Capstone project_Accidents severity

November 1, 2020

1 Car Accident Severity in Seattle

1.1 Applied Data Science Capstone - Coursera

NOTE: This notebook shows the process of building a machine learning model for accident severity prediction. It is part of the final capstone project in Coursera to obtain the IBM Professional Certificate in Data Science.

1.2 Introduction

Car accidents happen every day for a variety of reasons and these have significant socioeconomic costs. Efforts to raise drivers' awareness towards mindful driving have been promoted across the USA and the authorities try to provide the conditions (e.g. road signs, traffic lights, traffic information, radars) to mitigate the probability of accidents happening. Today we have the data and the modeling capacities to even better understand the conditions that promote severe accidents and this project intends to build a machine learning model to better inform decision-makers in the city of Seattle using available data. This model will help the authorities to take appropriate measures to reduce accident severity and improve traffic safety.

1.3 Data

[33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[34]:
```
df= pd.read_csv(r'C:\Users\marco\Desktop\Data Science\IBM Coursera\Capstone␣
 ↪project\Data-Collisions.csv')
```

```
C:\Users\marco\anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (33) have
mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

The data was provided by the Seattle Police Department and corresponds to collisions registered between 2004 and 2020. The data is stored in a CSV file, presenting 38 columns and 194673 rows.

It describes the details of each accident, including weather conditions, collision type, date/time of accident and location.

In the dataset we have 3 types of variables: integers (12), floats (4) and objects (22), as we can see below:

[35]:
```
df.dtypes
```

[35]:
```
SEVERITYCODE         int64
X                  float64
Y                  float64
OBJECTID             int64
INCKEY               int64
COLDETKEY            int64
REPORTNO            object
STATUS              object
ADDRTYPE            object
INTKEY             float64
LOCATION            object
EXCEPTRSNCODE       object
EXCEPTRSNDESC       object
SEVERITYCODE.1       int64
SEVERITYDESC        object
COLLISIONTYPE       object
PERSONCOUNT          int64
PEDCOUNT             int64
PEDCYLCOUNT          int64
VEHCOUNT             int64
INCDATE             object
INCDTTM             object
JUNCTIONTYPE        object
SDOT_COLCODE         int64
SDOT_COLDESC        object
INATTENTIONIND      object
UNDERINFL           object
WEATHER             object
ROADCOND            object
LIGHTCOND           object
PEDROWNOTGRNT       object
SDOTCOLNUM         float64
SPEEDING            object
ST_COLCODE          object
ST_COLDESC          object
SEGLANEKEY           int64
CROSSWALKKEY         int64
HITPARKEDCAR        object
dtype: object
```

The variable SEVERITYCODE encodes the Seattle Department of Transport accident severity metric and this will be our 'dependent variable' (the variable we want to predict). The numerical codes and their meaning are as follows:

- 0: Unknown
- 1: Property damage
- 2: Injury
- 2b: Serious injury
- 3: Fatality

[36]:
```
df['SEVERITYCODE'].value_counts().to_frame()
```

[36]:
```
   SEVERITYCODE
1        136485
2         58188
```

By analysing the dataset, we can see that there are only two levels (out of five) of 'severity' registered: - 1: 136485 registrations - 2: 58188 registrations

The data is unbalanced, since we have many more instances of 'severity 1' compared with 'severity 2'. Data must be balanced and normalized in the data processsing step.

We have 37 attributes (columns) that can be used for building the model , but not all are useful.

At this stage, the following columns were dropped from the dataset as they were deemed not useful for the model (e.g. unnecessary, uninformative or redundant columns).

[37]:
```
df.
 ↪drop(columns=['JUNCTIONTYPE','INCDATE','PEDROWNOTGRNT','ST_COLCODE','PEDCYLCOUNT','PERSONCOUN
 ↪'OBJECTID', 'X', 'Y', 'INCKEY', 'REPORTNO', 'EXCEPTRSNCODE', 'EXCEPTRSNDESC',␣
 ↪'SEVERITYCODE.1', 'SEVERITYDESC', 'STATUS', 'COLDETKEY', 'LOCATION', 'INTKEY',␣
 ↪'INCDTTM','SDOT_COLDESC', 'SDOT_COLCODE', 'INATTENTIONIND', 'SDOTCOLNUM',␣
 ↪'ST_COLDESC', 'SEGLANEKEY', 'CROSSWALKKEY'], inplace= True)
df.head()
```

[37]:
```
   SEVERITYCODE      ADDRTYPE  VEHCOUNT UNDERINFL   WEATHER ROADCOND  \
0             2  Intersection         2         N  Overcast      Wet
1             1         Block         2         0   Raining      Wet
2             1         Block         3         0  Overcast      Dry
3             1         Block         3         N     Clear      Dry
4             2  Intersection         2         0   Raining      Wet

