

IBM Data Science Professional Certificate

Course 9: Applied Capstone Project

A Project Report on:

Accident severity prediction using the classification algorithm for Seattle city

Student Name: Shivpalsinh Rana
Submitted on: 25.09.2020

Table of Content

Table of Content	I
1 Introduction	1
1.1 Business Understanding	1
2 Data Understanding	2
2.1 Data Source.....	2
2.2 Data Description	2
3 Data Cleaning and Preparation	3
3.1 Data Cleaning	3
3.1.1 Removing columns with a higher number of missing values	3
3.1.2 Removing irrelevant columns	3
3.1.3 Removing repeating columns and inconsistent data	4
3.2 Feature / attribute selection.....	4
3.3 Data Formatting	4
3.3.1 One Hot Encoding.....	5
3.3.2 Balancing the Data	6
3.4 Exploratory Data Analysis.....	6
3.5 Data Normalization.....	7
3.5.1 Splitting the Data.....	8
3.5.2 Data standardization	8
4 Modeling	9
4.1 KNN Model	9
4.2 Decision Tree Model	10
4.3 SVM Model	10
4.4 Logistic Regression Model.....	10
5 Evaluation and Conclusion	12
5.1 Evaluation and Accuracy	12
5.1.1 Accuracy Scores:.....	12
5.1.2 Jaccard Similarity Score:.....	12
5.1.3 F1-Score:	12
5.2 Results and Conclusion	13
5.3 Future Scopes	13

1 Introduction

In this section, we will discuss the business understanding to seek clarification for the problem.

1.1 Business Understanding

Automobiles are the most popular means of transport in the USA. On daily basis, hundreds of thousands of people travel from A to B using road infrastructure. One of the less fortunate side or aspect of road transportation is road accidents. Road accidents not only cause injuries and deaths but also results in loss or damage of properties worth millions of Dollars. Accidents also hinder the traffic flow which results in loss of time for many commuters. Unnecessary congestion and stop and go traffic built-up cause the vehicle to run inefficiently and emit more pollutants.

There are several factors that influence the occurrence of road accidents. Some of the factors are weather conditions, road conditions, attention of the driver, condition of the driver, light conditions, etc. In this project, we try to predict the severity of the accident given the road condition, weather condition, type of accident, and many more influencing factors. To effectively predict the severity, we will be using the machine learning algorithms.

The following stack holders can be benefited from an effective machine learning model which can predict the accident severity:

1. People who are commuters and would like to plan their travel. For example, in a situation where weather is windy or rainy, the road conditions are bad and light condition is also not ideal, Commuter can obtain the severity of an accident and can avoid the traffic congestion and drive more cautiously.
2. Secondly, the government regulatory bodies can take precautionary steps, to avoid or to reduce the fatality of an accident.

In the following section, we will be looking for the data which can be used to train the machine learning model and help us to solve the problem.

2 Data Understanding

Here, we will discuss the data set selected for this project and investigate the data.

2.1 Data Source

The data set used for the project is procured from the Seattle Department of Transport and consists of the vehicle accident information. The data set is in .csv format with 38 columns and a total of 194673 entries or rows. Overall, the data seems to be rich, has many of the observations and the attributes which are suitable for training a machine learning model.

2.2 Data Description

The label for the selected .csv data set is the column 'SEVERITYCODE'. Each row in the data set has given a severity code which is either '1' or '2' meaning 'property damage only collision' and 'injury collision', respectively. The rest 37 columns contain various attributes in which not all of them are useful.

By examining the number of 1s and 2s in the label of the given data set, we see that the data is having an imbalance i.e. there are more rows with 1s compared to 2s. Such unbalanced data can result in the bias behavior of the machine learning model.

As we have observed, there are many irrelevant attributes in the data set which are not useful in training a machine learning model. Moreover, like most of the data sets available, there are some columns with missing values or inconsistent values. Thus, the data must be cleaned in such a way that it avoids the irrelevant attributes and increase the accuracy of the machine learning model without any bias.

