Project Documentation

CREDIT CARD FRAUD DETECTION SYSTEM

Team Members

Divya Sahu
Naveen Prakash
Rajkumar Rathinam
Ronald Jose
Sambhaji Shinde
Wasim Ahmad

TABLE OF CONTENTS

Page

Abstarct

The use of credit cards is prevalent in modern day society. But it is obvious that the number of credit card fraud cases is constantly increasing in spite of the chip cards worldwide integration and existing protection systems. This is why the problem of fraud detection is very important now.

Credit card fraud detection is the most frequently occurring problem in the present world. This is due to the rise in both online transactions and e-commerce platforms.

Credit card fraud generally happens when the card was stolen for any of the unauthorized purposes or even when the fraudster uses the credit card information for his use.

In the present world, we are facing a lot of credit card problems. To detect the fraudulent activities the credit card fraud detection system was introduced.

This project aims to focus mainly on validating various Business Rules to Identify whether the transaction happened is Fraud/Genuine and report the same accordingly.

Problem statement

With the increasing digitalization and online transactions, It becomes ever so important for the credit card companies to be able to recognize “genuine” and “fraudulent” transactions in order to provide their customers with a more secure and a seamless experience.

As a big data engineer, we should architect and design a solution using all the technologies learnt during this program to meet the following requirements:

1.Detect fraudulent transactions at the shortest possible time (Since the transactions are happening in real time, timing constraint plays a very important role). Whenever a card member swipes his/her card for payment, the transaction should be classified as fraudulent / authentic based on a set of predefined rules.

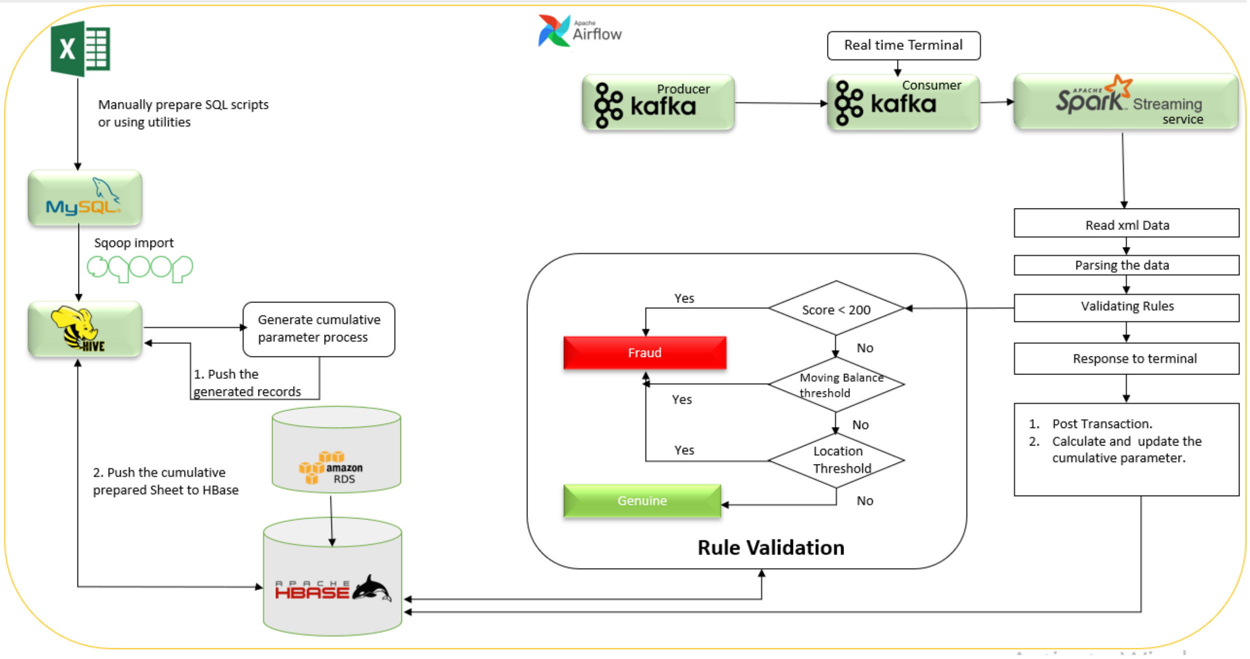
2. To resolve the customer complaints and queries, the support team should be made available with the latest customer details(by constantly keeping them updated.)