                  LIGHTCOND SPEEDING HITPARKEDCAR
0                  Daylight      NaN            N
1  Dark - Street Lights On      NaN            N
2                  Daylight      NaN            N
3                  Daylight      NaN            N
4                  Daylight      NaN            N
```

1.4 Methodology

Now we will perform a data wrangling step to prepare the dataset for analysis. First lets look at missing data:

[38]:
```
#Looking for missing data
missing_data = df.isnull()
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("---------------------")
```

```
SEVERITYCODE
False    194673
Name: SEVERITYCODE, dtype: int64
---------------------
ADDRTYPE
False    192747
True       1926
Name: ADDRTYPE, dtype: int64
---------------------
VEHCOUNT
False    194673
Name: VEHCOUNT, dtype: int64
---------------------
UNDERINFL
False    189789
True       4884
Name: UNDERINFL, dtype: int64
---------------------
WEATHER
False    189592
True       5081
Name: WEATHER, dtype: int64
---------------------
ROADCOND
False    189661
True       5012
Name: ROADCOND, dtype: int64
---------------------
LIGHTCOND
False    189503
True       5170
Name: LIGHTCOND, dtype: int64
---------------------
SPEEDING
True     185340
False      9333
```

```
Name: SPEEDING, dtype: int64
---------------------
HITPARKEDCAR
False    194673
Name: HITPARKEDCAR, dtype: int64
---------------------
```

From the output above, we can see that columns 'SPEEDING' has much more missing data cells than not. Therefore we will drop this columns from our analysis. Columns 'ADDRTYPE', 'JUNCTIONTYPE', 'UNDERINFL', 'WEATHER', 'ROADCOND' and 'LIGHTCOND' have some missing data. In this case, we will drop the rows with missing values on those columns.

[39]:
```
df.drop(columns=['SPEEDING'], inplace= True)
```

[40]:
```
df=df.dropna(axis=0)
df.head()
```

[40]:
```
   SEVERITYCODE      ADDRTYPE  VEHCOUNT UNDERINFL   WEATHER ROADCOND  \
0             2  Intersection         2         N  Overcast      Wet
1             1         Block         2         0   Raining      Wet
2             1         Block         3         0  Overcast      Dry
3             1         Block         3         N     Clear      Dry
4             2  Intersection         2         0   Raining      Wet