Initially, we convert the data into the Pandas data frame.

3 Data Cleaning and Preparation

Before feeding the data to a machine learning model, it should be cleaned or wrangled to avoid problems and false predictions. Unnecessary and irrelevant attributes must be dropped and should not be used to train the model. Moreover, most machine learning model does not work well with the categorical variables so such variable must be encoded to numerical variables.

In this section, we will discuss the steps and methodology involved in the process of data cleaning and preparation in detail.

3.1 Data Cleaning

The data cleaning or wrangling has been carried out in the following steps:

3.1.1 Removing columns with a higher number of missing values

By executing `df.isnull().sum()` method on the data frame `df`, we the total number of missing values in each of the columns (see figure on left).

df.isnull().sum()	
SEVERITYCODE	0
X	5334
Y	5334
OBJECTID	0
INCKEY	0
COLDKEY	0
REPORTNO	0
STATUS	0
ADDRTYPE	1926
INTKEY	129603
LOCATION	2677
EXCEPTSNCODE	109862
EXCEPTSNDESC	189035
SEVERITYCODE.1	0
SEVERITYDESC	0
COLLISIONTYPE	4904
PERSONCOUNT	0
PEDCOUNT	0
PEDCYLCOUNT	0
VEHCOUNT	0
INCDATE	0
INCDTTM	0
JUNCTIONTYPE	6329
SDOT_COLCODE	0
SDOT_COLDESC	0
INATTENTIONIND	164868
UNDERINFL	4884
WEATHER	5081
ROADCOND	5012
LIGHTCOND	5170
PEDROWNOTGRNT	190006
SDOTCOLNUM	79737
SPEEDING	185340
ST_COLCODE	18
ST_COLDESC	4904
SEGLANEKEY	0
CROSSWALKKEY	0
HITPARKEDCAR	0
dtype: int64	

Here we can see that columns like 'INTKEY', 'EXCEPTSNCODE', 'INATTENTIONIND' etc. has a very large number of missing values. Such columns do not add any value in training a machine learning model. Thus, these columns or attributes are dropped from the data frame.

We use the following line of code to drop 7 columns with a higher number of mission values.

```
df = df.drop(['INTKEY', 'EXCEPTSNCODE', 'EXCEPTSNDESC', 'SPEEDING', 'SDOTCOLNUM', 'PEDROWNOTGRNT', 'INATTENTIONIND'], axis=1)
```

3.1.2 Removing irrelevant columns

The data frame has many columns that cannot help us in predicting the severity of an accident. The columns containing various IDs, Codes, or Keys such as 'INTKEY', 'OBJECTID', 'REPORTNO' etc. adds no value in training a machine learning model.

We eliminate the irrelevant attributes as follows:

```
df=df.drop(['X','Y','OBJECTID','INCKEY','LOCATION','COLDETKEY','REPORTNO','STATUS','SEVERITYDESC','INCDATE','SDOT_COLCODE','SDOT_COLDESC','ST_COLCODE','ST_COLDESC','SEGLANEKEY','CROSSWALKKEY','INCDTTM'],axis=1)
```

3.1.3 Removing repeating columns and inconsistent data

Firstly, we remove the column ‘SEVERITYCODE.1’ because it is repeated. Then the column ‘UNDERINFL’ is dropped as it contains data in an inconsistent manner. Some of the entries are in 0s and 1s while others are in Ns and Ys.

3.2 Feature / attribute selection

After cleaning the data, we are left with 12 columns which will be our attributes for training the machine learning model. The following figure shows the first five entries of the remaining data set.

