WEATHER','ROADCOND','LIGHTCOND','SPEEDING']].copy()
df.head()
```

[7]:

	SEVERITYCODE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	INCDATES	ADDRTYPE	COLLISIONTYPE	JUNCTIONTYPE	INATTENTIONIND	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	SPEEDING
0	2	2	0	0	2	2013/03/27	Intersection	Angles	At Intersection (intersection related)	NaN	N	Overcast	Wet	Daylight	NaN
1	1	2	0	0	2	2006/12/20	Block	Sideswipe	Mid-Block (not related to intersection)	NaN	0	Raining	Wet	Dark - Street Lights On	NaN
2	1	4	0	0	3	2004/11/18	Block	Parked Car	Mid-Block (not related to intersection)	NaN	0	Overcast	Dry	Daylight	NaN
3	1	3	0	0	3	2013/03/29	Block	Other	Mid-Block (not related to intersection)	NaN	N	Clear	Dry	Daylight	NaN
4	2	2	0	0	2	2004/01/28	Intersection	Angles	At Intersection (intersection related)	NaN	0	Raining	Wet	Daylight	NaN

[8]:
```
df.shape
```

[8]:
```
(194673, 15)
```

Cleaning the Data

As part of cleaning the data, the following shall be done:

1) UNDERINFL seem to be a mix of Y & N and Boolean Values 0 & 1. Here We change all to Boolean
2) Some data is set to "unkown" which we will change to Nan (Python's default missing value marker)
3) We will need month and year in separate columns to perform some analysis on the data later
4) Change "INCDATES" to indicate either weekday or weekend
5) Dealing with missing Data (Drop Incidents with missing Data)
6) Drop incidents caused by inattention as this will cause noise to the relation of accidents with weather
7) Drop incidents caused by driving under the influence of alcohol or drugs as this will cause noise to the relation of accidents with weather
8) Drop incidents that have a missing information whether the driver was driving under the influence
9) Drop incidents related to speeding as this will cause noise to the relation of accidents with weather
10) Drop incidents that do not have WEATHER, ROADCOND, LIGHTCOND, ADDRTYPE or COLLISIONTYP indicated
11) We will not need the attributes INATTENTIONIND, UNDERINFL & SPEEDING. So we will drop them
12) Convert Categorical features to numerical value

```
[28]: df.tail(200)
```

[28]:

	SEVERITYCODE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	INCDATES	ADDRTYPE	COLLISIONTYPE	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND	INCMONTH	INCYEAR
194340	1	2	0	0	2	3	2	8	1	4	7	6	1	2019
194341	2	3	0	0	2	1	2	0	1	1	0	5	12	2018
194343	2	3	0	0	2	6	2	0	1	1	0	2	1	2019
194344	2	2	0	0	2	3	2	9	1	6	7	5	12	2018
194345	1	1	0	0	1	1	1	4	4	1	0	2	12	2018
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
194666	2	2	0	0	2	4	1	0	4	1	7	5	1	2019
194668	2	3	0	0	2	0	1	2	4	1	0	5	11	2018
194670	2	3	0	0	2	5	2	3	1	1	0	5	1	2019
194671	2	2	0	1	1	1	2	1	1	1	0	6	1	2019
194672	1	2	0	0	2	4	1	7	4	1	7	5	11	2018

200 rows × 14 columns

The numerical codes of the categorical variables are defines as the below dictonaries:

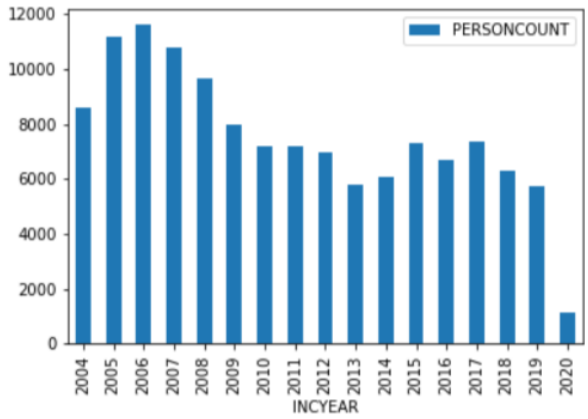
```
{0: 'At Intersection (but not related to intersection)', 1: 'At Intersection (intersection related)', 2: 'Driveway Junction', 3: 'Mid-Block (but intersection related)', 4: 'Mid-Block (not related to intersectio
n)', 5: 'Ramp Junction'}
{0: 'Blowing Sand/Dirt', 1: 'Clear', 2: 'Fog/Smog/Smoke', 3: 'Other', 4: 'Overcast', 5: 'Partly Cloudy', 6: 'Raining', 7: 'Severe Crosswind', 8: 'Sleet/Hail/Freezing Rain', 9: 'Snowing'}
{0: 'Dry', 1: 'Ice', 2: 'Oil', 3: 'Other', 4: 'Sand/Mud/Dirt', 5: 'Snow/Slush', 6: 'Standing Water', 7: 'Wet'}
{0: 'Dark - No Street Lights', 1: 'Dark - Street Lights Off', 2: 'Dark - Street Lights On', 3: 'Dark - Unknown Lighting', 4: 'Dawn', 5: 'Daylight', 6: 'Dusk', 7: 'Other'}
{0: 'Alley', 1: 'Block', 2: 'Intersection'}
{0: 'Angles', 1: 'Cycles', 2: 'Head On', 3: 'Left Turn', 4: 'Other', 5: 'Parked Car', 6: 'Pedestrian', 7: 'Rear Ended', 8: 'Right Turn', 9: 'Sideswipe'}
```

Data Visualization

The graph below shows the change in number of accidents per year from 2004 till 2020 (2020 Data is only till May)

