```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
# reading the data

train_df = pd.read_csv("/content/drive/MyDrive/CSV files/Hackathon- Mach
test_df = pd.read_csv("/content/drive/MyDrive/CSV files/Hackathon- Mach
print(train_df.shape)
print(test_df.shape)
```

```
(51490, 42)
(77235, 42)
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2718: Dt
  interactivity=interactivity, compiler=compiler, result=result)
```

```
# train_df = pd.read_csv('/content/train.csv')
# test_df = pd.read_csv('/content/test.csv')
```

```
train_df.head()
```

	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Name	Cross-Street Type
0	MP060D	10196	Montgomery County Police	Injury Crash	06/05/2017 04:27:00 PM	Maryland (State)	OLD HUNDRED RD	County
1	MP000X	20080	Montgomery County Police	Injury Crash	02/22/2020 10:00:00 AM	County	JANET RD	County
2	MP070N	10103	Montgomery County Police	Property Damage Crash	06/11/2017 08:21:00 AM	Maryland (State)	COLESVILLE RD	Maryland (State)

```
test_df.head()
```

	Id	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Name	
0	0	MP810M	NaN	MONTGOMERY	Property Damage Crash	12/16/2015 05:42:00 PM	County	WHITTIER BLVD	
1	1	MP2001	NaN	Montgomery County Police	Property Damage Crash	05/01/2016 07:25:00 PM	County	RAILROAD ST	Mu
2	2	MP6404	10125	Montgomery County Police	Property Damage Crash	03/16/2018 02:30:00 PM	Maryland (State)	GEORGIA AVE	
3	3	MP0405	10270	Montgomery County Police	Property Damage Crash	05/25/2018 05:00:00 PM	NaN	NaN	
4	4	MP090D	10374	Montgomery County Police	Property Damage Crash	09/23/2017 05:40:00 PM	Interstate (State)	EISENHOWER MEMORIAL HWY	

```
# column location can be split with delimiter as space or comma

new = train_df['Location'].str.split(",| " , expand = True)
train_df['Location_0'] = new[0].astype(float)
train_df['Location_1'] = new[1].astype(float)
```

```
train_df.drop(['Location'] , axis = 'columns', inplace = True)
train_df.head()
```

	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Name	Cr St
0	MP060D	10196	Montgomery County Police	Injury Crash	06/05/2017 04:27:00 PM	Maryland (State)	OLD HUNDRED RD	C
1	MP000X	20080	Montgomery County Police	Injury Crash	02/22/2020 10:00:00 AM	County	JANET RD	C
2	MP070N	10103	Montgomery County Police	Property Damage Crash	06/11/2017 08:21:00 AM	Maryland (State)	COLESVILLE RD	Mar (S
3	MP130Y	10208	Montgomery County Police	Injury Crash	05/10/2019 07:30:00 AM	County	PARKLAND DR	C
4	MP770Y	1091	Montgomery County Police	Injury Crash	08/02/2016 05:00:00 PM	Maryland (State)	OLD GEORGETOWN RD	C

```
# Repeating same steps on test data

new = test_df['Location'].str.split(",| " , expand = True)
test_df['Location_0'] = new[0].astype(float)
test_df['Location_1'] = new[1].astype(float)
test_df.drop(['Location'] , axis = 'columns', inplace = True)
test_df.head()
```

	Id	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Nam
0	0	MP810M	NaN	MONTGOMERY	Property Damage Crash	12/16/2015 05:42:00 PM	County	WHITTIE BLV
1	1	MP2001	NaN	Montgomery County Police	Property Damage Crash	05/01/2016 07:25:00 PM	County	RAILROAD S
2	2	MP6404	10125	Montgomery County Police	Property Damage	03/16/2018 02:30:00	Maryland (State)	GEORG AV

```
# We assert that both Location_0 and Location_1 coordinates are between

for index,row in train_df.iterrows():
    x = row['Location_0']
    y = row['Location_1']
    while(x < -10 or x > 10):
        x /= 10
    train_df.at[index , 'Location_0']  =  x

    while(y < -10 or y > 10):
         y /= 10
    train_df.at[index , 'Location_1']  =  y

for index,row in test_df.iterrows():
    x = row['Location_0']
    y = row['Location_1']
    while(x < -10 or x > 10):
        x /= 10
    test_df.at[index , 'Location_0']  =  x

    while(y < -10 or y > 10):
         y /= 10
    test_df.at[index , 'Location_1']  =  y