SEVERITYCODE	ADDRTYPE	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	JUNCTIONTYPE	WEATHER	ROADCOND	LIGHTCOND	HITPARKEDCAR
2	Intersection	Angles	2	0	0	2	At Intersection (intersection related)	Overcast	Wet	Daylight	N
1	Block	Sideswipe	2	0	0	2	Mid-Block (not related to intersection)	Raining	Wet	Dark - Street Lights On	N
1	Block	Parked Car	4	0	0	3	Mid-Block (not related to intersection)	Overcast	Dry	Daylight	N
1	Block	Other	3	0	0	3	Mid-Block (not related to intersection)	Clear	Dry	Daylight	N
2	Intersection	Angles	2	0	0	2	At Intersection (intersection related)	Raining	Wet	Daylight	N

As discussed in the introduction part, the column ‘SEVERITYCODE’ is the dependent variable and the rest of the variables are independent. Furthermore, the columns ‘PERSONCOUNT’, ‘PEDCOUNT’, ‘PEDCYLCOUNT’, and ‘VEHCOUNT’ have numeric values and the rest of the columns has categorical values. Before formatting and standardization of the data, we investigate the unique values in each of the columns. An example of the column ‘WEATHER’ is shown in the figure below.

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

```
Clear          111135
Raining        33145
Overcast       27714
Unknown        15091
Snowing         907
Other           832
Fog/Smog/Smoke  569
Sleet/Hail/Freezing Rain  113
Blowing Sand/Dirt  56
Severe Crosswind  25
Partly Cloudy   5
Name: WEATHER, dtype: int64
```

In this figure, we can see that many entries have some irrelevant information such as ‘Unknown’ or ‘Other’. Such attributes do not describe a particular situation or condition thus these values must be dropped.

The values ‘Unknown’ and ‘Other’ are first converted into NaNs (Not-a-number) values and then the rows with NaN are dropped using the *df.dropna()* method.

3.3 Data Formatting

After the data is cleaned, the categorical values and numerical values are separated into two different variables to convert only the categorical attributes. After that, the unbalance in the data set must be corrected.

3.3.1 One Hot Encoding

One Hot Encoding is a method in which the categorical values are automatically converted into numeric values. This is done by using `pd.get_dummies` method of the Pandas library. The original columns are automatically omitted. As the next step, we can merge the resulted columns with the columns containing the numerical values.

```
df_dummies = pd.get_dummies(df[["ADDRTYPE", "COLLISIONTYPE", "JUNCTIONTYPE", "WEATHER", "ROADCOND", "LIGHTCOND", "HITPARKEDCAR"]])
```