```
[6]: df[["INCYEAR","PERSONCOUNT"]].groupby("INCYEAR").count().plot(kind='bar')
```

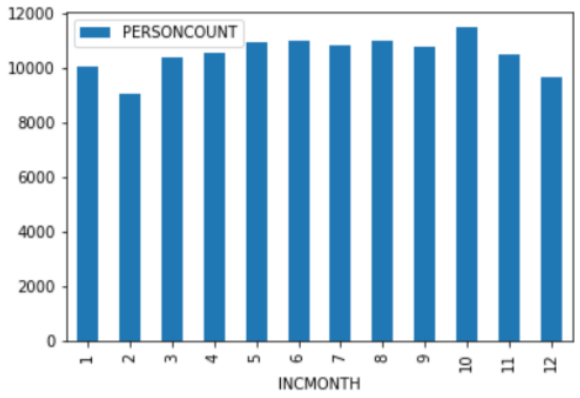
```
[6]: <AxesSubplot:xlabel='INCYEAR'>
```



The graph below shows the change in number of accidents over months of the year. Since 2020 data is not for a complete year we will exclude it in this part of the analysis

```
[7]: df2=df[df["INCYEAR"]!=2020]
     df2[["INCMONTH","PERSONCOUNT"]].groupby("INCMONTH").count().plot(kind='bar')
```

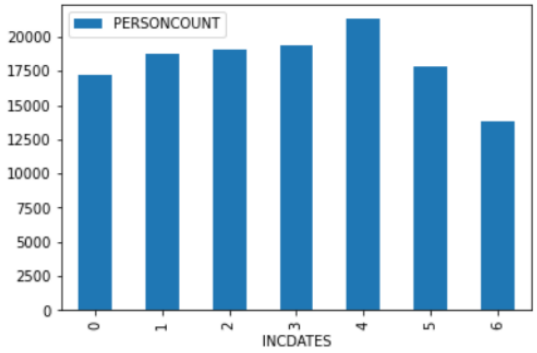
```
[7]: <AxesSubplot:xlabel='INCMONTH'>
```



The graph below shows the change in number of accidents over the days of the week

```
[8]: df[["INCDATES","PERSONCOUNT"]].groupby("INCDATES").count().plot(kind='bar')
```

```
[8]: <AxesSubplot:xlabel='INCDATES'>
```



Balancing the Data

```
[9]: df["SEVERITYCODE"].value_counts()
```

```
[9]: 1    86651
     2    40708
     Name: SEVERITYCODE, dtype: int64
```

As you can see, our data has more incidents of severity code 1 than code 2, we need to balance the data set.

[10]:
```
from sklearn.utils import resample
df_max =df[df.SEVERITYCODE ==1]
df_min =df[df.SEVERITYCODE ==2]
df_max_ds = resample(df_max,
                                replace = False,
                                n_samples= 40708,
                                random_state = 123
                               )