                  LIGHTCOND HITPARKEDCAR
0                  Daylight            N
1  Dark - Street Lights On            N
2                  Daylight            N
3                  Daylight            N
4                  Daylight            N
```

In the column 'UNDERINFL' we have a mix of numerical (0 and 1) and categorical data (Y and N). We will convert Y to 1 and N to 0 to uniformize the data to numerical.

[41]:
```
df['UNDERINFL'].replace('Y', 1, inplace=True)
df['UNDERINFL'].replace('N', 0, inplace=True)
df['UNDERINFL']= df['UNDERINFL'].astype('int')
```

Lets also convert 'HITPARKEDCAR' to numeric variables 0 and 1

[42]:
```
df['HITPARKEDCAR'].replace('Y', 1, inplace=True)
df['HITPARKEDCAR'].replace('N', 0, inplace=True)
df['HITPARKEDCAR']= df['HITPARKEDCAR'].astype('int')
```

Moreover, in column 'LIGHTCOND' we will merge the categorie levels 'Dark - No Street Lights' and 'Dark - Street Lights Off' to a single category 'Dark - Street Light Off'. We will eliminate rows with 'Other' and 'Dark - Unknown Lighting' as they represent a small fraction of cases and do not provide relevant information. Let first look at value counts.

[43]:
```
df['LIGHTCOND'].value_counts().to_frame()
```

[43]:

	LIGHTCOND
Daylight	115408
Dark - Street Lights On	48236
Unknown	12599
Dusk	5843
Dawn	2491
Dark - No Street Lights	1526
Dark - Street Lights Off	1184
Other	227
Dark - Unknown Lighting	11

[44]:
```
df['LIGHTCOND'].replace('Dark - No Street Lights', 'Night', inplace=True)
df['LIGHTCOND'].replace('Dusk', 'Dusk/Dawn', inplace=True)
df['LIGHTCOND'].replace('Dawn', 'Dusk/Dawn', inplace=True)
df['LIGHTCOND'].replace('Dark - Street Lights On', 'Night', inplace=True)
df['LIGHTCOND'].replace('Dark - Street Lights Off', 'Night', inplace=True)
indexNames = df[df['LIGHTCOND'] == 'Other'].index
df.drop(indexNames, inplace=True)
```

[45]:
```
indexNames = df[df['LIGHTCOND'] == 'Dark - Unknown Lighting'].index
df.drop(indexNames, inplace=True)
```

[46]:
```
indexNames = df[df['LIGHTCOND'] == 'Unknown'].index
df.drop(indexNames, inplace=True)
```

Columns 'WEATHER' will also have merged and dropped categories to reduce category levels. New category 'Elements' will include all categories that involve weather elements like rain, snow and wind. We will drop 'Partly Cloudy', 'Unknown' and 'Other'

[47]:
```
df['WEATHER'].value_counts().to_frame()
```

[47]:

	WEATHER
Clear	108848
Raining	32549
Overcast	27135
Unknown	4128
Snowing	824
Fog/Smog/Smoke	554
Other	468
Sleet/Hail/Freezing Rain	109
Blowing Sand/Dirt	43
Severe Crosswind	25
Partly Cloudy	5

[48]:
```python
df['WEATHER'].replace('Raining', 'Elements', inplace=True)
df['WEATHER'].replace('Snowing', 'Elements', inplace=True)
df['WEATHER'].replace('Sleet/Hail/Freezing Rain', 'Elements', inplace=True)
df['WEATHER'].replace('Raining', 'Elements', inplace=True)
df['WEATHER'].replace('Fog/Smog/Smoke', 'Elements', inplace=True)
df['WEATHER'].replace('Blowing Sand/Dirt', 'Elements', inplace=True)
df['WEATHER'].replace('Severe Crosswind', 'Elements', inplace=True)
```

[49]:
```python
indexNames = df[df['WEATHER'] == 'Other'].index
df.drop(indexNames, inplace=True)
```

[50]:
```python
indexNames = df[df['WEATHER'] == 'Unknown'].index
df.drop(indexNames, inplace=True)
```

[51]:
```python
indexNames = df[df['WEATHER'] == 'Partly Cloudy'].index
df.drop(indexNames, inplace=True)
```

Columns 'ROADCOND' will also have merged and dropped categories to reduce category levels. New category 'Elements' will include all categories that involve elements like water, sand and ice. We will drop 'Unknown' and 'Other'

[52]:
```python
df['ROADCOND'].value_counts().to_frame()
```

[52]:

	ROADCOND
Dry	120974
Wet	46134
Ice	1072
Snow/Slush	829
Unknown	753
Standing Water	101
Other	100
Sand/Mud/Dirt	64
Oil	60

[53]:
```python
df['ROADCOND'].replace('Ice', 'Elements', inplace=True)
df['ROADCOND'].replace('Wet', 'Elements', inplace=True)
df['ROADCOND'].replace('Snow/Slush', 'Elements', inplace=True)
df['ROADCOND'].replace('Standing Water', 'Elements', inplace=True)
df['ROADCOND'].replace('Sand/Mud/Dirt', 'Elements', inplace=True)
df['ROADCOND'].replace('Oil', 'Elements', inplace=True)
```

[54]:
```python
indexNames = df[df['ROADCOND'] == 'Other'].index
df.drop(indexNames, inplace=True)
```

[55]:
```python
indexNames = df[df['ROADCOND'] == 'Unknown'].index
df.drop(indexNames, inplace=True)
```

[56]:
```
df.head()
```

[56]:
```
   SEVERITYCODE      ADDRTYPE  VEHCOUNT  UNDERINFL   WEATHER  ROADCOND  \
0             2  Intersection         2          0  Overcast  Elements
1             1         Block         2          0  Elements  Elements
2             1         Block         3          0  Overcast       Dry
3             1         Block         3          0     Clear       Dry
4             2  Intersection         2          0  Elements  Elements