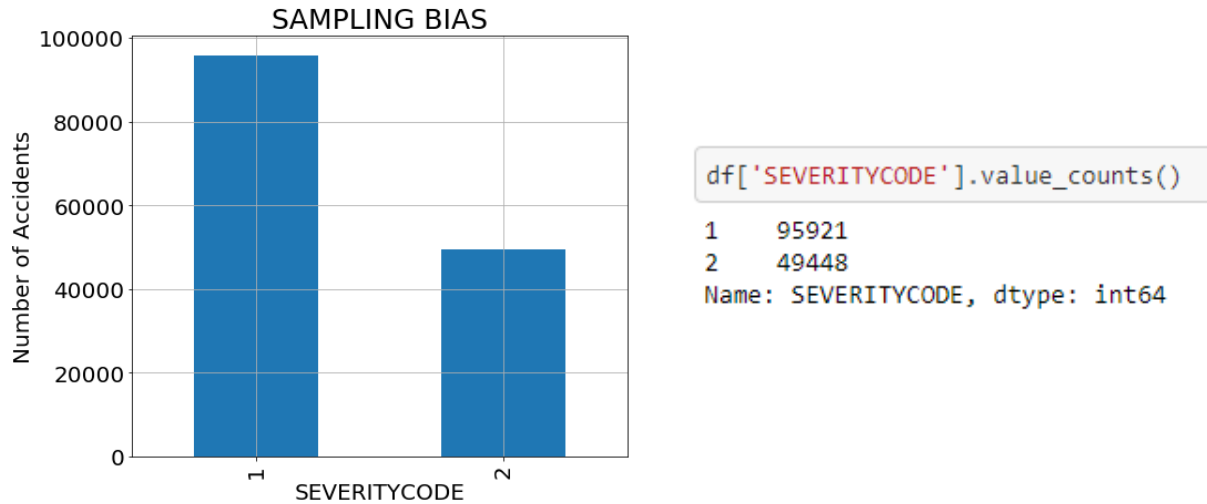
The total number of an independent variable resulted after one hot encoding are as shown in the screenshot below:

```
Pre_X.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 145369 entries, 0 to 194672
Data columns (total 47 columns):
#   Column                                                                 Non-Null Count  Dtype
---  -
0   PERSONCOUNT                                                            145369 non-null  int64
1   PEDCOUNT                                                                145369 non-null  int64
2   PEDCYLCOUNT                                                             145369 non-null  int64
3   VEHCOUNT                                                                145369 non-null  int64
4   ADDRTYPE_Alley                                                           145369 non-null  uint8
5   ADDRTYPE_Block                                                           145369 non-null  uint8
6   ADDRTYPE_Intersection                                                    145369 non-null  uint8
7   COLLISIONTYPE_Angles                                                     145369 non-null  uint8
8   COLLISIONTYPE_Cycles                                                     145369 non-null  uint8
9   COLLISIONTYPE_Head On                                                    145369 non-null  uint8
10  COLLISIONTYPE_Left Turn                                                  145369 non-null  uint8
11  COLLISIONTYPE_Parked Car                                                 145369 non-null  uint8
12  COLLISIONTYPE_Pedestrian                                                 145369 non-null  uint8
13  COLLISIONTYPE_Rear Ended                                                 145369 non-null  uint8
14  COLLISIONTYPE_Right Turn                                                 145369 non-null  uint8
15  COLLISIONTYPE_Sideswipe                                                  145369 non-null  uint8
16  JUNCTIONTYPE_At Intersection (but not related to intersection)          145369 non-null  uint8
17  JUNCTIONTYPE_At Intersection (intersection related)                     145369 non-null  uint8
18  JUNCTIONTYPE_Driveway Junction                                           145369 non-null  uint8
19  JUNCTIONTYPE_Mid-Block (but intersection related)                       145369 non-null  uint8
20  JUNCTIONTYPE_Mid-Block (not related to intersection)                   145369 non-null  uint8
21  JUNCTIONTYPE_Ramp Junction                                               145369 non-null  uint8
22  WEATHER_Blowing Sand/Dirt                                                145369 non-null  uint8
23  WEATHER_Clear                                                            145369 non-null  uint8
24  WEATHER_Fog/Smog/Smoke                                                  145369 non-null  uint8
25  WEATHER_Overcast                                                         145369 non-null  uint8
26  WEATHER_Partly Cloudy                                                    145369 non-null  uint8
27  WEATHER_Raining                                                         145369 non-null  uint8
28  WEATHER_Severe Crosswind                                                 145369 non-null  uint8
29  WEATHER_Sleet/Hail/Freezing Rain                                        145369 non-null  uint8
30  WEATHER_Snowing                                                         145369 non-null  uint8
31  ROADCOND_Dry                                                            145369 non-null  uint8
32  ROADCOND_Ice                                                            145369 non-null  uint8
33  ROADCOND_Oil                                                            145369 non-null  uint8
34  ROADCOND_Sand/Mud/Dirt                                                  145369 non-null  uint8
35  ROADCOND_Snow/Slush                                                     145369 non-null  uint8
36  ROADCOND_Standing Water                                                 145369 non-null  uint8
37  ROADCOND_Wet                                                            145369 non-null  uint8
38  LIGHTCOND_Dark - No Street Lights                                       145369 non-null  uint8
39  LIGHTCOND_Dark - Street Lights Off                                     145369 non-null  uint8
40  LIGHTCOND_Dark - Street Lights On                                       145369 non-null  uint8
41  LIGHTCOND_Dark - Unknown Lighting                                       145369 non-null  uint8
42  LIGHTCOND_Dawn                                                           145369 non-null  uint8
43  LIGHTCOND_Daylight                                                       145369 non-null  uint8
44  LIGHTCOND_Dusk                                                           145369 non-null  uint8
45  HITPARKEDCAR_N                                                           145369 non-null  uint8
46  HITPARKEDCAR_Y                                                           145369 non-null  uint8
dtypes: int64(4), uint8(43)
memory usage: 11.5 MB
```

3.3.2 Balancing the Data

As we discussed earlier, our data set has an unbalance in the labelled attribute 'SEVERITYCODE', which can result in a bias while predicting the unknown values. The balance in the data is achieved by making the number 1s the same as 2s.