Process is broadly divided into two categories

a) Batch Processing – Loading Historical Card Transaction Data into Hive-HBASE table, loading AWS-RDS files of Member Score and Member Details to Hive Tables and populating Card Transactions and Card Lookup table using Airflow Batch Job

b) Stream Processing – Reading Streaming Data from Kafka, pass it to Spark for joining with Master Data and Mark the transaction as Genuine/Fraud accordingly by Validating Various Business Rules

High level Architectural design solution



Datasets-

Card Transaction History Data – Excel File

Member Score Dataset – AWS RDS

Member Details Dataset – AWS RDS

AWS RDS Credentials for Datasets “member_details” and “member_score”

Hostname:database-2.cl4c0rtglkdz.ap-south-1.rds.amazonaws.com

Username: admin

Password: Bankingprj1

Database: BankingPrj

Tables: member_details and member_score

JSON Streaming Live Data – Kafka Producer

Sample-

{ “card_id” : 487654323445689, “member_id” : 000987654123456, “amount” : 245600, “pos_id” : 78765324158934, “postcode” : 33946, “transaction_dt” : 09-01-2021 18:00:00 }

Task Segregation-

Task 1- Copy "card_transactions.csv" file from local system to HDFS.

Table creation tasks-

Task 2- Create the "card_transactions" table in MySQL based on the card_transactions.csv file structure.

Task 3- Do a sqoop export to the database for card_transactions.csv and delete the file from HDFS.

Task 4- On "member_score" and "member_details" create a normal hive external table.

Task 5- Create a special "card_transactions" Hbase table managed by Hive.

Task 6- Create a Hbase "lookup" table with columns - member_id, card_id, UCL, timestamp, zipcode, credit_score.

Batch Processing tasks-

Task 7- Sqoop import member_score from AWS-RDS to Hive. (Full load import, has to be refreshed every week)

Task 8- Sqoop import member_details from AWS-RDS to Hive. (Incremental load import in append mode based on member_id for every 8hrs)

Task 9- Sqoop import card_transactions to HDFS from MySQL. (This is a one-time full load activity. The card_transactions table will be updated with new transactions while in streaming mode.)

Scheduling tasks-

Task 10- Schedule a sqoop import job using Airflow to import member_score from AWS-RDS to Hive on a full-load.

Task 11- Schedule a sqoop import job using Airflow to import member_details from AWS-RDS to Hive on an incremental append mode for every 8hrs.

Integration tasks-

Task 12- Spark-HBase Integration

a) For populating the card_transactions table.

b) For populating the look_up table.

Task 13- Spark-Hive Integration for spark stream processing.

Task 14- Access the hive tables using apache spark and calculate the UCL.

Streaming tasks-

Task 15- Producer to create the transactions in JSON format, to be added and queued in Kafka topics.

Task 16- Spark structured streaming program as a Consumer that will consume the data from the kafka topics.

Task 17- Retrieve the timestamp and zipcode of the last transaction of each card.

Task 18- Processing in Spark Streaming -

Task 18.1 Validating RULE 1 -> “credit_score > 200”

Task 18.2 Validating RULE 2 -> “transaction amount <= UCL”

Task 18.3 Validating RULE 3 -> “zipcode distance within threshold”

Task 19- Based on the above rules, the entire transaction along with status should be updated in the card_transactions table.

Task 20- Schedule a job for validating rules by comparing the incoming data from the POS terminals in JSON format with the values in the lookup table.

