# **IBM Data Science Capstone: Accident Severity Report**

**Introduction | Business Undertanding**

There is a huge impact on the society due to traffic accidents where there is a great costs of fatalities and injuries. In recent years, there is a increase in the researches attention to determine the significantly affect the severity of the drivers injuries which is caused due to the road accidents. Accurate and comprehensive accident records are the basis of accident analysis. the effective use of accident records depends on some factors, like the accuracy of the data, record retention, and data analysis. There is many approaches applied to this scenario to study this problem.

A recent study illustrated that the residential and shopping sites are more hazardous than village areas.as might have been predicted , the frequencies of the casualties were higher near the zones of residence possibly because of the higher exposure.A study revealed that the casualty rates among the residential areas are classified as relatively deprived and significantly higher than those from relatively affluent areas.

**Data Understanding**

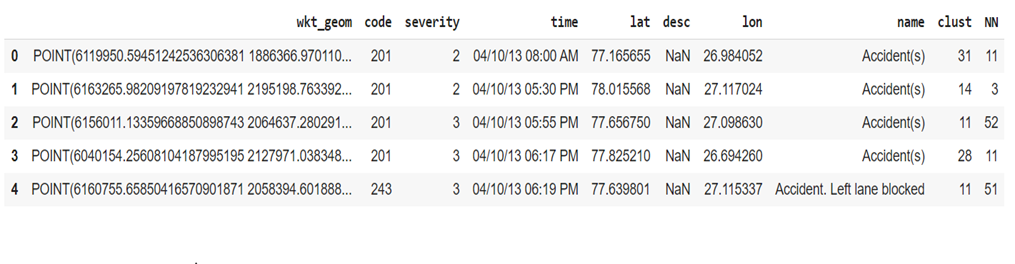
Our predictor or target variable will be 'SEVERITYCODE' because it is used measure the severity of an accident from 0 to 5 within the dataset. Attributes used to weigh the severity of an accident are 'Latitude', 'longitude' and 'time'.

Severity codes are as follows:

* 0:Little to no Probability (Clear Conditions)
* 1 : Very Low Probablility - Chance or Property Damage
* 2 : Low Probability - Chance of Injury
* 3 : Mild Probability - Chance of Serious Injury
* 4 : High Probability - Chance of Fatality

### **Extract Dataset & Convert**

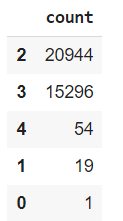
In it's original form, this data is not fit for analysis. For one, there are many columns that we will not use for this model. Also, most of the features are of type object, when they should be numerical type.



Balancing the Dataset

Our target variable SEVERITYCODE is unbalaced . In fact, severitycode in class 4,1,0 is nearly negligible as compare to 2 and 3.

We can fix this by downsampling the minority class



With the new columns, we can now use this data in our analysis and ML models!

Now let's check the data types of the new columns in our dataframe. Moving forward, we will only use the new columns for our analysis.

Preprocessing, covert the time format to months, weekdays, years, hours

1. df["month"] = df["time"].apply(**lambda** x:int(x[:2]))
2. df["day"] = df["time"].apply(**lambda** x:int(x[3:5]))
3. df["year"] = df["time"].apply(**lambda** x:int(x[6:8]))
4. df["hour"] =  df["time"].apply(**lambda** x: int(x[9:11]) **if** str(x)[15] == 'A' **else** 12 + int(x[9:11])  )
5. df["lon"] = df["lon"].apply(**lambda** x:abs(x)) #so that multinomialNB works (only with positive features)
6. #creating the date at the datetime format (easier to deal with)
7. df[ "date" ]= df[["month" , "day" ,"year"]].apply(**lambda** x:pd.datetime(month = x['month'] , day = x['day']  , year = 2000+x["year"]), axis = 1)

df["weekday"] =  df["date"].apply(**lambda** x:x.weekday())  Error’s and Outlier removing

1. #here we check for outlier and boundaries
2. pd.DataFrame( {"count": df["severity"].value\_counts().values } , index = df["severity"].value\_counts().index )
3. df = df.loc[df["severity"] >  1].loc[df["severity"] < 4]

Distributions of features and labels

1. #plotting the dataset with a different color depending on the severity (2,3)
2. #longitude vs lattitude
4. df2 = df.loc[df["severity"] == 2]
5. df3 = df.loc[df["severity"] == 3]
7. xx2 , yy2 = df2["lat"] , -df2["lon"]
8. xx3 , yy3 = df3["lat"] , -df3["lon"]
10. pts2 = plt.scatter(xx2,yy2,color = 'b' )
11. pts3 = plt.scatter(xx3,yy3,color = 'r' )
12. plt.legend((pts2, pts3), ('Severity = 2', 'Severity = 3'),loc='lower left')
13. plt.title("Accident Severity Map")
14. plt.tight\_layout

Applying Cross Validation and Splitting the data into training set and test sets , ratio of 2:8

1. X = df[["month" , "hour" , "year", "weekday" ,"lon" , "lat"]]
2. y = df["severity"].apply(**lambda** x:x-2) # shifting to 0-1 values instead of 2-3
4. #here we assign test size as 20% of actual data set
5. # random state is set as 42 ( or 1 ) also from the reference of " The Hitchhiker's Guide to the Galaxy"
6. # we defined random state to get consistent and same results , regardless of the training iterations
7. # so that the values in the train and test sets are homogenous
9. **from** sklearn.cross\_validation **import** train\_test\_split
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

## **Methodology**

Our data is now ready to be fed into machine learning models.

We will use the following models:

**K-Nearest Neighbor (KNN)**

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

**Decision Tree**

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

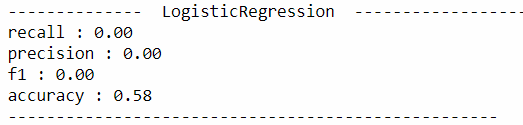
**Logistic Regression**

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

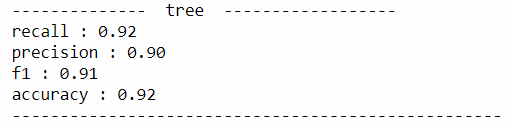
1. #logreg (it predicts everything to 0, the most common class)
2. # a predictive analysis , where we comapre a relation between binary varaible
3. # and other ordinal and nominal and indepdendent variables
4. #also here we are not using any paramneter's
5. #use from training set and test sets
6. #for cross valiadtion

9. **from** sklearn.linear\_model **import** LogisticRegression
10. clf = LogisticRegression()
11. clf.fit(X\_train,y\_train)
12. y\_pred = pd.Series(clf.predict(X\_test))
13. printScores(y\_test, y\_pred, "LogisticRegression")



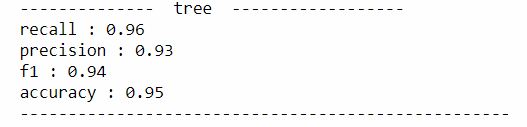
Tree Algorithm

1. #tree
3. #structured tree where it containts root nodes , leaf node
4. #each brach of the tre depicts outcome of the tree
6. # use from training set and test sets
7. #for cross valiadtion
9. **from** sklearn **import** tree
10. clf = tree.DecisionTreeClassifier()
11. clf.fit(X\_train,y\_train)
12. y\_pred = clf.predict(X\_test)
13. printScores(y\_test, y\_pred, "tree")



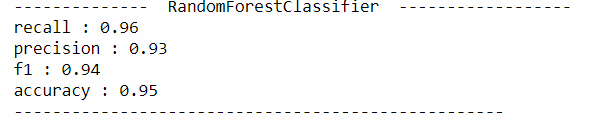
Tree algorithm (modified training set)

1. #tree
2. #MODIFIED TRAINING SET of lat and longitude
3. #this one is parametrized
4. # use from training set and test sets , only 2 variables , lat and lon
5. #for cross validation
7. X\_train2 , X\_test2 = X\_train[["lat" , "lon" ]]  , X\_test[["lat" , "lon" ]]
8. **from** sklearn **import** tree
9. clf = tree.DecisionTreeClassifier()
10. clf.fit(X\_train2,y\_train)
11. y\_pred = clf.predict(X\_test2)
12. printScores(y\_test, y\_pred, "tree")



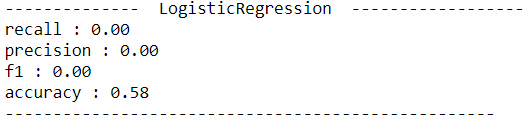
Random Forest Algorithm (modified training set)

1. #random forest
3. # use from training set and test sets
4. # for cross valiadtion
5. #n\_estiamotor is 100 , as default from update of 2.10
7. **from** sklearn.ensemble **import** RandomForestClassifier
8. clf = RandomForestClassifier(n\_estimators = 100)
9. clf.fit(X\_train2,y\_train)
10. y\_pred = clf.predict(X\_test2)
11. printScores(y\_test, y\_pred, "RandomForestClassifier")



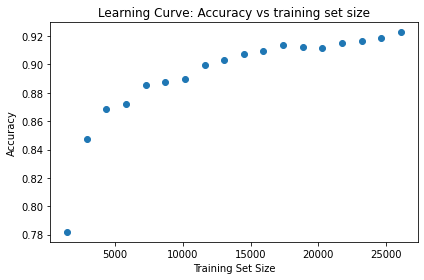
Logistic Regression (Modified training Set)

1. #logreg (it predicts everything to 0, the most common class)
3. # use from training set and test sets
4. #for cross valiadtion
6. **from** sklearn.linear\_model **import** LogisticRegression
7. clf = LogisticRegression()
8. clf.fit(X\_train2,y\_train)
9. y\_pred = pd.Series(clf.predict(X\_test2))
10. printScores(y\_test, y\_pred, "LogisticRegression")



Training set graph prediction

1. #drawing the prediction graph
3. training\_set\_size = [0.05\*i **for** i **in** range(1,19)]
4. accuracy = []
5. **from** sklearn **import** tree
6. **for** size **in** training\_set\_size:
7. # won't be using the test in that case...this is just a way of splitting the data
8. X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X\_train, y\_train, test\_size=1-size, random\_state=42)
9. clf = tree.DecisionTreeClassifier()
10. clf.fit(X\_train2,y\_train2)
11. y\_pred = clf.predict(X\_test)
12. #printScores(y\_test, y\_pred, "tree")
13. accuracy.append(  (X\_train2.shape[0] ,accuracy\_score(y\_test, y\_pred) )  )
15. xx = [w[0] **for** w **in** accuracy]
16. yy = [w[1] **for** w **in** accuracy]
17. plt.scatter(xx,yy)
18. plt.xlabel('Training Set Size')
19. plt.ylabel('Accuracy')
20. plt.title("Learning Curve: Accuracy vs training set size")
21. plt.tight\_layout()



## **Results & Evaluation**

Now we will check the accuracy of our models.

By evaluating all models , tree model have the highest accuracy among all.

## **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 algoritim, so label encoding was used to created new classes that were of type int8; a numerical data type.

After solving that issue we were presented with another - imbalanced data. As mentioned earlier, class 1,0,4 are very less as compare to 2,3. The solution to this was downsampling the minority class with sklearn's resample tool. We downsampled to match the match the class and finally created our model with only two predicted variable 2,3.