Final Project Report - NYC Parking Tickets

CS4830: Big Data Laboratory

Sushant Uttam Wadavkar (ME16B172)
Department of Mechanical Engineering
Indian Institute of Technology Madras
Chennai, TN 600036, India
me16b172@smail.iitm.ac.in

Instructor : Prof. Balaraman Ravindran.
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, TN 600036, India
ravi@cse.iitm.ac.in

I. Introduction

New York City is a thriving metropolis. Just like most other metros its size, one of the biggest problems its citizens face is parking. The classic combination of a huge number of cars and cramped geography leads to a huge number of parking tickets. In an attempt to scientifically analyse this phenomenon, the NYC Police Department has collected data for parking tickets. Of these, the data files for multiple years are publicly available on Kaggle. We will try and perform some exploratory analysis on a part of this data. Spark will allow us to analyse the full files at high speeds as opposed to taking a series of random samples that will approximate the population. For the scope of this analysis, we will analyse the parking tickets over the year 2017.

The NYC Department of Finance collects data on every parking ticket issued in NYC (nearly 10M per year!). This data is made publicly available to aid in ticket resolution and to guide policymakers.

The purpose of this case study is to conduct an exploratory data analysis that will help you understand the data. Since the size of the dataset is large, your queries will take some time to run, and you will need to identify the correct queries quicker. The questions given below will guide your analysis.

II. Objectives

- To train a learning model on Big Dataset of NYC Parking tickets to predict the violation location.
- To be able to perform real-time computation using the trained model using Big data technologies taught in course CS4830.

III. Broad Information about Dataset

- Number of features (Columns): 50 columns
- Number of Rows/Entries: 1,18,09,233
- 16 features have only NaN entries for all the Rows and can be removed without affecting the model

IV. Data Pre-processing

Data cleaning is an important part of any data science project because anomalies and outliers in the data can negatively influence many statistical analyses. This could lead us to make bad conclusions about the data and build machine learning models that don't stand up over time. Therefore, it's important that we get the data cleaned up as much as possible before moving on to exploratory analysis.

Out[57]:

jistration State	Violation Code	Vehicle Body Type	Vehicle Make	Issuing Agency	Street Code1	Street Code2	Street Code3	...	Vehicle Year	Feet From Curb	Violation Post Code	Violation Description	Plate Type	Vehicle Color	Issue Date	Citation Issued Month Year	Temp	monthYear
NY	46	SUBN	AUDI	P	37250	13610	21190	...	2013.0	0.0	Unknown	Unknown	PAS	GY	2013-08-04	08-2013	74.6	08-2013
NY	46	VAN	FORD	P	37290	40404	40404	...	2012.0	0.0	Unknown	Unknown	COM	WH	2013-08-04	08-2013	74.6	08-2013
NY	46	P-U	CHEVR	P	37030	31190	13610	...	0.0	0.0	Unknown	Unknown	COM	GY	2013-08-05	08-2013	74.6	08-2013
NY	46	VAN	FORD	P	37270	11710	12010	...	2010.0	0.0	Unknown	Unknown	COM	WH	2013-08-05	08-2013	74.6	08-2013
NY	41	TRLR	GMC	P	37240	12010	31190	...	2012.0	0.0	Unknown	Unknown	COM	BR	2013-08-08	08-2013	74.6	08-2013
NJ	14	P-U	DODGE	P	37250	10495	12010	...	0.0	0.0	Unknown	Unknown	PAS	RD	2013-08-11	08-2013	74.6	08-2013
NJ	24	DELV	FORD	X	63430	0	0	...	0.0	0.0	Unknown	Unknown	PAS	WHITE	2013-08-07	08-2013	74.6	08-2013
NY	24	SDN	TOYOT	X	63430	0	0	...	2001.0	0.0	Unknown	Unknown	PAS	WHITE	2013-08-07	08-2013	74.6	08-2013
NY	24	SDN	NISSA	X	23230	41330	83330	...	2012.0	0.0	Unknown	Unknown	PAS	WHITE	2013-08-12	08-2013	74.6	08-2013
NY	20	SDN	VOLKS	T	28930	27530	29830	...	2012.0	0.0	Unknown	Unknown	PAS	WHITE	2013-08-12	08-2013	74.6	08-2013
LA	17	SUBN	HONDA	T	0	0	0	...	0.0	0.0	Unknown	Unknown	PAS	TAN	2013-08-07	08-2013	74.6	08-2013
IL	40	SDN	SCIO	T	26630	40930	18630	...	0.0	6.0	Unknown	Unknown	PAS	BK	2013-08-10	08-2013	74.6	08-2013
PA	20	SDN	TOYOT	T	21130	71330	89930	...	0.0	0.0	Unknown	Unknown	PAS	GR	2013-08-06	08-2013	74.6	08-2013
NY	40	VAN	MERCU	T	23190	27290	20340	...	2003.0	0.0	Unknown	Unknown	COM	RD	2013-08-07	08-2013	74.6	08-2013
NY	51	VAN	TOYOT	X	93230	74830	67030	...	2013.0	0.0	Unknown	Unknown	PAS	GY	2013-08-06	08-2013	74.6	08-2013