  LIGHTCOND  HITPARKEDCAR
0  Daylight             0
1     Night             0
2  Daylight             0
3  Daylight             0
4  Daylight             0
```

As we saw above, 'SEVERITYCODE' (the dependent varaiable we want to predict based on selected features) is unbalanced, since we have many more cells with severity code '1' than '2'. Let's use resampling to balance the data.

[57]:
```
df['SEVERITYCODE'].value_counts().to_frame()
```

[57]:
```
   SEVERITYCODE
1        113698
2         55536
```

[58]:
```
from sklearn.utils import resample
df_sevcode_1= df[df["SEVERITYCODE"]== 1]
df_sevcode_2= df[df["SEVERITYCODE"]== 2]
df_sevcode_1_down= resample(df_sevcode_1,
                           replace= False,
                           n_samples= 55536,
                           random_state= 123)
df= pd.concat([df_sevcode_1_down, df_sevcode_2])
df
```

[58]:
```
        SEVERITYCODE      ADDRTYPE  VEHCOUNT  UNDERINFL   WEATHER  ROADCOND  \
169416             1  Intersection         2          0     Clear       Dry
143128             1         Block         2          0     Clear       Dry
54715              1         Block         2          0  Overcast  Elements
111355             1         Block         1          0     Clear  Elements
45723              1         Block         2          0     Clear       Dry
...              ...           ...       ...        ...       ...       ...
194663             2         Block         2          0  Elements  Elements
194666             2         Block         2          0     Clear  Elements
194668             2         Block         2          0     Clear       Dry
194670             2  Intersection         2          0     Clear       Dry
```

```
194671              2  Intersection        1          0     Clear       Dry

        LIGHTCOND  HITPARKEDCAR
169416   Daylight             0
143128   Daylight             0
54715    Daylight             0
111355      Night             0
45723       Night             0
...           ...           ...
194663   Daylight             0
194666   Daylight             0
194668   Daylight             0
194670   Daylight             0
194671  Dusk/Dawn             0

[111072 rows x 8 columns]
```

[59]:
```python
df['SEVERITYCODE'].value_counts().to_frame()
```

[59]:
```
   SEVERITYCODE
2         55536
1         55536
```

Since most Machine Learning models require numerical data, we need to convert categorical variables to numerical ones. We will us one-hot enconding for this step.

[60]:
```python
df2=pd.get_dummies(df[["ADDRTYPE"]])
df3=pd.get_dummies(df[["WEATHER"]])
df4=pd.get_dummies(df[["ROADCOND"]])
df5=pd.get_dummies(df[["LIGHTCOND"]])
df=pd.concat([df,df2,df3,df4,df5],axis=1)
df.drop(columns=["ADDRTYPE", "WEATHER", "ROADCOND", "LIGHTCOND"], inplace= True)
```

[61]:
```python
df.head()
```

[61]:
```
        SEVERITYCODE  VEHCOUNT  UNDERINFL  HITPARKEDCAR  ADDRTYPE_Alley  \
169416             1         2          0             0               0
143128             1         2          0             0               0
54715              1         2          0             0               0
111355             1         1          0             0               0
45723              1         2          0             0               0

        ADDRTYPE_Block  ADDRTYPE_Intersection  WEATHER_Clear  \
169416               0                      1              1
143128               1                      0              1
54715                1                      0              0
111355               1                      0              1
```

```
45723                    1                      0              1

        WEATHER_Elements  WEATHER_Overcast  ROADCOND_Dry  ROADCOND_Elements  \
169416                 0                 0             1                  0
143128                 0                 0             1                  0
54715                  0                 1             0                  1
111355                 0                 0             0                  1
45723                  0                 0             1                  0