The balance is achieved by using the NearMiss package of imblearn library. By using the following command, the balance is implemented.

```
from imblearn.under_sampling import NearMiss
```

```
nm = NearMiss()
X, y = nm.fit_sample(Pre_X, Pre_y)
```

Firstly, we create NearMiss() object, and then fit our Pre_X (independent) and Pre_Y (dependent) variable. The NearMiss() will randomly shuffle the data and the .fit_sample() method will delete the remaining rows such that no. of 1s is equal to no. 2s.

```
y.value_counts()
2    49448
1    49448
Name: SEVERITYCODE, dtype: int64
```

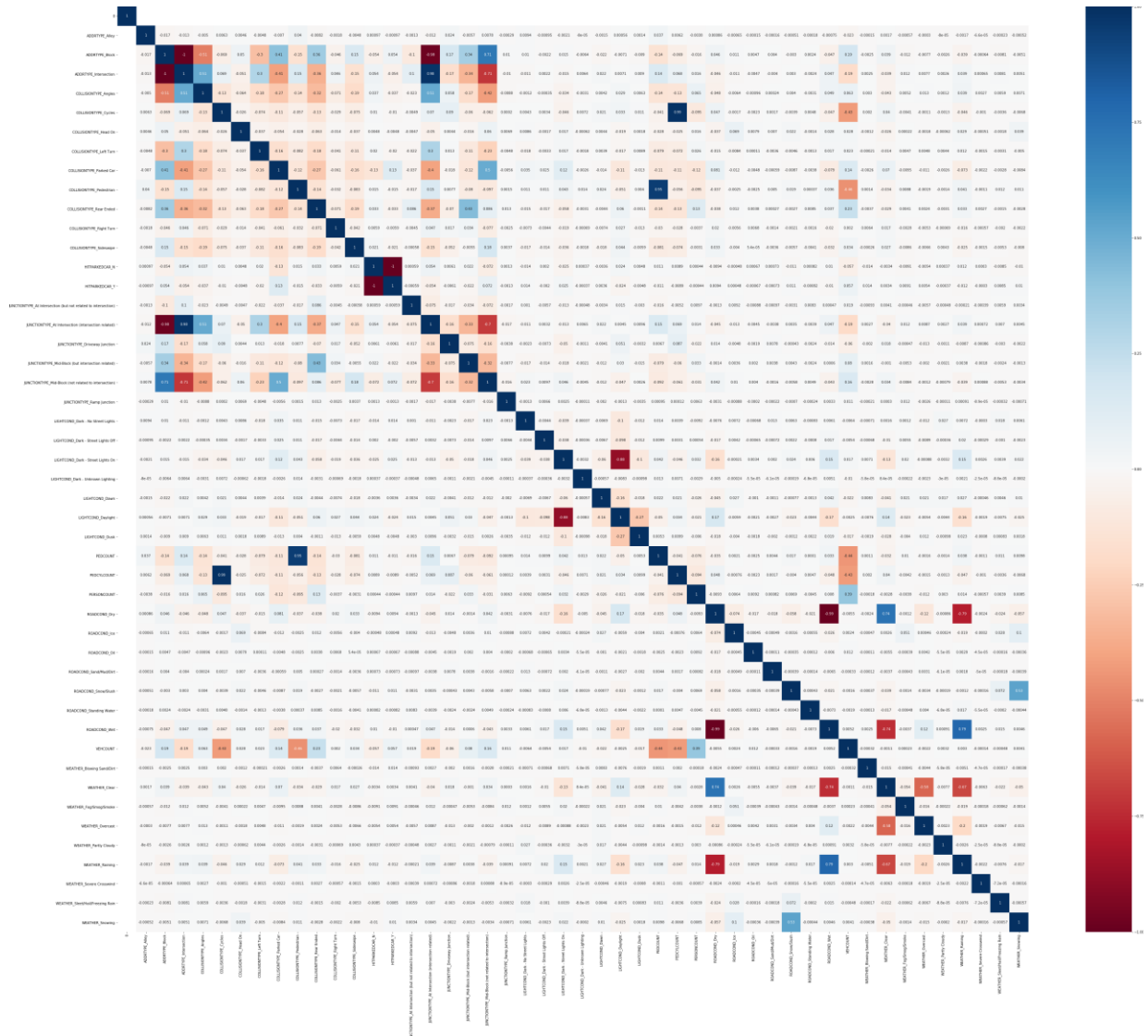
The output will be two data frames X and y which we will use to create our training set and testing set.

