Capstone Project

In this notebook we try to practice all the classification algorithms that we learned in this course.

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

Lets first load required libraries:

In [32]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

Lets download the dataset

In [2]:
```
df = pd.read_csv("/Users/auguststapf/Downloads/Data-Collisions.csv")
```

```
/Users/auguststapf/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3057: DtypeWarning: Columns (33) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Lets look into the dataset and the attributes

In [3]:
```
df.head()
```

Out[3]:

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

5 rows × 38 columns

Next, we look into what values the target variable, severity code, obtains in this dataset.

In [4]:
```
df['SEVERITYCODE'].value_counts()
```

Out[4]:
```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

We next confirm what columns we have in our dataset in order to cut out the unnecessary.

In [5]:
```
df.columns
```

Out[5]:
```
Index(['SEVERITYCODE', 'X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPORTNO',
       'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
       'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC', 'COLLISIONTYPE',
       'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INCDATE',
       'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE', 'SDOT_COLDESC',
       'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',
       'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING', 'ST_COLCODE', 'ST_COLDESC',
       'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR'],
      dtype='object')
```

We determine the foloowing attributes in Feature are necessary for the creation of our models.

In [6]:
```
Feature = df[['PERSONCOUNT','VEHCOUNT','WEATHER','ROADCOND','LIGHTCOND','SPEEDING','JUNCTIONTYPE']]
```

In [7]:
```
Feature.head()
```

Out[7]:

	PERSONCOUNT	VEHCOUNT	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	JUNCTIONTYPE
0	2	2	Overcast	Wet	Daylight	NaN	At Intersection (intersection related)
1	2	2	Raining	Wet	Dark - Street Lights On	NaN	Mid-Block (not related to intersection)
2	4	3	Overcast	Dry	Daylight	NaN	Mid-Block (not related to intersection)
3	3	3	Clear	Dry	Daylight	NaN	Mid-Block (not related to intersection)
4	2	2	Raining	Wet	Daylight	NaN	At Intersection (intersection related)

Now, we begin the process of changing the categorical variables into a usable format.

In [8]:
```
Feature = pd.concat([Feature,pd.get_dummies(df['WEATHER'])], axis=1)
Feature = pd.concat([Feature,pd.get_dummies(df['ROADCOND'])], axis=1)
Feature = pd.concat([Feature,pd.get_dummies(df['LIGHTCOND'])], axis=1)
Feature = pd.concat([Feature,pd.get_dummies(df['JUNCTIONTYPE'])], axis=1)
Feature.columns
```

Out[8]:
```
Index(['PERSONCOUNT', 'VEHCOUNT', 'WEATHER', 'ROADCOND', 'LIGHTCOND',
       'SPEEDING', 'JUNCTIONTYPE', 'Blowing Sand/Dirt', 'Clear',
       'Fog/Smog/Smoke', 'Other', 'Overcast', 'Partly Cloudy', 'Raining',
       'Severe Crosswind', 'Sleet/Hail/Freezing Rain', 'Snowing', 'Unknown',
       'Dry', 'Ice', 'Oil', 'Other', 'Sand/Mud/Dirt', 'Snow/Slush',
       'Standing Water', 'Unknown', 'Wet', 'Dark - No Street Lights',
       'Dark - Street Lights Off', 'Dark - Street Lights On',
       'Dark - Unknown Lighting', 'Dawn', 'Daylight', 'Dusk', 'Other',
       'Unknown', 'At Intersection (but not related to intersection)',
       'At Intersection (intersection related)', 'Driveway Junction',
       'Mid-Block (but intersection related)',
       'Mid-Block (not related to intersection)', 'Ramp Junction', 'Unknown'],
      dtype='object')
```

In [9]:
```
Feature.drop(['Unknown','WEATHER','ROADCOND','LIGHTCOND','JUNCTIONTYPE','Other'], axis = 1,inplace=True)
Feature.columns
```

Out[9]:
```
Index(['PERSONCOUNT', 'VEHCOUNT', 'SPEEDING', 'Blowing Sand/Dirt', 'Clear',
       'Fog/Smog/Smoke', 'Overcast', 'Partly Cloudy', 'Raining',
       'Severe Crosswind', 'Sleet/Hail/Freezing Rain', 'Snowing', 'Dry', 'Ice',
       'Oil', 'Sand/Mud/Dirt', 'Snow/Slush', 'Standing Water', 'Wet',
       'Dark - No Street Lights', 'Dark - Street Lights Off',
       'Dark - Street Lights On', 'Dark - Unknown Lighting', 'Dawn',
       'Daylight', 'Dusk', 'At Intersection (but not related to intersection)',
       'At Intersection (intersection related)', 'Driveway Junction',
       'Mid-Block (but intersection related)',
       'Mid-Block (not related to intersection)', 'Ramp Junction'],
      dtype='object')
```