Task 21- If the transaction was marked genuine, then we need to update the lookup table with the new timestamp and the zipcode.

Task 22- Schedule a job for populating the lookup table

Setting up the Environment(Prerequisites)-

- Java 1.8
- Scala for Linux
- Compatible Spark
- Airflow Setup for Job Scheduling
  - ✓ Python version 3.5 or above
- Kafka
- Hbase(Make sure RegionServer & Master Services are up and running)

Note- Entire Setup should be in Cloudera even Scala IDE so that we can integrate everything hassle free.

Overall Processing Summary-

Broadly classified into Batch & Streaming Processing

Batch Processing-

- Historical Card Transactions Dataset has to be loaded from Excel Sheet to Local System, from there to HDFS.Further this data will be moved to RDBMS using Sqoop Export Utility and to Hive HBase table.
- Member Score & Member Details data have been loaded to Hive Tables.
- Card Lookup table is generated with Card Transactions and Member Data.

Streaming Processing-

- Data arrives via Kafka Topics and post validating in Kafka based on Business rules ,marked as Fraud/Genuine and posted to card_transactions and card_lookup tables

Development Steps-

Copy “card_transactions.csv” file from local system to HDFS

--Creating Project Folder

```
hadoop fs -mkdir project_input_data
```

--Copying card_transactions csv file from local to cloudera folder

```
hadoop fs -put Desktop/card_transactions_orig.csv project_input_data/
```

--Validating File RowCount

```
hadoop fs -cat project_input_data/card_transactions_orig.csv | wc -l
```

MySQL Staging & Main Tables Creation Steps for loading Card_Transactions History Data

```
create table stg_card_transactions (
card_id bigint,
member_id bigint,
amount int,
postcode int,
pos_id bigint,
transaction_dt varchar(255),
status varchar(50)
);

create table card_transactions (
card_id bigint,
member_id bigint,
amount int,
postcode int,
pos_id bigint,
transaction_dt datetime,
status varchar(50),

PRIMARY KEY(card_id, transaction_dt)
);
```

Sqoop export to the database for card_transactions.csv(Using Airflow) and delete the file from HDFS.

--Encrypting MYSQL Password

```
hadoop credential create mysql.bigdataproject.password -provider jceks://hdfs/user/cloudera/mysql.dbpassword.jceks
```

Sqoop Export for Card Transactions Script

export_card_txns.py

--Verify count

```
select count(*) from stg_card_transactions;
```

--Remove Dups from Stg Table

```
alter ignore table stg_card_transactions

add unique index idx_card_txns (card_id,transaction_dt);
```

--Verify no dups

```
select card_id,transaction_dt,count(*) from stg_card_transactions group by card_id,transaction_dt having count(*) >1;
```

--Dropping index used for removing dups

```
alter table stg_card_transactions drop index idx_card_txns;
```

--Loading main table

```
insert into card_transactions
select card_id,member_id,amount,postcode,pos_id,STR_TO_DATE(transaction_dt,'%d-%m-%Y %H:%i:%s'),status from stg_card_transactions;
commit;
```

--Verify the count

```
select count(*) from card_transactions;
```

--Deleting the file from HDFS

```
hadoop fs -rm /project_input_data/card_transactions.csv
```

Note- We can remove duplicates in HDFS as it give parallelism for bigger files though we did it in MYSQL since our input file was small in size.