3.4 Exploratory Data Analysis

Before creating and splitting the data, we can explore our clean and balanced data for the correlation between different variables. We can first concatenate the data frame X and y and then apply .corr() method to get a grid of Pearson's correlation values between all the attributes. This displays values between 1 and -1. If the value is near zero, it means that the

correlation is weak. Values close to 1 and -1 shows a strong positive or negative correlation, respectively.

This correlation can be visualized into a heat map using `.heatmap()` method of seaborn library. Please visit the [link](#) to see the heat map in the original resolution.



From the correlation matrix, we can see a strong positive correlation between the ‘ROADCOND_Wet’ and ‘WEATHERCOND_Rainy’, ‘ADDRTYPE_Intersection’, and ‘JUNCTIONTYPE_At Intersection’, and ‘PEDCOUNT’ and ‘COLLISIONTYPE_Pedestrian’. Moreover, a strong negative correlation is observed between the attributes like ‘WEATHER_Raining’ and ‘ROADCOND_Dry’ as the given condition is contradictory. This correlation matrix and visualizations help us to understand the extent of the interdependencies between the variables in a simple manner.

3.5 Data Normalization

The data should be standardized to reduce the error in the prediction because of large values in the columns. The standardization only applies to the column with the numerical value, not

to the already encoded categorical values. But before carrying out data standardization, the data set must be split into testing and training data set.

3.5.1 Splitting the Data

The splitting of training and testing data is done using `.train_test_split()` method of the sci-kit-learn library. By running the following lines of code below, the data will be split in such a way that 80% of the data becomes the training data, written in the variable `X_train` and `y_train`, and rest 20% will be our testing data represented in variable `X_test` and `y_test`. The figure below also shows the dimensions of our resulted data sets.

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)

Train set: (79116, 47) (79116,)
Test set: (19780, 47) (19780,)
```

3.5.2 Data standardization

To increase the accuracy and effectiveness of our machine learning model, the numeric columns with many distinct values should be normalized in such a way that they produce zero mean. We can use `StandardScaler()` from the preprocessing module of the Scikit Learn library. First, we create a `StandardScaler()` object and then apply `.fit_transform` method on the numerical columns of our testing and training data set separately. The implemented code is shown in the image below.

```
scaler = StandardScaler()
X_train[['PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT']] = scaler.fit_transform(X_train[['PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT']])

scaler = StandardScaler()
X_test[['PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT']] = scaler.fit_transform(X_test[['PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT']])
```

4 Modeling

As discussed in the introductory part of the report, our goal is to predict the severity of an accident in binary classes i.e. the severity could be either 1 or 2. This clearly shows that this problem can be solved by using Classification machine learning algorithms as the target attribute is a categorical variable. Classification is a supervised learning approach in which the label of our data is categorized into a discrete set of categories or classes. Classification determines a class label for an unlabeled case.

For this problem, four different classification models will be trained and tested with or training and testing data set, respectively. We will discuss all the implemented machine classifiers with a short explanation in the coming sections.

4.1 KNN Model

k-nearest neighbor is an algorithm that takes several labeled points and uses them to label other points based on the similarity to other cases. The cases that are near to each other are called neighbors. Initially, a value of $k = 4$ is assumed and the distance of unknown cases is calculated. Then k-observations in the training data that are nearest to the unknown data point is selected. Afterward, the response of the unknown data point using the most popular response value from the k-nearest neighbors is predicted. The code below shows how the model data is fitted with the training data and the first 5 predictions from the testing data set.

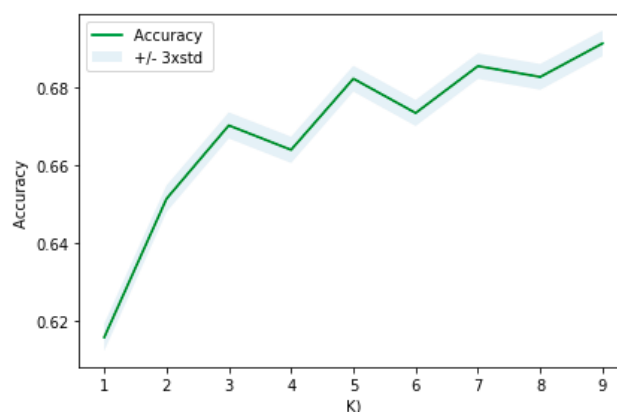
```
k = 4
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh

KNeighborsClassifier(n_neighbors=7)

yhat_knn = neigh.predict(X_test)
yhat_knn[0:5]

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

Now, to calculate the optimum value of k , we can loop different values of k using for loop and then visualize the value of k vs. the accuracy score. The visualization is shown below. After the evaluation, we select $k = 7$ for our model.



4.2 Decision Tree Model

The basic intuition behind a decision tree is to map out all the possible decision paths in the form of a tree. First, the attributes are considered and chosen one after another, then the significance of the attributes in the splitting of data is calculated. Thereafter, the data based on the value of the best attribute is split. Finally, this process is repeated until all the attributes are split into the leaf nodes. The aim here is to increase the accuracy as much as possible and keep the entropy smallest.

The image below shows the code by which the decision tree classifier model is been implemented and the first 10 predictions from the test data set.

```
DecTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)

DecTree.fit(X_train, y_train)

DecisionTreeClassifier(criterion='entropy', max_depth=4)

yhat_DecTree = DecTree.predict(X_test)
yhat_DecTree[0:10]

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

4.3 SVM Model

The support vector machine algorithm classifies cases by finding a separator. SVM works by mapping data to a high-dimensional feature space in such a way that data points can be classified even when the data is not linearly separable in 2D space. When plotting the data in multi-dimensional space, the category or class or the data points are separated using a separator represented as a hyperplane. The mapping of the data in higher dimensions is called kernelling. When creating an SVM object *clf*, RBF (Random Basis Function) type of kernel is been selected.

The following code below shows how an SVM model is been trained and tested.

```
clf = svm.SVC(kernel='rbf')

clf.fit(X_train, y_train)

SVC()

yhat_clf = clf.predict(X_test)
yhat_clf [0:5]

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

4.4 Logistic Regression Model

Logistic regression is one of the widely used classifiers for the categorical variable. Logistic regression can also be used to determine the probability of the outcome and to understand the impact of the feature. The probability is predicted by fitting the data using a sigmoid or logistic function. After fitting the data, the model output *yhat* and the actual output of severity are compared, and the error value is recorded.

The aim here is to reduce the cost i.e. maximizing the accuracy by constantly changing the value of θ . In our instance of logistic regression, the value of C (indicates inverse of regularization strength) is taken as 0.01 (must be a positive float). The numerical optimizer used is 'liblinear'.

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

yhat_LR = LR.predict(X_test)
yhat_LR[0:5]

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

After all the models are fitted with the train data, the next step is to evaluate the accuracy of each model. This will be explained in the next chapter.

5 Evaluation and Conclusion

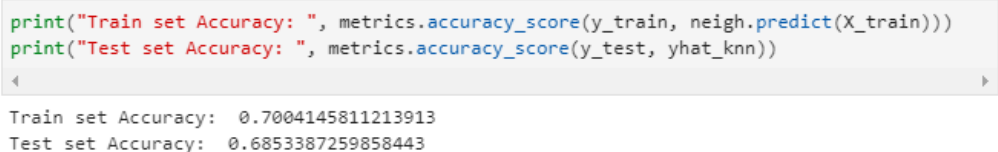
In this final chapter, we will discuss the criteria selected for model evaluation, results and conclude the project with future its future scopes.

5.1 Evaluation and Accuracy

For the evaluation of the models, we have selected 3 types of evaluation criteria. They explained in short detail. All the implemented method gives a score between 0 and 1 where 0 being least accurate and 1 being very accurate.

5.1.1 Accuracy Scores:

There are two types of accuracy scores calculated for our selected models namely for the training set and the testing set. The *metrics.accuracy_score* method is used to determine the accuracy scores. This method computes subset accuracy. Essentially, it calculates how closely the actual labels and predicted labels are matched in the test set. The code for calculating the accuracy score for the KNN model is shown in the image below.



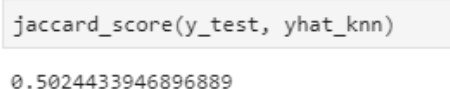
```
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat_knn))
```

Train set Accuracy: 0.7004145811213913
Test set Accuracy: 0.6853387259858443

Here, we must give more emphasis to the test set accuracy because it is the out of sample data.

5.1.2 Jaccard Similarity Score:

The Jaccard index, or Jaccard similarity coefficient, defined as the size of the intersection divided by the size of the union of two label sets, is used to compare a set of predicted labels for a sample to the corresponding set of labels in *y_test*. The following image shows how to calculate the Jaccard Similarity Score.



```
jaccard_score(y_test, yhat_knn)
```

0.5024433946896889

5.1.3 F1-Score:

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and the worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

The method `f1_score` is used to calculate the f1 score. Here the value of the average parameter is taken as 'weighted' which calculates metrics for each label and find their average weighted by support (the number of true instances for each label).

```
f1_score(y_test, yhat_knn, average='weighted')
0.6898577849912421
```

5.2 Results and Conclusion

The following table shows the above-mentioned accuracy scores for 4 of our selected models.

Algorithm	Jaccard	F1-score	Train set Accuracy	Test set Accuracy
KNN	0.4884	0.6832	0.7004	0.6853
Decision Tree	0.4173	0.6679	0.6922	0.6861
SVM	0.4943	0.6981	0.7125	0.7023
Logistic Regression	0.4736	0.6815	0.6960	0.6860

Here, we can see that no algorithm has resulted in an accuracy of more than 71%. Considering the result, we can conclude that most model has shown the score in very close range of approx. +- 2%. The Support Vector Machine provides the highest accuracy scores in all the categories. Thus, for this project SVM classifiers proves to be the relatively most accurate model to predict the severity of the accident given different attributes or conditions.

Moreover, it has also been observed that the time required to fit the model and predicting the output, Logistic regression, and Decision tree took the least time.

The maximum test accuracy of the SNM model is 70% which can be a result of underfitting and can be improved using more data. Due to an imbalance in the selected data set, many observations are dropped, so in the future, a more balanced data set can be used.

5.3 Future Scopes

As mentioned in chapter 3, many important columns such as 'UNDERINFL' and 'INATTENTIONIND' which shows whether the driver is under influence of drugs and driver inattention respectively, has been dropped due to data inconstancy and missing values. These could have been good factors in predicting accident severity. In the future, we can select more consistent data.

Moreover, we only had a binary label in our data set (1s and 2s), in the future, we can have data with multiclass labels to make machine learning model more challenging.