In [10]:
```
Feature.head()
```

Out[10]:

	PERSONCOUNT	VEHCOUNT	SPEEDING	Blowing Sand/Dirt	Clear	Fog/Smog/Smoke	Overcast	Partly Cloudy	Raining	Severe Crosswind	...	Dark - Unknown Lighting	Dawn	Daylight	Dusk
0	2	2	NaN	0	0	0	1	0	0	0	...	0	0	1	0
1	2	2	NaN	0	0	0	0	0	1	0	...	0	0	0	0
2	4	3	NaN	0	0	0	1	0	0	0	...	0	0	1	0
3	3	3	NaN	0	1	0	0	0	0	0	...	0	0	1	0
4	2	2	NaN	0	0	0	0	0	1	0	...	0	0	1	0

5 rows × 32 columns

In [11]:
```
Feature['SPEEDING'].fillna(0, inplace=True)
Feature['SPEEDING'].replace(to_replace=['0','Y'], value=[0,1],inplace=True)
```

In [12]:
```
Feature.head()
```

Out[12]:

	PERSONCOUNT	VEHCOUNT	SPEEDING	Blowing Sand/Dirt	Clear	Fog/Smog/Smoke	Overcast	Partly Cloudy	Raining	Severe Crosswind	...	Dark - Unknown Lighting	Dawn	Daylight	Dusk
0	2	2	0	0	0	0	1	0	0	0	...	0	0	1	0
1	2	2	0	0	0	0	0	0	1	0	...	0	0	0	0
2	4	3	0	0	0	0	1	0	0	0	...	0	0	1	0
3	3	3	0	0	1	0	0	0	0	0	...	0	0	1	0
4	2	2	0	0	0	0	0	0	1	0	...	0	0	1	0

5 rows × 32 columns

In [13]:
```
Feature['SPEEDING'].value_counts()
```

Out[13]:
```
0    185340
1      9333
Name: SPEEDING, dtype: int64
```

In [14]:
```
X = Feature
X[0:5]
```

Out[14]:

	PERSONCOUNT	VEHCOUNT	SPEEDING	Blowing Sand/Dirt	Clear	Fog/Smog/Smoke	Overcast	Partly Cloudy	Raining	Severe Crosswind	...	Dark - Unknown Lighting	Dawn	Daylight	Dusk
0	2	2	0	0	0	0	1	0	0	0	...	0	0	1	0
1	2	2	0	0	0	0	0	0	1	0	...	0	0	0	0
2	4	3	0	0	0	0	1	0	0	0	...	0	0	1	0
3	3	3	0	0	1	0	0	0	0	0	...	0	0	1	0
4	2	2	0	0	0	0	0	0	1	0	...	0	0	1	0

5 rows × 32 columns

In [15]:
```
Y = df['SEVERITYCODE'].values
Y[0:5]
```

Out[15]:
```
array([2, 1, 1, 1, 2])
```

To finish our pre-processing of the data, we standardize all of the remaining attributes.

In [16]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[16]:
```
array([[-0.33020207,  0.12553783, -0.22440165, -0.01696304, -1.15340914,
        -0.05414257,  2.45445634, -0.00506801, -0.45298634, -0.011333  ,
        -0.02409974, -0.06841713, -1.33213439, -0.07905204, -0.01813462,
        -0.01963186, -0.07200071, -0.02431221,  1.76085874, -0.08920831,
        -0.07872239, -0.576075  , -0.00751719, -0.1141037 ,  0.82233559,
        -0.17682024, -0.10437651,  1.44892892, -0.2408193 , -0.36412936,
        -0.92535062, -0.02921369],
       [-0.33020207,  0.12553783, -0.22440165, -0.01696304, -1.15340914,
        -0.05414257, -0.4074222 , -0.00506801,  2.2075721 , -0.011333  ,
        -0.02409974, -0.06841713, -1.33213439, -0.07905204, -0.01813462,
        -0.01963186, -0.07200071, -0.02431221,  1.76085874, -0.08920831,
        -0.07872239,  1.73588509, -0.00751719, -0.1141037 , -1.21604855,
        -0.17682024, -0.10437651, -0.69016498, -0.2408193 , -0.36412936,
         1.08067145, -0.02921369],
       [ 1.15576451,  1.7102107 , -0.22440165, -0.01696304, -1.15340914,
        -0.05414257,  2.45445634, -0.00506801, -0.45298634, -0.011333  ,
        -0.02409974, -0.06841713,  0.75067501, -0.07905204, -0.01813462,
        -0.01963186, -0.07200071, -0.02431221, -0.56790473, -0.08920831,
        -0.07872239, -0.576075  , -0.00751719, -0.1141037 ,  0.82233559,
        -0.17682024, -0.10437651, -0.69016498, -0.2408193 , -0.36412936,
         1.08067145, -0.02921369],
       [ 0.41278122,  1.7102107 , -0.22440165, -0.01696304,  0.86699503,
        -0.05414257, -0.4074222 , -0.00506801, -0.45298634, -0.011333  ,
        -0.02409974, -0.06841713,  0.75067501, -0.07905204, -0.01813462,
        -0.01963186, -0.07200071, -0.02431221, -0.56790473, -0.08920831,
        -0.07872239, -0.576075  , -0.00751719, -0.1141037 ,  0.82233559,
        -0.17682024, -0.10437651, -0.69016498, -0.2408193 , -0.36412936,
         1.08067145, -0.02921369],
       [-0.33020207,  0.12553783, -0.22440165, -0.01696304, -1.15340914,
        -0.05414257, -0.4074222 , -0.00506801,  2.2075721 , -0.011333  ,
        -0.02409974, -0.06841713, -1.33213439, -0.07905204, -0.01813462,
        -0.01963186, -0.07200071, -0.02431221,  1.76085874, -0.08920831,
        -0.07872239, -0.576075  , -0.00751719, -0.1141037 ,  0.82233559,
        -0.17682024, -0.10437651,  1.44892892, -0.2408193 , -0.36412936,
        -0.92535062, -0.02921369]])
```