Hive MEMBER_SCORE & MEMBER_DETAILS Tables Creation(External Tables and Bucketed Tables)

```
--Enabling Bucketing
SET HIVE.ENFORCE.BUCKETING=TRUE;
```

```
--Member Score External Table
create external table if not exists member_score
(
 member_id string,
 score float
)
row format delimited fields terminated by ','
stored as textfile
location '/project_input_data/member_score/';
```

```
--Member Details External Table
create external table if not exists member_details
(
card_id bigint,
member_id bigint,
member_joining_dt timestamp ,
card_purchase_dt timestamp ,
country string,
city string,
score float
)
row format delimited fields terminated by ','
stored as textfile
location '/project_input_data/member_details/';
```

```
--Member Score Bucketed Table(8 Buckets)
create table if not exists member_score_bucketed
```

```
(
 member_id string,
 score float
)
CLUSTERED BY (member_id) into 8 buckets;
```

```
--Member Details Bucketed Table(8 Buckets)
create table if not exists member_details_bucketed
(
card_id bigint,
member_id bigint,
member_joining_dt timestamp ,
card_purchase_dt timestamp ,
country string,
city string,
score float
)
CLUSTERED BY (card_id) into 8 buckets;
```

Hive-Hbase CARD_TRANSACTIONS Tables Creation(External Tables and Bucketed Tables)

```
--Card_transactions external table

create external table if not exists card_transactions (
card_id bigint,
member_id bigint,
amount float,
postcode int,
pos_id bigint,
transaction_dt timestamp,
status string
)
row format delimited fields terminated by ','
stored as textfile
location '/project_input_data/card_transactions/';
```

--Card_transactions bucketed Hive-HBase table with rowkey on concatenated combination of card_id & transaction_dt columns to get all transactions

```
create table card_transactions_bucketed
(
cardid_txnts string,
card_id bigint,
member_id bigint,
amount float,
postcode int,
pos_id bigint,
transaction_dt timestamp,
status string
)
CLUSTERED by (card_id) into 8 buckets
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH
SERDEPROPERTIES("hbase.columns.mapping"=":key,trans_data:card_id,trans_data:member_id,trans_data:amount,
trans_data:postcode,trans_data:pos_id,trans_data:transaction_dt,trans_data:Status")
TBLPROPERTIES ("hbase.table.name" = "card_transactions");
```

Note- Rowkey is very important in HBase and make sure to choose it in right way

Hive-Hbase CARD_LOOKUP Table Creation(Bucketed Tables)

--Card_lookup Bucketed Hive-HBase table

```
create table card_lookup
(
member_id bigint,
card_id bigint ,
ucl float ,
score float,
last_txn_time timestamp,
last_txn_zip string
)
CLUSTERED by (card_id) into 8 buckets
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH
SERDEPROPERTIES("hbase.columns.mapping"=":key,lkp_data:member_id,lkp_data:ucl,lkp_data:score, lkp_data:last_txn_time,lkp_data:last_txn_zip")
TBLPROPERTIES ("hbase.table.name" = "card_lookup");
```

Sqoop import card_transactions to HDFS from MySQL. (This is a one-time full load activity using Airflow. The card_transactions table will be updated with new transactions while in streaming mode.)

--Airflow Script for importing card transactions data to HDFS card_transactions external table file path from MySQL(Attached)

import_card_txns.py

--Load card_txns_bucketed table with concatenated row key from card_transactions external table

```
insert into table card_transactions_bucketed

select concat_ws('~',cast(card_id as string),cast(transaction_dt as string)) as cardid_txnts,card_id,member_id,amount,postcode,pos_id,transaction_dt,status

from card_transactions;
```

--HBase Search functionality sample based on rowkey

```
scan 'card_transactions', {FILTER => "(PrefixFilter('340028465709212')"}
```

Schedule a sqoop import job using Airflow to import member_score from AWS-RDS to Hive on a full-load.

--Encrypting AWS RDS Password

```
hadoop credential create amazonrds.bigdataproject.password -provider jceks://hdfs/user/cloudera/amazonrds.dbpassword.jceks
```

--Script for connecting to AWS RDS and load data to Hive member score external table file path

member_score.py

--Inserting data into member_score_bucketed table

```
insert into table member_score_bucketed
select * from member_score;
```

Schedule a sqoop import job using Airflow to import member_details from AWS-RDS to Hive on an incremental append mode for every 8hrs.