```

```
train_df.head()
```

	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Name	Cr St
0	MP060D	10196	Montgomery County Police	Injury Crash	06/05/2017 04:27:00 PM	Maryland (State)	OLD HUNDRED RD	Co
1	MP000X	20080	Montgomery County Police	Injury Crash	02/22/2020 10:00:00 AM	County	JANET RD	Co
2	MP070N	10103	Montgomery County Police	Property Damage Crash	06/11/2017 08:21:00 AM	Maryland (State)	COLESVILLE RD	Mar (S
3	MP130Y	10208	Montgomery County Police	Injury Crash	05/10/2019 07:30:00 AM	County	PARKLAND DR	Co

```
# We drop columns Report Number,Local Case Number, Person Id from both c
# We store column ID in test data for further use and then drop it

indices = test_df['Id']

train_df.drop(['Report Number', 'Local Case Number' , 'Vehicle ID'] , ax
test_df.drop(['Report Number',  'Local Case Number' , 'Id', 'Vehicle ID
```

```
# Since most columns are catagorical, the idea is to drop columns with N
# But since many of the columns are having NaNs, replace them with most
# Not before dropping columns with more than 20000 Nans
pd.isna(train_df).sum()

for column in train_df.columns:
    if (pd.isna(train_df).sum() > 20000 )[column] ==True:
        train_df.drop([column] , axis = 'columns' , inplace = True)

train_df = train_df.apply(lambda x: x.fillna(x.value_counts().index[0])
```

```
# retrieving Fault column and storing it as target variable
y_train = np.array(train_df['Fault'])

train_df.drop(['Fault'] , axis = 'columns' , inplace = True)
```

```
# rearranging the columns of test_df in the same order as train_df
test_df = test_df[train_df.columns]
```

```

# filling in the NaNs of a column of test_df with most repeated values i
# same column of train_df
for column in test_df.columns:
    test_df[column].fillna(train_df[column].mode()[0] , inplace = True)

```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4536: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  downcast=downcast,
```

```
train_df.head()
```

	Agency Name	ACRS Report Type	Crash Date/Time	Route Type	Road Name	Cross-Street Type	Cross-Street Name
0	Montgomery County Police	Injury Crash	06/05/2017 04:27:00 PM	Maryland (State)	OLD HUNDRED RD	County	THURSTON RD
1	Montgomery County Police	Injury Crash	02/22/2020 10:00:00 AM	County	JANET RD	County	FLACK ST
2	Montgomery County Police	Property Damage Crash	06/11/2017 08:21:00 AM	Maryland (State)	COLESVILLE RD	Maryland (State)	STRUC #15082
3	Montgomery County Police	Injury Crash	05/10/2019 07:30:00 AM	County	PARKLAND DR	County	FRANKFORT DR
4	Montgomery County Police	Injury Crash	08/02/2016 05:00:00 PM	Maryland (State)	OLD GEORGETOWN RD	County	MCKINLEY ST

```
# we change the crash date/time column into a column which tells us if
# the crash was during hours AM or PM

# extracting AM and PM from Crash Date/Time column

new = train_df['Crash Date/Time'].str.split(" " ,  expand = True)
train_df['time'] = new[2]

# drop Crash Date/Time column
```

```
train_df.drop(['Crash Date/Time'] , axis = 'columns' , inplace = True)

# considering more prioirity to PM
train_df['time'] = train_df['time'].replace(['AM' , 'PM'] , [0,1])
```

```
# repeating the same steps on Crash Date/Time on test_df

# we change the crash date/time column into a column which tells us if
# the crash was during hours AM or PM

# extracting AM and PM from Crash Date/Time column

new = test_df['Crash Date/Time'].str.split(" " ,  expand = True)
test_df['time'] = new[2]

# drop Crash Date/Time column
test_df.drop(['Crash Date/Time'] , axis = 'columns' , inplace = True)

# considering more prioirity to PM
test_df['time'] = test_df['time'].replace(['AM' , 'PM'] , [0,1])
```

```
# finding the number of unique values per column with dtype==object

for column in train_df.columns:
    if train_df[column].dtypes == object:
        print(column , len(pd.unique(train_df[column])))
```

```
Agency Name 10
ACRS Report Type 3
Route Type 10
Road Name 2226
Cross-Street Type 10
Cross-Street Name 4525
Collision Type 18
Weather 12
Surface Condition 11
Light 8
Traffic Control 11
Driver Substance Abuse 11
Person ID 49209
Injury Severity 5
Drivers License State 65
```