        LIGHTCOND_Daylight  LIGHTCOND_Dusk/Dawn  LIGHTCOND_Night
169416                   1                    0                0
143128                   1                    0                0
54715                    1                    0                0
111355                   0                    0                1
45723                    0                    0                1
```

[62]:
```
df.dtypes
```

[62]:
```
SEVERITYCODE             int64
VEHCOUNT                 int64
UNDERINFL                int32
HITPARKEDCAR             int32
ADDRTYPE_Alley           uint8
ADDRTYPE_Block           uint8
ADDRTYPE_Intersection    uint8
WEATHER_Clear            uint8
WEATHER_Elements         uint8
WEATHER_Overcast         uint8
ROADCOND_Dry             uint8
ROADCOND_Elements        uint8
LIGHTCOND_Daylight       uint8
LIGHTCOND_Dusk/Dawn      uint8
LIGHTCOND_Night          uint8
dtype: object
```

The data will be now standardised (i.e. every column re-scaled to have ~zero mean and ~unit variance) using the scikit learn StandardScaler routine. But first let's define the feature set(s).

[63]:
```
df['SEVERITYCODE']= df['SEVERITYCODE'].astype(int)
```

[68]:
```
X1 = df[['VEHCOUNT',␣
 ↪'UNDERINFL','HITPARKEDCAR','ADDRTYPE_Alley','ADDRTYPE_Block',␣
 ↪'ADDRTYPE_Intersection','WEATHER_Clear',
           'WEATHER_Elements','WEATHER_Overcast',␣
 ↪'ROADCOND_Dry','ROADCOND_Elements','LIGHTCOND_Daylight','LIGHTCOND_Night',
           'LIGHTCOND_Dusk/Dawn']].values.astype(float)
```

[69]:
```
from sklearn import preprocessing
X1 = preprocessing.StandardScaler().fit(X1).transform(X1.astype(float))
X1[0:5]
```

[69]:
```
array([[ 0.07357415, -0.24302078, -0.15325831, -0.05508397, -1.23052533,
         1.23832995,  0.75036755, -0.50367739, -0.43297335,  0.63208088,
        -0.63208088,  0.70418669, -0.62997501, -0.22302594],
       [ 0.07357415, -0.24302078, -0.15325831, -0.05508397,  0.81266105,
        -0.80753922,  0.75036755, -0.50367739, -0.43297335,  0.63208088,
        -0.63208088,  0.70418669, -0.62997501, -0.22302594],
       [ 0.07357415, -0.24302078, -0.15325831, -0.05508397,  0.81266105,
        -0.80753922, -1.33268023, -0.50367739,  2.30961097, -1.58207603,
         1.58207603,  0.70418669, -0.62997501, -0.22302594],
       [-1.54625297, -0.24302078, -0.15325831, -0.05508397,  0.81266105,
        -0.80753922,  0.75036755, -0.50367739, -0.43297335, -1.58207603,
         1.58207603, -1.42007796,  1.58736455, -0.22302594],
       [ 0.07357415, -0.24302078, -0.15325831, -0.05508397,  0.81266105,
        -0.80753922,  0.75036755, -0.50367739, -0.43297335,  0.63208088,
        -0.63208088, -1.42007796,  1.58736455, -0.22302594]])
```

[70]:
```
y = df['SEVERITYCODE'].values
y[0:5]
```

[70]:
```
array([1, 1, 1, 1, 1])
```

1.4.1 Model development

Train-Test Split

[71]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, 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: (77750, 14) (77750,)
Test set: (33322, 14) (33322,)
```

1- k-Nearest Neighbours (kNN) model

[73]:
```
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):
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    KNNyhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, KNNyhat)

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

mean_acc
```

[73]:
```
array([0.51245423, 0.5423444 , 0.53400156, 0.56551227, 0.54417502,
       0.60149451, 0.55278795, 0.61394874, 0.55482864])
```

[74]:
```
k = 8
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

[74]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                     weights='uniform')
```

[75]:
```
KNNyhat = neigh.predict(X_test)
```

[76]:
```
print(KNNyhat [0:5])
print(y_test [0:5])
```

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

[77]:
```
print("Test set Accuracy: ", metrics.accuracy_score(y_test, KNNyhat))
```

```
Test set Accuracy:  0.6139487425724747
```