--Script for connecting to AWS RDS and load data to Hive member details external table file path

member_details.py

--Inserting into member_details_bucketed table

insert into table member_details_bucketed

*select * from member_details;*

Spark-HBase Integration(Batch Job Processing)

- **For populating the card_transactions table.**
- **For populating the look_up table.**

--Below Script does the Hive-Spark & HBase-Spark Integration part, calculates UCL from card_trasactions based on last 10 transactions on the card_id, latest transaction timestamp & postal zip code

Batch_Job

--Final Jar which consists of Batch Job class to be scheduled to run from Airflow

Demo_Final.jar

--Airflow Script to run above Jar, class batch_job

cc_batch_job.py

Producer to create the transactions in JSON format, to be added and queued in Kafka topics.

--Start Kafka

```
cd /home/cloudera/Desktop/Softwares/kafka_2.12-2.6.0/bin
./kafka-server-start.sh ../config/server.properties
```

--Create Topic(In new terminal)

```
cd /home/cloudera/Desktop/Softwares/kafka_2.12-2.6.0/bin
./kafka-topics.sh --create --topic cctxnstopic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

--Kafka Producer

```
./kafka-console-producer.sh --broker-list localhost:9092 --topic cctxnstopic
```

Sample records to pass-

```
{"card_id": 340028465709212,"member_id": 9250698176266,"amount": 900,"pos_id": 4444,"post_code":10101,"transc_dt":"2021-03-11 24:19:41"}
```

```
{"card_id": 340028465709212,"member_id": 9250698176266,"amount": 900,"pos_id": 4444,"post_code":10451,"transc_dt":"2021-03-11 24:19:41"}
```

Spark structured streaming program as a Consumer that will consume the data from the kafka topics(Script Attached)

Read_Data_From_Kafka

Processing in Spark Streaming –

- Validating RULE 1 -> “credit_score > 200”
- Validating RULE 2 -> “transaction amount <= UCL”
- Validating RULE 3 -> “zipcode distance within threshold”

Below Scripts Validates above 3 rules and post the transaction along with its status to Hive HBase card_transactions table

Validation_Rules

We will ReadfromKafka class from below Jar which internally does the validation

Demo_Final.jar

--Airflow Script to run above Jar, class readFromKafka

cc_streaming_job.py

Appendix

Airflow

Below Connection and Variables have to be created under Admin Tab in Airflow for the scripts to function properly.

- **# Airflow Cloudera Connection**

  cloudera SSH quickstart.cloudera cloudera cloudera

- **Airflow Variables**

  memberscore_shell_command ./sqoop_import_member_score.sh database-2.cl4c0rtglkdz.ap-south-1.rds.amazonaws.com BankingPrj admin member_score

  memberdetails_shell_command ./sqoop_import_member_details.sh database-2.cl4c0rtglkdz.ap-south-1.rds.amazonaws.com BankingPrj admin member_details

  card_txns_export_shell_command ./sqoop_export_card_txns.sh quickstart.cloudera:3306 bigdataproject root stg_card_transactions

  card_txns_import_shell_command ./sqoop_import_card_txns.sh quickstart.cloudera:3306 bigdataproject root card_transactions

Distance Finder Jar

- Distance Finder Jar for Validating RULE 3 -> “zipcode distance within threshold” and list of valid ZIP Codes to be used.

distanceFinderJar.jar

zipCodePosId.csv

Kafka-Spark Dependency Jars

kafka-spark(dependancies).zip

Spark-HBase Dependency Jars

HBase_Jars-master.zip

Spark-Hive Dependency Jars

spark-hive_2.11-2.4.3.zip

Conclusion-

In this Project we developed an optimized system to find whether a real time credit card transaction is Fraud/Genuine based on Various Business rules and scheduled all Jobs using Airflow.

Further tested Genuine/Fraud transactions and got successfully updated in the respective tables.