```
Vehicle Damage Extent 7
Vehicle First Impact Location 16
Vehicle Second Impact Location 16
Vehicle Body Type 30
Vehicle Movement 22
Vehicle Continuing Dir 5
Vehicle Going Dir 5
Driverless Vehicle 2
Parked Vehicle 2
Vehicle Make 886
Vehicle Model 3128
Equipment Problems 8
```

```
# we drop columns with more than 20 unique values

for column in train_df.columns:
    if len(pd.unique(train_df[column])) > 50:
        train_df.drop(column, axis = 'columns' , inplace = True)
```

```
# keeping same columns in test_df
test_df = test_df[train_df.columns]

# comparing unique values per column in test and train data:
for column in test_df.columns:
    if test_df[column].dtypes == object:
            print(column , ", train= " ,len(pd.unique(train_df[column]))
```

```
Agency Name , train=  10 , test =  10
ACRS Report Type , train=  3 , test =  3
Route Type , train=  10 , test =  10
Cross-Street Type , train=  10 , test =  10
Collision Type , train=  18 , test =  18
Weather , train=  12 , test =  12
Surface Condition , train=  11 , test =  11
Light , train=  8 , test =  8
Traffic Control , train=  11 , test =  11
Driver Substance Abuse , train=  11 , test =  11
Injury Severity , train=  5 , test =  5
Vehicle Damage Extent , train=  7 , test =  7
Vehicle First Impact Location , train=  16 , test =  16
Vehicle Second Impact Location , train=  16 , test =  16
Vehicle Body Type , train=  30 , test =  30
Vehicle Movement , train=  22 , test =  22
Vehicle Continuing Dir , train=  5 , test =  5
Vehicle Going Dir , train=  5 , test =  5
Driverless Vehicle , train=  2 , test =  2
Parked Vehicle , train=  2 , test =  2
Equipment Problems , train=  8 , test =  10
```

```
# dropping Equipment problems because testing data has more unique value
# training data
```

```

train_df.drop(['Equipment Problems'] , axis = 'columns' , inplace = True
test_df.drop(['Equipment Problems'] , axis = 'columns' , inplace = True
```

```
# one hot encoding

train_df = pd.get_dummies(train_df)
test_df = pd.get_dummies(test_df)

print(train_df.shape, test_df.shape)
```

```
(51490, 216) (77235, 216)
```

```
X_train = np.array(train_df)
X_test = np.array(test_df)
```

```
model = KNeighborsClassifier(n_neighbors=7)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)


predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('KNN.csv' , columns = ['Id' , 'Fault'] , header = ['Id' ,

```

```
model = RandomForestClassifier(n_estimators=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

```

```
pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('RFC_1000.csv' , columns = ['Id' , 'Fault'] , header = [
```

```
model = GradientBoostingClassifier(n_estimators=2500)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('GBC_2500.csv' , columns = ['Id' , 'Fault'] , header =
```

```
model =  MLPClassifier( max_iter=700)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('MLP_700.csv' , columns = ['Id' , 'Fault'] , header = ['I
```

```
model = XGBClassifier(max_depth = 200, subsample = 0.75)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

```

```
pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('XGB_2.csv' , columns = ['Id' , 'Fault'] , header = ['Id
```

```
model = LGBMClassifier(n_estimators = 1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

predictions = []
for i in range(len(indices)):
    predictions.append([indices[i] , y_pred[i]])

predictions = np.array(predictions)

pred_df = pd.DataFrame(predictions , columns = ['Id' , 'Fault'])
pred_df

pred_df.to_csv('LGBM_1000.csv' , columns = ['Id' , 'Fault'] , header =
```

Conclusion

The final models we used was a Gradient Boosting Classifier with the number of estimators as 2500, and a Light Gradient Boosting Model with n_estimators = 1000.

The reason why we chose Gradient Boosting Classifier is because it combines various weak learners, by improving on the mistakes made by the previous models, and combines the results of all models. It is robust to over-fitting. It is a greedy algorithm, choosing trees based on best split, and the weak learners can be constrained, to ensure they remain week, but greedy.

Light Gradient Boosting Model is an algorithm,that significantly cuts on memory, and is extremely fast-paced, with note-worthy accuracies. It splits trees leaf-wise, rather than level-wise , and hence, obtains better accuracies compared to many of the other models.

The reason why we chose these models is because of the high number of features in our data due to the one hot encoding done, and an iterative approach to finding the pattern seemed like the best idea.

✓ 42s completed at 1:03 PM