df_balance = pd.concat([df_max_ds,df_min])
df_balance.SEVERITYCODE.value_counts()
```

[10]:
```
2    40708
1    40708
Name: SEVERITYCODE, dtype: int64
```

Train-Test Split

Creating the X and Y features

[12]:
```
# We will define our independent variables in a feature we will call X
X=df_balance[["PERSONCOUNT","PEDCOUNT","PEDCYLCOUNT","VEHCOUNT","INCDATES","ADDRTYPE","COLLISIONTYPE","JUNCTIONTYPE","WEATHER","ROADCOND","LIGHTCOND"]].values
X[0:5]
```

[12]:
```
array([[2, 0, 0, 2, 2, 1, 0, 4, 1, 0, 5],
       [2, 0, 0, 2, 5, 1, 9, 4, 4, 0, 5],
       [2, 0, 0, 2, 2, 2, 0, 1, 1, 0, 5],
       [5, 0, 0, 2, 2, 1, 8, 2, 6, 7, 2],
       [3, 0, 0, 2, 1, 1, 7, 3, 1, 0, 5]])
```

[13]:
```
# We will define out dependent variables in a feature we will call Y
Y=df_balance[["SEVERITYCODE"]].values
Y[0:5]
```

[13]:
```
array([[1],
       [1],
       [1],
       [1],
       [1]])
```

Next, we will have to normalize the data

Normalize the data

[14]:
```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/utils/validation.py:5
  warnings.warn(msg, DataConversionWarning)
```

[14]:
```
array([[-0.37058857, -0.25089875, -0.2243384 ,  0.09771941, -0.47624075,
        -0.88225354, -1.3620328 ,  1.12570881, -0.7030964 , -0.61502854,
         0.57384267],
       [-0.37058857, -0.25089875, -0.2243384 ,  0.09771941,  1.09600259,
        -0.88225354,  1.64613961,  1.12570881,  0.74354951, -0.61502854,
         0.57384267],
       [-0.37058857, -0.25089875, -0.2243384 ,  0.09771941, -0.47624075,
         1.12404651, -1.3620328 , -1.02787323, -0.7030964 , -0.61502854,
         0.57384267],
       [ 1.72399924, -0.25089875, -0.2243384 ,  0.09771941, -0.47624075,
        -0.88225354,  1.31189823, -0.31001255,  1.70798012,  1.64034522,
        -1.55369302],
       [ 0.32760736, -0.25089875, -0.2243384 ,  0.09771941, -1.00032187,
        -0.88225354,  0.97765685,  0.40784813, -0.7030964 , -0.61502854,
         0.57384267]])
```

We will define 30 % of the data as testing anf 70% as training

[15]:
```
#Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.3, random_state=4)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (56991, 11) (56991, 1)
Test set: (24425, 11) (24425, 1)
```

Methodology

At this point we are ready to begin testing different classification algorithms. Our project is a typical classification problem as we attempt to learn the relationship between a set of feature variables and a target variable of interest. The aim is to find the best classification algorithm that will be able to predict the "SEVERITYCODE" of a certain incident given a defined X attributes with the highest accuracy. We will test k-nearest neighbor, Decision Tree, Support Vector Machine & Logistic Regression.

K-Nearest Neighbors

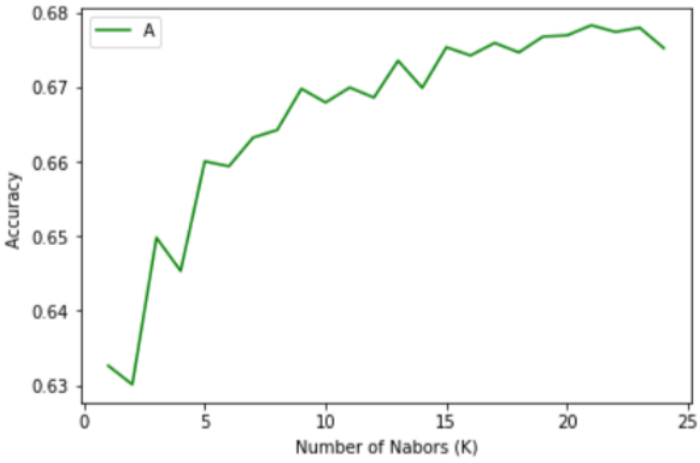
In the K-Nearest Neighbors, we need to analyze which value ok K gives us the best accuracy. We will perform the analysis on a range ok K values from 1 to 25 and see how the accuracy changes.

[16]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
Ks = 25
mean_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.ravel())
    Yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(Y_test, Yhat)

mean_acc
```

[16]:
```
array([0.6326305 , 0.63009212, 0.64978506, 0.64532242, 0.66002047,
       0.6593654 , 0.66321392, 0.66423746, 0.66976459, 0.66792221,
       0.66992835, 0.66857728, 0.67353122, 0.66988741, 0.67533265,
       0.67422723, 0.67590583, 0.67463664, 0.67676561, 0.67692938,
       0.67828045, 0.67737973, 0.67795292, 0.67520983])
```

Plotting the accuracy vs the K values

[17]:
```
plt.plot(range(1,Ks),mean_acc,'g')
plt.legend('Accuracy')
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 0.6782804503582395 with k= 21
```

We can see that the best accuracy is with a K value of 21. Therefore, we will build the model using K value of 21

[18]:
```
k =21
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,Y_train.ravel())
yhat = neigh.predict(X_test)
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(Y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(Y_test, Yhat))
```

```
Train set Accuracy:  0.6987243599866646
Test set Accuracy:  0.6752098259979529
```

KNN Model Evaluation:

[19]:
```
from sklearn.metrics import jaccard_similarity_score
jaccard_KNN=jaccard_similarity_score(Y_test, Yhat)
jaccard_KNN
```

[19]:
```
0.6752098259979529
```

[20]:
```
from sklearn.metrics import f1_score
f1_KNN=f1_score(Y_test, Yhat, average='weighted')
f1_KNN
```

[20]:
```
0.6752108560431431
```

Decision Tree

[21]:
```
from sklearn.tree import DecisionTreeClassifier
AccidentTree=DecisionTreeClassifier(criterion="entropy", max_depth = 4)
AccidentTree
```

[21]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            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=False, random_state=None,
            splitter='best')
```

[22]:
```
AccidentTree.fit(X_train,Y_train)
```

[22]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            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=False, random_state=None,
            splitter='best')
```

[23]:
```
predAccident= AccidentTree.predict(X_test)
```

[24]:
```
from sklearn import metrics
print("AccidentTrees's Accuracy: ", metrics.accuracy_score(Y_test, predAccident))
```

```
AccidentTrees's Accuracy:  0.6893756397134084
```

Decision Tree Model Evaluation:

```
[25]: from sklearn.metrics import jaccard_similarity_score
      jaccard_DT=jaccard_similarity_score(Y_test, predAccident)
      jaccard_DT

[25]: 0.6893756397134084

[26]: from sklearn.metrics import f1_score
      f1_DT=f1_score(Y_test, predAccident, average='weighted')
      f1_DT

[26]: 0.6843349221458243
```

Support Vector Machine

```
[27]: from sklearn import svm
      clf = svm.SVC(kernel='rbf')
      clf.fit(X_train, Y_train.ravel())

      /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/s
      eatures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning
        "avoid this warning.", FutureWarning)

[27]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
        kernel='rbf', max_iter=-1, probability=False, random_state=None,
        shrinking=True, tol=0.001, verbose=False)

[28]: YhatSVM = clf.predict(X_test)
```

Support Vector Machine Model Evaluation:

SVM Model Evaulation

```
[29]: jaccard_SVM=jaccard_similarity_score(Y_test, YhatSVM)
      jaccard_SVM

[29]: 0.6738587512794268

[30]: f1_SVM=f1_score(Y_test, YhatSVM, average='weighted')
      f1_SVM

[30]: 0.6738235318096121
```

Logistic Regression

Since our target variable is of a binary nature, it would be perfect for logistic regression.

[31]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,Y_train.ravel())
Yhat = LR.predict(X_test)
Yhat
```

[31]:
```
array([1, 2, 1, ..., 1, 2, 1])
```

[32]:
```
Yhat_prob = LR.predict_proba(X_test)
```

Logistic Regression Model Evaluation:

Logistic Regression Model Evaulation

[33]:
```
jaccard_LR=jaccard_similarity_score(Y_test, Yhat)
jaccard_LR
```

[33]:
```
0.6612896622313204
```

[34]:
```
f1_LR=f1_score(Y_test, Yhat, average='weighted')
f1_LR
```

[34]:
```
0.6603246096341822
```

[35]:
```
from sklearn.metrics import log_loss
log_loss_LR=log_loss(Y_test, Yhat_prob)
log_loss_LR
```

[35]:
```
0.6023200134410428
```

Models Comparison

[36]:
```
jaccard_Score=[jaccard_KNN,jaccard_DT,jaccard_SVM,jaccard_LR]
f1_score=[f1_KNN,f1_DT,f1_SVM,f1_LR]
Log_loss=["NA","NA","NA",log_loss_LR]
df_Report=pd.DataFrame(jaccard_Score,index=["KNN","Decision Tree","SVM","Logistic Regression"])
df_Report.columns=["Jaccard"]
df_Report.insert(loc=1,column="F1-Score",value=f1_score)
df_Report.insert(loc=2,column="Log Loss",value=Log_loss)
df_Report.columns.name='Algorithm'
df_Report
```

[36]:

Algorithm	Jaccard	F1-Score	Log Loss
KNN	0.675210	0.675211	NA
Decision Tree	0.689376	0.684335	NA
SVM	0.673859	0.673824	NA
Logistic Regression	0.661290	0.660325	0.60232

Discussion & Conclusion

Based on the evaluation metric results listed above, we can say that all 4 algorithms have resulted in almost a similar result. The best accuracy is with the Decision Tree algorithm which has an accuracy percentage of about 69%. Although, we can confidently say that our model with the decision tree can predict whether an accident will result in a severity code 1 or 2, the accuracy is not that high. Our model seems to be underfitted. Since we have many attributes which contribute to the severity of the accident, we can say that collecting more data of more accidents will help increase the accuracy of the model. We have lost quite a number of data in the cleaning process which was mainly due to some attributes being missed in the collection of the data.