Fig. 1: Data Pre-processing Output

- **Duplicate Summons Numbers Removal**
  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. Every feature was inspected manually. It has been observed that 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 were removed. These features were 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.

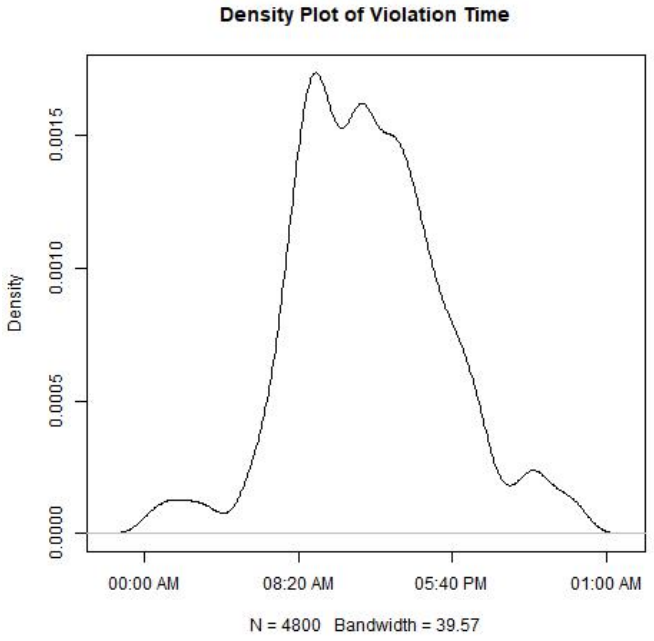

Fig. 2: Violation Time Variations

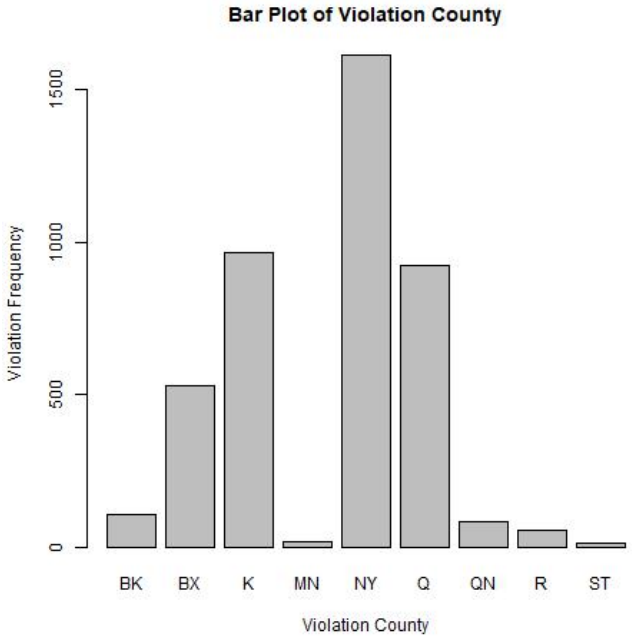

Fig. 3: Violation County Variations

Henceforth, in **total 16+17 = 33 features** in total were removed during this phase.

