Credit card Fraud Detection

Student : Jagadish Janakiraman

Question : Explanation of the solution to the batch layer problem in detail should be provided properly in a document.

- **Task 5**: Create a streaming data processing framework that ingests real-time POS transaction data from Kafka. The transaction data is then validated based on the three rules' parameters (stored in the MongoDB database) discussed previously.
- **Task 6**: Update the transaction data along with the status (fraud/genuine) in the card_transactions table.
- **Task 7**: Store the "postcode" and "transaction_dt" of the current transaction in the look-up table in the NoSQL database if the transaction was classified as genuine.

Function used to calculate distance between two different latitude and longitude. Zip code, lat/long mapping is provided in the uszipsv.csv file.

In [6]:

```
class GEO_Map():
    """
    It hold the  map for zip code and its latitute and longitute
    """
    __instance = None

    @staticmethod
    def get_instance():
        """ Static access method. """
        if GEO_Map.__instance == None:
            GEO_Map()
        return GEO_Map.__instance

    def __init__(self):
        """ Virtually private constructor. """
        if GEO_Map.__instance != None:
            raise Exception("This class is a singleton!")
        else:
            GEO_Map.__instance = self
            self.map = pd.read_csv("uszipsv.csv", header=None, names=['A',"B",'C','D','E'])
            self.map['A'] =  self.map['A'].astype(str)

    def get_lat(self, pos_id):
        #print("printing value from class ", self.map[self.map.A == pos_id ].B)
        return self.map[self.map.A == pos_id ].B

    def get_long(self, pos_id):
        return self.map[self.map.A == pos_id ].C

    def distance(self, lat1, long1, lat2, long2):
        theta = long1 - long2
        dist = math.sin(self.deg2rad(lat1)) * math.sin(self.deg2rad(lat2)) + math.cos(self.deg2rad(lat1)) * math.cos(self.deg
        dist = math.acos(dist)
```

Two rules are validated here - Function used to check if amount is less than UCL and UCL greater than credit score of 200

In [7]:

```
# Function to check rules for UCL and Credit Score
def verify_ucl_data(card_id, amount):
    try:
        client = MongoClient();
        # Database Name
        db = client["CREDIT_CARD_DB"]
        # Collection Name
        lookupTable = db["lookup_table"]
        lookupValue = lookupTable.find_one({'card_id': card_id}) #378303738095292
        if amount < float(lookupValue["ucl"]) and float(lookupValue["ucl"]) > 200:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

Function used to calculate distance between two given zip codes. Verify if the speed is not greater than 900 km/hr. This rule is used to determine if a transaction is fraud or genuine.

```
In [8]:
"""
Function to verify the following zipcode rules
ZIP code distance
:param card_id: (Long) Card id of the card customer
:param postcode: (Integer) Post code of the card transaction
:param transaction_dt: (String) Timestamp
:return: (Boolean)
"""
def verify_postcode_data(card_id, postcode, transaction_dt):

    try:

        client = MongoClient();
        # Database Name
        db = client["CREDIT_CARD_DB"]
        # Collection Name
        lookupTable = db["lookup_table"]
        lookupValue = lookupTable.find_one({'card_id': card_id}) #378303738095292
        geo_map = GEO_Map.get_instance()
        last_postcode = lookupValue["postcode"]
        last_transaction_dt = lookupValue["transaction_dt2"]
        current_lat = geo_map.get_lat(str(postcode))
        for data in current_lat:
            current_lat1 = data
        current_lon = geo_map.get_long(str(postcode))
        for data in current_lon:
            current_lon1 = data
        previous_lat = geo_map.get_lat(str(last_postcode))
        for data in previous_lat:
            previous_lat1 = data
        previous_lon = geo_map.get_long(str(last_postcode))
        for data in previous_lon:
            previous_lon1 = data

        dist = geo_map.distance(lat1=current_lat1, long1=current_lon1, lat2=previous_lat1, long2=previous_lon1)
        speed = calculate_speed(dist, transaction_dt, last_transaction_dt)

        if speed < speed_threshold:
            return True
        else:
            return False

    except Exception as e:
        raise Exception(e)
```

```
In [9]:
#A function to calculate the speed from distance and transaction timestamp differentials

def calculate_speed(dist, transaction_dt1, transaction_dt2):
    transaction_dt1 = datetime.datetime.strptime(transaction_dt1, '%d-%m-%Y %H:%M:%S')
    transaction_dt2 = datetime.datetime.strptime(transaction_dt2, '%Y-%m-%d %H:%M:%S')
    elapsed_time = transaction_dt1 - transaction_dt2
    elapsed_time = elapsed_time.total_seconds()
    try:
        return dist / elapsed_time
    except ZeroDivisionError:
        return 299792.458  # (Speed of light)
```

1. updateCardTransactions function is used to insert record into the collection with Genuine or Fraud Status.
2. updateLookUpTransaction function is used to update card_id with Genuine transactions only
3. validateFraud is a wrapper function that checks the 3 rules

In [10]:

```
#A function to update genuine and fraud transactions into card_transactions collections

def updateCardTransactions(kafkajsonObj, Status):
    client = MongoClient();
    # Database Name
    mydb = client["CREDIT_CARD_DB"]
    mycol = mydb["card_transactions"]
    if (Status==True):
        newStatus = "GENUINE"
    else:
        newStatus = "Fraud"

    mydict = {"card_id" : kafkajsonObj["card_id"], "member_id" : kafkajsonObj["member_id"], "amount" : kafkajsonObj["amount"]
    mycol.insert_one(mydict)
```

In [15]:

```
# A function to update genuine transactions into lookup_table collections
def updateLookUpTransactions(kafkajsonObj):
    client = MongoClient();
    # Database Name
    mydb = client["CREDIT_CARD_DB"]
    mycol = mydb["lookup_table"]
    lookupUpdQryOld = {"card_id" : kafkajsonObj["card_id"]}
    lookupUpdQryNew = { "$set": {"card_id" : kafkajsonObj["card_id"], "member_id" : kafkajsonObj["member_id"], "amount" : kaf
    mycol.update_one(lookupUpdQryOld, lookupUpdQryNew)
```

In [13]:

```
# wrapper function to validate all 3 rules
def validateFraud(amount, card_id, postcode, txndate):
    status_ucl_crdScore = verify_ucl_data(card_id, amount)
    status_distance = verify_postcode_data(card_id, postcode, txndate)
    if status_ucl_crdScore==True and status_distance==True :
        #print("3 Rules check passed!!!!!! ; congratulations! ")
        return True
    else :
        return False
```

1. updateCardTransactions function is used to insert record into the collection with Genuine or Fraud Status.
2. updateLookUpTransaction function is used to update card_id with Genuine transactions only
3. validateFraud is a wrapper function that checks the 3 rules

```python
from kafka import KafkaConsumer
import sys

### Setting up the Python consumer
bootstrap_servers = ['18.211.252.152:9092']
topicName = 'transactions-topic-verified'
consumer = KafkaConsumer (topicName, group_id = 'my_group_id12',bootstrap_servers = bootstrap_servers,
auto_offset_reset = 'earliest')   ## You can also set it as latest
i=0
### Reading the message from consumer
try:
    for message in consumer:
        kafkajsonObj = json.loads(message.value)
        status = validateFraud(kafkajsonObj["amount"], kafkajsonObj["card_id"],kafkajsonObj["postcode"], kafkajsonObj["transa
        updateCardTransactions(kafkajsonObj, status)
        print("Record inserted in transactions table")
        if (status==True):
            updateLookUpTransactions(kafkajsonObj)
            print("Record updated in lookup table")

except KeyboardInterrupt:
    sys.exit()

import datetime
now = datetime.datetime.now()
print("Current date and time: ")
print(str(now))
```

```
Record inserted in transactions table
Record updated in lookup table
Record inserted in transactions table
Record updated in lookup table
Record inserted in transactions table
Record updated in lookup table
Record inserted in transactions table
Record updated in lookup table
Record inserted in transactions table
Record updated in lookup table
Record inserted in transactions table
Record updated in lookup table
```

It took ~10 minutes to process ~6075 records from Kafka

```
> show collections
card_transactions
lookup_table
> db.card_transactions.count()
53292
> db.card_transactions.count()
53292
> db.card_transactions.count()
53303
> date
uncaught exception: ReferenceError: date is not defined :
@(shell):1:1
> Date()
Sun Dec 04 2022 00:43:48 GMT-0500 (Eastern Standard Time)
> db.card_transactions.count()
53640
> db.card_transactions.count()
59367
> Date()
Sun Dec 04 2022 00:56:04 GMT-0500 (Eastern Standard Time)
> db.card_transactions.count()
59367
>
```

- 59367 records exist in card_transactions table
- 59260 of them are genuine transactions

```
> db.card_transactions.count()
59367
> db.card_transactions.find({"status" :"GENUINE"}).count()
59260
>
```

Thank You