2- Decision Tree model

[78]:
```
from sklearn.tree import DecisionTreeClassifier
severityTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
severityTree
```

[78]:
```
DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=None, splitter='best')
```

[79]:
```
severityTree.fit(X_train,y_train)
```

[79]: DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=None, splitter='best')

[80]:
```
DTyhat = severityTree.predict(X_test)
```

[81]:
```
print (DTyhat [0:5])
print (y_test [0:5])
```

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

[82]:
```
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, DTyhat))
```

```
DecisionTrees's Accuracy:  0.6270331912850369
```

1.4.2 3- Support Vector Machine (SVM) model:

[83]:
```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

[83]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

[84]:
```
SVMyhat = clf.predict(X_test)
```

[85]:
```
print (SVMyhat [0:5])
print (y_test [0:5])
```

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

[86]:
```
print("SVM Accuracy: ", metrics.accuracy_score(y_test, SVMyhat))
```

```
SVM Accuracy:  0.6278134565752356
```

1.5 Model evaluation

[88]:
```
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_similarity_score
import itertools

print('Jaccard Similarity Score:')
print('')
print('KNN model:', jaccard_similarity_score(y_test, KNNyhat))
print('Decision Tree model:', jaccard_similarity_score(y_test, DTyhat))
print('SVM model:', jaccard_similarity_score(y_test, SVMyhat))
print('--------------------')
print('F1 Score')
print('')
print('KNN model:', f1_score(y_test, KNNyhat, average='weighted'))
print('Decision Tree model:', f1_score(y_test, DTyhat, average='weighted'))
print('SVM model:', f1_score(y_test, SVMyhat, average='weighted'))
print('--------------------')
```

```
Jaccard Similarity Score:

KNN model: 0.6139487425724747
Decision Tree model: 0.6270331912850369
SVM model: 0.6278134565752356
--------------------
F1 Score

KNN model: 0.6122249347483049
Decision Tree model: 0.6264018955531346
SVM model: 0.6270639867285103
--------------------

C:\Users\marco\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:664: FutureWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  FutureWarning)
```

2 Results and Discussion

In order to develop a model for predicting accident severity, the re-sampled, cleaned dataset was split in to testing and training sub-samples (containing 30% and 70% of the samples, respectively) using the scikit learn "train_test_split" method. In total, 3 models were trained and evaluated.

2.0.1 KNN model

The value of'k' was established by running kNN models for k=1–10 using the kNeighborsClassifier function from scikit learn. The model is optimised at k=8, at which the model correctly predicts accident severity 61% of the time. The Jaccard Index and F1 score are respectively 0.614 and 0.612.

2.0.2 Decision Tree model

A decision tree model was trained on the data according to the "entropy" criterion, and allowed to run until covergence. The decision tree correctly predicts accident severity 63% of the time and has Jaccard Index and F1 scores of 0.627 and 0.626 respectively.

2.0.3 SVM model

An SVM model was built using the scikit learn C-Support Vector Classification method (svm.svc), with a linear mapping kernel employed in order that the model could return a list of the features with the most diagnostic power for determining accident severity. The SVM model correctly predicts accident severity 63% of the time, and has Jaccard Index and F1 scores of 0.628 and 0.627 respectively.

From the model evaluation indexes, we can conclude that the 3 models have a similar capacity to predict accident severity. Models could be further explored by changing the features set and see if prediction accuracy increases by removing or including features.

3 Conclusion

Car accident data for the city of Seattle between 2004–2019 have been used to train and evaluate machine learning models for predicting accident severity based on the context of the accident. Three classes of models have been trained and evaluated: (i) k-Nearest Neighbors, (ii) Decision Tree and (iii) Support Vector Machine. The three models performed similarly, predicting correctly 62-63% of severity scores, with a slightly better performance by the SVM model. This work highlights that machine learning techniques can be used to probe historical data in order to make reliable predictions about the outcome of road traffic accidents, given information which is available at the time when an accident is reported. This model can be extended to include new features and can give city planners insight into the road conditions/features which are associated with higher accident severity. By predicting accident severity as a function of weather, date, location and road conditions, this model may be able to help aid the decision making of city roads planning.

```
[ ]:
```