- **Correction in Violation Location**
  All violations Locations with Nan values were dropped as this was the target column and we cannot have supervised learning without labels. Some of the target labels have been names differently. Hence, the differently named labels were mapped to the actual prominent label as shown below.
  It's needed to map cases like Manhattan 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, it had been hence mapped accordingly.
- **Time for Bucketing - Using 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.
- **Issue Date - Using Feature Engineering**
  The respective columns are also converted to integers. We have also added a column, Day of the week which is calculated form year.
  The year column is 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.
- **Label Encoding**

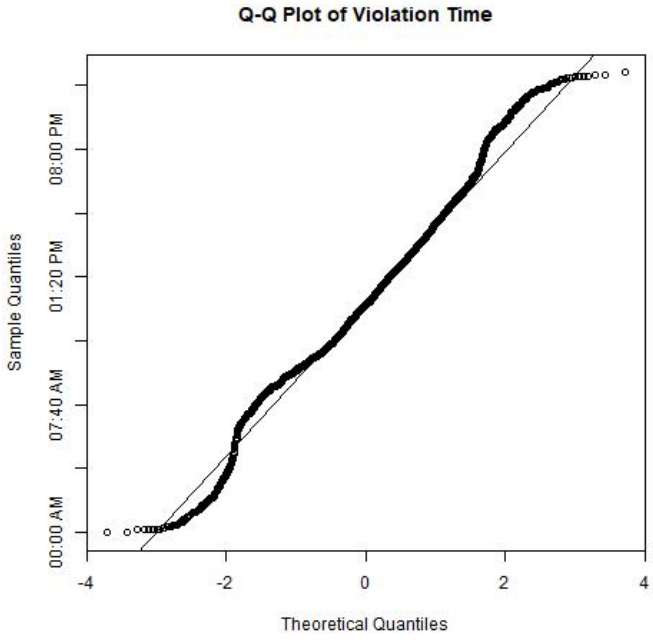

Fig. 4: Violation Time Variations

All string/ text features are encoded using (StringIndexer) label encoding.
Eg: Registration States are Mapped as NY: 0, TX: 7 in (StringIndexer).

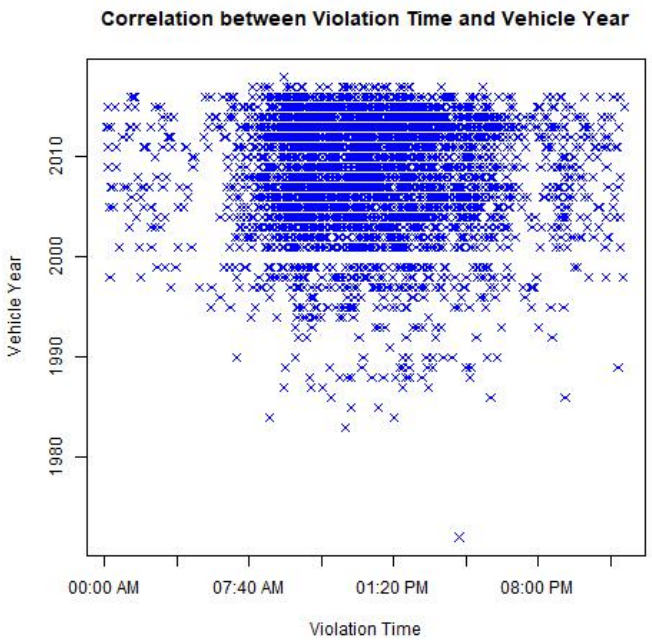

Fig. 5: Violation Time - Vehicle Year Variations

V. Feature Selection

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons: simplification of models to make them easier to interpret by researchers/users, shorter training times, to avoid the curse of dimensionality, enhanced generalization by reducing overfitting.

The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information.[2] Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Feature selection techniques should be distinguished from feature extraction. Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features. Feature selection techniques are often used in domains where there are many features and comparatively few samples (or data points).

VI. Model Building

Features had been identified. A simple Machine Learning model was built using Logistic and Lasso-Ridge Regression techniques.

Pyspark was used to operate on the entire dataset.

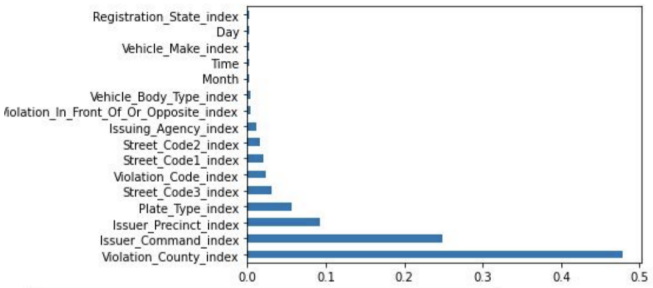

Fig. 6: Order - Importance of Features

//Code: ...columns = [”Summons Number”, ”Registration State”, ”Issuing Agency”, ”Street Code1”, ”Street Code2”, ”Street Code3”, ”Violation Location”]

...data = data.select(columns)

...train, test = data.randomSplit([0.8,0.2]) //

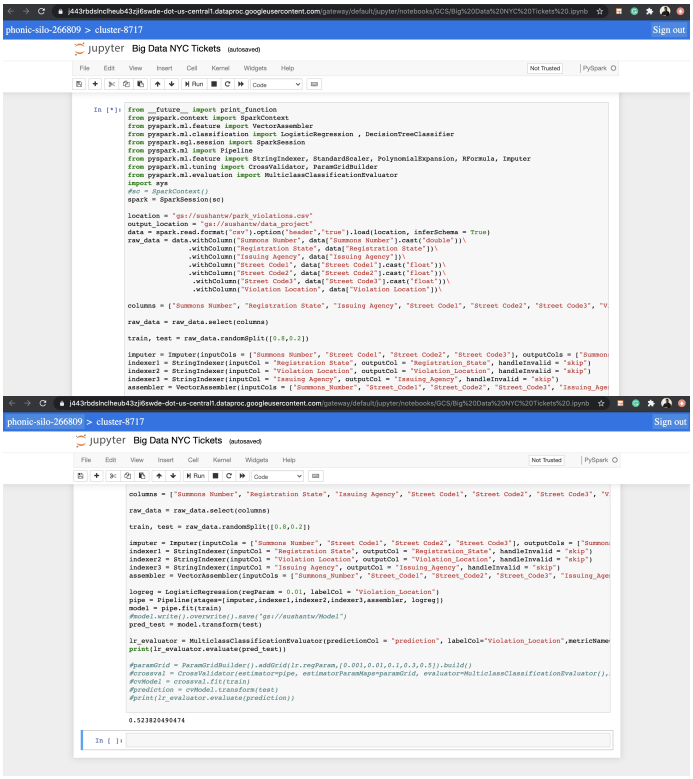

Fig. 7: Model Code Snippet

Using (Pipeline) command we build the model and fitted it using training data. We later then saved the model in the GCP Bucket. Accuracy of 52.382 was recorded.

We can get the best accuracy with the XGBoost Classifier aroud 99.90. Also, XGBoost can be trained pretty faster than Random Forest.

VII. Setting Up Kafka

Kafka is a distributed event streaming platform that lets you read, write, store, and process events (also called records or messages in the documentation) across many machines.

Example events are payment transactions, geolocation updates from mobile phones, shipping orders, sensor measurements from IoT devices or medical equipment, and much more. These events are organized and stored in topics. Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder.