In [17]:
```
from sklearn.model_selection import train_test_split
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)
```

```
Train set: (155738, 32) (155738,)
```

KNN

K Nearest Neighbors will be our first approach at modeling using a categorical approach.

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

In [19]:
```
k = 7
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[19]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

In [33]:
```
KNN_yhat = neigh.predict(X_test)
KNN_yhat.shape
```

Out[33]:
```
(38935,)
```

In [34]:
```
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, KNN_yhat))
jc1 = metrics.accuracy_score(y_test, KNN_yhat)
jc1
```

```
Train set Accuracy:  0.7304896685458911
Test set Accuracy:  0.7198150764094002
```

Out[34]:
```
0.7198150764094002
```

In [39]:
```
fs1 = .7198
```

Decision Tree

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [21]:
```
loanTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
loanTree # it shows the default parameters
```

Out[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')
```

In [22]:
```
loanTree.fit(X_train,y_train)
```

Out[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')
```

In [23]:
```
predTree = loanTree.predict(X_test)
```

In [24]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, predTree))
jc2 = metrics.accuracy_score(y_test, predTree)
jc2
```

```
DecisionTrees's Accuracy:  0.7424682162578656
```

Out[24]:
```
0.7424682162578656
```

In [25]:
```
from sklearn.metrics import f1_score
f1_score(y_test, predTree, average='weighted')
fs2 = f1_score(y_test, predTree, average='weighted')
fs2
```

Out[25]:
```
0.6937193088387895
```

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[26]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [27]:
```
log_yhat = LR.predict(X_test)
log_yhat
```

Out[27]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [28]:
```
log_yhat_prob = LR.predict_proba(X_test)
log_yhat_prob[0:5]
```

Out[28]:
```
array([[0.71318231, 0.28681769],
       [0.61107664, 0.38892336],
       [0.8113742 , 0.1886258 ],
       [0.66162734, 0.33837266],
       [0.73389257, 0.26610743]])
```

In [29]:
```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, log_yhat)
jc4 = jaccard_similarity_score(y_test, log_yhat)
jc4
```

```
/Users/auguststapf/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: 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.
  'and multiclass classification tasks.', DeprecationWarning)
/Users/auguststapf/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: 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.
  'and multiclass classification tasks.', DeprecationWarning)
```

Out[29]:
```
0.7278541158340824
```

In [30]:
```
from sklearn.metrics import log_loss
log_loss(y_test, log_yhat_prob)
ll = log_loss(y_test, log_yhat_prob)
ll
```

Out[30]:
```
0.5554503391691368
```

In [31]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
print (classification_report(y_test, log_yhat))
fs4 = .63
```

```
              precision    recall  f1-score   support

           1       0.74      0.96      0.83     27425
           2       0.64      0.19      0.29     11510

    accuracy                           0.73     38935
   macro avg       0.69      0.57      0.56     38935
weighted avg       0.71      0.73      0.67     38935
```

In [40]:
```
list_jc = [jc1, jc2, 'NA', jc4]
list_fs = [fs1, fs2, 'NA', fs4]
list_ll = ['NA', 'NA', 'NA', ll]

import pandas as pd

# fomulate the report format

df = pd.DataFrame(list_jc, index=['KNN','Decision Tree','SVM','Logistic Regression'])
df.columns = ['Jaccard']
df.insert(loc=1, column='F1-score', value=list_fs)
df.insert(loc=2, column='LogLoss', value=list_ll)
df.columns.name = 'Algorithm'
df
```

Out[40]:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.719815	0.7198	NA
Decision Tree	0.742468	0.693719	NA
SVM	NA	NA	NA
Logistic Regression	0.727854	0.63	0.55545

In [ ]: