CS4830 - BIG DATA LABORATORY
FINAL PROJECT REPORT

Group Name: Quarantined Cops

Members:

Roll Number	Name
CH16B024	Vishwesh Ramanathan
CH16B033	Ashwin S Nair
CH16B045	Raghav Rajesh Moar

➢ Objective:
1) To train a machine learning model on big dataset of NYC Parking tickets to predict the violation location
2) To be able to perform real-time computation using the trained model using Big data technologies taught in course CS4830.

➢ Initial Data Inspection:
No. of Rows/Entries: **1,18,09,233**
No of features (Columns): **50 columns**
16 features have only NaN entries for all the Rows and can be removed without affecting the model

➢ Pre-processing:
1) **Removing Duplicate Summons numbers:**
Since Summons numbers are unique for each row entry and having a duplicate summon implies that full row is repeated. Therefore all repeated rows are removed.

```
data_tickets.iloc[[141927,141928]]
```

	Summons Number	Plate ID	Registration State	Plate Type	Issue Date	Violation Code	Vehicle Body Type	Vehicle Make	Issuing Agency	Street Code1	Street Code2	Street Code3	Vehicle Expiration Date	Violati Coun
141927	1370137552	GJY9810	NY	PAS	08/22/2014	21	SDN	TOYOT	S	0	0	0	01/01/20160306 12:00:00 PM	73
141928	1370137552	GJY9810	NY	PAS	08/22/2014	21	SDN	TOYOT	S	0	0	0	01/01/20160306 12:00:00 PM	73

Fig 1. Duplicate Summon Number implies duplicate Row

2) **Removing not important Features:**
We inspected every feature manually. We found many features to not correlate well with the violation locations. Some of them are:
'Summons Number' as it just indicates new entry and has nothing to with violation location
'**Vehicle Expiration Date**' again its specific to vehicles and nothing indicating about violation location
'**Violation Legal Code**' as it indicates just the type of violation and not specific to some location
Similarly, other 14 not important features are removed. These features are removed based on either having a lot of categorical values with fewer data points for each category or having lots of missing data points. Features having lots of categorical values will only add noise to the modeling step. Hence on this basis, a lot of the features are removed.
So in total 16+17 = 33 features in total are removed.

```
#Data preprocessing pipeline
df=df.dropDuplicates(subset=['Summons Number'])
columns_to_drop = ['Plate ID','Issuer Code','Time First Observed','Vehicle Expiration Date','House Number','Street Name','Intersecting Street','[
df = df.drop(*columns_to_drop)
df.show(5)
```

```
+-------------+-----------------+----------+----------+--------------+-----------------+------------+--------------+------------+------------+-
|Summons Number|Registration State|Plate Type|Issue Date|Violation Code|Vehicle Body Type|Vehicle Make|Issuing Agency|Street Code1|Street Code2|S
+-------------+-----------------+----------+----------+--------------+-----------------+------------+--------------+------------+------------+-
|    1366539345|                NY|       PAS|07/01/2014|            31|              SDN|       HONDA|             P|       34430|       10410|
|    1367471345|                NY|       PAS|08/09/2014|            71|              SDN|       TOYOT|             P|       27820|       31390|
|    1368841090|                NY|       SRF|11/20/2014|            40|             SUBN|       CHEVR|             P|        8440|       20190|
|    1369109532|                NY|       COM|06/23/2014|            46|              VAN|       MERCU|             P|       18770|       10010|
|    1369992300|                ME|       PAS|08/20/2014|            66|             TRAI|        null|             S|       62030|       81050|
+-------------+-----------------+----------+----------+--------------+-----------------+------------+--------------+------------+------------+-
only showing top 5 rows
```

Fig 2. Dropping Unnecessary Columns

(*will edit image with removing summon number as well and showing all features that are removed)

3) **Extracting Issue Date (Feature Engineering):**
We split the Issue date into Year Month and Day columns. These columns are also converted to integers. We have also added a column, Day of the week which is calculated form year (since it can be an important feature) (using a python function)
The year column is further used to remove years with 2012 or less since this data corresponds to the years 2013-2014 and the data before these years are considered as outliers/ faulty rows. The Day column is useful as it can help to distinguish between Weekdays and Weekends based on Violation tickets issued at specific locations.

```python
#UDFS
#Extracting information from date
def day_finder(x):
    return datetime.datetime.strptime(x, '%m/%d/%Y').weekday()
#Bucketizing violation time
def time_bucket(x):
  #Bucketizing the time into 8 buckets
  if x is None:
    return 3
  if x[-1]!='P' and x[-1]!='A':
    return 3
  try:
    time = int(x[:-1])
  except:
    return 3
  if x[-1]=='P':
    time = 1200+time
  for i in range(8):
    if time>=300*i and time<300*(i+1):
      return i
time_udf = udf(lambda x: time_bucket(x), IntegerType())
day_udf = udf(lambda x: day_finder(x), IntegerType())
```

```python
#Splitting the issue date into month,year,day
df_new = df.withColumn('Month',split('Issue Date','/')[0]).withColumn('Year',split('Issue Date','/')[2]).withColumn('Day',day_udf(col('Issue Date'))).withColumn(
#converting the columns into integers
df_new = df_new.withColumn("Year",df_new["Year"].cast(IntegerType())).withColumn("Month",df_new["Month"].cast(DoubleType())).withColumn("Day",df_new["Day"].cast(
#Removing outliers and some filtering
df_new = df_new.where(f.col("Year")>2012)
#Dropping columns
df_new =df_new.drop(*['Issue Date','Violation Time','Year','Issuer Squad'])
#Filling na
df_new = df_new.fillna({'Time':3})
#Removing na locaions of violation location and violation count
df_new = df_new.dropna(how='any',subset=['Violation Location','Violation County'])
#Fill na of these columns using  respective max values
# cols = ['Vehicle Body Type','Vehicle Make','Violation County','Violation In Front Of Or Opposite']
# agg_expr = [mode(f.collect_list(col)).alias(col) for col in cols]
# max_vals = df_new.agg(*agg_expr).collect()[0]
# df_new = df_new.fillna({'Vehicle Body Type':max_vals['Vehicle Body Type'],'Vehicle Make':max_vals['Vehicle Make'],'Violation County':max_vals['Violation County
df_new=df_new.dropna(how='any')
#Renaming columns
names = df_new.schema.names
for name in names:
  df_new = df_new.withColumnRenamed(name, name.replace(" ","_"))
#Mapping violation location
df_new = df_new.withColumn('Violation_Location', regexp_replace('Violation_Location', 'KINGS', 'K'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'KING', 'K'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'QUEEN', 'Q'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'QU', 'Q'))\
```

Fig 3. Date and Time Feature Engineering

4) **Bucketing time (Feature Engineering):**
The Violation time column is put into 8 buckets of 3 hours each as certain time buckets will have more violations and certain localities will have violation tickets issued during specific hours. Later violation time is dropped.

5) **Correction in Violation Location**
Firstly, we drop all violations Locations with Nan values as this is our target column and we cannot have supervised learning without labels. Some of the target labels have been names differently. Hence, we mapped the differently named labels to the actual prominent label as shown below.
Also, note that we have mapped Manhattan(MH/MAN) to NY because of only 6 training data points being available which won't lead to a good prediction of that class. Since Manhattan also comes under the broad category of New York, we have hence mapped it as proposed.

```
for name in names:
  df_new = df_new.withColumnRenamed(name, name.replace(" ","_"))
#Mapping violation location
df_new = df_new.withColumn('Violation_Location', regexp_replace('Violation_Location', 'KINGS', 'K'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'KING', 'K'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'QUEEN', 'Q'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'QU', 'Q'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'NEWY', 'NY'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'NEW Y', 'NY'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'MAN', 'NY'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'MH', 'NY'))\
.withColumn('Violation_Location', regexp_replace('Violation_Location', 'BRONX', 'BX'))
```

Fig 4. Violation Location correction

6) **Label Encoding:**
Finally, all string(text) features are encoded using StringIndexer label encoding. For example registration states are Mapped as NY: 0, TX: 7 in StringIndexer.

==

➢ Performance of the best model and Inferences

We get the best accuracy with the XGBoost Classifier of 99.9% train and test accuracy (See Fig 5).
(Note: We also tried RandomForestClassifier giving 95.0% test and train accuracy,
and logistic regression giving 49% accuracy)

Also, XGBoost trained pretty fast than Random Forest

```
########################################################################## XGBOOST Training ##########################################################################################
#Training XGBOOST only on first few data
df_pandas = df_r1.limit(100000).toPandas()
column_names = ['Registration_State_index','Plate_Type_index','Violation_Code_index','Vehicle_Body_Type_index','Vehicle_Make_index','Issuing_Agency_index','Street_Code1_index','Street_Code2_index','Street_Code3_index','
X_pandas = df_pandas[column_names].values
y_pandas = df_pandas['label'].values
X_train, X_test, y_train, y_test = train_test_split(X_pandas, y_pandas, test_size = 0.2)
xgboost_model = XGBClassifier(max_depth=7, n_estimators=100, objective='multi:softprob')
xgboost_model.fit(X_train, y_train)
#Testing the model
y_pred_train = xgboost_model.predict(X_train)
print("For train\nAccuracy score:{},Balanced accuracy score:{},f1 score:{}".format(accuracy_score(y_train,y_pred_train),balanced_accuracy_score(y_train,y_pred_train),f1_score(y_train,y_pred_train, average='weighted')))
y_pred_test = xgboost_model.predict(X_test)
print("For test\nAccuracy score:{},Balanced accuracy score:{},f1 score:{}".format(accuracy_score(y_test,y_pred_test),balanced_accuracy_score(y_test,y_pred_test),f1_score(y_test,y_pred_test, average = 'weighted')))
#For saving model
filename = 'XGB_final_model_v1.pkl'
pickle.dump(xgboost_model, open(filename, 'wb'))
```

```
For train
Accuracy score:0.9993125,Balanced accuracy score:0.9986929698062774,f1 score:0.9993124867241515
For test
Accuracy score:0.99865,Balanced accuracy score:0.9966405271390689,f1 score:0.9986498110888233
```

Fig 5. Checking accuracy scores for XGBoost model

```
#Defining XGBOOST prediction UDF
@f.pandas_udf(returnType=DoubleType())
def predict_pandas_udf(*cols):
    # cols will be a tuple of pandas.Series here.
    X = pd.concat(cols, axis=1)
    return pd.Series(xgboost_model.predict(np.array(X)))
#Get the transformed data
df_full = model.transform(df_new)
#Apply XGBOOST on it
df_r3 = df_full.withColumn('prediction_xgb',predict_pandas_udf(*column_names))
```

Fig 6: Defining UDF for XGBoost prediction

```
################################################################ TESTING THE MODEL PERFORMANCE ##########################################################
#Train set for random forest
#Create an evaluation object for the model using the R^2 metric
lr_evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol="label",metricName="accuracy")
#Print the Evaluation Result
print("Train accuracy for Random forest:{}\n".format(lr_evaluator.evaluate(df_r1)))

#Test set for random forest
df_test = model.transform(test)
#Create an evaluation object for the model using the R^2 metric
lr_evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol="label",metricName="accuracy")
#Print the Evaluation Result
print("Test accuracy for Random forest:{}\n".format(lr_evaluator.evaluate(df_test)))

#Test set for XGBOOST
#Create an evaluation object for the model using the R^2 metric
lr_evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction_xgb", labelCol="label",metricName="accuracy")
#Print the Evaluation Result
print("Test accuracy for XGBOOST:{}\n".format(lr_evaluator.evaluate(df_r3)))
```

```
Train accuracy for Random forest:0.9454161301650474

Test accuracy for Random forest:0.9439258191842617

Test accuracy for XGBOOST:0.9948276087411125
```

Fig 7. Accuracy scores of XGBoost and RandomForest

```
In [7]:
#Test set for random forest
df_test = model.transform(test)
#Create an evaluation object for the model using the R^2 metric
lr_evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol="label",metricName="accuracy")
#Print the Evaluation Result
print("Test accuracy for Random forest:{}\n".format(lr_evaluator.evaluate(df_test)))

Test accuracy for Random forest:0.950346995301
```

Fig 8. Accuracy for Random Forest

➢ Real-time computation and latency of processing each window

Even though XGBoost outperformed Random forest by a whopping 4% accuracy increase. However, in real-time, we were facing an issue of loading the saved model of XGBoost from Google Cloud Storage. Hence we have used only Random Forest for real-time computation.

```
import os
from google.cloud import pubsub_v1
import time
from google.cloud import storage

# publisher = pubsub_v1.PublisherClient.from_service_account_json("./gcloudcredentials.json")
publisher = pubsub_v1.PublisherClient()
topic_name = 'projects/big-data-lab-266809/topics/{topic}'.format(
    project_id=os.getenv('GOOGLE_CLOUD_PROJECT'),
    topic='to-kafka',  # Set this to something appropriate.
)

#publisher.create_topic(topic_name)
# client = storage.Client.from_service_account_json("./gcloudcredentials.json")
client = storage.Client()

bucket = client.get_bucket("ch16b024")
blob = bucket.get_blob("small_temp.csv")
print("Loading data...")
x = blob.download_as_string()
x = x.decode('utf-8')
data = x.split('\n')
print("Done. Pushing data to kafka server...")
for lines in data[1:]:
    if len(lines)==0:
        break
    #Sleeps for 10 seconds
    time.sleep(10)
    publisher.publish(topic_name, lines.encode(), spam=lines)
```

Fig 7. Publish function

```python
kafka_topic = 'from-pubsub'
zk = '10.138.0.3:2181'
app_name = 'from-pubsub' # Can be some other name
sc = SparkContext(appName="KafkaPubsub")
ssc = StreamingContext(sc, 30)
sc.setLogLevel("FATAL")

kafkaStream = KafkaUtils.createStream(ssc, zk, app_name, {kafka_topic: 1})


def getSparkSessionInstance(sparkConf):
    if ("sparkSessionSingletonInstance" not in globals()):
        globals()["sparkSessionSingletonInstance"] = SparkSession \
            .builder \
            .config(conf=sparkConf) \
            .getOrCreate()
    return globals()["sparkSessionSingletonInstance"]
```

Fig 8a. For Real-time implementation using Kafka stream

```python
#     return pd.Series(xgboost_model.predict(np.array(x)))
udfValueToCategory = udf(valueToCategory, StringType())
udfaccuracy = udf(accuracy_calc,IntegerType())
time_udf = udf(time_bucket,IntegerType())
day_udf = udf(day_finder,IntegerType())
print("############################################ START ##############################################################

lines = kafkaStream.map(lambda x: json.loads(x[1])["spam"]).map(lambda x: x.split(","))
#Loading pipeline
model = PipelineModel.load('gs://ch16b024/model_finalproject_v1/')
#Considered columns
column_names = ['Registration_State_index','Plate_Type_index','Violation_Code_index',
'Vehicle_Body_Type_index','Vehicle_Make_index','Issuing_Agency_index','Street_Code1_index',
'Street_Code2_index','Street_Code3_index','Issuer_Precinct_index','Issuer_Command_index',
'Violation_In_Front_Of_Or_Opposite_index','Violation_County_index','Month','Day','Time']
#Loading xgboost model
# xgboost_model = pkl.load(open("gs://ch16b024/XGB_final_model_v1.pkl", "rb"))
accuracy = 0
completed = 0
def process(rdd):
        start = time.time()
        global accuracy
        global completed
        # Get the singleton instance of SparkSession
        spark = getSparkSessionInstance(rdd.context.getConf())
        # Convert RDD[String] to RDD[Row] to DataFrame
        rowRdd = rdd.map(lambda x: Row(Summons_Number=str(x[0]),Registration_State=str(x[2]),Plate_Type=str(x[3]),
        Violation_Code=str(x[5]),Vehicle_Body_Type=str(x[6]),Vehicle_Make=str(x[7]),
        Issuing_Agency=str(x[8]),Street_Code1=str(x[9]),Street_Code2=str(x[10]),
        Street_Code3=str(x[11]),Violation_County=str(x[13]),
        Issuer_Precinct=str(x[14]),Issuer_Command=str(x[16]),Issuer_Squad=str(x[17]),
        Violation_In_Front_Of_Or_Opposite=str(x[21]),Issue_Date=str(x[4]),
        Violation_Time=str(x[18]),Violation_Location=str(x[20])))
        df = spark.createDataFrame(rowRdd)
```

Fig 8b. For Real-time implementation using Kafka stream job-project29

Start time: Aug 1, 2020, 7:00:11 PM Elapsed time: 21 min 0 sec Status:

Output Configuration

Line wrapping Equivalent command line

```
|         T101|              5|              T|       PAS|                NY|       16410|       13460|       25890|    7047383785|             SUBN|       LEXUS|              16|
|         0066|             66|              P|       PAS|                NY|           0|           0|           0|    1372046290|             SUBN|       HONDA|              46|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-

Labels correct till now:111/115
Completed batch of 3 in 4.16602110863sec
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|Issuer_Command|Issuer_Precinct|Issuing_Agency|Plate_Type|Registration_State|Street_Code1|Street_Code2|Street_Code3|Summons_Number|Vehicle_Body_Type|Vehicle_Make|Violation_Code|V:
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|         T501|              1|              T|       PAS|                NY|       15350|       24750|       10510|    7818597608|             SUBN|       NISSA|              17|
|         T401|            108|              T|       PAS|                PA|        9140|       68290|       11390|    7008624748|             4DSD|       VOLVO|              40|
|         T401|            115|              T|       PAS|                NY|       16740|        8590|        8790|    7613059390|             4DSD|       HONDA|              38|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-

Labels correct till now:113/118
Completed batch of 3 in 4.7607178688sec
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|Issuer_Command|Issuer_Precinct|Issuing_Agency|Plate_Type|Registration_State|Street_Code1|Street_Code2|Street_Code3|Summons_Number|Vehicle_Body_Type|Vehicle_Make|Violation_Code|V:
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|         T303|            120|              T|       PAS|                NC|       44010|       49876|       17450|    7163117346|             SUBN|        FORD|              20|
|         T201|             52|              T|       PAS|                CT|       42820|       71220|       76790|    7773113440|             2DSD|       HONDA|              14|
|         0063|             63|              P|       PAS|                NY|       13980|       36830|       36880|    1381130574|              SDN|       NISSA|              14|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
```

Fig 9. 118 samples completed after 21 min

job-project29

Start time: Aug 1, 2020, 7:00:11 PM Elapsed time: 37 min 39 sec Status:

Output Configuration

Line wrapping Equivalent command line

```
|         T401|            108|              T|       PAS|                NJ|       59990|        9790|        9890|    7002227452|             TR/C|        FORD|              38|
|         T401|            108|              T|       PAS|                NJ|       59990|        9790|        9890|    7002227452|             TR/C|        FORD|              38|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-

Labels correct till now:205/214
Completed batch of 3 in 4.82632708549sec
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|Issuer_Command|Issuer_Precinct|Issuing_Agency|Plate_Type|Registration_State|Street_Code1|Street_Code2|Street_Code3|Summons_Number|Vehicle_Body_Type|Vehicle_Make|Violation_Code|V
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|         T301|             73|              T|       PAS|                NY|       73470|       70930|       18830|    7134344770|             4DSD|       NISSA|              38|
|         T302|             63|              T|       PAS|                NY|       37460|       14630|       82480|    7104778330|             4DSD|       TOYOT|              46|
|         T401|            112|              T|       PAS|                NY|       68690|       41490|       41460|    8024520813|             4DSD|       CHEVR|              38|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-

Labels correct till now:208/217
Completed batch of 3 in 4.6467320919sec
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|Issuer_Command|Issuer_Precinct|Issuing_Agency|Plate_Type|Registration_State|Street_Code1|Street_Code2|Street_Code3|Summons_Number|Vehicle_Body_Type|Vehicle_Make|Violation_Code|V
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
|         T105|              5|              T|       PAS|                NJ|       32900|       24390|       14590|    7627892790|             4DSD|       TOYOT|              37|
|         T401|            108|              T|       PAS|                NY|       10540|       10440|       10590|    8024605788|             4DSD|       SUBAR|              20|
|         T401|            108|              T|       PAS|                NY|       10540|       10440|       10590|    8024605788|             4DSD|       SUBAR|              20|
+-------------+---------------+---------------+----------+------------------+------------+------------+------------+--------------+-----------------+------------+----------------+-
```

Fig 10. 217 samples completed after 38 min

38 min/217samples ≅ 0.175 min per sample
21min/118 samples ≅ 0.177 min per sample

➢ Conclusion

1) XGBoostClassifier and RandomForestClassifier both perform well (99.9% and 95% accuracy respectively) for multi-classification when feature space is not changed significantly

2) XGBoost performed faster than Random Forest
3) LogisticRegressionClassifier doesn't perform well for multiclass regression unless we do kernel feature engineering.

Reason of why tree-based methods perform well than logistic regression:
This happens because Tree-based methods bisect the space into several smaller spaces and can, therefore, classify each point correctly given enough tree depth. Whereas logistic regression uses single line space divider and therefore cannot get 100% accuracy even during training if feature space is not transformed

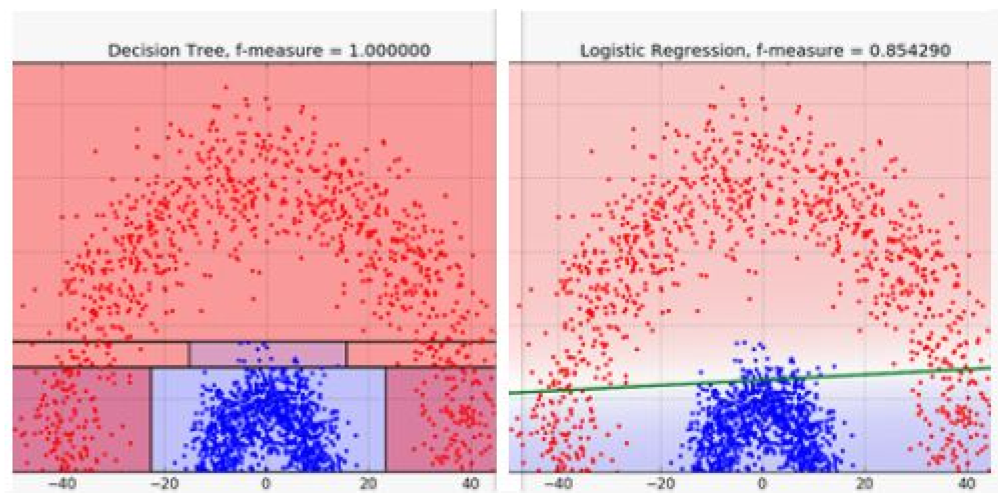

Fig11. Showing 100% accuracy for tree-based classifier whereas logistic can perform best at F-measure= 0.854

4) Constant latency time for the batch of samples was observed. That is time increases linearly by numbers of samples to evaluate. As streaming of data happens in constant time and processing is fast compared to it.

5) Feature Importance graph clearly shows the Violation County Index being the most important feature as each county has a unique index and location(region) that can be guessed correctly based on the county index.

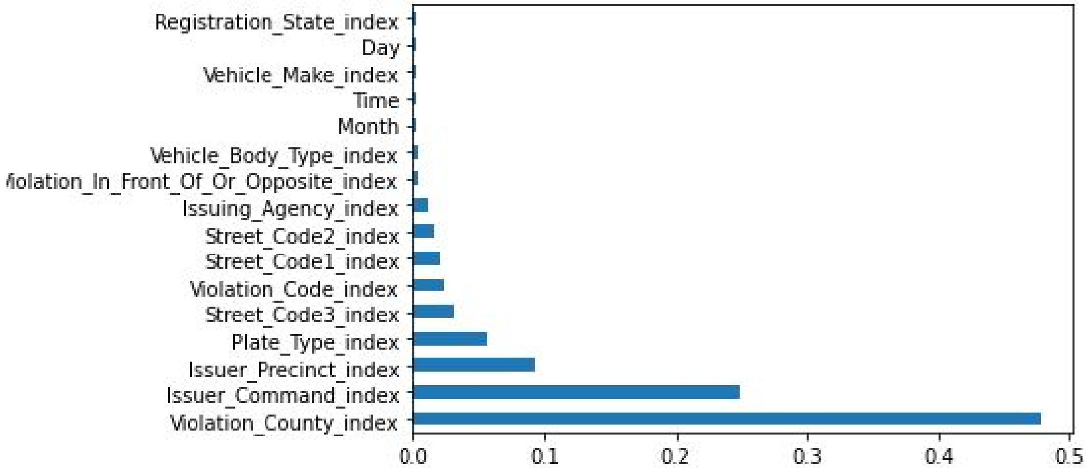

Fig 12. Feature Importance