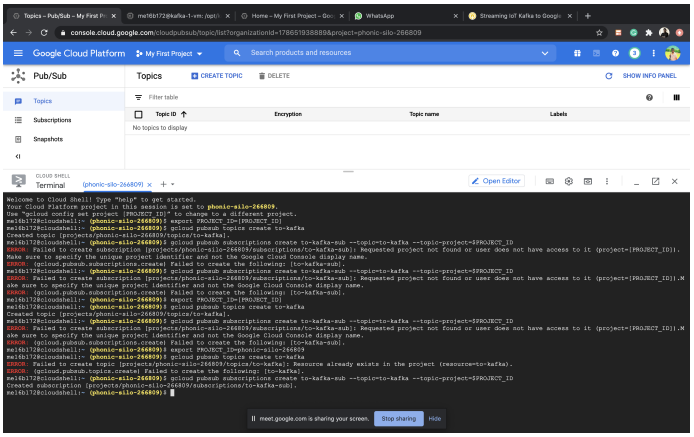

Fig. 8: Cloud Shell Terminal Snippet

A Kafka client communicates with the Kafka brokers via the network for writing (or reading) events. Once received, the brokers will store the events in a durable and fault-tolerant manner for as long as we need—even forever. When we have lots of data in existing systems like relational databases or traditional messaging systems, along with many applications that already use these systems. Kafka Connect allows you to continuously ingest data from external systems into Kafka, and vice versa. It is thus very easy to integrate existing systems with Kafka. To make this process even easier, there are hundreds of such connectors readily available.

Once data is stored in Kafka as events, we can process the data with the Kafka Streams client library for Java/Scala. It allows us to implement mission-critical real-time applications and microservices, where the input and/or output data is stored in Kafka topics. Kafka Streams combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the benefits of Kafka's server-side cluster technology to make these applications highly scalable, elastic, fault-tolerant, and distributed. The library supports exactly-once processing, stateful operations and aggregations, windowing, joins, processing based on event-time, and much more.

Successfully setup the Kafka platform for the project.

VIII. Kafka Streaming

Kafka works well as a replacement for a more traditional message broker. Message brokers are used for a variety of reasons (to decouple processing from data producers, to buffer unprocessed messages, etc). In comparison to most messaging systems Kafka has better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution

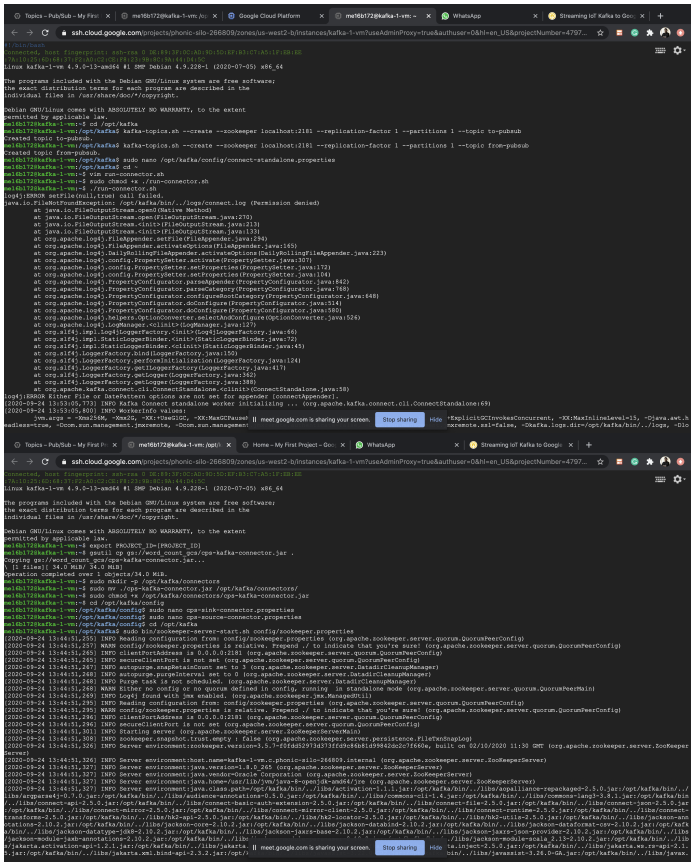

Fig. 9: Kafka Setup Snippet

for large scale message processing applications. In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency and often depend on the strong durability guarantees Kafka provides.

In this domain Kafka is comparable to traditional messaging systems such as ActiveMQ or RabbitMQ.

```
import io

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


publisher = pubsub_v1.PublisherClient()
topic_name = 'projects/
topic_path = publisher.topic_path('phonic-silo-266809-47f66d1424ac','to-kafka')
client = storage.Client()
bucket = client.get_bucket('sushantw')
blob = bucket.get_blob('small_temp.csv')
print("Loading the Data...")
x = blob.download_as_string()
x = x.decode('utf-8')
x = x.split('\n')

print("Done. Pushing data to the Kafka Server")
for lines in data[I:]:
        if len(lines)==0:
                break
        time.sleep(10)
        publisher.publish(topic_name, lines.encode(), spam=lines)
```

Fig. 10: Publisher Code Snippet

Many users of Kafka process data in processing pipelines consisting of multiple stages, where raw input data is consumed from Kafka topics and then aggregated, enriched, or

otherwise transformed into new topics for further consumption or follow-up processing. For example, a processing pipeline for recommending news articles might crawl article content from RSS feeds and publish it to an ”articles” topic; further processing might normalize or deduplicate this content and publish the cleansed article content to a new topic; a final processing stage might attempt to recommend this content to users. Such processing pipelines create graphs of real-time data flows based on the individual topics. Starting in 0.10.0.0, a light-weight but powerful stream processing library called Kafka Streams is available in Apache Kafka to perform such data processing as described above. Apart from Kafka Streams, alternative open source stream processing tools include Apache Storm and Apache Samza.

```
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"]
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. 11: Kafka Streaming In Real-Time: Code Snippet

IX. Conclusions

- Model Performances can be significantly increased using different ML Techniques.
- LogisticRegressionClassifier doesn't perform well for multiclass regression unless we do kernel feature engineering.
- 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.
- Constant latency time for the batch of samples was observed. That is time increases linearly by numbers of samples to evaluate.
- Reasons tree-based methods could perform better 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
- Successfully used Kafka platform for streaming the data in real-time through Google Cloud Platform and tested the machine learning model on the NYC Parking Tickets Dataset.
- As streaming of data happens in constant time and processing is fast compared to it.

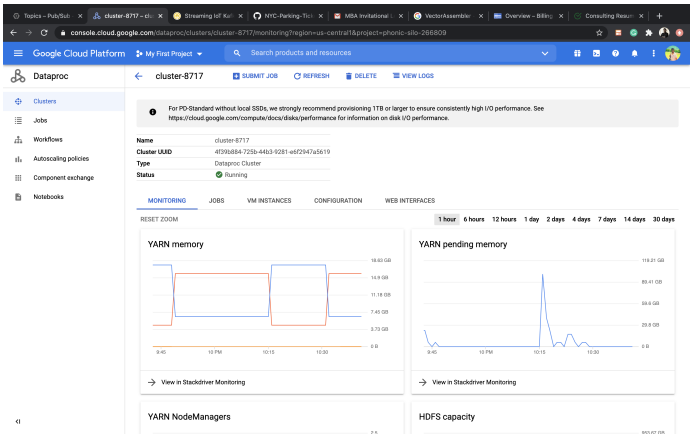

Fig. 12: Cluster Stackdriver Monitoring

X. Acknowledgement

I would like to thank Professor Ravindran. More importantly the Teaching Assistants of this course for clarifying my doubts and the entire class of CS4830: Big Data Laboratory (January - May 2020) for their guidance throughout this project. I would also like to thank Computer Sciences Department at Indian Institute of Technology Madras.